Statistics: New Foundations, Toolbox, and Machine Learning Recipes

By Vincent Granville, Ph.D.

www.DataScienceCentral.com

July 2019.

This book is intended for busy professionals working with data of any kind: engineers, BI analysts, statisticians, operations research, AI and machine learning professionals, economists, data scientists, biologists, and quants, ranging from beginners to executives. In about 300 pages, it covers many new topics, offering a fresh perspective on the subject, including rules of thumb and recipes that are easy to automate or integrate in black-box systems, as well as new model-free, data-driven foundations to statistical science and predictive analytics. The approach focuses on robust techniques; it is bottom-up (from applications to theory), in contrast to the traditional top-down approach. The material is accessible to practitioners with a one-year college-level exposure to statistics and probability. The compact and tutorial style, featuring many applications with numerous illustrations, is aimed at practitioners, researchers, and executives in various quantitative fields.

New ideas, advanced topics and state-of-the-art research are discussed in simple English, without using jargon or arcane theory. It unifies topics that are usually part of different fields (machine learning, statistics, computer science), broadening the knowledge and interest of the reader in ways that are not found in any other book. This short book contains a large amount of condensed material that would typically be covered in 1,000 pages in traditional publications, including data sets, source code, and Excel spreadsheets. Thanks to cross-references and redundancy, the chapters can be read independently, in random order.

This book is based on several core articles and many tutorials that I have written over the last few years. Chapters are organized and grouped by themes: natural language processing (NLP), resampling, time series, central limit theorem, statistical tests, boosted models (ensemble methods), tricks and special topics, appendices, and so on. It is available for Data Science Central members exclusively. The text in blue consists of clickable links to provide the reader with additional references. Source code and Excel spreadsheets summarizing computations, are also accessible as hyperlinks for easy copy-and-paste or replication purposes. The most recent version of this book is available from this link, accessible to DSC members only.

About the author

Vincent Granville is a start-up entrepreneur, patent owner, author, investor, pioneering data scientist with 30 years of corporate experience in companies small and large (eBay, Microsoft, NBC, Wells Fargo, Visa, CNET) and a former VC-funded executive, with a strong academic and research background including Cambridge University.

Content

Part 2 - Applied Probability and Statistical Science

We discuss traditional statistical tests to detect departure from randomness (the null hypothesis) with applications to sequences (the observations) that behave like stochastic processes. The central limit theorem (CLT) is revisited and generalized with applications to time series (both univariate and multivariate) and Brownian motions. We discuss how weighted sums of random variables and stable distributions are related to the CLT, and then explore mixture models -- a better framework to represent a rich class of phenomena. Applications are numerous, including optimum binning for instance. The last chapter summarizes many of the statistical tests used earlier.

We set the foundations for a new type of statistical methodology fit for modern machine learning problems, based on generalized resampling. Applications are numerous, ranging from optimizing cross-validation to computing confidence intervals, without using classic statistical theory, *p*-values, or probability distributions. Yet we introduce a few new fundamental theorems, including one regarding the asymptotic properties of generic, model-free confidence intervals.

Part 4 - Case Studies, Business Applications

These chapters deal with real life business applications. Chapter 18 is peculiar in the sense that it features a very original business application (in gaming) described in details with all its components, based on the material from the previous chapters. Then we move to more traditional machine learning use cases. Emphasis is on providing sound business advice to data science managers and executives, by showing how data science can be successfully leveraged to solve problems. The presentation style is compact, focusing on strategy rather than technicalities.

Part 5 - Additional Topics

Here we cover a large number of topics, including sample size problems, automated exploratory data analysis, extreme events, outliers, detecting the number of clusters, *p*-values, random walks, scale-invariant methods, feature selection, growth models, visualizations, density estimation, Markov chains, A/B testing, polynomial regression, strong correlation and causation, stochastic geometry, *K* nearest neighbors, and even the exact value of an intriguing integral computed using statistical science, just to name a few.

1. Multi-use, Robust Pseudo-regression

We discuss a simple technique, first developed around 2002 when I was working for Visa, to blend different models to produce better predictions. It was developed independently from Stanford University's *boosted trees* and similar techniques. It blends pseudo linear regression with a large number of simple decision trees, without explicitly building decision trees. Among its advantages is ease of implementation, robustness (no risk of over-fitting), interpretability, and low number of hyper-parameters, making it suitable for black box machine learning applications. It is known as *hidden decision trees* (HDT).

This chapter focuses on the pseudo linear regression, one of the two components of HDT's. It is a constrained regression, similar to ridge or Lasso regression, or to methods based on penalized likelihood. We also show how it can be used to cluster either the variables (features) or the observations.

.

1. Introduction

Without loss of generality, we focus on linear regression with centered variables (with zero mean), and no intercept. Generalization to logistic or non-centered variables is straightforward. Let

$$\mathrm{Y} = a_1X_1 + ... + a_nX_n + e$$

Here e is the noise or error term. A solution, if you want the regression coefficients a_k and the correlations Cor(Y, X_k) with the response Y to have the same sign, is:

- $a_k = M\ b_k$, with b_k = Cov(Y, X_k) / Var(X_k), k = 1, ..., n and
- M (a real number) is chosen to minimize Var(e).

Let $W = b_1X_1 + ... + b_nX_n$. You must find M that minimizes Var(Y - MW). The solution is M = Cov(Y, W) / Var(W). If the independent variables (features) are non-correlated, then this regression and the classical regression produce the same results, and M = 1. You can add an intercept parameter c to the model; the final estimate becomes $W^* = c + W$, where c = Mean(Y - W).

Terminology: $S = a_1X_1 + ... + a_nX_n$ is the estimated or predicted response; the X_k's are the independent variables or features.

2. Example: simulated data with correlated features

I tested this methodology on a data set with 10,000 observations and 4 features. The data, source code to generate the data, graphs and results, are found in this spreadsheet. By construction, the correlation between the first two features is 0.99.

Whether you use an exact linear regression, or the approximated method described here, the goodness of fit (measured using R-squared) is similar. However the approximated method is far more robust. Robustness was tested by adding extra noise in the data. The exact regression coefficients are very sensitive to noise, and their values are very volatile. To the contrary, the regression coefficients obtained with this method are stable despite the high internal correlations.

3. Clustering the variables: solution based on two *M*'s

We can improve the estimates by considering a model with two *M*'s, namely *M* and *M*', where *M* applies to a subset of variables, and *M*' to the remaining variables. Now the estimated response is

$$S = \left(M \cdot \sum_{i \in I} b_i X_i\right) + \left(M' \cdot \sum_{j \in J} b_j X_j\right)$$

where *I* and *J* constitute a partition of {1, ... , *n*}. In short we are clustering the variables into two clusters. Again, the goal is to minimize Var(*Z*) = Var(*Y* - *S*), this time with respect to *M*, *M*', *I* and *J*. There are 2^n possible partitions (*I*, *J*), so we can loop over all these partitions, and for each partition, find the *M*, *M*' that minimizes Var(*Y* - *S*). Then identify the partition with absolute minimum for Var(*Y* - *S*).

The optimum partition will put highly correlated variables into a same cluster. In my example, since the first two features are highly correlated by construction, one would hope that the optimum partition will be {X_1, X_2} forming one cluster of variables, that is *I* = {1, 2}, and {X_3, X_4} forming the second cluster, that is *J* = {3, 4}. So I manually picked up this particular partition ({X_1, X_2}, {X_3, X_4}) as good enough for our test. I then tried a few values of *M*, *M*' for this particular partition, and settled with *M* = 0.1 and *M*'= 1.0. Clearly, there is no overfitting here. The parameters *M* and *M*' are located in cells P19 and R19 respectively, in the “data & results” tab in the spreadsheet.

Computations

Of course if you have many variables (*n* is large) then you might need more than two M and M'. Do not use more than 4 *M*'s for robustness. Note that 4 *M*'s require visiting 4^n partitions to identify the optimum one. In practice, I recommend to visit only 1,000 partitions out of 4^n, and choose the best one among these 1,000. To make the algorithm run much faster, you can do your computations using just 1% of the data set (but no less than 100 observations). Now you have a robust algorithm with a computational complexity that does not depend on the number of observations (if your computations are based on a sample of 100 observations), nor on the number of variables. Pretty amazing!

Note that in the case where we use two *M*s, namely *M* and *M*', given a partition (*I*, *J*), it is straightforward to compute the optimum *M*, *M*' depending on (*I*, *J*). Let's use the following notation:

$$S_I = \sum_{i \in I} b_i X_i, \; S_J = \sum_{j \in J} b_j X_j, \; S = M S_I + M' S_J.$$

Then the optimum is obtained by differentiating Var(Y - S) = Var(Y - MS_I - $M'S_J$) with respect to M and M'. This leads to a straightforward system of 2 linear equations with 2 unknowns M and M'. You need to solve that system to find M and M'. If you work with three M's, you would have to solve a similar system, but this time with 3 unknowns M, M', and M''.

4. Clustering the observations

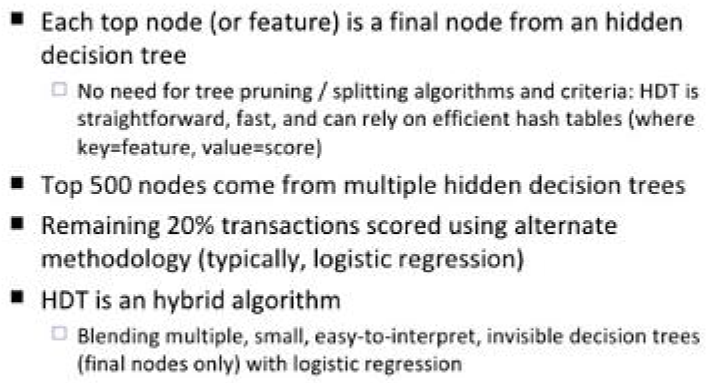
Just like we can cluster variables, we can apply the same methodology to cluster observations into two (or more) groups, using a different M for each group. Or you can cluster both variables and observations simultaneously.

However, in practice, if observations are too disparate for regression to make sense, I suggest using other techniques to cluster the observations. Adding one or two carefully crafted new variables can help solve the problem. Another approach is to apply hidden decision tree technology (see chapter 2) to bin the observations in hundreds or thousands of data buckets (each with at least 100 observations if possible), and apply a specific regression (that is, specifics M, M') to each bucket. This works well with big data.

2. A Simple Ensemble Method

The technique presented here, known as hidden decision trees, blends non-standard, robust versions of decision trees and regression. It has been successfully used in black-box ML implementations. Here we describe a case study to optimize website content. It is NLP-intensive. Source code in Perl, R, Julia and Python is provided.

We discuss a general machine learning technique to make predictions or score transactional data, applicable to very big, streaming data. This hybrid technique combines different algorithms to boost accuracy, outperforming each algorithm taken separately, yet it is simple enough to be reliably automated. It is illustrated in the context of predicting the performance of articles published in media outlets or blogs, and has been used by the author to build an AI (artificial intelligence) system to detect articles worth curating, as well as to automatically schedule tweets and other postings in social networks, with a goal of eventually fully automating digital publishing. This application is broad enough that the methodology can be applied to most NLP (natural language processing) contexts with large amounts of unstructured data.



- Each top node (or feature) is a final node from an hidden decision tree
  - No need for tree pruning / splitting algorithms and criteria: HDT is straightforward, fast, and can rely on efficient hash tables (where key=feature, value=score)
- Top 500 nodes come from multiple hidden decision trees
- Remaining 20% transactions scored using alternate methodology (typically, logistic regression)
- HDT is an hybrid algorithm
  - Blending multiple, small, easy-to-interpret, invisible decision trees (final nodes only) with logistic regression

Figure 1: *HDT 1.0. Here we will describe HDT 2.0.*

The algorithmic framework described here applies to any data set, text or not, with quantitative, non-quantitative (gender, race) or a mix of variables. It consists of several components; we discuss in details those that are new and original. No deep technical expertise and no mathematical knowledge is required to understand the concepts and methodology described here. The methodology, though state-of-the-art, is simple enough that it can even be implemented in Excel, for small data sets (one million observations.).

1. The Problem

Rather than first presenting a general, abstract framework and then showing how it applies to a specific problem (case study), we proceed the other way around, as we believe that it will help the reader understand better our methodology. We then generalize to any kind of data set.

In its simplest form, our particular problem consists of analyzing historical data about articles and blog posts, to identify features (also called metrics or variables) that are good predictors of blog popularity when combined together, to build a system that can predict the popularity of an article before it gets published. The goal is to select the right mix of relevant articles to publish, to increase web traffic, and thus advertising dollars, for a niche digital publisher.

As in any similar problem, the historical data is called *training set*, and it is split into *test data* and *control data* for *cross-validation* purposes to avoid *over-fitting.* The *features* are selected to maximize some measure of *predictive power,* as described in chapter 4. All of this is (so far) standard practice; the reader not familiar with this can Google the keywords introduced in this paragraph. In our particular case, we use our domain expertise to identify great features. These features are pretty generic and apply to numerous NLP contexts, so you can re-use them for your own data sets.

Feature Selection and Best Practices

One caveat is that some metrics are very sensitive to manipulation. In our case, the *response* (that is, what we are trying to predict, also called *dependent variable* by statisticians) is the traffic volume. It can be measured in page views, unique page views, or number of users who read the article. Page views can easily be manipulated and the number is inflated by web robots, especially for articles that have little traffic. So instead, we chose "unique page views", a more robust metric available through Google Analytics. Also, older articles have accumulated more page views over time, so we need to correct for this effect. Correcting for time is explained in chapter 19. Here we used a very simple approach instead: focusing on articles from the most recent, main channel instead (the time window is about two years), and taking **the logarithm of unique page views** (denoted as **pv** in the source code in the last section).

Taking the logarithm not only smooths out the effect of time and web robots, but also it makes perfect sense as the page view distribution is highly skewed -- well modeled using a Zipf distribution -- with a few incredibly popular (viral) articles and a large number of articles with average traffic: it is a bit like the income distribution.
As for selecting the features, we have two kinds of metrics that we can choose as predictors:

Metrics based on the article title, easy to compute:

- Keywords found in the title
- Article category (blog, event, forum question)
- Channel

- Creation date
- Title contains numbers?
- Title is a question?
- Title contains special characters?
- Length of title

Metrics based on the article body, more difficult to compute:

- Size of article
- Does it contain pictures?
- Keywords found in body
- Author (and author popularity)
- First few words

Despite focusing only on a subset of features associated with the article title, we were able to get very interesting, actionable insights; we only used title keywords, and whether the posting is a blog, or not. The methodology used here takes into account all potential *key-value* combinations, where a *key* is a subset of features, and *value*, the respective values: for instance key = (keyword$_1$, keyword$_2$, article category) and value = ("Python", "tutorial", "Blog"). So it is important to appropriately bin the variables (see section 4 in chapter 16) when turning them into features, to prevent the number of key-value pairs from exploding. Another mechanism described later in this chapter is also used to keep the key-value database, stored as an hash table or associate array, manageable. Finally, it can easily be implemented in a distributed environment (Hadoop.)

Due to the analogy with decision trees, a key-value is also called a *node*, and plays the same role as a node in a decision tree.

2. Methodology and Solution

As we have seen in the previous section, the problem consists of predicting **pv**, the logarithm of unique page views for an article (over some time period), as a function of keywords found in the title, and whether the article in question is a blog or not.
In order to do so, we created lists of all one-token and two-token keywords found in all the titles, as well as blog status, after cleaning the titles and eliminating some stop word such as "that", "and" or "the", that don't have impacts on the predictions. We were also careful about not eliminating all keywords made up of one or two letters: the one-letter keyword "R", corresponding to the programming language R, has a high predictive power.

For each element in our lists, we recorded the frequency and traffic popularity. More precisely, for each key-value pair, we recorder the number of articles (titles, actually) that are associated with it, as well as the average, minimum and maximum **pv** across these articles.

Example

For instance, the element or key-value (keyword$_1$ = "R", keyword$_2$ = "Python", article = "Blog") is associated with 6 articles, and has the following statistics: average **pv** = 8.52, minimum **pv** = 7.41, and maximum **pv** = 10.45.

Since the average **pv** across all articles is equal to 6.83, this specific key-value pair (also called *node*) generates exp(8.52 - 6.83) = 5.42 times more traffic than an average article. It is thus a great node. Even the worst article, among the 6 articles belonging to this node, with a **pv** of 7.41, outperforms the average article across all nodes. So not only this is a great node, but also a stable one. Some nodes have a far larger volatility, for instance when one of the keywords has different meanings, such as the word "training", in "training deep learning" (training set) versus "deep learning training" (courses.)

Hidden decision trees

Note that here, the nodes are overlapping, allowing considerable flexibility. In particular, nodes with two keywords are sub-nodes of nodes with one keyword. A previous version of this technique, described here, did not consider overlapping nodes. Also, with highly granular features, the number of nodes explodes exponentially. A solution to this problem consists of

- Shuffling the observations
- Working with nodes built on no more than 4 or 5 features
- Proper binning
- Visiting the observations sequentially (after the shuffle) and every one million observations, deleting nodes that contain only one observation

The general idea behind this technique is to group articles into buckets that are large enough to provide predictions that are sound, without explicitly building decision trees. Not only the nodes are simple and easy to interpret, but unstable nodes are easy to detect and discard. There is no splitting/pruning involved as with classical decision trees, making this methodology simple and robust, and thus fit for artificial intelligence (automated processing.)

General framework

Whether you are dealing with predicting the popularity of an article, or the risk for a client to default on a loan, the basic methodology is identical. It involves training sets, cross-validation, feature selection, binning, and populating hash tables of key-value pairs (referred to here as the nodes).

When you process a new observation, you check which node(s) it belongs to. If the best node it belongs to is stable and not too small, you use it to predict the future performance or value of your observation, or to score the transaction if you are dealing with transactional data such as credit card transactions. In our example, if the performance metric (the average **pv** in the node in question) is significantly above the global average, and other constraints are met (the node is not too small, and the minimum **pv** in the node in question not too low to guarantee stability), then we classify the observation as good, just like the node it belongs to. In our case, the observation is a potential article.

Also, you need to update your training set and the node table (including automatically discovered new nodes) every six months or so.

Parameters must be calibrated to guarantee that

- Error rate (classifying a good observation as bad or the other way around) is small enough; it is measured using a *confusion matrix*
- The system is robust: we have a reasonable number of stable nodes that are big enough; it is great if less than 3,000 stable, not too small nodes cover 80% of the observations (by stable, we mean nodes with low variance) with an average of at least 10 observations per node
- The binning and feature selection mechanism offer real predictive power: the average response (our **pv**) measured in a node classified as good, is much above the general average, and the other way around for nodes classified as bad; in addition, the response shows little volatility within each node (in our case, **pv** is relatively stable across all observations within a same usable node)
- We have enough usable nodes (that is, after excluding the small ones) to cover at least 50% of all observations, and if possible up to 95% of all observations (100% would be ideal but never exists in practice)

We discuss the parameters of our technique, and how to fine-tune them, in the next section. Fine-tuning can be automated or made more robust by testing (say) 2,000 sets of parameters and identify regions of stability that meet our criteria (in terms of error rate and so on) in the parameter space.

A big question is what to do with observations not belonging to any usable node: they cannot be classified. In our example it does not matter if 30% of the observations cannot be classified, but in many applications, it does matter. One way to address this issue is to use super-nodes: in our case, a node for all posts that are blogs, and another one for all posts that are not blogs (these two nodes cover 100% of observations, both past and future.) The problem is that usually, these super-nodes don't have much predictive power. A better solution consists of using two algorithms: the one described here based on usable nodes (let's call it algorithm A) and another one called algorithm B that classifies all observations. Observations that can't be classified or scored with algorithm A are classified/scored with algorithm B. You can read the details about how to blend the results of two algorithms, in one of my patents. In practice, we have used the technique described in chapter 1 for algorithm B, a technique easy to implement, easy to understand, leading to simple interpretations, and robust. These features are important for systems that are designed to run automatically.

The resulting hybrid algorithm is called *Hidden Decision Trees* - hidden because you don't even realize that you have created a bunch of mini decision trees: it was all implicit. The version described here is version 2, with new features to prevent the node table from exploding, and allowing nodes to overlap, making it more suitable for data sets with a larger number of variables.

3. Case Study: Results

Our application about predicting page views for an article has been explained in detail in the previous sections. So here we focus on the results obtained.

Output from the algorithm

If you run the script listed in the next section, besides producing the table of key-value pairs (the nodes) as a text file for further automated processing, it displays summary statistics that look like the following:

```
Average pv: 6.81
Number of articles marked as good: 865 (real number is 1079)
Number of articles marked as bad: 1752 (real number is 1538)
Avg pv: articles marked as good: 8.23
Avg pv: articles marked as bad: 6.13
Number of false positive: 50 (bad marked as good)
Number of false negative: 264 (good marked as bad)
Number of articles: 2617
Error Rate: 0.12
Number of feature values: 16712 (marked as good: 3409)
Aggregation factor: 1.62
```

The number of “feature values” is the total number of key-value pairs found, including the small unstable ones, regardless as to whether they are classified as good or bad. Any article with a **pv** above the arbitrary value `pv_threshold` = 7.1 (see source code) is considered as good. This corresponds to articles having about 1.3 times more traffic than average, since we use a log scale and the average **pv** is 6.81. The traffic for articles classified as good by the algorithm (**pv** = 8.23) is about 4.2 times above the traffic that an average article receives.

Two important metrics are:

- Aggregation factor: it is an indicator of the average size of a node. The minimum is 1, corresponding to nodes that only have one observation. A value above 5 is highly desirable, but here, because we are dealing with a small data set and with niche articles, even a small value is OK.
- The error rate is the number of articles wrongly classified. Here we care much more about bad articles classified as good.

Also note that we correctly identify the vast majority of good articles, but this is because we work with small nodes. Finally an article is marked as good if it triggers at least one node marked as good (that is, satisfying the criterion defined in the next sub-section.)

Parameters

Besides `pv_threshold`, the algorithm uses 12 parameters to identify a usable, stable node classified as good. These parameters are illustrated in the following piece of code (see source code):

```
if ( (($n > 3)&&($n < 8)&&($min > 6.9)&&($avg > 7.6)) ||
     (($n >= 8)&&($n < 16)&&($min > 6.7)&&($avg > 7.4)) ||
     (($n >= 16)&&($n < 200)&&($min > 6.1)&&($avg > 7.2)) ) {
```

Here, *n* represents the number of observations in a node, while, `avg` and `min` are the average and minimum **pv** for the node in question. We tested many combinations of values for these parameters. Increasing the required size (denoted as *n*) of a usable node will do the following:

- Decrease the number of good articles correctly identified as good
- Increase the error rate
- Increase the stability of the system
- Decrease the predictive power
- Increase the aggregation factor (see previous sub-section)

Improving the methodology

Here we share some caveats and possible improvements to our technique.

You need to use a table of one-token keywords that look like two tokens, for increased efficiency, and consider these keywords as being one-token. For instance “San Francisco” is a one-token keyword, despite its appearance. Such tables are easy to build as you always see the two parts together.

Also, we looked at nodes containing (keyword$_1$, keyword$_2$) where the two keywords are adjacent. If you allow the two keywords not to be adjacent, the number of key-value pairs (the nodes) increases significantly, but you don't get much additional predictive power in return: there is even a risk of over-fitting.

Another improvement consists of having/favoring nodes containing observations spread over a long time period, to avoid any kind of concentration (which could otherwise result in over-fitting.)

Finally, in our case, we cannot exclusively focus on articles with great potential. It is important to have many, less popular articles as well: they constitute the long tail. Without these articles, we face problems such as excessive content concentration, which have negative impacts in the long term. The obvious negative impact is that we might miss nascent topics, and thus getting stuck into a non-adaptive mix of articles at some point, thus slowing growth.

Interesting findings

Titles with the following features work well:

- Contains a number (10, 15 and so on) as we have many popular articles such as "10 great deep learning articles"
- Contains the current year
- Is a question (how to)
- Not a blog, but a book category
- A blog

Titles containing the following keywords work well:

- everyone (as in "10 regression techniques everyone should know")
- libraries
- infographic
- explained
- algorithms
- languages
- amazing
- must read
- r python
- job interview questions
- should know (as in "10 regression techniques everyone should know")
- nosql databases
- versus
- decision trees
- logistic regression
- correlations
- tutorials
- code
- free

4. Source Code

The source code is easy to read and has deliberately made longer than needed to provide enough details, avoid complicated iterations, and facilitate maintenance and translation into Python or R. The output file `hdt-out2.txt` stores the key-value pairs (or nodes) that are usable, corresponding to popular articles. Here is the input data set: `HDT-data3.txt`.

The code has been written in Perl, R and Python. Perl and Python run faster than R. Click on the relevant link below to access the source code, available as a text file. The code was originally written in Perl, and translated to Python and R by Naveenkumar Ramaraju.

- Python version
- Perl version
- R version
- Improved R version

For those learning Python or R, this is a great opportunity. HDT (a light version) has been implemented in Excel too, see chapter 3.

Note regarding the R implementation

Required library: `hash` (R doesn't have inbuilt hash or dictionary without imports.)

- Standard version is the literal translation of the Perl code with same variable names to the maximum extent possible.
- Improved version uses functions, more data frames and more R-like approach to reduce code running time (~30 % faster) and less lines of code. Variable names would vary from Perl. Output file would have comma(,) as delimiter between IDs.

Instructions to run: Place the R file and `HDT-data3.txt` (input file) in root folder of R environment. Execute the ".R" file in R studio or using command line script:

```
> Rscript HDT_improved.R
```

R is known to be slow in text parsing. We can optimize further if all inputs are within double quotes or no quotes at all by using data frames.

Julia version

This was added later by Andre Bieler. A few remarks about:

- This code is absolutely not tuned for performance since everything is done in global scope. (In Julia it would be good practice to put everything in small functions)
- Generally for run times of only a few 0.1 s Python will be faster due to the compilation times of Julia.

Julia really starts paying off for longer execution times. Contact the author to get the Julia code.

3. Excel Implementation

The technique described in the previous chapter is adapted here to Excel. While it obviously shows the limitations of Excel, more surprisingly, it shows how far you can go, and how much you can do with Excel, on the same case study and data set. Essentially, it leads to the same business insights.

Here we focus on an Excel version that does not even require any Excel macros, coding, plug-ins, or anything other than the most basic version of Excel. It is actually easily implemented in standard, basic SQL too, and we invite readers to work on an SQL version.

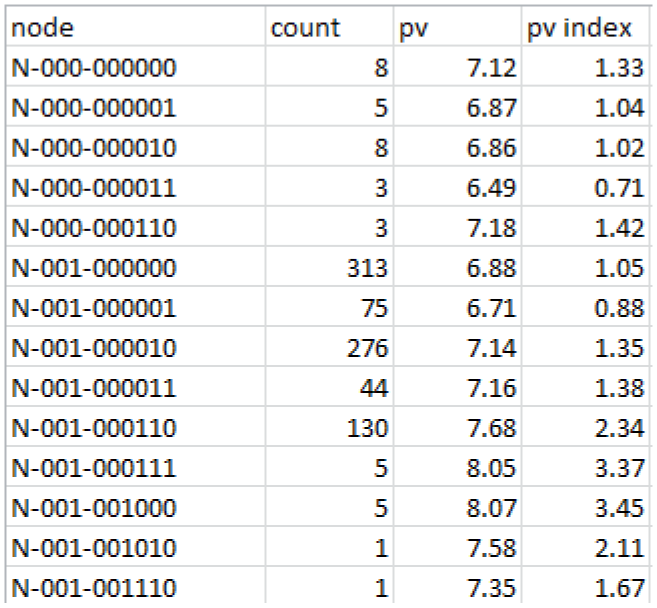
1. Excel template for general machine learning

In short, we offer here an Excel template for machine learning and statistical computing, and it is quite powerful for an Excel spreadsheet. The techniques have been used by the author in automated data science (AI to automate content production, selection and scheduling articles for digital publishers) but also in the following contexts:

- Spam detection
- click, website, and keyword scoring (assigning a commercial value to a keyword, group of keywords, or content category)
- Credit card fraud detection
- Botnet detection and predicting blog popularity.

The technique blends multiple algorithms that at first glance look traditional and math-heavy, such as decision trees, regression (logistic or linear) and confidence intervals. But they are radically different, can fit in a small spreadsheet (though the Python version is more powerful, flexible, and efficient), and do not involve math beyond high-school level. In particular, no matrix algebra is required to understand the methodology.

The methodology presented here is the result of 20 years' worth of applied research on various large industrial data sets.



node	count	pv	pv index
N-000-000000	8	7.12	1.33
N-000-000001	5	6.87	1.04
N-000-000010	8	6.86	1.02
N-000-000011	3	6.49	0.71
N-000-000110	3	7.18	1.42
N-001-000000	313	6.88	1.05
N-001-000001	75	6.71	0.88
N-001-000010	276	7.14	1.35
N-001-000011	44	7.16	1.38
N-001-000110	130	7.68	2.34
N-001-000111	5	8.05	3.37
N-001-001000	5	8.07	3.45
N-001-001010	1	7.58	2.11
N-001-001110	1	7.35	1.67

Node table (extract, from spreadsheet)

Who should use the spreadsheet?

First, the spreadsheet (as well as the Python, R, Perl or Julia version) are free to use and modify, even for commercial purposes, or to make a product out of it and sell it. It is part of my concept of open patent, in which I share all my intellectual property publicly and for free.

The spreadsheet is designed as a tutorial, though it processes the same data set as the one used for the Python version. It is aimed at people that are not professional coders, people who manage data scientists, BI experts, MBA professionals, and people from other fields, with an interest in understanding the mechanics of some state-of-the-art machine learning techniques, without having to spend months or years learning mathematics, programming, and computer science. A few hours is needed to understand the details. This spreadsheet can be the first step to help you transition to a new, more analytical career path, or to better understand the data scientists that you manage or interact with, or to spark a career in data science. Or even to teach machine learning concepts to high school students.

The spreadsheet also features a traditional technique (linear regression) for comparison purposes.

2. Description of the techniques used

Here we explain the differences between the standard and the Excel versions, and we provide an overview, at a high level, of the techniques being used, as well as why they are better in pretty much all applications, especially with unstructured and large data sets. Detailed descriptions are available in the articles referenced in this section.

Spreadsheet versus Python version

The Python version (also available in R, Perl and Julia) of the core technique is described in the previous chapter. Python / Perl offer the following advantages over Excel:

- It easily handles version 2.0 of HDT (see chapter 2) including overlapping nodes
- It easily handles big datasets, even in a distributed environment if needed
- It easily handles a large number of nodes
- Of course, it is incredibly faster for large data sets

The Excel version has the advantage of being interactive, and you can share it with people who are not data scientists.

But Excel (at least the template provided here) is mostly limited to nodes that form a partition of the feature space, that is, it is limited to non-overlapping nodes: see HDT version 1.0. So even if we have two nodes, one for the keyword *data*, and one for the keyword *data science*, in version 1,0, they are not overlapping: text buckets contain either *data* and not *data science*, or *data science*. In version 2.0, we no longer have this restriction. Note that nodes can be a combination of any number of keyword values or any other variables (called *features* in machine learning), and these variables can be quantitative or not.

For those familiar with computer science, nodes, both in the Excel or the Python version, are represented here as key-value pairs, typically stored as hash tables in Perl or Python, and as concatenated strings in Excel. For statisticians, nodes are just nodes of decision trees, though no tree structure is used (nor built) in my methodology -- and this is why it is sometimes referred to as *hidden decision trees* (HDT). But you don't need to understand this to use the methodology or understand how the spreadsheet works.

What is it about?

The methodology features an hybrid algorithm with essentially two components:

- Data aggregation into bins, based on sound feature selection, binning continuous and discrete features, and metric design, not unlike decision trees. However, no tree is actually built, and the nodes may belong to several overlapping small decision trees, each one corresponding to a case or cluster easy to interpret. This is particularly true in HDT 2.0. I will call this the pseudo decision tree algorithm.
- The regression algorithm described in chapter 1, requiring much fewer parameters than classical regression models, and more meaningful parameters, to avoid over-fitting and to be able to cope with cross-correlated features, while at the same time offering a simple interpretation. In the application discussed in the spreadsheet, one could argue that the regression used here is closer to logistic

than linear regression as data is transformed using a logit mapping, and we are predict, for an article, the odds of being popular.

Data points belonging to a small node (say *n* < 10 observations) have the estimated / predicted response computed using the regression (algorithm #2 above), the remaining points get scored using the pseudo-decision tree algorithm (algorithm #1 above.)

A lot of intelligence and creativity is put into creating great predictors (the features) and then perform sound feature selection. However, the features used in the spreadsheet and in the previous chapter (dealing with the same data set) apply to all NLP (natural language processing) systems in numerous contexts.

In addition, while not incorporated in the spreadsheet, confidence intervals can be computed for each node with at least *n* observations (say *n* = 10) using percentiles for the response, computed for all data points (in this case, representing articles) in the node in question, see example at the bottom of section 3. This percentile function is even available in Excel. Then, data points in a node with too large a confidence interval are scored using the pseudo regression (first chapter) rather than the pseudo decision trees. By scoring, I mean having the response estimated or predicted. By response, I mean the variable that we are trying to predict: in this case the page views number attached to an article (indeed, its logarithm, to smooth out big spikes due to external factors, or the fact that older articles have by definition more page views -- see chapter 19 for details.)

So no statistical theory is used anywhere in the methodology, not even to compute confidence intervals.

Why a brand new set of machine learning tools?

The HDT methodology offers the following advantages:

- The loss of accuracy, compared with standard procedures, is so small in the **control data set**, that it is negligible and much smaller than the inherent noise present in the data. This has been illustrated before on a different data set (see chapter 1), and it is confirmed again here (see next section.).
- The accuracy is much higher in the **test data set**, in a cross-validation framework where HDT is performed on a control data set, and performance measured on a different data set called test data set. So the methodology simply works better in the real world. This is easy to understand: HDT was designed as a robust method, to avoid over-fitting and issues caused by outliers, as well as to withstand model failures, messy data, and violations of assumptions.

In addition HDT also offers the following benefits:

- Easy interpretation of the results

- Simplicity, scalability, easy to implement in a distributed environment, and tested on unstructured big data
- No need to know statistical or mathematical theory to understand its inner workings
- Great to use as a machine learning tutorial for people who do not code or not interesting in learning more about machine learning and coming from a different field (software engineering, management consulting, bioinformatics, econometrics, journalism, and so on.)
- Could be used in STEM programs in high schools, to give kids the chance to work on real machine learning problems using modern techniques.
- Few parameters to deal with, this is essentially a non-parametric, data-driven (as opposed to model-driven) technique.
- Since most companies use standard tools and software, using HDT can give you a competitive advantage (if you are allowed to choose your own method), and the learning curve is minimum.

Another way to highlight the benefits is to compare with Naive Bayes. Naive Bayes assumes that the features are independent. It is the workhorse of spam detection, and we all know how bad it performs. For instance, a message containing the keyword “breast cancer” is flagged because it contains the keyword “breast”, and Naive Bayes erroneously assumes that “breast” and “cancer” are independent. Not true with HDT.

Classical decision trees, especially the large ones with millions of nodes from just one single decision tree and involving more than 5 or 6 features at each final node, suffer from similar issues: over-fitting, artificial feature selection resulting in difficulties interpreting the results, maintenance challenges, over-parameterization making it more difficult to fine-tune, and most importantly, lack of robustness.

3. The Spreadsheet

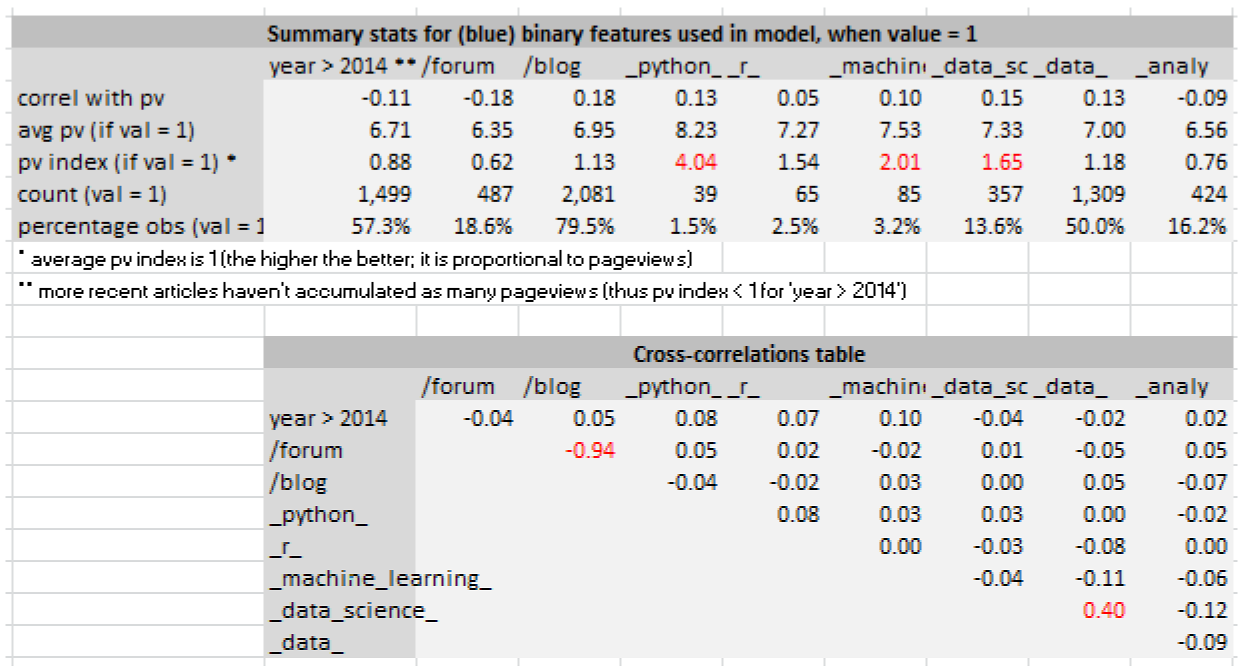
The data set and features used in this analysis are described in the previous chapter. The spreadsheet only uses a subset of the original features, as it is provided mostly as a template and for tutorial purposes. Yet even with this restricted set of features, it reveals interesting insights about some keywords (Python, R, data, data science) associated with popularity (Python being more popular than R), and some keywords that surprisingly, are not (keywords containing “analy”, such as analytic.) Besides keywords found in the title, other features are used such as time of publication, and have also been binarized to increase stability and avoid an explosion in the number of nodes. Note that HDT 2.0 can easily handle a large number of nodes, and even HDT 1.0 (used in the spreadsheet) easily handles non-binary features.

There are 2,616 observations (articles) and 74 nodes. By grouping all nodes with less than 10 observations into one node, we get down to 24 nodes. Interestingly, these small nodes perform much better than the average node. The correlations between the features and the response are very low, mostly because the keyword-like features trigger very few observations: very few articles contain the keyword R in the title (less than 3%.) As a result, the correlation between the response and predicted response is not high, around 0.33 regardless of the model. The solution is of course to add many more keywords to cover a much larger proportion of articles.

Notes

- In the Python version (see chapter 2), keyword detection / selection (to create features) is part of the process, and included in the source code. Here, the keywords used as features are assumed to be pre-selected.
- Page view index (see spreadsheet) is a much better performance indicator than R-squared or correlation with response, to measure the predictive power of a feature. This is clearly the case with the feature “Python”.
- The Excel version is slightly different from the Python version, from a methodological point of view, as described in section 2.
- The goodness-of-fit for pseudo and linear regressions are very close, despite the fact that the pseudo-regression is a very rough (but robust) approximation of the linear regression.
- Pseudo-regression has been used in its most elementary version, with only one *M*. When the cross-correlation structure is more complex, I recommend using it with two *M*'s as described in the first chapter.
- Some of the features are correlated, for instance “being a blog” with “being a forum question”, or “containing data but not data science” with "containing data science".
- When combining pseudo-regression with the pseudo-decision trees (applying pseudo-regression to small nodes) we get a result that is better than pseudo-regression, pseudo-decision trees, or linear regression taken separately.
- For much larger data sets that include all sorts (categories) of articles (not just about data science), I recommend creating and adding a feature called *category*. Such a feature can be build using an indexation algorithm (see chapter 6).
- The response is denoted as **pv**.

Click here to get the spreadsheet. Below are some screenshots from the spreadsheet. Here **pv** is the response (logarithm of page views.)

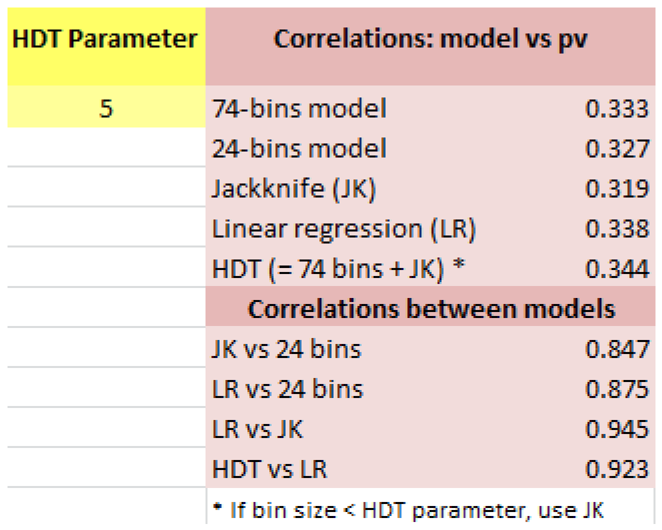


Summary stats for (blue) binary features used in model, when value = 1									
	year > 2014 **	/forum	/blog	_python_	_r_	_machin	_data_sc	_data_	_analy
correl with pv	-0.11	-0.18	0.18	0.13	0.05	0.10	0.15	0.13	-0.09
avg pv (if val = 1)	6.71	6.35	6.95	8.23	7.27	7.53	7.33	7.00	6.56
pv index (if val = 1) *	0.88	0.62	1.13	4.04	1.54	2.01	1.65	1.18	0.76
count (val = 1)	1,499	487	2,081	39	65	85	357	1,309	424
percentage obs (val = 1	57.3%	18.6%	79.5%	1.5%	2.5%	3.2%	13.6%	50.0%	16.2%

* average pv index is 1 (the higher the better; it is proportional to pageviews)

** more recent articles haven't accumulated as many pageviews (thus pv index < 1 for 'year > 2014')

Cross-correlations table								
	/forum	/blog	_python_	_r_	_machin	_data_sc	_data_	_analy
year > 2014	-0.04	0.05	0.08	0.07	0.10	-0.04	-0.02	0.02
/forum		-0.94	0.05	0.02	-0.02	0.01	-0.05	0.05
/blog			-0.04	-0.02	0.03	0.00	0.05	-0.07
python				0.08	0.03	0.03	0.00	-0.02
_r_					0.00	-0.03	-0.08	0.00
_machine_learning_						-0.04	-0.11	-0.06
_data_science_							0.40	-0.12
data								-0.09



HDT Parameter	Correlations: model vs pv	
5	74-bins model	0.333
	24-bins model	0.327
	Jackknife (JK)	0.319
	Linear regression (LR)	0.338
	HDT (= 74 bins + JK) *	0.344
	Correlations between models	
	JK vs 24 bins	0.847
	LR vs 24 bins	0.875
	LR vs JK	0.945
	HDT vs LR	0.923

* If bin size < HDT parameter, use JK

Confidence intervals for the response: example

Node N-100-000000 in the spreadsheet has an average **pv** of 5.85 (**pv** is the response), and consists of the following pv values: 5.10, 6.80, 5.56, 5.66, 6.19, 6.01, 5.56, 5.10, 6.80, 5.69. The 10th and 90th percentiles for **pv** are respectively 5.10 and 6.80, so [5.10, 6.80] is our confidence interval (CI) for this node. This computation of CI is similar to the methodology discussed here. This particular CI is well below the average **pv** -- even the upper bound 6.80 is below the average **pv** of 6.83. In fact this node corresponds to articles posted after 2014, not a blog or forum question (it could be a video or event announcement), and with a title containing none of the keywords from the keyword feature list. The business question is: Should we continue to accept and promote such poor performing content? The answer is yes, but not as much as we used to.

4. Fast Feature Selection

In all machine learning problems, deciding which metrics to use is one of the core problems. This chapter addresses this topic.

I propose a simple metric to measure **predictive power**. It is used for combinatorial feature selection, when a large number of feature combinations need to be ranked automatically and very fast, for instance in the context of transaction scoring, in order to optimize predictive models. It can easily be implemented in a Map Reduce framework. It was developed by the author in the context of credit card fraud detection, and click/keyword scoring.

Feature selection is used to detect the best subset of features, out of dozens or hundreds of features (also called variables or rules). By “best”, we mean with highest *predictive power*, a concept defined in the following subsection. In short, we want to remove duplicate features, correlations between features, and features lacking predictive power, or features (sometimes called rules) that are rarely triggered -- except if they are excellent predictors of rare but costly fraud for instance.

The problem is combinatorial in nature. You want a manageable, small set of features (say 20 features) selected from (say) a set of 500 features, to run algorithms such as *hidden decision trees* (see chapter 2) in a way that is statistically robust. But there are $2.7 * 10^{35}$ combinations of 20 features out of 500, and you need to compute all of them to find the feature set with maximum predictive power. This problem is computationally intractable, and you need to find an alternate solution. The good thing is that you don’t need to find the absolute maximum; you just need to find a subset of 20 features that is good enough.

One way to proceed is to compute the predictive power of each feature. Then, add one feature at a time to the subset (starting with 0 feature) until you reach either

- 20 features (your limit)
- Adding a new feature does not significantly improve the overall predictive power of the subset (in short, convergence has been attained)

At each iteration, choose the feature to be added, among the two remaining features with the highest predictive power: you will choose (among these two features) the one that increases the overall predictive power (of the subset under construction) most. Now you have reduced your computations from $2.7 * 10^{35}$ to $40 = 2 * 20$.

Technical note: Additional step to boost predictive power**.** Remove one feature at a time from the subset, and replace it with a feature randomly selected from the remaining features. If this new feature boosts the overall predictive power of the feature subset, keep it, and otherwise switch back to old subset. Repeat this step 10,000 times or until no more gain is achieved (whichever comes first).

Finally, you can add two or three features at a time, rather than one. Sometimes, combined features have better predictive power than isolated features. For instance if feature A = country, with values in {USA, UK} and feature B = hour of the day, with values in {“day - Pacific Time”, “night - Pacific Time”}, both features separately have little if any predictive power. But when you combine both of them, you have a much more powerful feature: UK/night is good, USA/night is bad, UK/day is bad, and USA/day is good, if your response (what you are predicting) is Internet traffic quality. Using these two features together also reduces the risk of false positives / false negatives.

Also, in order to avoid highly granular features, use lists. So instead of having feature A = country (with 200 potential country values), and feature B = IP address (with billions of potential values), use:

- Feature A = country group, with 3 list of countries (high risk, low risk, neutral). These groups can change over time.
- Feature B = type of IP address (with 6-7 types, one being for instance “IP address is in some whitelist”.

1. Predictive Power of a Feature, Cross-Validation

Here we illustrate the concept of predictive power on a subset of 2 features. Let’s say that we have two binary features A and B taking two possible values 0 or 1. Also, in the context of fraud detection, we assume that each observation in the training set is either **Good** (no fraud) or **Bad** (fraud). The fraud status (**G** or **B**) is called the *response* or *dependent variable* in statistics. The features A and B are also called *rules* or *independent variables*.

Cross validation

First, split your training set (the data where the response B or G is known) into two parts: control and test. Make sure that both parts are data-rich: if the test set is big (millions of observations) but contain only one or two clients (out of 200), it is data-poor and your statistical inference will be negatively impacted (low robustness) when dealing with data outside the training set. It is a good idea to use two different time periods for control and test. You are going to compute the predictive power (including rule selection) on the control data. When you have decided on a final, optimum subset of features, you will then compute the predictive power on the test data. If the drop in predictive power is significant in the test data (compared with control), something is wrong with your analysis: detect the problem, fix it, start over. You can use multiple control and test sets: this will give you an idea of how the predictive power varies from one control set to another one. Too much variance is an issue that should be addressed.

Predictive power

Using our above example with two binary features A, B taking on two values 0, 1, we can break the observations from the control data set into 8 categories

Category #	A	B	Response
1	0	0	G
2	0	1	G
3	1	0	G
4	1	1	G
5	0	0	B
6	0	1	B
7	1	0	B
8	1	1	B

Let denote as n_1, n_2 … n_8 the number of observations in each of these 8 categories, and let us introduce the following quantities:

$$P_{00} = n_5 / (n_1 + n_5),\ P_{01} = n_6 / (n_2 + n_6),\ P_{10} = n_7 / (n_3 + n_7),\ P_{11} = n_8 / (n_4 + n_8)$$
$$p = (n_5 + n_6 + n_7 + n_8) / (n_1 + n_2 + \ldots + n_8).$$

Let's assume that p, measuring the overall proportion of fraud, is less than 50% (that is, $p < 0.5$, otherwise we can swap between fraud and non-fraud). For any r between 0 and 1, define the W function (shaped like a W), based on a parameter $\boldsymbol{a}$ (0 < a < 1, typically $\mathbf{a} = 0.5 - p$) as follows:

- $W(r) = 1 - (r / p)$, if $0 < r < p$
- $W(r) = \boldsymbol{a}\,(r - p) / (0.5 - p)$, if $p < r < 0.5$
- $W(r) = \boldsymbol{a}\,(r - 1 + p) / (p - 0.5)$, if $0.5 < r < 1 - p$
- $W(r) = (r - 1 + p) / p$, if $1 - p < r < 1$

Typically, $r = P_{00}$, P_{01}, P_{10} or P_{11}. The W function has the following properties:

- It is minimum and equal to 0 when $r = p$ or $r = 1 - p$, that is, when r does not provide any information about fraud / non fraud,
- It is maximum and equal to 1when $r = 1$ or $r = 0$, that is, when we have perfect discrimination between fraud and non-fraud, in a given bin.
- It is symmetric: $W(r) = W(1 - r)$ for $0 < r < 1$. So if you swap Good and Bad (G and B), it still provides the same predictive power.

Now let's define the predictive power:

$$H = P_{00}\, W(P_{00}) + P_{01}\, W(P_{01}) + P_{10}\, W(P_{10}) + P_{11}\, W(P_{11})$$

The function H is the predictive power for the feature subset {A, B} having four bins 00, 01, 10, and 11, corresponding to (A = 0, B = 0), (A = 0, B = 1), (A = 1, B = 0) and (A = 1, B = 1). Although H appears to be remotely related to the *entropy*, our H was designed to satisfy desirable properties, and to be parameter-driven, thanks to ***a***. Unlike entropy, our H is not based on physical concepts or models; it is actually a synthetic (though useful) metric.

Note that the weights P_{00}… P_{11} in H guarantee that bins with low frequency (that is, low triggering rate) have low impact on H. Indeed, I recommend setting $W(r)$ to 0 for any bin that has less than 20 observations. For instance, the triggering rate for bin 00 is $(n_1 + n_5) / (n_1 + \ldots + n_8)$, its size is $n_1 + n_5$, and $r = P_{00} = n_5 / (n_1 + n_5)$ for this bin. If $n_1 + n_5 = 0$, set P_{00} to 0 and $W(P_{00})$ to 0. I actually recommend to do this not just if $n_1 + n_5 = 0$, but also whenever $n_1 + n_5 < 20$, especially if p is low (if p is very low, say $p < 0.01$, you need to over-sample bad transactions when building your training set, and weight the counts accordingly). Of course, the same rule applies to P_{01}, P_{10}, and P_{11}. Note that you should avoid feature subsets that have a large proportion of observations spread across a large number of almost empty bins, as well as feature subsets that produce a large number of empty bins: observations outside the training set are likely to belong to an empty or almost empty bin, and it leads to high-variance predictions. To avoid this drawback, stick to binary features, select up to 20 features, and use our (hybrid) hidden decision tree methodology for scoring transactions. Finally, P_{kl} is the naive estimator of the probability P(A = k, B = l) for k, l = 0,1.

The predictive power H has interesting properties:

- It is always between 0 and 1, equal to 0 if the feature subset has no predictive power, and equal to 1 if the feature subset has maximum predictive power.
- A generic version of H (not depending on p) can be created by setting $p = 0.5$. Then the W functions are not shaped like a W anymore, they are shaped like a V.

2. Data structure, computations

You can pre-compute all the bin counts n_k for the top 20 features (that is, features with highest predictive power) and store them in a small hash table with at most $2 * 2^{20}$ entries (approx. 2 million; the factor two is because you need two measurements per bin: number of B's, and number of G's). An entry in this hash table would look like

```
$Hash{01101001010110100100_G} = 56,
```

meaning that Bin # 01101001010110100100 has 56 good (G) observations.

The hash table is produced by parsing your training set one time, sequentially: for each observation, compute the flag vector (which rules are triggered, that is the 01101001010110100100 vector in this example), check if it's good or bad, and update (increase count by 1) the associated hash table entry accordingly, with the following instruction:

```
$Hash{01101001010110100100_G}++
```

Then whenever you need to measure the predictive power of a subset of these 20 features, you don't need to parse your big data set again (potentially billion of observations), but instead, just access this small hash table: this table contains all you need to build your flag vectors and compute scores, for any combination of features that is a subset of the top 20.

You can even do better than top 20, maybe top 30. While this would create a hash table with 2 billion entries, most of these entries would correspond to empty bins and thus would not be in the hash table. Your hash table might contain only 200,000,000 entries, maybe too big to fit in memory, and requiring a Map Reduce / Hadoop implementation. Even better: build this hash table for the top 40 features. Then it will fully solve your feature selection problem described earlier. However now, your hash table could have up to 2 trillion entries. But if your dataset only has 100 billion observations, then of course your hash table cannot have more than 100 billion entries. In this case, I suggest that you create a training set with 20 million observations, so that your hash table will have at most 20 million entries (and probably less than 10 million non-empty bins). Thus, it can fit in memory.

You can compute the predictive power of a large number (say 100) of feature subsets by parsing the big 40-feature input hash table obtained in the previous step, then for each flag vector and G/B entry in the input hash table, loop over the 100 target feature subsets to update counts (the n_k's) for these 100 feature subsets: these counts are stored / updated in an output hash table. The key in the output hash table has two components: feature ID and flag vector. You then loop over the output hash table to compute the predictive power for each feature subset. This step can be further optimized.

5. Fast Clustering for Big Data

Here we discuss two potential algorithms that can perform fast clustering on big data sets, as well as the graphical representation of such complex clustering structures. By fast, we mean a computational complexity of order *O(n)* and even faster such as *O(n* / log *n)*. This is much faster than good Hierarchical Agglomerative Clustering which are typically $O(n^2 \log n)$. By big data, we mean several millions, possibly billions of observations.

Potential applications:

- Creating a keyword taxonomy to categorize the entire universe of cleaned (standardized), valuable English keywords. We are talking of about 10 million keywords made up of one, two or three tokens, that is, about 300 times the number of keywords found in a good English dictionary. The purpose might be to categorize all bid keywords that could be purchased by eBay and Amazon on Google (for pay-per-click ad campaigns), to better price them. This is the application discussed in this chapter.
- Clustering millions of documents (e.g. books on Amazon.com) or
- Clustering web pages, or even the entire Internet, which consists of about 100 million top websites and billions of web pages.

We also discuss whether it makes sense to perform such massive clustering, and how Map Reduce can help.

1. Building a keyword taxonomy

Here's the answer, from my earlier article What MapReduce can't do. Step 2 is the clustering part.

Step #1: pre-processing

You gather tons of keywords over the Internet with a web crawler (crawling Wikipedia or Google), and compute the frequencies for each keyword, and for each “keyword pair”. A “keyword pair” is two keywords found on a same webpage, or close to each other on a same web page. Also by keyword, I mean stuff like "California insurance", so a keyword usually contains more than one token, but rarely more than three. With all the keyword frequencies, you can create a table (typically containing many million keywords, even after keyword cleaning), where each entry is a pair of keywords and 3 numbers, e.g.

A=”California insurance”, B=”home insurance”, x=543, y=998, z=11

where

- *x* is the number of occurrences of keyword A in all the web pages crawled
- *y* is the number of occurrences of keyword B in all the web pages crawled
- *z* is the number of occurrences where A and B form a pair (e.g. they are found on a same page)

This “keyword pair” table can be easily and efficiently built using MapReduce (distributed architecture). Note that the vast majority of keywords A and B do not form a “keyword pair”, in other words, *z*=0. So by ignoring these null entries, your “keyword pair” table is still manageable, and might contain as little as 100 million entries.

Note: This step #1 constitutes the final step of a number of interesting applications. For instance, it is used in search engine technology to identify or recommend keywords related to some other keywords. See example here.

Step #2: clustering

To create a taxonomy, you want to group the keywords found into similar clusters. One way to do it is to compute a dissimilarity d(A, B) between two keywords A, B. For instances d(A, B) = $z / (xy)^{1/2}$, although other choices are possible. Note that the denominator prevents extremely popular keywords (e.g. “free”) from being close to all the keywords, and from dominating the entire keyword relationship structure: indeed, it favors better keyword bonds, such as “lemon” with “law” or “pie”, rather than “lemon” with “free”.

The higher d(A, B), the closer keywords A and B are to each other. Now the big problem is to perform clustering - any kind of clustering, e.g. hierarchical - on the “keyword pair” table, using any kind of dissimilarity metric. We now discuss our solution, and a potential alternate solution.

2. Fast clustering algorithm

While this algorithm is described in the context of keyword clustering, it is straightforward to adapt it to other contexts. Here we assume that we have *n* = 10,000,000 unique keywords and *m* = 100,000,000 keyword pairs {A, B}, where d(A,B)>0. That is, an average of *r* = 10 related keywords attached to each keyword. Our algorithm incrementally proceeds in several (5 or 6) rounds, as follows:

BEGIN

Initialization (Round #0): The small data (or seeding) step

Select 10,000 seed keywords, create (say) 100 categories and create a hash table `$hash` where the key is one of the 10,000 seed keywords, and the value is a list of categories the keyword is assigned to.

For instance, `$hash{"cheap car insurance"} = {"automotive", "finance"}`

The choice of the initial 10,000 seed keywords is very important. I suggest to pick up the top 10,000 keywords, in terms of number of associations: that is, keywords A with many B's where d(A, B) > 0. This will speed up the convergence of the algorithm.

Round #1: The big data step

Browse the table of *m* keyword pairs, from beginning to end.

When you find a pair {A, B} where (say) `$hash{A}` exists and $hash{B} does not, do:

- `$hash{B} = $hash{A};`
- `$weight{B} = d(A, B)`

When you find a pair {A, B} where both A and B are already in `$hash`, do

- `if $d(A,B) > $weight(B) then { $hash{B} = $hash{A}; $weight{B} = $d(A, B); } # Note: B gets re-categorized to A's category`
- `if $d(A,B) > $weight(A) then { $hash{A} = $hash{B}; $weight{A} = $d(A, B); } # Note: A gets re-categorized to B's category`

Round #2: Repeat Round #1 ($hash and $weight are kept in memory and keep growing at each subsequent round)

Round #3: Repeat Round #1, one more time

Round #4: Repeat Round #1, one more time

Round #5: Repeat Round #1, one more time

END

The computational complexity is $qm = O(n)$, with q being the number of rounds. This is n = 10,000,000 times faster than good clustering algorithms. However, all these hash table accesses will slow it a bit to $O(n \log n)$, as `$hash` and `$weight` grow bigger at each subsequent round.

Would pre-sorting the big table of *m* pairs help? Sorting by d(A, B) would allow us to drastically reduce the number of hash table accesses (by making all the re-categorizations not needed anymore), but sorting is $O(n \log n)$, so we would not gain anything. Note that sorting can be efficiently performed with Map Reduce. The reduce step in this case, consists of merging a bunch of small, sorted tables.

This clustering algorithm seems easy to implement using Map Reduce (a distributed architecture), however since the big table only has 100,000,000 entries, it might fit in RAM.

You can improve the computational complexity by keeping the most important *m* / log *n* entries (based on volume and d(A,B)) in the big table, and deleting the remaining entries. In practice, deleting 65% of the big table (the *very* long tail only, but not the entire long tail, from a keyword distribution point of view) will have very little impact on the performance: you will have a large bucket of un-categorized keywords, but in terms of volume, these keywords might represent less than 0.1%.

Comments

- **Alternate algorithm**: One could use Tarjan's strongly connected components algorithm to perform the clustering. To proceed, you first bin the distances: d(A, B) is set to 1 if it is above some pre-specified threshold, 0 otherwise. This is a graph theory algorithm: each keyword represents a node, each pair of keywords where d(A, B) = 1, represents an edge. The computational complexity of the algorithm is $O(n + m)$, where *n* is the number of keywords and *m* is the number of pairs (edges). To take advantage of this algorithm, you might want to store the big "keyword pair" table in a graph database (a type of NoSQL database).

- **Visualization**. How do you represent these keywords, with their cluster structure determined by d(A, B), in a nice graph? 10 million keywords would fit in a 3,000 x 3,000 pixels image. For those interested in graphical representations, see the Fruchterman and Rheingold algorithm, extensively used to produce such graphs. Note that its computational complexity is $O(n^3)$ though, so we need to very significantly improve it for this keyword clustering application - including the graphical representation. The graphical representation could be a raster image with millions of pixels, like a heat map where color represents category and, when you point to a pixel, a keyword value shows up (rather than a vector image with dozens of nodes, see graph below). Neighboring pixels would represent strongly related keywords.

6. Structuring Unstructured Data

You have gathered gigabytes or terabytes of unstructured text, for instance scraping the Internet, or pieces of email from your employees or users, or tweets, or millions of products that you want to categorize (only product description and product name is available - sometimes with typos). Now you want to make sense of it, and extract value, possibly design a nice search engine so that your customers can easily find your products. The core algorithm that you need is an automated cataloguer, also called indexer. I explain here in layman's terms how it works. First, let's assume that the data consists of

- Pages or articles (a web page or the body of an email, etc.)
- Subject lines (or page titles),
- Authors (for a web page or an email).

Typically, these “pages” are stored as large repositories containing millions or billions of (sometimes compressed) text files spread across a number of folders and sub-folders, or multiple servers. Sometimes a time stamp is attached to each document, and can be leveraged to increase the accuracy of the indexer.

The technique also works even if you only have pages (no user information, no titles). If you have pages and authors, you can classify the pages separately, then the authors separately (or in parallel), then blend the results to maximize accuracy. The same indexation algorithm (sometimes called tagging algorithm) is used in both cases. Despite the fact that classifying billions of documents seems mathematically unfeasible due to the computational complexity of traditional clustering algorithms (the time spent to cluster is growing much faster than linearly, as a function of the size of your repository), this algorithm is different, run very fast, and is easy to implement using a distributed architecture.

The indexer algorithm creates a taxonomy of your pages (or products, articles, documents etc.) Each page is assigned a category and sub-category.

1. Indexation algorithm

- *Step 1*: Create a data dictionary (that is, a frequency table, see section 8 in chapter 25) of all one-token and two-token keywords found in all pages (both in the title and in the body of the article). This assumes that you crawled all your articles to extract all the text.
- *Step 2*: Filter / clean results. Ignore keywords with less than 5 occurrences. Check all *n*-grams of a keyword (*data science* and *science data*) and eliminate *n*-grams with low frequency, for each keyword
- *Step 3*: Look at top 300 entries, called *seed keywords*. Manually assign seed keywords to 10-20 categories, (these categories are manually pre-selected, after looking at the top 300 entries.) For instance, the top category *data plumbing* will

have the following seed keywords: data engineer, data architect, data warehouse, Hadoop, Spark, data lake, IoT and many more. Don't forget to have a top category called *Unknown*.

- *Step 4*. Based on keywords found in the title and body of an article, assign the article in question to the top category that has the biggest overlap with the article, in terms of seed keywords. Note that keywords found in the title might be assigned a higher weight than those found in the body. Likewise, a different weight can be attached to each seed keyword, in each top category.

This technique is called *indexation* because it is very similar to the creation of a search engine. A business application is described in chapter 20.

2. Potential improvement

These improvements will boost the performance (accuracy).

- Add 3-token keywords in your dictionary, not just 1- and 2-token. For 3-token keywords, you have 3! (factorial 3) = 6 *n*-grams. Usually, only one or two of these 6 *n*-grams will show up in the articles, for any keyword (*data science central* will show up, but *central science data* won't).
- Use stop words to clean your data. Examples: it, where, how, why, for and so on. Be careful though: *IT Job* cannot be reduced to *Job* by filtering out the token *IT*. You can replace plurals by singular, and normalize the keywords.
- Some one-token words don't make sense. Do not break “San Francisco” into “San” and “Francisco”. Used a table of keywords that should not be split.

Even without improvements, the methodology will work well, because you focus on top keywords in terms of frequency. For instance, in *Best San Francisco Hotels*, the keywords *Best San* and *Francisco Hotels* won't show up at the top, and if they do, you can remove them, as you manually review the top 3,000 entries (a manual process that takes 30 minutes).

Finally, you can use the BerkeleyDB open source software (combined with a bunch of lookup tables such as stop keywords, synonyms and so on) to do many of these tasks.

7. Testing for Randomness

This chapter is intended for practitioners who might not necessarily be statisticians or statistically-savvy. The mathematical level is kept as simple as possible, yet I present an original, simple approach to test for randomness, with an interesting application to illustrate the methodology. This material is not something usually discussed in textbooks or classrooms (even for statistical students), offering a fresh perspective, and out-of-the-box tools that are useful in many contexts, as an addition or alternative to traditional tests that are widely used. This chapter is written as a tutorial, but it also features an interesting research result in the last section.

1. Context

Let us assume that you are dealing with a time series with discrete time increments (for instance, daily observations) as opposed to a time-continuous process. The approach here is to apply and adapt techniques used for time-continuous processes, to time-discrete processes. More specifically (for those familiar with stochastic processes) we are dealing here with discrete Poisson processes. The main question that we want to answer is: Are some events occurring randomly, or is there a mechanism making the events not occurring randomly? What is the gap distribution between two successive events of the same type?

In a time-continuous setting (Poisson process) the distribution in question is modeled by the exponential distribution. In the discrete case investigated here, the discrete Poisson process turns out to be a Markov chain, and we are dealing with geometric, rather than exponential distributions. Let us illustrate this with an example.

Example

The digits of $2^{1/2}$ are believed to be distributed as if they were occurring randomly. Each of the 10 digits 0, 1, ... , 9 appears with a frequency of 10% based on observations, and at any position in the decimal expansion of $2^{1/2}$, on average the next digit does not seem to depend on the value of the previous digit (in short, its value is unpredictable.) An event in this context is defined, for example, as a digit being equal to (say) 3. The next event is the first time when we find a subsequent digit also equal to 3. The *gap* (or time elapsed) between two occurrences of the same digit is the main metric that we are interested in, and it is denoted as *G*. If the digits were distributed just like random numbers, the distribution of the gap *G* between two occurrences of the same digit, would be geometric, that is,

$$P(G=k)=(1-p)^{k-1}p^{k},\ k=1,2,\cdots$$

with *p* = 1/10 in this case, as each of the 10 digits (0, 1, ..., 9) seems -- based on observations -- to have a frequency of 10%. We will show that this is indeed the case: In other words, in our example, the gap *G* is very well approximated by a geometric distribution of parameter $p = 1/10$, based on an analysis of the first 10 million digits of $2^{1/2}$.

What else should I look for, and how to proceed?

Studying the distribution of gaps can reveal patterns that standard tests might fail to catch. Another statistic worth studying is the maximum gap, see chapter 14. This is sometimes referred to as extreme events / outlier analysis. Also, in our above example, studying gaps between groups of digits (not just single digits, but for instance how frequently the "word" 234567 repeats itself in the sequence of digits, and what is the distribution of the gap for that word. For any word consisting of 6 digits, $p = 1 / 1{,}000{,}000$. In our case, our data set only has 10 million digits, so you may find 234567 maybe only 2 times, maybe not even once, and looking at the gap between successive occurrences of 234567, is pointless. Shorter words make more sense. This and other issues are discussed in the next section.

2. Methodology

The first step is to estimate the probabilities *p* associated with the model, that is, the probability for a specific event, to occur at any time. It can easily be estimated from your data set, and generally, you get a different *p* for each type of event. Then you need to use an algorithm to compute the empirical (observed) distribution of gaps between two successive occurrences of the same event. In our example, we have 10 types of events, each associated with the occurrence of one of the 10 digits 0, 1,..., 9 in the decimal representation of $2^{1/2}$. The gap computation can be efficiently performed as follows:

Algorithm to compute the observed gap distribution

Do a loop over all your observations (in our case, the 10 first million digits of $2^{1/2}$, stored in a file; each of these 10 million digits is one observation). Within the loop, at each iteration *t*, do:

- Let *E* be the event showing up in the data set, at iteration *t*. For instance, the occurrence of (say) digit 3 in our case. Retrieve its last occurrence stored in an array, say `LastOccurrences[E]`
- Compute the gap *G* as `G = t - LastOccurrences[E]`
- Update the LastOccurrences table as follows: `LastOccurrences[E] = t`
- Update the gap distribution table, denoted as GapTable (a two-dimensional array or better, an hash table) as follows: `GapTable[E, G]++`

Once you have completed the loop, all the information that you need is stored in the GapTable summary table.

Statistical testing

If some events occur randomly, the theoretical distribution of the gap, for these events, is known to be geometric, see above formula in first section. So you must test whether the empirical gap distribution (computed with the above algorithm) is statistically different from the theoretical geometric distribution of parameter *p* (remember that each type of event may have a different *p*.) If not statistically different, then the assumption of randomness should be discarded: you've found some patterns. This work is typically done using a Kolmogorov- Smirnov test. If you are not a statistician but instead a BI analyst or engineer, other techniques can be used instead, and are illustrated in the last section:

- You can simulate events that are perfectly randomly distributed, and compare the gap distribution obtained in your simulations, with that computed on your observations. See here how to do it, especially the last comment featuring an efficient way to do it. This Monte-Carlo simulation approach will appeal to operations research analysts.

- In Excel, plot the gap distribution computed on your observations (one for each type of event), add a trendline, and optionally, display the trendline equation and its R-Squared. When choosing a trendline (model fitting) in Excel, you must select the Exponential one. This is what we did (see next section) and the good news is that, despite the very limited selection of models that Excel offers, Exponential is one of them. You can actually test other trendlines in Excel (polynomial, linear, power, or logarithmic) and you will see that by far, Exponential offers the best fit -- if your events are really randomly distributed.

Further advice

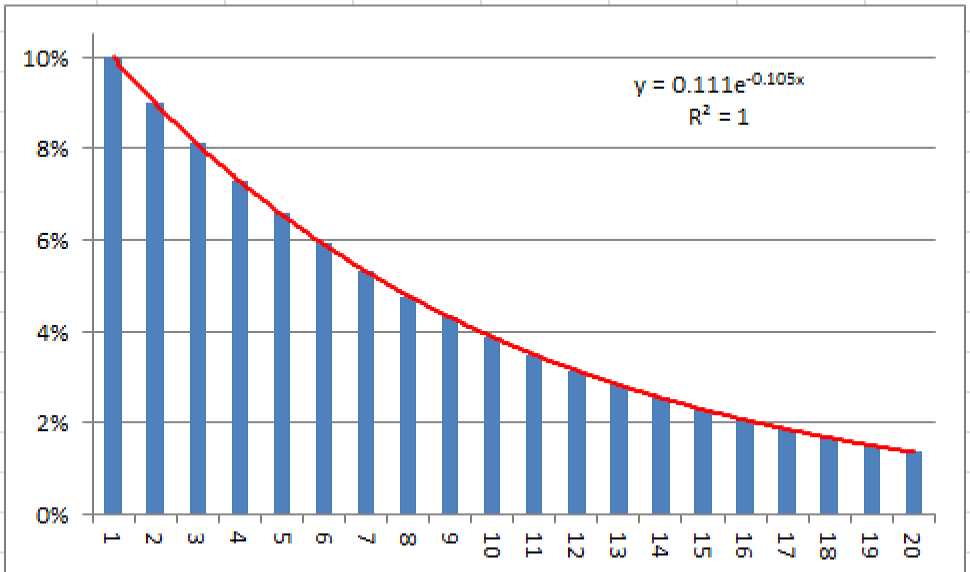
If you have collected a large number of observations (say 10 million) you can do the testing on samples of increasing sizes (1,000, 10,000, 100,000 consecutive observations and so on) to see how fast the empirical distribution converges (or not) to the theoretical geometric distribution. You can also compare the behavior across samples (cross-validation), or across types of events (variance analysis). If your data set is too small (100 data points) or your events too rare (*p* less than 1%), consider increasing the size of your data set if possible.

Even with big data, if you are testing a large number of rare events (in our case, tons of large “words” such as occurrences 234567 rather than single digits in the decimal representation of $2^{1/2}$ expect many tests to result in false negatives (failure to detect true randomness.) You can even compute the probability for this to happen, assuming all your events are perfectly randomly distributed. This is known as the curse of big data.

3. Application to Number Theory Problem

Here, we further discuss the example used throughout this chapter to illustrate the concepts. Mathematical constants (and indeed the immense majority of all numbers) are thought to have their digits distributed as if they were randomly generated, see chapter 10 for details.

Many tests have been performed on many well-known constants (see here), and none of them was able to identify any departure from randomness. The gap test illustrated here is less well known, and when applied to $2^{1/2}$, it was also unable to find departure from randomness. In fact, the fit with a random distribution, as shown in the figure below, is almost perfect.



Gaps of length 1 to 20, with their frequency (red curve is theoretical geometric distribution, or Exponential trendline in Excel)

There is a simple formula to compute any digit of $2^{1/2}$ separately, see here, however it is not practical. Instead, we used a table of 10 million digits published here by NASA. The source claims that digits beyond the first five million have not been double-checked, so we only used the first 5 million digits. The summary gap table, methodological details, and the above picture, can be found in my spreadsheet. You can download it here.

The above chart shows a perfect fit between the observed distribution of gap lengths (averaged across the 10 digits 0, 1, ..., 9) between successive occurrences of a same digit in the first 5 million decimals of $2^{1/2}$, and the geometric distribution model, using the Exponential trendline in Excel.

I also explored the last 2 million decimals available in the NASA table and despite the fact that they have not been double-checked, they also display the exact same random behavior. Maybe these decimals are all wrong but the mechanism that generates them preserves randomness, or maybe all or most of them are correct.

A counter-example

The number 0.123456789101112131415161718192021... known as the Champernowne constant, and obtained by concatenating the decimal representations of the natural numbers in order, has been proved to be "random", in the sense that no digit or group of digits, occurs more frequently than any other. Such a number is known as a normal number. However, it fails miserably the gap test, with the limit distribution for the gaps (if it even exists) being totally different from a geometric distribution. I tested it on the first 8, 30, 50, 100 and 400 million decimals, and you can try too, as an exercise. All tests failed dramatically.

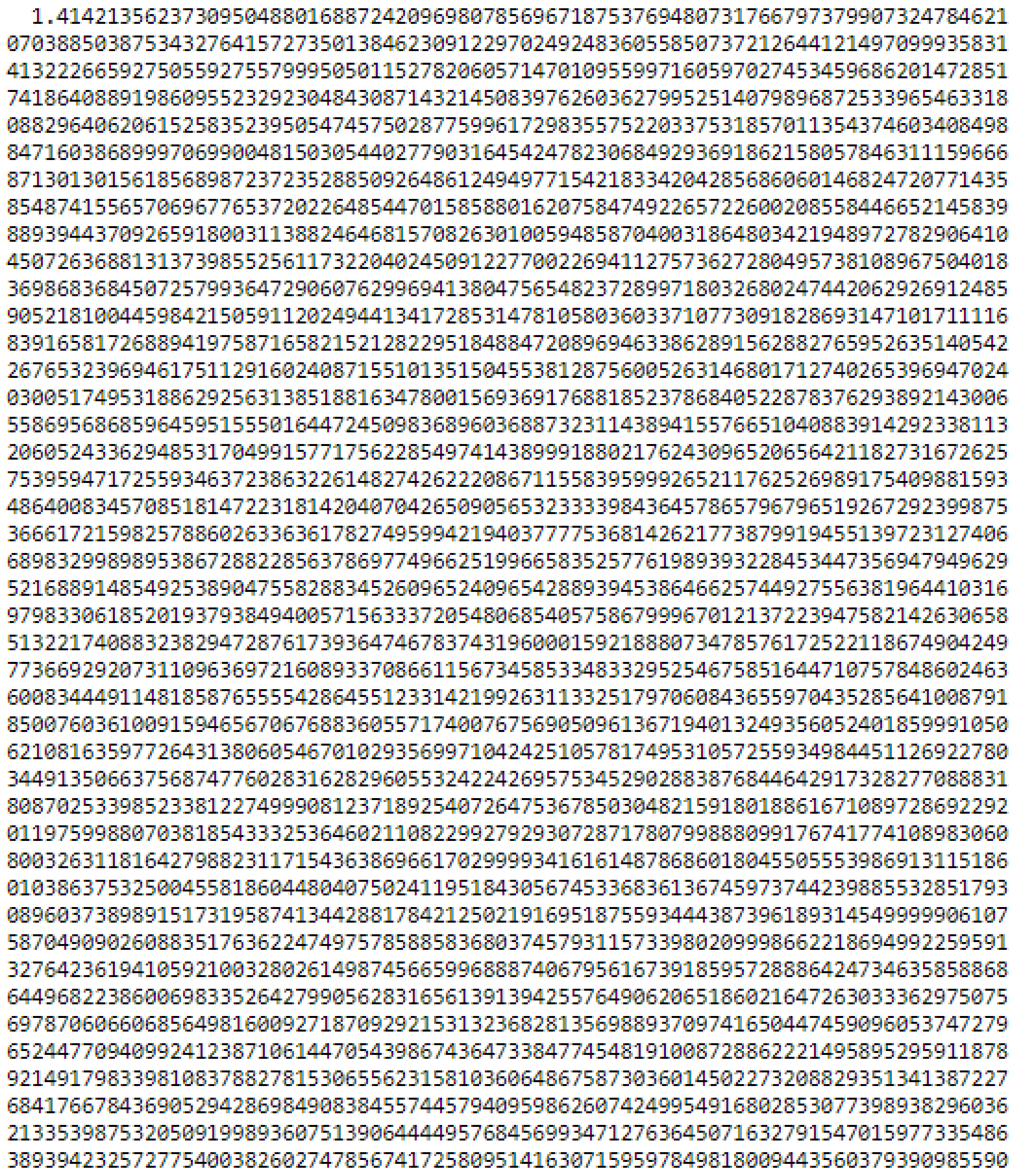
Ironically, no one known if $2^{1/2}$ is a normal number, yet it passed the gap test incredibly well. Maybe a better definition of a "random" number, rather than being normal, would be a number with a geometric distribution as the limit distribution for the gaps. Can you create an artificial number that passes this test, yet exhibits strong patterns of non-randomness? Is it possible to construct a non-normal number that passes the gap test?

Potential use in cryptography

A potential application is to use digits that appear to be randomly generated (like white noise, and the digits of $2^{1/2}$ seem to fit the bill) in documents, at random positions that only the recipient could reconstruct, perhaps three or four random digits on average for each real character in the original document, before encrypting it, to increase security -- a bit like steganography. Encoding the same document a second time would result in a different kind of white noise added to the original document, and peppered randomly, each time differently -- with a different intensity, and at different locations each time. This would make the task of hackers more complicated.

4. Conclusion

Finally, this is an example where intuition can be wrong, and why you need data science. In the digits of $2^{1/2}$, while looking at the first few thousand digits (see picture below), it looked to me like it was anything but random. There were too many 99, two few 37 (among other things), according to my intuition and visual inspection (you may call it *gut feelings*.) It turns out that I was wrong. Look at the first few thousand digits below, chances are that your intuition will also mislead you into thinking that there are some patterns. This can be explained by the fact that patterns such as 99 are easily detected by the human brain and do stand out visually, yet in this case, they do occur with the right frequency if you use analytic tools to analyze the digits.



```
  1.4142135623730950488016887242096980785696718753769480731766797379907324784621
070388503875343276415727350138462309122970249248360558507372126441214970999358314
1322266592750559275579995050115278206057147010955997160597027453459686201472851
741864088919860955232923048430871432145083976260362799525140798968725339654633180
882964062061525835239505474575028775996172983557522033753185701135437460340849884
716038689997069900481503054402779031645424782306849293691862158057846311159666871
301301561856898723723528850926486124949771542183342042856860601468247207714358548
741556570696776537202264854470158588016207584749226572260020855844665214583988939
443709265918003113882464681570826301005948587040031864803421948972782906410450726
368813137398552561173220402450912277002269411275736272804957381089675040183698683
684507257993647290607629969413804756548237289971803268024744206292691248590521810
044598421505911202494413417285314781058036033710773091828693147101711116839165817
268894197587165821521282295184884720896946338628915628827659526351405422676532396
946175112916024087155101351504553812875600526314680171274026539694702403005174953
188629256313851881634780015693691768818523786840522878376293892143006558695686859
645951555016447245098368960368873231143894155766510408839142923381132060524336294
853170499157717562285497414389991880217624309652065642118273167262575395947172559
346372386322614827426222086711558395999265211762526989175409881593486400834570851
814722318142040704265090565323333984364578657967965192672923998753666172159825788
602633636178274959942194037777536814262177387991945513972312740668983299898953867
288228563786977496625199665835257761989393228453447356947949629521688914854925389
047558288345260965240965428893945386466257449275563819644103169798330618520193793
849400571563337205480685405758679996701213722394758214263065851322174088323829472
876173936474678374319600015921888073478576172522118674904249773669292073110963697
216089337086611567345853348332952546758516447107578486024636008344491148185876555
542864551233142199263113325179706084365597043528564100879185007603610091594656706
768836055717400767569050961367194013249356052401859991050621081635977264313806054
670102935699710424251057817495310572559349844511269227803449135066375687477602831
628296055324224269575345290288387684464291732827708883180870253398523381227499908
123718925407264753678503048215918018861671089728692292011975998807038185433325364
602110822992792930728717807998880991767417741089830608003263118164279882311715436
386966170299993416161487868601804550555398691311518601038637532500455818604480407
502411951843056745336836136745973744239885532851793089603738989151731958741344288
178421250219169518755934443873961893145499999061075870490902608835176362247497578
588583680374579311573398020999866221869499225959132764236194105921003280261498745
665996888740679561673918595728886424734635858868644968223860069833526427990562831
656139139425576490620651860216472630333629750756978706066068564981600927187092921
531323682813569889370974165044745909605374727965244770940992412387106144705439867
436473384774548191008728862221495895295911878921491798339810837882781530655623158
103606486758730360145022732088293513413872276841766784369052942869849083845574457
940959862607424995491680285307739893829603621335398753205091998936075139064444957
684569934712763645071632791547015977335486389394232572775400382602747856741725809
514163071595978498180094435603793909855901
```

```
  1.4142135623730950488016887242096980785696718753769480731766797379907324784621
0703885038753432764157273501384623091229702492483605585073721264412149709993583
1413222665927505592755799950501152782060571470109559971605970274534596862014728
5174186408891986095523292304843087143214508397626036279952514079896872533965463
3180882964062061525835239505474575028775996172983557522033753185701135437460340
849884716038689997069900481503054402779031645424782306849293691862158057846311159666
```

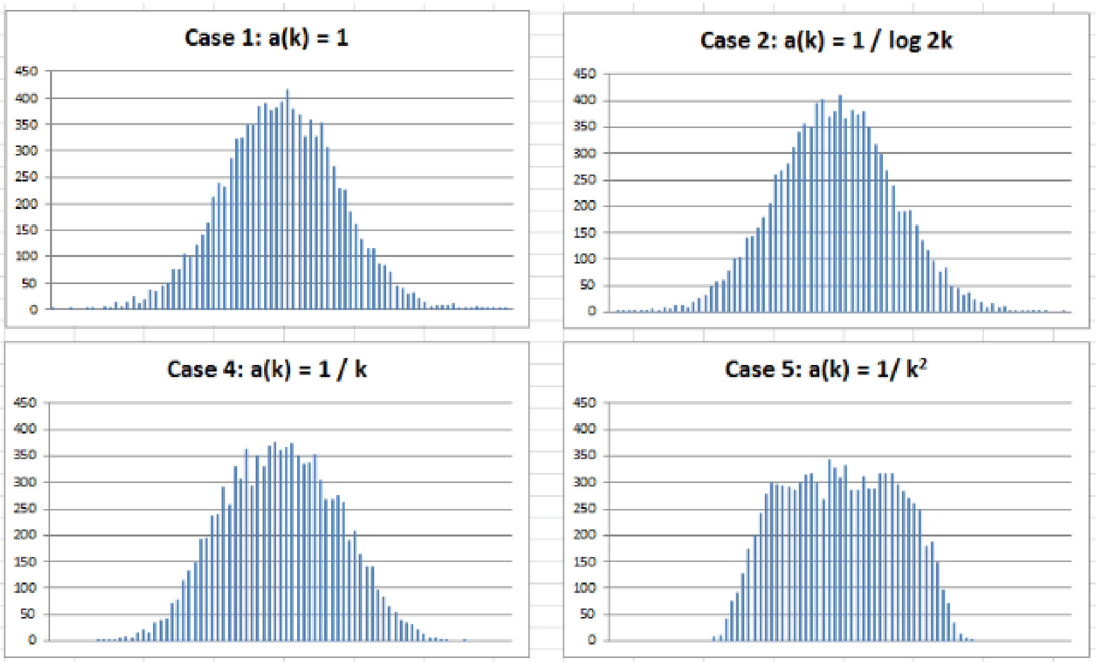
First few hundred digits of $2^{1/2}$. Do you see any pattern?

8. Central Limit Theorem Revisited

In this chapter, we explore in layman's terms the most fundamental statistics theorem. We investigate a special but interesting and useful case, which is not discussed in textbooks, data camps, or data science classes. This material is part of a series about off-the-beaten-path data science and mathematics, offering a fresh, original and simple perspective on a number of topics.

The theorem discussed here is the central limit theorem. It states that if you average a large number of well-behaved observations or errors, eventually, once normalized appropriately, it has a standard normal distribution. Despite the fact that we are dealing here with a more advanced and exciting version of this theorem (discussing the Lyapunov condition), we focus on applications

In short, we are dealing here with not-so-well-behaved observations, and we show that even in that case, the limiting distribution of the "average" can be normal (Gaussian.). More precisely, we show when it is and when it is not normal, based on simulations and non-standard (but easy to understand) statistical tests.



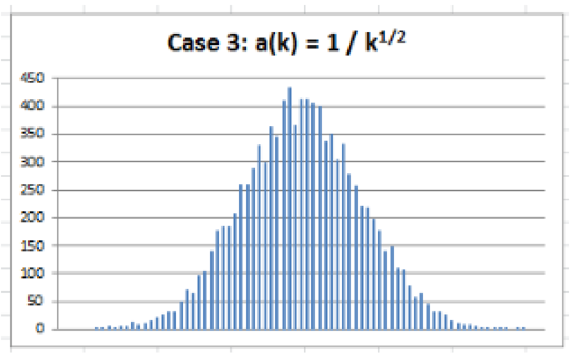


Figure 1*: Cases #1, 2 and 3 (section 2) show convergence to the Gaussian distribution (Click here for a higher resolution picture)*

1. A special case of the Central Limit Theorem

Let's say that we have *n* independent observations $X_1,..., X_n$ and we compute a weighted sum

$$S = a_1X_1 + ... + a_nX_n.$$

Under appropriate conditions to be discussed later, (*S* - E(*S*)) / Stdev(*S*) has a normal distribution of mean 0 and variance 1. Here E denotes the expectation and Stdev denotes the standard deviation, that is, the square root of the variance.

This is a non-basic case of the central limit theorem, as we are dealing with a weighted sum. The classic, basic version of the theorem assumes that all the weights a_1, ..., a_n are equal. Furthermore, we focus here on the particular case where

- The highest weights (in absolute value) are concentrated on the first few observations,
- The weight a_k tends to 0 as *k* tends to infinity.

The surprising result is the fact that even with putting so much weight on the first few observations, depending on how slowly a_k converges to 0, the limiting distribution is still Gaussian.

Context

You might wonder: how is this of any practical value in the data sets that I have to process in my job? Interestingly, I started to study this type of problems long ago, in the context of *k*-NN (nearest neighbors) classification algorithms. One of the questions, to estimate the local or global intensity of a stochastic point process, and also related to density estimation techniques, was: how many neighbors should we use, and which weights should we put on these neighbors to get robust and accurate estimates? It turned out that putting more weight on close neighbors, and increasingly lower weight on far away neighbors (with weights slowly decaying to zero based on the distance to the neighbor in question) was the solution to the problem. I actually found optimum decaying schedules for the a_k's, as k tends to infinity. You can read the detail here.

2. Simulations, testing, and conclusions

Let's get back to the problem of assessing when the weighted sum $S = a_1X_1 + ... + a_nX_n$, after normalization, converges to a Gaussian distribution. By normalization, I mean considering (S - E(S)) / Stdev(S), instead of S.

In order to solve this problem, we performed simulations as follows:

Simulations

Repeat m = 10,000 times:

- Produce n = 10,000 random deviates $X_1, ..., X_n$ uniformly distributed on [0, 1]
- Compute $S = a_1X_1 + ... + a_nX_n$ based on a specific set of weights $a_1, ..., a_n$
- Compute the normalized S, denoted as W = (S - E(S)) / Stdev(S).

Each of the above m iterations provides one value of the limiting distribution. In order to investigate the limiting distribution (associated with a specific set of weights), we just need to look at all these m values, and see whether they behave like deviates from a Gaussian distribution of mean 0 and variance 1. Note that we found n = 10,000 and m = 10,000 to be large enough to provide relatively accurate results. We tested various values of n before settling for n = 10,000, looking at what (little) incremental precision we got from increasing n (say) from 500 to 2,000 and so on. Also note that the random number generator is not perfect, and due to numerical approximations made by the computer, indefinitely increasing n (beyond a certain point) is not the solution to get more accurate results. That said, since we investigated 5 sets of weights, we performed 5 x n x m = 500 million computations in very little time. A value of m = 10,000 provides about two correct digits when computing the percentiles of the limiting distribution (except for the most extreme ones), provided n is large enough. The source code is provided in the last section.

Analysis and results

We tested 5 sets of weights, see Figure1:

- **Case 1**: $a_k = 1$, corresponding to the classic version of the Central Limit Theorem, and with guaranteed convergence to the Gaussian distribution.
- **Case 2**: $a_k = 1 / \log 2k$, still with guaranteed convergence to the Gaussian distribution
- **Case 3**: $a_k = k^{-1/2}$, the last exponent (-1/2) that still provides guaranteed convergence to the Gaussian distribution, according to the Central Limit Theorem

with the Lyapunov condition (more on this below.) A value below -1/2 violates the Lyapunov condition.

- **Case 4**: $a_k = k^{-1}$, the limiting distribution looks Gaussian (see Figure 1) but it is too thick to be Gaussian, indeed the maximum is also too low, and the kurtosis is now significantly different from zero, thus the limiting distribution is not Gaussian (though almost).
- **Case 5**: $a_k = k^{-2}$, not converging to the Gaussian distribution, but instead to an hybrid continuous distribution, half-way Gaussian, half-way uniform.

Note that by design, all normalized *S*'s have mean 0 and variance 1.

We computed (in Excel) the percentiles of the limiting distributions for each of the five cases. Computations are found in this spreadsheet. We compared the cases 2 to 5 with case 1, computing the differences (also called deltas) for each of the 100 percentiles. Since case 1 corresponds to a normal distribution, we actually computed the deltas to the normal distribution, see Figure 2. The deltas are especially large for the very top or very bottom percentiles in cases 4 and 5. Cases 2 and 3 show deltas close to zero (not statistically significantly different from zero), and this is expected since these cases also yield a normal distribution. To assess the statistical significance of these deltas, one can use the model-free confidence interval technique described here: it does not require any statistical or probability knowledge to understand how it works. Indeed you don't even need a table of the Gaussian distribution for testing purposes here (you don't even need to know what a Gaussian distribution is) as case 1 automatically provides one.

The Lyapunov connection

For those interested in the theory, the fact that cases 1, 2 and 3 yield convergence to the Gaussian distribution is a consequence of the Central Limit Theorem under the Lyapunov condition. More specifically, and because the samples produced here come from uniformly bounded distributions (we use a random number generator to simulate uniform deviates), all that is needed for convergence to the Gaussian distribution is that the sum of the squares of the weights -- and thus Stdev(*S*) as *n* tends to infinity -- must be infinite. This result is mentioned in A. Renyi's book *Probability Theory* (Dover edition, 1998, page 443.)

Note that in cases 1, 2, and 3, the sum of the squares of the weights is infinite. In cases 4 and 5, it is finite, respectively equal to $\pi^2/6$ and $\pi^4/90$ (see here for details.) I am very curious to know what the limiting distribution is for case 4.

3. Generalizations

Here we discuss generalizations of the central limit theorem, as well as potential areas of research

3.1. Correlated observations

One of the simplest ways to introduce correlation is to define a stochastic auto-regressive process using

$$Y_k = pY_{k-1} + qX_k$$

where X_1, X_2, ... are independent with identical distribution, with $Y_1 = X_1$ and where p, q are positive integers with $p + q = 1$. The Y_k's are auto-correlated, but clearly, Y_k is a weighted sum of the X_j's ($1 \leq j \leq k$), and thus, $S = a_1Y_1 + ... + a_nY_n$ is also a weighted sum of the X_k's, with higher weights on the first X_k's. Thus we are back to the problem discussed in this chapter, but convergence to the Gaussian distribution will occur in fewer cases due to the shift in the weights.

More generally, we can work with more complex auto-regressive processes with a covariance matrix as general as possible, then compute S as a weighted sum of the X_k's, and find a relationship between the weights and the covariance matrix, to eventually identify conditions on the covariance matrix that guarantee convergence to the Gaussian distribution.

3.2. Generalization to non-random (static) observations

Is randomness really necessary for the central limit theorem to be applicable and provable? What about the following experiment:

- Compute all unordered sums S made up of n integers, 0 or 1, with repetitions allowed. For instance, if $n = 2$, the four possibilities are 0+0, 0+1, 1+0, 1+1. For an arbitrary n, we have 2^n possibilities.
- Normalize S as usual. For normalization, here use $E(S) = n/2$ and $\text{Stdev}(S) = n^{1/2} / 2$.

Do these 2^n normalized values of S (generated via this non-random experiment) follow a Gaussian distribution as n tends to infinity? Ironically, one way to prove that this is the case (I haven't checked if it is the case or not, but I suspect that it is) would be to randomly sample m out of these 2^n values, and then apply the central limit theorem to the randomly selected values as m tends to infinity. Then by increasing m until m is as large as 2^n we would conclude that the central limit theorem also holds for the non-random (static) version of this problem. The limiting distribution definitely has a symmetric, bell-like shape, just like the Gaussian distribution, though this was also the case in our above "case 4" example -- yet the limit was not Gaussian.

3.3. Other interesting stuff related to the Central Limit Theorem

There is a lot of interesting stuff on the Wikipedia entry, including about the Lyapunov condition. But the most interesting things, at least in my opinion, were the following:

- The area S of a convex hull of n points X_1, ..., X_n also converges to a normal distribution, once standardized, that is when considering (S - E(S)) / Stdev(S).
- Under some conditions, the result below applies, with C being a universal constant:

$$\left|\mathbb{P}\left(a \leq \frac{X_1 + \cdots + X_n}{\sqrt{n}} \leq b\right) - \frac{1}{\sqrt{2\pi}} \int_a^b \mathrm{e}^{-t^2/2}\, \mathrm{d}t\right| \leq \frac{C}{n}$$

- If instead of a weighted average S, we consider the maximum M = max(X_1, ..., X_n), then we also have a limiting distribution for (M - E(M)) / Stdev(M) after proper standardization. This is known as the Fisher–Tippett–Gnedenko theorem in extreme value theory. The limit distribution is not Gaussian. What would happen if instead of the maximum or weighted average, we consider the empirical percentiles? See also chapter 16.
- The digits for the vast majority of numbers, in all number representation systems, can be used to emulate Brownian motions, thanks to the central limit theorem. See appendix B in this book.

Another potential generalization consists of developing a central limit theorem that is based on L^1 rather than L^2 measures of centrality and dispersion, that is, the median and absolute deviations rather than the mean and variance. This would be useful when the observations come from a distribution that does not have a mean or variance, such as Cauchy.

Also, does the limit distribution in case 4 depend on the distribution of the X_k's -- in this case uniform -- or is it a universal distribution that is the same regardless of the X_k's distribution? Unfortunately, the answer is negative: after trying with the square of uniform deviates for the X_k's, the limit distribution was not symmetric, and thus different from the one obtained with uniform deviates.

4. Appendix: source code

Below is the source code (Perl) used to produce the simulations:

```
$seed=100;
$c=-0.5; # the exponent in a(k) = k^c
$n=10000;
open(OUT,">out.txt");
for ($m=0; $m<10000; $m++) {
  $den=0;
  $num=0;
  $ss=0;
  for ($k=1; $k<=$n; $k++) {
    $r=rand();
    $aux=exp($c*log($k)); # k^c
    $num+=$r*$aux;
    $den+=$aux;
    $ss+=($aux*$aux);
  }
  $dev=$num/$den;
  $std=sqrt(1/12) * (sqrt($ss)/$den); # 1/12 for Uni[0,1]
  $dev2=($dev-0.5)/$std;
  print OUT "$m\t$dev2\n";
}
close(OUT);
```

Also, Figure 2 below is referenced earlier in this chapter.

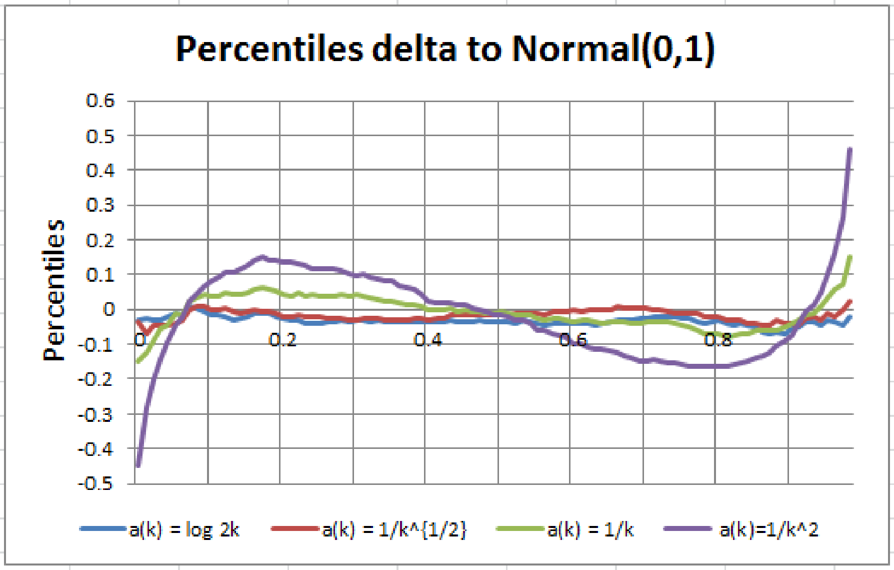


Figure 2: *Two weight sets (green and purple) produce non-Gaussian limit distributions*

9. More Tests of Randomness

We explore here some deterministic sequences of numbers, behaving like stochastic processes or chaotic systems, together with another interesting application of the central limit theorem.

In this chapter, you will learn some modern techniques to detect whether a sequence appears as random or not, whether it satisfies the central limit theorem (CLT) or not -- and what the limiting distribution is if CLT does not apply -- as well as some tricks to detect abnormalities. Detecting lack of randomness is also referred to as signal versus noise detection, or pattern recognition.

It leads to the exploration of time series with massive, large-scale (long term) auto-correlation structure, as well as model-free, data-driven statistical testing. No statistical knowledge is required: we will discuss deep results that can be expressed in simple English. Most of the testing involved here uses big data (more than a billion computations) and data science, to the point that we reached the accuracy limits of our machines. So there is even a tiny piece of numerical analysis in this article.

Potential applications include testing randomness, Monte Carlo simulations for statistical testing, encryption, blurring, and steganography (encoding secret messages into images) using pseudo-random numbers. A number of open questions are discussed here, offering professional statisticians new research topics both in theoretical statistics and advanced number theory. The level here is state-of-the-art, but we avoid jargon and some technicalities to allow beginners and non-statisticians to understand and enjoy most of the content. An Excel spreadsheet, attached to this document, summarizes my computations and will help you further understand the methodology used here.

Interestingly, I started to research this topic by trying to apply the notorious CLT (see previous chapter) to non-random (static) variables -- that is, to fixed sequences of numbers that look chaotic enough to simulate randomness. Ironically, it turned out to be far more complicated than using CLT for regular random variables. So I start here by describing what the initial CLT problem was, before moving into other directions such as testing randomness, and the distribution of the largest gap in seemingly random sequences. As we will see, these problems are connected.

1. Central Limit Theorem for Non-Random Variables

Here we are interested in sequences generated by a periodic function f(x) that has an irrational period T, that is f(x+T) = f(x). Examples include f(x) = sin x with $T = 2\pi$, or f(x) = $\{\alpha x\}$ where $\alpha > 0$ is an irrational number, { } represents the fractional part and $T = 1/\alpha$. The kth element in the infinite sequence (starting with $k = 1$) is f(k). The central limit theorem can be stated as follows:

Under certain conditions to be investigated -- mostly the fact that the sequence seems to represent or simulate numbers generated by a well-behaved stochastic process -- we would have:

$$U(n) = \frac{\sqrt{n}}{\sigma} \cdot \left(\frac{1}{n}\sum_{k=1}^{n} f(k) - \mu\right) \to N(0,1) \text{ with } \mu = \frac{1}{T}\int_0^T f(x)dx \text{ and } \sigma^2 = \frac{1}{T}\int_0^T (f(x)-\mu)^2 dx$$

In short, $U(n)$ tends to a normal distribution of mean 0 and variance 1 as n tends to infinity, which means that as both n and m tends to infinity, the values $U(n+1)$, $U(n+2)$ ... $U(n+m)$ have a distribution that converges to the standard bell curve.

In this chapter, we are dealing exclusively with sequences that are equidistributed over [0, 1], thus $\mu = 1/2$ and $\sigma = 1/12^{1/2}$. In particular, we investigate $f(x) = \{\alpha x\}$ where $\alpha > 0$ is an irrational number and { } the fractional part. While this function produces a sequence of numbers that seems fairly random, there are major differences with truly random numbers, to the point that CLT is no longer valid. The main difference is the fact that these numbers, while somewhat random and chaotic, are much more evenly spread than random numbers. True random numbers tend to create some clustering as well as empty spaces. Another difference is that these sequences produce highly auto-correlated numbers.

As a result, we propose a more general version of CLT, redefining $U(n)$ by adding two parameters a and b:

$$U(n) = \frac{n^a(\log n)^b}{\sigma} \cdot \left(\frac{1}{n}\sum_{k=1}^{n} f(k) - \mu\right)$$

This more general version of CLT can handle cases like our sequences. Note that the classic CLT corresponds to $a = 1/2$ and $b = 0$. In our case, we suspect that $a = 1$ and b is between 0 and -1. This is discussed in the next section.

Note that if instead of $f(k)$, the k^{th} element of the sequence is replaced by $f(k^2)$ then the numbers generated behave more like random numbers: they are less evenly distributed and less auto-correlated, and thus the CLT might apply. We haven't tested it yet.

2. Testing Randomness: Max Gap, Auto-Correlations and More

The sequence $f(1)$, $f(2)$, … generated by our function $f(x)$ is called an *α-sequence* or perfect process (see appendix B in this book.) Here we compare properties of α-sequences with those of random numbers on [0, 1] and we highlight the striking differences. Both sequences, when n tends to infinity, have a mean value converging to 1/2, a variance converging to 1/12 (just like any uniform distribution on [0, 1]), and they both look quite random at first glance. But the similarities almost stop here.

Maximum gap

The maximum gap among *n* points scattered between 0 and 1 is another way to test for randomness. If the points were truly randomly distributed, the expected value for the length of the maximum gap (also called longest segment) is known and is equal to

$$\frac{1}{n}\sum_{k=1}^{n}\frac{1}{k} \sim \frac{\log(n)}{n}$$

See this article for details, or the book Order Statistics published by Wiley, page 135. The max gap values have been computed in the spreadsheet (see section below to download the spreadsheet) both for random numbers and for α-sequences. It is pretty clear from the Excel spreadsheet computations (and confirmed in chapter 17) that the average maximum gaps have the following expected values, as *n* becomes very large:

- Maximum gap for random numbers: log(*n*)/*n* as expected from the above theoretical formula
- Maximum gap for α-sequences: *c*/*n* (*c* is a constant close to 1.5; the result needs to be formally proved)

So α-sequences have points that are far more evenly distributed than random numbers, by an order of magnitude, not just by a constant factor! This is true for the eight α-sequences (eight different values of α) investigated in the spreadsheet, corresponding to eight "nice" irrational numbers (more on this in the research section below, about what a "nice" irrational number might be in this context.)

Auto-correlations

Unlike random numbers, values of *f*(*k*) exhibit strong, large-scale auto-correlations: *f*(*k*) is strongly correlated with *f*(*k*+*p*) for some values of *p* as large as 100. The successive lag-*p* auto-correlations do not seem to decay with increasing values of *p.* To the contrary, it seems that the maximum lag-*p* auto-correlation (in absolute value) seems to be increasing with *p*, and possibly reaching very close to 1 eventually. This is in stark contrast with random numbers: random numbers do not show auto-correlations significantly different from zero, and this is confirmed in the spreadsheet. Also, the vast majority of time series have auto-correlations that quickly decay to 0. This surprising lack of decay could be the subject of some interesting number theoretic research. These auto-correlations are computed and illustrated in the Excel spreadsheet (see section below) and are worth checking out. Exact values are computed in chapter 13.

Convergence of *U*(*n*) to a non-degenerate distribution

Figures 2 and 3 in the next section (extracts from our spreadsheet) illustrate why the classic central limit theorem (that is, *a* = 1/2, *b* =0 for the *U*(*n*) formula) does not apply to α-sequences, and why *a* = 1 and *b = 0* might be the correct parameters to use instead. However, with the data gathered so far, we can't tell whether *a* = 1 and *b* = 0 is correct, or whether *a* = 1 and *b* = -1 is correct: both exhibit similar asymptotic behavior, and the data collected is not accurate enough to make a final decision on this. The answer could come from theoretical considerations rather than from big data analysis. Note that the correct parameters should produce a somewhat horizontal band for *U*(*n*) in figure 2, with values mostly concentrated between -2 and +2 due to normalization of *U*(*n*) by design. And *a* = 1, *b* = 0, as well as *a* = 1, *b* = -1, both do just that, while it is clear that *a* = 1/2 and *b* = 0 (classic CTL) fails as illustrated in figure 3. You can play with parameters *a* and *b* in the spreadsheet, and see how it changes figure 2 or 3, interactively.

One issue is that we computed *U*(*n*) for *n* up to 100,000,000 using a formula that is ill-conditioned: multiplying a large quantity *n* by a value close to zero (for large *n*) to compute *U*(*n*), when the precision available is probably less than 12 digits. This might explain the large, unexpected oscillations found in Figure 2. Note that oscillations are expected (after all, *U*(*n*) is supposed to converge to a statistical distribution, possibly the bell curve, even though we are dealing with non-random sequences) but such large-scale, smooth oscillations, are suspicious.

3. Excel Spreadsheet with Computations

Click here to download the spreadsheet. The spreadsheet has 3 tabs: One for α-sequences, one for random numbers -- each providing auto-correlation, max gap, and some computations related to estimating *a* and *b* for *U*(*n*) -- and a tab summarizing *n* = 100,000,000 values of *U*(*n*) for α-sequences, as shown in figures 2 and 3. That tab, based on data computed using a Perl script, also features moving maxima and moving minima, a concept similar to moving averages, to better identify the correct parameters *a* and *b* to use in *U*(*n*).

Confidence intervals (CI) can be empirically derived to test a number of assumptions, as illustrated in Figure 1: in this example, based on eight measurements, it is clear that maximum gap CI's for α-sequences are very different from those for random numbers, meaning that α-sequences do not behave like random numbers.

1.49	1.72	1.50	1.10	1.64	1.73	1.40	1.94
9.78	14.89	12.51	9.28	9.60	9.25	9.33	9.10

Figure 1*: max gap times n (n = 10,000), for eight α-sequences (top) and eight sequences of random numbers (bottom)*

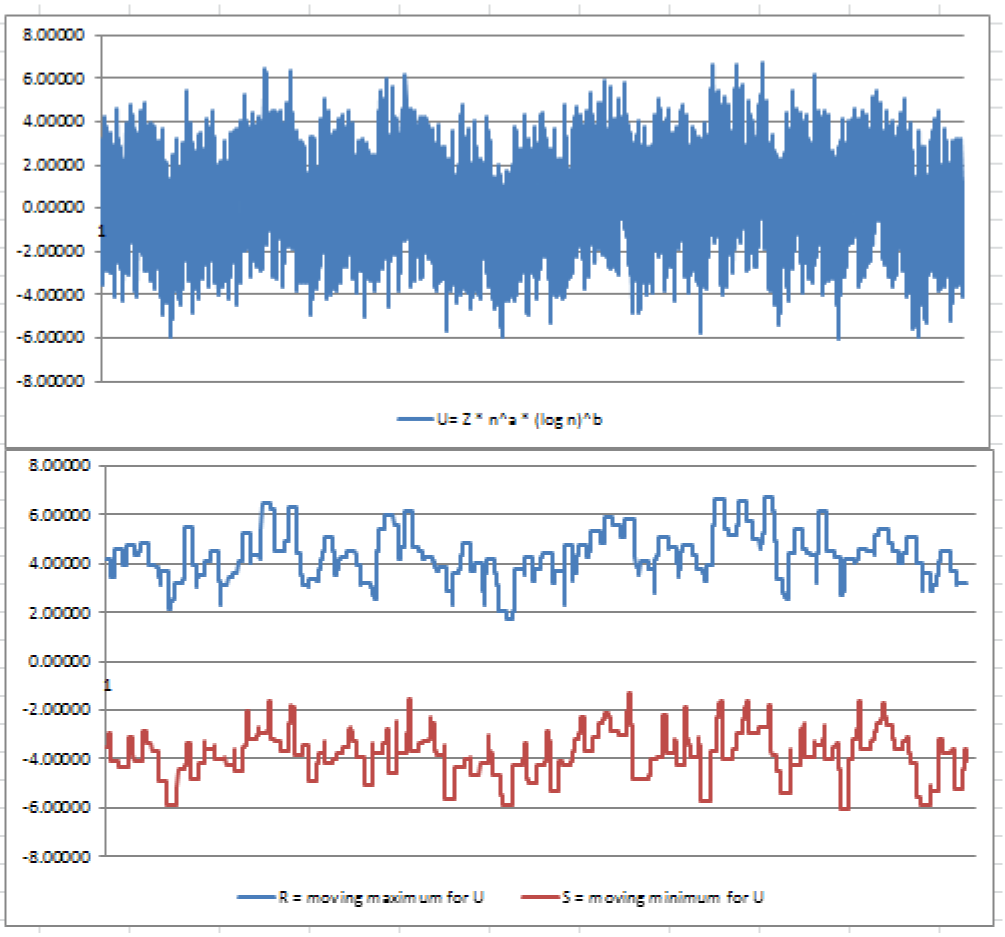


Figure 2: *U(n) with a = 1, b = 0 (top) and moving max / min bands (bottom) for α-sequences*

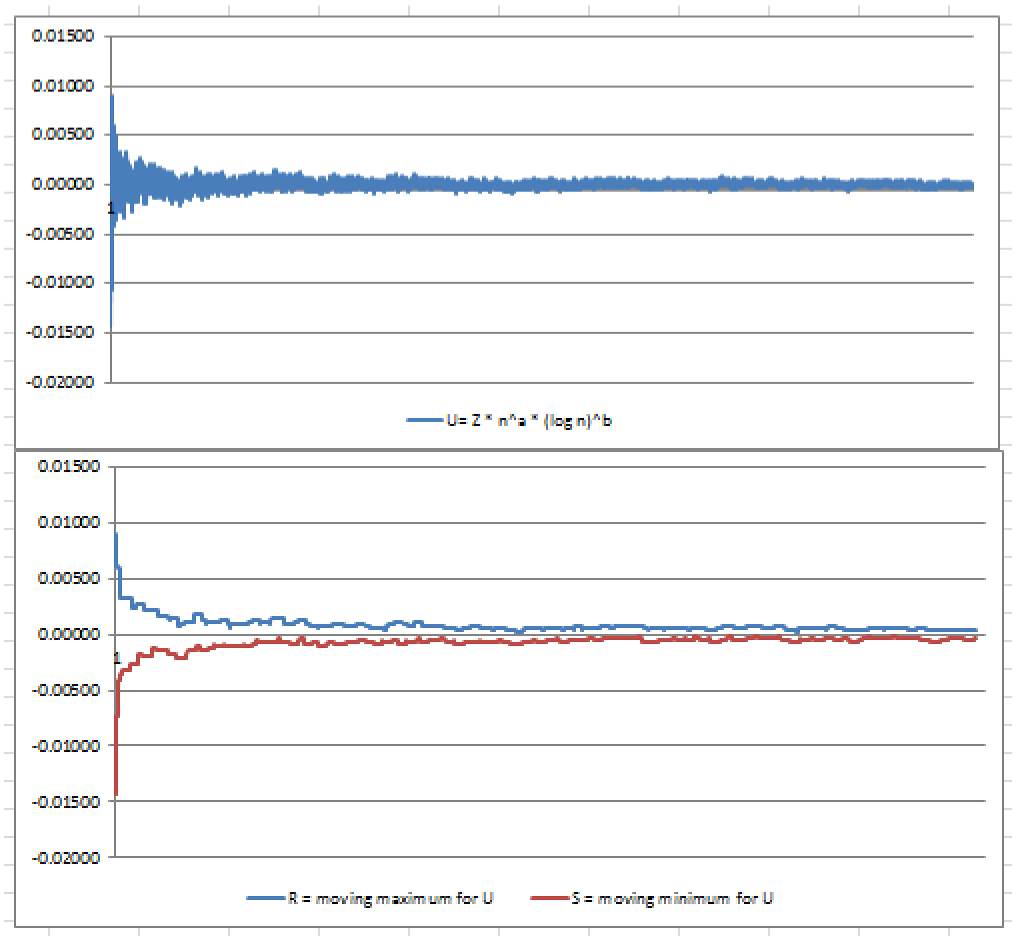


Figure 3: *U(n) with a = 0.5, b = 0 (top) and moving max / min bands (bottom) for α–sequences*

4. Potential Research Areas

Here we mention some interesting areas for future research. By sequence, we mean α-sequence as defined in section 2, unless otherwise specified.

- Using $f(k^c)$ as the k^{th} element of the sequence, instead of $f(k)$. Which values of $c > 0$ lead to equidistribution over [0, 1], as well as yielding the classic version of CLT with $a = 1/2$ and $b = 0$ for $U(n)$? Also what happens if $f(k) = \{\alpha p(k)\}$ where $p(k)$ is the k^{th} prime number and { } represents the fractional part? This sequence was proved to be equidistributed on [0, 1] (this by itself.is a famous result of analytic number theory, published by Vinogradov in 1948) and has a behavior much more similar to random numbers, so maybe the classic CLT applies to this sequence? Nobody knows.

- What is the asymptotic distribution of the moments and distribution of the maximum gap among the n first terms of the sequence, both for random numbers on [0, 1] and for the sequences investigated in this article? Does it depend on the parameter α? Same question for minimum gap and other metrics used to test randomness, such as point concentration, defined for instance in the article On Uniformly Distributed Dilates of Finite Integer Sequences?

- Does $U(n)$ depend on α? What are the best choices for α, to get as much randomness as possible? In a similar context, $2^{1/2}$ - 1 and ($5^{1/2}$ - 1)/2 are found to be good candidates: see this Wikipedia article (read the section on additive recurrence.) Also, what are the values of the coefficients a and b in $U(n)$, for α-sequences? It seems that a must be equal to 1 to guarantee convergence to a non-degenerate distribution. Is the limiting distribution for $U(n)$ also normal for α-sequences, when using the correct a and b?

- What happens if α is very close to a simple rational number, for instance if the first 500 digits of α are identical to those of 3/2?

Generalization to higher dimensions

So far we worked in dimension 1, the support domain being the interval [0, 1]. In dimension 2, $f(x) = \{\alpha x\}$ becomes $f(x, y) = (\{\alpha x\}, \{\beta y\})$ with α, β, and α/β irrational; $f(k)$ becomes $f(k,k)$. Just like the interval [0, 1] can be replaced by a circle to avoid boundary effects when deriving theoretical results, the square [0, 1] x [0, 1] can be replaced by the surface of the torus. The maximum gap becomes the maximum circle (on the torus) with no point inside it. The range statistic (maximum minus minimum) becomes the area of the convex hull of the n points. For a famous result regarding the asymptotic behavior of the area of the convex hull of a set of n points, see previous chapter and check out the sub-section entitled “Other interesting stuff related to the Central Limit Theorem.” Note that as the dimension increases, boundary effects become more important.

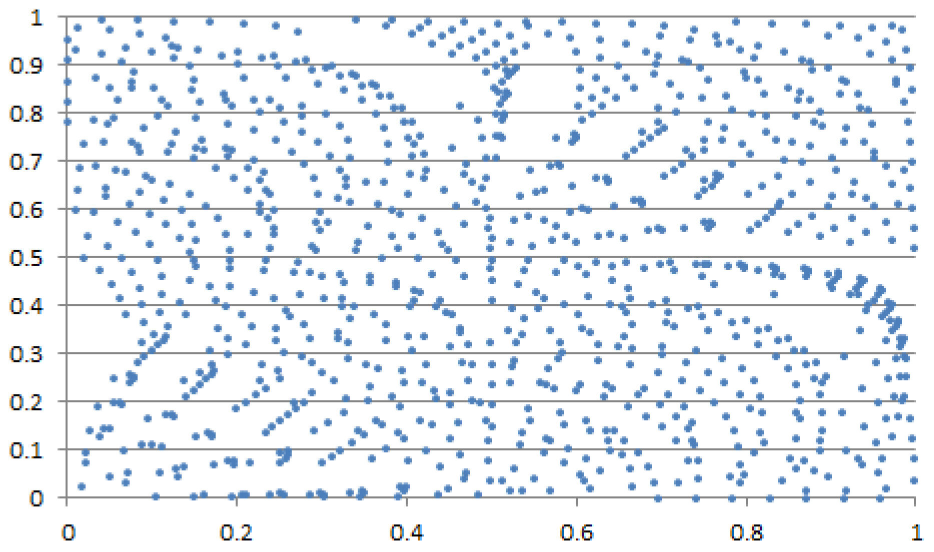


Figure 4: *bi-variate example with c = 1/2, $\alpha = 31^{1/2}$, $\beta = 17^{1/2}$ and n = 1000 points*

Figure 4 shows an unusual example in two dimensions, with strong departure from randomness, at least when looking at the first 1,000 points. Usually, the point pattern looks much more random, albeit not perfectly random, as in Figure 5.

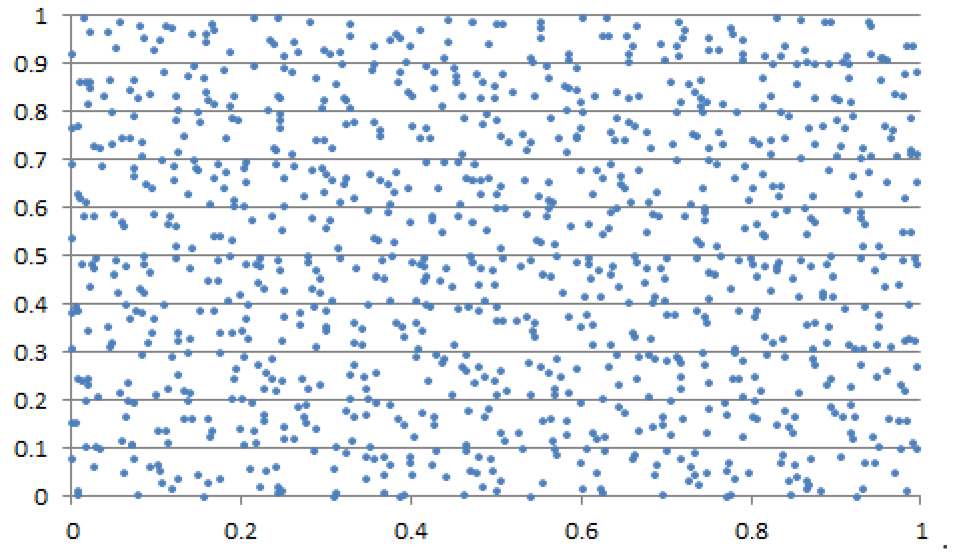


Figure 5: *bi-variate example with c = 1/2, $\alpha = 13^{1/2}$, $\beta = 26^{1/2}$ and n = 1000 points*

Computations are found in this spreadsheet. Note that we've mostly discussed the case *c* = 1 in this chapter. The case *c* = 1/2 creates interesting patterns, and the case *c* = 2 produces more random patterns. The case *c* = 1 creates very regular patterns (points evenly spread, just like in one dimension.)

10. Random Weighted Sums

You won't learn this in textbooks, college classes, or data camps. Some of the material in this chapter is very advanced yet presented in simple English, with an Excel implementation for various statistical tests, and no arcane theory, jargon, or obscure theorems. It has a number of applications, in finance in particular. This chapter covers several topics under a unified approach, so it was not easy to find a title. In particular, we discuss:

- When the central limit theorem fails: what to do, and case study
- Various original statistical tests, some unpublished, for instance to test if an empirical statistical distribution (based on observations) is symmetric or not, or whether two distributions are identical
- The power and mysteries of stable (also called divisible) statistical distributions
- Dealing with weighted sums of random variables, especially with decaying weights
- Fun number theory problems and algorithms associated with these statistical problems
- Decomposing a (theoretical or empirical / observed) statistical distribution into elementary components, just like decomposing a complex molecule into atoms

The focus is on principles, methodology, and techniques applicable to, and useful in many applications. For those willing to do a deeper dive on these topics, many references are provided. This chapter, written as a tutorial, is accessible to professionals with elementary statistical knowledge, like stats 101. It is also written in a compact style, so that you can grasp all the material in hours rather than days. This simple chapter covers topics that you could learn in MIT, Stanford, Berkeley, Princeton or Harvard classes aimed at PhD students. Some is state-of-the-art research results published here for the first time, and made accessible to the data science of data engineer novice. I think mathematicians (being one myself) will also enjoy it. Yet, emphasis is on applications rather than theory.

Finally, we focus here on sums of random variables. The next chapter will focus on mixtures rather than sums, providing more flexibility for modeling purposes, or to decompose a complex distribution in elementary components. In both cases, my approach is mostly non-parametric, and based on robust statistical techniques, capable of handling outliers without problems, and not subject to over-fitting.

Finally, many statistical tests are introduced in this chapter, in addition to those mentioned in the previous chapters.

1. Central Limit Theorem: New Approach

Let us consider a weighted sum of *n* independent and identically distributed random variables, with finite variance and mean equal to zero:

$$Z_n = \frac{\sum_{k=1}^n a_k X_k}{\sqrt{\sum_{k=1}^n a_k^2}}.$$

When the weights are identical, the central limit theorem (CLT) states that this converges to a Gaussian distribution. Regardless of the weights or *n*, we have:

$$\mathrm{Var}(Z_n) = \mathrm{Var}(X_1).$$

This is also true when *n* tends to infinity. Without loss of generality, we assume here that all these random variables have their expectation equal to 0. Now, let us decompose the weighted sum into two components. First, let us introduce a partition of *N* = {1, ..., n} into two subsets *I* and *J*. For instance, the set *I* consists of the odd integers in *N*, and *J* consists of the even integers in *N*. Or *I* consists of the integers smaller than *n*/2, and *J* consists of the integers larger or equal to *n*/2. The decomposition is as follows:

$$Z_n = \frac{v_n}{s_n} \cdot V_n + \frac{w_n}{s_n} \cdot W_n,$$

$$V_n = v_n^{-1} \cdot \sum_{k \in I} a_k X_k,\ W_n = w_n^{-1} \cdot \sum_{k \in J} a_k X_k,\ s_n = \sqrt{\sum_{k \in N} a_k^2},\ v_n = \sqrt{\sum_{k \in I} a_k^2},\ w_n = \sqrt{\sum_{k \in J} a_k^2}.$$

The variables Z_n, V_n and W_n have the same variance as X_1. Let us define $p_n = v_n / s_n$, and $q_n = w_n / s_n$. At the limit as *n* tends to infinity, assuming the limits exist, we have

$$Z = pV + qW,$$

$$Z = \lim_{n\to\infty} Z_n,\ V = \lim_{n\to\infty} V_n,\ W = \lim_{n\to\infty} W_n,\ p = \lim_{n\to\infty} p_n,\ q = \lim_{n\to\infty} q_n,$$

$$p^2 + q^2 = p_n^2 + q_n^2 = 1.$$

When the weights a_k's are identical (corresponding to the standard CLT) then the factors *p* and *q* can be made arbitrarily close to any real number (by choosing *I* and *J* appropriately) and thus the limit distribution satisfies the following property, presented here as a theorem:

Theorem

If the weights a_k are identical (classic CLT framework) then Z can be written as any linear combination pV + qW of tho independent random variables V and W that have the same exact distribution as Z (with same variance), provided that $p^2 + q^2 = 1$.

Note that by convergence and limit, here we mean convergence in distribution. A distribution that satisfies the property stated in the above theorem is called a **stable distribution**. Evidently the Gaussian distribution is one example of a stable distribution. But is it the only one? That is, does this property uniquely characterize the Gaussian distribution?

Let us introduce the concept of semi-stable distribution. A random variable Z has a **semi-stable distribution** if for any strictly positive integer n, it can be written as the sum of n independent random variables, divided by $n^{1/2}$, with each random variable having the same distribution as Z. Gaussian distributions are one example. All stable distributions are also semi-stable (see exercise below), but the converse might not be true.

Let's say that the variables X_k have a semi-stable distribution, but one that is non-Gaussian. For each value of n including at the limit as n tends to infinity, Z_n would have (by construction) that exact same semi-stable distribution, which is non-Gaussian. But the central limit theorem states that at the limit, the distribution must be Gaussian. This contradiction makes you think that the only semi-stable distribution is the Gaussian one. I won't spend much time on this paradox, but I invite you to think about it. There are other stable and semi-stable distributions, as we shall see in the next section. Generally speaking, they have a thick tail and infinite variance. The Gaussian one is the only one with a finite variance.

Exercise

Prove that any stable distribution is also semi-stable. Hint: For a stable distribution, Z can be written as $Z = (2/3)^{1/2}\{ (V + W) / 2^{1/2} \} + (1/3)^{1/2}U = (V + W + U) / 3^{1/2}$, with U, V, W, $(V + W) / 2^{1/2}$ having the same distribution as Z. This easily generalizes to 4, 5 or more variables, and can be proved by induction. Here, U, V and W are independent.

2. Stable and Attractor Distributions

The initial problem I was interested in, is to approximate a random variable Z with a complicated distribution, by a weighted sum of independent, identically distributed random variables that have a simple distribution. These random variables, denoted as X_1, X_2 and so on throughout this article, are also called *kernels*.

We have seen that if these kernels have a stable distribution, then Z must also have a stable (and identical) distribution. So, unless we want to restrict ourselves to the small family of stable distributions, we must consider unstable kernels. From now on, we will mostly focus on (unstable) kernels that have a uniform distribution on [-1/2, 1/2].

The next question is: can any Z (regardless of its distribution) be represented (that is, decomposed) in this manner? It is similar to asking whether any real function can be represented by a Taylor series. The answer, in both cases, is no. The kind of random variables Z that can be represented in this manner are called *attractors*. Their distribution is called an **attractor distribution**. We are curious to find out

- Which distributions can be attractors,
- Whether Z's distribution depends on the choice of the kernels, and
- Whether an attractor distribution must necessarily be a stable distribution, even if the kernel is not.

So clearly, we are here in a context where the assumptions of the CLT (central limit theorem) are violated, and the CLT does not apply. Let us introduce one additional notation:

$$b_{nk} = \frac{a_k}{\sqrt{\sum_{k=1}^{n} a_k^2}}.$$

With this notation, Z_n can be re-written as

$$Z_n = \sum_{k=1}^{n} b_{nk} \cdot X_k.$$

Depending on the weights a_k, the coefficient b_{nk} for any fixed k, may not depend on n, as n tends to infinity.

Using decaying weights

We shall consider, moving forward, decaying weights of the form $a_k = 1/k^c$, where c is a positive parameter in the interval [0.5, 1]. If c is less than or equal to 0.5, then we are back under the CLT conditions and Z has a Gaussian distribution, even if the kernel has a uniform distribution. If c is larger than 0.5, then we have the following:

- The distribution of Z is NOT a Gaussian one (see chapter 8 for details.)
- For any fixed k, b_{nk} does not depend on n, as n tends to infinity. For instance, if c = 1, then b_{nk} tends to $6^{1/2}a_k / \pi$ as n tends to infinity.

Decaying weights have plenty of applications. For instance, I used a decaying weighted sum of nearest neighbor distances in a k-NN clustering problems, rather than the k^{th} nearest neighbor alone, to boost robustness. The resulting distribution of Z was rather special; the details are available in this article.

Exact distribution of *Z*

This topic is more advanced for statisticians, as it is based on the characteristic function of a statistical distribution. For physicists, mathematicians, signal processing experts, and engineers, it is not very advanced, in the sense that it is a simple application of the convolution theorem and Fourier transforms. I assume here that c = 1.

The characteristic function of Z is a product of characteristic functions of uniform distributions (the kernels):

$$\phi_Z(t) = \prod_{k=1}^{\infty} \frac{2k\pi}{\sqrt{6}t} \cdot \sin\left(\frac{\sqrt{6}t}{2k\pi}\right).$$

The infinite product is converging. But this is not the characteristic function of a semi-stable distribution, unlike with *c* = 1/2 or *c* = 0. To find the density attached to *Z*, one has to take the inverse Fourier transform of the characteristic function:

$$f_Z(z) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-itz} \cdot \phi_Z(t)dt.$$

The characteristic function can be re-written as *G*(*t*)*H*(*t*), with

$$H(t) = \frac{2}{t}\sin\frac{t}{2}, \text{ and } G(t) = \frac{1}{H(t)}\lim_{n\to\infty}\prod_{k=1}^{n}\frac{2k\pi}{\sqrt{6}t}\sin\frac{\sqrt{6}t}{2k\pi}.$$

Written that way, *Z* appears as the sum of two independent random variables with characteristic functions respectively equal to *H*(*t*) and *G*(*t*). *H*(*t*) corresponds to the kernel distribution: uniform on [-1/2, 1/2]. As for *G*(*t*), you get a good approximation if you only use the first 20 factors (*n* = 20) in the above infinite product. Indeed, *G*(*t*) is not very much different from a constant function equal to 1, corresponding to a Dirac distribution when you take the inverse Fourier transform, especially when *t* is close to zero.

Note that we have:

$$H(t) = \prod_{k=1}^{\infty}\left\{1 - \frac{t^2}{4k^2\pi^2}\right\}, \text{ and } \frac{2k\pi}{\sqrt{6}t}\sin\frac{\sqrt{6}t}{2k\pi} \approx 1 - \frac{t^2}{4k^2\pi^2} \text{ as } k\to\infty.$$

See here for the above infinite product formula for the sine function. The asymptotic approximation on the right-hand side is based on the Taylor series for the sine function. This is what makes this representation particularly interesting, and beautiful. However, since *Z* and the kernel have the same variance, and since the sum of two independent random variables has a variance greater than or equal to that of each summand, the distribution attached to *G* (if it exists) must be improper. In other words, *G* may not be a characteristic function.

More about stable distributions and their applications

In some sense, stable distributions are invariant under linear transformations, while semi-stable distributions are invariant under addition. The Wikipedia entry for stable distributions is worth reading, especially the section related to the CLT. Examples of stable distributions, besides the Gaussian one, include the Levy and Cauchy distributions; both of them have infinite variance (sometimes called *heavy tail*.) Stable distributions are also related to the Levy process, with applications to financial markets.

The following references provide additional insights about the topics discussed here:

- Stable Distributions Models for Heavy Tailed Data (book published in 2018; focuses on multivariate distributions, with application to financial modeling, $370 on Amazon)
- Limit Distributions for Sums of Independent Random Variables (seminal book published in 1954; costs $686 on Amazon, for a used book!)
- Heavy Tails in Theory and Practice (book published in 2001; $200 on Amazon)
- Random Summation: Limit Theorems and Applications (book published in 1996; costs $245 on CRC Press; focuses on sums where the number of summands is itself random; costs $41 on Amazon)
- Indecomposable distributions (Wikipedia entry)

Stable distributions were once considered just a mathematical curiosity. Around 1960 researchers began to discover evidence of heavy tail fluctuations in financial data. This line of research led to the discovery of fractals. By now, stable models are firmly established in the area of finance. Stable distributions are also used in electrical engineering and hydrology. Applications in other areas of science are emerging rapidly, and the subject continues to gain momentum.

3. Non CLT-compliant Weighted Sums, and their Attractors

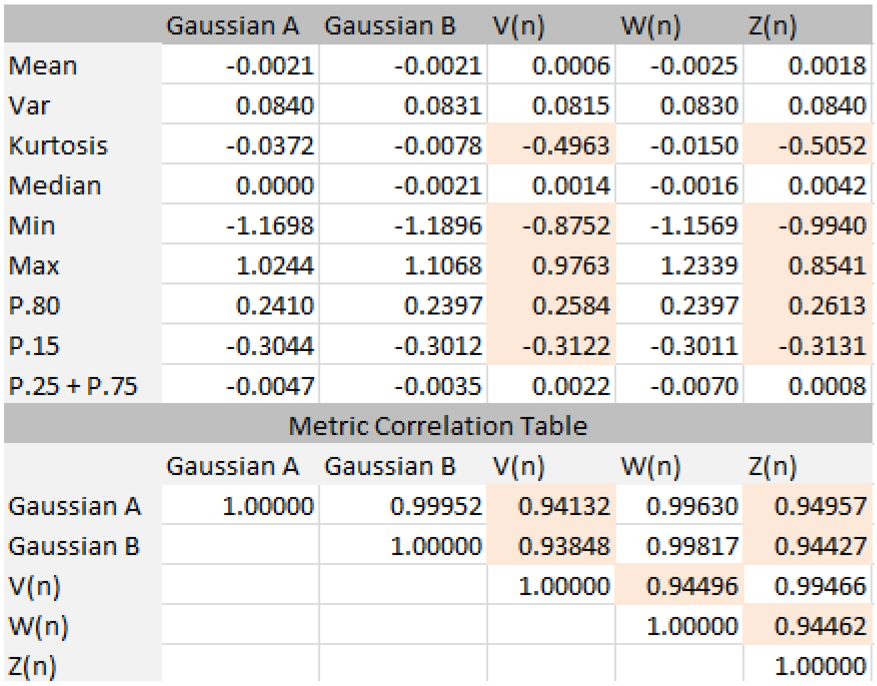
In this section, we investigate what happens when using decaying weights $a_k = 1/k^c$, with $c = 1$, and with uniform kernels on [-0.5, 0.5], as discussed in the previous section. The busy reader can jump to the conclusion at the bottom. For the statistician, data scientist, or machine learning architect, I present simple, interesting statistical tests that were used during my analysis, to obtain my results and conclusions.

Testing for normality

In our main test, we divided $N = \{1, ..., n\}$ into two subsets I and J as follows: I contains the integers less than 20; and J the integers greater or equal to 20. We also tested other partitions of N, see tests for semi-stability, below. Here, n was set to 100. We sampled from 5 different distributions, generating m = 20,000 deviates for each one:

- A Gaussian distribution for comparison purpose (Gaussian A)
- Another Gaussian distribution (Gaussian B) to assess variations across two samples from a same distribution
- V_n, W_n, and Z_n

The testing was done in Excel. By construction, all these distributions have a theoretical mean and median equal to zero, and a variance equal to 1/12 (that's the variance of the kernel), as evidenced by the estimates in the table below. The number $P_{.80}$ represents the 80^{th} percentile. The number $P_{.25} + P_{.75}$ is zero if the distribution is symmetric. This is the case in this example.



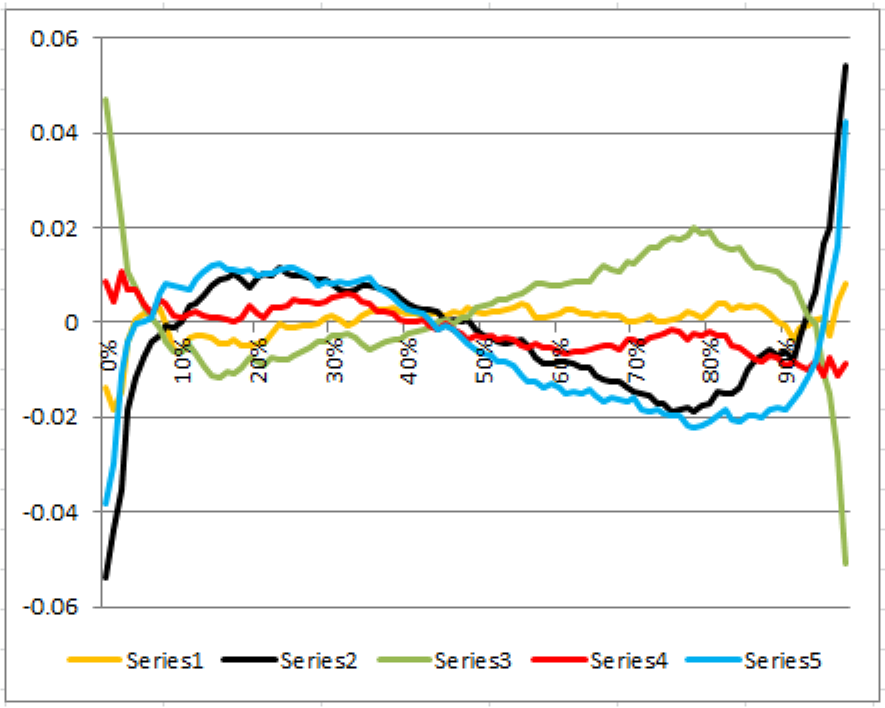
	Gaussian A	Gaussian B	V(n)	W(n)	Z(n)
Mean	-0.0021	-0.0021	0.0006	-0.0025	0.0018
Var	0.0840	0.0831	0.0815	0.0830	0.0840
Kurtosis	-0.0372	-0.0078	-0.4963	-0.0150	-0.5052
Median	0.0000	-0.0021	0.0014	-0.0016	0.0042
Min	-1.1698	-1.1896	-0.8752	-1.1569	-0.9940
Max	1.0244	1.1068	0.9763	1.2339	0.8541
P.80	0.2410	0.2397	0.2584	0.2397	0.2613
P.15	-0.3044	-0.3012	-0.3122	-0.3011	-0.3131
P.25 + P.75	-0.0047	-0.0035	0.0022	-0.0070	0.0008

Metric Correlation Table

	Gaussian A	Gaussian B	V(n)	W(n)	Z(n)
Gaussian A	1.00000	0.99952	0.94132	0.99630	0.94957
Gaussian B		1.00000	0.93848	0.99817	0.94427
V(n)			1.00000	0.94496	0.99466
W(n)				1.00000	0.94462
Z(n)					1.00000

The sample size (m = 20,000) gives about two correct decimals in the table. The rule of thumb is that the precision is equal to about $1/m^{1/2}$ except for Max and Min, which are always highly volatile. Clearly, the two Gaussian A and B look identical as expected, the distribution of V_n and Z_n look identical too though clearly non Gaussian, and the distribution of W_n looks Gaussian. In fact it is not Gaussian, but close: the sample size is too small to pinpoint the difference. Likewise, V_n and W_n do not have the same exact distribution (almost) but the sample size is too small to pinpoint the difference. This framework actually serves as a great benchmark to assess the power of the statistical tests involved.

We also performed visual tests to measure the difference between pairs of percentile distributions, see figure below.



The yellow curve shows the empirical (observed) percentiles delta between the two Gaussians A and B. The differences are negligible. The red curve shows the percentile delta between the (almost identical) distributions of V_n and W_n. To the contrary, the three other curves are comparing distributions that are significantly different. For instance, the black curve represents the difference between Gaussian A and the distribution of V_n. These percentile tests are similar to a Kolmogorov-Smirnov test, except that Kolmogorov-Smirnov is based on the empirical cumulative distribution (CDF) while ours is based on the empirical percentile function, which is the inverse of the CDF. They clearly show when distributions are statistically different.

All the Excel computations are available in this spreadsheet.

Testing for symmetry and dependence on kernel

One can compare R(x) = | 2 Median - $P_{.x}$ - $P_{.1-x}$ | with that of a symmetric distribution, for various values of x between 0 and 0.5, to check if a distribution is symmetric around the median. The theoretical value of R(x) is zero regardless of x, if your empirical distribution is symmetric. Here P.x represents the x^{th} percentile. Other tests for symmetry can be found here.

In this test, we used a highly non-symmetric kernel: a mixture of Bernoulli and uniform distributions. We discovered that while the limit (attractor) distribution is much less asymmetric than the kernel, it is still clearly non symmetric. This also means that the distribution of Z depends on the kernel: when the kernel has a symmetric distribution, Z also has a symmetric distribution.

Testing for uni-modality and other peculiarities

All standard attractors investigated in the literature have a unimodal distribution. We haven't tested if Z is unimodal, but we believe it is, regardless of the kernel. To test if a distribution is unimodal, several tests have been devised: the bandwidth test, the dip test, the excess mass test, the MAP test, the mode existence test, the run test, the span test, and the saddle test. The dip test is available in R. Read more here. Some of these tests, in case of multimodality, can tell you how many modes (or clusters) are in your data sets.

Other potential tests, not discussed here, could be used to check if Z has an an unbounded support domain, or to check if its density is bounded. The answer is believed to be positive in both cases. An example of unbounded density is $f(x) = 0.25 / |x|^{1/2}$ with x in [-1, 1]. We could also test for infinite mean or infinite variance (a feature all stable distributions have, except the Gaussian) however it is irrelevant here since by construction the mean is zero, and the variance is equal to the variance of the kernel.

Testing for semi-stability

As stated in the first section, if the limiting distribution is semi-stable, it must satisfy $Z = pV + qW$, with $p = q = 1/2^{1/2}$. I tried various combinations for the subsets of indices I and J, but could not get close to $p= q$. Indeed, the closest you can get is with $I = \{1\}$ and $J = \{2, 3, ..., n\}$. When n tends to infinity, this leads to $p = 0.78$ and $q = 0.63$ (approximately); the exact values are $p = 6^{1/2} / \pi$ and $q = (\pi^2 - 6)^{1/2} / \pi$.

If you try $a_k = 1/k^c$, with $c = 0.6$ rather than $c = 1$, you might be able to get $p = q$ with a judicious choice of I and J. Likewise, if you try with $a_k = 1/(k + 5)$ rather than $a_k = 1/k$, it might work. However the resulting Z, V and W still have different distributions, so it fails to prove that Z is semi-stable. It does not disprove it either.

So how do you choose I and J to get $p = q$, assuming this equality is reachable in the first place? Whether or not it is feasible depends on how fast the weights decay. This is actually a number theory problem. For instance, if $n = 18$, $a_k = 1/(k+5)$, $I = \{1, 4, 6, 8, 10, 12, 14, 15, 18\}$ and $J = \{2, 3, 5, 7, 9, 11, 13, 16, 17\}$, then we get very close to $p = q$. This configuration was obtained using a greedy algorithm, as described fox example in this article.

Out of curiosity, we generated two independent samples of Z, say Z' and Z''. We checked whether Z and $(Z' + Z'') / 2^{1/2}$ had the same distribution, using the tests described in the above sub-section. It turns out that the two distributions are clearly different (they have the same mean and variance, but not the same kurtosis), thus Z is not semi-stable. In fact,$(Z' + Z'') / 2^{1/2}$ looks surprisingly close to a Gaussian distribution, while Z does not. This result is in contrast with what other authors wrote on the subject, 70 years ago, stating that any attractor must be semi-stable. The explanation is that these authors used different assumptions than ours, when analyzing converging weighted sums. Finally, note that $(Z' + Z'') / 2^{1/2}$ cannot be Gaussian: if it was, both *Z'* and *Z''* would have to be Gaussian too according to Cramer's theorem, and this is clearly not the case.

4. Conclusions

The framework discussed here produces a Gaussian distribution as the limit distribution for the weighted sum, if the weights are decaying slowly. This is just the standard CLT. If the weights are decaying a bit too fast -- faster than $1/k^{1/2}$ but not faster than $1/k$ -- the following issues and benefits arise:

- The limiting distribution (attached to *Z*) is not Gaussian: good, that is what we were looking for.
- The limiting distribution (also called *attractor*) may not be stable either, offering more flexibility. It might not even be symmetrical, depending on the kernel. We haven't checked if the attractor must be unimodal (even if the kernel is not) however in all our tests, it was unimodal.
- Few distributions can be an attractor, so the amount of flexibility offered with fast-decaying weights is still limited. In fact, attractors don't look very much different from Gaussian distributions, though they are clearly not Gaussian; this offers limited possibilities for modeling.
- The limiting distribution depends both on the choice for the kernel, and the first few terms in the weighted sum, unlike with the classic CLT; it is a combination of a normal distribution, with a non-normal distribution. The non-normal part is attached to the first few terms of the weighted sum. This is indeed not bad, as it allows you to decompose these sums into two parts: the sum of the first 10 terms or so (non-normal) and the remaining of the sum (almost normal.) This is helpful for modeling purposes, allowing you to separate background noise (Gaussian-like tail) from the true signal.

A better tool to decompose a potential attractor into an infinite collection of basic kernels, is a mixture model, rather than a weighted sum. In that case, any distribution can be an attractor. This is discussed in the next chapter.

11. Mixture Models

In this chapter, emphasis is on *science*, not just on data. State-of-the art material is presented in simple English, from multiple perspectives: applications, theoretical research asking more questions than it answers, scientific computing, machine learning, and algorithms. I attempt here to lay the foundations of a new statistical technology, hoping that it will plant the seeds for further research on a topic with a broad range of potential applications. It is based on mixture models. Mixtures have been studied and used in applications for a long time, and it is still a subject of active research. Yet you will find here plenty of new material.

Content

1. Introduction and Context

In the previous chapter, I attempted to approximate a random variable representing real data, by a weighted sum of simple *kernels* such as uniformly and independently, identically distributed random variables. The purpose was to build Taylor-like series approximations to more complex models (each term in the series being a random variable), to

- Avoid over-fitting,
- Approximate any empirical distribution (the inverse of the percentiles function) attached to real data,
- Easily compute data-driven confidence intervals regardless of the underlying distribution,
- Derive simple tests of hypothesis,
- Perform model reduction,
- Optimize data binning to facilitate feature selection, and to improve visualizations of histograms
- Create perfect histograms,
- Build simple density estimators,
- Perform interpolations, extrapolations, or predictive analytics,
- Perform clustering and detect the number of clusters,
- Create deep learning Bayesian systems.

Why I've found very interesting properties about stable distributions during this research project, I could not come up with a solution to solve all these problems. The fact is that these weighed sums would usually converge (in distribution) to a normal distribution if the weights did not decay too fast -- a consequence of the central limit theorem. And even if using uniform kernels (as opposed to Gaussian ones) with fast-decaying weights, it would converge to an almost symmetrical, Gaussian-like distribution. In short, very few real-life data sets could be approximated by this type of model.

I also tried with independently but NOT identically distributed kernels, and again, failed to make any progress. By “not identically distributed kernels”, I mean basic random variables from a same family, say with a uniform or Gaussian distribution, but with parameters (mean and variance) that are different for each term in the weighted sum. The reason being that sums of Gaussian's, even with different parameters, are still Gaussian, and sums of Uniform's end up being Gaussian too unless the weights decay fast enough. Details about why this is happening are provided in the last section.

Now, in this chapter, starting in the next section, I offer a full solution, using mixtures rather than sums. The possibilities are endless.

2. Approximations Using Mixture Models

The problem is specified as follows. You have an univariate random variable *Y* that represents any of your quantitative features in your data set, and you want to approximate or decompose it using a mixture of *n* elementary independent random variables called *kernels* and denoted as *X*(*n*, *k*) for *k* = 1, ..., *n*, with decreasing probability weights *p*(*n*, *k*) that converge to zero. The approximation of *Y* based on the first *n* kernels, is denoted as *Y*(*n*). By approximation, I mean that the data-generated empirical distribution of *Y* is well approximated by the known, theoretical distribution of *Y*(*n*) and that as *n* tends to infinity, both become identical (hopefully).

Moving forward, *N* denotes your sample size, that is the number of observations; *N* can be very large, even infinite, but you want to keep *n* as small as possible.

Generalizations to the multivariate case is possible but not covered in this article. The theoretical version of this consists in approximating any known statistical distribution (not just empirical distributions derived from data sets) by a small mixture of elementary (also called atomic) kernels.

In statistical notation, we have:

$$P(Y(n) < y) = \sum_{k=1}^{n} p(n,k) \cdot P(X(n,k) < y) \text{ with}$$

$$0 \leq p(n,k) \leq 1,\ p(n,k) \geq p(n,k+1),\ \sum_{k=1}^{n} p(n,k) = 1.$$

We also want *Y*(*n*) to converge to *Y*, in distribution, as *n* tends to infinity. This implies that for large *n*, the weights *p*(*n*, *k*) must tend to zero as *k* tends to infinity.

The error term

There are various ways to define the distance between two distributions, say between *Y*(*n*) and *Y*. See here for details; one of the most popular ones is the Kolmogorov-Smirnov metric. Or you can also use the distance between the inverse of the cumulative distributions, see chapter 14 for details (read the section on testing for normality, in particular the *percentile test*.) Regardless of the metric used, the error term is denoted as *E*(*n*) = ||*Y* - *Y*(*n*)||. Of course, the problem, for a given value of *n*, is to minimize *E*(*n*). As *n* tends to infinity, by carefully choosing the parameters in the model (that is, the weights, as well the the means and variances of the kernels,) the error *E*(*n*) is supposed to converge to 0. Note that the kernels are independent random variables, but not identically distributed: a mix of kernels with different means and variances is not only allowed, but necessary to solve this optimization problem.

Kernels and model parameters

Besides the weights, the other parameters of the models are the parameters attached to each kernel *X*(*n*, *k*). Typically, each kernel *X*(*n*, *k*) is characterized by two parameters: *a*(n, *k*) and *b*(*n*, *k*). In the case of Gaussian kernels, *a*(*n*, *k*) is the mean and *b*(*n*, *k*) is the variance; *b*(*n*, *k*) is set to 1. In the case of Uniform kernels with *Y* taking on positive values, *a*(*n*, *k*) is the lower bound of the support interval, while *b*(*n*, *k*) is the upper bound; in this case, since we want the support domains to form a partition of the set of positive real numbers (the set of potential observations), we use, for any fixed value of *n*, *a*(*n*, 1) = 0 and *b*(*n*, *k*) = *a*(*n*, *k*+1).

Finally, the various kernels should be re-arranged (sorted) in such a way that *X*(*n*, 1) always has the highest weight attached to it, followed by *X*(*n*, 2), *X*(*n*, 3) and so on. The methodology can also be adapted to discrete observations and distributions, as we will discuss later in this chapter.

Algorithms to find the optimum parameters

The goal is to find optimum model parameters, for a specific *n*, to minimize the error *E*(*n*). And then try bigger and bigger values of *n*, until the error is small enough. This can be accomplished in various ways.

The solution consists in computing the derivatives of *E*(*n*) with respect to all the model parameters, and then finding the roots (parameter values that make the derivatives vanish, see for instance section 12 in chapter 28.) For a specific value of *n*, you will have to solve a non-linear system of *m* equations with *m* parameters. In the case of Gaussian kernels, *m* = 2*n*. For uniform kernels, *m* = 2*n* + 1 (*n* weights, *n* interval lower bounds, plus upper bound for the rightmost interval.) No exact solution can be found, so you need to use an iterative algorithm. Potential modern techniques used to solve this kind of problem include:

- Swarm gradient optimization
- EM algorithm
- Stochastic search (see also here)
- Stochastic gradient descent

You can also use Monte-Carlo simulations, however here you face the curse of dimensionality, the dimension being the number *m* of parameters in your model. In short, even for *n* as small as *n* = 4 (that is, *m* = 8), you will need to test trillions of randomly sampled parameter values (*m*-dimensional vectors) to get a solution close enough to the optimum, assuming that you use raw Monte-Carlo techniques. The speed of convergence is an exponential function of *m*. Huge improvements to this method are discussed later in this section, using some kind of step-wise algorithm to find local optima, reducing it to a 2-dimensional problem. By contrast, speed of convergence is quadratic for gradient-based methods, if *E*(*n*) is convex in the parameter space. Note that here, *E*(*n*) may not always be convex though.

Convergence and uniqueness of solution

In theory, both convergence and the fact that there is only one global optimum, are guaranteed. It is easy to see that, under the constraints imposed here on the model parameters, two different mixture models must have two distinct distributions. In the case of Uniform kernels, this is because the support domains of the kernels form a partition, and are thus disjoint. In the case of Gaussian kernels, as long as each kernel has a different mean, no two mixtures can have the same distribution: the proof is left as an exercise. To put it differently, **any relatively well behaved statistical distribution is uniquely characterized by its set of parameters associated with its mixture decomposition**. When using Gaussian kernels, this is equivalent to the fact that any infinitely differentiable density function is uniquely characterized by its coefficients in its Taylor series expansion. This is discussed in the last section.

The fact that under certain conditions, some of the optimization algorithms described in the previous subsection, converge to the global optimum, is more difficult to establish. It is always the case with the highly inefficient Monte Carlo simulations. In that case, the proof is pretty simple and proceeds as follows

- Consider the discrete case where *Y* takes only on positive integer values (for example, your observations consist of counts,) and use the discrete Uniform kernel.
- In that case, the solution will converge to a mixture model where each kernel support domain is a set with one value, and its associate weight is the frequency of that value, in your observed data. This is actually the global optimum, with *E*(*n*) converging to 0 as *n* tends to infinity.
- Continuous distributions can be approximated by discrete distributions after proper re-scaling. For instance, a Gaussian distribution can be perfectly approximated by sequences of increasingly granular binomial distributions. Thus, the convergence to a global optimum, can be derived from the convergence obtained for the discrete approximations.

The stopping rule, that is, deciding when *n* is large enough, is based on how fast *E*(*n*) continue to improve as *n* increases. Initially, for small but increasing values of *n*, *E*(*n*) will drop sharply, but for some value of *n* usually between *n* = 3 and *n* = 10, improvements will start to taper off, with *E*(*n*) slowing converging to 0. If you plot *E*(*n*) versus *n*, the curve will exhibit an elbow, and you can decide to stop at the elbow. See the elbow rule in chapter 25 (section 3.)

Finally, let us denote as *a*(*k*) the limit of *a*(n, *k*) as *n* tends to infinity; *b*(*k*) and *p*(*k*) are defined in a similar manner. Keep in mind that the kernels must be ordered by decreasing value of their associated weights. In the continuous case, a theoretical question is whether or not these limits exist. With Uniform kernels, *p*(*n*, *k*), as well as *b*(*n*, *k*) - *a*(*n*, *k*), that is, the length of the k^{th} interval, should both converge to 0, regardless of *k*, as *n* tends to infinity. The limiting quotient represents the value of *Y*'s density at the point covered by the interval in question. Also, the sum of *p*(*n*, *k*) over all *k*'s, should still be equal to one, at the limit as *n* tends to infinity. In practice, we are only interested in small values of *n*, typically much smaller than 20.

Find near-optimum with fast step-wise algorithm

A near optimum may be obtained fairly quickly with small values of *n*, and in practice this is good enough. To further accelerate the convergence, one can use the following step-wise algorithm, with the Uniform kernel. At iteration *n*+1, modify only two adjacent kernels that were obtained at iteration *n* (that is, kernels with adjacent support domains) as follows:

- Increase the upper bound of the left interval, and decrease the lower bound of the right interval accordingly. Or do the other way around. Note that the cumulative density within each interval, before or after modification, is always equal to 1, since we are using uniform kernels.

- Adjust the two weights, but keep the sum of the two weights unchanged.

So in fact you are only modifying two parameters (degrees of freedom is 2.) Pick up the two adjacent intervals, as well as the new weights and lower/upper bounds, in such a way as to minimize E(n+1).

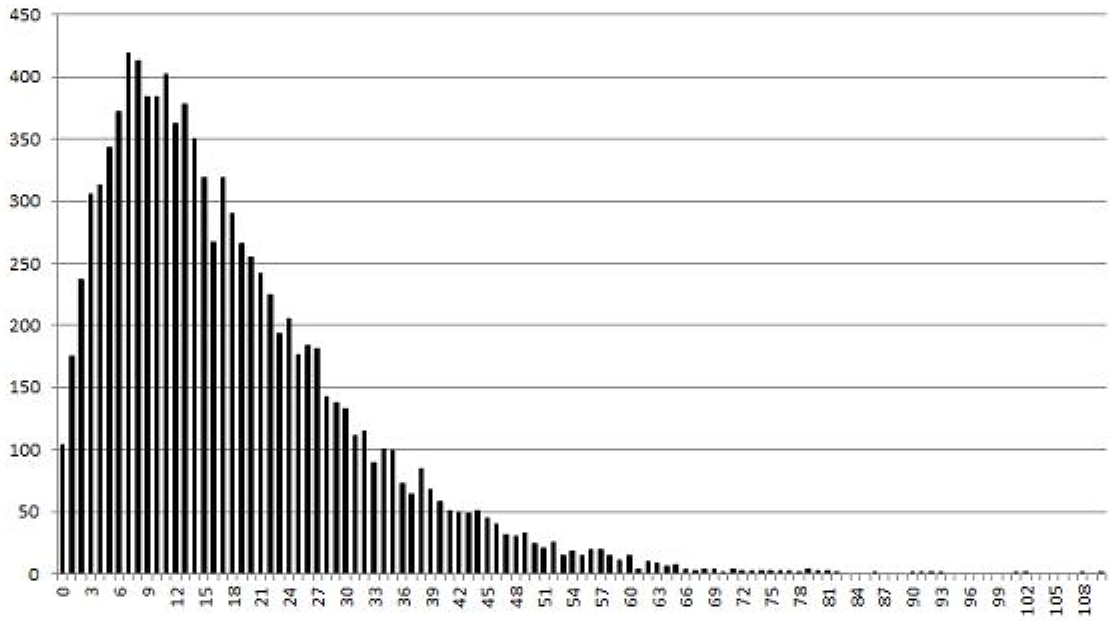
3. Example

Here, I illustrate some of the concepts explained earlier, with an example based on simulated data. The source code and the data is provided so that my experiment can be replicated, and the technical details understood. The 10,000 data points generated (representing Y) are deviates from a skewed, non-symmetrical negative binomial distribution, taking integer values between 0 and 110. Thus we are dealing with discrete observations and distributions. The kernels have discrete uniform distributions, for instance uniform on {5, 6, 7, 8, 9, 10, 11} or on {41, 42, 43, 44}. The choice of a non-symmetrical target distribution (for Y) is to illustrate the fact that the methodology also works for non-Gaussian target variables, unlike the classic central limit theorem framework applying to sums (rather than mixtures) and where convergence is always towards a Gaussian. Here instead, convergence is towards the simulated negative binomial target.

I tried to find online tools to generate deviates from any statistical distribution, but haven't found any interesting ones. Instead, I used R to generate the 10,000 deviates, with the following commands:

```
x = rnbinom(10000, 2, 0.1)
hist(x, breaks = 50)
write.table(x, "c://vincent/deviates.txt")
```

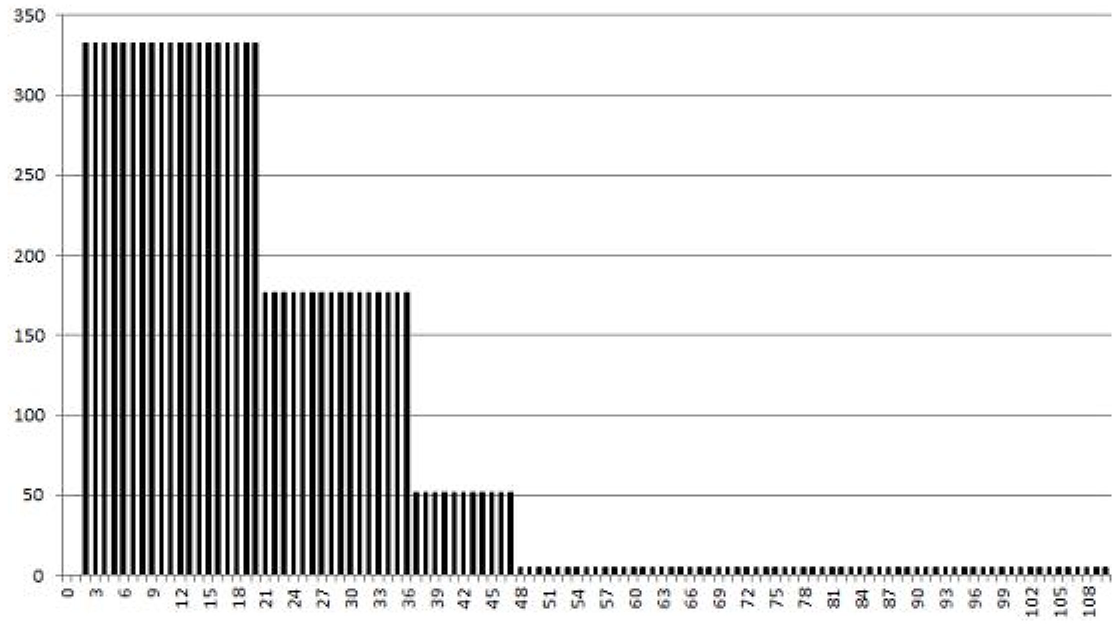
The first line of code generates the 10,000 deviates from a negative binomial distribution, the second line produces its histogram with 50 bins (see picture below, where the vertical axis represents frequency counts, and the horizontal axis represents values of Y.) The third line of code exports the data to an output file that will first be aggregated and then used as an input for the script that (1) computes the model parameters, and (2) computes and minimizes the error $E(n)$.


Histogram for the 10,000 deviates (negative binomial distribution) used in our example

Data and source code

The input data set for the script that processes the data, can be found here. It consists of the 10,000 negative binomial deviates (generated with the above R code), and aggregated / sorted by value. For instance, the first entry (104) means that among the 10,000 deviates, 104 of them have a value equal to 0. The second entry (175) means that among the 10,000 deviates, 175 of them have a value equal to 1. And so on.

The script is written in Perl (you are invited to write a Python version) but it is very easy to read and well documented. It illustrates the raw Monte-Carlo simulations with 4 discrete uniform kernels. So it is very inefficient in terms of speed, but easy to understand, with few lines of code. You can find it here. It produced the distribution (mixture of 4 kernels) that best approximates the above histogram, see picture below.


Approximation of above histogram with mixture model, using 4 uniform kernels

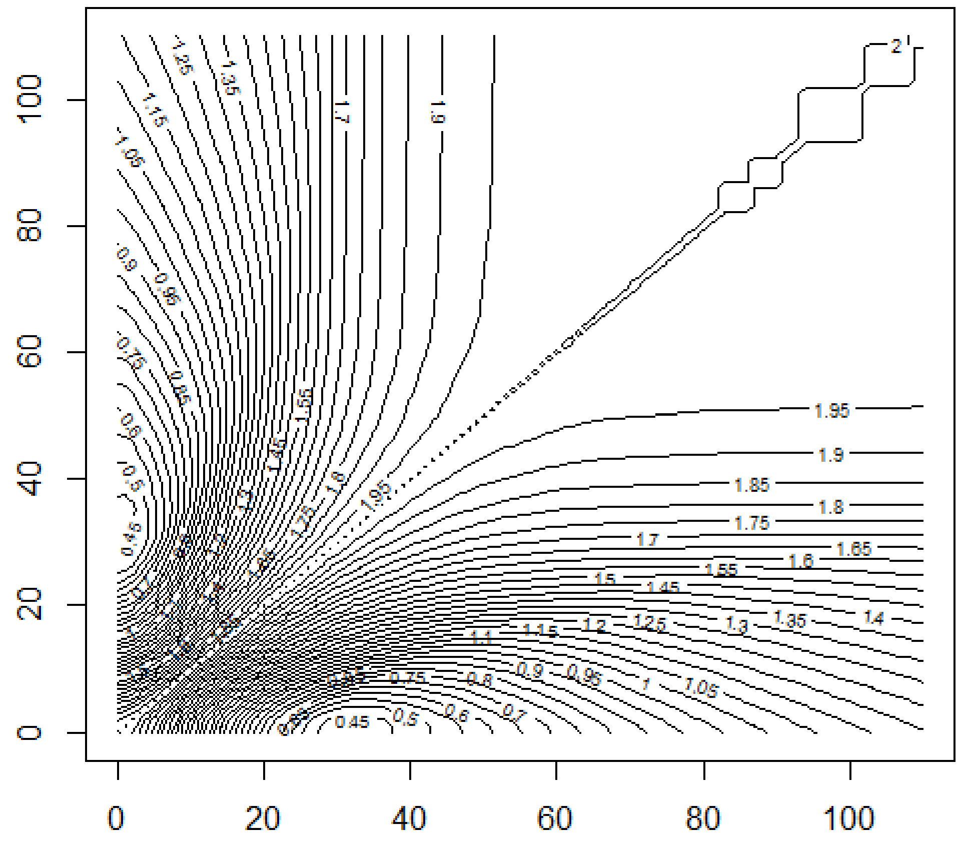
Results

The chart below shows a contour plot for the error E(2), when using *n* = 2 discrete uniform kernels, that is two intervals, with lower bounds of the first interval displayed on the vertical axis, and upper bounds on the horizontal axis. The upper bound of the second (rightmost) interval was set to the maximum observed value, equal to 110. Ignore the curves above the diagonal; they are just a mirror of the contours below. Outside the kernel intervals, densities were kept to 0. Clearly the best kernel (discrete) intervals to approximate the distribution of *Y*, are visually around {1, 2, ... , 33} and {34, ..., 110} corresponding to a lower bound of 1, and an upper bound of 33 for the first interval; it yields an error E(2) less than 0.45.

The contour plot below was produced using the contour function in R, using this data set as input, and the following code:

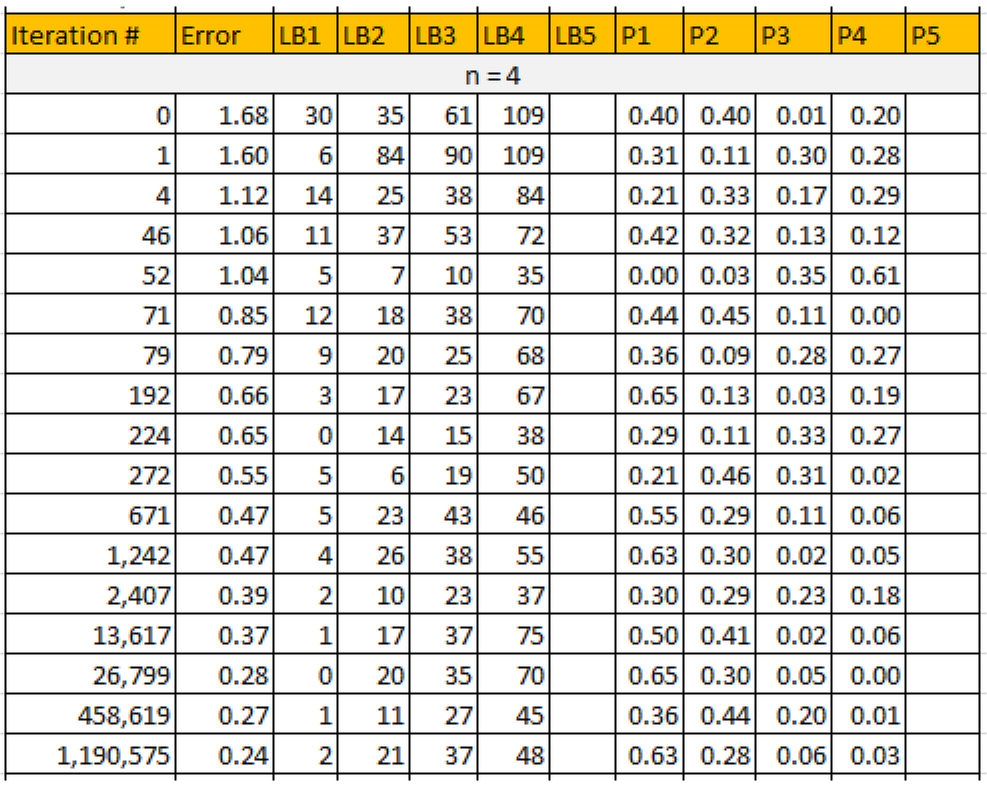
```
data<-read.table("c:/vincentg/rnout2.txt",header=TRUE)
e<-data$error
z <- matrix(e, nrow = 111, ncol = 111, byrow = TRUE)
contour(z, x = seq(0, 110, length.out = nrow(z)),
    y = seq(0, 110, length.out = ncol(z)), nlevels = 40)
```

The interesting thing is that the error function *E*(*n*), as a function of the mixture model parameters, exhibits large areas of convexity containing the optimum parameters, when *n* = 2. This means that gradient descent algorithms (adapted to the discrete space here) can be used to find the optimum parameters. These algorithms are far more efficient than Monte-Carlo simulations.


Contour plot showing the area where optimum parameters are located, minimizing E(1)

I haven't checked if the convexity property still holds in the continuous case, or when you include the weight parameters in the chart, or for higher values of *n*. It still might, if you use the fast step-wise optimization algorithm described earlier. This could be the best way to go numerically, taking advantage of gradient descent algorithms, and optimizing only a few parameters at a time.

Now I discuss the speed of convergence, and improvements obtained by increasing the number of kernels in the model. Here, optimization was carried out via very slow, raw Monte-Carlo simulations. The table below shows the interval lower bounds and weights associated with the discrete uniform kernel, for *n* = 4, obtained by running 2 million simulations. The upper bound of the rightmost interval was set to the maximum observed value, equal to 110. For any given *n*, only simulations performing better than all the previous ones are displayed: in short, these are the records. Using *n* = 5 does not significantly improve the final error E(*n*). Low errors with *n* = 2, 3, and 4 were respectively 0.41, 0.31, and 0.24. They were obtained respectively at iterations 7,662 (*n* = 2), 96,821 (*n* = 3) and 1,190,575 (*n*=4). It shows how slow Monte-Carlo converges, and the fact that the number of required simulations grows exponentially with the dimension *n*. The Excel spreadsheet, featuring the same table for *n* = 2, 3, 4, and 5, can be found here.



Iteration #	Error	LB1	LB2	LB3	LB4	LB5	P1	P2	P3	P4	P5
n = 4											
0	1.68	30	35	61	109		0.40	0.40	0.01	0.20	
1	1.60	6	84	90	109		0.31	0.11	0.30	0.28	
4	1.12	14	25	38	84		0.21	0.33	0.17	0.29	
46	1.06	11	37	53	72		0.42	0.32	0.13	0.12	
52	1.04	5	7	10	35		0.00	0.03	0.35	0.61	
71	0.85	12	18	38	70		0.44	0.45	0.11	0.00	
79	0.79	9	20	25	68		0.36	0.09	0.28	0.27	
192	0.66	3	17	23	67		0.65	0.13	0.03	0.19	
224	0.65	0	14	15	38		0.29	0.11	0.33	0.27	
272	0.55	5	6	19	50		0.21	0.46	0.31	0.02	
671	0.47	5	23	43	46		0.55	0.29	0.11	0.06	
1,242	0.47	4	26	38	55		0.63	0.30	0.02	0.05	
2,407	0.39	2	10	23	37		0.30	0.29	0.23	0.18	
13,617	0.37	1	17	37	75		0.50	0.41	0.02	0.06	
26,799	0.28	0	20	35	70		0.65	0.30	0.05	0.00	
458,619	0.27	1	11	27	45		0.36	0.44	0.20	0.01	
1,190,575	0.24	2	21	37	48		0.63	0.28	0.06	0.03	

4. Applications

The methodology proposed here has many potential applications in machine learning and statistical science. These applications were listed in the introduction. Here, I just describe a few of them in more details.

Optimal binning

These mixtures allow you to automatically create optimum binning of univariate data, with bins of different widths and different sizes. In addition, the optimum number of bins can be detected using the elbow rule described earlier. Optimum binning is useful in several contexts: visualization (to display meaningful histograms), in decision trees, and in feature selection procedures. Some machine learning algorithms, for instance the one described in chapter 2, rely on features that are not too granular and properly binned, to avoid over-fitting and improve accuracy and processing time. These mixture models are handy tools to help with this.

For more on optimal binning, read this article (2013) or check the relevant R package smbinning.

Predictive analytics

Since this methodology creates a simple model to fit with your data, you can use that model to predict frequencies, densities, (including perform full density estimation), intensities, or counts attached to unobserved data points, especially if using kernels with infinite support domains, such as Gaussian kernels. It can be used as a regression technique, or for interpolation or extrapolation, or for imputation (assigning a value to a missing data point), all of this without over-fitting. Generalizing this methodology to multivariate data will make it even more useful.

Test of hypothesis and confidence intervals

These mixtures help build data-driven intermediate models, something in-between a basic Gaussian or exponential or whatever fit (depending on the shape of the kernels) and non-parametric empirical distributions. It also comes with core parameters (the model parameters) automatically estimated. Confidence intervals and tests of hypothesis are easy to derive, using the approximate mixture model distribution to determine statistical significance, *p*-values, or confidence levels, the same way you would do with standard, traditional parametric distributions.

Clustering

Mixture models were invented long ago for clustering purposes, in particular under a Bayesian framework. This is also the case here, and even more so as this methodology gets extended to deal with multivariate data. One advantage is that it can automatically detect the optimum number of clusters thanks to its built-in stopping rule, known as the elbow rule. Taking advantage of convexity properties in the parameter space, to use gradient descent algorithm for optimization, the techniques described in this chapter could perform unsupervised clustering faster than classical algorithms, and be less computer intensive.

Deep learning: Bayesian decision trees

See the subsection on nested mixtures, in section 5, for details.

5. Interesting Problems

We discuss here, from a more theoretical point of view, two fundamental results mentioned earlier, as well as new topics of interest about mixtures, including stable, nested mixtures and potential use in deep learning. All mixtures here may be infinite, and the kernels (in the mixture model) can be correlated.

Gaussian mixtures uniquely characterize a broad class of distributions

Let us consider an infinite mixture model with Gaussian kernels, each with a different mean a_k, same variance equal to 1, and weights p_k that are strictly decreasing. Then the density associated with this mixture is

$$f_Y(y) = \exp(-y^2/2) \cdot \sum_{k=1}^{\infty} c_k \exp(a_k y), \text{ with } c_k = p_k \cdot \exp(-a_k^2/2)/\sqrt{2\pi}.$$

Two different sets of (a_k, p_k) will result in two different density functions, thus the representation uniquely characterizes a distribution. Also, the exponential functions in the sum can be expanded as Taylor series. Thus we have:

$$\sum_{k=1}^{\infty} c_k \exp(a_k y) = \sum_{k=0}^{\infty} d_k \cdot \frac{y^k}{k!}, \text{ with } d_k = \sum_{i=1}^{\infty} c_i a_i^k.$$

Density functions infinitely differentiable at $y = 0$, can be represented in this way. Convergence issues are beyond the scope of this chapter.

Weighted sums fail to achieve what mixture models do

It is not possible, using an infinite weighted sum of independent kernels of the same family, to represent any arbitrary distribution. This fact was established in chapter 10 in the case where all the kernels have the exact same distribution. It is mostly an application of the central limit theorem. Here we generalize this theorem to kernels from a same family of distributions, but not necessarily identical. By contrast, the opposite is true if you use mixtures instead of weighted sums.

With a weighted sum of Gaussian kernels of various means and variances, we always end up with a Gaussian distribution (see here for explanation.) With Uniform kernels (or any other kernel family) we can prove the result as follows:

- Consider a sum of n kernels from a same family. Say n_1 of them have (almost) the same parameters, another n_2 of them have the same parameters but different from the first group, another n_3 of them have the same parameters but different from the first two groups, and so on, with $n = n_1 + n_2 + ...$
- Let n tends to infinity, with n_1, n_2 and so on also tend to infinity. The weighted sum in each group will converge to Gaussian, by virtue of the central limit theorem.
- The overall sum across all groups will tend to a sum of Gaussian, and thus must be Gaussian. This depends on how fast the weights are decaying. Details about the decaying rate, for the result to be correct, are provided in the previous chapter.

By contrast, a mixture or any number of Gaussian kernels with different means is not Gaussian.

Stable mixtures

Just like the Gaussian family is stable with respect to weighted sums, in the sense that the weighted sum of independent Gaussian is Gaussian (it is indeed the only type of distribution with finite variance, stable under addition, see previous chapter), is it possible to find families of kernel distributions that are stable when mixed? In order to answer this question, it is enough to identify two different kernels *X* and Z belonging to a same family, such that the densities (regardless of the kernel parameters) satisfy

$$f_Y(y) = pf_X(y) + (1-p)f_Z(y),$$

with *Y* also belonging to the same family, regardless of the weight *p*. Note that *X* and *Z* can be correlated here. Clearly, the Gaussian family is not stable under mixing.

However, there is actually a large number of stable kernels for mixture models. Let *g* and *h* be arbitrary density functions. Then we have the following result:

$$\text{If } f_X(y) = ag(y) + (1-a)h(y),\ f_Z(y) = a'g(y) + (1-a')h(y),\ \text{with } a, a' \in [0,1],$$
$$\text{then } f_Y(y) = a''g(y) + (1-a'')h(y), \text{with } a'' = pa + (1-p)a' \in [0,1].$$

In short, for mixtures, we have an infinite class of stable kernel families, of all shapes. Interestingly, if you choose two Gaussian with different means for *g* and *h*, then the resulting kernel (a mixture itself), is stable under mixing. That is, it belongs to the same family. So a mixture of different Gaussian constitutes a stable family of distributions for mixtures, but not for weighted sums. Yet the Gaussian kernel itself is not stable for mixtures, while it is the only stable family for weighted sums.

Note that stable kernels are not limited to two components. It easily generalizes to *n* components.

Nested mixtures and Hierarchical Bayesian Systems

In the previous subsection on stable mixtures, we've seen that the components (kernels) of a mixture can be mixtures themselves. So you can recursively build a tree of nested mixtures, with as many nodes as you wish, and as deep as you wish. What's more, all the mixtures, at any level in the hierarchy, can share the same arbitrary family of distributions (with any number of parameters), each mixture with its own set of parameters. This is just a standardized deep learning, Bayesian hierarchical system. For instance, with the notations used in the previous subsection, P(*Y*), P(*Z* | *Y*) and P(*X* | *Y*) have the same distribution, up to a change in parameters. This also works if the distributions are multivariate.

For a standard treatment of nested mixtures, see here (*Deep Gaussian Mixture Models*, paper submitted for publication in November 2017) and here (*Hierarchical Mixture Models for Nested Data Structures*, undated).

Correlations

A sum *Y* = *X* + *Z* of independent random variables is always correlated with each of its summands (unless *Y* is constant, which is not possible if *X* and *Z* are independent.) This is also true for mixtures. Using the same mixture (with two components) as in the subsection on stable mixtures, prove the following:

$$\rho(Y,X) = p\cdot\sigma_X/\sigma_Y,\ \rho(Y,Z) = (1-p)\cdot\sigma_Z/\sigma_Y,$$
$$\sigma_Y^2 = pE(X^2) + (1-p)E(Z^2) - (pE(X) + (1-p)E(Z))^2.$$

Here ρ and σ are used to denote the correlation and standard deviation, respectively. How does this formula generalize to any number of kernels?

A consequence is that for kernels with identical variances (as in the theoretical model), ordered by decreasing weights, the successive correlations between a component (kernel) and the target distribution *Y*, are also decreasing. This is a bit like a principal component analysis, and it can also be used for data reduction. The difference here is that the components are created from scratch, using the algorithms described in section 2. In practice, unequal kernel variances are allowed: they boost the speed of convergence, but the price to pay, depending on the kernel family being used, is that two different sets of parameters can lead to the same target distribution *Y.* The solution may no longer be unique.

Note that if instead of a mixture, we consider the weighted sum *Y* = *pX* + *qZ*, with *X* and *Z* independent, the correlation formulas above (as well as the conclusions) are still valid; the only thing that changes is the formula for the variance of *Y*.

12. Heavily Auto-correlated Time Series

We investigate a large class of auto-correlated, stationary time series, proposing a new statistical test to measure departure from the base model, known as Brownian motion. We also discuss a methodology to deconstruct these time series, in order to identify the root mechanism that generates the observations. The time series studied here can be discrete or continuous in time, they can have various degrees of smoothness (typically measured using the Hurst exponent) as well as long-range or short-range correlations between successive values. Applications are numerous, and we focus here on a case study arising from some interesting number theory problem. In particular, we show that one of the times series investigated in my article on randomness theory [see Appendix B, read section 4.1.(c)] is not Brownian despite the appearance. It has important implications regarding the problem in question. Applied to finance or economics, it makes the difference between an efficient market, and one that can be gamed.

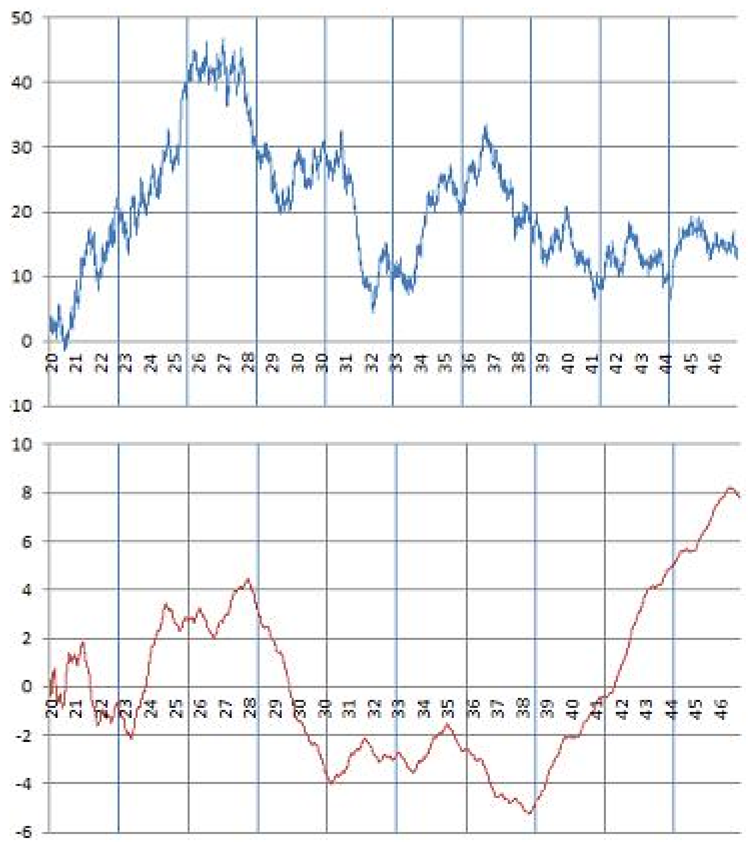
This chapter it accessible to a large audience, thanks to its tutorial style, illustrations, and easily replicable simulations. Nevertheless, we discuss modern, advanced, and state-of-the-art concepts. This is an area of active research.

1. Introduction and time series deconstruction

We are dealing with a series of N observations or events denoted as z_1, ..., z_N and indexed by time. The respective times of arrival are denoted as T_1, ..., T_N. Events are equally spaced in time, and typically, N is large while time intervals are small, thus providing a good approximation to a time-continuous process. The time series discussed here are assumed to have stationary increments with unit variance and zero mean. We will define what this means exactly when needed.

1.1. Example

The picture below shows typical examples of the time series that we are dealing with in this chapter. The X-axis represents the time. These are discrete approximations of time-continuous series found in many contexts, in particular in finance.


Smooth (bottom) versus rugged time series (top)

Interestingly, these two examples come from number theory, and are studied later in this article. In each case, it consists of 22,000 observations. The chart at the top is a classic example of a Brownian motion, while the one at the bottom exhibits long-range auto-correlations not found in traditional Brownian motions. The statistical tests discussed in section 2 help assess which type of time series we are dealing with. .

1.2. Deconstructing time series

The observed time series considered here are typically the result of a cumulative process. The parent process { y_n } causing the pattern usually results (but not always as we shall see) in { z_n } being a Brownian motion or a fractional Brownian motion. The parent process, sometimes called *differential process*, is defined as follows:

$$y(n) = z(n)\sqrt{T(n)} - z(n-1)\sqrt{T(n-1)} \text{ for } n > 1, \text{ and } y(1) = z(1).$$

The square root factors in the above formula are needed, as increments z_n - z_{n-1} are very small, since { z_n } mimics a time-continuous process. For instance, in the above figure, we have 750 observations in a time interval of length 1. And indeed, if you do the reverse operation, starting with the parent process -- consisting (say) of independent and identically distributed random variables with mean 0 and variance 1 -- then

$$z(n) = \frac{y(1) + \cdots + y(n)}{\sqrt{T(n)}}.$$

The square root factor is clearly mandatory here, by virtue of the central limit theorem, to keep the variance finite and non-zero in { $z(n)$ }. For details, see chapter 1 in my book on applied stochastic processes.

Note:

If your observed { z_n } is stationary, proceed as follows. Shift the time axis (that is, shift the T_n values) so that the new origin is far in the future. This is implemented and illustrated in my spreadsheet (shared later in this article) via the *offset* parameter, and it fixes issues near the origin. Indeed, if { z_n } is stationary, then time location does no matter as far as probabilistic properties are concerned, because of the very definition of stationarity. By doing so, the parent process { y_n } is also (almost) stationary.

1.3. Correlations, Fractional Brownian motions

The traditional setting consists of { y_n } being a white noise, that is, a sequence of independent and identically distributed random variables with mean 0 and variance 1 in this case. The resulting time-continuous limit of { z_n } is then called a Brownian motion. In most cases investigated here, the y_n's are not independent and exhibit auto-correlations. The resulting process is then called a fractional Brownian motion. And in some cases, { y_n } may not even be stationary. We will show what happens then.
The stronger the long-range correlations in { y_n }, the smoother { z_n } looks like. The degree of smoothness is usually measured using the Hurst exponent, described in the next section.

2. Smoothness, Hurst exponent, and Brownian test

The traditional and simple metric to measure the smoothness in your data is called the *detrending moving average*, and it is abbreviated as DMA. It is the mean square error between your observations and its various moving averages of order m = 1, 2, 3, and so on. The exact definition can be found in this article (*Statistical test for fractional Brownian motion based on detrending moving*, by Grzegorz Sikoraa, 2018, see section 2). Other criteria are also used, such as FA and DFA. A comparison of these metrics can be found in this article (*Comparing the performance of FA, DFA and DMA using different synthetic long-range correlated time series*, by Ying-Hui Shao *et al*., 2018). DMA, along with other metrics, are used in our computations.

With the notation DMA(m) to emphasize the fact that it depends on m, we have this well-known result:

$$\text{DMA}(m) \sim C(H) \cdot m^{2H}.$$

This is an asymptotic result, meaning that it becomes more accurate as m grows to infinity. The constant H is known as the Hurst exponent. See here (section 2) for details. H takes on values between 0 and 1, with $H = 1/2$ corresponding to the Brownian motion (see also chapter 1 in this book.) Higher values correspond to smoother time series, and lower values to more rugged data.

Let's N be the number of observations in your time series. We used $N = 22{,}000$ in all our examples, and typically, m of the order $N^{1/2}$. The above asymptotic result is not applicable in our context, and we use a slightly different methodology.

2.1. Our Brownian tests of hypothesis

Testing the Brownian character of a time series is typically done using the above formula with the Hurst exponent. See here and here for details. Our approach here is different, as we are more interested in small-range and mid-range correlations, than in long-range ones. We use the notation $S(N, m)$ instead of DMA(m), since this metric also depends on your sample size N.

We performed two types of tests. The first one is based on $S(N, m)$, and we used m = 100, 200, up to 500. We used the correlation R between $\{ S(N, m) \}$ and $\{ m \}$ computed on these 5 values of m, with $N = 22{,}000$. Since the correlation is very close to 1 in all examples, the actual test statistic is $-\log(1 - R)$. Its distribution can be empirically computed by simulations. The second test is based on auto-correlations of lag m, with m = 1, 100, 200, 300, 400 and 500, both for the observations $\{ z_n \}$ and the deconstructed time series $\{ y_n \}$. The result with detailed computations, using 6 time series A, B, C, D, E, F, are available in my spreadsheet in section 2.2.

2.2. Data

We tested the methodology on different types of time series. The results are illustrated in the pictures below, and replicable using my spreadsheet. The time series $\{ z(n) \}$ were constructed as follows:

- Step 1: Create a base process $\{ x_n \}$.
- Step 2: Standardize $\{ x_n \}$ so that its mean and variance become 0 and 1 respectively. The resulting sequence is $\{ y_n \}$.
- Step 3: Create the cumulative process $\{ z_n \}$ using the formula in section 1.2.

The time T_n was set to $T(_n) = 600 + n/750$. That's what makes the series $\{ z_n \}$ look like continuous in time.

The six time series (simulations) investigated here are constructed as follows. They are also pictured in section 3.1. Here INT is the integer part function.

- **Series A**: Use $x_{n+1} = bx_n - \text{INT}(bx_n)$ with $x_1 = \log 2$ and $b = (1 + 5^{1/2})/2$.
- **Series B**: Use $x_{n+1} = bx_n - \text{INT}(bx_n)$ with $x_1 = \pi/4$ and $b = (1 + 5^{1/2})/2$.

- **Series C**: Here x_n is a Bernouilli deviate of parameter 1/2. The x_n's are independent.
- **Series D**: Use $x_{n+1} = b + x_n$ - INT($b + x_n$) with $x_1 = 1$ and $b = 2^{1/2}$. In addition, use $z'_n = z_n - n^{1/2}/2$, rather than the standard z_n.
- **Series E**: Use $x_{n+1} = b + x_n$ - INT($b + x_n$) with $x_1 = \log 2$ and $b = 16\pi$. In addition, use z'_n defined by $z'_{n+1} = z'_n + z_{n+1}/(T_n)^3$, with $z'_1 = z_1$ and $T_1 = 24$, rather than the standard z_n.
- **Series F**: Use $x_{n+1} = b + x_n$ - INT($b + x_n$) with $x_1 = 0$ and $b = (1 + 5^{1/2})/2$.

Series A and B are generated by a *b*-process, while series D, E, and F are generated by a perfect process. The purpose of this study was to compare *b*-processes with perfect processes, and their ability to generate Brownian motions of fractional Brownian motions. Perfect processes and *b*-processes were introduced in my article on the theory of randomness, see Appendix B in this book. Series D is actually pictured in section 4.1(c) in that appendix. Series C corresponds to the classic Brownian motion.

The data and computations are available in my spreadsheet, here. Both columns D and I represent the same exact { y_n } in the spreadsheet. But column D is used to build { z_n }, while column I assumes that you only observe { z_n } and must compute { y_n } from scratch, by deconstructing { z_n }.

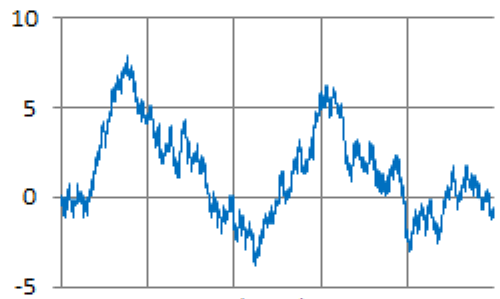
3. Results and conclusions

In this section, we summarize our findings. Many illustrations are provided.

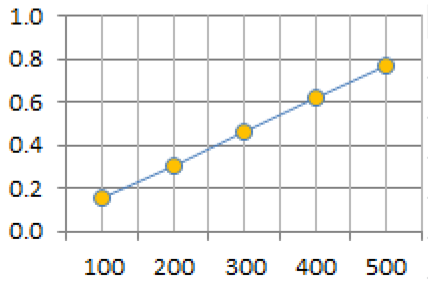
3.1. Charts and interpretation

The first three series A, B, C in our picture below feature processes that behave pretty much like Brownian motions, with an Hurst exponent H equal or close to 1/2. Series A and B are two realizations of the exact same processes, as b is identical in both cases. Series C illustrates a perfect Brownian motion, with $H = 1/2$.

Note that the auto-correlations in the deconstructed time series { y_n } rapidly drop to 0, while the correlations in { z_n } are very high, but slowly drop to a value between 0.80 and 0.90 when $m = 500$. As a result, $S(N, m)$, as a function of m, is a perfect straight line (m is the order of the moving average; N = 22,000 is the total number of observations.)

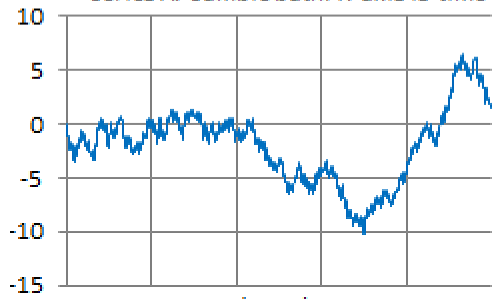


Series A: Sample path, X-axis is time

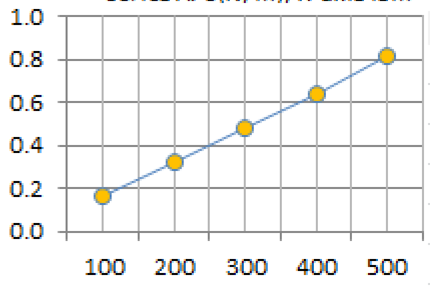


Series A: $S(N, m)$, X-axis is m

Series A	lag-*m* autocorrel	
m	{ *z* (*n*) }	{ *y* (*n*) }
1	1.00	0.31
100	0.96	0.00
200	0.92	0.00
300	0.88	-0.01
400	0.84	-0.01
500	0.81	0.00

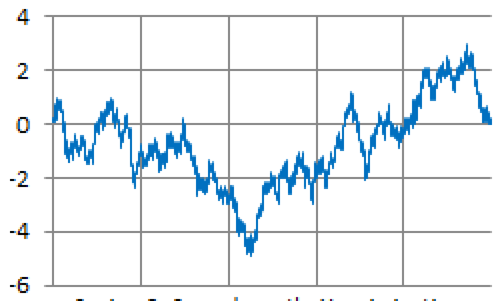


Series B: Sample path, X-axis is time

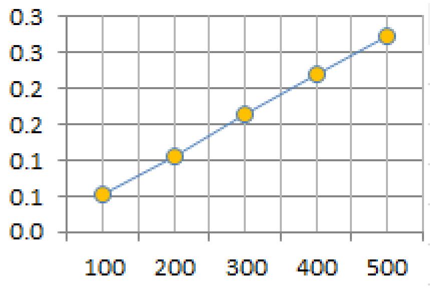


Series B: $S(N, m)$, X-axis is m

Series B	lag-*m* autocorrel	
m	{ *z* (*n*) }	{ *y* (*n*) }
1	1.00	0.31
100	0.98	0.00
200	0.96	0.00
300	0.94	0.00
400	0.92	0.01
500	0.90	0.01



Series C: Sample path, X-axis is time

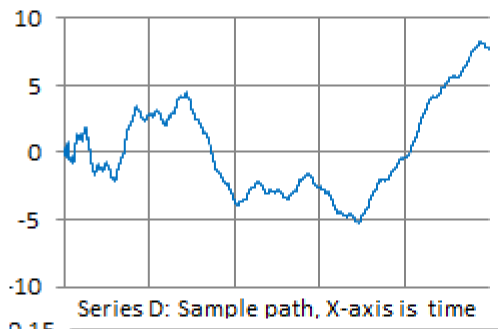


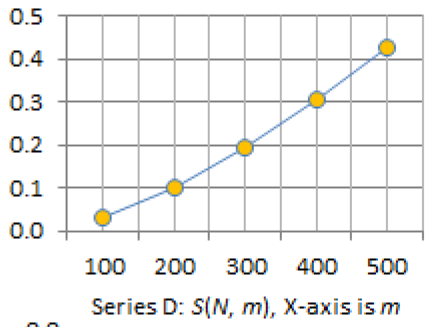
Series C: $S(N, m)$, X-axis is m

Series C	lag-*m* autocorrel	
m	{ *z* (*n*) }	{ *y* (*n*) }
1	1.00	0.00
100	0.97	0.00
200	0.93	0.01
300	0.90	0.01
400	0.86	-0.01
500	0.83	0.00

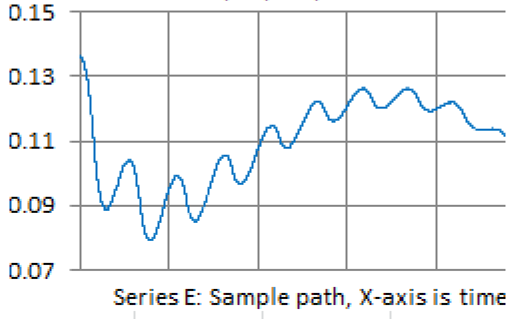
Time series D, E, and F, pictured below, behave very differently from A, B, and C. Series D exhibits very high auto-correlations in { z_n } while auto-correlations in { y_n } slowly drop to 0. It is smoother than A, B, and C. As a result, $S(N, m)$, as a function of m, is no longer a straight line: it is now a convex function. If this was a financial time series, it would correspond to a non-efficient market. So the statistical tests described in section 2 can be used to test market efficiency.

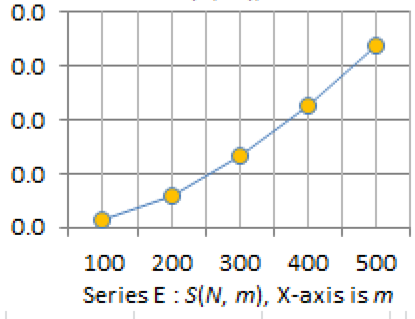
The smoothness is even more pronounced in series E. In this case, auto-correlations in { y_n } are long-range and do not drop to zero. Series F, to the contrary, is very rugged. Auto-correlations in { z_n } are lower than in the other examples, and $S(N, m)$ is now mostly concave. There are still long-range auto-correlations if you look at { y_n }. We are dealing with a mixture of smoothness and bumpiness, though the smooth part is not visible with the naked eye. The rugged part also shows up in the first half of the $S(N, m)$ curve, which is concave, while the smooth part shows up in the second half of the $S(N, m)$ curve, which is convex.



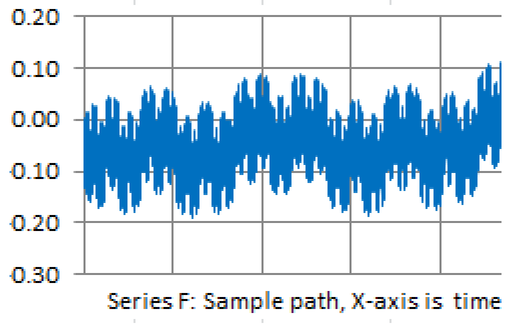


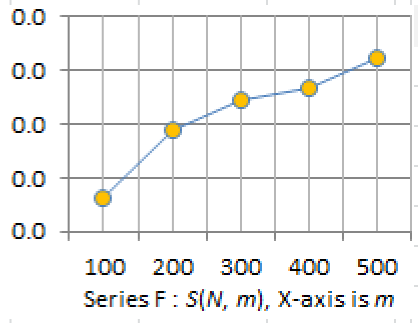
Series D	lag-*m* autocorrel	
m	{ *z* (*n*) }	{ *y* (*n*) }
1	1.00	0.58
100	1.00	0.23
200	0.99	0.13
300	0.97	-0.11
400	0.96	0.10
500	0.94	0.04





Series E	lag-*m* autocorrel	
m	{ *z* (*n*) }	{ *y* (*n*) }
1	1.00	0.94
100	0.99	0.81
200	0.96	0.68
300	0.92	0.47
400	0.87	0.36
500	0.81	0.19





Series F	lag-*m* autocorrel	
m	{ *z* (*n*) }	{ *y* (*n*) }
1	0.73	-0.42
100	0.25	0.05
200	0.25	-0.43
300	0.10	-0.45
400	0.15	-0.01
500	0.34	0.90

3.2. Conclusions

We have explored four types of time series, and characterized them using auto-correlation indicators:

- Brownian-like with very short-range auto-correlations in the deconstructed time series { y_n }. Examples: series A and B.
- Brownian for series C, with no auto-correlation in the deconstructed time series { y_n }.
- Smooth, fractional Brownian-like for series D (in series E, { y_n } is not stationary, so it is not Brownian at all).
- Rugged, fractional Brownian-like for series E

The tests presented here can be integrated in a Python library. The initial purpose was to compare *b*-processes (series A and B) with perfect processes (series D, E and F). These two processes have been found here to be very different. Indeed, perfect processes are so peculiar that the standard division by $n^{1/2}$ in the construction of { z_n } [section 1.2.] does not work. Factors other than $n^{1/2}$ must be used, and even then, the final time series { z_n } is not a perfect Brownian motion, not even close: it usually has a smooth component and long-range auto-correlations in { y_n }. This makes perfect processes less attractive than *b*-processes, for use in cryptographic applications. But more attractive, to model inefficient markets or less than perfect randomness.

13. Multivariate Time Series

We study some interesting multivariate time series, using a number theory problem related to the material in Appendix B, and building on the univariate time series studied in the previous chapter. The multivariate case is discussed in section 2 as well as in the last section of this chapter, featuring a gaming application.

So many fascinating and deep results have been written about the number $(1 + 5^{1/2})/2$ and its related sequence - the Fibonacci numbers - that it would take years to read all of them. This number has been studied both for its applications (population growth, architecture) and its mathematical properties, for over 2,000 years. It is still a topic of active research.

I show here how I used the golden ratio for a new number guessing game (to generate chaos and randomness in ergodic time series) as well as new intriguing results, in particular:

- Proof that the rabbit constant is not normal in any base; this might be the first instance of a non-artificial mathematical constant for which the normalcy status is formally established.
- Beatty sequences, pseudo-periodicity, and infinite-range auto-correlations for the digits of irrational numbers in the numeration system derived from perfect stochastic processes
- Properties of multivariate *b*-processes, including integer or non-integer bases.
- Weird behavior of auto-correlations for the digits of normal numbers (good seeds) in the numeration system derived from stochastic *b*-processes
- A strange recursion that generates all the digits of the rabbit constant

This chapter also features techniques to de-correlate time series.

1. Some Definitions

We use the following concepts in this article:

- A *normal number* is a number that has its digits uniformly distributed. If you pick up a number at random, its binary digits are uniformly distributed: the proportion of zero's is 50% and the digits are not auto-correlated, among other things. No one knows if constants such as π, log 2, $2^{1/2}$, or the Euler constant, are normal or not.

- Rather than normal numbers, we rely on the concept of *good seeds*, which is a generalization to numeration systems where the base *b* might not be an integer.

In such systems, the vast majority of numbers (good seeds) have digits that are distributed according to some specific *equilibrium distribution*, usually not a uniform distribution. Also they have a specific auto-correlation structure. Any number with a different digit distribution or auto-correlation structure is called a *bad seed*. Typically, rational numbers are bad seeds. Examples of numeration systems, with their equilibrium distribution, are discussed here, also in Appendix B and in my book on stochastic processes.

- The concept of numeration system can be extended to non-integer bases. Two systems have been studied in detail: *perfect processes* and *b-processes*, see Appendix B. The *b*-process generalizes traditional numeration systems. In that system, a sequence $x_{n+1} = bx_n$ - INT(bx_n) is attached to a seed x_1, where INT represents the integer part function, and b is a real number larger than 1. The nth digit of the seed x_1 is defined as INT(bx_n). When b is an integer, it corresponds to the traditional base-b numeration system.

- The perfect process of base b is defined by the recursion $x_{n+1} = b + x_n$ - INT(b + x_n) and a seed x_1, where b is a positive irrational number. In that system, the nth digit of the seed x_1 is defined as INT($2x_n$). All seeds including $x_1 = 0$ are good seeds. Also, $x_{n+1} = nb + x_1$ - INT($nb + x_1$). A table comparing *b*-processes with perfect processes is provided in section 4.1(b) in Appendix B. Perfect processes are related to *Beatty sequence* (see also here.)

- By *gentle chaos*, we mean systems that behave completely chaotically, but that are ergodic. By ergodicity, we mean that these systems have equilibrium distributions, also called attractor distributions in the context of dynamical systems, or a stable distribution (see chapter 10) in the context or probability theory. The equilibrium can be found using a very long sequence x_n starting with any good seed, or using x_1 only and a large number of different (good) seeds.

2. Digits Distribution in *b*-processes

It is known that the digits are not correlated, and that the digit distribution is uniform if b is an integer. If the base b is not an integer, the digits take values 0, 1, 2, and so on, up to INT(b). Then, the digit distribution and auto-correlation (for good seeds) is known only for special bases, such as the golden ratio, the super-golden ratio, and the plastic number: see section 4.2 in Appendix B for details. Also, the lag-k auto-correlation in base b is equal to the lag-1 auto-correlation in base b^k. The picture below shows the empirical lag-1 auto-correlation for b in]1, 4]. The bumps are real and not caused by small sample sizes in our computations.

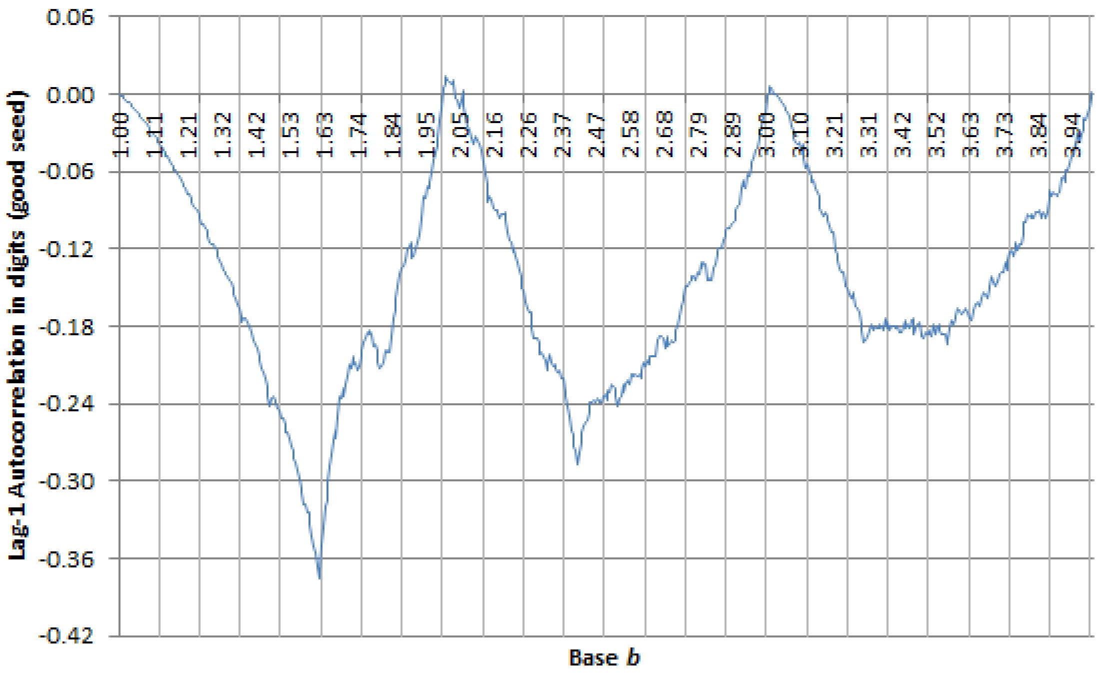


Figure 1: *Lag-1 auto-correlation in digit distribution of good seeds, for b-processes*

Figure 1 shows that the lag-1 auto-correlation, for any good seed, is almost always negative. In particular, it is always negative if *b* is in]1, 2[. It is minimum for the golden ratio $b = (1 + 5^{1/2})/2$ and in that case, its value is $(-3 + 5^{1/2})/2$. This fact can be proved using results in Appendix B (see section 3.2.(a) about the golden ratio process.)

Finally, unlike perfect processes that have long range (indeed, infinite range) auto-correlations just like periodic time series, for *b*-processes auto-correlations are decaying exponentially fast. See Chapter 12 for illustrations. For an exact formula for the cross-correlation between the two components of a bivariate perfect process, see section 3.1 in Chapter 15, or see here.

The digit distribution, for *b* in]1, 2], is pictured in section 4.3.(b) in Appendix B. If *b* is in]1, 2[, the digits are binary and the proportion of zero's is always less than 50%. .

3. Strange Facts and Conjectures about the Rabbit Constant

The rabbit constant *R* = 0.709803442861291 ... is related to Fibonacci numbers (and thus to the golden ratio) used to model demographics in rabbit populations. It is typically defined by its sequence of binary digits in the ordinary binary numeration system (a special case of *b*-processes with *b* = 2) and it has an interesting continued fraction expansion, see here.

We use here a different approach to construct this number, leading to some interesting results. First, let us introduce a new constant. We call it the *twin rabbit constant*, and it is denoted as *R**.

The twin rabbit constant R^* is built as follows:

- $x_n = n\ (-1 + 5^{1/2})/2 - \text{INT}(n\ (-1 + 5^{1/2})/2)$ for $n = 1, 2$, and so on
- $d_n = \text{INT}(2x_n)$ is equal to 0 or 1
- $R^* = d_1/2 + d_2/4 + d_3/8 + d_4/16 + d_5/32 + \dots = 0.6470592723139 \dots$

The rabbit constant R is built as follows, using the same sequence x_n:

- $x_n = n\ (-1 +5^{1/2})/2 - \text{INT}(n\ (-1 +5^{1/2})/2)$ for $n = 1, 2$, and so on
- $g(n) = \text{INT}(x_n)$
- $e_n = g(n+1) - g(n)$ and is thus equal to 0 or 1
- $R = e_1/2 + e_2/4 + e_3/8 + e_4/16 + e_5/32 + \dots = 0.709803442861291 \dots$

Note that x_n is a perfect process with $b = (-1 + 5^{1/2})/2$. We have the following properties:

3.1. Facts and Conjectures

Here are a few surprising facts:

- The digits d_n and e_n, respectively of R^* and R, are identical about 88% of the time. The exact figure is probably $(4 - 5^{1/2})/2$.
- If d_n and e_n are different, d_m and e_m are different, with $m > n$, and for all values between n and m, the digits are identical, then $m - n$ must be equal to 5, 8 or 13. This is still a conjecture; I haven't proved it.
- The function $g(n)$ satisfies the recurrence relation $g(n) = n - g(g(n-1))$ with $g(0) = 0$. I published the proof in 1988, in Journal of Number Theory (download the proof).
- The lag-1 auto-correlation in the digit sequence $\{ e_n \}$ is equal to $(1 - 5^{1/2})/2$. You can try to prove this fact, as an exercise. This is lower than the lowest value that can be achieved with any good seed, in any b-process. We have the same issue with the sequence $\{ d_n \}$. Thus, **the binary digit sequences $\{ e_n \}$ and $\{ d_n \}$ of the rabbit and twin rabbit numbers cannot generate a good seed (or normal number) in any base**.
- The proportion of digits equal to zero in the rabbit number, is $(3 - 5^{1/2})/2$, also too low to be a good seed, regardless of the base. For the twin rabbit number, the proportion is 50%.

It would be interesting to study the more general case where b is any positive irrational number, constructing twin numbers using the same methodology, and analyze their properties. Some of the candidate numbers include those listed in the Beatty sequence. Here we only focused on $b = (-1 + 5^{1/2})/2$. As a general result, the binary digits of the twin numbers generated this way, can never generate a good seed in any base, because they are too strongly auto-correlated.

4. Gaming Application

We use this technology in a generic number guessing game. The gaming platform features pre-computable winning numbers, and payout based on the distance between guesses and winning numbers. This system is described in chapter 18. It mimics a stock market or lottery game depending on the model parameters. At its core, among many sequences, we also use the golden ration *b*-process $\{ x_n \}$ described in section 3.2 in the Appendix. Here $b = (1 + 5^{1/2})/2$. Of course, we start with a good seed.

In order to make the number guessing process more challenging, we de-correlate the digits. For this purpose, we consider two options.

4.1. De-correlating Using Mapping and Thinning Techniques

This option consists of de-correlating the sequence $\{ x_n \}$. The first step is to map $\{ x_n \}$ onto a new sequence $\{ y_n \}$, so that the new equilibrium distribution becomes uniform on [0, 1]. This is achieved as follows:

If $x_n < b - 1$, then $y_n = x_n / (b - 1)$ else $y_n = (x_n - (b-1)) / (2-b)$.

Now the $\{ y_n \}$ sequence has a uniform equilibrium distribution on [0, 1]. However, this new sequence has a major problem: high auto-correlations, and frequently, two or three successive values that are identical (this would not happen with a random *b*, but here *b* is the golden ratio -- a very special value -- and this is what is causing the problem.)

A workaround is to ignore all values of x_n that are larger than $b - 1$, that is, discarding y_n if x_n is larger than $b - 1$. This is really a magic trick. Now, not only the lag-1 auto-correlation in the remaining $\{ y_n \}$ sequence is equal to 1/2, the same value as for the full $\{ x_n \}$ sequence with $b = 2$, but the lag-1 auto-correlation in the remaining sequence of binary digits (digits are defined as INT(by_n) is also equal to zero, just like for ordinary digits in base 2.

4.2. Dissolving the Auto-correlation Structure Using Multivariate *b*-processes

An interesting property of *b*-processes is the fact that auto-correlations in $\{ x_n \}$ are decaying exponentially fast. In fact, for any good seed, the lag-*k* auto-correlation in base *b* is equal to the lag-1 auto-correlation in base b^k. Note that if *b* is an integer, the lag-1 auto-correlation is equal to $1/b$.

Another interesting property is the fact that two sequences $\{ x_n \}$ and $\{ y_n \}$ using different (good) seeds x_1 and y_1, and the same base *b*, are independent if the seeds are independent in base *b*. The concept of *independent seeds* will be formally defined in a future article, but it is rather intuitive. For instance, the seeds x_1 and $y_1 = x_3$ are not independent, regardless of the base.

Thus, in order to dilute the auto-correlations by a factor b^k, one has to interlace k sequences using the same base b for each sequence, but using k independent good seeds, one for each sequence. Doing so, we are actually working with multivariate b-processes that are not cross-correlated. The same mechanism can be applied recursively to each of the k sequences, eventually resulting in multiple layers of nested sequences (a tree structure) to further reduce auto-correlations. Finally, re-mapping the resulting process may be necessary to obtain a uniform equilibrium distribution.

Note that if b is an integer, there is no need to de-correlate as the sequence of digits is automatically free of auto-correlations. Also, in that case, no re-mapping is needed as the equilibrium distribution is uniform to begin with.

14. Statistical Tests: Summary

We have explored many statistical tests in the previous chapters. Here is a summary. More generic model-free tests are discussed in Part 5 of this book.

Many of the following statistical tests are rarely discussed in textbooks or in college classes, much less in data camps. Yet they help answer a lot of different and interesting questions. I used most of them without even computing the underlying distribution under the null hypothesis, but instead, using simulations to check whether my assumptions were plausible or not. In short, my approach to statistical testing is model-free, data-driven. Some are easy to implement even in Excel. Some of them are illustrated here, with examples that do not require statistical knowledge for understanding or implementation.

This material should appeal to managers, executives, industrial engineers, software engineers, operations research professionals, economists, and to anyone dealing with data, such as biometricians, analytical chemists, astronomers, epidemiologists, journalists, or physicists. Statisticians with a different perspective are invited to discuss my methodology and the tests described here. In my case, I used these tests mostly in the context of experimental mathematics, which is a branch of data science that few people talk about. In that context, the theoretical answer to a statistical test is sometimes known, making it a great benchmarking tool to assess the power of these tests, and determine the minimum sample size to make them valid.

I provide here a general overview, as well as my simple approach to statistical testing, accessible to professionals with little or no formal statistical training. Detailed applications of these tests are found in my recent book and throughout this book. Precise references to these documents are provided as needed, in this article.

1. General Methodology

Despite my strong background in statistical science, over the years, I moved away from relying too much on traditional statistical tests and statistical inference. I am not the only one: these tests have been abused and misused, see for instance section 3 in chapter 28, on *p*-hacking. Instead, I favored a methodology of my own, mostly empirical, based on simulations, data- rather than model-driven. It is essentially a non-parametric approach. It has the advantage of being far easier to use, implement, understand, and interpret, especially to the non-initiated. It was initially designed to be integrated in black-box, automated decision systems. Here I share some of these tests, and many can be implemented easily in Excel. Also keep in mind that the methodology presented here works with data sets that have at least a few thousand observations. The bigger the better.

The concept

I illustrate the concept on a simple problem, but it generalizes easily to any test. Here you want to test whether a univariate data set consists of numerical values (observations) that follow a normal distribution, or not. In order to do so, in a nutshell, you can proceed as follows:

- Normalize your data, so that the mean is zero and variance is equal to one.
- Simulate 10 samples (of same size as your data set) from a normal distribution of mean zero and variance one. The easiest way to do this, in Excel, might be to simulate 25 uniform deviates with the function RAND, then average and normalize, for each normal deviate being created. There are more efficient ways to do it though, see here.
- Compute the percentile distribution for your normalized data, as well as for the 10 simulated samples that you created. Easy to do in Excel, see section 2.4.
- Look at how much variance there is between the percentiles distributions computed on the 10 simulated data sets. This will give you an idea of what the natural or internal variance is.
- Compare the percentile distribution computed on your real data, with those from the simulated data. Does it look like the curve is similar to those produced with the simulated data? Or is there some kind of departure? Maybe it clearly grows more slowly initially, then catches up later, compared to the 10 curves resulting from simulation?

Ideally, you would want to have more than one real data set, to compare variations between real samples, with variations between simulated samples, and then cross-differences between real data and simulated samples. If your data set is large enough (say 3,000 observations) one way to achieve this is to split your data set into three subsets.

Now that you have an idea of the principles, we can dive in the details. The above test is one of those described in more detail in the next section, with Excel spreadsheets to illustrate the computations.

2. Off-the-beaten-path Statistical Tests

Below is our selection of unusual statistical tests, as well as some well-known tests presented in an non-standard (yet simpler) way.

2.1. Testing for symmetry

This test is used to check if the underlying distribution of your data has the same shape (mirrored) both on the left side and the right side of the median. It can be performed as follows.

One can compare R(x) = | 2 Median - $P_{.x}$ - $P_{.1-x}$ | with that of a symmetric distribution, for various values of x between 0 and 0.5, to check if a distribution is symmetric around the median. The theoretical value of R(x) is zero regardless of x, if your empirical distribution is symmetric. Here $P_{.x}$ represents the x-th percentile. Other tests for symmetry can be found here. See illustration in chapter 10.

2.2. Testing for un-imodality and other peculiarities

To test if a distribution is unimodal, several tests have been devised: the bandwidth test, the dip test, the excess mass test, the MAP test, the mode existence test, the run test, the span test, and the saddle test. The dip test is available in R. Read more here. Some of these tests, in case of multimodality, can tell you how many modes (or clusters) are in your data sets.

Other potential tests could be used, for instance to check if your data

- Has an unbounded support domain (values can be arbitrarily large in absolute value given a large enough sample size),
- If its support domain has some gaps (no value can exist in some particular sub-interval),
- If its empirical density function (histogram) is bounded (an example of unbounded density is $f(x) = 0.25 / |x|^{1/2}$ with x in [-1, 1])
- Or test for infinite mean or infinite variance

2.3. Testing whether or not there is some structure in your data

I investigated a metric that measures the presence or absence of a structure or pattern in a data set. The purpose is to measure the strength of the association between two variables, and generalizes the correlation coefficient in a few ways. In particular, it applies to non-numeric data, for instance a list of pairs of keywords, with a number attached to each pair, measuring how close to each other the two keywords are. You would assume that if there is no pattern, these distance distributions (for successive values of the sample size) would have some kind of behavior uniquely characterizing the absence of structure, behavior that can be identified via simulations. Any deviation from this behavior would indicate the presence of a structure. See here for more details.

2.4. Testing for normality, with Excel

Traditional tests exist, for instance Chi-square or Kolmogorov-Smirnov. This also works for any distribution, not just the normal (Gaussian) one. And you can use it to compare too sets of data, or two-subsets corresponding to two different time periods, to check whether they have the same distribution or not, regardless of what that distribution is. Instead of comparing empirical PDF's (probability distribution function) as in Kolmogorov-Smirnow, I use empirical percentiles (the inverse of the PDF), which are very easy to compute in Excel. See illustration (with Excel spreadsheet) in this article. I call it the *percentile test*. I typically use it after normalizing the data, so that the median value is zero.

Among other things, I have used the percentile test to solve stochastic integral equations, that is, to find the exact equilibrium distribution attached to some chaotic dynamical systems. See my previous book, page 18 (download the spreadsheet listed below the chart on page 18) and page 74.

Note: A curious normality test consists in splitting your data Z in two subsets X and Y of same size, and testing whether $(X + Y)/2^{1/2}$ has the same distribution as Z. Explanations are provided in chapter 10, and it works as long as the underlying theoretical variance is not infinite.

2.5. Tests for time series

Many assumptions could be tested, when dealing with time series observations. Sometimes, it is useful to first normalize the data by removing the trend, periodicity, outliers, and some noise. You could test if the data exhibits change points, that is, a sudden and long-term increase or decrease in observed values, usually the result of some event that took place at some point in time; see here for illustration. Or whether the change is more subtle, for instance there is no discontinuity, but the slope (trend) suddenly changes at one point. Or test whether some auto-correlations (lag-1, lag-2, and so on) are present. You can even compare the whole correlation structures of two paired time series, to check if they come from the same statistical model. Or you can perform model fitting: for instance, you suspect that your data follows an ARIMA time series model; then

- You estimate the coefficients of that tentative model,
- Then simulate values from the exact same model with same coefficients (it is much better to simulate several instances of that model to get an idea of what natural variations between same-model time series should be),
- Then test - by comparing the correlation structure in the observed and simulated data - whether the model is a good fit,
- Then try again with a different model to see if you can get a better fit.

In my case, I used some home-made tests to check whether a time series exhibits some sort of periodicity, and, as a result, I found that pseudo-random generators available in some programming languages, have a very short period, making them unfit for industrial applications. See my previous book, page 33.

2.6 Gap test, with Excel

Along with the percentile test described in section 2.4, this is one of my favorite tests to detect patterns. It is best illustrated in chapter 9. In essence, the gap test consists of measuring the largest gap with no observation, in a set of ordered values (observations). That is, the largest interval with no data point. It generalizes to higher dimensions, where the gap can be a square or circle with no data point in it. The exact distribution of the gap area or length, assuming data points are uniformly distributed, is known. If the data points take on integer values only, the distribution is a geometric one, readily available in Excel. More on this in my previous book, page 84. The test can also be used for outlier detection: a point too far away from its nearest neighbor could be an outlier.

2.7. Sparsity test

Is your data voluminous but sparse, a bit like the night sky where trillions of stars occupy a tiny portion of the sky? Or is it full of holes of moderate sizes, like Gruyere cheese? We tested this assumption in a setting that is similar to fractional factorial tables. You can check it out in my previous book, page 71.

Along the same topic, are apparent patterns real, or an illusion? For instance, in the night sky, many stars seem to be very close to each other despite the vast emptiness of the universe. Are there too many of them (called twin points) to just be a coincidence? An answer to this question, based on a statistical test, is provided here. See also a related problem about Mars craters, here.

2.8. Elbow test

The elbow test is traditionally used as a rule-of-thumb to detect the number of clusters when implementing a clustering algorithm, see section 3 in chapter 25 for illustration. I also used it to determine how many digits are accurately computed, when using high precision libraries available in some programming language. The answer was far below what is advertised in the manuals, especially when working with a mathematically ill-conditioned problem that requires an unstable iterative algorithm for computations, as in some chaotic dynamical systems. See my book, page 48.

2.9. Testing for accelerating growth

This could be used, for instance to check if glaciers are melting down at an accelerating pace. It is based on the distribution of records, and in particular, the arrival times of these records. Again simulations can be performed for this test. It is illustrated in section 2 in chapter 28, focusing on the distribution of arrival times of extreme events: the exact distribution, in the absence of growth, does not depend on the distribution of the observations (neither observed nor extreme values) making it a pretty generic non parametric test.

2.10. Run test

I used the run test in the context of stock trading, to assess how likely a *run* (say, 6 successive days with stock prices going up) is followed by a reversal, trying to find patterns to increase gains. The same can apply to sport bets. In general, run tests can be used in situations in which the underlying process behaves like a Markov chain. It helps you assess the probability of getting a + or - after any sequence of ups and downs, such as ++-+---+-+++. This test has also been used (among many other tests) to check if the distribution of the digits of some number (say π in base 2) appears to be uniform and without auto-correlations. Note that in the case of a random walk, for instance when throwing a dice, even after an extremely improbable run of 1,000 heads, the chance of obtaining an head next time is still 50%. The same seems to be true with the digits of π in base 2: after any sequence of 1,000 consecutive digits all equal to 1, the chance that the next digit is 1, is also 50%. This is indeed true regardless of the combination of 0's and 1's in the previous 1,000 digits. So the run test can be used to measure departures from randomness.

15. Modern Resampling Techniques

This crash course features a new fundamental statistics theorem -- even more important than the central limit theorem -- and a new set of statistical rules and recipes. We discuss concepts related to determining the optimum sample size, the optimum *k* in *k*-fold cross-validation, bootstrapping, new re-sampling techniques, simulations, tests of hypotheses, confidence intervals, and statistical inference using a unified, robust, simple approach with easy formulas, efficient algorithms and illustration on complex data.

Little statistical knowledge is required to understand and apply the methodology described here, yet it is more advanced, more general, and more applied than standard literature on the subject. The intended audience is beginners as well as professionals in any field faced with data challenges on a daily basis. This chapter presents statistical science in a different light, hopefully in a style more accessible, intuitive, and exciting than standard textbooks, and in a compact format yet covering a large chunk of the traditional statistical curriculum and beyond.

In particular, the concept of *p*-value is not explicitly included in this tutorial. Instead, following the new trend after the recent *p*-value debacle (addressed here by the president of the American Statistical Association), it is replaced with a range of values computed on multiple sub-samples.

Our algorithms are suitable for inclusion in black-box systems, batch processing, and automated data science. Our technology is data-driven and model-free. Finally, our approach to this problem shows the contrast between the data science unified, bottom-up, and computationally-driven perspective, and the traditional top-down statistical analysis (see here) consisting of a collection of disparate results that emphasizes the theory.

Contents

1. Re-sampling and Statistical Inference

We are dealing with a set of *N* (possibly multivariate) observations, called *population*. We want to split it into *M* subsets called *sub-samples*, each with *n* observations. In each sub-sample, observations may or may not be duplicated. The total number of data points in all the sub-samples is *nM*. Usually, *nM* is less than or equal to *N*, however here, we allow *nM* to be of any size, even larger than *N*. The purpose is to study the empirical distribution of some statistical quantities of interest, such as mean, variance, percentiles, correlations, mean squared error, and so on. In particular, we want to determine how large the sample size *N* must be, in order to achieve a pre-specified level of accuracy, typically measured by the width of some confidence intervals.

In each sub-sample, observations are drawn from the population, either with or without replacement. Whenever possible, drawing without replacement is the preferred method as it leads to maximum variance reduction, with no duplicated observations. Interesting cases include:

- **Leaving-one-out method**: Each sub-sample consists of *N*-1 distinct observations. We just remove one observation in turn from the population, to create each sub-sample.
- **Adding-one-over method**: This is the dual version of the leaving-one-out method, but it is never mentioned in the literature. Each sub-sample consists of *N*+1 observations, with *N* distinct observations. One observation (different for each sub-sample) is duplicated in each sub-sample.
- **Bootstrapping**: Each sub-sample consists of *N* observations, drawn with replacement from the original population. So the proportion of duplicated observations in each sub-sample is high. The expected number of distinct observations in each sub-sample is of the order $(1 - e^{-1})$ *N*.
- ***K*-fold cross-validation**: In this machine learning technique, the original data set is split into *K* subsets, with one of them used for validation, and *K*-1 used for training. Typically, sampling is done without replacement, and the sub-samples form a partition of the original data set.

If you want to measure some quantity *T*, say the mean value, you can do it using the entire population with *N* observations, or using the *M* sub-samples. The formula below is fundamental to solve most statistical inference problems in this context.

1.1. Main Result

If *V* denotes the variance of some estimated quantity *T* using the entire population, and *W* the variance using the *M* sub-samples, then:

$$\mathrm{V}[T] = \frac{\sigma^2}{N}, \quad \mathrm{W}[T] = \frac{\sigma^2}{n^2M^2} \cdot \Big(\sum_{k=1}^{N} w_k^2\Big),$$

$$F = \sqrt{W(T)/V(T)} = \frac{1}{nM} \cdot \Big(N \cdot \sum_{k=1}^{N} w_k^2\Big)^{1/2},$$

$$\sum_{k=1}^{N} w_k = nM.$$

Here σ is a positive constant, and the weight w_k represents the multiplicity of the *k*th observation across the *M* sub-samples. It is assumed that the *N* observations are independently and identically distributed, and that *T* can be computed as a combination of sums over the observations: this is the case for the estimated mean, variance, correlation, and many other estimators. Even when these assumptions are violated, the above results can still be used as a rule of thumb, thanks to the central limit theorem. An example is discussed in section 3.

We are mostly interested in the ratio *F*. In particular, if sampling is done without replacement, then $F = N^{1/2}/(nM))$. The quantity *F* represents the ratio of the confidence interval widths, when comparing the re-sampled estimate (numerator) versus the population estimate (denominator). Thus ***F* is a precision indicator used to determine ideal sample sizes**. The smaller *F*, the better. Note that *F* is always larger or equal to 1. Also, the ratio of two *F*'s associated with two different re-sampling schemes can be used to determine which one is the most efficient.

1.2. Sampling with or without Replacement

We have two cases:

- **Without replacement**. There is no duplicate observation in the sub-samples: in this case, all the weights are equal to 0 or 1, and *nM* is smaller or equal to *N*. In the cross-validation example, *nM* = *N* and all weights are equal to 1.
- **With replacement**. There may be duplicates: in this case, some weights are higher than 1, penalizing the variance *W*(*T*).

In both cases, the weights can be explicitly and efficiently computed at once for multiple values of *n*, using the algorithm in section 2. Thus, it is easy to compute *F*. In addition, we have the following results:

If sampling without replacement, then

$$\sum_{k=1}^{N} w_k^2 = \sum_{k=1}^{N} w_k = nM.$$

If sampling with replacement, on average we have:

$$\sum_{k=1}^{N} w_k^2 = nM \cdot \left(1 + \frac{nM}{N}\right).$$

Also, in that case, the expected number of distinct (non-duplicate) observations across the *M* sub-samples, is equal to

$$N \cdot \left[1 - \left(1 - \frac{1}{N}\right)^{nM}\right].$$

These results are easy to prove. The proof is left as an exercise, see also here.

1.3. Illustration

Here we illustrate the computation with the following simple example, with $N = 5$, $M = 2$, and $n = 3$:

- Population = (1, 2, 3, 4, 5)
- Sample 1 = (2, 4, 4)
- Sample 2 = (2, 4, 5)

In this case, $w_1 = 0$, $w_2 = 2$, $w_3 = 0$, $w_4 = 3$, $w_5 = 1$.

1.4. Optimum Sample Size

The *F*, *V* and *W* statistics can be used to determine the minimum sample size needed to achieve the desired level of accuracy for the estimator *T*. The width of your confidence interval being proportional to $N^{-1/2}$, by multiplying *N* by a factor 4, you increase accuracy by a factor 2.

Another option to determine the sample size, especially when some of the assumptions are violated (the fact that the observations must be identically and identically distributed) consists in extrapolating the variance *V* or *W* as *N* increases, to find when *N* is large enough so that (say) *V* is small enough. This is discussed in section 3.

1.5. Optimum *K* in *K*-fold Cross-Validation

The number of sub-samples used in cross-validation is usually denoted as *K* (rather than *M*) and one of the main problems is to determine the optimum *K*. Typically, *K* is small, between 2 and 20. The statistics *T* of interest, in this case, is a goodness-of-fit metric, for instance the mean squared error for predictions computed on the control sub-sample. The model is trained on *K*-1 sub-samples, called training sets. One approach to this problem is to plot *T* as a function of *K*, and check when an increase in *K* stops producing a significant improvement (error reduction) in *T*. Then you reached the ideal *K*. This can be automated using our elbow rule algorithm (see section 3 in chapter 25.)

1.6. Confidence Intervals, Tests of Hypotheses

Confidence intervals (CI) for an estimate *T* are easy to obtain. The first step consists of computing the empirical percentiles for *T*, based on *M* sub-samples, each with *n* observations. Then, a 90% CI is defined as $[T_{.05}, T_{.95}]$ with

- $T_{.05}$ being the 5th percentile for *T*, computed across the *M* sub-samples. If $M = 100$, then $T_{.05}$ is the fifth lowest value of *T* among the 100 computed values (one for each sub-sample.)
- $T_{.95}$ being the 95th percentile for *T*, computed across the *M* sub-samples. If $M = 100$, then $T_{.95}$ is the fifth highest value of *T* among the 100 computed values (one for each sub-sample.)

To narrow the width $T_{.95} - T_{.05}$ of the 90% confidence interval, one must increase *n* and *N*. To test with a 90% confidence level whether an estimate *T* is equal to a particular, pre-specified value *t*, one has to check whether *t* is in the 90% confidence interval $[T_{.05}, T_{.95}]$. All of this is illustrated in section 3.

2. Generic, All-purposes Algorithm

In this section we share a generic algorithm that performs most of the computations described earlier.

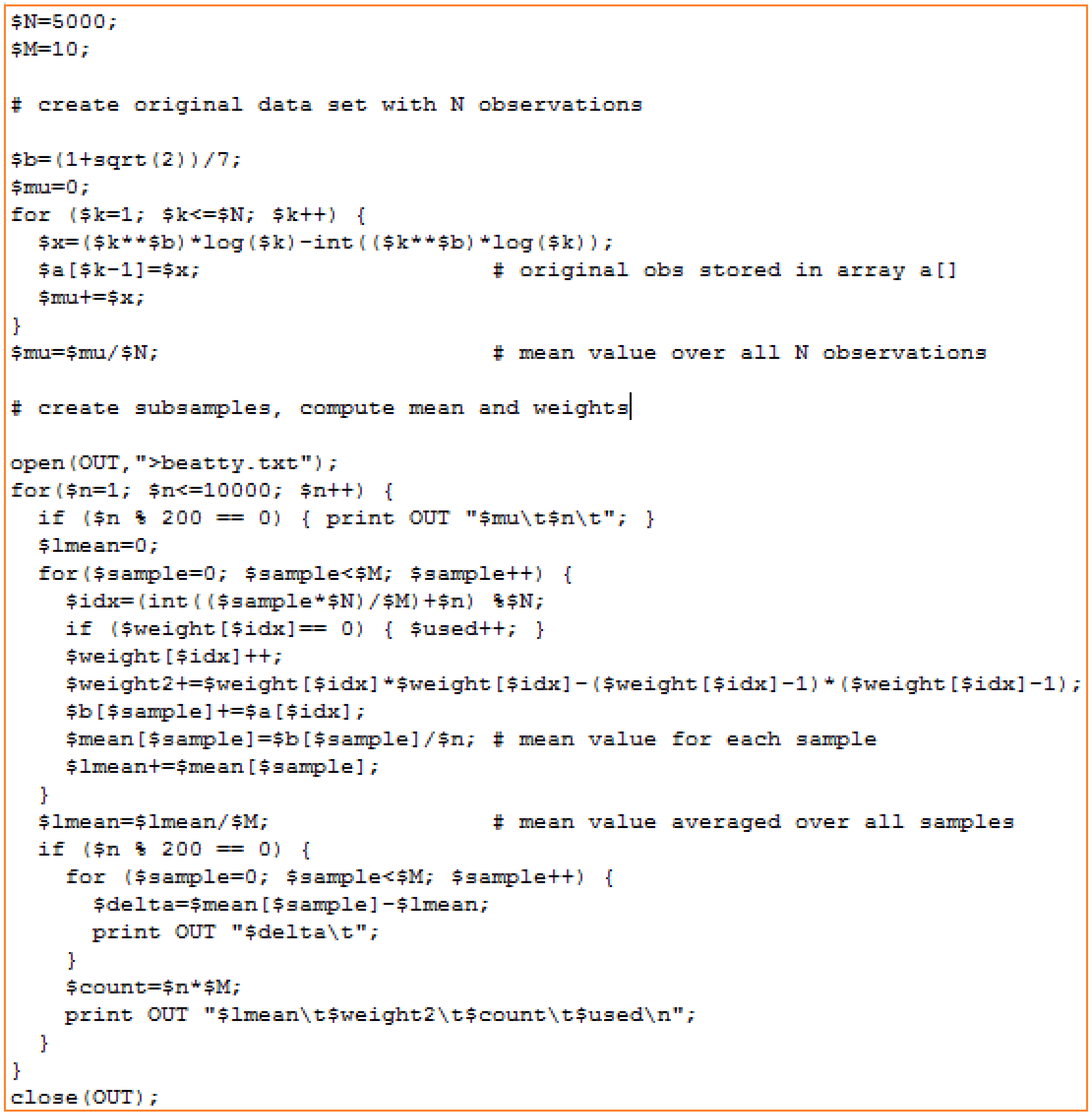
2.1. Re-sampling Algorithm with Source Code

This algorithm generates the *M* samples, each with *n* observations, based on the original data set with *N* observations. It can perform re-sampling with or without replacement, and covers all cases. In the case of re-sampling without replacement, it sequentially browses the list of *N* observations, incrementally building the sub-samples by adding one new observation each time. Thus each sub-sample consists of a block of *n* consecutive observations from the original data set. This feature is useful when processing time series data. The sub-samples contain duplicate observations if and only if *nM* is larger than *N*.

The algorithm computes the estimate *T* for each sample and each value of *n*, as well as the weights w_1, w_2, and so on, for each *n*. Here, *T* is the mean. In section 3, a different version computes the correlation for bivariate data, instead of the mean. The computations are done efficiently: the computational complexity is $O(nM)$.

Source code:

See picture below, or download the text file version here.



```
$N=5000;
$M=10;

# create original data set with N observations

$b=(1+sqrt(2))/7;
$mu=0;
for ($k=1; $k<=$N; $k++) {
  $x=($k**$b)*log($k)-int(($k**$b)*log($k));
  $a[$k-1]=$x;                          # original obs stored in array a[]
  $mu+=$x;
}
$mu=$mu/$N;                             # mean value over all N observations

# create subsamples, compute mean and weights

open(OUT,">beatty.txt");
for($n=1; $n<=10000; $n++) {
  if ($n % 200 == 0) { print OUT "$mu\t$n\t"; }
  $lmean=0;
  for($sample=0; $sample<$M; $sample++) {
    $idx=(int(($sample*$N)/$M)+$n) %$N;
    if ($weight[$idx]== 0) { $used++; }
    $weight[$idx]++;
    $weight2+=$weight[$idx]*$weight[$idx]-($weight[$idx]-1)*($weight[$idx]-1);
    $b[$sample]+=$a[$idx];
    $mean[$sample]=$b[$sample]/$n; # mean value for each sample
    $lmean+=$mean[$sample];
  }
  $lmean=$lmean/$M;                     # mean value averaged over all samples
  if ($n % 200 == 0) {
    for ($sample=0; $sample<$M; $sample++) {
      $delta=$mean[$sample]-$lmean;
      print OUT "$delta\t";
    }
    $count=$n*$M;
    print OUT "$lmean\t$weight2\t$count\t$used\n";
  }
}
close(OUT);
```

The above version performs re-sampling without replacement. For re-sampling with replacement, replace the line

`$idx=(int(($sample*$N)/$M)+$n)%$N`

by

`$idx=int($N*rand()).`

Notes:

- `$weight2` is the sum of squared weights at iteration *n*
- `$used` is the number of distinct observations in the *M* sub-samples at iteration *n*

- % is the symbol representing the modulo operator

In section 2.2 we discuss sampling without replacement, picking up observations randomly rather than sequentially.

2.2. Alternative Algorithm

If the *N* observations in the original population are somewhat clustered, it is better not to create sub-samples consisting of blocks of successive observations, when sampling without replacement. In this case, one can proceed iteratively as follows:

- **Initialization**: Iteration *k* = 1. Select the first observation randomly among the *N* observations in the original population. Assign it to sub-sample number 1.
- **Loop**: At iteration *k*+1, randomly and repeatedly pick up an observation among the *N* observations in the original population, until you find one that has not been picked up already in a previous iteration. Assign the newly found observation to segment number (*k*+1) modulo *M*.
- **Stop** when each sub-sample has *n* observations.

The number of trials required at iteration *k*, to find an observation that has not been picked up already in a previous iteration, is equal to *N* / (*N* - *k* + 1) on average. Thus the computational complexity of this procedure is

$$O\left(N \cdot \log \frac{N}{N - nM + 1}\right), \text{ with } nM \leq N.$$

2.3. Using a Good Random Number Generator

Modern programming languages and even Excel provide reliable pseudo-random number generators, capable of generating up to 10^{15} distinct values. This limit is due to machine precision, but it is more than enough for our purpose, especially since re-sampling methods are particularly useful for relatively small data sets.

However, Perl is a notable exception, capable of generating only 32,767 distinct pseudo-random numbers; see here for details. This is why we created our own generator. With our generator, the k^{th} deviate is equal to the fractional part of k^b log(*k*), with $b = (1 + 2^{1/2})/7$. These deviates are uniformly distributed on [0, 1] and exhibit no auto-correlation. Proving the random character of these deviates is an interesting and difficult number theory problem, beyond the scope of this tutorial.

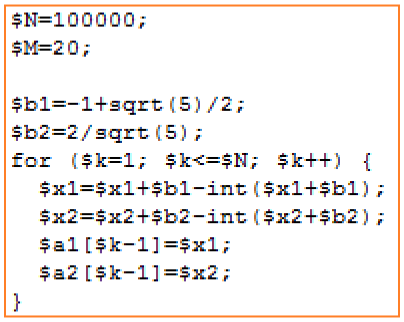
We also used our generator to create the original data set with *N* observations, in section 2.1.

3. Applications

In this section, we use a more complex data set to illustrate the concepts discussed earlier, to obtain new theoretical results, and to create new statistical recipes. The framework described here could be called *statistics 2.0*. One amazing feature discussed in section 3.4 is a recipe to design an estimator more accurate than the best possible estimator available for a fixed value of *N*, without increasing *N* (the number of observations) in essence seemingly working with much more information than the data set actually contains, as if *N* was larger than it actually is.

3.1. A Challenging Data Set

The data set consists of *N* bi-variate observations (x_k, y_k) with k = 1, 2, ..., n. It is built as follows: x_k is the fractional part of b_1k, and y_k is the fractional part of b_2k, with b_1 = - 1 + $5^{1/2}$/2 and b_2 = 2/$5^{1/2}$. The data (the two variables) is stored in two arrays `a1[]` and `a2[]`, using the code below:



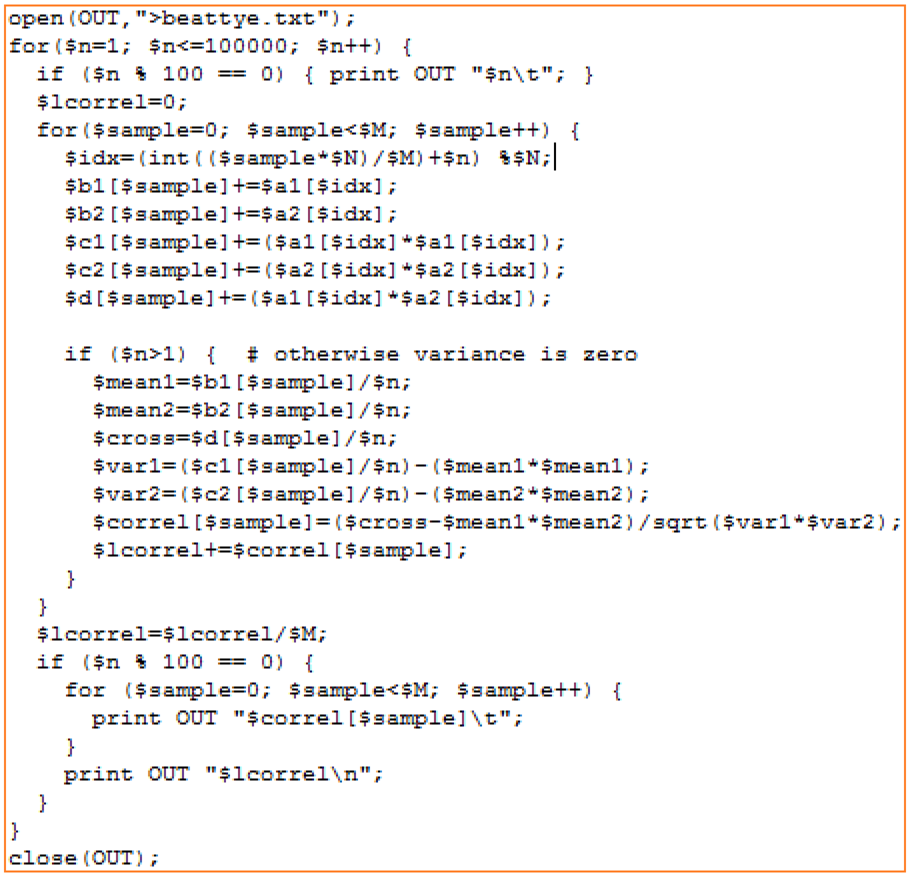
```
$N=100000;
$M=20;

$b1=-1+sqrt(5)/2;
$b2=2/sqrt(5);
for ($k=1; $k<=$N; $k++) {
  $x1=$x1+$b1-int($x1+$b1);
  $x2=$x2+$b2-int($x2+$b2);
  $a1[$k-1]=$x1;
  $a2[$k-1]=$x2;
}
```

The data set is a realization of a bi-variate perfect process with *N* = 100,000 points. These processes are studied in chapter 13 and appendix B. In particular, each of the two variables exhibits strong, long-range (indeed, infinite range) auto-correlations. The exact values of these auto-correlations are known, making this data set a good candidate to benchmark statistical tests. The cross-correlation *T* between the two variables is also known and equal to

$$T = -3 + 12 \cdot \lim_{T\to\infty} \frac{1}{T} \int_0^T \{b_1 x\}\{b_2 x\} dx.$$

In particular, with the values of b_1 and b_2 chosen here, the cross-correlation, as *N* tends to infinity, is equal to *T* = 1/20, see here. Pretty close to zero, but distinct from zero. The brackets in the above formula represent the fractional part function.

We make statistical inference about the cross-correlation *T*, using *M* = 20 sub-samples, each containing up to *n* = 5,000 points. Sampling is done without replacement, and *nM* = *N*. So there is no duplicate observation in the sub-samples. The source code, adapted from section 2.1, becomes



```
open(OUT,">beattye.txt");
for($n=1; $n<=100000; $n++) {
  if ($n % 100 == 0) { print OUT "$n\t"; }
  $lcorrel=0;
  for($sample=0; $sample<$M; $sample++) {
    $idx=(int(($sample*$N)/$M)+$n) %$N;
    $b1[$sample]+=$a1[$idx];
    $b2[$sample]+=$a2[$idx];
    $c1[$sample]+=($a1[$idx]*$a1[$idx]);
    $c2[$sample]+=($a2[$idx]*$a2[$idx]);
    $d[$sample]+=($a1[$idx]*$a2[$idx]);

    if ($n>1) {  # otherwise variance is zero
      $mean1=$b1[$sample]/$n;
      $mean2=$b2[$sample]/$n;
      $cross=$d[$sample]/$n;
      $var1=($c1[$sample]/$n)-($mean1*$mean1);
      $var2=($c2[$sample]/$n)-($mean2*$mean2);
      $correl[$sample]=($cross-$mean1*$mean2)/sqrt($var1*$var2);
      $lcorrel+=$correl[$sample];
    }
  }
  $lcorrel=$lcorrel/$M;
  if ($n % 100 == 0) {
    for ($sample=0; $sample<$M; $sample++) {
      print OUT "$correl[$sample]\t";
    }
    print OUT "$lcorrel\n";
  }
}
close(OUT);
```

The source code is available in text format, here.

3.2. Results and Excel Spreadsheet

We computed the cross-correlation *T* for all sub-sample sizes *n* between *n* = 2 and *n* = 5,000, for the *M* = 20 sub-samples, using the code in section 3.1. We then computed (in Excel), for each *n*, the percentiles $T_{.05}$ and $T_{.95}$, as well as the width $L = T_{.95} - T_{.05}$ of the confidence intervals, across the *M* sub-samples. The results are available in this spreadsheet. You can change the percentile thresholds (0.05 and 0.95) in the spreadsheet to interactively visualize the impact on the charts. These thresholds are stored in cells AC2 and AD2. The two charts of interest are shown below.

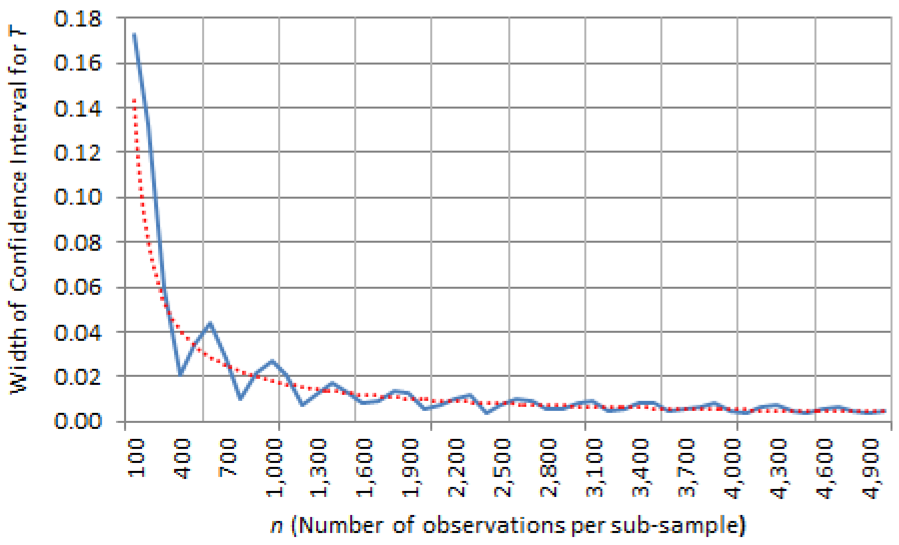


Figure 1: *Width L of the confidence interval for T, as a function of n (the dotted line is an approximation)*

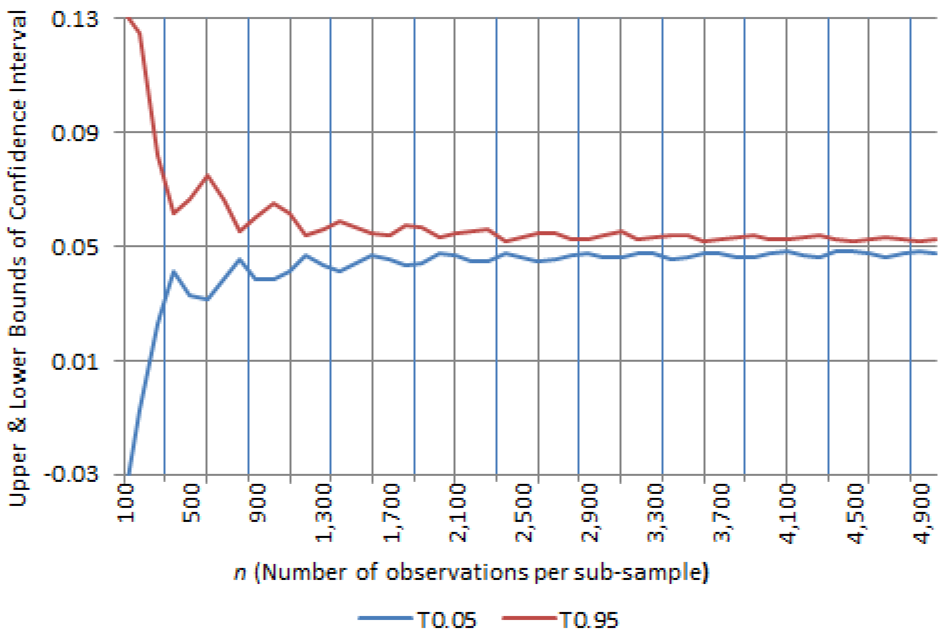


Figure 2: *Upper and lower bounds of the confidence interval for T, as a function of n*

The true, theoretical value of *T* (when *N* is infinite) is *T* = 1/20 = 0.05. The charts speak for themselves: they provide the confidence intervals and suggest that indeed, based on our computations, a value of 0.05 for *T* is highly plausible, and that *T* is clearly not equal to 0 (it would be zero if the two irrational bases b_1 and b_2 used to build our data set, were not related.) But there is much more to that, as we shall see in section 3.3 and 3.4.

Note that we did not use any statistical theory to arrive to our conclusions, not even the concept of random variable, statistical distribution, or the central limit theorem.

3.3. A New Fundamental Statistics Theorem

In the example in section 3.2, the assumptions of the central limit theorem are severely violated, in particular, the observations are not independent at all. In other cases, observations may have different variances and are not identically distributed. Yet, the width L of the confidence interval ($L = T_{.95} - T_{.05}$, using $M = 10$ or $M = 20$) is very well approximated by a power function of n as illustrated in Figure 1. Actually, this is also true with $L = T_{.90} - T_{.10}$, and indeed, with any percentile thresholds (that is, with any confidence level, to use the standard statistical terminology.)

In our example in Figure 1, the relationship is $L = 8.781 / n^{0.894}$ which becomes more and more accurate as n increases (see the dotted line.) Generally speaking, the relationship is $L = A/n^B$, where A and B are two constants that depend on your data set. This leads to the following theorem:

Theorem: *The width L of any confidence interval is asymptotically equal (as n tends to infinity) to a power function of n, namely $L = A / n^B$ where A and B are two positive constants depending on the data set, and n is the sample size. We discuss here the conditions required for the theorem to be valid.*

The exponent B bears some resemblance with the Hurst exponent in time series, see here and here. The standard case, when the data is well behaved and satisfies the assumptions of the central limit theorem (CLT), yields $B = 1/2$. The constant A is linked to the estimated variance attached to a single observation, and is further discussed in chapter 16. Any departure from $B = 1/2$ indicates that the data has some patterns, and that the CLT assumptions are violated. The same is true with the Hurst exponent. In some sense, the above theorem is a generalization of the CLT. Other generalizations exist, see for instance here, but the one featured in our theorem is of an entirely different nature. It can be used to very accurately determine the value of n (and thus N) needed to achieve a specific level of accuracy for an estimator T, even when the CLT assumptions are severely violated.

In our example, we picked up $T_{.05}$ and $T_{.95}$, as opposed to (say) $T_{.25}$ and $T_{.75}$, because the thresholds 0.05 and 0.95 provide a better fit with the power function, and especially, a more symmetrical confidence interval. Finally, L is the difference between two values of the empirical percentile distribution for T. This distribution is known to converge to that of a normal distribution under certain conditions, and this fact can be used if you are interested in digging into the mathematical details. Note that $T_{.95}$ and $T_{.05}$ are **not** independent random variables.

You can use our theorem on smaller data sets, for instance with $M = 10$ and $n = 2{,}000$, that is, $N = 20{,}000$ observations.

3.3. Some Statistical Magic

With N between 98,000 and 100,000 observations, the value for the cross-correlation estimator T discussed in section 3.1 and 3.2 is quite stable, and oscillates between 0.049962093 and 0.050143847, with an average of 0.050039804. The exact value (when N is infinite) is 0.050000000. There is an intuitive way to get a much more precise estimate without increasing the sample size, and this applies to any data set and any estimator.

Let us look at the value computed on each of the 20 sub-samples, when n is between 2,500 and 5,000, using increments of 1,000 in n. In particular, let us focus on the 5% and 95% percentiles ($T_{.05}$ and $T_{.95}$). The median value computed on these 52 percentile data points is 0.049992500.

Note that in practice, the sample size N is determined in advance. You have to assume that the best possible estimate is obtained by taking all the 100,000 observations into account. In our case, this yields -- by pure chance -- an unexpectedly very good value of 0.050011877, more accurate than (say) with N = 99,999 or 99,998, but less accurate than with N = 98,900 (with N = 98,900 the estimated value is 0.050002867 and it is one of the very best that you can get from the data set.) Of course, in practice, on a real data set, there is no way to known that N = 98,900 yields a more accurate value than N = 100,000, and you won't know if N = 100,000 works better or not than N = 99,999. By contrast, the technique described in the previous paragraph is replicable, and also provides an incredibly accurate value.

Let us summarize our findings:

- Without using our trick, expect to get an estimated value of 0.050039804
- With our trick, the estimated value is 0.049992500
- The true value is 0.050000000

The error reduction factor with our trick is | 0.050039804 - 0.050000000 | / | 0.049992500 - 0.050000000 | = 5.3. If you were to get that kind of improvement simply by increasing the sample size, you would need a sample size about 5.3 x 5.3 = 28 times bigger. Our trick essentially gives you one extra digit of accuracy, without increasing N. For a related method that accomplishes similar results, download this PDF presentation (originally posted here) by Nathaniel E. Helwig, entitled *Bootstrap Confidence Intervals*, and look for second-order accurate intervals starting at slide 29.

How does this work?

We have simply smoothed out little variations in the estimated value around N = 100,000, using an high-pass filter to sharpen the signal. This is similar to using an high-pass filter in image processing to remove noise and increase sharpness. This type of filter (in image processing) also uses medians rather than averages. Averages are actually used to do the opposite effect: blur the image, and the filter is then called a low-pass filter. Also, because at *N* around 100,000, the estimated values are mostly above the exact value, using medians that include $T_{.05}$ allows you to correct for the tiny bias. This would also work if the opposite was true, that is, if the values were mostly below the exact value, thanks to using *T*.95 as well. And this feature (the tiny bias) is present in any data set, when looking at short windows such as *N* between 98,000 and 100,000.

Does this contradict entropy principles?

It does not contradict entropy principles, not more than accuracy boosts obtained by removing outliers (thus reducing the sample size) that do better than increasing the sample size.

The basic principle is that the insights you get from a data set are based on the amount of information that it contains. If you use all the information in your data set (and the standard estimator based on the *N* observations does that) then there is no way you can get anything better unless you increase the sample size. This is true in theory, but not always in practice: removing errors is a counter-example, with an error-free data set seemingly containing more information than if errors were added to it.

However, in compliance with the entropy principle, over sub-sampling (creating sub-samples with duplicated observations, resulting in *nM* larger than *N*) in hopes of getting more accurate estimators, would not solve the problem. We have established this fact in section 1.1, proving that *F* is always above 1, or equal to 1 if and only if all the information in the data set is used. Whether it is used only once or duplicated, does not change the fact that *F* cannot be lower than 1. Thus the confidence level cannot be improved that way.

4. Conclusions

It is sometimes said that data science needs statistics to make things work. Here it is the other way around, with statistics benefiting from mathematical discoveries arising from applied data science research, to improve existing statistical methods.

In this article, we discussed a new way to process, analyze and extrapolate data sets, leading to a new fundamental statistical theorem, and a way to find more information in a data set, than what traditional entropy and statistical theory suggest. It allows you to increase the accuracy of statistical estimators without increasing the sample size. The boost in accuracy is equivalent to increasing the sample size by a factor 25. The magic in this technique is not more spectacular than the magic used to enhance blurred images and make them look perfect. Indeed, these two extrapolation techniques are closely related.

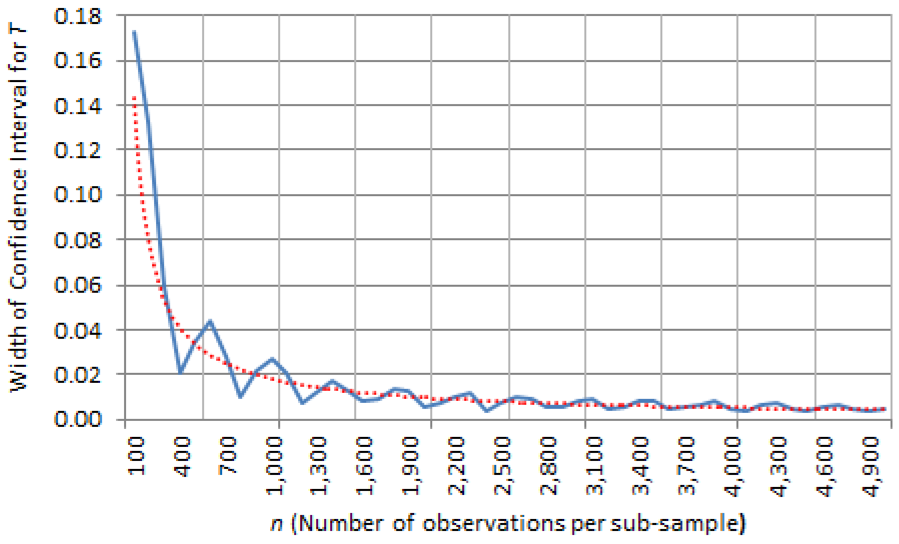
Under the umbrella of re-sampling, many statistical problems are solved in a simple way, ranging from optimizing cross-validation experiments to designing sound tests of hypotheses when traditional assumptions imposed on the data set are severely violated.

All of this is discussed without using even basic statistical concepts such as random variable, *p*-value, or statistical distribution, making the material not only accessible to the layman, but also easy to integrate in black-box machine learning systems.

16. Model-free Confidence Intervals

We propose a simple model-free solution to compute any confidence interval and to extrapolate these intervals beyond the observations available in your data set. In addition we propose a mechanism to sharpen the confidence intervals, to reduce their width by an order of magnitude. The methodology works with any estimator (mean, median, variance, quantile, correlation and so on) even when the data set violates the classical requirements necessary to make traditional statistical techniques work. In particular, our method also applies to observations that are auto-correlated, non-identically distributed, non-normal, and even non stationary.

No statistical knowledge is required to understand, implement, and test our algorithm, nor to interpret the results. Its robustness makes it suitable for black-box, automated machine learning technology. It will appeal to anyone dealing with data on a regular basis, such as data scientists, statisticians, software engineers, economists, quants, physicists, biologists, psychologists, system and business analysts, and industrial engineers.



Power curve fitting: see here

In particular, we provide a confidence interval (CI) for the width of confidence intervals without using Bayesian statistics. The width is modeled as $L = A/n^B$ and we compute, using Excel alone, a 95% CI for B in the classic case where $B = 1/2$. We also exhibit an artificial data set where $L = (\log n)^{-\pi}$. Here n is the sample size.

Despite the apparent simplicity of our approach, we are dealing here with martingales. But you don't need to know what a martingale is to understand the concepts and use our methodology.

Contents

1. Principle

We have tested our methodology in cases that are challenging when using traditional methods, such as a non-zero correlation coefficient for non-normal bi-variate data. Our technique is based on re-sampling and on the following, new fundamental theorem:

***Theorem**: The width L of any confidence interval is asymptotically equal (as n tends to infinity) to a power function of n, namely $L = A / n^B$ where A and B are two positive constants depending on the data set, and n is the sample size. The requirements for the theorem to be valid are discussed in section 6.*

The standard (textbook) case is when *B* = 1/2. Typically, *B* is a function of the intrinsic, underlying variance attached to each observation, in your data set. Any value of *B* larger than 1/2 results in confidence intervals converging faster than what traditional techniques are able to offer. In particular, it is possible to obtain *B* = 1. Departure from *B* = 1/2 can be caused by unusual data (see sections 2.3, 2.4, and 6.3) or by a-typical estimators (see section 2.1 and 2.2.) Indeed, testing whether *B* = 1/2 can be useful to check if your data has hidden patterns such as uneven variances.

Our new technique is described in details (with source code, spreadsheet and illustrations) in chapter 15. When reading that chapter, you may skip section, 1, and focus on section 2, and especially section 3, where all the results are presented.

2. Examples

We provide here a few examples where the exponent *B* is different from 1/2.

2.1. Estimator used in nearest neighbors clustering

An example of a non-standard case is the following. In the context of supervised classification, one sometimes uses a function of the distances between a point *x* outside of the training set, and its *n* nearest neighbors in the training set, to decide which cluster *x* should be assigned to. This function uses decaying weights, with the highest weights attached the closest neighbors within a same cluster. Depending on how fast these *n* weights decay, the resulting cluster density estimators measured at location *x*, may have an exponent *B* different from 1/2. Also, if the confidence intervals attached to two or more clusters overlap, it means that *x* could belong to any of these clusters.

2.2. Weighted averages when dealing with outliers

One way to eliminate or reduce the impact of outliers, when estimating the mean *T*, is to use weighted averages with positive weights, the weight attached to each observation being a decreasing function of the distance between the median and the observation in question. It is defined as follows:

$$T_n = \frac{\sum_{k=1}^{n} w_k X_k}{\sum_{k=1}^{n} w_k}, \text{ with } w_k = k^{-b} \text{ and } 0 \leq b \leq 1.$$

We assume here that the *n* independently and identically distributed observations are ordered according to their distance to the median, with the closest one corresponding to the first term in the sum. If $b < 1/2$, then $B = 1/2$ as in the standard case. An example is when all weights are identical ($b = 0$). If *b* is in]0.5, 1[then it is easy to prove (see chapter 10) that $B = 1 - b$. The same is true for other estimators, not just the mean.

The general rule is as follows. If you use weights w_k decaying more slowly than $1/k^{1/2}$, then $B = 1/2$. If the weights decay faster than $1/k^{1/2}$ but more slowly than $1/k$, then you still end up with a power function, but *B* is strictly between 0 and 1/2. More specifically, with the notation

$$s_n = \frac{\sqrt{\sum_{k=1}^{n} w_k^2}}{\sum_{k=1}^{n} w_k},$$

the exponent *B* is equal to

$$B = -\lim_{n\to\infty} \frac{\log s_n}{\log n}.$$

In particular, s_n is minimum and equal to $1/n^{1/2}$ if all the weights w_k are identical. In that case, $B= 1/2$.

2.3. Auto-correlated time series, *U*-statistics

We provide here an example where $B = 1$. In a time series $X(1)$, $X(2)$, and so on, the most extreme case (producing the highest value for *B*) is when $X(k+1)$ depends solely on $X(k)$. This is actually the case in the example discussed in section 6.3.

For instance, in the artificial case where $X(k+1) = aX(k) + b$ with $|a| < 1$, the mean (or any other estimator) is a function of $X(1)$ and the constants a and b alone. The variance of the mean is

$$\operatorname{Var}\left[\frac{X_1 + \cdots + X_n}{n}\right] = \operatorname{Var}[X_1] \cdot \frac{1 + a + a^2 + \cdots + a^{n-1}}{n^2} \sim \frac{\operatorname{Var}[X_1]}{(1-a) \cdot n^2} \text{ as } n \to \infty,$$

and thus $B = 1$. The same upper bound $B = 1$ can be achieved with actual (non-degenerate) stationary auto-regressive time series, see the answer to a question I asked on CrossValidated.com, here.

The problem with time series is that if you re-shuffle the observations, you lose the auto-correlation structure, and thus B may revert back to $B = 1/2$. Can you find an estimator that keeps a value of B higher than 1/2 even if you reshuffle the observations? The answer is positive if you consider *U*-statistics: they provide estimators that can converge faster ($B > 1/2$) to the true value, than standard estimators. See this paper (originally posted here), especially the sentence above example 3.11 on page 3, and this article. The most well-known *U*-statistic is Gini's mean difference, defined as

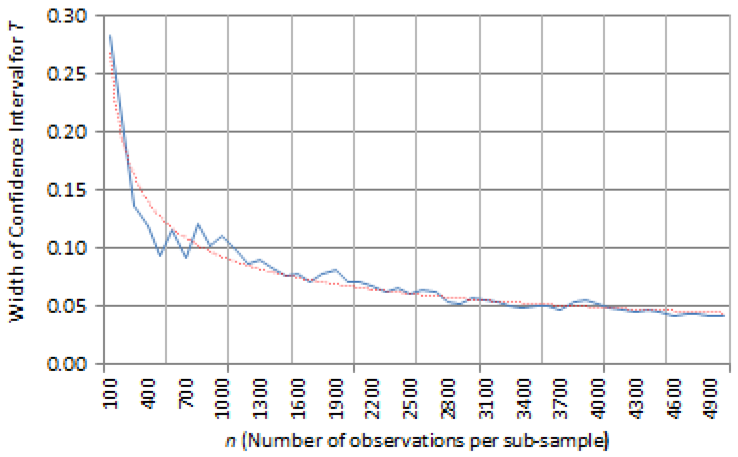
$$G_n = \left[\frac{2}{n(n-1)} \cdot \sum_{i=1}^{n} \sum_{j=i+1}^{n} |X_i - X_j|^p\right]^{1/p},$$

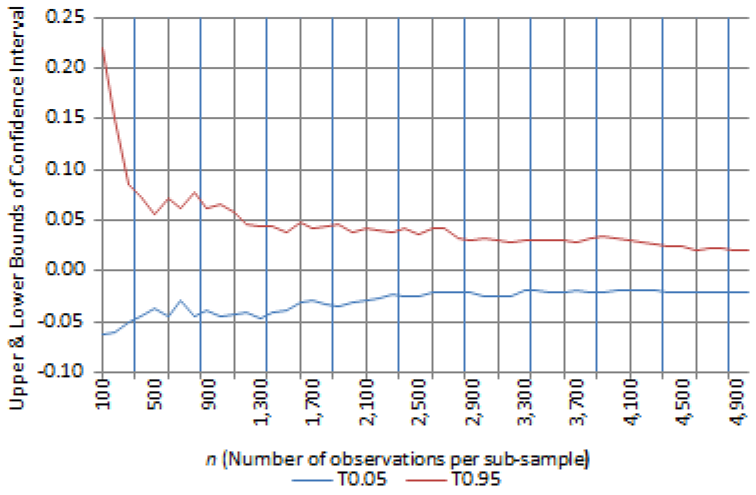
with $p = 1$. While more efficient than traditional dispersion estimators, its B exponent is also 1/2. The interesting case is when p tends to infinity: then $G(n)$ is the range (maximum minus minimum observation). If the observations are uniformly and independently distributed on [0, 1], then the range has a Beta(n - 1, 2) distribution, thus its variance is $2(n - 1) / [(n + 2)(n + 1)^2]$ and $B = 1$. It would be interesting to see what happens if the observations have a fat tail distribution instead. If the distribution is exponential, $B = 0$ (see chapter 17 for the proof.) If the distribution is Gaussian, $B = 1/2$ (see chapter 17.)

2.4. Correlation coefficient estimated via re-sampling

Below is an illustration for the correlation coefficient using a bi-variate artificial data set that simulates somewhat random observations. The illustration in this section is based on re-sampling, using the approach discussed in sections 2 and 3 in chapter 15.

The resulting value of B is 0.46. The boosting technique (to improve B) has not been used here, but it is described and illustrated in section 3.3, in the chapter in question. The power function mentioned in our above theorem, fitted to this particular data set, is represented by the red, dotted curve, in the top chart below.





The simulated data set was built as follows:

$$X_k = \sin\left(k^2 \cdot \frac{-2+\sqrt{5}}{2}\right),\ Y_k = \sin\left(\frac{2k^2}{\sqrt{23}}\right),\ k = 1, 2, \cdots.$$

More examples with more details, can be found in chapter 15. One way to get more accurate values for *A* and *B* is to re-do the same computations using 10 different re-ordering of the data set, by randomly shuffling the observations, then averaging *A* and *B* across these 10 sets: see discussion in section 5.

3. Counterexamples

The theorem is very general, and applies to most data sets. Exceptions consist of odd, artificially manufactured data sets that are not found in business applications, or non-stationary processes such as Brownian motions (see also chapter 2, here.)

For instance, the data set created in my new spreadsheet (download it here) consists of 10,000 observations: the first 5,000 are assigned to sample x[1], and the remaining ones to sample x'[2] (respectively column A and D in the spreadsheet). Observations in each sample are independently and uniformly distributed, and the correlation between the two samples is zero. Yet the confidence intervals for the mean have a width $L(n)$ of the form $A/(\log n)^B$, here with $A = 1$ and $B = \pi$: see column J. You can change the fitting curve (column K) as well as the values of the observations in x[1], and keep everything else unchanged, and get whatever curve you want for $L(n)$, even a trigonometric function. This is possible only because the data has been created to make this happen: while the first-order cross-correlation between the two samples is nil, the second-order cross-correlations are extremely high, see cell P9.

Usually, with these counterexamples, if you randomly sort the data set, re-compute the estimator and its width L on the reshuffled data for various n, the fitted curve for the width of the interval will be a totally different function. That's actually how you recognize that these data sets are artificial. However, even with real-life data, you are sometimes faced with data glitches that produce the same issues, and that are hard to detect as in my above example.

Yet there are some estimators that truly do not have a power function for the width of their confidence interval. An example is provided in my article on Poisson processes. It is about a local estimator of the intensity of a Poisson process based on the distances to the n nearest neighbors. The width L is asymptotically equivalent to $(\log n)/n^{1/2}$. See here, or download the article: this estimator is described page 118, just above Remark 1.

Our main theorem can be generalized as follows to cover even more cases, using a second order approximation:

$$L \sim A \cdot n^{-B} \cdot \Big(\log n\Big)^{C}.$$

The constant C may be positive or negative. Even then, the logarithm of the width L is *asymptotically* equivalent to a curve with only two parameters: $\log A$ and B.

4. Estimating *A*

The constant A attached to the power function (see theorem), is related to the intrinsic variance present at the individual observation level, in your data. We provide its value for common estimators (up to a factor that depends only on the confidence level), in the ideal case when observations are independently and identically distributed with an underlying normal distribution. These values can still provide a good approximation in the general case.

Estimator	A	Comment
Mean	σ	
Variance	$\sqrt{2}\cdot\sigma^2$	
Median	$\frac{1}{2f(m)}$	m is the median, f is the distribution density.
p-th quantile	$\frac{\sqrt{p(1-p)}}{f(x_p)}$	f is the distribution density, $x(p)$ is the p-th quantile.
Correlation	$1-\rho^2$	Assumes the true correlation is zero.

The formula for the median is a particular case of the p^{th} quantile with $p = 0.5$. All the values in the second column represent the unknown theoretical value of the quantities involved. In practice, these values are replaced by their estimates computed on the data set. These estimates converge to the true values, so using one or another does not matter, as far as the correctness of the table is concerned.

The exact formula for the correlation when it is not zero and the normal assumption is not satisfied, is very complicated. See for instance this article. By contrast, dealing with any correlation (or even more complicated estimators such as regression parameters) is just as easy as dealing with the mean, if you use our methodology. Even if none of the standard assumptions is satisfied.

Finally, *A*, *B* and *n* provide more useful information about your estimator, than *p*-values.

5. Estimating *B*

We denote as $L(n)$ the value of the interval width computed on a sample with *n* observations. The standard way to fit $L(n)$ with the power curve A/n^B is not very efficient, resulting in high volatility. Here we offer strategies to get much more accurate and stable values for *A* and *B*. All the illustrations in this section are based on re-sampling, using the approach discussed in sections 2 and 3 in chapter 15.

5.1. Getting more accurate values

The following strategies significantly improve the accuracy when estimating *B*:

- Use 10 different re-ordering of the data set, by randomly reshuffling the observations, then average *B* across these 10 sets
- Focus on small values of the sample size (less than n = 10,000.)
- When fitting $L(n)$ with A/n^B, do not use all the values of *n*, but only a fraction of them, that are unequally spaced: for instance n = 4, 9, 16, 25, 36, 49, 64 and so on. Cubes work even better if your samples are large enough.

- Large confidence intervals, with lower and upper bounds equal to the 2.5 and 97.5 empirical percentiles, work better than smaller ones. Also, *M* = 20 or *M* = 10 (see algorithm in section 3 in chapter 15) work better than *M* = 2 or *M* = 5. Large values of *M* (say *M* = 50) do not offer extra benefits.

We applied these strategies to compute a confidence interval for *B*, on a data set consisting of 4 non-overlapping samples (*M* = 4), each with *n* = 2,500 observations, drawn from a population of *N* = 10,000 observations. We repeated this procedure 10 times, by re-shuffling the 10,000 observations 10 times in the original data set. The value obtained for *B* is 0.484, while the theoretical value (with an infinite sample) would be 0.500. The data set, computations, and results are available in this spreadsheet (10 MB.)

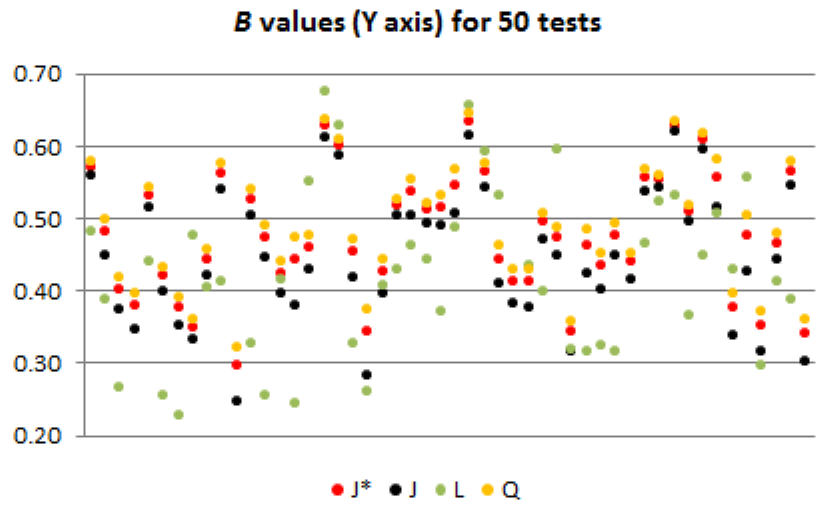
5.2. Getting even more accurate values

The estimated values for *B* are very volatile due to the fact that *L*(*n*) is computed recursively based on embedded samples of increasing sizes. Indeed, { *L*(*n*) } is a martingale. Instead of trying to fit *L*(*n*) with a power curve, you can fit the much smoother integrated *L*(*n*), denoted as *J*(*n*), with an appropriate curve, then estimate *B* using *J*(*n*) rather than *L*(*n*). It turns out that *J*(*n*) is also well approximated by a power curve with the same *B*, at least as a first order approximation. If for the theoretical (exact) value, $L(n) = A/B^n$, then we have:

$$J(n) = \frac{1}{n} \cdot \sum_{k=1}^{n} L(n) \approx \frac{A}{1-B}\left(\frac{1}{n^B} - \frac{B}{n}\right).$$

Note that *B* is in]0, 1[, and in most cases *B* = 1/2. If you ignore the term *B*/*N* in the above formula, then you also have a power curve for *J*(*n*). If you include that term, your estimation will be more accurate, but the model fitting technique (to find *B*) is a tiny bit more tricky.

Instead of using *J*(*n*), you could use the median value of *L*(*n*) computed on the first *n* observations in your sample. This median is denoted as *Q*(*n*). It provided the most accurate results for *B*, among the four methods tested.



We tested the four methods -- *L*(*n*), approximated *J*(*n*), bias-corrected *J*(*n*) denoted as *J**(*n*), and *Q*(n) -- using *M* = 2 samples each with up to *n* = 10,000 observations, reshuffling the samples 50 times to obtain the 50 x 4 values of *B* shown in the above figure. The estimated value of *B*, computed over the 50 tests using *Q*(*n*), is 0.49. The data set was designed so that the theoretical *B* should be the classic 1/2 value as *n* tends to infinity. The above figure gives you an idea of the confidence intervals for *B*. Note that the method based on *L*(*n*) is by far the worst, with bias and high volatility, including 5 outliers (very low values of *B*) not shown in the figure. All the details, with additional comments, are found in this spreadsheet.

6. Theoretical Background

We first compare the asymptotic formula for the re-scaled range, based on the Hurst exponent *H*, with our asymptotic formula for *L*(*n*), based on the *B* exponent. We then explain how the formula $L(n) = A/B^n$ can be derived under general assumptions, using some heuristics. All the illustrations in this section are based on resampling, using the approach discussed in sections 2 and 3 in chapter 15.

6.1. Connection with the re-scaled range and the Hurst exponent

We assume here that the number of samples is *M* = 2. Given a sample with *n* observations, the re-scaled range, in its simplest form, is defined as the range of the observations (the difference between the maximum and the minimum) divided by the standard deviation computed on the *n* observations, see here (or alternatively here) for details.

Also, if *M* = 2, the width of the confidence interval *L*(*n*) is proportional to $|T'(n) - T''(n)|$ where $T'(n)$ and $T''(n)$ are the values of the estimator of interest computed for each sample of size *n*. Let us denote as *p* the proportion in question, measuring the level of the confidence interval, so that

$$L(n) = p \cdot |T'(n) - T''(n)|.$$

Under some general conditions, *T*'(*n*) and *T*''(*n*) have a Gaussian distribution. Thus, *L*(*n*) has a folded normal distribution, and its expectation is

$$E[L(n)] = p \cdot \sqrt{\frac{2}{\pi}} \cdot \sqrt{\mathrm{Var}[T'(n) + T''(n)]} \sim An^{-B}.$$

Now, let's introduce the following notations:

$$Z'(k) = kT'(k) - (k-1)T'(k-1), \text{ with } T'(0) = 0,$$
$$R'(n) = \max(Z'(1), \cdots, Z'(n)) - \min(Z'(1), \cdots, Z'(n)),$$
$$V'(n) = \frac{1}{n} \cdot \sum_{k=1}^{n} (Z'_k - \bar{Z}'_n)^2, \text{ with } \bar{Z}'_n = \frac{1}{n} \cdot \sum_{k=1}^{n} Z'_k = T'(n).$$

For instance, if the estimator *T* in question is the mean, then *Z*'(*k*) is the *k*th observation in the first sample. Now we can write the well-known asymptotic expansion for the re-scaled range, as

$$E\left[R(n)/\sqrt{V'(n)}\right] \sim Cn^{H},$$

where *H* is the Hurst exponent. In the above formula, if you replace *V*'(n) by Var[*T*'(*n*)] and *R*(*n*) by a constant *R*, it becomes

$$C \cdot \sqrt{\mathrm{Var}[T'(n)]} \sim Rn^{-H}.$$

Since the two samples are independent, Var[*T*''(*n*)] ~ Var[*T*'(*n*)] and thus *H* = *B*.

6.2. General case

We assume here that we have *M* samples, say *M* = 20 (a small number.) Let us assume that *T*'(*n*), *T*''(*n*) and so on, computed on each sample, are also independent with asymptotically (as *n* tends to infinity) the same variance denoted as Var[*T*(*n*)]. Then the (say) 95% confidence interval for *T* has a width *L*(*n*) proportional to the range of its values computed on the *M* samples. Let *p* be the proportion in question, depending on the confidence level. In short, asymptotically, E[*L*(*n*)] is the expectation of the range of *M* independent Gaussian variables with same mean E[*T*(*n*)], and same variance Var[*T*(*n*)], multiplied by *p*. Thus, E[*L*(*n*)] ~ A/n^B is still proportional to $(\mathrm{Var}[T(n)])^{1/2}$.

Intuitively, when the observations have strong, long-range auto-correlations, the variance of any estimator -- which is itself an increasing function of the variance in the observations -- is small, and thus *B* is high. See this spreadsheet for such an example, with *B* = 0.8.

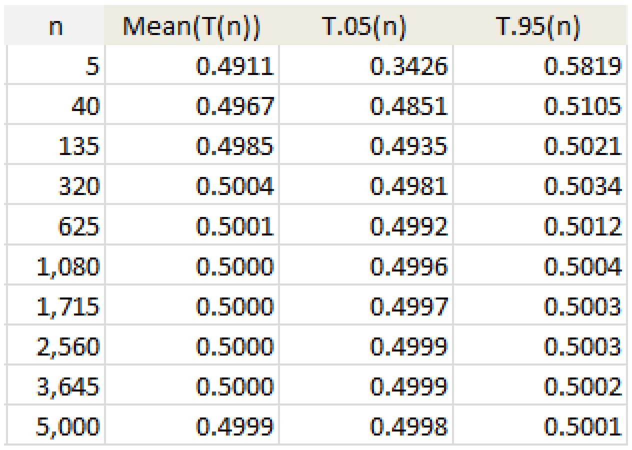
6.3. Another approach to building confidence intervals

This framework suggests yet another potential approach to estimating *B* and obtaining confidence intervals for *T*:

- Compute *T*(*n*) and its empirical percentiles on *M* independent samples of size *n*, with increasing values of *n*. Instead of independent samples, you could use *M* reshuffled versions of one sample if you don't have much data, but you need to be careful about biases.
- Compute the mean, empirical percentiles, and variance of *T*(*n*) across the *M* samples, for several values of *n*. The mean and empirical percentiles will give you confidence intervals (CI) of any level for *T*, for different values of *n*. You can extrapolate these CI's to any value of *n*, using model fitting techniques. If you do the same with the empirical variance of *T*(*n*), fitting it with a power curve A/n^B, you then get an estimate for *A* and *B*.

If you use this approach, you can use sample sizes that are smaller, but the number *M* of samples must be large enough, at least *M* = 20. If your samples come from just one sample set that you reshuffle *M* times, you still need to use a large sample size but focus on nested sub-samples of growing sizes that are not too large, as all your estimated values across all samples, will be identical once you reach the full sample size.

The computations and results, using this approach, are found in this spreadsheet. The data set used here also has an high *B* exponent, above 0.8. You would expect to find patterns in the data with such a high *B*, and this is the case here: look at column Y in the Details tab. Below are the model-free 90% confidence intervals for various values of *n*. The estimator *T* investigated here is the mean. Its true theoretical value is 0.5. Standard statistical techniques would not work here due to the long-range auto-correlations in the data. In particular, convergence to the theoretical value is much faster than with standard techniques applied to standard data corresponding to *B* = 1/2.



n	Mean(T(n))	T.05(n)	T.95(n)
5	0.4911	0.3426	0.5819
40	0.4967	0.4851	0.5105
135	0.4985	0.4935	0.5021
320	0.5004	0.4981	0.5034
625	0.5001	0.4992	0.5012
1,080	0.5000	0.4996	0.5004
1,715	0.5000	0.4997	0.5003
2,560	0.5000	0.4999	0.5003
3,645	0.5000	0.4999	0.5002
5,000	0.4999	0.4998	0.5001

The iterative computation of the 10,000 medians *Q*(*n*) is very slow. If you clear column S in the Details tab in the spreadsheet, computations will run much faster, but you will miss the estimate of *B* based on these medians. Finally, if you use one sample from the previous spreadsheet, and reshuffle it *M* times to produce *M* samples (as opposed to using non-overlapping samples as in the previous spreadsheet) then you kill many of the patterns present in that data set, and as a result your estimated *B* is much closer to

1/2, and the speed at which confidence intervals converge to the true theoretical value, will be slower, that is, more "normal". This is illustrated in this spreadsheet.

7. Conclusions

Some frameworks can handle unclean data with lack of independence between observations, lack of stationarity, non-Gaussian behavior and so on. For instance, in time series, the Hurst exponent *H* plays a role very similar to our exponent *B*. Both lie in [0, 1], take on similar values depending on whether the data is very chaotic (*H* and *B* < 1/2), unusually smooth (*H* and *B* > 1/2) or well behaved like a standard Brownian motion (*H* = *B* = 1/2).

In the theory of martingales (we are actually dealing with martingales here), there is a generalization of the central limit theorem, known as the martingale CLT, stating under which assumptions *B* = 1/2, even in cases where auto-correlations are strong. However, I could not find any general framework that deals with accurately extrapolating confidence intervals beyond the size of your data set, allowing you to perform robust statistical inference with all sorts of estimators applied to messy data, without using any statistical model. Traditional re-sampling techniques based on empirical percentiles are of some use. The novelty of our approach is to bring these re-sampling techniques to a whole new level, solving problems thought to be unsolvable, for instance getting confidence intervals much sharper than those obtained with traditional methods, or getting sharp estimates for *B*, even in the non-standard case when *B* is smaller or larger than 1/2.

17. A Beautiful Probability Theorem

This is another spectacular property of the exponential distribution, and also the first time an explicit formula is obtained for the variance of the range, besides the uniform distribution. It has important consequences, and the result is also useful in applications.

Theorem
The range R(n) associated with n independent random variables with an exponential distribution of parameter l satisfies

$$\mathrm{E}[R_n] = \frac{1}{\lambda} \cdot \sum_{k=1}^{n-1} \frac{1}{k},$$

$$\mathrm{Var}[R_n] = \frac{1}{\lambda^2} \cdot \sum_{k=1}^{n-1} \frac{1}{k^2}.$$

Before proving the theorem, note that the first formula is well known, only the second one is new. The standard proof for the expectation is not considered simple: it is based on computing the expectation for the maximum (see here) and the fact that the minimum also has an exponential distribution with known expectation (see here). Our proof is simpler and also covers the variance.

Proof

The general distribution of the range is known for any distribution, see here. The range is defined as

$$R_n = \max(X_1, \cdots, X_n) - \min(X_1, \cdots, X_n).$$

In the case of the exponential distribution, the range computed on *n* random variables has the following density (see here page 3):

$$f_{R_n}(x) = (n-1)\lambda e^{-\lambda x}(1-e^{-\lambda x})^{n-2}.$$

With a simple change of variable, the *k*-th moment of the range is equal to

$$\mathrm{E}[R_n^k] = \int_0^\infty x^k f_{R_n}(x)dx = \frac{n-1}{\lambda^k}|J_n(k)|,$$

$$J_n(k) = \int_0^1 \log^k(x) \cdot (1-x)^{n-2} dx.$$

Using WolframAlpha (see here and here) one obtains

$$J_n(1) = -\frac{H_{n-1}}{n-1} \text{ and } J_n(2) = \frac{1}{n-1} \cdot \left(H_{n-1}^2 - \psi_1(n) + \frac{\pi^2}{6}\right).$$

Thus,

$$\mathrm{E}[R_n] = \frac{H_{n-1}}{\lambda} \text{ and } \mathrm{Var}[R_n] = \mathrm{E}[R_n^2] - \mathrm{E}^2[R_n] = \frac{1}{\lambda^2}\left(\frac{\pi^2}{6} - \psi_1(n)\right).$$

The two symbols $H(n\text{-}1)$ and $\psi_1(n)$ represent the harmonic numbers and the Trigamma function, respectively. To complete the proof, use the fact that

$$H_{n-1} = \sum_{k=1}^{n-1} \frac{1}{k}, \quad \frac{\pi^2}{6} = \sum_{k=1}^{\infty} \frac{1}{k^2}, \quad \psi_1(n) = \sum_{k=n}^{\infty} \frac{1}{k^2}.$$

■

There are a number of interesting consequences to this result. First, the expectation of the range grows indefinitely and is asymptotically equal to log n. Also, the variance of the range grows slowly and eventually converges to $\pi^2/6$. This is in contrast to the uniform distribution: its range is bounded, and its variance tends to zero as fast as $1/n^2$, see section 2.3 in chapter 16.

This result is pretty deep. It is almost like the range, for the exponential distribution, is made up of a weighted sum of independent exponential variables with same parameter λ, with the k^{th} term added into the sum contributing with a weight equal to $1/k$.

But perhaps most importantly, we found the two extreme cases to a new statistical theorem (see chapter 16, section 1) stating that the length of any confidence interval attached to an estimator is asymptotically equal to A/n^B, with B between 0 and 1. This length is usually proportional to the standard deviation of the estimator in question. In practice, in almost all cases, $B = 1/2$. However, here we have:

- For the range, if the variables are independently and uniformly distributed, then $B = 1$.
- For the range, if the variables are independent with exponential distribution, then $B = 0$.

For normal variables, Var[Range] = $O(1/n)$ and E[Range] = $O((\log n)^{1/2})$, thus $B = 1/2$ (see here and here.) These results are summarized in the table below:

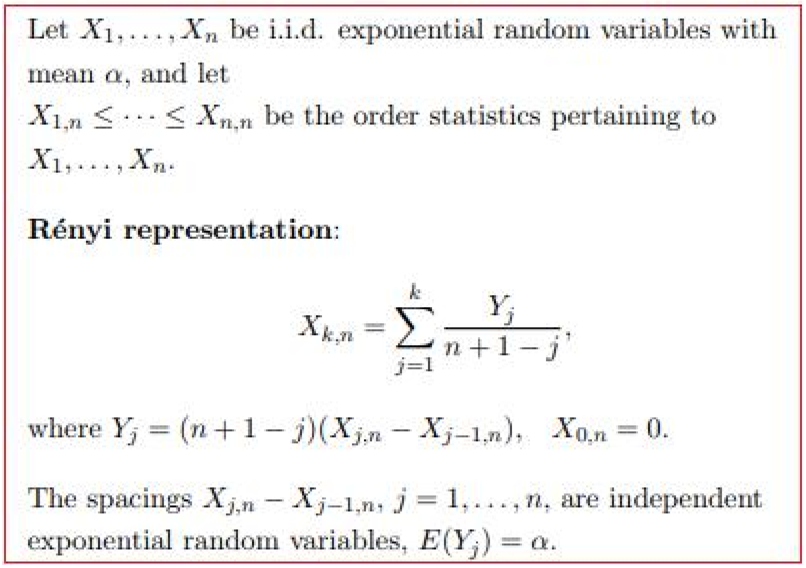
	Type of Distribution	Expectation	Standard deviation
Uniform	Short tail	1	$\frac{1}{n}$
Gaussian	Medium tail	$\sqrt{\log n}$	$\frac{1}{\sqrt{n}}$
Exponential	Fat tail	$\log n$	1

Order of magnitude for the expectation and Stdev of the range

Finally, the same technique could be used to compute higher moments, or to compute the variance of the range for other probability distributions. It could also help with studying the convergence of the re-scaled range and its associated Hurst exponent, see section 6.1in chapter 16 for details.

Connection with order statistics and the Renyi Representation

Joe Blitzstein (teaching probability at Harvard University) pointed out (see here) that my theorem is a particular case of a general result that applies to exponential distributions, known as the Renyi representation. This general result is illustrated in the picture below and in this document.


Let $X_1, \ldots, X_n$ be i.i.d. exponential random variables with mean α, and let
$X_{1,n} \leq \cdots \leq X_{n,n}$ be the order statistics pertaining to $X_1, \ldots, X_n$.

Rényi representation:

$$X_{k,n} = \sum_{j=1}^{k} \frac{Y_j}{n+1-j},$$

where $Y_j = (n+1-j)(X_{j,n} - X_{j-1,n}), \quad X_{0,n} = 0.$

The spacings $X_{j,n} - X_{j-1,n}$, $j = 1, \ldots, n$, are independent exponential random variables, $E(Y_j) = \alpha$.

This also brings something very interesting: since my proof relies on the fact that the sum of the inverse of the squares is Pi^2/6 and since Renyi's argument is entirely probabilistic, it is thus possible to prove, using probabilistic arguments alone, that the sum of the inverse of the squares is Pi^2/6. I will look at higher moments to see if there are some other facts about mathematical constants or integrals, that can be proved (thanks Renyi!) using probabilistic arguments alone. With some chance, I might even discover a new relationship.

Finally, another way to prove the result is to use the fact (see here) that

$$\int_0^1 \log^k(x) \cdot x^{n-2} dx = \frac{(-1)^k k!}{(n-1)^{k+1}}.$$

18. Deep Math Gaming Platform

I describe here the ultimate number guessing game, played with real money. It is a new trading and gaming system, based on state-of-the-art mathematical engineering, robust architecture, and patent-pending technology. It offers an alternative to the stock market and traditional gaming. This system is also far more transparent than the stock market, and cannot be manipulated, as formulas to win the biggest returns (with real money) are made public. Also, it simulates a neutral, efficient stock market. In short, there is nothing random, everything is deterministic and fixed in advance, and known to all users. Yet it behaves in a way that looks perfectly random, and public algorithms offered to win the biggest gains require so much computing power, that for all purposes, they are useless -- except to comply with gaming laws and to establish trustworthiness.

We use private algorithms to determine the winning numbers, and while they produce the exact same results as the public algorithms (we tested this extensively), they are incredibly more efficient, by many orders of magnitude. Also, it can be mathematically proved that the public and private algorithms are equivalent, and we actually proved it. We go through this verification process for any new algorithm introduced in our system.

In section 4.1, we offer a competition: can you use the public algorithm to identify the winning numbers computed with the private (secret) algorithm? If yes, the system is breakable, and a more sophisticated approach is needed, to make it work. I don't think anyone can find the winning numbers (you are welcome to prove me wrong), so the award will be offered to the contestant providing the best insights on how to improve the robustness of this system. And if by chance you manage to identify those winning numbers, great, you'll get a bonus! But it is not a requirement to win the award.

Content

- Smooth ROI Tables
- Systems with Winning Numbers in [0, 1]

1. Description, Main Features and Advantages

Rather than trading stocks or other financial instruments, participants (the users) purchase numbers. Sequences of winning numbers are generated all the time, and if you can predict the next winning number in a given sequence, your return is maximum. If your prediction is not too far from a winning number, you still make money, but not as much. Our system has the following features:

- The algorithms to find the winning numbers are public and regularly updated. Winning is not a question of chance: all future winning numbers are known in advance and can be computed using the public algorithm.
- The public algorithm, though very simple in appearance, is not easy to implement efficiently. In fact, it is hard enough that mathematicians or computer scientists do not have advantages over the layman, to find winning numbers.
- To each public algorithm, corresponds a private version that runs much, much faster. We use the private version to compute the winning numbers, but both versions produce the exact same numbers.
- Reverse-engineering the system to discover any of the private algorithms, is more difficult than breaking strong encryption.
- The exact return is known in advance and specified in public *ROI tables*. It is based on how close you are to a winning number, no matter what that winning number is. Thus, your gains or losses are not influenced by the transactions of other participants.
- The system is not rigged and cannot be manipulated, since winning numbers are known in advance.
- The system is fair: it simulates a perfectly neutral stock market.
- Participants can cancel a transaction at any time, even 5 minutes before the winning number is announced.
- Trading on margin is allowed, depending on model parameters.
- The money played by the participants is not used to fund the company or pay employees or executives. It goes back, in its entirety, to the participants. Participants pay a fee to participate.

Comprehensive tables of previous winning numbers are published, even well before a new sequence (based on these past numbers) is offered to players. It helps participants to design or improve their strategies to find winning numbers. Actually, past winning numbers are part of the public data that is needed to compute the next winning numbers, both for participants and the platform operators.

Various ROI tables are available to participants, and you can even design your own ones. If you are conservative, you can choose one offering a maximum return of 10% (for finding the exact value of a winning number), a 54% chance of winning on any transaction, and a maximum potential loss of 4%. This table is safe enough that we will allow you to "trade" on margin. Another interesting ROI table offers a maximum return of 330%, and the same 54% chance of winning on any transaction, with a maximum potential loss of 4%. Keep in mind that this return is what you can make (or lose) in one day, on one sequence. New winning numbers are issued every day for each life sequence, so your return (negative or positive) gets compounded if you play frequently.

If you are a risk taker, you may like a table offering a maximum return of 500%, a 68% chance of winning on any transaction, and a maximum potential loss of 60%. Or another table with a maximum return of 600%, a 80% chance of winning, but a maximum potential loss of 100%. To download all the sample ROI tables discussed in this presentation, click here.

All the sequences currently offered on the market consist of 8-bit numbers: each winning number (a new one per day per sequence) is an integer between 0 and 255. We will soon offer 16-bit numbers. By design, all ROI tables (even if you use a customized one) offer an average return of 0%. This is true regardless of the sequence you are playing with: sequences and ROI tables are independent.

The participant can test various strategies: for instance:

- Try various ROI tables
- Play every day until you experience your first win (this may not happen for a long time)
- Play every day until you experience your first loss (this may not happen for a long time)
- Play until you have achieved a pre-specified goal, or exit (similar to a stop order on the stock market) if your losses reaches some threshold (some participants might want to continue hoping to recoup some losses)
- Increase or decrease how much you spend depending on your results
- Look if you can find patterns in the winning numbers, exploit them

Below, we explain how this works, using a real-life example.

2. How it Works: the Secret Sauce

Here is an example of a *sequence* being tested in our lab. It shows how the winning numbers are computed, for the sequence in question. The purpose is to illustrate the mechanics, applied to one of our 8-bit systems. The 32-bit version offers more flexibility, as well as potential returns that can beat those of a state lottery jackpot. Our sample 8-bit sequence is defined by the public algorithm below.

2.1. Public Algorithm

Start with initial values x_0 and y_0 that are positive integers, called *seeds*. Then for t = 0, 1, 2, and so on, compute x_{t+1} and y_{t+1} iteratively as follows:

```
If 4x(t) + 1 < 2y(t) Then
    y(t+1) = 4y(t) - 8x(t) - 2
    x(t+1) = 2x(t) + 1
Else
    x(t+1) = 2x(t)
    y(t+1) = 4y(t).
```

2.2. The Winning Numbers

The future winning numbers for a particular sequence, start at a specific machine-generated iteration *T* that no one knows, not even the platform operator nor its software engineers. Typically, $T > 30,000,000$ and can be chosen randomly. The iterations represent the time. The future winning numbers are always integers between 0 and 255, and they occur only at iterations $t = T$, $T+8$, $T+16$, $T+24$, and so on. Their value at iteration *t* is $x_t - 256\, x_{t-8}$. The reason for skipping 7 out of 8 numbers is to make sure that winning numbers are not auto-correlated.

Past winning numbers are those occurring at iterations $t = T-8$, $T-16$, $T-24$, and so on. The last 2,000 of them are published before the sequence is available (life) on the platform, allowing participants to predict future winning numbers, using the public algorithm or by other means, and make (or lose) money. For our above test sequence, the 2,000 past winning numbers in question, ordered chronologically, are available in this text file.

For each sequence, one new winning number is published each day. So, the time unit used here is 3 hours since one day is 8 x 3 hours. To win the maximum amount, one must correctly predict the winning number attached to a future day. Good and fair approximations also result in a gain, albeit lower. These gains and losses are explicitly specified beforehand, in very precise ROI tables, see below. Finally, by design, the winning numbers are not auto-correlated; they appear independently and uniformly distributed (more so than many software-generated pseudo-random numbers), and do not exhibit any known or visible pattern. In short, they look totally arbitrary, yet generated using a rudimentary formula.

2.3. Using Seeds to Find the Winning Numbers

Most participants are likely to do random trials to find or approximate winning numbers. The few who want to use the public algorithm need extra information to compute winning numbers, and even then, their chance of finding such numbers is virtually zero, due to the tremendous amount of computations required. In short, you need to know the seeds, and when to stop your computations. The stopping rule is simple: you stop when you have found numbers that match the past winning numbers publicly available. Then you know for sure that your next number will be a winning one.

We offer information about the seeds in two different ways:

- You can request seeds that work. The working seeds that we provide are integer numbers consisting of many digits. In our particular case, the following seeds work: x_0 and y_0. You can download them as text files, by clicking on these two links. Both x_0 and y_0 contain about 250,000 digits in base 10.
- Or you can use the information provided with the public algorithm: the fact that there is a set of seeds (and only one) leading to the winning numbers, and consisting of positive integers lower than 1,000.

We guarantee the following:

- With the wrong seeds, you won't find the winning sub-sequence (matching public past winning numbers) in your lifetime, no matter how much computing power you use.
- With the right seeds, you will find the winning sub-sequence (matching public past winning numbers) only once, and in less than 32 trillion iterations.

So we offer you a way to find the next winning numbers, and you know in advance how much you will win when finding them, using the ROI table. The question is: how many years would the most powerful computers in the world need, to make all these computations? By contrast, as of January 2019, only 31.4 trillion digits of π are known, and computing them require several months using a lot of computing power, together with very clever mathematical engineering bearing some resemblance to our private algorithms. And checking that all these digits (not just the first few trillion) are correct, is another big problem. Here, if you make any tiny mistake in your computations, you will miss the past sequence of winning numbers.

Of course, you could be a mathematical genius, and somehow figure out what the private algorithm is, to make your computations far more efficiently. This is highly unlikely to happen. There is a considerable amount of very advanced, unpublished mathematical research that has been done to make our systems robust. Also, we regularly change the type of sequences that we use in our system, every few months or so. And we work with white hat hackers (paid to hack our system) in order to identify potential vulnerabilities.

Finally, seeds that lead to unpredictable winning numbers (simulating an efficient market) are known as *good seeds*. Of course, all the sequences that we offer are based on seeds highly believed to be good ones, and that have been run through a battery of statistical tests. Using sequences based on bad seeds would not hurt the players, quite the contrary, but it would make our system easier to crack and cause problems with the ROI tables, thus hurting us.

Proving that specific seeds are good or bad, is one of the most challenging, unsolved mathematical problems of all times. If solved, we would know for sure whether the digits of a number such as π, are evenly distributed or not. These mathematical concepts have been studied for some time; see recent material on this topic, here and here.

2.4. ROI Tables

The ROI tables tell you how much money you will make or lose when submitting a number. Your ROI is a function of the distance between your submitted number *z* and the actual winning number *x*. The distance, also called *error*, is computed as follows: $d(x, z) = \min(|x - z|, 256 - |x - z|)$. It is always an integer value between 0 and 128. A pre-determined ROI is attached to each of the 129 potential error values. These ROI's characterize the type of risk that you are willing to take, and can be customized by each user, as long as the theoretical expected return (automatically computed in the ROI spreadsheet) is zero.

You will find these values in the ROI tables, available in spreadsheet format, here. Look at the second row in the spreadsheet, between column K and EI. The spreadsheet also contains 1,000 user-submitted numbers (simulations) with the ROI computed for each submitted number. Other summary statistics of interest are available in the spreadsheet: highest and lowest potential payout, chances of winning, and more.

3. Business Model and Applications

Accredited investors, hedge funds, stock trading brokers, stock exchange companies, cryptocurrency operators, government organizations (for instance, state lotteries and agencies interested in creating a lottery at the federal level) as well as game developers and companies in the gaming industry, are welcome to contact us. Investors potentially interested in participating in a first round of funding to create and scale this platform, and who can bring clients and/or a CEO of their choosing, are also invited. We traditionally work smart and fast, with very small efficient teams in a lean environment, with people located all over the world.

This short presentation only features the tip of the iceberg. The possibilities are endless, including the implementation of:

- ROI tables that favor participating brokers over players (or the other way around),
- 16 or 32 bit systems offering spectacular potential returns yet no potential big loss, see section 5
- Short-selling,
- Sequences that are cross-correlated or auto-correlated, offered to VIP clients to help them gain a competitive advantage,
- Sequences with variable ROI tables, sometimes favoring the players, and sometimes favoring the operators,
- Allowing participants to schedule purchases ahead of time, and to upload guessed numbers in bulk
- Automated black-box trading (we create your daily guesses -- they consist of pseudo-random numbers; you choose your ROI tables).

Some of these features allow players to sometimes slightly beat the official and neutral odds of winning, offering a true positive return on average for some short periods of time, at the expense of the operators. For the organization implementing these features, this can be seen as marketing costs to attract new customers. Other potential applications includes Blockchain and cryptocurrency technology, strong encryption, patent and security laws, and state-of-the-art, innovative research in statistical science, computer science, and number theory. Finally, the system can also be used for simulated trading, to test various strategies with various ROI tables.
Let's now look at how the money flows.

3.1. Managing the Money Flow

Managing the money involves subtracting or adding dollars to user accounts after each completed transaction. On a given day, how do we know whether on average, gains and losses will balance out, since we don't control the numbers entered by the participants?

Actually, we don't know. Sometimes the balance is slightly negative, sometimes slightly positive. However, by using fair ROI tables and good seeds, we are guaranteed to be flat on average. You can even compute the daily volatility resulting from the daily winning and losing transactions. Example: with 1,000 transactions in a single day, each one consisting of a $20 bet, the most conservative ROI table introduced in this presentation produces a theoretical standard deviation of $24, over a volume of $20,000. The most aggressive one produces a standard deviation of $314, still entirely manageable. These theoretical numbers have been confirmed by simulations, and are included in each ROI table, for internal use. When offering customized ROI tables, you might want to put a cap on the standard deviation being allowed. See section 4.2 for more details.

3.2. Virtual Currency

Rather than actual dollars, the operator could use a virtual currency. The currency is issued by the operator (it could even be tokens), while the real money is held by an escrow company. Since on average no real (nor virtual) money is made by the operator on the gains and losses of the participants (assuming the system is fair,) no tax should be paid by the operator on the deposits made by the participants, and no tax deduction allowed for the money distributed to participants. This is made easier using of a virtual currency. Of course, if users pay a fee to participate, the operator will have to pay taxes on this source of revenue.

Participants are expected to pay taxes on their gains, and in an ideal word, deduct losses. The "tax event", for the participant, occurs when the escrow company disburses the money, if it comes with a gain or a loss. This could take place after any bet or at any time that is convenient for the participant. The operator also deposits extra money on the escrow company, to cover the maximum *cash float* and keep a positive balance at all times. The cash float is the difference between aggregated gains and losses across all participants. The cash float typically represents less than 3% of the value of the portfolio of bets managed by the operator.

4. Challenge and Statistical Results

We discuss here two important statistical results that make this system works. But first, let's talk about the competition announced at the very beginning.

4.1. Data Science / Math Competition

We plan to organize a competition focusing on the public algorithm. The goal is to compute the next 200 winning numbers, in the right chronological order, using

- the public algorithms described in section 2.1,
- the two public seeds x(0) and y(0) provided in section 2.3.
- and the 2,000 past winning numbers provided in section 2.2.

You can use the methodology described in this article, or any other means. The award will be offered to participants providing the best insights on how to improve the robustness of our system. So it is not required to find the 200 next winning numbers to earn the award. But if you do find them, we offer a bonus. We will announce the competition on Data Science Central. To not miss the announcement, you can sign-up to receive our newsletter.

4.2. Controlling the Variance of the Portfolio Value

Any guess regarding a winning number results in a gain or a loss depending on how close your guess *z* is to the winning number x. The metric used to measure the proximity between x and z is

$$d(x, z) = \min(|x - z|, 256 - |x - z|)$$

All winning numbers are integers between 0 and 255. If the participants made random guesses, then the distance d(*x*, z) would be a random variable, say *D*, with the following distribution:

- $P(D = 0) = 1/256$,
- $P(D = 128) = 1/256$,
- $P(D = k) = 2/256$ if *k* is strictly between 0 and 128.

The money that you put on a number (your guess) is called *principal*, similarly to the money invested in a stock, in the stock market. Once the winning number is announced, your principal increases or decreases depending on how good your guess is. Your principal is actually multiplied by a factor *G*(d(*x*, *z*)) which is a function of the distance between the number you picked up, and the winning number.

The multipliers *G*(0), *G*(1), *G*(2) and so on, up to *G*(128), are known in advance and specified in the ROI table that you use. The ROI tables are fair, in the sense that the average gain for the player, is zero. In order to achieve this goal, ROI tables are designed so that

$$\mathrm{E}[\mathrm{Gain}] = \frac{1}{256}\left\{G(0) + G(128) + 2\cdot\sum_{k=1}^{127} G(k)\right\} - 1 = 0.$$

If the top multipliers offered are very high -- the highest being *G*(0) for a correct guess -- then, even though the system is fair (unbiased), the variance for the gain for a single guess, is also high. This variance, assuming E(gain) = 0 and the participant puts $1 per guess, is equal to

$$\mathrm{Var}[\mathrm{Gain}] = \frac{1}{256}\left\{G^2(0) + G^2(128) + 2\cdot\sum_{k=1}^{127} G^2(k)\right\} - 1.$$

The total value of the portfolio that we manage, defined as the aggregated principal across all guesses, is flat over time but experiences daily fluctuations. To compute its variance, use the previous formula, and multiply it by both the number of guesses and the dollar amount attached to them. The standard deviation values mentioned in section 3.1 (about money management) is the square root of this variance, assuming we have 1,000 guesses, each with a $20 price tag.

With 1,000 daily guesses each with the same price tag, the most extreme standard deviations for the portfolio, expressed as a percentage of the portfolio value, are:

- **Minimum**: 0%. ROI table where nobody wins, nobody loses, that is, if *G*(*k*) = 1 for all *k*.
- **Maximum** (worst case): 51%. ROI table where participants win only when they correctly guess the winning number (the chance is 1/256), and in that case their principal is multiplied by a factor 256. Otherwise they lose everything. That is, *G*(0) = 256 and all other *G*(*k*) are set to 0.

These percentages decrease as the number of guesses increases. In practice, we stick to ROI tables with a theoretical standard deviation that is less than 3% of the portfolio value. This guarantees our survival in case of extreme events, such as a very big client winning big time on a single guess and claiming her gain right away, or a "bank run".

4.3. Probability of Cracking the System

The sequences used in our system generate numbers that look random. The successive past winning numbers published to help you find the next one -- even though it is a small list of *K* = 2,000 integers between 0 and 255 -- look just as random. Without using working seeds x_0 and y_0 that are known to lead to the solution (albeit in a very large number of iterations, manipulating numbers with many millions of digits most of the time), the chance of finding in any given sequence, the *K* successive numbers matching the *K* past winning numbers, in less than *M* iterations (say *M* = 30 million), is about

$$\frac{1}{256^K}\left(\frac{M}{8} - K + 1\right).$$

See here for details. Even if you try a million set of seeds, knowing that one and only one of them leads to the solution in less than M iterations, it will take you a staggering amount of time to find it.

If a participant uses the wrong sequence, starting with one of the allowed sets of seeds other than the one that is guaranteed to work, and by some incredible chance the sequence also contains the K past winning numbers in the first M iterations, even if the participant submit a number that is not a winning number, we would still have to pay her as if she had found the winning number. The chance of this happening is virtually zero.

5. Designing 16-bit and 32-bit Systems

So far we discussed 8-bit systems only. As the name indicates, a b-bit system is where the winning numbers are b-bit integers. In a b-bit system, the public iterative algorithm in section 2.1 is still the same with the following adaptations (here b is the number of bits):

- New winning numbers occur at iterations $t = T$, $T + b$, $T + 2b$, $T + 3b$, and so on.
- Past, public winning numbers occur at iterations $t = T - b$, $T - 2b$, $T - 3b$, and so on.
- The formula for the winning numbers changes from $x_t - 256\, x_{t-8}$, to $x_t - 2^b\, x_{t-b}$.

A b-bit ROI table has $1 + 2^{b-1}$ multipliers $G(0)$, $G(1)$, $G(2)$, and so on, up to $G(2^{b-1})$. Also, $G(k+1)$ is chosen to be either equal to or less than $G(k)$, so that participants know that the more accurate their guess, the higher the return.

5.1. Layered ROI Tables

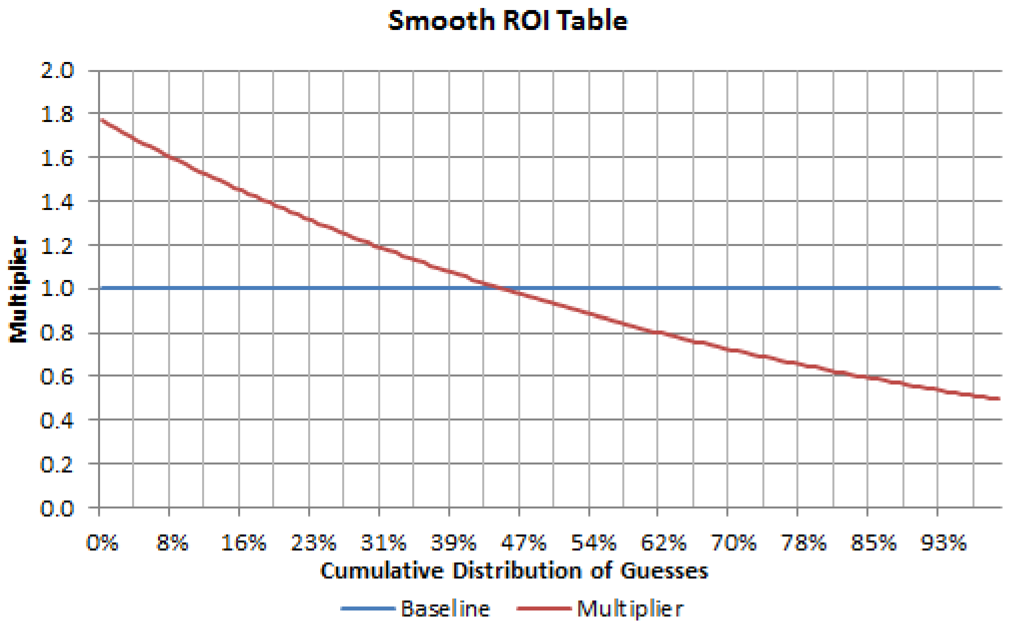
Below is an example of a fair, layered 32-bit ROI table. If your guess is within 19 points of the winning number (it will happen to about 39 people out of 4.3 billion) then your principal is multiplied by a factor one million. About 48.7% of the guesses are not winners, and they erode your principal by 30%. The lowest ROI you get if you are a winner is 15%, and 50.7% of all guesses fall in that category. About one in two hundred (0.54%) results in a 900% ROI. One in 100,000 would boost your principal by a factor 1,000.

k distance to winning number	G(*k*) multiplier	Proportion of guesses
0 to 19	1,000,000	0.00000091%
20 to 26,065	1,000	0.0012%
26,066 to 11,530,000	10	0.54%
11,530,001 to 1,100,765,000	1.15	50.7%
1,100,765,001 to 2,147,483,648	0.70	48.7%

An even more skewed table could be designed, guaranteeing an average return strictly above zero to the player. If the positive return is driven entirely by the multiplier offered for correctly guessing the winning number (and otherwise, excluding a perfect guess, the average return is just a very tiny bit below zero) you might be able to entice more players to participate, especially sophisticated, big ones. If the odds of winning the maximum is less than one in 4 billion in a single bet, it will take so much time and money to win the big prize, that the operator has time to accumulate gains and grow them slightly faster than inflation, so that when the big winning event takes place, enough funds are available to pay the big winner.

5.2. Smooth ROI Tables

It is possible to create smooth ROI tables, with a continuous, slow decline in the multiplier rather than sharp drops as in the above table. One of the most natural functions that comes to mind is the geometric function $G(k) = A/B^k$, with the parameters A and B chosen so that the table is fair both to the player and to the operator. It is illustrated below, using the 8-bit system. We are working on producing similar tables for the 32-bit system.



The smooth tables offer one advantage: no participant will be disappointed for missing a massive payout by only a few points. In the above table (8-bit), $G(k) = 1.7685/1.0100^k$. The table is fair. Note that only 44% of the guesses are winners. The highest multiplier is only 1.77. Also, you can lose as much as 50%. Yet you could argue that this table is far more equitable than those previously discussed.

5.3. Systems with Winning Numbers in [0, 1]

The theory can be extended to winning numbers that are real numbers in [0, 1]. For instance, one can use the seed x_0 = log 3 - log 2 with the sequence $x_{t+1} = \{ 2x_t \}$ where the brackets represent the fractional part function.

Then, $x_t = A_t \log 3 + B_t \log 2 + C_t$ where A_t, B_t, and C_t are integers. The geometric ROI function (above picture) becomes $G(k) = p/q^k$, with p, $q >$ 1, and k a real number in [0, 1]. It has the following features:

- The maximum multiplier is p
- The minimum multiplier is p/q
- The system is fair if $p(q - 1) = q \log q$
- If the system is fair, the probability of winning is $(q - 1)/(q \log^2 q)$
- $d(x, z) = \min(|x - z|, 1 - |x - z|)$.

All the numbers x_t are winning numbers, either past or future. The public information could consist of

- The formula $x_{t+1} = \{ 2x_t \}$
- The last 2,000 winning numbers $x_{T-1}, ..., x_{T-2000}$, computed with 20 correct digits
- The exact values of A_t, B_t and C_t for some secret t between 0 and T
- The two constants log 3 and log 2.

We can replace the third item in the above list, by the value x_t computed with one million correct digits, for some secret t between 0 and T. In that case, there is no need to mention log 3 and log 2.

You can replace log 3 and log 2 with two (or more) irrational numbers that are conjectured to be linearly independent over the set of rational numbers. For instance, π and $5^{1/2}$, or the values of

- The probability than an integer is not divisible by a cube
- The only solution between 0.5 and 1, of $\sin x + \sin(2x \sin x) = \sin 3x$.

Very few people know how to efficiently obtain millions of digits for these values -- which is the first step required to find the winning numbers. Finally, as long as x_0 is a good seed, the numbers generated by the sequence x_t will look random, after proper decorrelating [see how to decorrelate in section 3.2.(a) in appendix B]. The concept of good and bad seed is illustrated in appendix B and in my book on stochastic processes, available here.

Details on how to use multivariate sequences and de-correlating can be found in chapter 13. This system was presented at the INFORMS annual meeting (2019). You can access the PDF document here.

19. Decay-adjusted Rankings

This is a classic business problem. In most online rankings, the most popular books, authors, articles, restaurants and so on are always among those that have been around for a long time. New stars have no chance to beat old-timers, and must wait for a long time before showing up at the top. Here we address this issue and correct for the bias, allowing you to make fair value comparisons between old and new items. The example used here is about popular articles posted on Data Science Central. While time is a major source of bias, there are many other factors artificially inflating or deflating rankings. We review these factors, and propose a solution to create meaningful rankings.

In the process, we created a new, more robust scoring method. This scoring, based on a decay function, could be incorporated in recommendation engines.

1. Introduction

The data covers almost three years' worth of DSC (Data Science Central) traffic, extracted from Google Analytics: more than 50,000 posts, and more than 6 million page views (almost half of it in 2014 alone), across four channels: DSC, Hadoop360, BigDataNews, and AnalyticBridge.

Some articles have been filtered out as they belong to a cluster of similar articles (education, books, etc.) Finally some very popular pages are not included because the creation date is not available or because they should not be listed (my own profile page, the sign-up page, the front page, etc.) The new scoring model is described in section 4.

2. Top DSC blogs posted between 2008 and 2014

The number in parentheses represents the rank if instead of using our popularity score, we had used standard methodology. The date represents when the blog was created. By just looking at these fields, you might be able to guess what our new scoring engine is about. The data used for these computations was collected in October 2014.

1. 17 short tutorials all data scientists should read and practice (2) - 2/15/2014
2. How to Become a Data Scientist (39) - 8/27/2014
3. DSC weekly digest (1) - 6/20/2013
4. 38 Seminal Articles Every Data Scientist Should Read (34) - 8/15/2014
5. 10 types of regressions. Which one to use? (22) - 7/21/2014
6. Data Science Cheat Sheet (36) - 8/1/2014
7. 16 analytic disciplines compared to data science (31) - 7/14/2014
8. Data science book (6) - 11/23/2013
9. 66 job interview questions for data scientists (4) - 2/13/2013
10. One Page R: A Survival Guide to Data Science with R (14) - 2/14/2014
11. Six categories of Data Scientists (11) - 1/16/2014

12. Data scientist core skills (7) - 8/27/2013
13. Update about our Data Science Apprenticeship (5) - 3/10/2013
14. Our Data Science Apprenticeship is Now Live (38) - 5/22/2014
15. What your state is the worst at? (3) - 1/31/2011
16. How to detect spurious correlations, and how to find the real ones (41) - 5/22/2014
17. The best kept secret about linear and logistic regression (32) - 3/13/2014
18. 6000 Companies Hiring Data Scientists (24) - 12/29/2013
19. More than 100 data science, analytics, big data, visualization books (19) - 11/6/2013
20. Fast clustering algorithms for massive datasets (9) - 2/23/2013
21. Batch vs. Real Time Data Processing (17) - 8/13/2013
22. From the trenches: 360-degree data science (40) - 3/27/2014
23. Salary history and career path of a data scientist (37) - 2/19/2014
24. Data scientists making $300,000 a year (8) - 11/29/2012
25. Data Science eBook - 2nd Edition (15) - 5/19/2013
26. Practical illustration of Map-Reduce (Hadoop-style), on real data (35) - 1/25/2014
27. Salary surveys for data scientists and related job titles (23) - 10/10/2013
28. 16 resources to learn and understand Hadoop (44) - 4/18/2014
29. Big data sets available for free (33) - 12/30/2013
30. BI vs. Big Data vs. Data Analytics By Example (21) - 8/25/2013
31. Hadoop vs. NoSql vs. Sql vs. NewSql By Example (27) - 9/8/2013
32. Data Scientists vs. Data Engineers (20) - 7/2/2013
33. The Curse of Big Data (16) - 1/5/2013
34. Jackknife logistic and linear regression for clustering and predict... (46) - 3/19/2014
35. What MapReduce can't do (18) - 1/31/2013
36. The 8 worst predictive modeling techniques (13) - 9/13/2012
37. Berkeley course on Data Science (10) - 3/4/2012
38. Why Companies can't find analytic talent (43) - 1/17/2014
39. New, fast Excel to process billions of rows via the cloud (28) - 4/16/2013
40. How much does a data scientist make at Facebook? (25) - 3/12/2013
41. Fake data science (29) - 2/11/2013
42. Logit vs Probit Regression (12) - 2/19/2009
43. R Tutorial for Beginners: A Quick Start-Up Kit (42) - 10/24/2013
44. How to detect a pattern? Problem and solution (26) - 9/28/2011
45. Job titles for data scientists (45) - 6/3/2013
46. New, state-of-the-art random number generator: simple, strong and fast (30) - 9/11/2011

3. Interesting Insights

These top pages represent 21% of the traffic (back then). The front page amounts to another 9%. Here are some of the highlights:

- For top popular articles, page views peak in the first three days, but popularity remains high for many years. In short, *page view decay* (over time) is very low, see figure 1 below.

- Page view decay is very low for highly popular, generic articles that are time-insensitive, the type of articles that we try to write. Non popular articles or time-sensitive announcements have a very fast decay, typically exponential decay and short half-life.

- You don't notice any decay at all in figure 1. The reason is because decay is hidden by general traffic growth on DSC. The general growth more than compensate for the natural decay.

- Notice a change in subjects that are popular between new material (post 2012) and old material (earlier). You can notice the drastic change only if you use a sound popularity algorithm, as described in section 4.

- Many of the most popular articles are new (once you adjust for the time bias, using the methodology described in section 4). Part of it is because we have a better understanding of the type of articles that our readers are interested in, as well as how to successfully reach out to new users. Part of it is because of growth. An article posted today will immediately receive more than twice the volume of traffic it would have received on day one, if posted 2.5 years ago.

- We have used two data sources, always a good idea in any data science project: Google Analytics, which filters out robots, and Ning page counts, which does not. On average Ning numbers are 30% above Google Analytics - which we translate in the fact that about 30% of the traffic is by robots (Google crawlers etc.) Robot traffic has a time lag of a few months (on average) compared with human traffic.

- Google Analytics has one drawback: it counts two versions of a same page - with only the query string in the two URLs being different - as two different pages. A bit of post-processing can quickly fix this issue. This issue explains why sometimes the discrepancy between Ning and Google Analytics is as high as 60%, as opposed to the average 10-30% range.

- Many popular articles have been posted in last 2 weeks, but I decided not to include them (unless page view count is so high that they naturally appear in our list, after correcting for time bias, as explained in section 4). The reason not to include them is because of high page view volatility for new articles.

- We had to do some time adjustments as our Google Analytic data goes back to 2012 only, while Ning goes back to 2007. Non-experts working on the same project are likely to not even notice the issue, let alone fixing it.

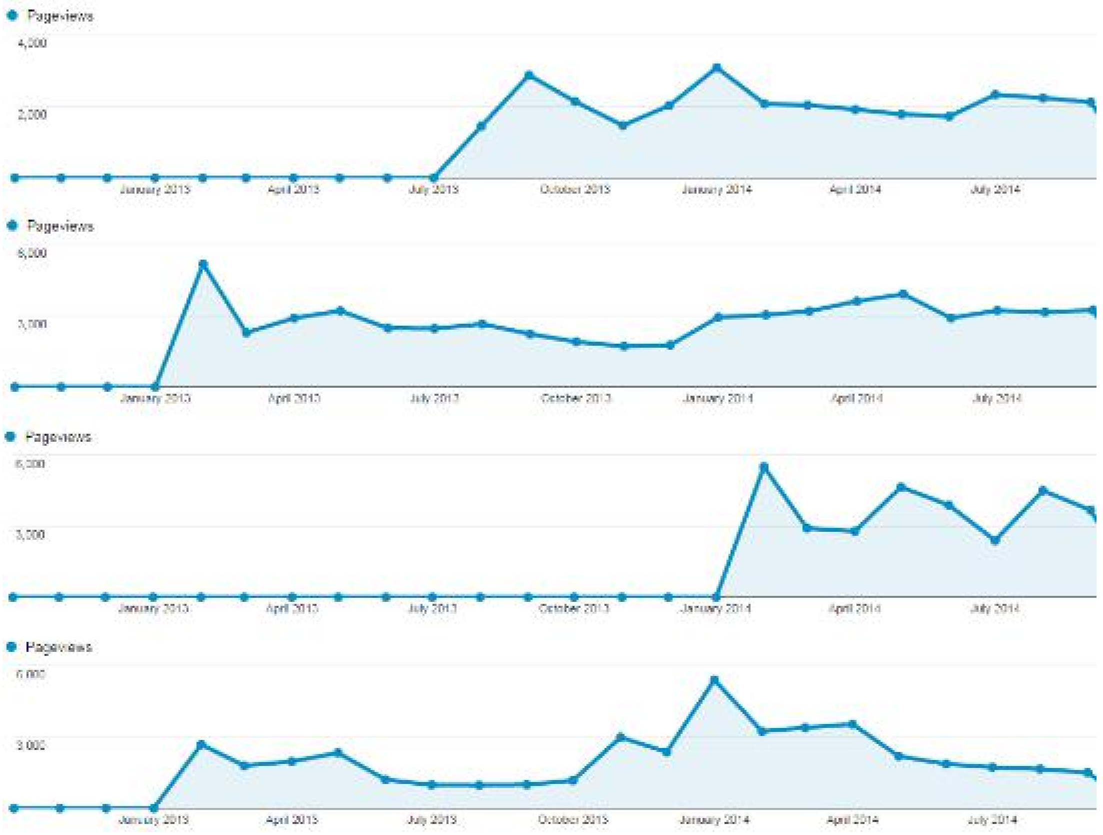


Figure 2: *Page view decay (or absence of decay!) for 4 top blogs listed above*

4. New Scoring Engine

Let's say that you measure the page view count for a specific article, and your data frame is between $t = t_0$ and $t = t_1$. Models like this typically involve exponential decay of rate r, meaning that at time t, the page view count velocity is $f(r, t) = \exp(-rt)$. With this model, the theoretical number of page views between t_0 and t_1 is simply

$$P = g(r, t_1) - g(r, t_0),$$

where $g(r, t) = (1 - \exp(-rt)) / r$.

If t_0 is set to zero, then $g(r, t_0) = 0$, regardless of r. On a different note, the issue we try to address here (adjusting for time bias) is known as left- and right-censored data in statistical science: right-censored because we don't have data about the future and left-censored because we don't have data prior to 2012.

To adjust for time bias, define the **new popularity score** as $S = Q / P$, where Q is the observed page view count during the time period in question. When $r = 0$ (no noticeable decay, which is the case here) and $t_0 = 0$, then $P = t$. Note that the only two metrics required to compute the popularity score S, for a specific article, are: time elapsed since creation date, and page view count during the time frame in question, according to Google Analytics, after aggregating identical pages with different URL query strings.

Note: To make sure that we were not missing popular articles posted recently, we collected the data using two overlapping time frames: one data set for 2012-2014, and one just for 2014, using CSV exports from Google Analytics. Several articles that did not show up in the 2012-2014 data set (because their raw page view count was below our threshold of about 10,000 page views), actually had top scores *S* when adjusted for time, and could only be found by using the 2014 data. Another way to eliminate this issue is to get statistics for all articles (not just the ones with lots of traffic) for the whole time period. That's the automated approach, and in our case it would have required writing extra pieces of code, and possibly Google API calls, to download time stamps on Ning (via web crawling) and the entire Google Analytic data for the 50,000 articles - not worth the effort, especially since I allowed myself only a couple of hours to complete this project.

5. Good versus perfect model

Using the basic model with *r* = 0 (in section 4) makes a big difference with traditional rankings (the base model), as you can see when comparing our rankings to traditional rankings, in our list of top articles in section 2 (sorted according to our popularity score with *r* = 0). It allows you to detect trends about what is getting popular over time.

Note that refining the model, estimating a different *r* for each article, testing the exponential decay assumption, and adjusting for growth, may be time-prohibitive, and it makes your model subject to over-fitting, and may jeopardize the value (ROI) of the project.

Data science is not about perfectionism, but about delivering on time and within budget. In this case, if I spend one month on this project (or outsource to people who work with me), it's time wasted on something that could yield far more value than the little incremental gain obtained by seeking perfection. Yet ignoring the decay is equally bad, it makes this whole project worthless. The data scientist must instinctively find what level of perfection is needed, in his models. Data is always imperfect anyway.

6. Next steps

One interesting project is to group pages by categories and aggregate popularity scores, and create popularity scores for categories. Other potential improvements include:

- Estimating *r* rather than using *r* = 0
- Estimating *r* for each article (risk of over-fitting)
- Scoring bloggers rather than blogs
- Testing the exponential decay assumption
- Adjusting scores to take traffic growth into account (favoring new blogs over old ones)

Another area of research is to understand why page view counts closely follow a Zipf distribution.

20. Building a Website Taxonomy

We built a taxonomy of data science websites in 2014, by analyzing our member database (100,000 members back then), extracting websites that our members mentioned or liked, and for each web site, identifying

- When it is first mentioned by one of our members
- The number of times it was mentioned
- Keywords found when visiting the front page with a web crawler, using a pre-selected list of seed keywords.

1. Seed Keywords

Seed keywords were used to identify, for each website, whether one or more of the keywords in our list was found on the front page, using a web crawler. This helps categorize websites - the final goal being the creation of a data science website taxonomy. The seed keywords that we used (hand-picked) were popular data science related keywords:

- analytics
- data science
- database
- hadoop
- predictive modeling
- big data
- business intelligence
- machine learning
- data mining
- text mining
- operations research
- statistics.

2. General Methodology

We used a web crawler to browse all the URLs, after identifying and cleaning the websites fields (URLs listed by members), in our member database. Click here to get the script used to summarize the data, as well as a sample of raw data. The ultimate goal was to create a niche search engine for data science, better than Google, and a categorization of these websites. Because this is based on data submitted by users, the raw data is quite messy and requires both cleaning and filtering. Details are found in my script - it's a good example of code used to clean relatively unstructured data.

Here we categorize the websites in four major clusters:

- Websites mentioned at least 3 times, containing at least one of the seed keywords in our list
- Websites mentioned less than three times, containing at least one of the seed keywords in our list
- Websites that we were unable to crawl, mentioned at least twice
- Websites containing no seed keywords from our list, and mentioned at least 4 times

We provide direct clickable links for domains in category 1 (above and below) only. The choices of these various parameters is to guarantee robustness in our results, filter out noise, and for internal security reasons: listing hundreds of little know websites (with clickable links) can get you penalized by Google, can result in many requests for link removal, and many links can die over time, require regular crawling for maintenance.

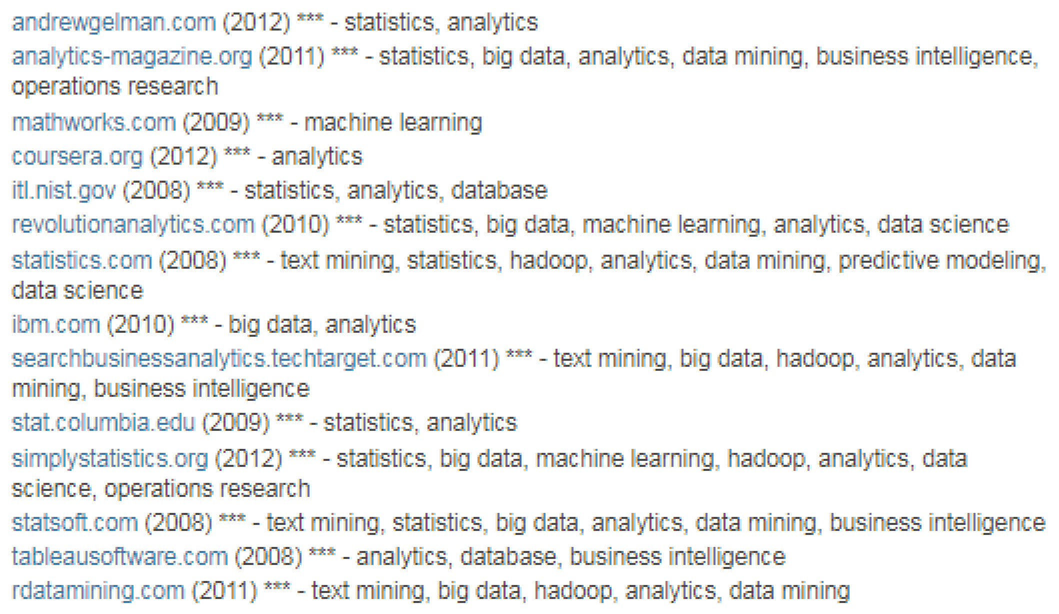
3. Top 2,500 Data Science Websites

Below are the links to the top data science websites. Keep in mind that this dates back to 2014.

- Top 2,500 Websites - top of the top
- Top 2,500 Websites - mentioned only a few times -- Page 1 | Page 2
- Top 2,500 Websites - not crawlable
- Top 2,500 Websites - not containing seed keywords

The field between parentheses represents the year when the website in question was first mentioned - it does not represent when the website was created, though it's a good proxy to tell how old the website is. The member database goes as far back as 2007. The list of keywords attached to each website represents which seed keywords were found on the front page, when crawling the website. The number of stars (1, 2 or 3) represents how popular the website is: it's an indicator of how many members mentioned it. Of course, brand new websites might not have 3 stars yet.

Below is an extract from the list:


andrewgelman.com (2012) *** - statistics, analytics
analytics-magazine.org (2011) *** - statistics, big data, analytics, data mining, business intelligence, operations research
mathworks.com (2009) *** - machine learning
coursera.org (2012) *** - analytics
itl.nist.gov (2008) *** - statistics, analytics, database
revolutionanalytics.com (2010) *** - statistics, big data, machine learning, analytics, data science
statistics.com (2008) *** - text mining, statistics, hadoop, analytics, data mining, predictive modeling, data science
ibm.com (2010) *** - big data, analytics
searchbusinessanalytics.techtarget.com (2011) *** - text mining, big data, hadoop, analytics, data mining, business intelligence
stat.columbia.edu (2009) *** - statistics, analytics
simplystatistics.org (2012) *** - statistics, big data, machine learning, hadoop, analytics, data science, operations research
statsoft.com (2008) *** - text mining, statistics, big data, analytics, data mining, business intelligence
tableausoftware.com (2008) *** - analytics, database, business intelligence
rdatamining.com (2011) *** - text mining, big data, hadoop, analytics, data mining

4. Data and Source Code

Source code (two scripts including a web crawler / parser / summarizer, and code to produce final HTML pages), as well as raw, intermediate and final data (samples, screen shots), and details about the 3-step procedure used to publish these listings, can be found here.

5. Detailed Methodology

Our methodology, to build our semi-categorized website listing, has the following additional features:

- All webpages were stored as strings (after download), all in lower case. The seed keywords were also in lower case.
- Within each sub-group in each of the four major categories, websites are displayed in random order: using a stars system (rather than detailed score) makes for more robust, accurate results. Sorting websites by score (score = number of members mentioning the website in question) would result in various drawbacks: website owners complaining about their score, and sometimes for good reasons!
- My script takes about 20 minutes to run on one machine to crawl 2,800 websites. I only read the first 64K of each page, and the http requests times out after 1 second. It would be much faster if multi-threaded.
- The fourth major category of websites (those containing no seed keywords, and mentioned at least 4 times) is interesting nevertheless: it shows which *non-analytic* (general, mainstream) websites our members also visit.

- Some of the websites where no seed keywords were found are actually analytic websites, and the lack of analytic keywords might be caused either by a glitch in our script, or in the way the webpage is encoded (iFrames, heavy Javascript, Flash and other page creation techniques giving a headache to our webcrawler, and indeed to all webcrawlers including Google). These represent only a small percentage (< 5%) of all websites. Maybe crawling a few webpages, not just the frontpage (for each website returning no seed keywords) could fix the issue. This implies *deep crawling*, following internal links found on the frontpage.

Uncrawlable websites, bad domains

- As many as 800 out of 2,800 original all websites could not be crawled. I re-run the crawler on these websites a few hours later, increasing the value of the time-out parameter, and using a different *user agent string* in the code (that is, the `$ua->agent` argument for those familiar with the web crawling `LWP::UserAgent` library). I then re-run it a few more times the same day, and eventually managed to reduce the number of un-crawlable websites to about 300. Maybe trying another day, with a different IP address, following the robot.txt protocol (crawling `robots.txt` on each failed website) might further reduce the number of failed crawls. However, about 250 of the uncrawlable websites were just simply non-existent, mostly because of typos in member fields (user-entered information) in our database.
- Some of the uncrawlable websites result from various redirect mechanisms that cause my script not to work, or sometimes because it redirects to an https address (rather than http).
- I extracted the error information for all uncrawlable websites. Typically, the "500 Bad Domain" error means that the domain does not exist (rarely, it is a redirect issue). Sometimes adding www will help (changing mydomain.com to www.mydomain.com).
- Some of the "bad domains" listed by only 1 or 2 members were actually irrelevant and dead websites posted by spammers. So this analysis allowed us to find a few spammers, and eliminate them!

6. Possible Improvements

There are various ways to improve my methodology and the quality of the results. Here I mention a few:

- Order results by year (showing most recent websites first)
- Perform some real clustering on these websites, using the stars, year and keyword metrics available in my listings.
- Create your own seed keywords list, by extracting all one- and two-tokens keywords found on these 2,500 webpages (nice seed keywords to add are *deep learning*, *Python*, *data* and *visualization*)

- Break down websites into two groups: those containing *data* or *analytic* in the domain name, versus those who don't
- Browse multiple webpages per website (identify internal pages with web crawler)
- Browse multiple external pages per website, to grow your list of 2,500 websites to a much bigger list (make sure the new websites added are analytic-related, use the seed keywords list for this purpose). You can go two levels deep in your external crawling.
- Create and use a segmented seed keywords list (keywords related to visualization, big data, infrastructure, storage, databases, analytics and so on; this will help with website clustering)
- Run the crawler on Hadoop or at least use some distributed architecture
- Run the crawler in batch mode. Allow your script to easily resume if it stops for whatever reasons (Internet goes off etc.) One way to do this is to save the results for each website, one at a time, immediately after crawling it, and produce a log history of all websites that have been crawled, as your script progresses over the website list. This way, you can resume your crawling with a single command, at any time.
- Use recent data only. Some old websites have three stars because they were popular back then, but have now little traffic.
- Handle *https* as well as *http* requests
- Look at keyword density. Rather than checking if *data science* is found on a webpage, look at how many times it is found.

Another similar project -- creating a taxonomy of the most popular data scientists -- can be found here. It is based on keywords found in their LinkedIn profiles.

21. Predicting Home Values

This topic was the subject of a $1.2 million Kaggle competition sponsored by Zillow, see here. Here we show how Zillow could improve his model.

We have published in the past about home value forecasting, see here, and also here and here. In this chapter, I provide specific advice for the competition in question. More specifically, I provide here high-level advice, rather than about selecting specific statistical models or algorithms, though I also discuss algorithm selection in the last section. I believe that designing sound (usually compound) metrics, assessing data quality and taking it into account, as well as finding external data sources to get a competitive edge and for cross-validation, is even more important than the purely statistical modeling aspects.

1. The data

In my neighboring, all homes are valued close to $1.0 million. There are however, significant differences: some lots are much bigger, home sizes vary from 2,600 to 3,100 square feet, and some houses have a great view. These differences are not well reflected in the home values, even though Zillow has access to some of this data.

Regarding my vacation home 90 miles North, there are huge variations (due mostly to home size and view) and the true (real) value ranges from $500k to $1.5 million. The market is much less fluid. But some spots are erroneously listed at $124k: if you look at the aerial picture below (available from Zillow by entering the address), some lots do not have a home, while in reality the house was built and sold two years ago. This outdated data might affect the estimated value of neighboring houses, if Zillow does not discriminate between lots (not built) and homes: you would think that the main factor in Zillow's model is the value of neighboring homes with known value (e.g. following a recent sale.)

So the first questions are:

- How do I get more accurate data?
- How can I rely on Zillow data to further improve Zillow estimates?

We answer these questions in the next section.

2. Leveraging available data, getting additional data

It is possible that Zillow is currently conservative in its home value estimates, putting too much emphasis on the average value in the close neighborhood of the target home, and not enough in the home features. If this is the case, an easy improvement consists of increasing value differences between adjacent homes, by boosting the importance of lot area and square footage in locations that have very homogeneous Zillow value estimates.

Getting competitor data about home values, for instance from Trulia, and blending it with Zillow data, could help improve predictions. Such data can be obtained with a web crawler. Indeed, with distributed crawling, one could easily extract data for more than 100 million homes, covering most of the US market. Other data sources to consider includes

- Demographics, education level, unemployment and household income data per zip code
- Foreclosure reports
- Interest rates if historical data is of any importance (unlikely to be the case here)
- Crime data and school ratings
- Weather data, correlated with home values
- MLS data including number of properties listed (for sale) in any area
- Price per square foot in any area

3. Potential metrics to consider

Many statisticians are just happy to work with the metrics found in the data. However, deriving more complex metrics from the initial features (not to mention obtaining external data sources and thus additional features or 'control' features), can prove very beneficial. The process of deriving complex metrics from base metrics is like building complex molecules using basic atoms.

In this case, I suggest computing home values at some aggregated level called bin or bucket. Here, a bin is possibly a zip code, as a lot of data is available at the zip code level. Then for each individual home, compute an estimate based on the bin average, and other metrics such as recent sales price for neighboring homes, trend indicator for the bin in question (using time series analysis), and home features such as school rating, square footage, number of bedrooms, 2 or 3 cars garage, lot area, view or not, fireplace(s), and when the home was built. Crime stats, household income and demographics are already factored in at the bin level.

Some decay function should be used to lower the impact of sales price older than a few months old, especially in hot markets. If possible, having an idea of the home mix in the neighborhood in question (number of investment properties, family homes, vacation homes, turnover, and rentals) can help further refine the predictions. Seasonality is also an important part of the mix. If possible, include property tax data in your model.

Differences between listed price and actual price when sold (if available,) can help you compute trends at the zip code level. Same with increases or decreases in 'time in market' (time elapsed between being listed, and being sold or pulled from the market.)

4. Model selection and performance

With just a few (properly binned) features, a simple predictive algorithm such as HDT (Hidden Decision Trees - a combination of multiple decision trees and special regression, see chapter 2) can work well, for homes in zip codes (or buckets of zip codes) with 200+ homes with recent historical sales price. This should cover most urban areas. For smaller zip codes, you might consider bundling them by county. The strength of HDT is its robustness and (if well executed) its ability to work for a long time period with little maintenance. This technique also allows you to easily compute CI (confidence intervals) for your estimate, based on bin (zip code) values.

However, chances are that performance, to assess the winner among all competitors, will be based on immediate, present data, just like with any POC (proof of concept.) If that is the case, a more unstable model might work well to win the $1.2 million prize. It is critical to know how performance will be assessed, and to do proper cross-validation no matter what model you use. Cross-validation consists of estimating the value of homes with known (recent) sales price that are not in your training set, or even better, located in a zip code outside your training set. It would be a good idea to use at least 50% of all zip codes in your training set, for cross-validation purposes, assuming you have access to this relatively 'big data'. And having a substantial proportion of zip codes with full 5-year worth of historical data (not just sampled homes) would be great: it would help you assess how well you can make local predictions based on a sample rather than on comprehensive data. If you only have access to a sample, make sure that it is not biased, and discuss the sampling procedure with the data provider.

It is important to know how the performance metrics (used to determine the winner) handle outlier data or volatile zip codes. If it is just a straight average of square of errors, you might need a bit of chance to win the competition, in addition to having a strong methodology -- though being good at predicting the value of expensive homes will also help in this case. Regardless, I would definitely stay away from classic linear models, unless you make them more robust by putting constraints on model parameters (as in the Lasso or pseudo regression, see chapter 1.)

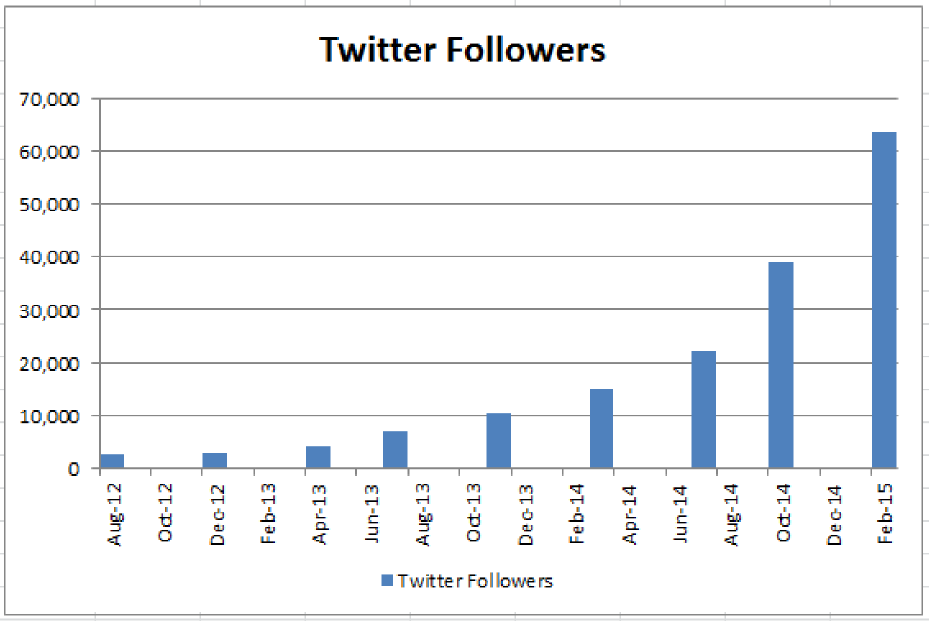
Finally it helps to have domain expertise to win such competitions -- at least to build scalable solutions that will work for a long time.

22. Growth Hacking

In this chapter, we discuss various strategies used to generate exponential traffic growth, while preserving traffic quality, and user loyalty. Our growth hacking engine is a combination of

- Raw data science: getting the right data sets, leveraging them,
- Playing with various tools and API's: designing an automated machine-to-machine communication service between Hootsuite and Twitter / LinkedIn based on insights automatically distilled from the following data sources: (1) data obtained via the Google Analytics API (traffic statistics about 50,000 live DSC articles), and (2) data collected via a web crawler written in Python
- A blend of high-level (strategic) data science and low-level (tactical or operational) data science. In the end, relatively little coding is involved in the process. Domain expertise and smart innovation play a critical role.
- Optimizing parameters of the statistical process used to select articles, create tweets, and schedule them, using experimental design and A/B testing
- Artificial intelligence: detection and removal of articles that are time-sensitive, automated creation of relevant hash-tags for selected tweets, and creation of a taxonomy of all our articles using simple indexing classification scheme
- Smart analytic-driven advertising on Twitter, using a good list of data science thought leaders worth following, as our core data set for advertising purposes. The creation of this list is an interesting data science project in itself.
- Smart analytical and ROI-driven advertising on Google, as well as LinkedIn hacks, to get new members

The results are best illustrated in the graph below representing @AnalyticBridge, the largest data science profile on Twitter, as well as in this article.



1. Growth Hacking: Part I

Here we describe a strategy consisting of tweeting your top articles over a long period of time, to generate incremental traffic. After testing it for one week, we have experienced a 10% growth in traffic. This strategy works well for getting new users, and we believe that it can triple your traffic when fully optimized, though it might reduce user engagement. To get new and loyal subscribers, another strategy is needed: read section 2. This works in fast-growth environments, though you can fine-tune the parameters if applying it to no-growth web sites.

1.1. Strategy

Our DSC network has more than 50,000 live articles at any time, and growing by more than 2,000 new articles per year. Our intern Livan analyzed our Google Analytics statistics, and found more than 2,000 articles each with more than 150 page views - and some with more than 100,000 page views. Back in 2015 when this analysis was performed, our Twitter account had 60,000 followers (growing by 5,000 new followers per month), and we also managed a LinkedIn group with 160,000 members (growing by 6,000 new members per month). We asked ourselves the following question: What if we tweet 25 articles each day, from our list of top 2,000 articles, updated monthly?

The answer, from our first tests, is an immediate 10% traffic boost. We could tweet 100 articles per day from that same list, not just 25. We could tweet from multiple accounts, not just @AnalyticBridge, and we could also post on LinkedIn or Google+ (now dead). With Hootsuite, this process can be fully automated. What would be the impact? Of course there is an optimum: too much tweeting will create dilution. But given the large number of new followers each day, and the fact that the top 2,000 articles could be replaced by entirely new articles after one year (because we produce new articles every day, and we are in the process of automating some postings, such as new books or new salary surveys), is it a clear indicator that 25 tweets a day is well below the optimum. And indeed, we have 50,000+ live articles, so we could tap in the whole list, not just the top 2,000.

Optimizing this tweeting process is discussed later in this chapter. Note that the way tweets work, it is OK if a user sees a same tweet 2 or 3 times over a one-year time period, as long as on average, he sees many tweets from us only one or two times. And given the fact that tweets are short-lived, even with 100 tweets per day (out of a list of 2,000 tweets updated monthly), randomly selected (according to some selection mechanism slightly favoring, new, or very old, or popular, time-insensitive tweets), we should be fine, if we proceed carefully, incrementally, with constant adaptation to new web traffic conditions whenever they occur.

The idea that very old, time-insensitive articles with few (say 150) page views are worth tweeting again today, is because our traffic grew up by 500 percent over the last several years, thanks to the techniques described here. So our older articles were not seen by most of our new visitors. This concept is explained chapter 19 where we discuss traffic decay and how to increase the lifetime and yield of old blog posts. For instance, by having top articles listed in a footer at the bottom of each article. The footer that can be updated at once across thousands of articles, when needed, using an *shtml* include directive in the webpage code, or an iframe to load the adaptive footer stored in one web location.

1.2. Methodology

The process consists of five steps:

- *Step 1*: Producing/updating each month a list of top DSC articles based on our Google Analytics data, including for each article, the total number of page views. Currently, we focus on articles with 150+ page views. If we want to extract much bigger lists, we would need to use the Google Analytics API for data extraction.
- *Step 2*: Scraping DSC (using an home-made web scraper written in Python) to identify in the list created in Step 1, the articles that are still live (not deleted), and for each live article, identify creation date, channel (AnalyticBridge.com, DataScienceCentral.com, BigDataNews.com, DataViZualization.com, Hadoop360.com) and title
- *Step 3*: Data cleaning: removing time-sensitive articles, adding hash tags to titles
- *Step 4*: Statistical modeling: creating a score for each article, based on page views, creation date, and a random number (see details below)
- *Step 5*: The scores attached to each article are based on new simulated random numbers produced every day. Each day, select the top 25 articles based on score, and add them to Hootsuite. Schedule the 25 tweets during the day, over a 4 hour time window corresponding to our peak in US traffic. Hootsuite will automatically generate the shortened URL's to be added in the tweets.

The score can be used to slightly favor (over-tweet) articles that are more recent, or popular. But it is random enough that any article has some chance to eventually be tweeted one day. The score reflects the fact that not all articles are created equal.

The final implementation will consist of a fully automated machine-to-machine communication service (between Google Analytics, Hootsuite, and Twitter), powered by robust black-box analytics, automated machine learning (hash tag creation, detection of time-sensitive articles) and automated, adaptive statistical scoring.

The number of tweets can be slightly adjusted each day (increased, decreased, or change in scoring parameters) as a response to performance. Performance is measured in terms of daily clicks arising from this activity (the stats are readily available from Hootsuite analytics), and the resulting average session duration for traffic coming from Twitter (available from Google Analytics).

1.3. Details about the scoring algorithm

This algorithm is used to score articles based on page views count (denoted as *P*), creation date (denoted as *T* for time), and a random number denoted as *R* (uniform deviate on [0, 1]). Note that older articles tend to have more page views, so *P* and *T* are not independent. The score *S* is computed as follows:

$$S = (b + R)\, P^a \,/\, (T - \text{Offset})^c$$

The parameter *a*, *b*, *c* are chosen so that the top 25 articles selected each day (for tweeting) have, on average, a median *P* (historical page views count) about twice as high as the median *P* computed across all 2,000 articles. This way, we slightly favor popular articles, but not too much. Details are in the spreadsheet described below. Offset is chosen so that *T* = Offset, for our oldest article. You should use the median for *P*, not the average, because it has a Zipf distribution. Note that page view decay occurs, especially for not popular article, though decay is masked by growth for popular articles, see chapter 19.

1.4. Data Sets, Excel spreadsheet

You can download our Excel spreadsheet with 2,000 articles, featuring the following fields, for each article:

- Title
- URL
- Creation Date
- Page View Count
- Channel
- Randomized Score (column I)

The parameters *a*, *b*, and *c* are in cells J2, J3, and K2 respectively. A low value for J3 will produce more random scores. Cross correlations are displayed in cells L1:O4, and the median score for top 25 articles, and for all 2,000 articles, are displayed in cells M8 and M7 respectively.

Note that the cross-correlations are not very useful: even when the correlation between *P* and *S* is as low as 0.04, the median *P* for the top 25 articles (those with highest *S*) is twice as much as the overall median score *S* computed on all articles. This is because traditional correlation is a poor indicator in this context, sensitive to the numerous outliers in the *P* values, since *P* has a Zipf rather than Gaussian distribution.

You can also download a full data set that contains the full text (not just the title) for each article. It is used for clustering articles (see section 3).

1.5. Python Source Code

Our intern Livan wrote some Python code to process Google Analytics reports, and scrape DSC articles to extract relevant fields (creation date, channel, and title.) Download Python code (rename this text file with a .py extension after downloading).

1.6. Next Steps

We can make this system more powerful by

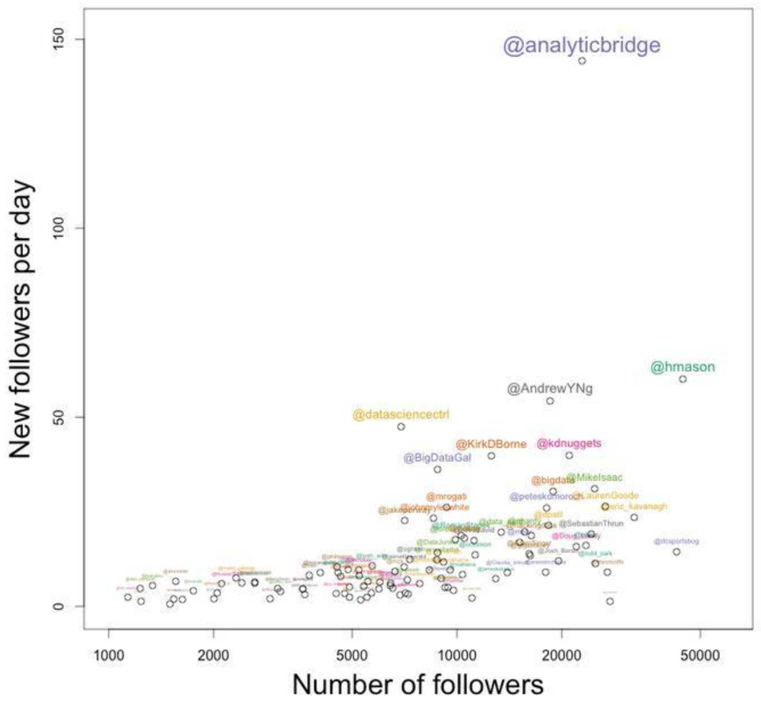
- Automatically removing time-sensitive articles, by detecting tokens in the URL such as *event*, *conference*, or *weekly-digest*
- We could deploy the system not just on Twitter, but on our large LinkedIn group or on multiple Twitter accounts
- Deduping duplicate URL's (sharing same path but different query strings)
- Use top 50,000 articles rather than 2,000
- Automate some of the content production (new books announcements and salary surveys are easy to automatically produce), to boost our number of tweet-able articles

2. Growth Hacking: Part II

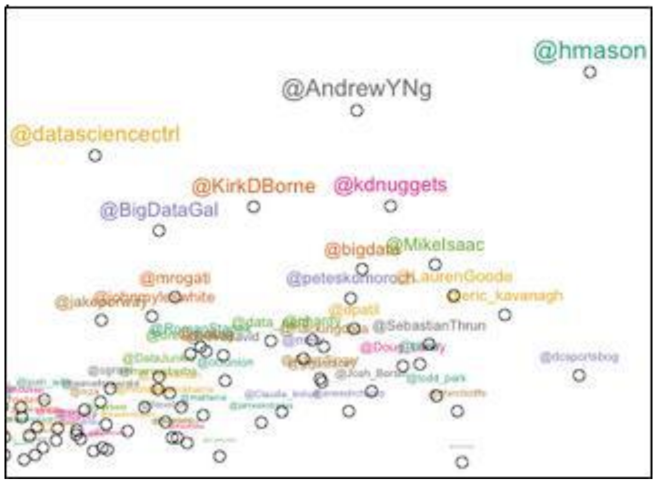
This section quickly describes the other fundamental component required to make our system (described in section 1) work. It is the creation and growth of at least one massive Twitter account, with highly relevant, high value followers, and use of automated tweeting systems. There is a feedback loop in the sense that having a lot of valuable content to tweet, helps generate large volume of good traffic to your website, and helps boost your Twitter growth, which in turn further fuels the traffic growth for your website.

Here, a significant part of our growth (150 new Twitter followers per day) is generated via Twitter advertising: we spend a little more on Twitter than on Google AdWords. With Twitter, it is possible to target US-based profiles (and their followers) that are similar to pre-selected profiles, and you can upload a list of pre-selected profiles when starting your advertising campaigns. Our list has hundreds if not thousands of pre-selected data science profiles. Such lists are easy to find, and regularly published on various websites. But ours also includes top profiles - indeed the very largest, most relevant ones - that are missing in the traditional published lists, as well as people who re-tweet or like our tweets.

The growth and volume of our two main Twitter profiles @analyticbridge and @datasciencectrl, is displayed in the figure below.



Below is a zoom in on the bottom right corner.



Since 2015, the landscape has changed, but not that much, and we still dominate. The strategy described in section 1 delivers more than 1,000 extra clicks per day to our network, at the current low levels (25 tweets per day).

We also use LinkedIn and Google AdWords, but for a different goal: generating new members, US-based in the case of AdWords. But we have encountered a number of issues with AdWords (low conversion), thus we have reduced our budget, optimized our Adwords strategies (adding negative keywords and conversion tracking, more on this coming soon), and shifted money to Twitter and to acquire high quality content. Read my article on 360-degree data science to understand how we blend domain expertise, business hacks, machine learning, engineering, and modern statistical science, to efficiently solve business problems in general. And in particular, to discover how we optimize our bidding strategies for Google keywords (how much to pay for a keyword).

3. Growth Hacking: Part III

Another part of our growth hacking strategy consists of creating new channels, for instance:

- DataViZualization
- DataPlumbing
- BigDataNews
- Hadoop360

One of the challenges is to populate these channels with new content. While we use syndicated feeds for this purpose, we also want to add our own content. One way to do so is to perform a clustering of all our articles, and assign them a category: visualization, data plumbing, big data, Hadoop and so on. Once the articles are categorized, we can publish (re-post) some popular articles from DSC on the appropriate sub-channels.

Here we describe a very simple and highly scalable NLP (natural language processing) technique, called indexation, to perform this clustering task. It works as follows.

Algorithm: categorizing / clustering articles

- *Step 1*: Create a data dictionary (see section 8 in chapter 25) of all one-token and two-token keywords found in all articles (both in the title and in the body of the article). This assumes that you crawled all your articles to extract all the text.
- *Step 2*: Filter / clean results. Ignore keywords with less than 5 occurrences. Check all n-grams of a keyword (*data science* and *science data*) and eliminate n-grams with low frequency, for each keywords
- *Step 3*: Look at top 300 entries, called *seed keywords*. Manually assign seed keywords to top categories. For instance, the top category *data plumbing* will have the following seed keywords: data engineer, data architect, data warehouse, Hadoop, Spark, data lakes, IoT and many more. Don't forget to have a top category called *Unknown*.
- *Step 4*. Based on keywords found in the title and body of an article, assign the article in question to the top category that has the biggest overlap with the article, in terms of seed keywords. Note that keywords found in the title might be assigned a higher weight than those found in the body. Likewise, a different weight can be attached to each seed keyword, in each top category.

I call this technique *indexation* because it is very similar to the creation of a search engine; another word that could be used is ***tagging algorithm***. See also chapters 6 and 20.

Instead of using this algorithm, you can just use customized Google Search for your website, and once installed, search for *data plumbing* to find articles in your website, that are a good fit for the data plumbing category or channel. We've actually implemented it on DSC. We switched to an internal home-made search engine once Google starting displaying (mostly irrelevant or competing) ads in the search results.

Potential improvement

Also add 3-token keywords in your dictionary. For 3 tokens keywords, you have 3! (factorial 3) = 6 *n*-grams. Usually, only one or two of these 6 n-grams will show up in the articles, for any keyword (*data science central* will show up, but *central science data* won't).

4. Conclusions

This DSC growth engine illustrates that data science is not just about programming. Indeed, here, programming is a small part of the project, compared with designing algorithms that efficiently make API's communicate with each other, based on data automatically gathered, with insights automatically extracted, and automatically leveraged. It also shows the limitation of traditional statistical science, with correlations (see the sub-section about the scoring engine) that are useless, and replaced by something else.

It certainly shows that there are different types of data scientists, and that indeed, data science is greater than the sum of its parts. It also shows how business and domain expertise are critical. For instance, if you don't know about the Twitter advertising capabilities or the Hootsuite or BufferApp platform, you will never even think of doing this kind of stuff, no matter how much you know about coding and algorithms, thus missing on a big opportunity. If you work in a bigger organization, of course finding and convincing the right person to start a project like this one, is a challenge, no matter how much business savvy you are. But my experience is that big organizations tend to hire specialists rather than people like me.

23. Time Series and Growth Modeling

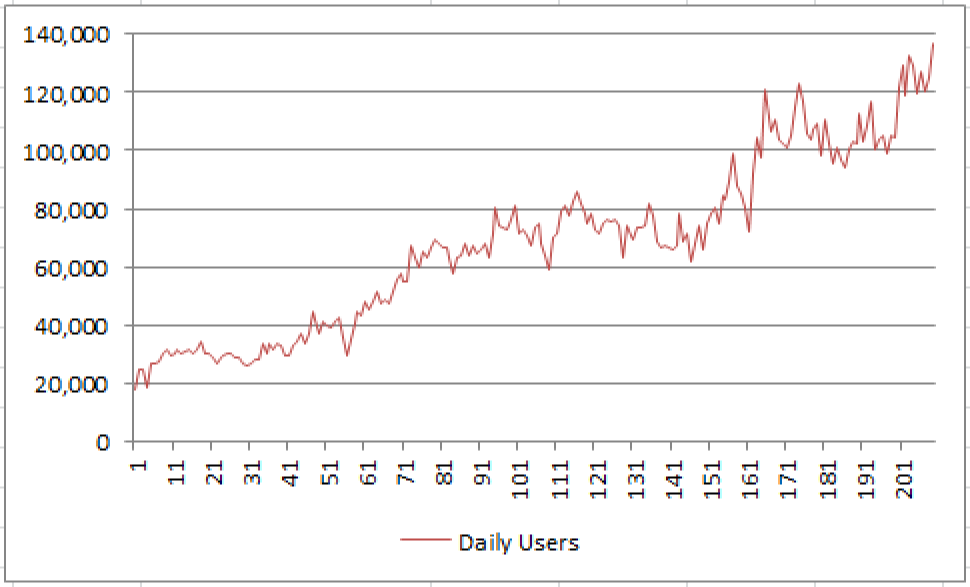
Time series models are studied throughout this book: chapters 12 and 13, section 2 in chapter 28, section 12 in chapter 25, appendix A and B. Here our perspective is purely business related and focused on growth modeling, especially to make sound predictions in the presence of growth. It is more important to understand what drives growth and its internal mechanics, than using sophisticated models, in order to make sound forecasts.

Many times, complex models are not enough (or too heavy), or not necessary, to get great, robust, sustainable insights out of data. Deep analytical thinking may prove more useful, and can be done by people not necessarily trained in data science, even by people with limited coding experience. Here we explore what we mean by deep analytical thinking, using a case study, and how it works: combining craftsmanship, business acumen, the use and creation of tricks and rules of thumb, to provide sound answers to business problems. These skills are usually acquired by experience more than by training, and data science generalists (see here how to become one) usually possess them.

This chapter is targeted to data science managers and decision makers, as well as to junior professionals who want to become one at some point in their career. Deep thinking, unlike deep learning, is also more difficult to automate, so it provides better job security. Those automating deep learning are actually the new data science wizards, who can think out-of-the box. Much of what is described in this chapter is also data science wizardry, and not taught in standard textbooks nor in the classroom. By reading this tutorial, you will learn and be able to use these data science secrets, and possibly change your perspective on data science. Data science is like an iceberg: everyone knows and can see the tip of the iceberg (regression models, neural nets, cross-validation, clustering, Python, and so on, as presented in textbooks.) Here I focus on the unseen bottom, using a statistical level almost accessible to the layman, avoiding jargon and complicated math formulas, yet discussing a few advanced concepts.

1. Case Study: The Problem

The real-life data set investigated here is a time series with 209 weeks' worth of observations. The data points are the number of daily users, averaged per week, for a specific website, over some time period. The data was extracted from Google Analytics, and summarized in the picture below. Some stock market data also shows similar patterns.



The data and all the detailed computations are available in the interactive spreadsheet provided in the last section. Below is an extract.

Week Index	0	1	2	3	4	5	6	7	8	9
Daily Users	18,196	24,692	24,539	18,554	27,030	26,841	27,323	30,417	31,886	29,808
Week Index	15	16	17	18	19	20	21	22	23	24
Daily Users	30,125	31,872	34,521	30,311	29,937	28,793	26,557	28,788	29,619	30,269
Week Index	100	101	102	103	104	105	106	107	108	109
Daily Users	71,229	72,711	70,508	67,543	73,579	74,856	67,777	64,146	58,972	70,234
Week Index	180	181	182	183	184	185	186	187	188	189
Daily Users	110,763	102,592	95,410	101,142	96,730	94,172	100,433	103,354	102,723	112,588

1.1. Business questions

We need to answer

- Whether there is growth over time, in the number of visiting users,
- Whether it can be extrapolated to the future (and how),
- What kind of growth do we see (linear, or faster than linear)
- Whether can we explain the dips, and avoid them in the future.

As in any business, growth is driven by a large number of factors, as every department tries its best to contribute to the growth. There are also forces going against the growth, such as reaching market saturation, market decline or competition. All the positive and negative forces combine together and can create a stable, predictable growth pattern, whether linear, exponential, seasonal, or a combination. This can be approximated by a

Gaussian model, by virtue of the central limit theorem. However, in practice, it would be much better to identify these factors to get a much better picture, if one wants to make realistic projections and measure the cost of growth.

Before diving into original data modeling considerations (data science wizardry) in section 3, including spreadsheets and computations, we first discuss general questions (deep analytical thinking) that should be addressed whenever such a project arises. This is the purpose of the next section.

2. Deep Analytical Thinking

Any data scientist can quickly run a model and conclude that there is a linear growth in the case discussed in section 1, and make projections based on that. However, this may not help the business if the projections, as we see so frequently in many projects, work for only 3 months or less. A deeper understanding of the opposite forces at play, balancing out and contributing to the overall growth, is needed to sustain the growth. And maybe this growth is not good after all. That's where deep analytical thinking comes in play.

Of course, the first thing to ponder is whether this is a critical business question, coming from an executive wondering about the health of its business (even and especially in good times,) or whether it is a post-mortem analysis related to a specific, narrow, tactical project. We will assume here that it is a critical, strategic question. In practice, data scientists know about the importance of each question, and treat them accordingly with the appropriate amount of deep thinking and prioritization. What I discuss next applies to a wide range of business situations.

2.1. Answering hidden questions

It is always good for a data scientist, to be involved in business aspects that are data related, but go beyond coding or implementing models. This is particularly true with small businesses, and it is one aspect of data science that is often overlooked. In bigger companies, this involves working with various teams, as a listener, challenger, and in an advisory role. The questions that we should ask are broken down below in three categories: business, data, and metrics related.

Business questions:

- Is your company pursuing the correct type of growth? Is it growing in the right segments? Is the growth shifting in the wrong direction? Do we now attract an audience that is not converting well (low ROI) or with high churn rate (low customer lifecycle value, high cost of user acquisition.) The data scientist is well positioned to access the relevant data and analyze it to answer this question.

- Is top management too much focused on bad growth? That is, growing for the sake of it to show to shareholders? There is good growth and bad growth. In many businesses, *some* bad growth (growth for the sake of it) is needed to impress clients, shareholders, employees, and because growth numbers from competitors are also fueled partly by bad growth. That is why you want to show that your company is growing as fast as your competition. Good growth, to the contrary, is focused on long terms outcomes. However, now that granular data from most companies is widely available or can be purchased and analyzed by experts, it is becoming more difficult to fake the growth. Anyway, when analyzing statistics, you must be able to discriminate between good and bad growth.

- What external factors impact the bottom line metrics? Competition and market trends are two of them. Knowing that a competitor just received a new round of funding and is spending it on advertising, can be very valuable to gain insights. Or in our example, the big dip corresponds to holiday traffic in December.

- What internal factors are at play and "influencing" your data? It could be increased marketing efforts by your company, a website that was made much more efficient, some business acquisition or new products, the definition of a metric that was changed internally (with impact on measured numbers). The data scientist should be informed about these events, and indeed, proactively ask questions when data trends are seen but cannot be explained. Even when data sounds stable, it could be the effect of two sources, one negative, and one positive, canceling out. Always be curious about what is happening in your company, with your competitors and the market in general.

Data questions:

- Are we gathering data from external sources, to validate internal data? In our case, data from Google Analytics can be wrong. Having an external source will help you pinpoint discrepancies and understand what is exactly measured by the various sources. A tool such as Alexa not only provides an alternate source of measurements, but it also provides data points about competition.

- Is some data duplicated, missing, corrupt, or not available? Are you working with the IT and BI team to collect the right data, get it properly summarized, accessible via dashboards or straight from databases, and archived appropriately, locally or externally? Do you maintain a data log that lists all changes to data over time?

- Do you know the biggest mechanisms introducing biases and errors in your data? In our case, Google Analytics is sensitive to smart bots generating artificial traffic, to websites not being tracked or tagged properly, and to new advertising campaigns being launched, introducing shifts in geolocation and traffic quality. Address all these issues with the right people in your company. Sometimes it requires having access to additional data.

Question about the metrics:

- Are you collecting the most useful metrics? What important metrics are missing or would be useful to have? Do we you enough granularity? Do you focus on the right metrics? New users might be more important than total users. Page view numbers are easily manipulated by third parties and thus less reliable. Session duration may be meaningless if users spend a lot of time watching videos on your website. A lot of traffic from US is not good if it is from demographic segments not bringing any value. High traffic numbers might not be good if users complain about the poor quality of your content.

- Finding proxy metrics when the exact ones are not available. For instance, zip code data could be used instead of income. When creating web forms, adding mandatory fields could result in more useful databases and better targeting, though changes also impact the data and create back-compatibility issues, making comparisons difficult when analyzing time series.

- A simple question such as the one discussed in section 1, is too generic. You must analyze growth in various segments, and sometimes, you may discover segments that need to shrink rather than grow. For instance, a website that accepts credit card transactions, written in English, might not be appropriate for countries where credit card use is non-existent, or in locales that can sue you because your content is in English rather than in the local, mandatory language.

- Should you use a longer time window (if available) to get a better picture, assuming the data is consistent over time? Or monthly rather than weekly data? How frequently should this analysis be done? Can it be automated if done frequently enough? Should it be included in dashboard reporting? Which charts to use to communicate the insights visually, with maximum impact and value to the stakeholder?

In the next section, we focus on the modeling aspects, offering different perspectives on how to better analyze the type of data discussed in our case study.

3. Data Science Wizardry

We focus on the problem and data presented in section 1, providing better ad-hoc alternatives (rarely used in a business setting) to regression modeling. This section is somewhat more technical.

Even without doing any analysis, the trained eye will recognize a linear trend for the growth, in the time series. Even with the naked eye, you can further refine the model and see three distinct patterns: a steady growth initially, followed by a flat plateau, and then the growth becoming fairly steep at the end. The big dip is caused by the holiday season. At this point, one would think that a mixed, piece-wise model, involving both linear and super-linear growth, represents the situation quite well. It takes less than 5 seconds to come to that conclusion.

The idea to represent the time series as a mixed stochastic process -- a blend of linear and exponential models, depending on the time period -- is rather original and reminiscent of mixture models (see chapter 11). Model blending is also discussed in chapter 2. However, in this section, we consider a simple parametric model. But rather than traditional model fitting, the technique discussed here is based on simulations, and should appeal to engineering and operations research professionals. It has the advantage of being easy to understand yet robust.

The idea is as follows, and it will become clearer in the illustration that follows:

3.1. Generic algorithm

- *Step 1*. Simulate 100 realizations, also called instances, of a stochastic process governed by a small number of easy-to-interpret parameters, each realization with 209 data points as in the original data set, with same starting and end values as in the observed data (or same mean and variance.) The parameters are set to fixed values.
- *Step 2*. Compute the estimated values (averaged over the 100 simulations) of some business quantities of interest, for instance the number of weeks followed by an increase in users, the average week gap between two increases, the average dip depth and width or number of dips (same with spikes), auto-correlations and so on. These quantities are called indicators.
- *Step 3*. Compute the error between the indicator values computed on the observed time series, and those estimated on the simulations.
- *Step 4*. Repeat with a different set of parameters until you get a fit that is good enough.

A potential improvement, not investigated here, is to consider parameters that change over time, acting as if they were priors in a Bayesian framework. It is also easy to build confidence intervals for the indicators, based on the 100 simulations used for each parameter set. This makes sense with bigger data sets, and it can be done even without being a statistical expert (a software engineer can do it.)

3.2. Illustration with three different models

I tested the following models (stochastic processes) to find a decent fit with the data, while avoiding over-fitting. Thus the models used here have few, intuitive parameters. In all cases, the models were standardized to provide the desired mean and variance associated with the observed time series. The models described below are the base models, before standardization.

Model #1:

This is a random walk with unequal probabilities, also known as a Markov chain. If we denote the average daily users at week t as X_t, then the model is defined as follows: $X_{t+1} = X_t - 1$ with probability p, and $X_t = X_t + 1$ with probability $1 - p$. Since we observe growth, the parameter p must be between 0 and 0.5. Also, it must be strictly above 0 to explain the dips, and strictly below 0.5 otherwise there would be no growth and no decline (on average), over time. Note that unlike a pure random walk (corresponding to $p = 0.5$), this Markov chain model produces deviates X_t that are highly auto-correlated. This is fine because, due to growth, the observed weekly numbers are also highly auto-correlated. A parameter value around $p= 0.4$ yields the lag-1 auto-correlation found in the data.

Model #2:

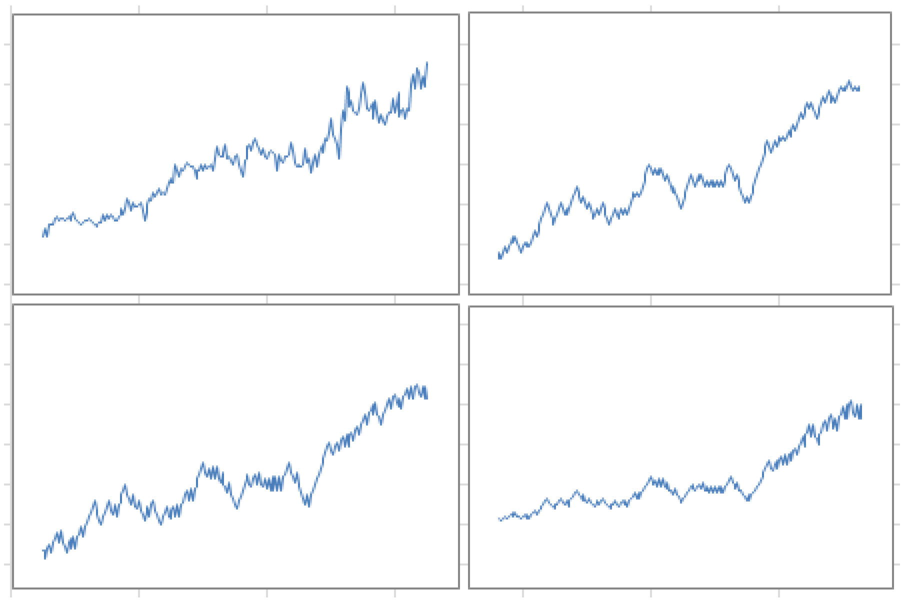
This model is a basic auto-regressive (AR) process, defined as $X_{t+1} = qX_t + (1-q)X_{t-1} + D_t$, with the parameter q between 0 and 1, and the D_t's being independent random variables equal to -1 with probability p, and to +1 with probability $1 - p$. It also provides a similar lag-1 auto-correlation in the $\{ X_t \}$ sequence, but in addition, now the sequence $Y_t = X_{t+1} - X_t$ also exhibits a lag-1 auto-correlation. Indeed, there is also in the data, a lag-1 auto-correlation in the $\{ Y_t \}$ sequence. A parameter value around $q = 0.8$ together with $p = 0.4$, yields that auto-correlation. Note that with the Markov chain (our first model), that auto-correlation (in the $\{ Y_t \}$ sequence) would be zero. So the AR process is a better model. An even better model would be an AR process with three parameters.

Model #3:

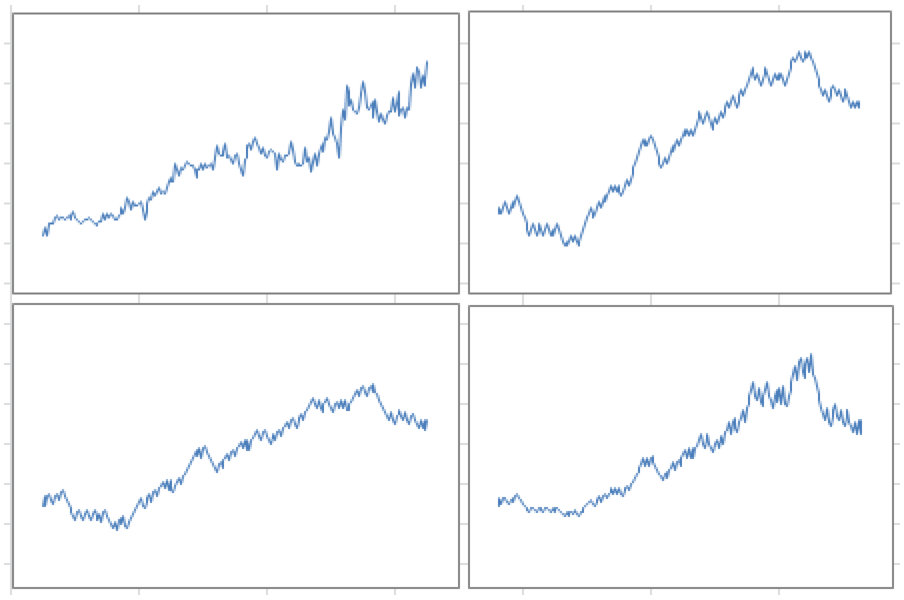
The two previous models can only produce linear growth trends. In order to introduce non-linear trends, we introduce a new model which is a simple transformation of the AR process. It is defined as $Z_t = \exp(rX_t)$, where $\{ X(_t) \}$ is the AR process. In addition to the parameters p and q, it has an extra parameter r. Note that when r is close to zero, it behaves almost as an AR process (after standardization), at least in the short term.

3.3. Results

The picture below shows the original data (top left), one realization of a Markov chain with $p = 0.4$ (top right), one realization of an AR process with $p =0.4$ and $q = 0.6$ (bottom left), and one realization of the exponential process with $p = 0.4$, $q = 0.6$ and $r = 0.062$. By one realization, we mean any one simulation among the 100 required in the algorithm.



The picture below features the same charts, but with another realization (that is, another simulation) of the same processes, with the same parameter values. Note that the dips and other patterns do not appear in the same order or at the same time, but the intensity, length of dips, overall growth, and auto-correlation structures are similar to those in the first picture, especially if you extend the time window from 209 weeks to a few hundred weeks, for the simulations: they are in the same confidence intervals. If you try many simulations and compute these statistics each time, you will have a clear idea of what these confidence intervals are.



Overall -- when you look at 100 simulations, not just two -- the exponential model with a small value of *r* provides the best fit for the first 209 weeks, with a nearly linear growth at least in the short term. However, as mentioned earlier, a piece-wise model would be best. The AR process, while good at representing a number of auto-correlations, seems too bumpy, and dips are not deep enough; a 3-parameter AR process can fix this issue. Finally, model calibration should be performed on test data, with model performance measured on control data. We did not perform this cross-validation step here, due to the small data set. One way to do it with a small data set is to use odd weeks for the test, and even weeks for the control. This is not a good approach here, since we would miss special week-to-week auto-correlations in the modeling process.

Download the spreadsheet, with raw data, computations, and charts. Play with the parameters!

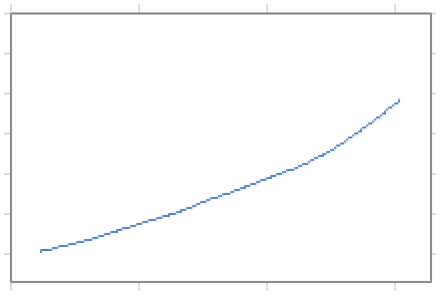
4. A few data science hacks

Here I share a few more of my secrets.

The Markov chain process can only produce a linear growth. This fact might not be very well known, but if you look at Brownian motions (the time-continuous version of these processes) the expectation and variance over time is well studied and very peculiar, so it can only model a narrow range of time series. More information on this can be found in the first chapters of my previous book, here. In this chapter, we overcome this obstacle by using an exponential transformation.

Also growth is usually non-sustainable long-term, and can create bubbles that eventually burst -- one thing that your model may be unable to simulate. One way to mitigate this effect is to use models with constrained growth, in which growth can only go so far and is limited by some thresholds. One such model is presented in my previous book, see chapter 3 here.

Finally, model fitting is usually easier when you do it on the integrated process (see chapter 2 in my previous book.) The integrated process is just the cumulative version of the original process, easy to compute, and also illustrated in my spreadsheet. The data / model fit, measured on the cumulative process, can be almost perfect, see picture below representing the cumulative process associated with some simulations performed in the previous sub-section.

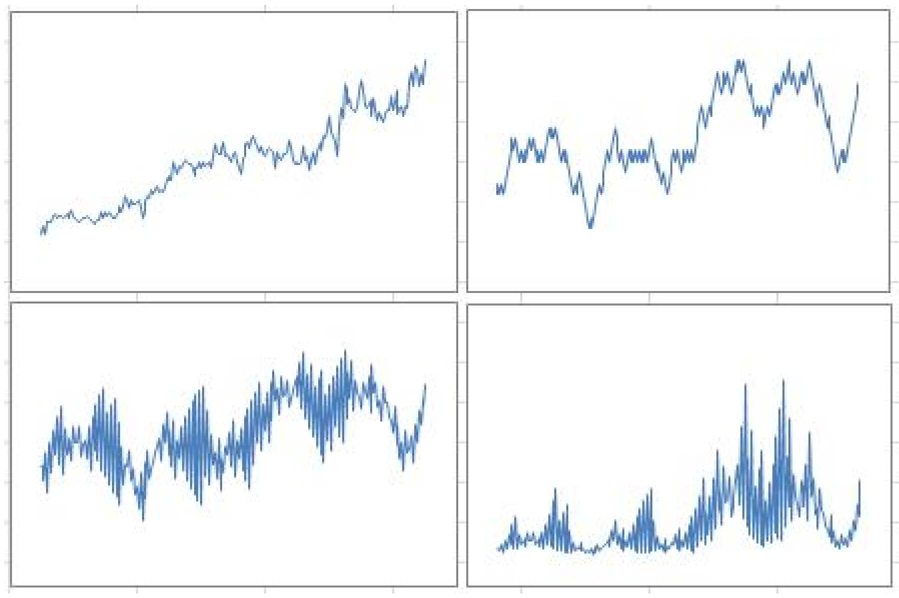


In the above chart, the curve is extremely well approximated by a second-degree polynomial. Its derivative provides the linear growth trend associated with our data. This concept is simple, though I have never seen it mentioned anywhere:

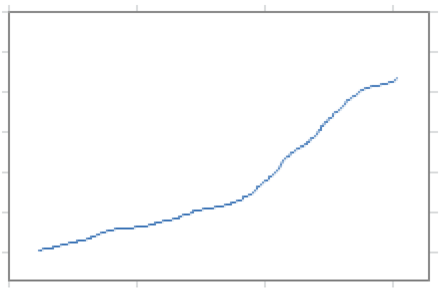
- Use cumulative instead of raw data
- Perform model fitting on the cumulative data
- The derivative of the function (best fit) attached to the cumulative process, provides a great fit with the raw data.

The cumulative function acts as a low-pass filter on the data, removing some noise and outliers.

Below is another picture similar to those presented earlier, but with a different set of parameter values. It shows that despite using basic models, we are able to accommodate a large class of growth patterns.



And below is the cumulative function associated with the chart in the bottom right corner in the above picture: it shows how smooth it is, despite the chaotic nature of the simulated process.



In some other simulations (not illustrated here, but you can fine tune the model parameters in the spreadsheet to generate them) the charts present spikes like Dirac distributions and are very familiar to physicists and signal processing professionals.

24. Designing Better Algorithms

In this chapter, using a few examples and solutions, I show that the "best" algorithm is many times not what data scientists or management think it is. As a result, too many times, misfit algorithms are implemented. Not that they are bad or simplistic. To the contrary, they are usually too complicated, but the biggest drawback is that they do not address the key problems. Sometimes they lack robustness, sometimes they are not properly maintained (for instance they rely on outdated lookup tables), sometimes they are unstable (they rely on a multi-million rule system), sometimes the data is not properly filtered or inaccurate, and sometimes they are based on poor metrics that are easy to manipulate by a third party seeking some advantage (for instance, click counts are easy to fake.) The solution usually consists in choosing a different approach and a very different, simple algorithm - or no algorithm at all in some cases.

1. Five Case Studies

Here I provide a few examples, as well as an easy, low-cost, robust fix in each case.

Many times, the problem is caused by data scientists lacking business understanding (they use generic techniques, and lack domain expertise) combined with management lacking basic understanding of analytical, automated, optimized (semi-intelligent, self-learning) decision systems based on data processing. The solution consists of educating both groups, or using hybrid data scientists (I sometimes call them business scientists) who might not design the most sophisticated algorithms, but instead the most efficient ones given the problems at stake - even if sometimes it means creating ad-hoc solutions. This may result in simpler, less costly, more robust, more adaptive, easier to maintain, and generally speaking, better suited solutions.

- **Click fraud detection**: This is an old problem, yet publishers using affiliates to generate traffic (including Google and its network of partners) still deliver vast amounts of fraudulent clicks. While these companies have become much smarter about pricing (very lowly) these worthless clicks, better and easier solutions exist. For instance, pricing per keyword per day rather than per click, or targeting specific people/audiences to make it difficult to create fake traffic (and at the same time increasing relevancy and thus ROI both for the advertiser and the ad network.) For instance, both Facebook and Twitter allow you to target friends of friends, or profiles similar to pre-specified people. The issue with the pay-per-click algorithms (specifically, fraud detection) is not so much the fact that the algorithms miss a lot of fraud, but rather caused by the business model which is flawed by design. My solution consists of changing the business model.

- **Ad matching or relevancy algorithms**: We've all seen too many times ads that are irrelevant to us - it is a waste of money for the advertiser and the ad network. Ad matching algorithms (or generally speaking, relevancy algorithms) aim at optimally serving the right ads (or content) to the right user on the right page at the right time. When several ads compete for a spot and can all be displayed, they need to be displayed in the right order (on a specific page, to a specific user.) Such algorithms can greatly be improved by assigning categories both to pages, users and ads, in order to optimize the match, even in the absence of a search query. This is described in this article, and modern techniques rely on tagging or indexation algorithms. (see chapter 4.) Such indexation algorithms are conceptually very simple and robust, and can quickly create taxonomies on big data, to help solve the problem. Another search engine algorithm that can benefit from substantial improvements is content attribution: assign the content to the original source (by displaying it at the top in search results), rather than to an authorized copy or worse, to a plagiarist. Click here for details; the solution might be as easy as pre-sorting index entries (for a same keyword and identical content) by time stamps. More on search engine technology here.

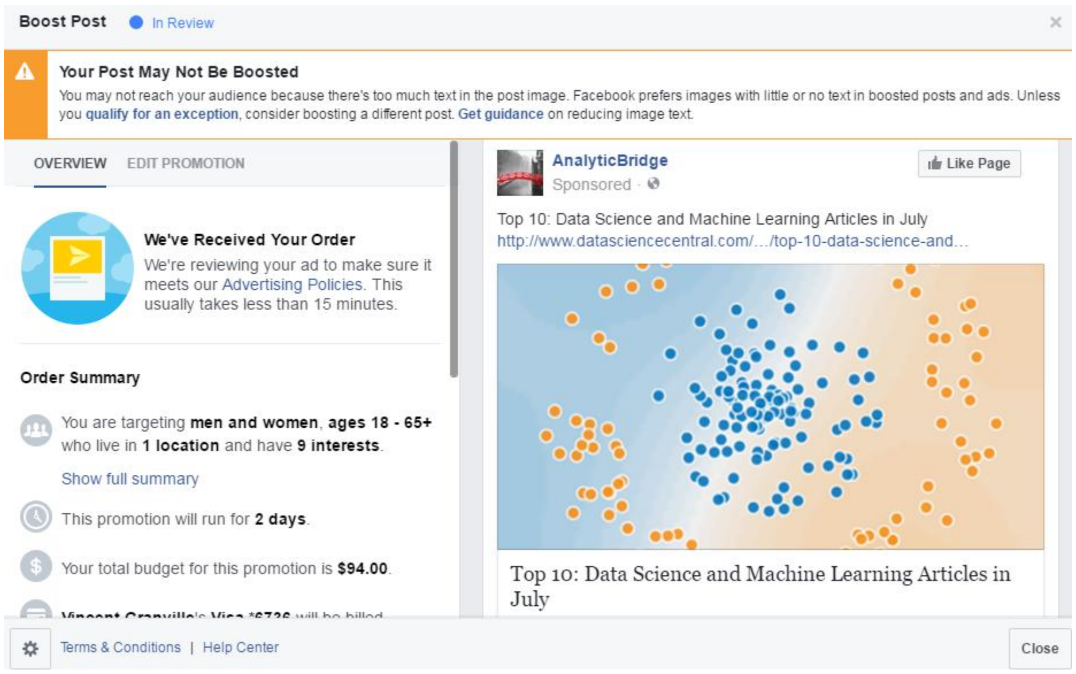
- **Optimum pricing**: Just like using the same drug for all patients (to cure a specific ailment) is a poor strategy, using the same price (for a specific product) for all customers may not be the best solution. Optimum pricing varies based on time, sales channel, and customer. I described this concept in an article on hotel rooms pricing.

- **Fake reviews detection**: Product and book reviews are notoriously biased as authors get reviews from friends, and blackmailers try to get your money to post good reviews about your product, or otherwise will write bad reviews. Read this article for details. The bulk production of fake reviews is indeed a striving business in its own (if executed properly with the right algorithms as described here.) It negatively impacts all websites (such as Amazon) that rely on product reviews to increase sales and attract users: it is a trust issue. The concept itself is subject to conflicts of interests - good reviews supposedly increasing sales, are thus encouraged or given more weight. So here we are facing a business flaw rather than poor detection algorithms. The solution is having professionals write the reviews and then the problem of fake reviews almost disappears - no need for an algorithm to handle it. If however you really want to implement user-based product reviews on your e-store, here is the way to do it right: as in the relevancy algorithm described above, assign categories to each user, each product and each reviewer. When there is a strong match (the user, the reviewer and the product categories all match) assign a high score to the product review in question. Eliminate reviews that are too short. First-time reviewers might be assigned a lower score. Then compute a weight for each star rating assigned to a product, by summing up all the individual scores for the star rating in question, possibly putting more emphasis on recent ratings. The global rating is the weighted sum of star ratings, for the product in question. This is far better than a flat average of the star ratings regardless of the quality of the review or reviewer, which is what Amazon is still doing.

- **Image recognition (Facebook ads)**: This is indeed a funny algorithm. As a Facebook advertiser promoting data science articles, most images in my ads are charts and do not contain text. For whatever business reason (probably an archaic rule invented long ago and never revisited) Facebook does not like postings (ads in particular) in which the image contains text. Such ads get penalized: they are displayed less frequently, and cost more per click; sometimes they are just rejected. Most of my ads are erroneously flagged as containing text, see Figure 1 for a typical example. Note that the Facebook algorithm (to detect text in images) processes a large number of ads in near real time, thus it must be rudimentary enough to make decisions very fast -- although it is very easy to use a distributed, Map-Reduce architecture to process these ads. The solution to this issue: get rid of your algorithm entirely, instead use a much better relevancy metric (rather than whether or not the image contains text): click-through rate. The computation is straightforward, though you might need an algorithm to detect and filter out fraudulent or robotic clicks. More on the text detection algorithm in the section below, where a simple, efficient solution is offered to advertisers facing this problem. Of course you could tell me to put arbitrary, irrelevant pictures of people, mountains, vegetables, or lakes in all my ads, to pass muster, but that is not the point -- it might backfire and data scientists are genuinely interested in ... charts. Read the next section for more details. Another faulty algorithm that I will analyze in a future article is the one used to detect posts (on Twitter or Facebook) that violate editorial policies against hate speech, bullying or raunchy language. This algorithm is so bad that it caused Walt Disney to pass on buying Twitter. I wouldn't be surprised if it relies on the Naive Bayes technique - still currently in use in many (poor) spam detection algorithms.

1.1. More about the Facebook ad processing system

The first four cases have been discussed in various articles highlighted above, so here I focus on the last example: the image recognition algorithm used by Facebook to detect whether an image contains text or not, to assess ad relevancy -- and best illustrated in Figure 1. This algorithm eventually controls to a large extent, the cost and relevancy associated with a specific ad. I will also briefly discuss a related algorithm used by Facebook, one that also needs significant improvement. I offer solutions both for Facebook (to nicely boost its revenue yet boost ROI for advertisers at the same time), as well as solutions for advertisers facing this problem, assuming Facebook sticks with its faulty algorithms.


Boost Post ● In Review ×

Your Post May Not Be Boosted
You may not reach your audience because there's too much text in the post image. Facebook prefers images with little or no text in boosted posts and ads. Unless you **qualify for an exception**, consider boosting a different post. **Get guidance** on reducing image text.

OVERVIEW EDIT PROMOTION

We've Received Your Order
We're reviewing your ad to make sure it meets our Advertising Policies. This usually takes less than 15 minutes.

Order Summary

You are targeting **men and women**, **ages 18 - 65+** who live in **1 location** and have **9 interests**.

Show full summary

This promotion will run for **2 days**.

Your total budget for this promotion is **$94.00**.

Vincent Granville's Visa *6736 will be billed

AnalyticBridge
Sponsored ·

Like Page

Top 10: Data Science and Machine Learning Articles in July
http://www.datasciencecentral.com/.../top-10-data-science-and...

Top 10: Data Science and Machine Learning Articles in July

Terms & Conditions | Help Center

Close

Figure 1: *Facebook image recognition algorithm thinks the above image contains text!*

Solution for advertisers

For each article that you want to promote on Facebook, starts with a small budget, maybe as small as $10 spread over 7 days, and target a specific audience. Do it for dozens of articles each day, adding new articles all the time. Regularly check articles that exhausted their ad spend; boost (that is, add more dollars of ad spend) to those that perform well. Performance is measured as the number of clicks per dollar of ad spend. All this can probably be automated using an API.

Solution for Facebook

Eliminate the algorithm that is supposed to detect text in images associated with ads. Instead focus on click-through rate (CTR) like other advertising platforms (Google, Twitter.) Correctly measure impressions and clicks to eliminate non-human traffic, in order to compute an accurate CTR.

Predicting reach based on ad spend

Facebook provides statistics to help you predict the reach for a specific budget (ad spend) and audience, but again, the algorithm doing this forecast is faulty, especially when you try to "add budget" prior to submitting your ad. Google also provides forecasts that in my experience are significantly off. I believe the problem is that for small buckets of traffic, the strength of this forecast is very weak. While confidence intervals are provided, they are essentially meaningless. The solution to this problem is to either provide the strength of the forecast (I call it *predictive power*, see chapter 4), or *not* provide a forecast at all: the advertiser can use the solution offered in the previous paragraph to optimize her ad spend. And if Facebook or Google really want to provide confidence intervals for their forecasts, they should consider this model-free technique (see also chapters 15 and 16) that does not rely on the normal distribution: it is especially fit for small buckets of data that have arbitrary, chaotic behavior.

2. Why so many Machine Learning Implementations Fail?

You would think that machine learning simply does not work, at least not as advertised. Here, I actually claim that this is not the case, further explaining what the issues might be, and in short, that machine learning might not be the culprit.

It seems that the issues appear in situations that are not critical - such as an ad badly targeted, a racist tweet that goes undetected, or a piece of fake news that goes viral. You don't hear stories about planes falling down because of poor auto-pilot systems, themselves powered by faulty machine learning algorithms.

So I classified machine learning (ML) implementations in four categories:

- Implementations that work well: for instance, automated cars, automated piloting (planes)
- Implementations that work for a while: high-frequency trading, with too much reliance on automation.
- Implementations that work more or less: Google search, ad targeting (by top companies), home price or weather forecasts, fraud detection.
- Implementations that do not work: spell check (absolutely atrocious for multi-lingual people), fake news detection, fake reviews detection, detection of illegal tweets.

I believe most implementations fall in the third category. Of course, we only see the fourth category (just like when you read the news: you only hear about people who die, not about people who get born.)

2.1. The fake news issue

I am not even sure that fake news detection is not working. Sure, fake news runs wild on Facebook, Google and everywhere. But they do generate traffic, and thus dollars, at least in the short term. There are two factors at play here:

- Politicians and other people placing fake news in automated news feed systems -- I call it news feed hijacking; if they use machine learning algorithms to avoid detection, and they beat Facebook, then it is not a failure of machine learning; it shows that the fraudsters have better machine learning tools.
- Facebook must decide between too many false positives (a real piece of news identified by error as fake), or false negatives (an undetected piece of fake

news.) Because false negatives are associated with increased revenue, they might be favored by the algorithm.

But maybe the biggest challenge here is how to define fake news in the first place. If not properly defined, it cannot be identified. It is indeed a very fuzzy concept.

2.2. When machine learning is used as a scapegoat

Here are a few reasons why we run into these problems.

- Internal business politics at Facebook, resulting in great algorithms not being used or used improperly.
- Algorithms/business rules (embedded into algorithmic systems) that are not revisited as needed, or at the mercy of unqualified people for maintenance (software engineers not working with data scientists.)
- Teams not collaborating effectively (e.g. data scientists vs software engineers vs business people.)
- Algorithms tested and prototyped on small data (say on 1% of all ads) thus missing a lot.
- Those criticizing only see the bad stuff, not the good stuff, yet overall these "flawed" algorithms produce good enough value for shareholders.
- Even in my article where I criticize some Facebook algorithms, I still consider and use Facebook as the best advertising platform for us.
- Some of this might be dictated by top executives. Most of what I see on Facebook is unidirectional (politically speaking) as if there is a political agenda. It is as if Facebook tries to influence people. It could be caused by Bay Area software engineers having their algorithms favoring posts or ads that they tend to agree with, with or without executives knowing about it.
- Even in the example where Facebook's machine learning technology for being unable to recognize pictures containing text, despite receiving threatening messages about my ads not running because being (erroneously) flagged as containing pictures with embedded text, indeed my ads are sometimes delivered without any problems, as if the message is ignored by the system itself.

3. Twenty four tips for better data science

Here I share a few general ideas to make data science more efficient and bring increased value and return.

- Leverage external data sources: tweets about your company or your competitors, or data from your vendors (for instance, customizable newsletter eBlast statistics available via vendor dashboards, or via submitting a ticket)
- Nuclear physicists, mechanical engineers, and bioinformatics experts can make great data scientists.

- State your problem correctly, and use sound metrics to measure yield (over baseline) provided by data science initiatives.
- Use the right KPIs (key metrics) and the right data from the beginning, in any project. Changes due to bad foundations are very costly. This requires careful analysis of your data to create useful databases.
- Fast delivery is better than extreme accuracy. All data sets are dirty anyway. Find the perfect compromise between perfection and fast return.
- With big data, strong signals (extremes) will usually be noise. See workaround in chapter 27.
- Big data has less value than useful data.
- Use big data from third party vendors, for competitive intelligence.
- You can build cheap, great, scalable, robust tools pretty fast, without using old-fashioned statistical science. Think about model-free techniques, explored in chapters 15 and 16.
- Big data is easier and less costly than you think. Get the right tools! Here's how to get started.
- Correlation is not causation. Chapter 27 discusses this issue. Read also this blog and this book.
- You don't have to store all your data permanently. Use smart compression techniques, and keep statistical summaries only, for old data. Don't forget to adjust your metrics when your data changes, see section 7 in chapter 25.
- A lot can be done without databases, especially for big data.
- Always include EDA and DOE (exploratory analysis / design of experiment) early on in any data science projects. Always create a data dictionary (section 8 in chapter 25.). See also the life cycle of any data science project (see section 13 in chapter 28.)
- Data can be used for many purposes:
  - quality assurance
  - to find actionable patterns (stock trading, fraud detection)
  - for resale to your business clients
  - to optimize decisions and processes (operations research)
  - for investigation and discovery (IRS, litigation, fraud detection, root cause analysis)
  - machine-to-machine communication (automated bidding systems, automated driving)
  - predictions (sales forecasts, growth and financial predictions, weather)
- Don't dump Excel. Embrace light analytics.
- Data + models + gut feelings + intuition is the perfect mix. Don't remove any of these ingredients in your decision process.
- Leverage the power of compound metrics: KPI's (key performance indicators) derived from database fields. These KPI's have a far better predictive power (see chapter 4) than the original database metrics. For instance your database might include a single keyword field but does not discriminate between user query and search category (sometimes because data comes from various sources and is

blended together). Detect the issue, and create a new metric called keyword type - or data source. Another example is IP address category, a fundamental metric that should be created and added to all digital analytics projects.

- When do you need true real time processing? When fraud detection is critical, or when processing sensitive transactional data (credit card fraud detection, 911 calls). Other than that, *delayed analytics* (with a latency of a few seconds to 24 hours) is good enough.
- Make sure your sensitive data is well protected. Make sure your algorithms can not be tampered by criminal hackers or business hackers (spying on your business and stealing everything they can, legally or illegally, and jeopardizing your algorithms, translating in revenue loss). An example of business hacking can be found in section 3 in this article.
- Blend multiple models together to detect many types of patterns. Average these models. See chapter 2.
- Ask the right questions before purchasing software.
- Run Monte-Carlo simulations before choosing between two scenarios.
- Use multiple sources for the same data: your internal source, and data from one or two vendors. Understand the discrepancies between these various sources, to have a better idea about what the real numbers should be. Sometimes big discrepancies occur when a metric definition is changed by one of the vendors, or changed internally, or data has changed (some fields no longer tracked). A classic example is web traffic data: use internal log files, Google Analytics and another vendor (say Accenture) to track this data.

25. Useful Machine Learning Tricks

We propose simple solutions to important challenges that all data scientists face almost every day. In short, this chapter provides a toolbox for the handyman, useful for busy professionals in any field.

This chapter contains the following sections:

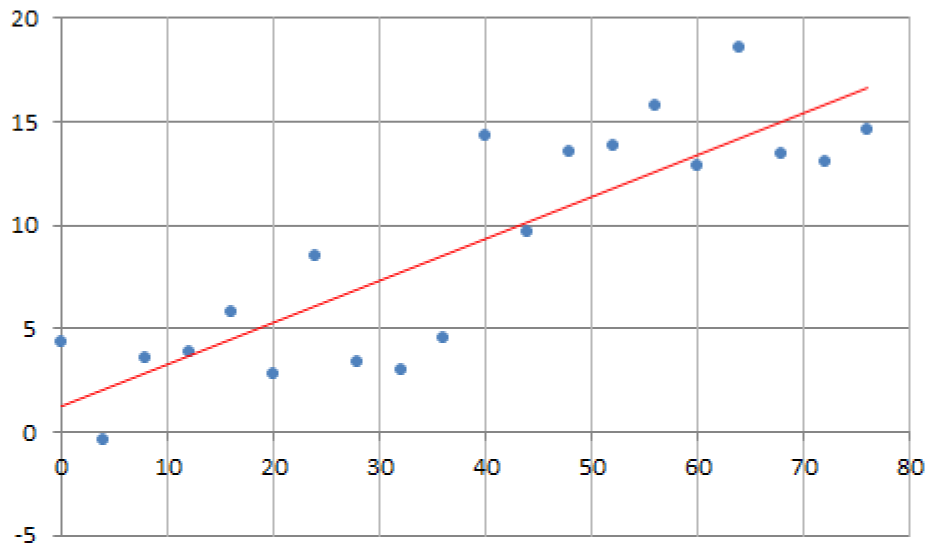
- Eliminating sample size effects
- Sample size determination
- Automatically detecting the number of clusters
- Fixing issues in regression models
- Performing joins on mismatched data
- Scale invariant techniques
- Blending data sets with non-compatible fields
- Automated exploratory data analysis
- Simple solution to feature selection problems
- Coefficient of Correlation for Non-Linear Relationships
- Choosing a regression model
- Growth modeling with Excel
- Interesting charts
- Simplified logistic regression

1. Eliminating sample size effects

Many statistics, such as correlations or R-squared, depend on the sample size, making it difficult to compare values computed on two data sets of different sizes. Based on re-sampling techniques, you can use this easy trick, to compare apples with other apples, not with oranges.

Many statistics, such as correlations or R-squared, depend on the sample size, making it difficult to compare values computed on two data sets of different sizes. Here, we address this issue.

Below is an example with 20 observations. The last 10 observations (the second half of the data set) is a mirror of the first 10, and the two correlations, computed on each subset, are identical and equal to 0.30. The full correlation computed on the 20 observations is 0.85.

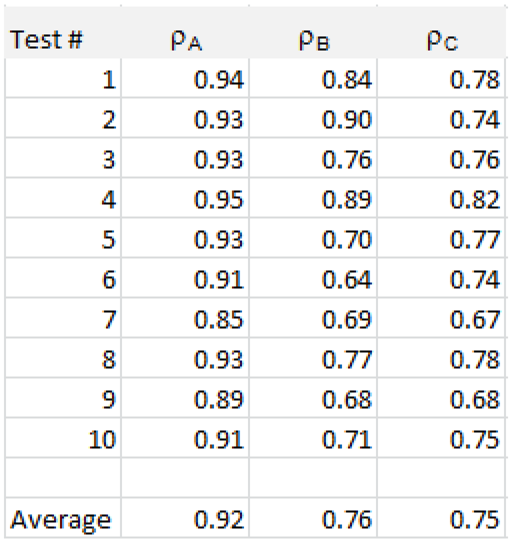


One would expect that since they represent the same association, these correlations should be identical. Of course, by doubling the number of observations (from 10 to 20) you get more statistical significance, and it strengthens the correlation. So from a statistical point of view, it makes sense that the correlation changes (increases) when adding new observations, if the new observations have the same behavior as the previous ones.

But it makes it impossible to make meaningful comparisons between data sets of different sizes. One way around this is to compute correlations on subsets of 10 points. There are 92,378 different ways to select 10 distinct observations out of 20, and thus 92,378 potential correlation values. If you average these values, you will get a number that you can truly be compared with that from a data set of size 10, yet it involves all the 20 observations.

In this case we simply averaged the 10 correlation values computed on all 10 subsets consisting of 10 consecutive observations. The final correlation, you can call it the *re-sampled correlation*, is equal to 0.67. Now you are no longer comparing apples and oranges.

Using the same data generation mechanism (that is, the same statistical model), I performed ten tests, each time with 20 observations, with the second half of the data set having the same correlation as the first half. This correlation is listed in the third column in the table below. The second column represents the correlation computed on the whole data set (20 observations) while the last (fourth) column represents the re-sampled correlation.



Test #	ρ_A	ρ_B	ρ_C
1	0.94	0.84	0.78
2	0.93	0.90	0.74
3	0.93	0.76	0.76
4	0.95	0.89	0.82
5	0.93	0.70	0.77
6	0.91	0.64	0.74
7	0.85	0.69	0.67
8	0.93	0.77	0.78
9	0.89	0.68	0.68
10	0.91	0.71	0.75
Average	0.92	0.76	0.75

The data, computations, and chart, is available in this spreadsheet. The data set consists of two variables stored in columns C and D. The same methodology could be applied to any coefficient, for instance the R-squared or the regression coefficients in a linear model. More about re-sampling techniques can be found in chapter 15.

2. Sample size determination

We propose a generic methodology, also based on re-sampling techniques, to compute any confidence interval and for testing hypotheses, without using any statistical theory. Also, it is easy to implement, even in Excel. The trick is based on the following new theorem:

***Theorem**: The width L of any confidence interval is asymptotically equal (as n tends to infinity) to a power function of n, namely $L = A / n^B$ where A and B are two positive constants depending on the data set, and n is the sample size, with B in [0, 1].*

The model-free methodology is explained in details I chapter 15. In short, *B* can be estimated via re-sampling, and even improved. Usually, *B* = 1/2, and for common estimators, we have

Estimator	A	Comment
Mean	σ	
Variance	$\sqrt{2}\cdot\sigma^2$	
Median	$\frac{1}{2f(m)}$	m is the median, f is the distribution density.
p-th quantile	$\frac{\sqrt{p(1-p)}}{f(x_p)}$	f is the distribution density, $x(p)$ is the p-th quantile.
Correlation	$1-\rho^2$	Assumes the true correlation is zero.

This allows you determine *n* in order to achieve a desired width *L* for your confidence interval. For some estimators, *B* may not be 1/2. For instance, if your estimator is the range (maximum minus minimum computed on your observations), its expectation and standard deviation are provided in the table below (source: see chapter 17).

	Type of Distribution	Expectation	Standard deviation
Uniform	Short tail	1	$\frac{1}{n}$
Gaussian	Medium tail	$\sqrt{\log n}$	$\frac{1}{\sqrt{n}}$
Exponential	Fat tail	$\log n$	1

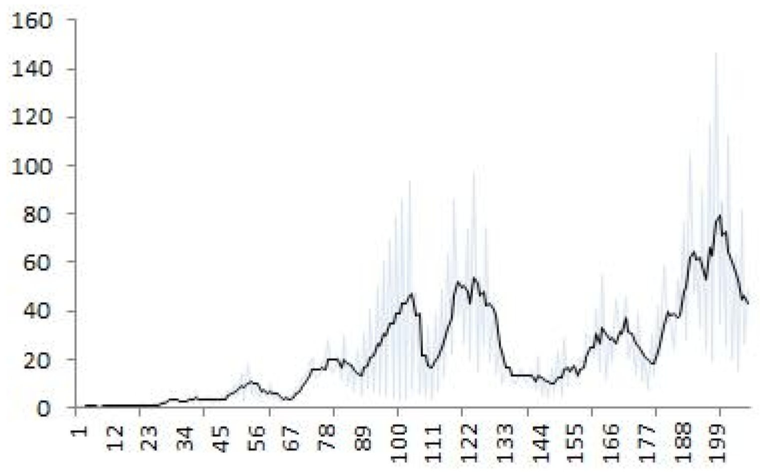
Order of magnitude for the expectation and Stdev of the range

3. Automatically detecting the number of clusters

We are dealing here with non-supervised clustering. This modern version of the elbow rule also tells you how strong the global optimum is, and can help you identify local optima too. It can also be automated.

Determining the number of clusters when performing unsupervised clustering is a tricky problem. Many data sets don't exhibit well separated clusters, and two human beings asked to visually tell the number of clusters by looking at a chart, are likely to provide two different answers. Sometimes clusters overlap with each other, and large clusters contain sub-clusters, making a decision not easy.

For instance, how many clusters do you see in the picture below? What is the optimum number of clusters? No one can tell with certainty, not AI, not a human being, not an algorithm.



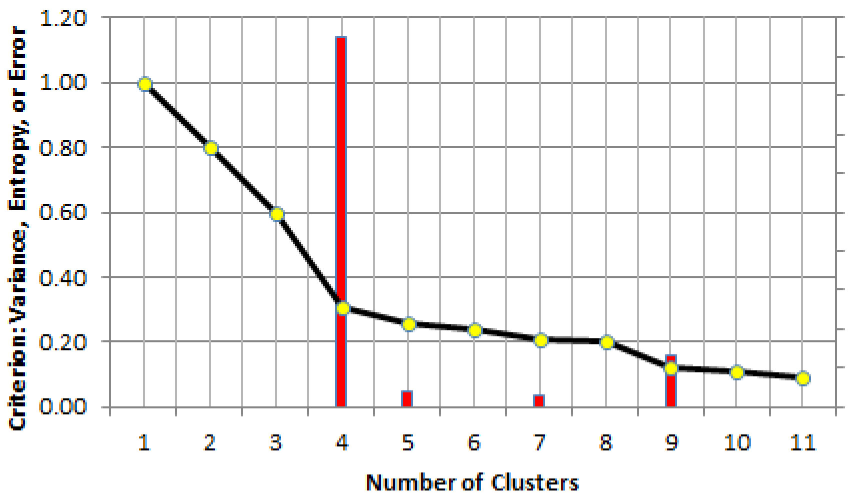
How many clusters here? (source: see here)

In the above picture, the underlying data suggests that there are three main clusters. But an answer such as 6 or 7, seems equally valid.

A number of empirical approaches have been used to determine the number of clusters in a data set. They usually fit into two categories:

- Model fitting techniques: an example is using a mixture model (see chapter 11) to fit with your data, and determine the optimum number of components; or use density estimation techniques, and test for the number of modes (see chapter 14.) Sometimes, the fit is compared with that of a model where observations are uniformly distributed on the entire support domain, thus with no cluster; you may have to estimate the support domain in question, and assume that it is not made of disjoint sub-domains; in many cases, the convex hull of your data set, as an estimate of the support domain, is good enough.
- Visual techniques: for instance, the silhouette or elbow rule (very popular.)

In both cases, you need a criterion to determine the optimum number of clusters. In the case of the elbow rule, one typically uses the percentage of unexplained variance. This number is 100% with zero cluster, and it decreases (initially sharply, then more modestly) as you increase the number of clusters in your model. When each point constitutes a cluster, this number drops to 0. Somewhere in between, the curve that displays your criterion, exhibits an elbow (see picture below), and that elbow determines the number of clusters. For instance, in the chart below, the optimum number of clusters is 4.



The elbow rule tells you that here, your data set has 4 clusters (elbow strength in red)

A good reference on the topic is this article. Some R functions are available too, for instance fviz_nbclust. However, I could not find in the literature, how the elbow point is explicitly computed. Most references mention that it is mostly hand-picked by visual inspection, or based on some predetermined but arbitrary threshold. In the next section, we solve this problem.

3.1. Automating the elbow rule

This is another example showing how data science can automate some tasks performed by statisticians, in this case in the context of exploratory data analysis. The solution is actually pretty simple, and applies to many problems not even related to clustering, that we will discuss later. Also, it is not limited to using the percentage of unexplained variance (Y- axis) to plot the elbow curve, but other criteria such as entropy, or error resulting from model fitting, work equally well. Indeed the solution provided here was designed to be integrated in black-box decision systems. The only requirement is that the elbow curve most be positive (above the X-axis) and decreasing.

In the next sections, we provide the context and formula to compute the elbow point, and to automate the procedure.

3.2. Elbow strength (with spreadsheet illustration)

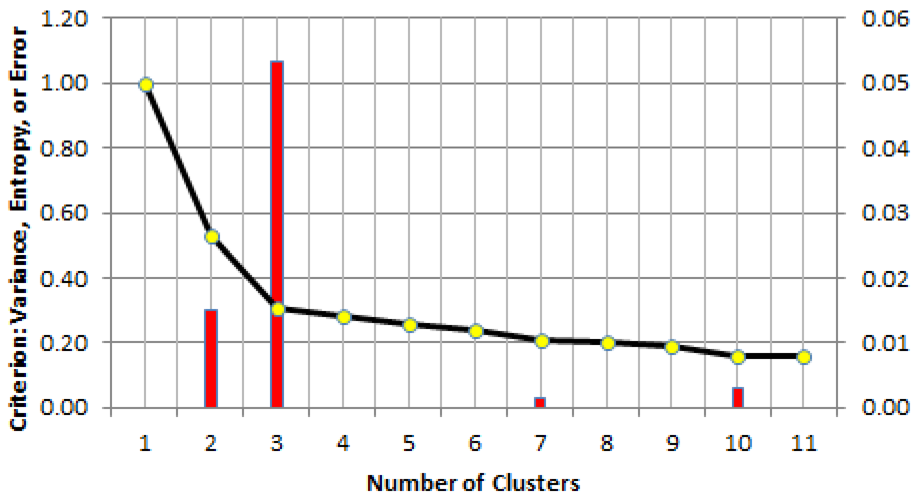
The formula to compute the elbow strength (the core concept in this article) is illustrated using the table below (corresponding to the figure in the beginning of this article) and available in our interactive spreadsheet (download the spreadsheet here).

Number of clusters	Criterion (error, entropy etc.)	Delta 1	Delta 2	Elbow?	Relative Strength	Strength
1	1.00					
2	0.80	0.20				
3	0.60	0.20	0.00			
4	0.31	0.29	-0.09	X	0.05	0.19
5	0.26	0.05	0.24	X	0.00	0.01
6	0.24	0.02	0.03			
7	0.21	0.03	-0.01	X	0.00	0.01
8	0.20	0.01	0.02			
9	0.12	0.08	-0.07	X	0.01	0.06
10	0.11	0.01	0.07			
11	0.09	0.02	-0.01			

The Delta 1 column in the table represents the differences between *k* and *k* + 1 clusters, measured on the criterion metric (the second column.) Delta 2 represents the difference computed on Delta 1, that is, the second-order differences. The strength (rightmost column) at line *k* (*k* is the number of clusters) is computed as the difference between Delta 2 and Delta 1, at line *k* +1. It is shown only if it is positive.

3.3. Number of clusters

The optimum number of clusters is the value of *k* that maximizes the strength. That's it! The strength tells, for a specific *k*, if there is a potential elbow at level *k* (corresponding to *k* clusters), and how strong the elbow signal is at that level. Sometimes the strongest signal is not the first one, though this is usually the case in many instances. Below is an example where this is not the case.

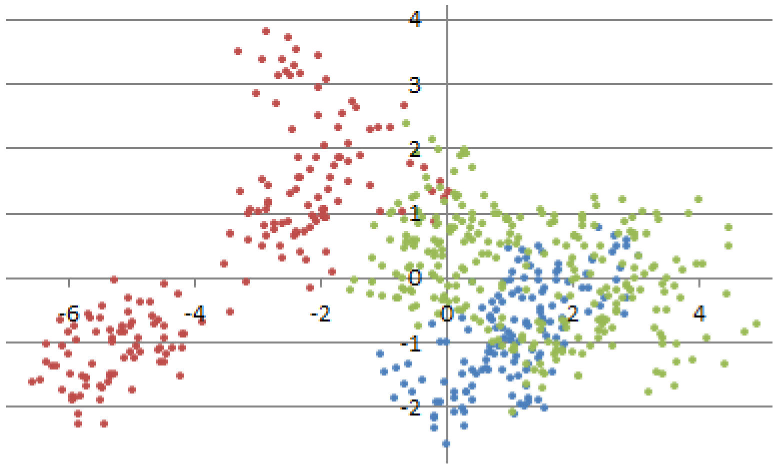


The above picture exhibits a situation where the data could conceivably have 2 or 3 clusters. However, the assumption of 3 clusters (instead of 2) is much more plausible, based on the height of the red bars. Rather than using the strength of the elbow, I actually used the *relative strength*: it is the strength, divided by *k*(the number of clusters). The relative strength dampens the strength of the elbow for large values of *k*, as these are usually less meaningful.

3.4. Testing

Three types of tests are worth doing to further assess the value of this method.

- Test with various elbow curves: we created curves, with multiple elbows or barely any elbow, to check the results produced by our procedure. We did not find counter-examples. Some of the test curves are included in our spreadsheet. Interestingly, if the shape of the elbow curve is like 1 / *k*, then two clusters are detected, which conforms to intuition. If is is decreasing at a much smaller pace, then the curve is too smooth to produce red bars, and no elbow is detected. This also conforms to intuition.
- Test on real data: these tests can be more difficult to interpret, since in many cases, nobody can tell the number of clusters, unless clusters are well separated or known a-priori.
- Test with simulated data: it is easy to generate data with a known number of clusters, see here. Then one can use a criterion, such as percentage of unexplained variance, and look at the elbow curve, to check when it correctly predicts the number of clusters. Below is an example of simulated clusters.



Simulated cluster structure to test the elbow rule (see here for source code)

3.5. Stopping rule for clustering algorithms

One open question is how the methodology performs when the data has more than two dimensions. The issue is not with the elbow curve itself, but with the criterion being used. Finally, when large clusters are found in a data set (especially with hierarchical clustering algorithms) it is a good idea to apply the elbow rule to any big cluster (split the big cluster into smaller clusters), in addition to the whole data set. In practice, once you hit the first red bar (or if there is another red bar just after the first one, and bigger than the first one), you can stop refining and splitting your clusters: you have reached an empirical optimum.

3.6. Other applications

The elbow rule can be used in various applications, not just to detect the number of clusters. We used it to detect how many decimals are correctly computed when using *high precision computing* libraries in Perl and Python, for a specific problem. You can check it out in my book on applied stochastic processes (available here) page 48. I also discuss the elbow rule in my optimum data binning procedure, see chapter 11. In time series, the elbow is sometimes referred to as a change point, signaling a change in the slope, and the elbow method can be used to identify these change points.

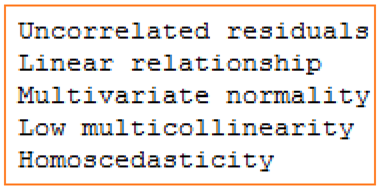
In fact, the elbow method can be used in any algorithm that has a stopping rule, where the criterion used to measure performance improvement at each new iteration, is a positive decreasing function. In particular, it can be used to detect how deep a decision tree should be (when to stop splitting nodes), or in numerical algorithms to detect when the accuracy level reached is good enough, and no longer steadily improving when adding more iterations.

An application of the elbow rule described here can be found here.

4. Fixing issues in regression models

What should you do if the model assumptions are violated? If your data has serial correlation, unequal variances and other similar problems, this simple trick will remove the issue and allows you to perform more meaningful regressions, or to detect flaws in your data set.

You cannot trust a linear or logistic regression performed on data if the error term (residuals) are auto-correlated. There are different approaches to de-correlate the observations, but they usually involve introducing a new matrix to take care of the resulting bias. See for instance here.


```
Uncorrelated residuals
Linear relationship
Multivariate normality
Low multicollinearity
Homoscedasticity
```

Requirements for linear regression

A radically different and much simpler approach is to re-shuffle the observations, randomly. If it does not take care of the issue (auto-correlations are weakened but still remain significant, after re-shuffling) it means that there is something fundamentally wrong about the data set, perhaps with the way the data was collected. In that case, cleaning the data or getting new data is the solution. But usually, re-shuffling - if done randomly - will eliminate these pesky correlations.

The trick

Reshuffling is done as follows:

- Add one column to your data set, consisting of pseudo random numbers, for instance generated with the function RAND in Excel.
- Sort the entire data set (all the columns, plus the new column containing the pseudo random numbers) according to the values in the newly added column.

Then do the regression again, and look at improvements in model performance. R-squared may not be a good indicator, but techniques based on cross-validation should be used instead.

Actually, any regression technique where the order of the observations does not matter, will not be sensitive to these auto-correlations. If you want to stick to standard, matrix-based regression techniques, then re-shuffling all your observations 10 times (to generate 10 new data sets, each one with the same observations but ordered in a different way) is the solution. Then you will end up with 10 different sets of estimates and predictors: one for each data set. You can compare them; if they differ significantly, there is something wrong in your data, unless auto-correlations are expected, as in time series models (in that case, you might want to use different techniques anyway, for instance techniques adapted to time series, see here.).

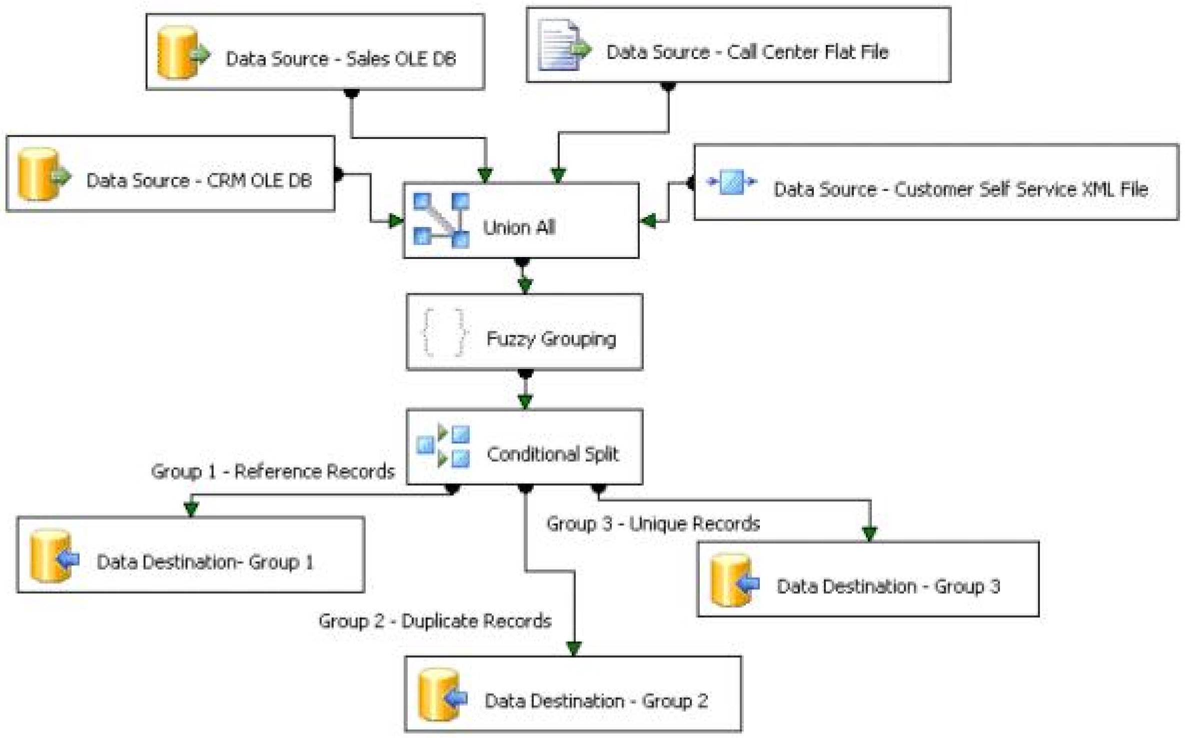
Testing for auto-correlations in the observations

If you have *n* observations and *p* variables, there is no global auto-correlation coefficient that measures the association between one observation and the next one. One way to do it is to compute it for each variable (column) separately. This will give you *p* lag-1 auto-correlation coefficients. Then you can look at the minimum (is it high in absolute value?) or the maximum (in absolute value) among these *p* coefficients. You can also check lag-2, lag-3 auto-correlations and so on. While auto-correlation between observations is not the same as auto-correlation between residuals, they are linked, and it is still a useful indicator of the quality of your data. For instance, if the data comes from sampling and consists of successive blocks of observations, each block corresponding to a segment, then you are likely to find auto-correlations, both in the observations and the residuals. Or if there is a data glitch and some observations are duplicated, you can experience the same issue.

5. Performing joins on mismatched data

This 40 year old trick allows you to perform a join when your data is infested with typos, multiple names representing the same entity, and other similar issues. In short, it performs a fuzzy join.

While much of data cleaning is performed before loading data in a database (especially for one-time, ad hoc analyses), there is a way to do it, continuously (like once a week or once a day), once the data is in its final database. It consists of adding look-up tables to help with the messy fields.


Data Source - Sales OLE DB
Data Source - Call Center Flat File
Data Source - CRM OLE DB
Union All
Data Source - Customer Self Service XML File
Fuzzy Grouping
Conditional Split
Group 1 - Reference Records
Group 3 - Unique Records
Data Destination- Group 1
Data Destination - Group 3
Group 2 - Duplicate Records
Data Destination - Group 2

When I was working at eBay, there was a database of clients from around the world. Some clients had names in a foreign language, containing accents and special characters. Somehow, it made some SQL joins very tricky. We created a lookup table of names, matching different spelling of a company name, to a standardized name and client ID. Think about names such as M.I.T and MIT that represent the same entity but can be spelled differently. It also helps dealing with duplicate records. This old trick allows you to do fuzzy matching, and the size of the lookup tables (updated daily) was manageable.

What do you think of this idea? Of course the best solution is to use this system, together with traditional cleaning techniques, if possible. But in systems where data is automatically uploaded and updated on a daily basis, lookup tables are very helpful.

6. Scale invariant techniques

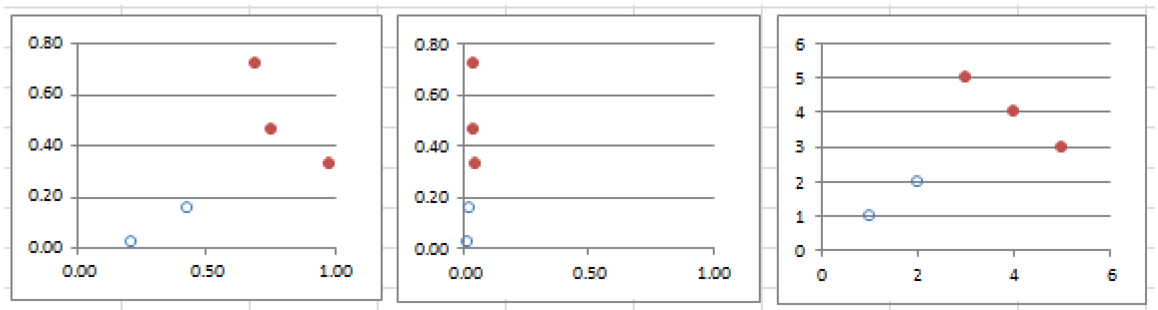
Sometimes, transforming your data, even changing the scale of one feature, say from meters to feet, have a dramatic impact on the results. Sometimes, you want your conclusions to be scale-independent. This trick solves this problem.

The impact of a change of scale, for instance using years instead of days as the unit of measurement for one variable in a clustering problem, can be dramatic. It can result in a totally different cluster structure. Frequently, this is not a desirable property, yet it is rarely mentioned in textbooks. I think all clustering software should state in their user guide, that the algorithm is sensitive to scale.

We illustrate the problem here, and propose a scale-invariant methodology for clustering. It applies to all clustering algorithms, as it consists of normalizing the observations before classifying the data points. It is not a magic solution, and it has its own drawbacks as we will see. In the case of linear regression, there is indeed no problem, and this is one of the few strengths of this technique.

6.1. Scale-invariant clustering

The problem may not be noticeable at first glance, especially in Excel, as charts are by default always re-scaled in spreadsheets (or when using charts in R or Python, for that matter). For simplicity, we consider here two clusters, see figure below.



Original data (left), X-axis re-scaled (middle), scale-invariant clustering (right)

The middle chart is obtained after re-scaling the X-axis, and as a result, the two-clusters structure is lost. Or maybe it is the one on the left-hand side that is wrong. Or both. Astute journalists and even researchers actually exploit this issue to present misleading, usually politically motivated, analyses. Students working on a clustering problem might not even be aware of the issue.

On the right-hand chart, we replaced each value for each axis, by their rank in the data set: it solves the problem, as re-scaling (or even applying any monotonic, non-linear transformation) preserves the order statistics (the ranks). Another way to do it is by normalizing each variable, so that the variance for each variable, after normalization, is equal to 1. Using the ranks is a better, more robust, noise-insensitive approach though, especially if the variables have a relatively unimodal distribution (with no big gaps), as in the above figure.

The main issue with scale-invariant clustering appears in the context of supervised classification. When adding new points to the training set, the augmented training set needs to be re-scaled again. There is no distance or similarity metric (the core metric used in clustering algorithms, be it K-NN, centroid or hierarchical clustering) that will consistently preserve the initial clustering structure after adding new points and re-scaling. See exercise in the last paragraph for a (failed) attempt to build such a distance. However, see my article on scale-invariant variance, which leads to a very weird kind of "variance" concept.

6.2. Scale-invariant regression

By design, linear regression is, in some way, scale-invariant. The fact is intuitive and certainly very easy to prove, and it is illustrated in our spreadsheet (see next section.) In short, if you multiply one or more dependent variable by a factor (which amounts to re-scaling them) then the corresponding regression coefficients will be inversely re-scaled by the same factor. To put it differently, if one dependent variable is measured in kilometers, and its attached regression coefficient is (say) 3.7, then if you change the measurement from kilometers to meters, its regression coefficient will change from 3.7 to 3.7 / 1000. This makes perfect sense, yet I don't remember having learned this fact in college classes nor textbooks.

Note that this works only if the re-scaling is linear. If you use a logarithm transformation instead, then this property is lost. Some authors have developed rank-regression techniques to handle non-linear re-scaling, using the same approach as in the previous section on clustering.

6.3. Excel spreadsheet with computations

To download the spreadsheet with the computations, click here. Probably the most interesting feature of the spreadsheet is to help you learn how to do linear regression in Excel, and how to produce scatter-plots with multiple clusters as in the above figure.

It is interesting to note that the 5 points in the above figure were all generated using random deviates on [0, 1] with the Excel function RAND(). Despite being "random", these points seem to exhibit a structure made of two clusters. This is a typical result: random points always exhibit some patterns (in particular, weak clustering, holes, weak linear structures and twin points.) See for instance section 7 in chapter 28. It is possible to test if these structures found in any data set are weak enough, yet not too weak, given the size of the data set, to assess whether it is a result of natural patterns found in randomness, or not. The easiest way to test this is by using Monte-Carlo simulations. If the points were too evenly distributed, they would not be the result of a random distribution.

So in the above figure, the two apparent clusters are an artifact or an optical illusion, and cannot be explained by any causal model. Repeat this experiment a thousand times, and you will find similar clusters in a majority of your simulations.

Exercise

Let's try to create a scale-invariant distance *d* between two points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ using this formula:

$$d(x,y) = \sup_{a,b \geq 0} \left(\frac{(ax_1 - ax_2)^2 + (by_1 - by_2)^2}{(a^2+b^2)/2} \right)^{1/2}.$$

Prove the following:

$$d(x,y) = \sqrt{2} \cdot \max(|x_1 - x_2|, |y_1 - y_2|)$$

and is thus not scale-invariant. It is proportional to the *infinity norm* distance. How does it generalize to more than two variables? Note that the the supremum in the first formula is attained either with $(a, b) = (1, 0)$ or $(a, b) = (0, 1)$. The case $(a, b) = (1, 1)$ corresponds to the classic Euclidean distance.

7. Blending data sets with non-compatible fields

Add consistency to your metrics! We are all too familiar with metrics that change over time and result in inconsistencies when comparing the past to the present, or when comparing different segments with incompatible measurements. This trick will allow you to design systems where again, apples are compared to other apples, not to oranges.

Here we describe a simple methodology to produce predictive scores that are consistent over time and compatible across various clients, to allow for meaningful comparisons and consistency in actions resulting from these scores, such as offering a loan. Scores are used in various contexts, such as web page rankings in search engines, credit score, risk score attached to loans or credit card transactions, the risk that someone might become a terrorist, and more. Typically a score is a function of a probability attached to some particular future event. They are built using training sets.

The reasons why scores can become meaningless over time is because data evolves. New features (variables) are added that were not available before, the definition of a metric is suddenly changed (for instance, the way income is measured) resulting in new data not compatible with prior data, and faulty scores. Also, when external data is gathered across multiple sources, each source may compute it differently, resulting in incompatibilities: for instance, when comparing individual credit scores from two people that are costumers at two different banks, each bank computes base metrics (income, recency, net worth, and so on) used to build the score, in a different way. Sometimes the issue is caused by missing data, especially when users with missing data are very different from those with full data attached to them.

Methodology

The idea to solve this problem is pretty simple. Let's say that you have two sets of data A and B, for instance corresponding to two different time periods: before, and after a change in the way the data is gathered or the scores are computed. Accordingly, you have two types of scores: S(A), computed on A, and T(B), computed on B. You proceed as follows.

- Compute the scores T(A) on A, using the scoring system T.
- Calibrate T on A; let Z be the calibrated score. Z might be a simple transformation (mapping) of T, so that Z(A) and S(A) have same mean (or median) and same variance. You can calibrate using more than two parameters, for instance, you might also want the kurtosis and/or skewness to be preserved.
- The new score to use moving forward, also called re-scaled score, is Z. It is compatible with the previous score S.
- Keep a log of all the changes happening to your score over time (for instance, the change from S to T, followed by transforming T into Z. This is similar to versioning in software development.

You can make it more robust if there is a transition period between A and B, when both scores S and T can be computed on overlapping data. This is the case if the score S can still be computed (in parallel with T) exactly up to 3 months after the score T was introduced.

An example of how this works in practice is given in chapter 2, in a very similar context. In that article, I discuss a scoring algorithm that blends two sub-scoring procedures, for increased performance: one based on robust decision trees (applying to a subset of the data set, say A), and one based on robust regression (see chapter 1.) Some data (say B) cannot be properly scored using the decision trees, and must be scored with the regression. You then apply the regression-based scoring to the whole data set, and then re-scale the score derived from the regression, so that it produces scores compatible with those generated with decision trees, on A. Moving forward, whether you have to use decision trees or regression, you get a consistent score everywhere. A detailed implementation with Excel spreadsheet and source code is available in chapter 3.

For more on this scoring technology, with application to scoring internet traffic (measuring its quality depending on the traffic source) read my technical article (PDF), here. Score preservation is discussed pages 22-26. Or you might want to check my patent on this topic, here.

8. Automated exploratory data analysis

Creating a data dictionary is the first exploratory step when dealing with a new data set. Here we explain how to do it. A data dictionary offers the following advantages:

- Identify areas of sparsity and areas of concentration in high-dimensional data sets
- Identify outliers and data glitches
- Get a good sense of what the data contains, and where to spend time (or not) in further data mining

What is a data dictionary?

A data dictionary is a table with 3 or 4 columns. The first column represents a label: that is, the name of a variable, or a combination of multiple (up to 3) variables. The second column is the value attached to the label: the first and second columns actually constitute a name-value pair. The third column is a frequency count: it measures how many times the value (attached to the label in question) is found in the data set. You can add a 4-th column that tells the dimension of the label (1 if it represents one variable, 2 if it represents a pair of two variables etc.)

Typically, you include all labels of dimension 1 and 2 with count > threshold (e.g. threshold = 5), but no or only very few values (the ones with high count) for labels of dimension 3. Labels of dimension 3 should be explored after having built the dictionary for dim 1 and 2, by drilling down on label/value of dim 2 that have a high count.

Example of dictionary entry

Look at the following entry:

```
category~keyword | travel~Tokyo | 756 | 2
```

In this example, the entry corresponds to a label of dimension 2 (as indicated in column 4), and the simultaneous combination of the two values (travel, Tokyo) is found 756 times in the data set.

The first thing you want to do with a dictionary is to sort it using the following 3-dim index: column 4, then column 1, then column 3. Then look at the data and find patterns.

How do you build a dictionary?

Browse your data set sequentially. For each observation, store all label/value of dim 1 and dim 2 as hash table keys, and increment count by 1 for each of these label/value. In Perl, it can be performed with code such as `$hash{"$label\t$value"}++`.

If the hash table grows very large, stop, save the hash table on file then delete it in memory and resume where you paused, with a new hash table. At the end, merge hash tables after ignoring hash entries where count is too small.

9. Simple solution to feature selection problems

We discuss a new approach for selecting features from a large set of features, in an unsupervised machine learning framework. In supervised learning such as linear regression or supervised clustering, it is possible to test the predicting power of a set of features (also called independent variables by statisticians, or predictors) using metrics such as goodness of fit with the response (the dependent variable), for instance using the R-squared coefficient. This makes the process of feature selection rather easy.

Here this is not feasible. The context could be pure clustering, with no training sets available, for instance in a fraud detection problem. We are also dealing with discrete and continuous variables, possibly including dummy variables that represent categories, such as gender. We assume that no simple statistical model explains the data, so the framework here is model-free, data-driven. In this context, traditional methods are based on information theory metrics to determine which subset of features brings the largest amount of information.

A classic approach consists of identifying the most information-rich feature, and then grow the set of selected features by adding new ones that maximize some criterion. There are many variants to this approach, for instance adding more than one feature at a time, or removing some features during the iterative feature selection algorithm. The search for an optimal solution to this combinatorial problem is not computationally feasible if the number of features is large, so an approximate solution (local optimum) is usually acceptable, and accurate enough for business purposes.

Review of popular methods

We focus here on the metric used to assess how information-rich a feature (or a set of features) is, as this is the key to find the best features in your data set. Features may be be correlated, or redundant. The same is true with observations.

A fairly comprehensive review on this topic can be found here. The simplest, probably oldest metric to measure the quantity of information associated with a feature, is the Shannon entropy, see here. It can be extended to measure the quantity of information associated with a set of features, see this article on joint entropy. However, this applies to discrete features only. It has also been generalized to continuous features: it is then called differential entropy. However this metric is scale-dependent, and model-dependent. Though in practice, in a model-free context, any statistical distribution can be replaced by the empirical distribution computed on the observations, or using the observed empirical percentiles.

Another popular metric is the Akaide information criterion. It was introduced in 1973, in what became one of the most popular scientific articles of all times -- in the top 100 citation index as of 2014. However, it is related to the likelihood function, and thus model-dependent. Related and somewhat equivalent to this criterion is the Kullback-Leibler divergence, but again having the same issue of being model-dependent.

In my more recent article on fast combinatorial feature selection (see chapter 5 and my Wiley book, page 224) I propose a data-driven, synthetic metric, called the predictive power of a feature.

New idea for feature selection

The idea is to add an artificial dependent variable (the response) to your data set, and perform feature selection as if you were dealing with a linear regression problem. That is, the criterion to select the features, would be based on a metric such as the residual error or R-squared, rather than using some kind of entropy metric. In short, you turn your problem into a problem of model fitting in a predictive analytics setting, which is easier. Another benefit is that the residual error or R-squared is not sensitive to changes in scale (re-scaling some variables) in your data set. Categorical variables such as gender can be replaced by dummy variables taking two values: 0 and 1. It also easily allows for cross-validation, selecting the features based on a subset of observations (the training set) and testing performance on the remaining data (the control set.)

All the regression coefficients could be set to 1 in the full model (involving all the features) to build the artificial response. Goodness-of-fit (e.g. R-square) is measured when an actual regression is performed on a subset of features. Features can be added one at a time as long as the goodness-of-fit metric continue to improve significantly when adding new features (by selecting features most efficiently accomplishing this goal), until you reach a pre-set, usually small number of "optimal" features.

Or you could test a large number of randomly generated regression coefficients for the response (via Monte Carlo simulations), and focus on those sets of regression coefficients that provides the best (or good enough) fit on a small set of features, still using the same goodness-of-fit criterion at each iteration, when selecting a new feature.

Testing on a dataset with known theoretical entropy

We illustrate here the concept explained in the previous section, on an artificial data set with known theoretical entropy attached to each feature. For simplicity, the data set has only two features. The data set consists of the first $n = 47$ digits of two numbers X_1 and X_2, expressed in two different bases: the digits of X_1 in base b_1, and the digits of X_2 in base b_2. The theoretical entropy attached to each feature is proportional to the logarithm of the base used for the feature in question. Using a number of digits (the number of observations) n larger than 50 causes accuracy issues (wrong digits) unless one uses high precision computing. This is discussed in details in my book *Applied Stochastic Processes, Chaos Modeling, and Probabilistic Properties of Numeration Systems*: see chapter 11.

We selected two numbers and bases causing some noticeable correlation between the two features, in order to better simulate a realistic data set. Auto-correlations within each feature were also strong. Our parameters are:

$$X_1 = \log(3/2)\ ,\ X_2 = 2^{-1/2},\ b_1 = 1.7,\ b_2 = 2.0.$$

Note that by using very large bases, you could produce observations (digits) that are very long, simulating actual continuous data, as opposed to binary data in this example. But then, you face again the accuracy issue (correctly computing the digits) described above.

Even with this small dataset, the classical Shannon entropy computed on the dataset, is equivalent to the theoretical entropy, in terms of deciding which feature is best. We also created an artificial response Y as discussed in the previous section, namely

$$Y = a_1 \text{ Feature}_1 + a_2 \text{ Feature}_2$$

with the regression coefficients a_1 and a_2 set to 1.

We then computed the correlations c_1 between Y and Feature$_1$, and c_2 between Y and Feature$_2$. In most cases (we tested with various numbers and various bases) the highest correlation corresponds to the feature with the highest entropy, thus proving compatibility with an entropy-based approach on a data set with no dependent variable. In the few cases where this was not true, it was because the bases b_1 and b_2 were very close to each other, and the entropy values almost identical for the two features. Even in that case, the two correlations were also very close to each other. In that case, picking one feature over the other does not make a difference. Moreover, the approach discussed here is model-free, data-driven. Also unlike data reduction techniques such as PCA (principal component analysis) or data compression, this approach preserves the original features: it does not transform and recombine them, making it easier for interpretation purposes.

You can download the spreadsheet with the simulated data set and all computations, here.

10. Coefficient of Correlation for Non-Linear Relationships

What is the best correlation coefficient R(X, Y) to measure non-linear dependencies between two variables X and Y? Let's say that you want to assess whether there is a linear or quadratic relationship between X and Y. One way to do it is to perform a polynomial regression such as $Y = a + bX + cX^2$, and then measure the standard coefficient of correlation between the predicted and observed values. How good is this approach?

Note that the proposed correlation coefficient R(X, Y) is not symmetric. One way to get a symmetric version, is to use the maximum between | R(X, Y) | and | R(Y, X) |. It will be equal to 1 if and only if there is an exact polynomial or inverse polynomial relationship between X and Y.

Note: For the model $Y = a + b\mathrm{X} + c\mathrm{X}^2$, the "inverse polynomial" model would be $X = a' + b'Y + c'Y^2$. So, R(X, Y) is computed on the first regression, while R(Y, X) is computed on the second (reversed, also called dual) regression.

Discussion

An issue with my approach is the risk of over-fitting. If you have *n* observations and *n* coefficients in the regression, my correlation will always be 1.

There are various ways to avoid this problem, for instance:

- Use a polynomial of degree 2 maximum, regardless of the number of observations.
- Use much smoother functions than polynomials, for instance functions that have one extremum (maximum or minimum) at most, and growing not faster than a linear function. Even in that case, use a small number of coefficients in the regression, maybe log(log *n*) where *n* is the number of observations.

The correlation coefficient in question can also be used for model selection: The best model would provide the correlation closest to 1.

11. Choosing a regression model

Should you use linear or logistic regression? In what contexts? There are hundreds of types of regressions. Here is an overview for data scientists and other analytic practitioners, to help you decide on what regression to use depending on your context. Many of the referenced articles are much better written (fully edited) in my data science Wiley book.

- Linear regression: Oldest type of regression, designed 250 years ago; computations (on small data) could easily be carried out by a human being, by design. Can be used for interpolation, but not suitable for predictive analytics; has many drawbacks when applied to modern data, e.g. sensitivity to both outliers and cross-correlations (both in the variable and observation domains), and subject to over-fitting. A better solution is piecewise-linear regression, in particular for time series.
- Logistic regression: Used extensively in clinical trials, scoring and fraud detection, when the response is binary (chance of succeeding or failing, e.g. for a new tested drug or a credit card transaction). Suffers same drawbacks as linear regression (not robust, model-dependent), and computing regression coefficients involves using complex iterative, numerically unstable algorithm. Can be well approximated by linear regression after transforming the response (logit transform). Some versions (Poisson or Cox regression) have been designed for a non-binary response, for categorical data (classification), ordered integer response (age groups), and even continuous response (regression trees).

- Ridge regression: A more robust version of linear regression, putting constraints on regression coefficients to make them much more natural, less subject to over-fitting, and easier to interpret. Click here for source code.
- Lasso regression: Similar to ridge regression, but automatically performs variable reduction (allowing regression coefficients to be zero).
- Ecologic regression: Consists in performing one regression per strata, if your data is segmented into several rather large core strata, groups, or bins. Beware about the curse of big data in this context: if you perform millions of regressions, some will be totally wrong, and the best ones will be overshadowed by noisy ones with great but artificial goodness-of-fit: a big concern if you try to identify extreme events and causal relationships (global warming, rare diseases or extreme flood modeling). See also chapter 27.
- Regression in unusual spaces: click here for details. Example: to detect if meteorite fragments come from a same celestial body, or to reverse-engineer Coca-Cola formula.
- Logic regression: Used when all variables are binary, typically in scoring algorithms. It is a specialized, more robust form of logistic regression (useful for fraud detection where each variable is a 0/1 rule), where all variables have been binned into binary variables.
- Bayesian regression: see entry in Wikipedia. It's a kind of *penalized likehood estimator*, and thus somewhat similar to ridge regression: more flexible and stable than traditional linear regression. It assumes that you have some prior knowledge about the regression coefficients and the error term - relaxing the assumption that the error must have a normal distribution (the error must still be independent across observations). However, in practice, the prior knowledge is translated into artificial (conjugate) priors - a weakness of this technique.
- Quantile regression: Used in connection with extreme events, read Common Errors in Statistics page 238 for details.
- LAD regression: Similar to linear regression, but using absolute values (L^1 space) rather than squares (L^2 space). More robust, see also our L^1 metric to assess goodness-of-fit (better than R^2) and our L^1 variance (one version of which is scale-invariant).
- Pseudo linear regression: This regression technique described in chapter 1 is also used as general clustering and data reduction technique. It solves all the drawbacks of traditional regression. It provides an approximate, yet very accurate, robust solution to regression problems, and work well with “independent” variables that are correlated and/or non-normal (for instance, data distributed according to a mixture model with several modes). Ideal for black-box predictive algorithms. It approximates linear regression quite well, but it is much more robust, and work when the assumptions of traditional regression (non-correlated variables, normal data, homoscedasticity) are violated.

Other Solutions

- Data reduction can also be performed with our feature selection algorithm (chapter 5.)
- It's always a good idea to blend multiple techniques together to improve your regression, clustering or segmentation algorithms. An example of such blending is described in chapter 2.
- Categorical independent variables such as race are sometimes coded using multiple (binary) dummy variables.

Before working on any project, read the lifecycle of a data science project (section 13, chapter 28.)

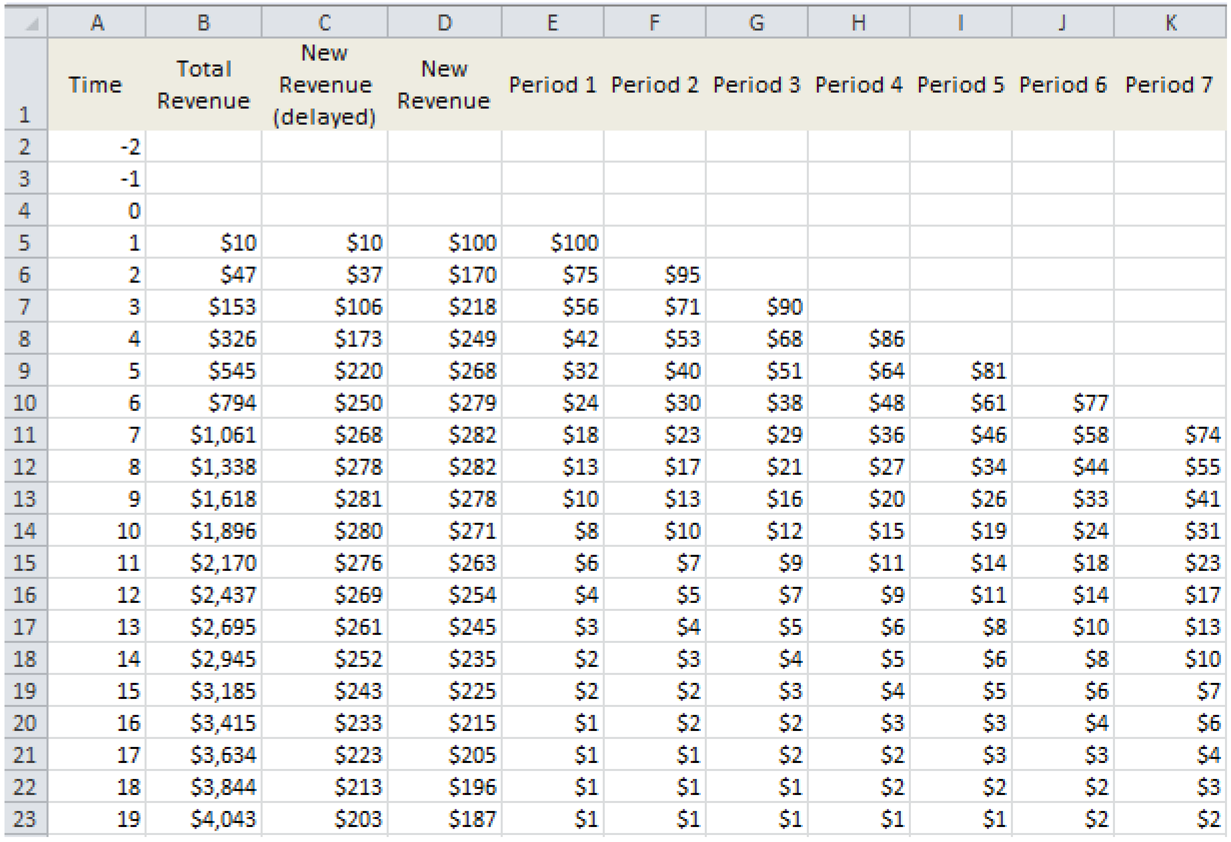
12. Growth modeling with Excel

You don't need a sophisticated model nor advanced machine learning techniques to quickly get a high level picture and trends for bottom-line business metrics. Not only the concepts explained here are easy to grasp, but while being high level, it nevertheless includes granular effects. The methodology presented here was used in business contexts in the past, when I was working with enterprise executives, particularly finance people, to assess the overall health of their business, and the short and long term impacts of new initiatives to boost growth. .

The model is available as an Excel spreadsheet, driven by four main parameters, as illustrated below. The growth can be in revenue, users, or any other fundamental metric. Time periods are measured in days when assessing the impact of an advertising campaign, or in months when assessing revenue growth caused by a new initiative. It typically involves the following dynamic:

- New growth occurs at each time period, for instance new users.
- It accumulates over time: new users become regular users, some of them eventually disappear -- this can be factored in in the growth curve.
- There is usually a time lag between an action and a reaction: the effect of TV advertising campaigns may peak after a while (not immediately) and eventually decay.

You can play with these factors separately in the spreadsheet, and even having your data science team track them separately: these are the model components. If the growth is due to more than one action (for instance multi-channel advertising), you might want to use attribution modeling techniques to separate the different sources and avoid double counting: see here for details. Some parameters may change over time, as you approach market saturation, of return on advertising may slow down over time if the campaigns and targeting are left unchanged: see the saturation parameter in the spreadsheet. Finally, some parameters can be adjusted for seasonality or holidays.

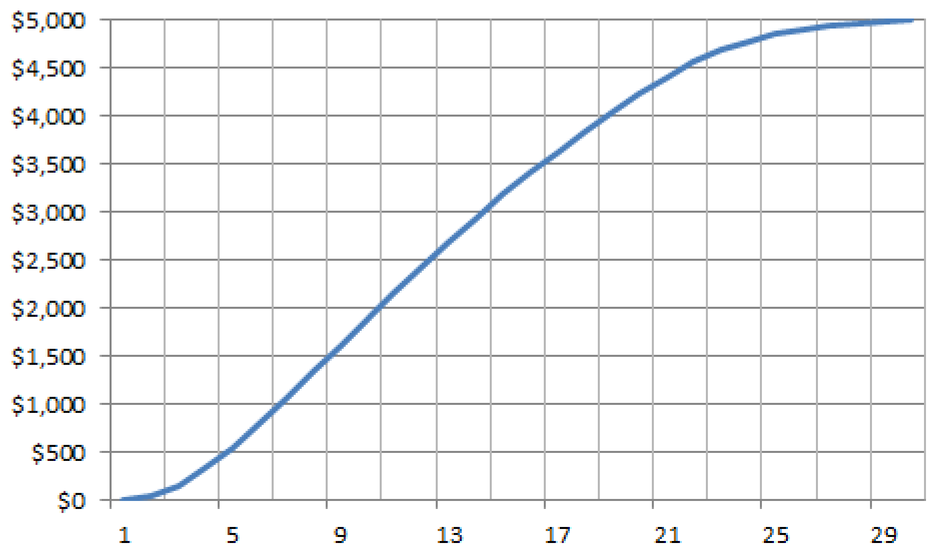


	A	B	C	D	E	F	G	H	I	J	K
1	Time	Total Revenue	New Revenue (delayed)	New Revenue	Period 1	Period 2	Period 3	Period 4	Period 5	Period 6	Period 7
2	-2										
3	-1										
4	0										
5	1	$10	$10	$100	$100						
6	2	$47	$37	$170	$75	$95					
7	3	$153	$106	$218	$56	$71	$90				
8	4	$326	$173	$249	$42	$53	$68	$86			
9	5	$545	$220	$268	$32	$40	$51	$64	$81		
10	6	$794	$250	$279	$24	$30	$38	$48	$61	$77	
11	7	$1,061	$268	$282	$18	$23	$29	$36	$46	$58	$74
12	8	$1,338	$278	$282	$13	$17	$21	$27	$34	$44	$55
13	9	$1,618	$281	$278	$10	$13	$16	$20	$26	$33	$41
14	10	$1,896	$280	$271	$8	$10	$12	$15	$19	$24	$31
15	11	$2,170	$276	$263	$6	$7	$9	$11	$14	$18	$23
16	12	$2,437	$269	$254	$4	$5	$7	$9	$11	$14	$17
17	13	$2,695	$261	$245	$3	$4	$5	$6	$8	$10	$13
18	14	$2,945	$252	$235	$2	$3	$4	$5	$6	$8	$10
19	15	$3,185	$243	$225	$2	$2	$3	$4	$5	$6	$7
20	16	$3,415	$233	$215	$1	$2	$2	$3	$3	$4	$6
21	17	$3,634	$223	$205	$1	$1	$2	$2	$3	$3	$4
22	18	$3,844	$213	$196	$1	$1	$1	$2	$2	$2	$3
23	19	$4,043	$203	$187	$1	$1	$1	$1	$1	$2	$2

Explanation

The two main columns are A and B, representing time and total revenue per time period. At each time period (columns E, F, and so on) new users are added, resulting from advertising efforts. They appear over the course of several time periods (for instance, cells E5 to E23 for the first batch of new users, corresponding to day 1 of your advertising campaign) and the number decays exponentially over time.

There is some erosion (saturation) in the advertising effectiveness: this is why E5 > F6 > G7 and so on. In addition, the revenue is delayed: this explains why columns C and D are different. But the sums over columns C and D are identical. Finally, there is attrition, which is incorporated in column B.



Growth curve corresponding to the above table (X-axis is the time period)

Parameters

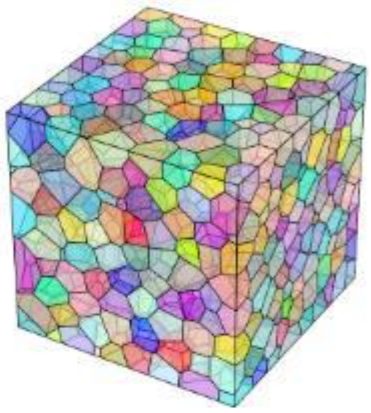
The parameters are chosen to match the growth curve with actual data (past data, or training data.) Then the growth numbers are automatically computed for the future, as in the spreadsheet. You should work with BI analysts or data scientists to make sure that all the numbers and projections are sound. The parameters are found in the Parameter tab in the spreadsheet, and you can fine-tune them to automatically adjust the chart. The parameters are:

- Saturation: To model decline in advertising effectiveness, over time.
- Decay: Advertising done during one time period has impact over several time periods, with a decaying effect.
- Attrition: Proportion of users dying during any time period.
- Time lags: Revenue resulting from one column (advertising done during a specific time period) is spread over several rows (it is time-delayed).

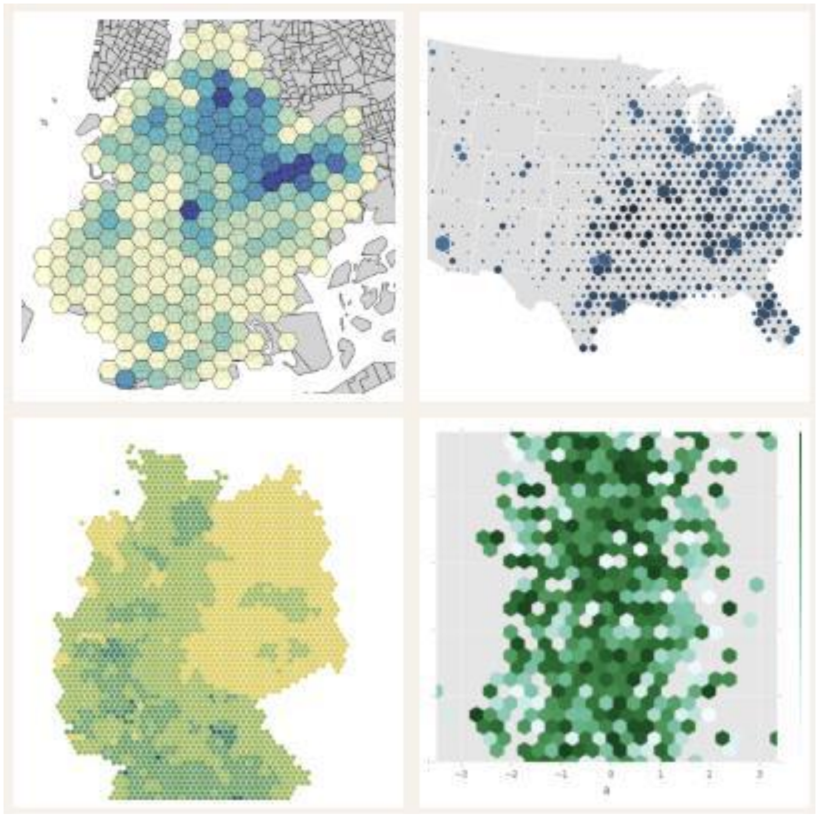
The campaign to boost your metric starts at period 1 (row 5 in the spreadsheet.) You can download the spreadsheet here. See also chapter 23. For a more technical presentation (fitting a growth curve with the logistic distribution), see a SAS article here. Our spreadsheet can model a large spectrum of growth scenarios, more than usually available in statistical packages.

13. Interesting charts

Hexagonal binning communicates the same insights as a contour plot. What is interesting is the choice of hexagonal buckets (rather than squares) to aggregate data. In fact, any tessellation would work, in particular Voronoi tessellations.


3-D Voronoi tessellation

The reason for using hexagons is that it is still pretty simple, and when you rotate the chart by 60 degrees (or a multiple of 60 degrees) you still get the same visualization. For squares, rotations of 60 degrees don't work, only multiples of 90 degrees work. Is it possible to find a tessellation such that smaller rotations, say 45 or 30 degrees, leave the chart unchanged? The answer is no. Octogonal tessellations don't really exist, so the hexagon is an optimum.


Hexagonal binning plots (source: here)

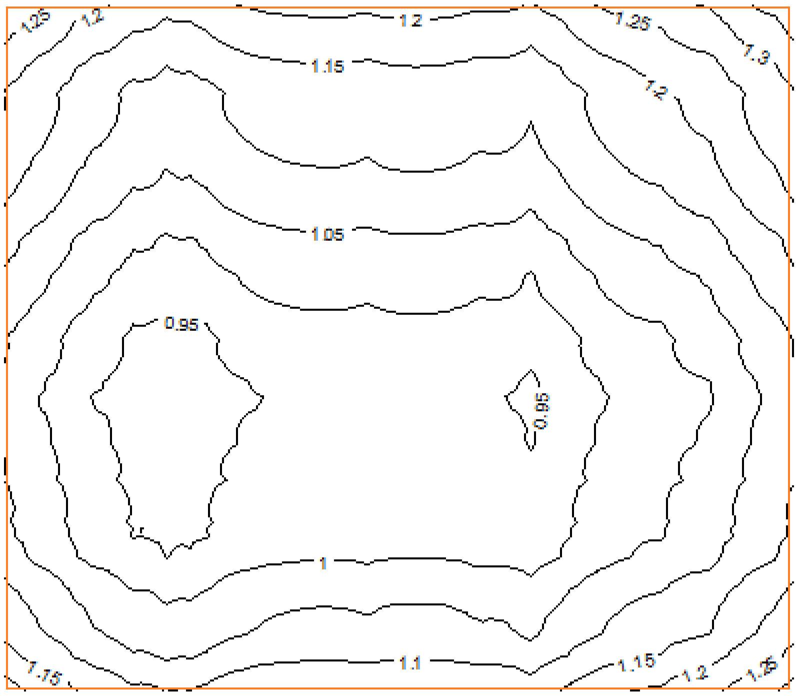
Implementation in R

The three plots described here (Voronoi diagram, hexagonal binning and contour plots) are available in the ggplot2 package.

- Hexagonal binning: `ggplot` function with the parameter `stat_binhex`, see here
- Contour plot: `ggplot` function with the parameter `geom_density2` or `stat_contour`, see here (also works with contour)
- Voronoi diagram: `ggplot` with the parameter `geom_segment`, see here

Applications

Voronoi diagrams can be used for nearest neighbor clustering or density estimation, the density estimate attached to a point being proportional to the inverse of the area of the Voronoi polygon containing it.



Example of contour map (see chapter 26)

14. Simplified logistic regression

Logistic regression is typically used when the response *Y* is a probability or a binary value (0 or 1). For instance, the chance for an email message to be spam, based on a number of features such as suspicious keywords or IP address. In matrix notation, the model can be written as

$$Y = \Big[1 + \exp(-(bX+e))\Big]^{-1} = \frac{1}{2}\left(1 + \frac{bX+e}{2} + \cdots\right).$$

where X is the observations matrix, b is the parameter vector that needs to be estimated, and e is a white noise. The first order approximation around zero, in the above Taylor series expansion, yields

$$4Y - 2 = bX + e.$$

If instead of the logistic function, you use a different one, you would still get the same first-order approximation in general. Replacing $4Y - 2$ by Z, we are left with a standard linear regression. When the response is binary (1 = spam, 0 = not spam), the technique can be further refined by introducing an extra parameter q called the threshold. The final estimate for a particular observation (an email with its set of attributes) is set to 1 (spam) if its Z value is larger than q, and to 0 (normal email) otherwise. By default, $q= 0$, but you could choose q to achieve the best classification of your training set (on the test set used in a cross-validation setting.) The correctness of the method can be measured for instance as a weighted proportion of false positives and false negatives.

The methodology can easily be extended to more than 2 classes, using multiple thresholds parameters and proper labeling (for instance: 3 for scam, 2 for spam, 1 for low priority email, 0 for normal email.) Even though the technique is not model-driven, confidence intervals can still be built using re-sampling techniques described here and here. In particular, it is possible to tell whether an email is very highly likely to be spam, or whether there is some non-conclusive evidence that it might be spam, based on the distance (its empirical distribution computed via re-sampling) between the observed Z and the threshold q.

It would be interesting to compare this method with a standard logistic regression, to see, using a confusion matrix, the differences (if any) in the way the messages are classified. More importantly, it would be useful to test when the approximated solution is not as good as the exact solution.

Other techniques to perform this type of clustering include neural networks, naive Bayes, and hybrid models (combining multiple techniques, see chapter 2.)

26. Dealing with Outliers

In addition to managing outliers in various machine learning problems, you will learn in this chapter how to simulate realistic cluster structures, make contour plots and other visualizations in R, and assess the convergence of an algorithm.

Here, we discuss a general framework to drastically reduce the influence of outliers in most contexts. It applies to problems such as clustering (finding centroids,) regression, measuring correlation or R-Squared, and many more. We will focus on the centroid problem here, as it is very similar and generalizes easily to solving a linear regression. The correlation / R-Squared issue was discussed in an earlier article and involves only a change of formula. Clustering and regression are more complex problems involving iterative algorithms.

This chapter also features interesting material for future data scientists, such as

- Several outlier detection techniques
- How to display contour maps and images corresponding to an intensity function or heatmap, in R (in just a few lines of code, and very easy to understand) -- see section 5 below
- How to produce data sets that simulate clustering structures or other patterns
- Distribution of arrival times for successive records in a time series

1. General Framework

We discuss replacing techniques used by statisticians, based on optimizing traditional L^2 metrics such as variance, by techniques based on L^p metrics, which are more robust when $1 < p < 2$. The case $p = 2$ corresponds to the traditional framework. Throughout this chapter, p is referred to as the power. The regression and centroid problems being equivalent, we focus here on finding a centroid using the L^p criterion (denoted as H), and we show how to modify it for regression problems. Illustrations are in a 2-dimensional space ($d = 2$) but easily generalize to any dimension, especially as we are not using any matrix inversions to solve the problem.

Finding a robust centroid

The focus here is on finding the point that minimizes the sum of the "distances" to n points in a d-dimensional space, called centroid or center, especially in the presence of outliers.

The sum of "distances" between an arbitrary point (u, v) and a set S = { (x_1, y_1) ... (x_n, y_n) } of n points is defined as follows:

$$H((u,v),S;p) = \frac{1}{n}\sum_{k=1}^{n}\Big((e + |x_k - u|)^p + (e + |y_k - v|^p)\Big),$$

where *e* is a very small positive quantity, equal to zero unless *p* is negative.

The function *H* has one parameter *p* called *power*, and when *p* = 2, we are facing the traditional problem of finding the centroid of a cloud of points: in this case, the solution is the classic average of the *n* points. This solution is notoriously sensitive to outliers. When 1 < *p* < 2, we get a more stable solution, less sensitive to outliers, yet when *n* is large (> 20) and the proportion of outliers is small (by definition it is always small!), the solution is pretty much the same regardless of *p* (assuming 1 < *p* < 3).

In short, what we want to build a robust measure of centrality in any dimension, just like the median which is a robust measure of centrality in dimension *d* = 1.

Generalization to linear regression problems

The same methodology can be used for regression. With 2 parameters *u* and *v* as in *Y* = *u*X + *v*, the function *H* becomes

$$H((u,v),S;p) = \frac{1}{n}\sum_{k=1}^{n}\Big(e + |y_k - ux_k - v|)^p\Big),$$

again with *e* = 0 if *p* > 0. It generalizes easily to more than 2 parameters *u*, *v*.

General outlier detection techniques

Our proposed method smooths out the impact of outliers rather than detecting them. For outlier detection and removal, you can use one of these methods:

- Using traditional centrality measures and eliminating the points farthest away from the center, then re-computing the center and proceeding iteratively with eliminating newly found outliers until stability is reached.
- Using the median both for the x- and y-coordinates, rather than the average.
- The *leaving-one-out* technique consists of computing the convex hull of your data set *S*, then removing one point at a time, and re-computing the convex hull after removing the point in question. The point resulting in the largest loss of volume in the convex hull, when removed, corresponds to the strongest outlier.
- Identifying points of lowest density using density estimation techniques (see section 5 in chapter 28.)
- Nearest neighbor distances: a point far away from its nearest neighbors is potentially an outlier. Compute all nearest neighbor distances and look for the most extremes.

Another way to attenuate the impact of outliers is to use a weighted sum for *H*, that is (in the case of the regression problem) to use the formula

$$H((u,v),S;p)=\sum_{k=1}^{n} q_k\cdot\Big(e+|y_k-ux_k-v|)^p\Big),\ \text{with}\ \sum_{k=1}^{n} q_k=1,$$

where q(*k*) is the weight attached to point *k*. In this case, a point with low density is assigned a smaller weight.

To read more about outlier detection, click here.

A related physics problem

When *p* < 0, and especially when *p* = -2, maximizing or minimizing *H* becomes an interesting physics problem of optimizing a potential *H*. The centroid problem is now equivalent to finding the point of maximum or minimum light, sound, radioactivity, or heat intensity, in the presence of an energy field produced by *n* energy source points. Both problems are closely related and use the same algorithm to find solutions.

However, the case *p* < 0 has no practical value to data scientists or statisticians (as far as I know) and it presents the following challenges:

- If (*u*, *v*) is a point of *S* and *p* < 0, then *H*((*u*,*v*), *S*; *p*) is infinite: we have singularities. This is dealt with by introducing the very small constant *e* > 0 in the definition of *H*. This quantity is used to address the fact that in the real world, no source of energy is a point with an area of zero and positive (but finite) intensity. This artifact of physics models is discussed here.
- If you want to minimize *H*, there will be an infinite number of solutions, all located far outside the cloud of points *S*. Think about finding the point in the universe that receives the least amount of light from the solar system. So typically one is interested in finding a maximum, not a minimum (unless you put constraints on the solution, such as being located inside the convex hull of *S*.)
- Even if searching for a maximum of *H*, the convergence may be slow and chaotic, as you can end up with several maxima (*H* is no longer a nice, smooth curve when *p* < 0.)

2. Algorithm to find centroid when *p* > 1

There are many algorithms available to solve this type of mathematical optimization problem. A popular class of algorithms is based on gradient descent and boosting. Here however, we use what is possibly the simplest algorithm. First it is easy to understand, still leads to some interesting research, is easy to replicate, and keep the focus on the concepts and the results associated with the centroids, rather than on a specific implementation. Second, with modern computers (even on my 10 years old laptop) the computing power is big enough that naive algorithms perform well. If you are OK with getting results accurate to two or three digits -- and in real life, many data sets, even web analytics measured using two different sources, do not have higher accuracy -- then a rudimentary algorithm is enough.

So here I used rudimentary Monte-Carlo to find the centroids. However, in higher dimensions (*d* > 3) be careful about the curse of dimensionality. Note that when *p* < 0, Monte Carlo is not that bad, as it allows you to visit and circle around several local minima. It is also easy to deploy in a Map-Reduce environment (Hadoop.) The algorithm is actually so simple that there is no need to describe it: the short, easy-to-read source code below (Perl) speaks for itself.

Source code to generate points and compute centroid, using Monte Carlo

Notes about the source code:

- The first *n* points in the output file centroid.txt are the simulated sample points (random deviates on [0, 1] x [0, 1]. The last point is the centroid computed on the sample points.
- Note that `$seed` (first line of code) is used to initiate the random generator and for reproducibility purposes. Also, I noticed that a value of `$seed` lower than 1,000 causes the first random deviate generated to be biased (unusually small.)
- The variable `$sum` stores, at each iteration `$iter`, the value of the function *H* that we try to minimize.

Below is the source code.

```
$seed=1000;
srand($seed);
$n=100;
open(OUT,">centroid.txt");

for ($k=0; $k<$n; $k++) {
  $x[$k]=rand();
  $y[$k]=rand();
  print OUT "$x[$k]\t$y[$k]\n"; # one of the n simulated points
}
$power = 1.25;  # corresponds to p (the power) in the article
$niter = 200000;
$eps = 0.00001; # the "e" in the H function (see article)
$min=99999999;
for ($iter=0; $iter<$niter; $iter++) {
  $u=rand();
  $v=rand();
  $sum=0;
  for ($k=0; $k<$n; $k++) {
    $dist=exp($power*log($eps+abs($x[$k]-
$u)))+exp($power*log($eps+abs($y[$k]-$v)));
    $sum+=$dist;
  }
  $sum = $sum/$n;
  if ($sum < $min) {
    $min=$sum;
```

```
          $x_centroid=$u;
          $y_centroid=$v;
        }
      }
      print OUT "$x_centroid\t$y_centroid\n";
      close(OUT);
```

Generating point clouds with simulation

In the few lines of the source code (above), we generated points randomly distributed in the unit square [0, 1] x [0, 1] . In order to simulate outliers or more complicated distributions that represent real-life problems, one has to use more sophisticated techniques. Click here to get the source code to easily generate a cluster structure (illustrated in figure 1 below).

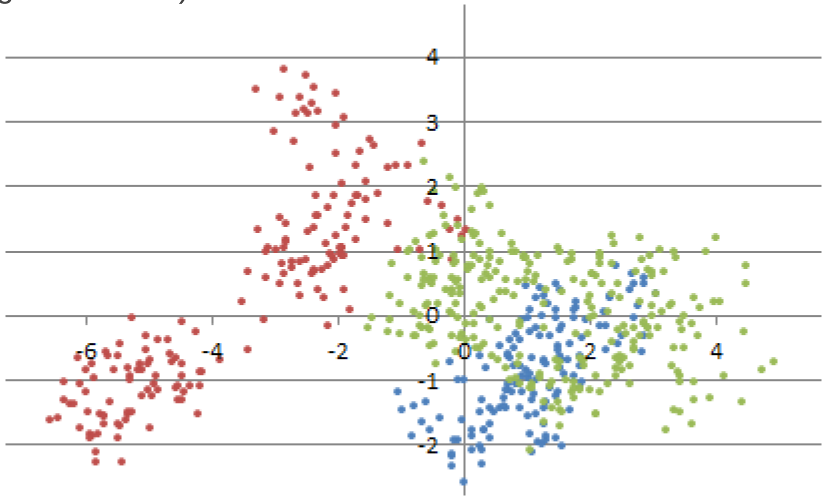


Figure 1: *example of simulated clustered point cloud*

More complex simulations (random clusters evolving over time) can be found here. Some simple stochastic processes can be simulated by first simulating random points (called centers) uniformly distributed in a rectangle, then, around each center, simulating a random number of points radially distributed around each center. Other techniques involve thinning, that is, removing some points after over-sampling a large number of points.

3. Examples and results

A few examples are provided in the next section to validate the methodology. In this section, rather than validation, we focus on showing its usefulness when 0 < p < 2, compared to the traditional solution consisting of using *p* = 2, found in all statistical packages. Note that the traditional solution (*p* = 2) was designed not out of practical considerations such as robustness, but because of its ease of computation at a time when computers did not exist, and data sets were manually built.

To test the methodology, we created various data sets, introduced outliers, and computed the centroid using different values of *p*. The example shown in Figure 2, consisting of five points (bottom left) plus an outlier (top right) illustrates the performance. The six data points are in blue. The centroids, computed for *p* = 0.75, 1.00, 1.25, 1.50, 1.75 and 2.00, are in red. The rightmost centroid corresponds to *p* = 2: this is the classic centroid. Due to the outlier, it is located outside the convex hull of the five remaining points, which is awkward. The leftmost centroid corresponds to *p* = 0.75 and is very close to the traditional centroid obtained after removing the outlier. I also tried the values *p* = 0.25 and *p* = 0.50, but failed to obtain convergence to a unique solution after 200,000 iterations.

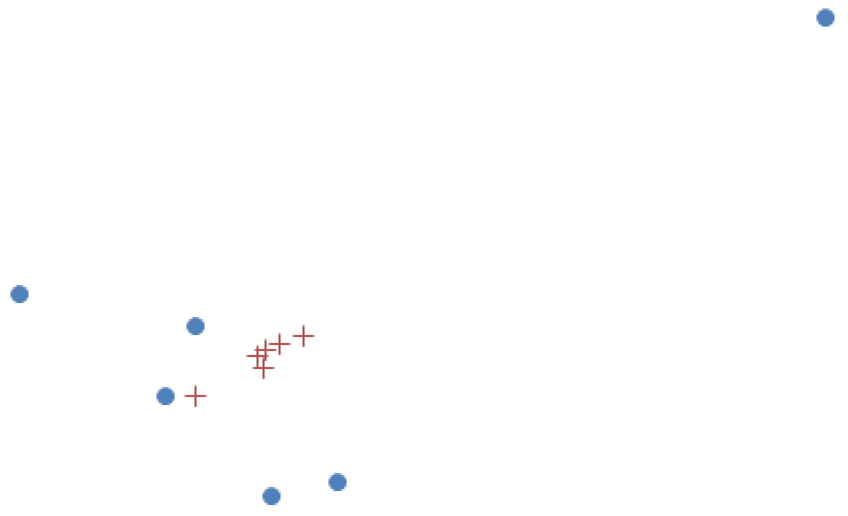


Figure 2: *the blue dots represent the data points, the red + the centroid (computed for 6 values of p)*

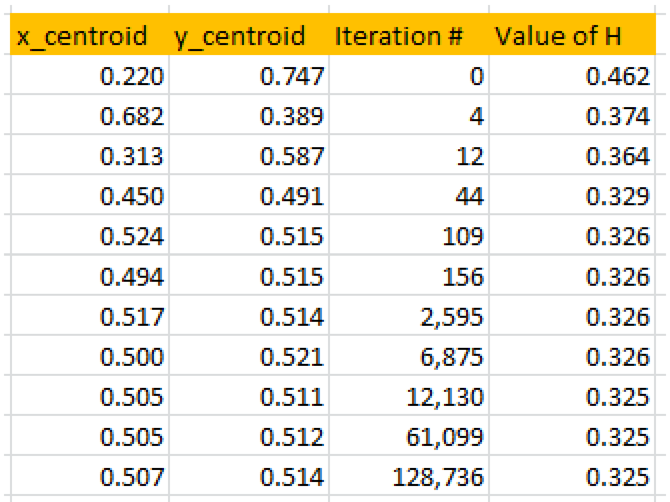
Note that the scale does not matter. Finally, using *p* < 2 does not fully get rid of the influence of the outlier, but instead, it reduces its impact when computing the centroid. To completely get rid of the outliers, a methodology using medians computed for the x- and y-axis, is more efficient.

If you are wondering how to produce a scatter plot in Excel with two data sets (points and centroids) as in Figure 2, click here for instructions: it is easier than you think.

4. Convergence of the algorithm

Figure 3 shows the speed of convergence of the algorithm, using the same source code as in section 2 with *p* = 1.4. So you can replicate these results. In this case, the data set *S* consists of *n* = 100 points randomly (uniformly) distributed on [0, 1] x [0, 1].

About 10,000 iterations in the outer loop are needed to reach two digits of accuracy, and this requires a tiny fraction of a second to compute. This “10,000 iterations” is actually a rule of thumb for any Monte-Carlo algorithm used to find an optimum with two correct digits. Note that in Figure 3, only iterations providing an improvement over the current approximation of the centroid -- that is, iterations where the value of *H* is smaller than those computed in all previous iterations -- are displayed. A potential research topic is to investigate the asymptotic behavior of these “records”, in the example below occurring at iterations 0, 4, 12, 44, 109, 156, and so on.



x_centroid	y_centroid	Iteration #	Value of H
0.220	0.747	0	0.462
0.682	0.389	4	0.374
0.313	0.587	12	0.364
0.450	0.491	44	0.329
0.524	0.515	109	0.326
0.494	0.515	156	0.326
0.517	0.514	2,595	0.326
0.500	0.521	6,875	0.326
0.505	0.511	12,130	0.325
0.505	0.512	61,099	0.325
0.507	0.514	128,736	0.325

Figure 3: *convergence of the algorithm (p = 1.4)*

Since the *n* = 100 simulated points were randomly (uniformly) distributed in [0, 1] x [0, 1] it is no surprise that the centroid found by the algorithm, after convergence, is very close to (0.5, 0.5).

Indeed, we tried values of *p* equally spaced between 1.25 and 3.50, and in each case, the centroid found was also very close to (0.5, 0.5), see Figure 4. That includes the special case *p* = 2 (it is located somewhere on the chart below) corresponding to the classic average of the *n* points. Note that here, no outliers were introduced in the simulations.

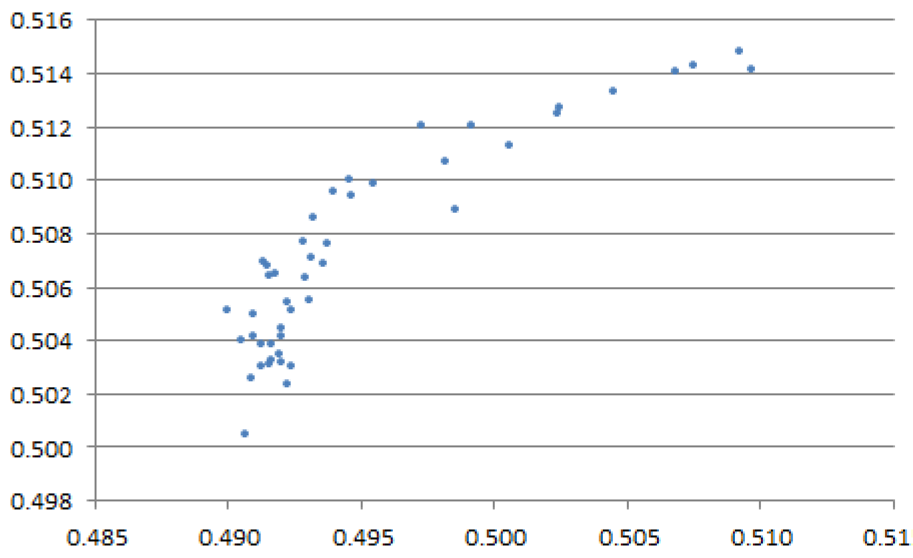


Figure 4: *x- and y-coordinates of centroid, obtained with various values of p*

5. Interesting Contour Maps

The following contour maps in Figure 5, produced with the contour function in R, show that as *p* gets closer to 0, the function *H* becomes more chaotic, exhibiting local minima. These charts were produced using a data set S consisting of *n* = 20 points randomly (uniformly) distributed on [0, 1] x [0, 1]. It is interesting to notice that, despite the random distribution of the *n* points, strong patterns emerge when *p* < 1 (the statistical significance of these patterns is weak though.)

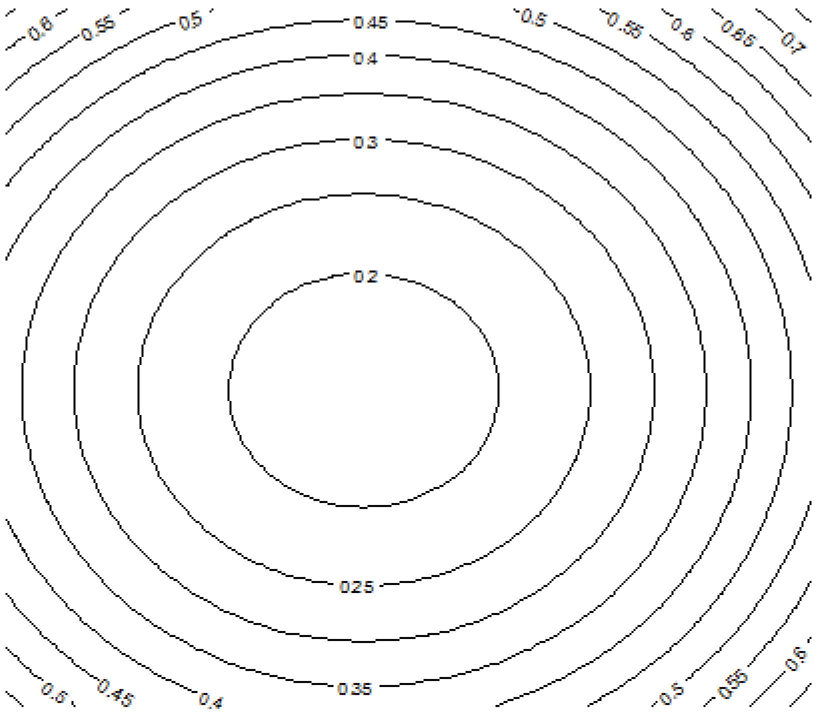


Figure 5A: *Contour map for H, with n = 20 and p =2*

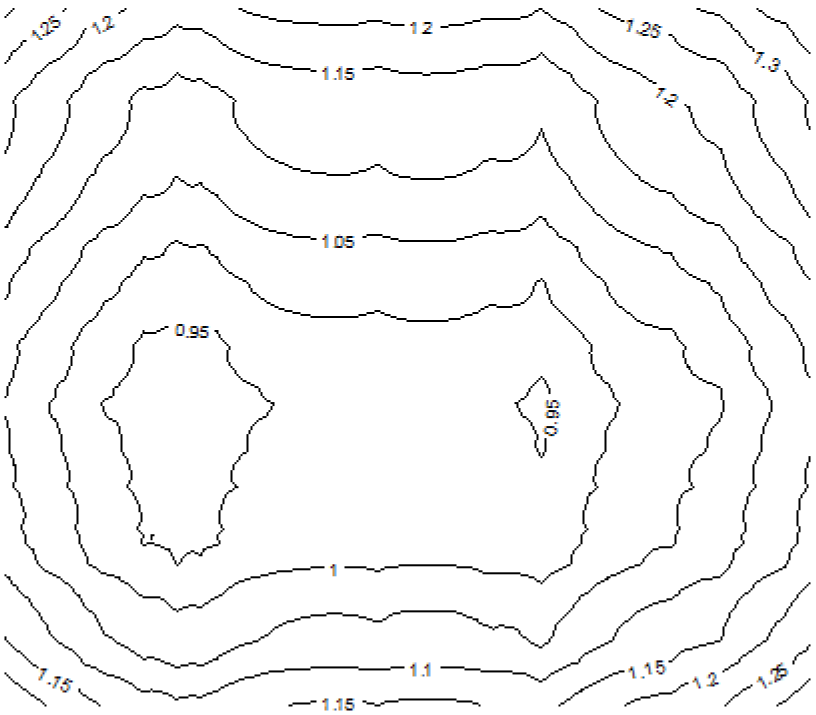


Figure 5B: *Contour map for H, with n = 20 and p = 0.50*

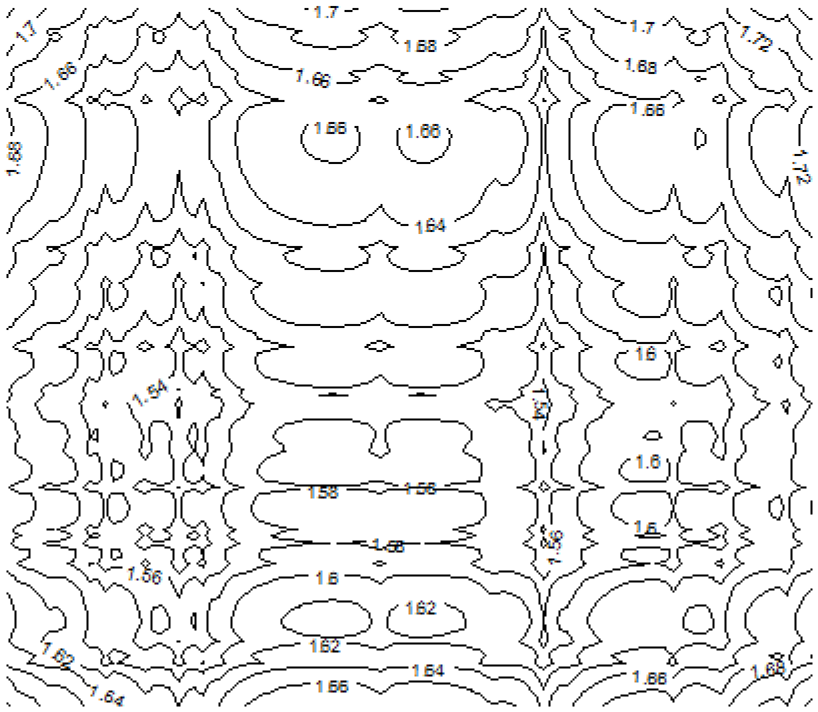


Figure 5C: *Contour map for H, with n = 20 and p = 0.15*

You can also plot images of *H*, using the function image in R with a gray palette with 300 levels of grey, using the command `image(z, col = gray.colors(300))` where `z` is an *m* x *m* matrix (in R) representing values of *H* computed at *m* x *m* locations (here *m* = 100 and *n* = 20.)

The source code (in R) looks like this:

```
data<-read.table("c:/vincentg/math2r.txt",header=TRUE)
w<-data$H
z <- matrix(w, nrow = 100, ncol = 100, byrow = TRUE)
image(z, col = gray.colors(300))
```

Here the file `math2r.txt` stores the *m* x *m* values of *H* sequentially in a one-column text file, row after row. The first row is the header (equal to *H*.) Note that in order to produce Figure 5, I used `contour(z)` rather than `image(z, col = gray.colors(300))`. In Figure 6 I actually used the function `image` to display *H* values for *p* = 0.5: low values of *H* (corresponding to proximity to centroid) are in a darker color, and unlike the case *p* = 2, you can notice multiple local minima. Figure 6 is based on the same data as Figure 5B. The source code to produce the input file `math2r.txt` can be found here.

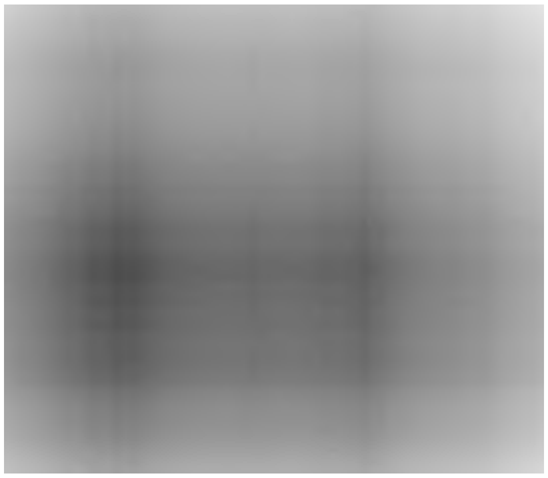


Figure 6*: H values displayed in an image using R (n = 20 and p = 0.5)*

As a bonus, below is Figure 7 corresponding to *p* = -2, which interestingly is very similar to *p* = 0.5 (see Figure 5B.) This value of *p* has tremendous applications in physics, as it corresponds to the inverse-square law.

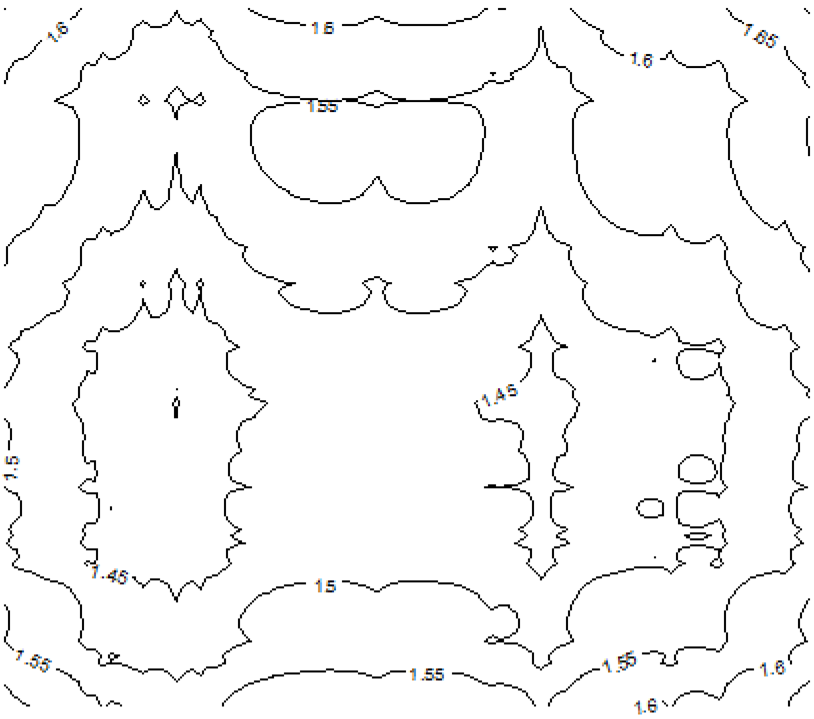


Figure 7*: this time with p=-2*

27. Strong Correlation Metric

The simple *strong correlation* synthetic metric proposed in this chapter should be used whenever you want to check if there is a real association between two variables.

In this chapter, the traditional correlation is referred to as the *weak correlation*, as it captures only a small part of the association between two variables. In short, our *strong correlation* (with a value between 0 and 1) is high (say above 0.80) if not only the weak correlation is also high (in absolute value), but when the internal structures (auto-dependencies) of both variables X and Y that you want to compare, exhibit a similar pattern or correlogram.

Yet this metric is simple and involves just one parameter $\boldsymbol{a}$ (with $\boldsymbol{a} = 0$ corresponding to weak correlation, and $\boldsymbol{a} = 1$ being the recommended value for *strong correlation*). This setting is designed to avoid over-fitting.

What makes two variables X and Y *seem* related is usually based on ordinary (weak) correlation. High strong correlation means that the two variables are really associated and share similar internal auto-dependencies and structure. To put it differently, two variables can be highly weakly correlated yet have no causal relationship (or see my Wiley book pages 165-168) with hidden factors explaining the link. An artificial example is provided below in figure 3. The strong correlation metric helps alleviate this issue, though it does not fix it.

1. Definition of strong correlation

Let's define

- **Weak correlation** $c(X, Y)$ as the absolute value of the ordinary correlation, with value between 0 and 1. This number is high (close to 1) if X and Y are highly correlated. I recommend using my rank-based, L^1 correlation to eliminate problems caused by outliers.
- $c_1(X)$ as the lag-1 auto-correlation in absolute value for X, that is, if $X = (X_1 \dots X_n)$ then $c_1(X) = c(X_1 \dots X_{n-1},\ X_2 \dots X_n)$.
- $c_1(Y)$ as the lag-1 auto-correlation for Y
- *d*-correlation $d(X, Y) = \exp\{ -\boldsymbol{a} \mid \log c_1(X) - \log c_1(Y) \mid \}$, with possible adjustment if the numerator or denominator is zero; the parameter $\boldsymbol{a}$ must be positive or zero. Of course, $d(X, Y)$ is in [0, 1], and close to 1 if X and Y have similar lag-1 auto-correlations.
- **Strong correlation** $r(X, Y) = \min(c(X, Y), d(X, Y))$

Thus $r(X, Y)$ is between 0 and 1, with 1 meaning strong similarity between X and Y, and 0 meaning either dissimilar lag-1 auto-correlations for X and Y, or lack of old-fashioned correlation.

2. Comparison with traditional (weak) correlation

When ***a*** = 0, weak and strong correlations are identical. Also the strong correlation *r*(*X*, *Y*) is symmetric and invariant under linear transformations (such as re-scaling) regardless of ***a***. We simulated more than 10,000 uniformly and independently distributed random variables *Y* each with *n* observations, and computed the correlation with an arbitrary variable *X* with pre-specified values. So you would expect all the correlations to be close to zero. In Figures 1 and 2 below, the horizontal axis represents *c*(*X*, *Y*) and the vertical axis *d*(*X*, *Y*). Note that *r*(*X*, *Y*) = min(*c*(*X*, *Y*), *d*(*X*, *Y*)).

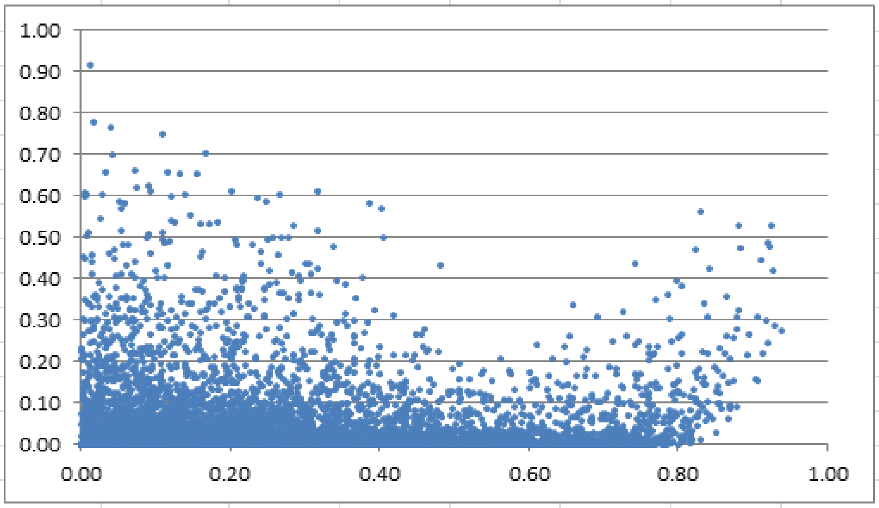


Figure 1: *10,000 (c(X,Y), d(X,Y) values computed on n = 9 observations.*

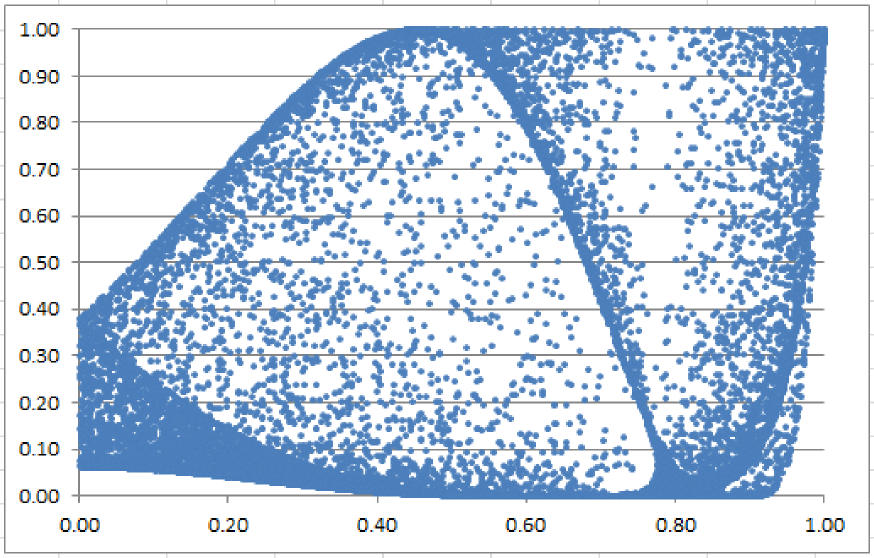


Figure 2: *Same as figure 1, but here with n = 4.*

Many weak correlations are still well above 0.60 if you look at Figure 1.But few strong correlations are above 0.20. Figure 2 is more difficult to interpret visually because *n* is too small (n = 4), though the conclusion is similar and obvious if you check the results in the spreadsheet (see next section). In this example, ***a*** = 4.

3. Excel spreadsheet with computations and examples

The spreadsheet shows simulation of a variable X with n observations, stored in first row, with thousands of simulated Y's in the subsequent rows. There are two tabs: one for n = 4, and one for n = 9. For instance, in the n = 9 tab, column J represents the weak correlation $c(X, Y)$, column M represents $c_1(Y)$, and column N represents the strong correlation $r(X, Y)$. The parameter ***a*** is stored in cell P1, and summary stats are found in cells Q1:T12. The spreadsheet is a bit unusual in the sense that rows represent variables, and columns represent observations. Download the spreadsheet (about 20 MB in compressed format.)

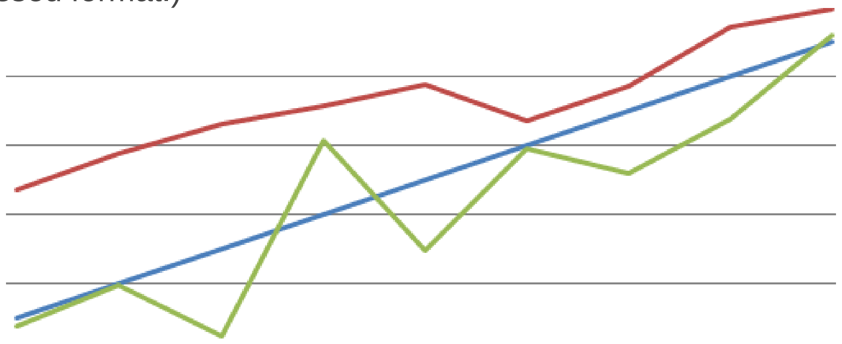


Figure 3: *The green series is highly "weakly correlated" but weakly "strongly correlated" to the blue and red series (**a** = 4)*

Confidence intervals for these correlations are easy to obtain, by running 10 times these simulations and see what min and max you get

4. When to use strong versus weak correlation?

The strong correlation is useful when comparing millions of small, local time series, for instance in the context of HFT (High Frequency Trading), when you try to find cross-correlations with time lags among thousands of stocks. Note that ***a*** = 4 (as used in my spreadsheet) is too high in most situations, and I recommend ***a*** = 1, which has the following advantages:

- Simplification of the formula for $r(X, Y)$
- The fact that $d(X, Y)$ is a raw un-transformed number, and thus likely to be more comparable with $c(X, Y)$.

In the spreadsheet, when n = 4 and ***a*** = 4, about 40% of all weak correlations $c(X, Y)$ are above 0.60, while only 5% of strong correlations $r(X, Y)$ are above 0.60. All the simulated Y's are uniform, random, independent variables, so it is a bit surprising to see so many strong but accidental (spurious) "weak correlations". It happens because n is small. Even with n = 9, the contrast between weak and strong correlations are still significant. The strong correlation metric clearly eliminates a very large chunk of the spurious correlations, especially when ***a*** > 2. But it can eliminate true correlations as well, thus my recommendation to use ***a*** = 1, as a compromise. A high value for ***a*** has effects similar to over-fitting and should be avoided.

5. Generalization

It is possible to take into account and add auto-correlations of lag 1, 2, and so on to generalize the concept of strong correlation, but it may cause overfitting, except if we put decaying weights on the various lags.

Also, it would be great to do this analysis on actual data, not just simulated random noise. Or even on non-random simulated data, using for instance the artificially correlated data set described in chapter 2 (section 2.) Finally there are other metrics available to measure other forms of correlations (for instance on unusual domains), see for instance my article on structuredness coefficient.

6. Other synthetic metrics

The strong correlation is a synthetic metric, and belongs to a family of synthetic metrics that I created over the last few years. Synthetic metrics are designed to efficiently solve a problem, rather than being crafted for their beauty, elegancy and mathematical properties: they are directly derived from data experiments (bottom-up approach) rather than the other way around (top-down: from theory to application) as in traditional science. Other synthetic metrics include:

- Synthetic variance
- Predictive power (see chapter 4) related to entropy (that is, information quantification), used for feature selection.
- Robust correlation defined by an algorithm and closely related to the optimum variance metric discussed here.
- Structuredness coefficient
- Bumpiness coefficient

28. Additional Topics

In this chapter, we briefly cover a number of machine learning topics ranging from stochastic geometry to pattern recognition and extreme events. The first section is non-technical but provides valuable information about what data science is about. ML stands for Machine Learning.

1. Comparing ML, Data Science, AI, Deep Learning, and Statistics

Here, I clarify the various roles of the data scientist, and how data science compares and overlaps with related fields such as machine learning, deep learning, AI, statistics, IoT, operations research, and applied mathematics. As data science is a broad discipline, I start by describing the different types of data scientists that one may encounter in any business setting: you might even discover that you are a data scientist yourself, without knowing it. As in any scientific discipline, data scientists may borrow techniques from related disciplines, though we have developed our own arsenal, especially techniques and algorithms to handle very large unstructured data sets in automated ways, even without human interactions, to perform transactions in real-time or to make predictions.

1.1. Different Types of Data Scientists

To get started and gain some historical perspective, you can read my article about 9 types of data scientists, published in 2014, or my article where I compare data science with 16 analytic disciplines, also published in 2014.

The following articles, published during the same time period, are still useful:

- Data Scientist versus Data Architect
- Data Scientist versus Data Engineer
- Data Scientist versus Statistician
- Data Scientist versus Business Analyst

More recently (August 2016) Ajit Jaokar discussed Type A (Analytics) versus Type B (Builder) data scientist:

- *The Type A Data Scientist can code well enough to work with data but is not necessarily an expert. The Type A data scientist may be an expert in experimental design, forecasting, modelling, statistical inference, or other things typically taught in statistics departments. Generally speaking though, the work product of a data scientist is not "p-values and confidence intervals" as academic statistics sometimes seems to suggest (and as it sometimes is for traditional statisticians working in the pharmaceutical industry, for example). At Google,* *Type A Data Scientists are known variously as Statistician, Quantitative Analyst, Decision Support Engineering Analyst, or Data Scientist, and probably a few more.*

- *Type B Data Scientist: The B is for Building. Type B Data Scientists share some statistical background with Type A, but they are also very strong coders and may be trained software engineers. The Type B Data Scientist is mainly interested in using data "in production." They build models which interact with users, often serving recommendations (products, people you may know, ads, movies, search results). Source: click here.*

I also wrote about the ABCD's of business processes optimization where D stands for data science, C for computer science, B for business science, and A for analytics science. Data science may or may not involve coding or mathematical practice, as you can read in my article on low-level versus high-level data science. In a startup, data scientists generally wear several hats, such as executive, data miner, data engineer or architect, researcher, statistician, modeler (as in predictive modeling) or developer. While the data scientist is generally portrayed as a coder experienced in R, Python, SQL, Hadoop and statistics, this is just the tip of the iceberg, made popular by data camps focusing on teaching some elements of data science. But just like a lab technician can call herself a physicist, the real physicist is much more than that, and her domains of expertise are varied: astronomy, mathematical physics, nuclear physics (which is borderline chemistry), mechanics, electrical engineering, signal processing (also a sub-field of data science) and many more. The same can be said about data scientists: fields are as varied as bioinformatics, information technology, simulations and quality control, computational finance, epidemiology, industrial engineering, and even number theory.

In my case, over the last 10 years, I specialized in machine-to-machine and device-to-device communications, developing systems to automatically process large data sets, to perform automated transactions: for instance, purchasing Internet traffic or automatically generating content. It implies developing algorithms that work with unstructured data, and it is at the intersection of AI (artificial intelligence,) IoT (Internet of things,) and data science. This is referred to as deep data science. It is relatively math-free, and it involves relatively little coding (mostly API's), but it is quite data-intensive (including building data systems) and based on brand new statistical technology designed specifically for this context.

Prior to that, I worked on credit card fraud detection in real time. Earlier in my career (circa 1990) I worked on image remote sensing technology, among other things to identify patterns (or shapes or features, for instance lakes) in satellite images and to perform image segmentation: at that time my research was labeled as computational statistics, but the people doing the exact same thing in the computer science department next door in my home university, called their research artificial intelligence. Today, it would be called data science or artificial intelligence, the sub-domains being signal processing, computer vision or IoT.

Also, data scientists can be found anywhere in the lifecycle of data science projects (see section 13), at the data gathering stage, or the data exploratory stage, all the way up to statistical modeling and maintaining existing systems.

1.2. Machine Learning versus Deep Learning

Before digging deeper into the link between data science and machine learning, let's briefly discuss machine learning and deep learning. Machine learning is a set of algorithms that train on a data set to make predictions or take actions in order to optimize some systems. For instance, supervised classification algorithms are used to classify potential clients into good or bad prospects, for loan purposes, based on historical data. The techniques involved, for a given task (e.g. supervised clustering), are varied: naive Bayes, SVM, neural nets, ensembles, association rules, decision trees, logistic regression, or a combination of many. For a detailed list of algorithms, click here. For a list of machine learning problems, click here.

All of this is a subset of data science. When these algorithms are automated, as in automated piloting or driver-less cars, it is called AI, and more specifically, deep learning. Click here for another article comparing machine learning with deep learning. If the data collected comes from sensors and if it is transmitted via the Internet, then it is machine learning or data science or deep learning applied to IoT.

Some people have a different definition for deep learning. They consider deep learning as neural networks (a machine learning technique) with a deeper layer. The question was asked on Quora recently, and below is a more detailed explanation (source: Quora)

- *AI (Artificial intelligence) is a subfield of computer science, that was created in the 1960s, and it was (is) concerned with solving tasks that are easy for humans, but hard for computers. In particular, a so-called Strong AI would be a system that can do anything a human can (perhaps without purely physical things). This is fairly generic, and includes all kinds of tasks, such as planning, moving around in the world, recognizing objects and sounds, speaking, translating, performing social or business transactions, creative work (making art or poetry), etc.*

- *NLP (Natural language processing) is simply the part of AI that has to do with language (usually written).*

- *Machine learning is concerned with one aspect of this: given some AI problem that can be described in discrete terms (e.g. out of a particular set of actions, which one is the right one), and given a lot of information about the world, figure out what is the "correct" action, without having the programmer program it in. Typically some outside process is needed to judge whether the action was correct or not. In mathematical terms, it's a function: you feed in some input, and you want it to to produce the right output, so the whole problem is simply to build a model of this mathematical function in some automatic way. To draw a distinction with AI, if I can write a very clever program that has human-like* *behavior, it can be AI, but unless its parameters are automatically learned from data, it's not machine learning.*

- *Deep learning is one kind of machine learning that's very popular now. It involves a particular kind of mathematical model that can be thought of as a composition of simple blocks (function composition) of a certain type, and where some of these blocks can be adjusted to better predict the final outcome.*

1.3. What is the difference between machine learning and statistics?

This article tries to answer the question. The author writes that statistics is machine learning with confidence intervals for the quantities being predicted or estimated. I tend to disagree, as I have built engineer-friendly confidence intervals (see also chapter 16) that don't require any mathematical or statistical knowledge.

1.4. Data Science versus Machine Learning

Machine learning and statistics are part of data science. The word *learning* in machine learning means that the algorithms depend on some data, used as a training set, to fine-tune some model or algorithm parameters. This encompasses many techniques such as regression, naive Bayes or supervised clustering. But not all techniques fit in this category. For instance, unsupervised clustering - a statistical and data science technique - aims at detecting clusters and cluster structures without any a-priori knowledge or training set to help the classification algorithm. A human being is needed to label the clusters found. Some techniques are hybrid, such as semi-supervised classification. Some pattern detection or density estimation techniques fit in this category.

Data science is much more than machine learning though. Data, in data science, may or may not come from a *machine* or mechanical process (survey data could be manually collected, clinical trials involve a specific type of small data) and it might have nothing to do with *learning* as I have just discussed. But the main difference is the fact that data science covers the whole spectrum of data processing, not just the algorithmic or statistical aspects. In particular, data science also covers

- Data integration
- Distributed architecture
- Automating machine learning
- Data visualization
- Dashboards and BI
- Data engineering
- Deployment in production mode
- Dutomated, data-driven decisions

Of course, in many organizations, data scientists focus on only one part of this process.

2. Distribution of Arrival Times for Extreme Events

Most of the articles on extreme events are focusing on the extreme values. Very little has been written about the arrival times of these events. This chapter fills the gap.

We are interested here in the distribution of arrival times of successive records in a time series, with potential applications to global warming assessment, sport analytics, or high frequency trading. The purpose here is to discover what the distribution of these arrival times is, in absence of any trends or auto-correlations, for instance to check if the global warming hypothesis is compatible with temperature data obtained over the last 200 years. In particular it can be used to detect subtle changes that are barely perceptible yet have a strong statistical significance. Examples of questions of interest are:

- How likely is it that 2016 was the warmest year on record, followed by 2015, then by 2014, then by 2013?
- How likely is it, in 200 years' worth of observations, to observe four successive records four years in a row, at any time during the 200 years in question?

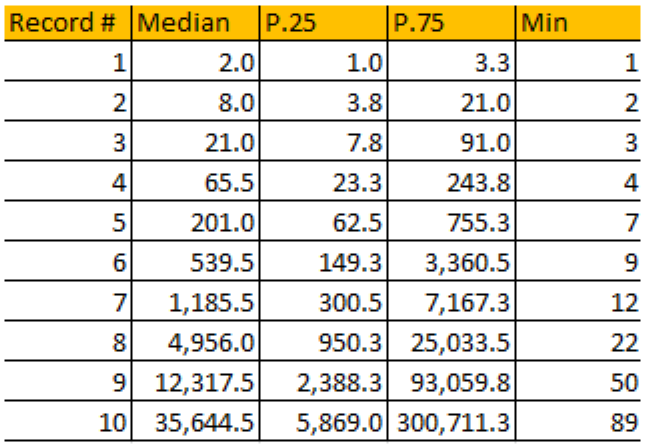
The answer to the first question is that it is very unlikely to happen just by chance.

Despite the relative simplicity of the concepts discussed here, and their great usefulness in practice, none of the material below is found in any statistics textbook, as far as I know. It would be good material to add to any statistics curriculum.

2.1. Simulations

I run a number of simulations, generating 100 time series each made up of millions of random, independent Gaussian deviates, without adding any trend up or down. The first few hundred points of one of these time series is pictured in Figure 1.

I computed the median, 25- and 75-percentiles for the first few records, see Figure 1. For instance, the median time of occurrence of the first record (after the first measurement) is after 2 years, if your time unit is a year. The next bigger record is expected 8 years after the first measurement and the next bigger one 21 years after the first measurement (see Figure 1.) Even if you look at the 25-percentile, it really takes a lot of years to beat the previous 4 or 5 records in a row. In short, it is nearly impossible to observe increasing records four years in a row, unless there is a trend that forces the observed values to become larger over time.



Record #	Median	P.25	P.75	Min
1	2.0	1.0	3.3	1
2	8.0	3.8	21.0	2
3	21.0	7.8	91.0	3
4	65.5	23.3	243.8	4
5	201.0	62.5	755.3	7
6	539.5	149.3	3,360.5	9
7	1,185.5	300.5	7,167.3	12
8	4,956.0	950.3	25,033.5	22
9	12,317.5	2,388.3	93,059.8	50
10	35,644.5	5,869.0	300,711.3	89

Figure 1*: Time of arrivals of successive records (in years if you time unit is a year)*

This study of arrival times for these records should allow you to detect even very tiny trends, either up or down, better than traditional models of change point detection hopefully. However it does not say anything about whether the increase is barely perceptible or rather large.

Note that the values of these records are a subject of much interest in statistics, known as extreme value theory. This theory has been criticized for failure to predict the amount of damage in modern cataclysms, resulting in big losses for insurance companies. Part of the problem is that these models are based on hundreds of years' worth of data (for instance to predict the biggest flood that can occur in 500 years) but over such long periods of time, the dynamics of the processes at play have shifted. Note that here, I focus on the arrival times or occurrences of these records, not on their intensity or value, contrarily to traditional extreme value theory.

Finally, arrival times for these records do not depend on the mean or variance of the underlying distribution. Figure 1 provides some good approximations, but more tests and simulations are needed to confirm my findings. Are these median arrival times the same regardless of the underlying distribution (temperature, stock market prices, and so on) just like the central limit theorem provides a same limiting distribution regardless of the original, underlying distribution? The theoretical statistician should be able to answer this question. I didn't find many articles on the subject in the literature, though this one is interesting. In the next section, I try to answer this question. The answer is positive.

2.2. Theoretical Distribution of Records over Time

This is an interesting combinatorial problem, and it bears some resemblance to the Analyticbridge Theorem. Let R_n be the value of the n^{th} record (n = 1, 2,...) and T_n its arrival time.

For instance, if the data points (observed values) are X_0 = 1.35, X_1 = **1.49**, X_2 = 1.43, X_3 = **1.78**, X_4 = 1.63, X_5 = 1.71, X_6 = 1.45, X_7 = **1.93**, X_8 = 1.84, then the records (highlighted in bold) are $R_1 = 1.49$, $R_2 = 1.78$, $R_3 = 1.93$, and the arrival times for these records are $T_1 = 1$, $T_2 = 3$, and $T_3 = 7$.

To compute the probability $P(T_n = k)$ for $n > 0$ and $k = n, n+1, n+2$, etc., let's define $T_{n, m}$ as the arrival time of the n^{th} record if we only observe the first $m+1$ observations X_0, X_1, ..., X_m. Then $P(T_n = k)$ is the limit of $P(T_{n, m} = k)$ as m tends to infinity, assuming the limit exists. If the underlying distribution of the values X_0, X_1, etc. is continuous, then, due to the symmetry of the problem, computing $P(T_{n, m} = k)$ can be done as follows:

1. Create a table of all $(m+1)!$ (factorial $m+1$) permutations of $(0, 1, \dots, m)$.
2. Compute $N(n, m, k)$, the number of permutations of $(0, 1, \dots, m)$ where the n^{th} record occurs at position k in the permutation (with $0 < k \leq m$). For instance, if $m = 2$, we have 6 permutations (0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1) and (2, 1, 0). The first record occurs at position $k = 1$ only for the following three permutations: (0, 1, 2), (0, 2, 1), and (1, 2, 0). Thus, $N(1, 2, 1) = 3$. Note that the first element in the permutation is assigned position 0, the second one is assigned position 1, and so on. The last one has position m.
3. Then $P(T_{n, m} = k) = N(n, m, k) / (m+1)!$

As a result, the distribution of arrival times, for the records, is universal: it does not depend on the underlying distribution of the identically and independently distributed observations X_0, X_1, X_2 etc.

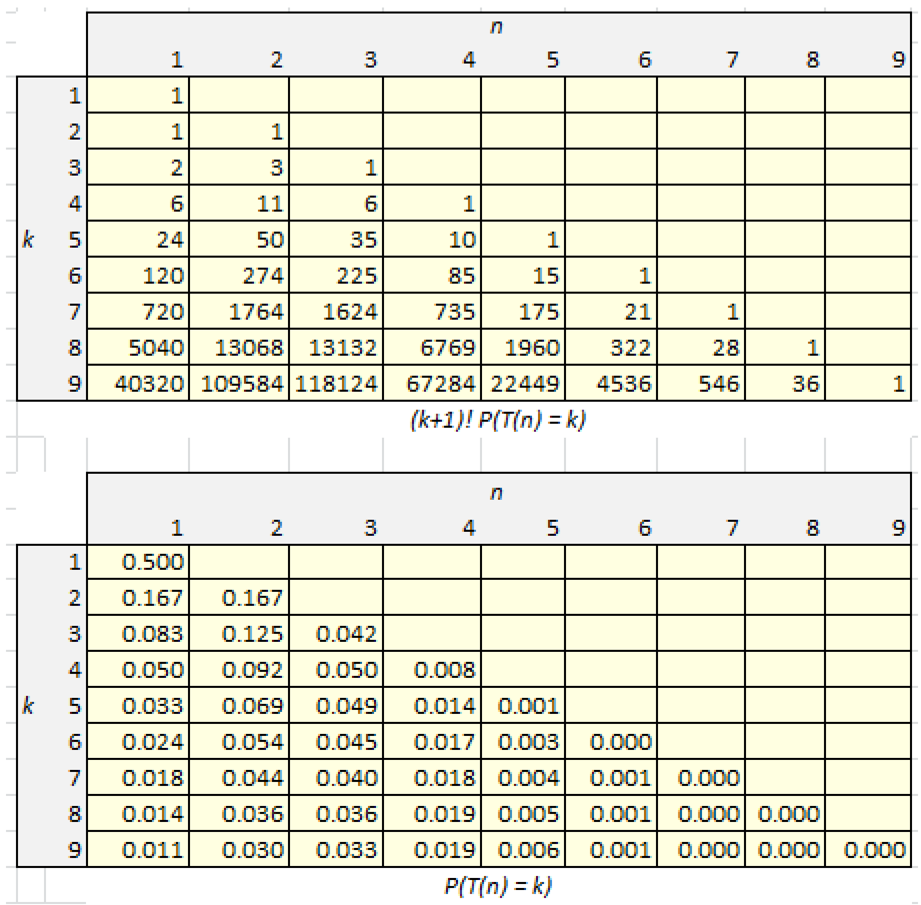
It is easy (with or without using my above combinatorial framework) to find that the probability to observe a record (any record) at position k is $1/(k+1)$ assuming again that the first position is position 0 (not 1). Also, it is easy to prove that $P(T_n = n) = 1/(n+1)!$. Now, $T_1 = k$ if and only if X_k is a record among $X_0, \dots, X_k$ and X_0 is the largest value among $X_0, \dots, X_{k-1}$. Thus:

$$P(T_1 = k) = 1 / \{ (k+1)k \}$$

This result is confirmed by my simulations. For the general case, recurrence formulas can be derived.

2.3. Useful Results

None of the arrival times T_n for the records has a finite expectation. Figure 2 displays the first few values for the probability that the n^{th} record occurs at position $T_n = k$, the first element in the data set being assigned to position 0. The distribution of these arrival times does not depend on the underlying distribution of the observations.



		n								
		1	2	3	4	5	6	7	8	9
	1	1								
	2	1	1							
	3	2	3	1						
	4	6	11	6	1					
k	5	24	50	35	10	1				
	6	120	274	225	85	15	1			
	7	720	1764	1624	735	175	21	1		
	8	5040	13068	13132	6769	1960	322	28	1	
	9	40320	109584	118124	67284	22449	4536	546	36	1

(k+1)! P(T(n) = k)

		n								
		1	2	3	4	5	6	7	8	9
	1	0.500								
	2	0.167	0.167							
	3	0.083	0.125	0.042						
	4	0.050	0.092	0.050	0.008					
k	5	0.033	0.069	0.049	0.014	0.001				
	6	0.024	0.054	0.045	0.017	0.003	0.000			
	7	0.018	0.044	0.040	0.018	0.004	0.001	0.000		
	8	0.014	0.036	0.036	0.019	0.005	0.001	0.000	0.000	
	9	0.011	0.030	0.033	0.019	0.006	0.001	0.000	0.000	0.000

P(T(n) = k)

Figure 2: *$P(T_n = k)$ at the bottom, $(k+1)!\ P(T_n = k)$ at the top*

These probabilities were computed using a small script that generates all (*k*+1)! permutations of (0, 1, ..., *k*) and checks, among these permutations, those having a record at position *k*: for each of these permutations, we computed the total number of records. If *N*(*n*, *k*) denotes the number of such permutations having *n* records, then $P(T_n = k) = N(n, k) / (k+1)!$.

Despite the fact that the above table is tiny, it required hundreds of millions of computations for its production.

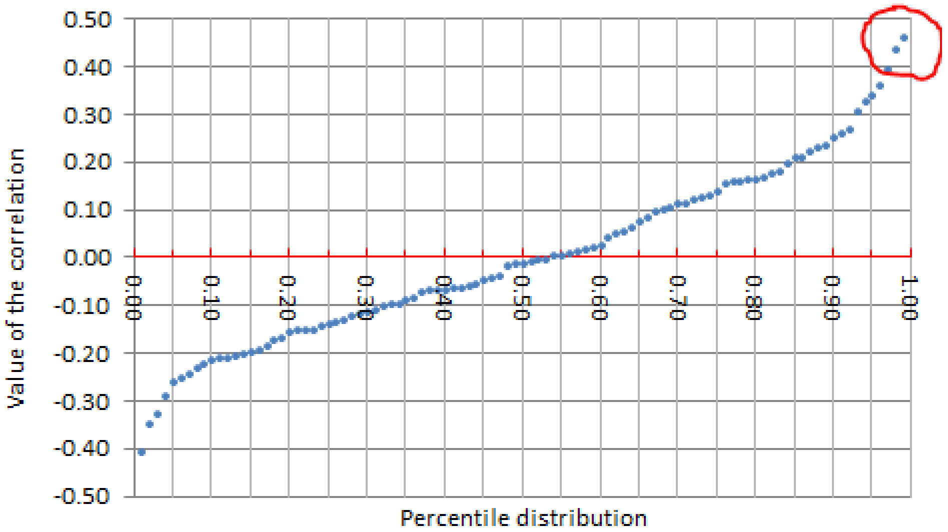
3. How to Lie with *p*-Values?

P-values are used in statistics and scientific publications, much less so in machine learning applications where re-sampling techniques are favored and easy to implement today thanks to modern computing power. In some sense, *p*-values are a relic from old times, when computing power was limited and mathematical / theoretical formulas were favored and easier to deal with than lengthy computations.

Recently, *p*-values have been criticized and even banned by some journals, because they are used by researchers, who cherry-pick observations and repeat experiments until they obtain a *p*-value worth publishing to obtain grant money, get tenure, or for political reasons. Even the American Statistical Association wrote a long article about why to avoid *p*-values, and what you should do instead: see here. For data scientists, obvious alternatives include re-sampling techniques: see chapters 15 and 16. One advantage is that they are model-independent, data-driven, and easy to understand.

Here we explain how the manipulation and treachery works, using a simple simulated data set consisting of purely random, non-correlated observations. Using *p*-values, you can tell anything you want about the data, even the fact that the features are highly correlated, when they are not. The data set consists of 16 variables and 30 observations, generated using the RAND function in Excel. You can download the spreadsheet here.

There is a total of (16 x 15) / 2 = 120 correlations (one for each pair of variables) and as you compute them one by one, you are bound, sooner rather than later, to find one that is significant. The most extreme correlation will *almost always* be above 0.4 in absolute value if you have 16 variables and 30 observations that are totally random. This is a statistically significant departure from zero. If you pick up that extreme correlation, now you can tell that my data set is not random, and that the chance for such a high correlation to occur is indeed 1/120. This number (1/120) is also your *p*-value, which is well below 5%, the threshold usually accepted to prove that the effect in question did not occur by chance. The truth is that it really did occur by chance: you were just cherry-picking.



The way the scheme works is by picking the least extreme case that meets your agenda (circled in red in the above picture), in this case a target *p*-value below 1%.

If you were to write an article about Excel using this faulty argumentation, you could claim, based on this experiment, that the random number generator in Excel is wrong and produces correlated numbers. You could do the same experiment in Python and come to the same conclusion. Or you could use a genuine hardware-based device that truly produces randomness, and still come to the same conclusion. Indeed you could write a philosophical article about the fact that randomness does not exist. You could also do the same experiment using the Perl programming language and come to the same conclusion. In this latter case interestingly, you would be correct: Perl's random number generator has a major design flaw (it can produce only 32,767 distinct values) but this little experiment would not be able to reveal this fact. You would be correct about Perl's faulty random numbers, but you would be correct just by chance, not because you used a sound methodology to identify the issue.

4. Off-the-beaten-path Machine Learning Topics

You will find here nine interesting topics that you won't learn in college classes. Most have interesting applications in business and elsewhere. They are not especially difficult, and I explain them in simple English. Yet they are not part of the traditional statistical curriculum, and even many experienced data scientists with a PhD degree have not heard about some of these concepts.

4.1. Random walks in one, two and three dimensions

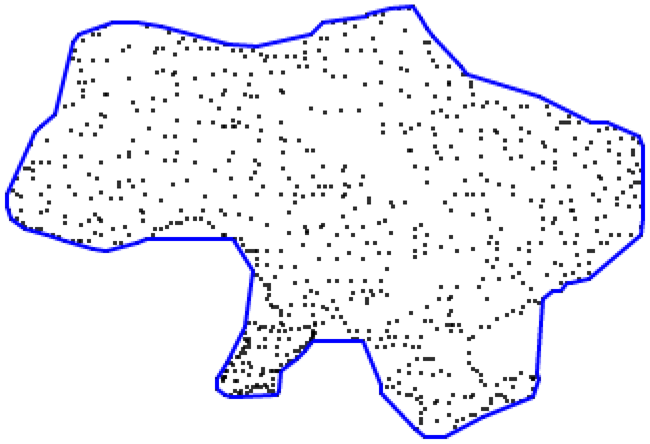
This is a well-known model, used as a base stochastic process to model the logarithm of stock prices, yet it has interesting properties (depending on dimension) that few people know about. In one dimension, it is described as follows: You start at 0 (on the X-axis) and at each iteration, you increase by +1 with probability 0.5, and decrease by +1 with probability 0.5. In one or two dimensions, the probability that it will get back to any previous state at one point, is one. But this is not the case in three dimensions. Yet the most probable number of sign changes (crossing the X-axis) in a walk is 0, followed by 1, then 2, etc. The time spent either above or below the X-axis (before a crossing) is modeled by the arc-sine law: Crossing the X-axis happens rarely. For self-correcting random walks, click here. For videos produced with R, simulating a 2-D random walk; follow this link.

4.2. Estimation of the convex hull of a set of *n* points

In one dimension, this is just the estimation of an interval when points are uniformly distributed, using the minimum and maximum observations, and multiplying the observed length (max - min) by a factor (*n*+1)/*n* to remove the bias. In two dimensions, computing the convex hull is easy, and again you need to expand the shape a little to correct for bias. Convex hulls are used in clustering problems, where clusters are modeled by (possibly) overlapping convex domains: This is a non-parametric alternative to clustering algorithms based on the Gaussian distribution.

A potential application is estimating the shape of an oil field when digging a number of test wells - some within the (unknown) oil field boundary, some (as few as possible) outside the boundary. It is also used to estimate the extent and shape of an underground contaminated area: It was used to identify whether the nuclear waste from the Hanford nuclear reservation, was spilling in the Columbia river located a few hundred yards away, and whether it got worse over time, by measuring chromium levels in a number of wells.

How about designing a fast algorithm to compute the convex hull of a set of points, in any dimension? This is a great exercise for a data scientist, but first you need to check the literature about existing algorithms. I implemented one when I was working on my PhD in computational statistics.



The first step to estimate this complex shape is to start with the convex hull (click here for details)

4.3. Constrained linear regression on unusual domains

Lasso and ridge regression are popular examples of constrained linear regression: Constraints are put on the regression coefficients to make it more stable, for instance, the coefficient between a dependent and independent variable must have the same sign as the correlation between the two variables in question. Such constraints are used for instance in the HDT algorithm, (see chapter 2) which is an hybrid regression / pseudo decision tree procedure.

In some cases, the constraints are dictated by the business problem itself. For instance, if a response depends on a mix of chemical ingredients (think about the taste of a beverage - how people like it or not) the weight or proportion attached to each ingredient is a regression coefficient: All these coefficients must be positive or zero, and they must add up to one. This is known as linear regression on the simplex domain. Click here for more similar problems (regression on a sphere and so on.)

4. Robust and scale-invariant variances

The traditional variance is impacted by erroneous data and outliers, and thus not very robust. I proposed a new variance that is more robust, and always positive, just like the standard variance. The positivity is guaranteed by the Jensen inequality, and from a mathematical point of view, it is a metric between an L^1 and L^2 version of the classical variance (L^2 yields the classical variance.) Click here for details.

I am currently working on a variance that is scale-invariant (also described in the same article) and this is really a bizarre object, though useful when the variance should stay the same, whether your metric is measured in miles or kilometers. The next step is to design scale-invariant clustering algorithms, as the scale of each variable (the units used for measurement) sometimes has a bigger impact on the resulting clusters, than the choice of the clustering algorithm itself.

4.5. The Tweedie distributions

In statistics, the Tweedie distributions are a family of probability distributions which include the purely continuous normal and gamma distributions, the purely discrete scaled Poisson distribution, and the class of mixed compound Poisson–gamma distributions which have positive mass at zero, but are otherwise continuous. Just like the exponential family of distributions, it includes several popular distributions. These distributions are characterized by the following property: The expectation is proportional to a power of the variance. It has many applications, including for modeling errors in signal processing, and even to model departure from the asymptotic representation in some prime number functions. Click here for details, and to see the various applications, including actuarial studies, survival analysis, ecology, medical applications, meteorology and climatology, fisheries, cancer metastasis, genomic structure and evolution.

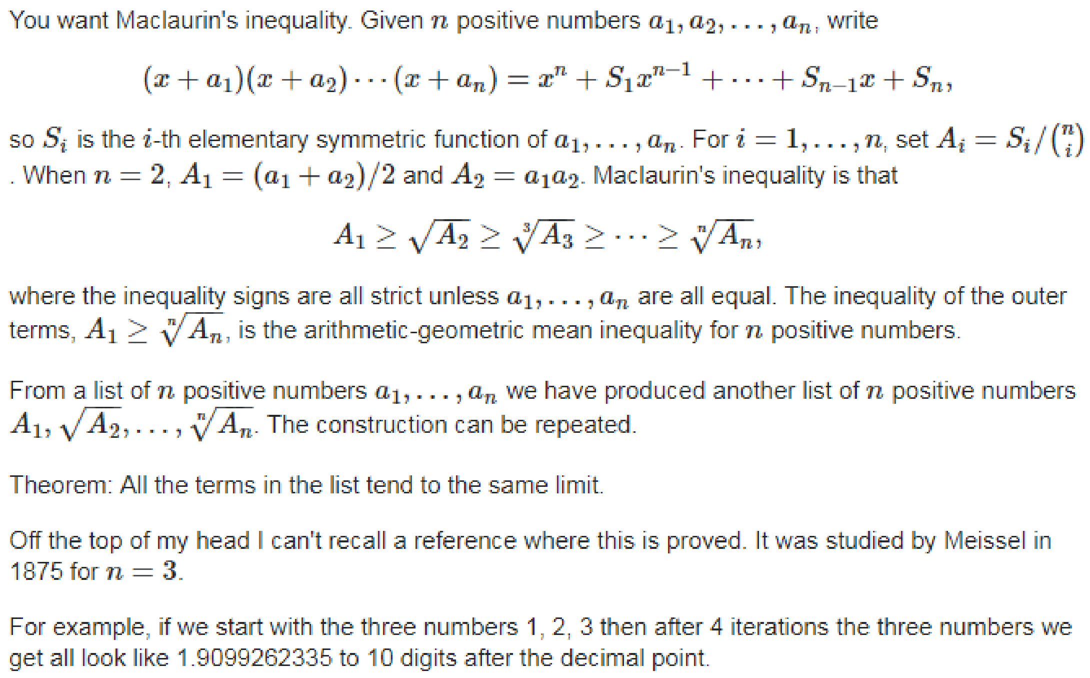
Another distribution with several practical applications is the Zipf distribution.

4.6. The arithmetic-geometric mean

This was initially designed to compute the mean of two numbers, and it comes with a very fast algorithm that converges to a value between the arithmetic and geometric means. It has a number of interesting mathematical properties, and has been used to compute the number π very efficiently (other very fast algorithms to compute π can be found here and here.)

To compute the arithmetic-geometric mean of two numbers, start with two initial estimates a_0 and b_0 equal respectively to the geometric and arithmetic mean. At each iteration k, compute a_k as the geometric mean of a_{k-1} and b_{k-1}, and compute b_k as the arithmetic mean of a_{k-1} and b_{k-1}. Both a_k and b_k converge very fast to the arithmetic-geometric mean. Click here for details.

It has been generalized to any number of variables, see here. The picture below summarizes one of the most interesting generalizations, involving a bunch of interesting averaging functions, besides the arithmetic and geometric means.


You want Maclaurin's inequality. Given n positive numbers $a_1, a_2, \dots, a_n$, write

$$(x+a_1)(x+a_2)\cdots(x+a_n) = x^n + S_1 x^{n-1} + \cdots + S_{n-1}x + S_n,$$

so S_i is the i-th elementary symmetric function of $a_1, \dots, a_n$. For $i = 1, \dots, n$, set $A_i = S_i / \binom{n}{i}$. When $n = 2$, $A_1 = (a_1 + a_2)/2$ and $A_2 = a_1 a_2$. Maclaurin's inequality is that

$$A_1 \geq \sqrt{A_2} \geq \sqrt[3]{A_3} \geq \cdots \geq \sqrt[n]{A_n},$$

where the inequality signs are all strict unless $a_1, \dots, a_n$ are all equal. The inequality of the outer terms, $A_1 \geq \sqrt[n]{A_n}$, is the arithmetic-geometric mean inequality for n positive numbers.

From a list of n positive numbers $a_1, \dots, a_n$ we have produced another list of n positive numbers $A_1, \sqrt{A_2}, \dots, \sqrt[n]{A_n}$. The construction can be repeated.

Theorem: All the terms in the list tend to the same limit.

Off the top of my head I can't recall a reference where this is proved. It was studied by Meissel in 1875 for $n = 3$.

For example, if we start with the three numbers 1, 2, 3 then after 4 iterations the three numbers we get all look like 1.9099262335 to 10 digits after the decimal point.

4.7. Weighted version of the *K*-NN clustering algorithm

It can be used to estimate the local or global intensity of a stochastic point process, and also related to density estimation techniques. How many neighbors should we use, and which weights should we put on these neighbors to get robust and accurate estimates? It turned out that putting more weight on close neighbors, and increasingly lower weight on far away neighbors (with weights slowly decaying to zero based on the distance to the neighbor in question) was the solution to the problem. I actually found optimum decaying schedules for the weights a_k attached to the k^{th} nearest neighbor, as k tends to infinity. You can read the details here. Obviously this can also be used when implementing clustering techniques based on the well-known *K*-NN algorithm (*k* nearest neighbors.)

For another generalization of the *K*-NN classifier, based on graph theory, see section 6. This version of *K*-NN can also be used for variable reduction while preserving the dimension of the original data set.

4.8. Multivariate exponential distribution and storm modeling

Intensity and duration of storm cells have been traditionally modeled using Gaussian distributions. Bivariate exponential distributions with negative correlation provide more flexibility and a better representation of the real world, that is, superior goodness of fit with actual data. You can read more about this topic and about how to simulate a multivariate exponential distribution with specific covariance matrix and known marginal distributions, here (PDF document.)

There is a limit on how negative the coefficient of correlation of a bivariate exponential distribution can be, and this is pictured in the theorem below (from the same paper):

Theorem 3.1 *If U and V are correlated random variables with exponential marginal distributions and possibly different marginal means, then*

$$\rho(U,V) \geq -1 + \int_0^1 \log(x)\log(1-x)dx = 1 - \frac{\pi^2}{6} \approx -0.644.$$

5. Variance, Clustering, and Density Estimation Revisited

We propose here a simple, robust and scalable technique to perform *supervised clustering* on numerical data. It can also be used for density estimation, and even to define a concept of variance that is scale-invariant. This is part of our general statistical framework for data science.

5.1. General Principle: Working on the Grid, not on the Original Space

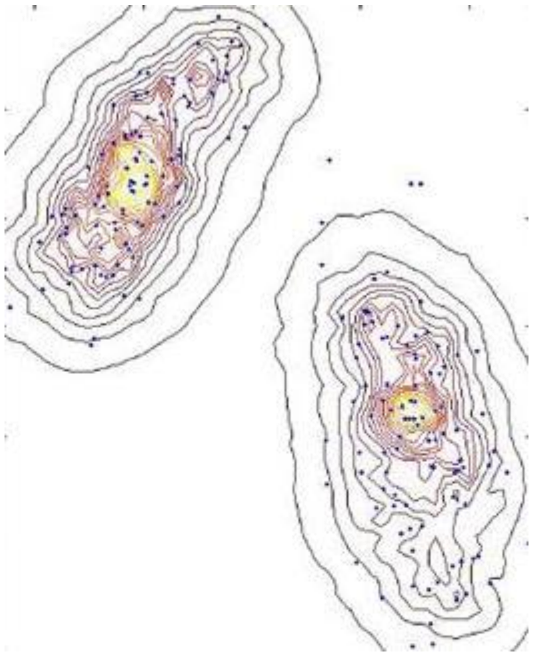
Here we discuss clustering and density estimation on the grid. The grid can be seen as an 2-dimensional or 3-dimensional array. We assume that you have selected your best predictors (for instance using our feature selection algorithm in chapter 5) so that the loss of yield, predictability, or accuracy, due to working in smaller dimensions, is minimum (and measurable), and is more than compensated by an increase in stability, scalability, simplicity, and robustness.

In addition, all your observations have been linearly transformed and discretized (the numerical values have been rounded) so that each observation, after this mapping, occupies a cell in the grid. For instance, if the grid associated with the data set in question consists of 1,000 columns and 2,000 rows, it can store at least 1,000 x 2,000 observations. It is OK to have two distinct but very close observations, mapped onto the same cell in the grid. Also, the worst outliers (say the top 10 most extreme points) are either ignored or put on the border of the grid, to avoid having an almost empty grid because of a few extreme outliers. Transforming data (using log of salary rather than salary) will also help here, as in any statistical methodology. As **rule of thumb #1**, I suggest that 30% of the cells (in the grid) should have at least one observation attached to them. At the end of the day, you can try with different data coverage (above or below 30% of the grid) until you get optimum results based on cross-validation testing.

Note that in the above example - a 1,000 x 2,000 grid - your rounded values in your data set, once mapped onto the grid, have accuracy above 99.9%. Most data collection processes have errors far worse than that, so the impact of discretizing your observations is almost zero, at least in most applications. Again, compare (using *confusion matrices* on test data from a cross-validation design) your full, *exact* supervised clustering system with this approximate setting, and you should not experience any significant prediction loss, in most applications.

Finally, this 1,000 x 2,000 grid (used for illustration purposes) fits easily in memory (RAM). You can even go to 4 dimensions: (say) 500 x 500 x 500 x 500 grid, and store that grid in memory, or slice it into 100 overlapping sub-grids, and do the processing with a Map-Reduce mechanism (in Hadoop for instance) on each sub-grid separately and in parallel.

For our purpose, the value of a cell in the grid will be in an integer between 0 and 255. In some cases, it could be just 0 or 1, with 1 meaning that there is a training set data point close to the location in question in the grid, 0 meaning that you are far enough away from any neighbor.



Non parametric density estimation (source: click here)

5.2. Density Estimation

We start with density estimation, as this is the base (first step) for the supervised clustering algorithm. We assume that we have *g* groups or classes: that is, a training set consisting of *g* known groups -- each observation (*x*, *y*) having a label representing its group. For simplicity, let's consider the 2-dimensional case.

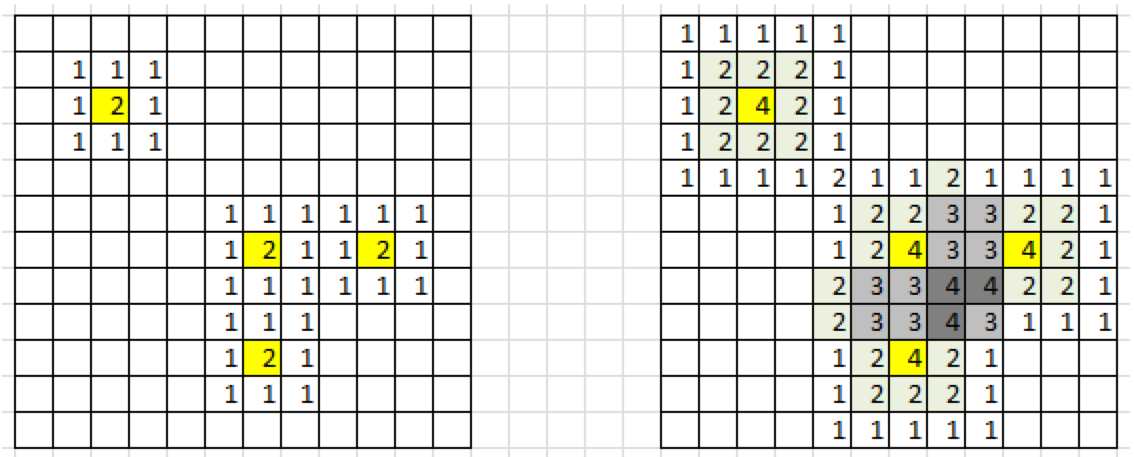


Figure 1: *data points in yellow; 3x3 kernel is too small (left), 5x5 kernel is OK (right)*

In figure 1, yellow cells represent locations corresponding to an actual observation (*x*, *y*) in the training set: that is, bi-variate coordinates of a point, where x could (for instance) represent the rounded monthly payment associated with a loan, and y the rounded salary, after log transformation of the salary. The groups could represent the risk level (risk of default on loan repayment), with three categories: low, medium, high.

To compute density estimates on each cell of the grid, draw a 3x3 window around each yellow cell, and add 1 to all locations (cells) in that 3x3 window. Or better, draw a 5x5 window around each yellow cell, add 1 at the center, add 2 at each location in the 3x3 sub-window, and add 1 to all locations at the border (inside) the 5x5 window. See figure 1 for illustration.

You can use bigger windows, circular windows and even infinite windows where the weight (the value added to each cell) decays exponentially fast with the distance to the center. I do not recommend it though, as it would allow you to classify any new observation (outside the training set) even if they are far away from the closest neighbor: this leads to misclassification; such extreme observations should remain un-classified. Finally, this methodology works too in higher dimensions (3- or 4-dim).

The result is (see Figure 1, right picture, for a specific group): after applying this procedure to each yellow cell and each class, you have density estimates for each class, for each cell. If we define *N*(*z*) as the value (density estimate for a specific group) computed on a particular cell z in the grid, **rule of thumb #2** says that 10% of the cells with *N*(*z*) > 0 must have an *N*(*z*) that is contributed by more than one neighboring data point. In figure 1, the percentages are respectively 0% for the left picture (3x3 window), and 13/79 = 16% for the right picture (5x5 window). So clearly, the 5x5 window is OK, but the 3x3 is not. If you violate this rule, try with a bigger window, e.g. 7x7 rather than 5x5, or 5x5 rather than 3x3.

How to handle non-numeric variables?

In our example in the previous section, if one of the variables is gender (M/F) and another one is age (young / medium / old), multiply the number of groups accordingly. In our example, the number of groups (*g*=3: low risk, medium risk, high risk of default) will be multiplied by 2 (M / F) x 3 (young / medium / old), resulting in 18 groups, for instance "young females with medium risk of default" being one of these 18 groups.

5.3. Supervised Clustering

Now we have a straightforward classification rule: for a new data point to be classified (a point that typically does not belong to the training set), with cell location equal to *z* in the grid, compute the density estimate $N(z \mid c)$ for each group *c*, then assign the point to the group c maximizing $N(z \mid c)$. The speed of this algorithm is phenomenal: the computational complexity is $O(g)$ where *g* is the number of groups. If you pre-classify your entire (say) 1,000 x 2,000 grid, then it is even faster, equal to $O(1)$. You can't beat that. Of course, accessing a cell in the grid (represented by a 2-dim array), while extremely fast and not depending on the number of observations, still requires a tiny bit of time, but it is entirely dependent only on the size of the array, and its dimension. In higher dimensions, where Map-Reduce strategies are used, more time is used to access a cell of the grid and return its value, yet it is nothing compared with the time required to perform standard supervised clustering.

Optimizing the Computations

In 2 dimensions, we have a little trick to compute the density estimates for each cell in the grid much faster, in a systematic way: it is illustrated in Figure 2 below. It also works in higher dimensions, though it is most efficient in 2 dimensions.

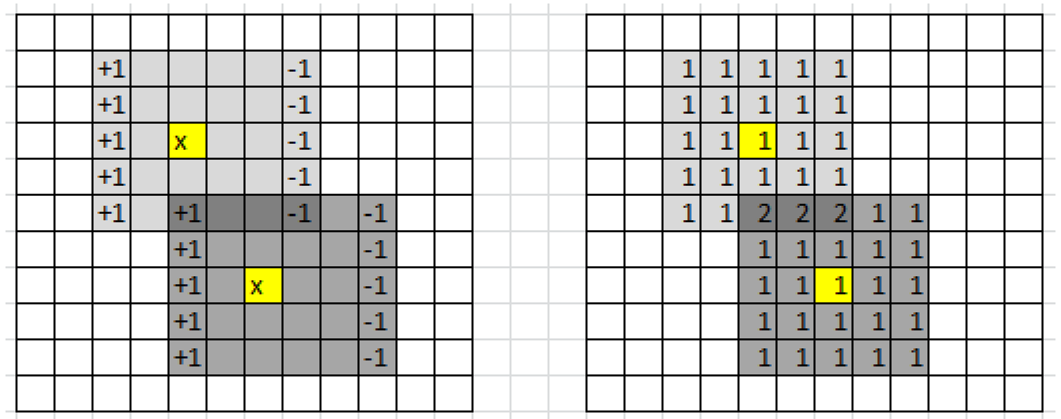


Figure 2: *data points in yellow; trick to compute densities row by row, to reduce computing time*

5.4. Scale-Invariant Variance

I have designed a scale-invariant variance not so long ago: you can check it out here. Interestingly, it relies on convex functions. The proof that it is always positive is based on some advanced mathematics, namely, the Jensen inequality.

Here, the purpose is a bit different. We want to create a new, scale-invariant variance, that is minimum when the data points are evenly distributed in some domain, and maximum when there are peaks and valleys or oceans of high and low density. This is still an area of intense research.

Let's denote as N(z) the density estimate at a specific location z (cell) on the grid, as in our earlier sections. The first definition of variance that comes to mind is related to the proportion (denoted as p) of cells z with $N(z) > 1$, among all cells with $N(z) > 0$. You can now define the new variance v as $v = p$. It has interesting properties, but unfortunately, it is dependent on the window size (3x3 or 5x5 as in our figure 1), though this drawback can be mitigated if you abide by our two rules of thumb mentioned above. I invite you to come up with a better definition.

Note that this new definition of variance applies to point distributions in any dimension, not just to univariate observations.

5.5. Historical Notes

This chapter has its origins in one of my earlier papers published in the *Journal of Royal Statistical Society, series B*, back in 1995: Multivariate Discriminant Analysis and Maximum Penalized Likelihood.... At that time, I was working on image analysis problems to automatically determine the proportions of various crops cultivated in several countries, based on satellite image data. This technology was cheaper than sending people in the fields to manually make the measurements via sampling - in short, it was (even back in 1995) an attempt at replacing men with robots. The same technology was used to identify tanks from enemies in the Iraq war. The images consisted of a few channels: infrared, radar, red, green, and blue -- that is, about 5 dimensions. So the (x, y) coordinates mentioned earlier in this chapter were (in this case) color intensities in two of the 5 channels, not physical locations of a particular point.

Indeed, what we call the *grid* here (in this article) actually corresponds to what is referred to as the *spectral* space by practitioners. The actual images were referred to as the *geometric* or *spatial* space. This image remote sensing problem was very familiar to mathematicians and operations research practitioners, and was typically referred to as signal processing. And in many ways, this was a precursor to modern AI.

6. New *K*-NN Clustering Algorithm and Data Reduction

I describe here an interesting and intuitive clustering algorithm (that can be used for data reduction as well) offering several advantages, over traditional classifiers:

- More robust against outliers and erroneous data
- Executing much faster

- Generalizing well known algorithms

You don't need to know *K*-NN to understand this chapter -- but click here if you want to learn more about it. You don't need a background in statistical science either.
Let's describe this new algorithm and its various components, in simple English

6.1. General Framework

We are dealing here with a supervised learning problem, and more specifically, clustering (also called supervised classification.). In particular, we want to assign a class label to a new observation that does not belong to the training set. Instead of checking out individual points (the nearest neighbors) and using a majority (voting) rule to assign the new observation to a cluster based on nearest neighbor counts, we are checking out *cliques* of points, and focus on the nearest cliques rather than on the nearest points.

6.2. Cliques and Clique Density

The cliques considered here are defined by circles (in two dimensions) or spheres (in three dimensions.) In the most basic version, we have one clique for each cluster, and the clique is defined as the smallest circle containing a pre-specified proportion *p* of the points from the cluster in question. If the clusters are well separated, we can even use *p* = 1. We define the density of a clique as the number of points per unit area. In general, we want to build cliques with high density.

Ideally, we want each cluster in the training set to be covered by a small number of (possibly slightly overlapping) cliques, each one having a high density. Also, as a general rule, a training set point can only belong to one clique, and (ideally) to only one cluster. But the circles associated with two cliques are allowed to overlap.

6.3. Classification Rule, Computational Complexity, Memory Requirements

Once we have built a set of cliques for each cluster, the classification rule is straightforward. Building these cliques is the complicated pre-processing step, but as discussed in the last section, we only need a rough approximation. The classification rule is as follows:

- **1-NC** algorithm: Assign the new observation to the cluster attached to the nearest clique
- **K-NC** algorithm: Assign the new observation to the cluster that has the largest number of cliques, among the *K* cliques closest to the observation in question (majority vote).

Note that if cliques consist of a single point, then *K*-NN and *K*-NC algorithms are identical. Also note that computing the distance between a point and a clique is straightforward, because cliques are circular. You just need to know the center of the circle in question, and its radius.

Finally, to assign a new observation to a cluster, one only has to check all the cliques, rather than all the points. Thus the *K*-NC algorithm is *v* times faster than *K*-NN, where *v* is number of points in the training set, divided by the number of cliques across all clusters. In practice, we have far fewer cliques than we have points in the training set, so *v* can be large, when dealing with very big training sets. This is especially true if there is not too much overlap among the cliques.

In short, the cliques summarize the training set data: we can discard all the data and only keep the cliques (with their center, radius, density, and cluster label), once these cliques have been computed. This also saves a lot of memory, and in itself can be used as a data reduction technique.

6.4. Cliques Building and Smallest-Circle-Problem

This concept could prove useful when building the clique system. The smallest-circle-problem (click here for details) consists of computing the smallest circle that contains all points in a given region. It is illustrated in Figure 1 below.

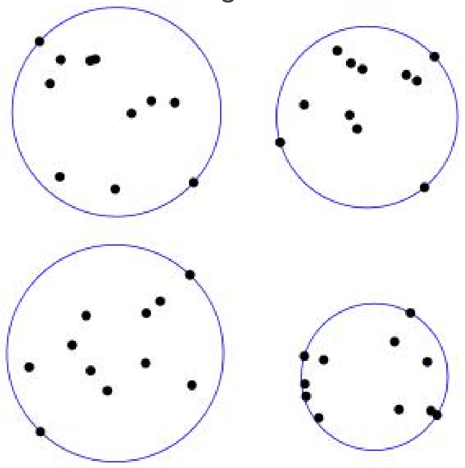


Figure 1: *Cliques computed based on the Smallest-Circle-Problem*

One would think that such cliques have maximum density, a desirable property. Several very efficient algorithms are available to solve this problem. Some even allow you to attach a weight to each point.

6.5. Gravitational Field Generated by Clusters

You can skip this section. It has been included for those interested in further improvements of the *K*-NC algorithm, as well as improving standard algorithms such as *K*-NN, Also, it highlights the fact that the square of the distance, could be a better metric than raw distance (from a modeling point of view) when averaging proximity among points, just like many physical laws involve the square of the distance between two objects, rather than the distance itself. You can look at a classification problem the same way that you would at the gravitation law: which cluster is going to "attract" a new observation (in the context of classification), versus which celestial body is going to attract and adsorb a meteoroid (in the context of celestial physics.) You would think that in both cases, similar laws apply.

For each point *x* (typically, a new point that we want to classify) and cluster *G*, the potential *V*(*x*, *G*) is defined as follows:

$$V(x,G) = \sum_{C \in G} \frac{1}{\mathrm{d}^2(x,C)}$$

where the sum is over all cliques in *G*. A better definition might be to take the sum only over the *k* nearest cliques in *G,* for a pre-specified value of *k*. A potential classification rule consists of assigning a point *x* to the group *G* that maximizes *V*(*x*, *G*).

A final improvement consists in attaching a weight to each term in the above sum: the weight being the density of the corresponding clique. Note that if cliques (their circles) significantly overlap, this should be addressed in the definition of the potential *V*. As a general rule, a training set point can only belong to one clique, and (ideally) to only one cluster.

6.6. Building an Efficient Clique System

This data pre-processing step is the more complicated step. However, easy-to-obtain approximate solutions work well. It provides good results even if each point is a clique (the *K*-NN particular case.

Different approaches are possible:

- Start with one clique per training set point, and iteratively merge the cliques
- Start with one clique per cluster, based on the smallest-circle-problem described earlier. Then shrink it and move the center till it contains a proportion *p* of the training set points, for the cluster in question, and the density of the clique is maximal (or close enough to maximum.) Repeat the operation for the remaining points in the cluster in question. Maybe choose *p* = 30% assuming each cluster has a circular core consisting of at least 30% of its points.
- Start with a pre-specified number of random cliques (random radius, random center, possibly corresponding to a point randomly selected in the training set.) in each cluster. Adjust the centers and radii one clique at a time to optimize some density-related criterion described below. Also, it would be a good idea to use a birth and death process to eliminate some cliques and create new ones over time.

The aim is to obtain as few cliques as possible, covering the entire training set without too much overlapping. Each clique must have a high density, and must contain more than (say) 1% of the training set points for the cluster in question. We could add another constraint: each clique must have at least two points.

6.7. Non-Circular Cliques

It would be interesting to test whether the shape of the clique matters. As long as we have sufficiently many cliques, the shape probably does not matter (this should be tested), and thus a circle -- because it leads to considerably simplified computations when measuring the distance between a clique and a point -- is ideal. Note that if a point is inside a clique (inside its circle) then the distance between the clique and the point is zero. We should test whether this rule leads to more robustness against outliers or erroneous data.

6.8. Potential Enhancements, Data Reduction, and Conclusions

Potential improvements to our methodology include:

- Using density in addition to proximity, putting more emphasis on dense cliques when assigning a new observation to a cluster
- Allowing training set points to belong to multiple cliques within the same cluster
- Allowing training set points to belong to multiple clusters
- Testing cliques that are made up of closest neighbors
- Reducing the overlap among cliques

Also, we need to test this new algorithm, and see when it performs best, depending on how cliques are created. Is *K*-NC almost equivalent to *K*-NN, provided a different *K* is used in each case? Even if they are equivalent, *K*-NC has one great benefit: it can be used for data reduction, in a way that is different from traditional data reduction techniques. Traditional data reduction techniques such as PCA try to project the data set onto a space that has a lower dimension. The clique structure produced by *K*-NC is the result of a data reduction technique that does not perform dimension reduction nor projections, but instead, performs something akin to data compaction or data compression or entropy reduction, in the same original space.

Another benefit is the fact that *K*-NC, thanks to its pre-processing step to build the clique structure, runs much faster than *K*-NN. It is also more intuitive, as it is based on the intuitive concept that each cluster is made up of sub-clusters: the cliques. It is indeed similar to model fitting with stochastic point processes representing cluster structures, such as Neyman-Scott cluster processes.

Finally, in many cases, one way to improve a classifier is by re-scaling or transforming the data, for instance using a logarithmic scale. Most classifiers are scale-dependent. See section 6 in the first chapter in Part 5 of this book, for scale-invariant clustering.

7. Spatial Patterns Found in Random Points

Check the three charts below: only one corresponds to pure randomness. Which one?

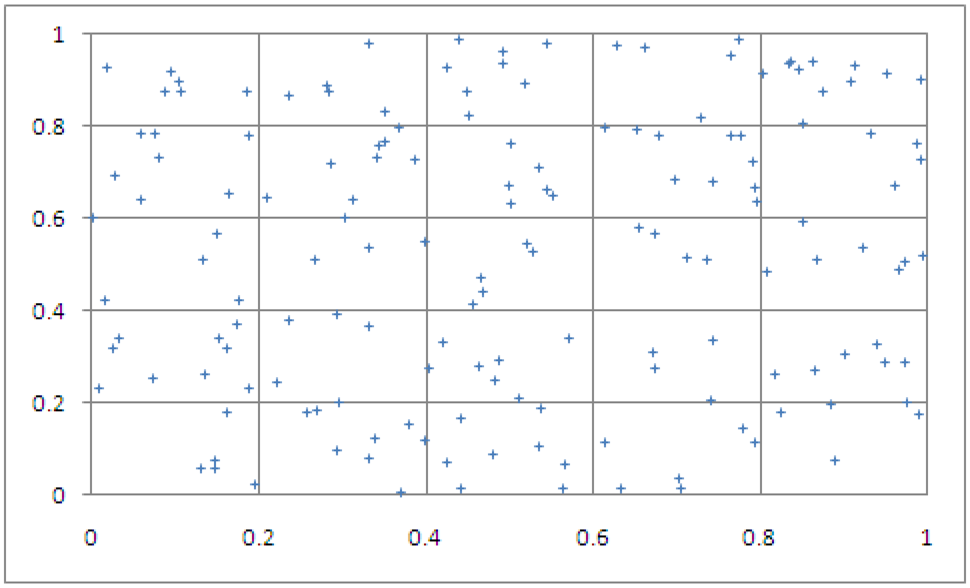


Chart #1

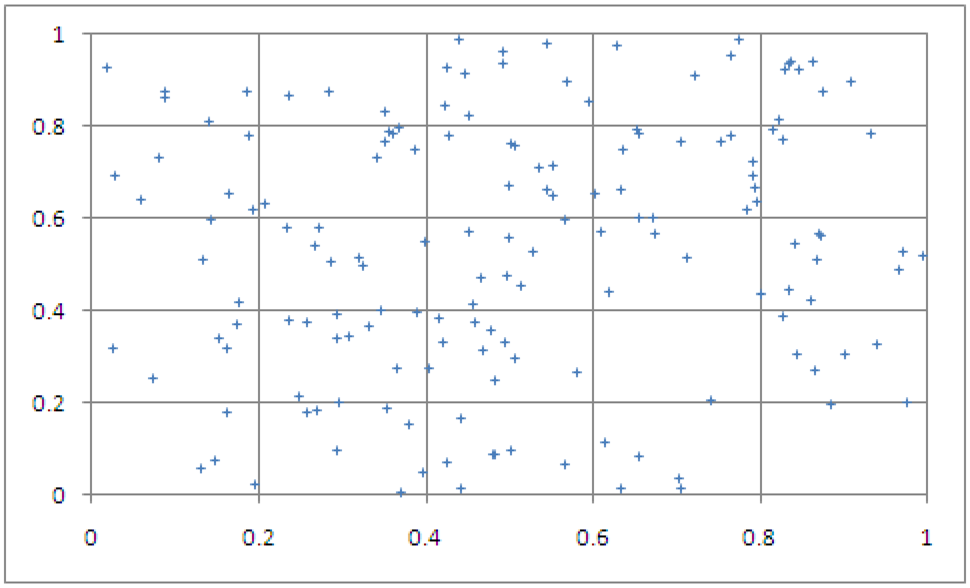


Chart #2

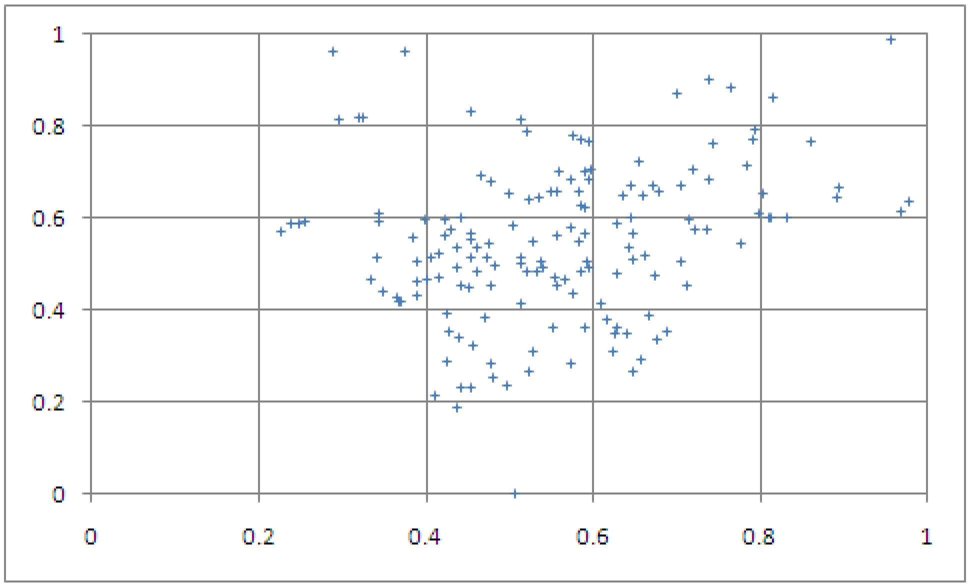


Chart #3

It is clear that chart #3 exhibits a strong clustering pattern, unless you define your problem as points randomly distributed in an unknown domain whose boundary has to be estimated. So, the big question is: between chart #1 and #2, which one represents randomness? Look at these charts very closely for 60 seconds, then make a guess, then read on. Note that all three charts contain the same number of points - so there's no scaling issue involved here.

Let's assume that we are dealing with a spatial distribution of points over the entire 2-dimentional space, and that observations are seen through a small square window. For instance, points (observations) could be stars as seen on a picture taken from a telescope.

The **first issue** is the fact that the data is censored: if you look at the distribution of nearest neighbor distances to draw conclusions, you must take into account the fact that points near the boundary have fewer neighbors because some neighbors are outside the boundary. You can eliminate the bias by

- Tiling the observation window to produce a mathematical tessellation
- Mapping the square observation window onto the surface of a torus
- Apply statistical bias-correction techniques
- Use Monte-Carlo simulations to estimate what the true distribution is (with confidence intervals) if the data was truly random

Second issue: you need to use better visualization tools to see the patterns. The fact that I use a + rather than a dot symbol to represents the points, helps: some points are so close to each other that if you represent points with dots, you won't visually see the double points (in our example, double points could correspond to double star systems - and these very small-scale point interactions are part of what makes the distribution non-random in two of our charts). But you can do much better: you could measure a number of metric (averages, standard deviations, correlation between x and y, number of points in each sub-square, density estimates, etc.) and identify metrics proving that we are not dealing with pure randomness.

In these 3 charts, the standard deviation for either *x* or *y* - in case of pure randomness - should be 0.290 plus or minus 0.005. Only one of the 3 charts succeeds with this randomness test.

Third issue: even if multiple statistical tests suggest that the data is truly random, it does not mean it really is. For instance, all three charts show zero correlation between *x* and *y*, and have mean *x* and *y* close to 0.50 (a requirement to qualify as random distribution in this case). However, only one chart exhibits randomness.

Fourth issue: we need a mathematical framework to define and check randomness. True randomness is the realization of a Poisson stochastic process, and we need to use metrics that uniquely characterizes a Poisson process to check whether a point distribution is truly random or not. Such metrics could be e.g.

- The inter-point distance distributions
- Number of observations in sub-squares (these counts should be uniformly distributed over the sub-squares, and a Chi-square test could provide the answer; however in our charts, we don't have enough points in each sub-square to provide a valid test result)

Fifth issue: some of the great metrics (distances to k^{th} neighbors) might not have a simple mathematical formula. But we can use Monte-Carlo simulations to address this issue: simulate a random process, compute the distribution of distances (with confidence intervals) based on thousands of simulations, and compare with distances computed on your data. If the distance distribution computed on the data set matches results from simulations, it means our data is probably random. However, we would have to make sure that the distance distribution uniquely characterizes a Poisson process, and that no *non-random* processes could yield the same distance distribution. This exercise is known as goodness-of-fit testing: you try to see if your data support a specific hypothesis of randomness.

Sixth issue: if you have a million points (and in high dimensions, you need much more than a million points due to the curse of dimension), then you have a trillion distances to compute. No computer, not even in the cloud, will be able to make all these computations in less than a thousand year. So you need to pick up 10,000 points randomly, compute distances, and compare with equivalent computations based on simulated data. You need to make 1,000 simulations to get confidence intervals, but this is feasible.

Here's how the data (charts 1-3) was created:

- Produce 158 random points [a_n, b_n], n=1, ...,158

- Produce 158 random deviates (u_n, v_n), n=1, ...,158
- Define x_n as follows for $n > 1$: if $u_n < r$, then $x_n = a_n$, else $x_n = sv_na_n + (1 - sv_n)x_{n-1}$, with $x_1 = a_1$
- Define y_n as follows for $n > 1$: if $u_n < r$, then $y_n = b_n$, else $y_n = sv_nb_n + (1 - sv_n)y_{n-1}$, with $y_1 = b_1$
- Chart 1: $x_n = a_n$, $y_n = b_n$
- Chart 2: $r = 0.5$, $s = 0.5$
- Chart 3: $r = 0.4$, $s = 0.1$

Notes:

- The only chart exhibiting randomness is chart #1. Chart #2 has significantly too low standard deviations for x and y, too few points near boundaries, and too many points that are very close to each other
- Note that chart #1 (the random distribution) exhibits a little bit of clustering, as well as some point alignments: this is however perfectly expected from a random distribution. If the number of points in each sub-square was identical, the distribution would not be random, but would correspond to a situation where antagonist forces make points to stay as far away as possible from each other.
- How would you test randomness if you had only two points (impossible to test), three points, or just 10 points?
- Finally, once a pattern is detected (e.g. abnormal close proximity between neighboring points), it should be interpreted and/or leveraged, that is, it should lead to (say) ROI-positive trading rules if the framework is about stock trading, or the conclusion that double stars do exist (based on chart #2, more on this here) if the framework is astronomy

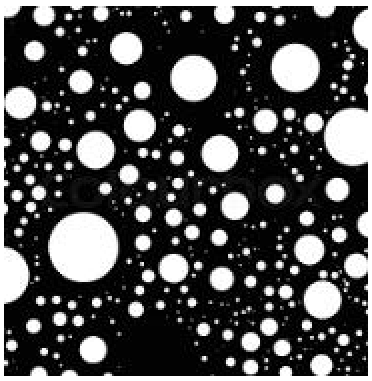
8. Stochastic Geometry: Spatial Coverage Problem

This should remind you of your probability classes during college years. Can you solve this problem in 30 minutes? This could make for an interesting job interview question.

Problem

Points are randomly distributed on the plane, with an average of m points per unit area. A circle of radius R is drawn around each point. What is the proportion of the plane covered by these (possibly overlapping) circles?

What if circles can have two different radii, either $R = r$, or $R = s$, with same probability? What if R is a random variable, so that we are dealing with random circles? Before reading further, try to solve the problem yourself.



Solution

The points are distributed according to a Poisson point process of intensity *m*. The chance that an arbitrary point *x* in the plane is not covered by any circle, is the chance that there is zero point from the process, in a circle of radius *R* centered at *x*. This is equal to exp($-m\pi R^2$). Thus the proportion of the plane covered by the circles is

$$p(m,R) = 1 - \exp(-m\pi R^2)$$

If circles have radii equal to *r* or *s*, it is like having two independent (superimposed) Poisson processes, each with intensity *m*/2: one for each type of circle. The chance that *x* is not covered by any circle is thus a product of two probabilities,

$$1 - p(m,R) = \exp\Big(-\frac{m\pi r^2}{2}\Big) \times \exp\Big(-\frac{m\pi s^2}{2}\Big) = \exp\Big(-m\pi \cdot \frac{r^2+s^2}{2}\Big)$$

If *R* is a positive random variable and *E* denotes its expectation, then the general solution is

$$p(m,R) = 1 - \exp\Big(-m\pi E[R^2]\Big)$$

You can easily simulate a large number of these circles over a broad area, and then, pick up 1,000 random points and see how many of them are covered by at least one circle, to check whether your solution is correct or not. You can also use these simulations to get an approximation for exp($-\pi$).

Application

The formula for *p*(*m*, *R*) and confidence intervals obtained for its value via simulations under the assumption of pure randomness (Poisson process), can be used to check if a process is really random. For instance, are Moon's craters spread randomly? They might not, if large meteorites break up in small pieces before impact, resulting in clustered craters. In that case, the area covered by the (overlapping) craters might be smaller than theoretically expected.

9. Markov Chains and the Collatz Conjecture

Take any positive integer n. If n is even, divide it by 2 to get $n/2$. If n is odd, multiply it by 3 and add 1 to obtain $3n + 1$. Repeat the process indefinitely. Does the sequence eventually reach 1, regardless of the initial value? For instance, if you start with the number 75,128,138,247, you eventually reach 1 after 1,228 steps. If you start with the number 27, you climb as high as 9,232, but eventually reach 1 after 41 steps.

This is supposed to be a very difficult problem. Note that if a sequence reaches any power of 2 (say, 64) or any intermediate number found in the trillions of trillions of such sequences that are known to reach 1, then the sequence in question will obviously reach 1 too. For a sequence not to reach 1, the first element (as well as any subsequent element) would have to be different from any initial or intermediate number found in any series identified as reaching 1 so far. This makes it highly unlikely, yet the conjecture has not been proved yet. For more on this problem, click here.

It is interesting to note that if you replace the deterministic algorithm by a probabilistic one, for instance n becomes $n/2$ with probability 0.5 and $3n + 1$ with probability 0.5, then instead of reaching 1, you reach infinity. Also with the deterministic algorithm, if you replace $3n + 1$ by $2n + 1$, you would think that you would reach 1 even faster, but this is not the case: you reach 1 only if the initial value is a power of 2, and in all cases you eventually reach infinity.

Possible proof

If you want to prove (or disprove) this conjecture, a possible methodology is as follows. Let's recursively define $f(k+1, n) = f(f(k, n))$ for $k = 0, 1, 2$ and so on, with $f(0, n) = n$, and $f(x) = x/2$ if x is even, $3x + 1$ otherwise. The conjecture states that no matter the initial value n, there is always a number k (function of n) such that $f(k, n) = 1$: in short, you reach 1 after k steps.

Consider the following four cases, each occurring with a probability 0.25 (Mod stands for the modulo operator):

- $\text{Mod}(n, 4) = 0$. Then $f(2, n) = n/4$.
- $\text{Mod}(n, 4) = 1$. Then $f(3, n) = (3n + 1)/4$
- $\text{Mod}(n, 4) = 2$. Then $f(1, n) = n/2$.
- $\text{Mod}(n, 4) = 3$. This case is broken down into two sub-cases, see below.

The case $\text{Mod}(n, 4) = 3$ is broken down into the following two sub-cases, each occurring with probability 0.125:

- If $\text{Mod}(n, 8) = 3$ then $f(2, n) = (3n + 1)/2$ and in this case we are back to case #2 above after 2 steps.
- If $\text{Mod}(n, 8) = 7$ then $f(2, n) = (3n + 1)/2$ and in this case we are back to case #4 above after 2 steps.

In both sub-cases, the sequence has been increasing, though we know that if Mod(*n*, 8) = 3, it will go down a bit (but still stay a little above *n*, more specifically around 9*n*/8) after 3 additional steps.

So it looks like on average, we are decreasing over time (thus the fact that we would eventually reach 1 seems likely), but the challenging case if when Mod(*n*, 4) = 3, and even more challenging when Mod(*n*, 8) = 7. Can we get stuck in a sequence where every two steps, the residual modulo 8 is equal to 7 (the worst case that makes the sequence grows at its fastest pace?) And for how many cycles can we get stuck in such a configuration? These are difficult issues to address if you want to prove this conjecture.

The problem might also be approximately modeled as some kind of **Markov chain**, with 5 different states corresponding to the first 3 cases and the 2 sub-cases discussed earlier. One single iteration in the Markov chain corresponds respectively to 2, 3, 1, 5, and 2 steps of the above algorithm, to reach the next local dip in value (if any).

For *n* large enough, one iteration of the Markov chain is thus approximately as follows:

- we reduce *n* by 75% with probability 0.25
- we reduce *n* by 25% with probability 0.25
- we reduce *n* by 50% with probability 0.25
- we increase *n* by 12.5% with probability 0.125
- we increase *n* by 50% with probability 0.125

It is easy to compute the probability *p*(*N)* that the initial value *n* will not be reduced after *N* iterations of the Markov chain, for any positive *N*. However, even for very large *N*, this probability is still strictly positive, albeit very close to zero. Also, it is not clear if the memory-less property of Markov chains is violated here, which would either invalidate this approach, or possibly make it more difficult to handle this problem. Most likely, if it results in a proof, it would be an heuristic one.

10. Special Integral Solved Using Statistical Concepts

Below are a few integrals that you won't find in textbooks. Solving them is a good exercise for college students with some advanced calculus training. We provide the solution, as well as a general framework to compute many similar integrals. Maybe this material should be part of the standard math curriculum. Here, *p*, *q*, *r* are positive real numbers, with *q* larger than *p*.

$$\int_0^\infty \frac{|\sin qx| - |\sin px|}{x} dx = \frac{2}{\pi} \log \frac{q}{p} = \int_0^\infty \frac{|\sin \sqrt{qx}| - |\sin \sqrt{px}|}{x} dx$$

$$\int_0^\infty \frac{|\sin qx|^r - |\sin px|^r}{x} dx = \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{1+r}{2})}{\Gamma(1+\frac{r}{2})} \log \frac{q}{p}$$

The Gamma symbol represents the gamma function. It is possible that these results are published here for the first time. These are known as Frullani integrals, although the ones mentioned here are not covered by Frullani's theorem, nor by any recent generalization that I am aware of (see here and here for recent contributions to this topic.) Indeed, AI-based automated integration platforms such as WolframAlpha cannot find the exact value (only an approximation) while they are able to compute standard Frullani integrals exactly. My approach to derive the exact values is different from the classical approaches, as it relies on the statistical concept of expectation, possibly leading to interesting areas of research.

How to compute such integrals?

These integrals are a particular case of the following main result, proved in the next section:

$$\int_0^\infty \frac{qf(qx)}{g(qx)} - \frac{pf(px)}{g(px)}dx = \int_0^\infty \frac{qf(qx)g(px) - pf(px)g(qx)}{g(qx)g(px)}dx = E(f)\cdot\log\frac{q}{p}$$

where *g*(*x*)/*x* tends to 1 as *x* tends to infinity, and *f* is a bounded function with a finite expectation. Some additional conditions may be required, for instance the fact that the integral of *f*(x)/*g*(x), between 0 and 1, is finite. The expectation of *f*, also called average value, is defined as

$$E(f) = \lim_{T\to\infty}\frac{1}{T}\cdot\int_0^T f(x)dx.$$

For instance, if *f*(*x*) = |SIN($x^{1/2}$)|, then the expectation exists, and it is equal to *E*(*f*) = 2/π.

The main result introduced at the beginning of this section, is rather intuitive but needs great care to prove it rigorously, including correctly stating the required assumptions on *f* and *g* to make it valid. Some cases might require working with non-Riemann integrals. Here we only provide the intuitive explanation.

Proof of the main result (sketch)

Here *p*, *q* and *n* are integers, with *q* greater than *p*. We are interested in the case where *n* tends to infinity. We approximate integrals using the Euler-Maclaurin summation formula. The approximations below become equalities as *n* tends to infinity.

$$\sum_{k=pn+1}^{qn}\frac{f(k)}{g(k)} \approx E(f)\cdot\sum_{k=pn+1}^{qn}\frac{1}{g(k)} \approx E(f)\cdot\sum_{k=pn+1}^{qn}\frac{1}{k} = E(f)\cdot\Big(\sum_{k=1}^{qn}\frac{1}{k} - \sum_{k=1}^{pn}\frac{1}{k}\Big)$$
$$\approx E(f)\cdot\Big(\log qn - \log pn\Big) = E(f).\log\frac{q}{p}.$$

The first approximation is related to the Abelian theorem. We also used the classic approximation of the harmonic series to make the logarithm terms appear. Note that for

large values of *k*, *g*(*k*) is asymptotically equal to *k*. This was one of the requirements for the formula to be valid.
We also have:

$$\sum_{k=pn+1}^{qn} \frac{f(k)}{g(k)} = \sum_{k=1}^{qn} \frac{f(k)}{g(k)} - \sum_{k=1}^{pn} \frac{f(k)}{g(k)}$$

$$\sum_{k=1}^{qn} \frac{f(k)}{g(k)} \approx \int_0^{qn} \frac{f(x)}{g(x)} dx, \quad \sum_{k=1}^{pn} \frac{f(k)}{g(k)} \approx \int_0^{pn} \frac{f(x)}{g(x)} dx.$$

Using the change of variable $y = x / q$ in the first integral, and $y = x / p$ in the second integral, we obtain:

$$\sum_{k=pn+1}^{qn} \frac{f(k)}{g(k)} \approx \int_0^n q\frac{f(qy)}{g(qy)} dy - \int_0^n p\frac{f(py)}{g(py)} dy \to \int_0^\infty \frac{qf(qy)}{g(qy)} - \frac{pf(py)}{g(py)} dy.$$

This concludes the proof.

Note that for all the examples in the introduction, I used $g(x) = x$. I also tested it with different functions such as $g(x) = x + 1$, and it also worked. The results are not posted here.

Generalization

So far, we assumed that $g(x)/x$ tends to 1 as *x* tends to infinity. What if instead, we make the more general assumption that $g(x) / x$, is equal to 1 on average? Using the notation *E*(*f*) or *E*(*f*(*x*)) interchangeably to denote the expectation of a function *f*, the main results becomes:

$$\int_0^\infty \frac{qf(qx)}{g(qx)} - \frac{pf(px)}{g(px)} dx = E(f(x)) \cdot E\left(\frac{x}{g(x)}\right) \cdot \log \frac{q}{p}.$$

This formula works even if *f* or *g* is not Riemann-integrable, as long as the expectations are finite and different from zero. In this case, the integrals can be replaced by infinite sums or averages over equally-spaced points.

An example of function *g* not covered by the main theorem, but covered by its generalization, is

$$g(x) = \sqrt{1 + x^2} \cdot (1 + \sin^2 x).$$

In this case, $E(x/g(x)) = 1/2^{1/2}$.

11. From A/B Testing to Discrete Choice Analysis

Let's say you want to test the optimum price for some items sold online. One way to do it is to set two different prices and do some A/B testing to see which price generates the most revenue, or comparing user-customized versus flat prices, using Thompson sampling, the Taguchi method or multi-armed A/B testing.

How to proceed if you want to test a continuous set of prices, not just two or three prices A/B/C? Is testing (say) 10,000 different prices any better than standard A/B testing, or does it lead to over-fitting and thus a non-robust solution? Likewise, if you want to test which background color works best for a website, is testing one million different colors more efficient than standard testing, and how to do it?

Also, let's say you want to modify 20 features on your website, each one having 4 potential values (color, font size, font face and so on). In short, instead of A/B testing with 2 potential outcomes (A or B), you perform a multivariate test with 4^{20} outcomes. Of course you will be able to test only a tiny fraction of all the possibilities, but is it more efficient than sequentially doing an A/B test for one feature, then another A/B test for another feature, and so on? The latter approach would take a lot of time and would result in a very local optimum. For instance, for the first feature, maybe A works best, for the second one (after choosing A for the first one) C works best, but for both featured combined, maybe (D, B) works best. How to do such a test when the number of potential combinations is 4^{20}?

Finally, how do you determine the sample size for these types of experiments? Or in other words, what is the stopping criterion?

The technique used by leading market research professionals to solve the problems described here is called **conjoint** or **discrete choice analysis**. Here is a brief example of the type of problem conjoint analysis helps solve. Conjoint analysis helps you develop the preferred product or service based on almost an unlimited number of attributes (price, color, font size, etc.) and levels (different prices, different colors, different font sizes, etc.). The ultimate output of the conjoint model is a simulator that lets you test for the best product (based on market share, sales, other objectives) and can provide you a price elasticity of demand curve for a price. Sawtooth Software uses Hierarchical Bayesian Regression to drive their models.

12. Deep Dive into Polynomial Regression and Overfitting

Here, we show that the issue with polynomial regression is not over-fitting, but numerical precision. Even if done right, numerical precision still remains an insurmountable challenge. We focus here on step-wise polynomial regression, which is supposed to be more stable than the traditional model. In step-wise regression, we estimate one coefficient at a time, using the classic least square technique.

Even if the function to be estimated is very smooth, due to machine precision, only the first three or four coefficients can be accurately computed. With infinite precision, all coefficients would be correctly computed without over-fitting. We first explore this problem from a mathematical point of view in the next section, then provide recommendations for practical model implementations in the last section.

This is also a good read for professionals with a math background interested in learning more about data science, as we start with some simple math, then discuss how it relates to data science. Also, this is an original article, not something you will learn in college classes or data camps, and it even features the solution to a regression problem involving an infinite number of variables.

12.1. Polynomial regression for Taylor series

Here we show how the mathematical machinery behind polynomial regression creates big challenges, even in a perfect environment where the response is well behaved and the exact theoretical model is known. In the next section, we shall show how these findings apply in the context of statistical polynomial regression, to design a better modeling tool for data scientists and statisticians.

Let us consider a function *f*(*x*) that can be represented by a Taylor series:

$$f(x)=\sum_{k=0}^{\infty} b_k x^k.$$

Here we assume that the coefficients are bounded, though the theory also works with a less restrictive assumption, provided that the coefficients do not increase too fast. In most cases, for instance the exponential function, the successive coefficients actually get closer and closer to 0, guaranteeing convergence at least when |*x*| < 1. The function *f*(*x*) does not need to be differentiable; it could even be differentiable nowhere, such as for the Weierstrass function. So the context here is more general than the standard Taylor series framework.

Stepwise polynomial regression: algorithm

We introduce here an iterative algorithm to estimate the coefficients b_k one at a time, in the above Taylor series. Note that we are dealing with a regression problem with an infinite number of variables. It is still solved using classic least square approximations. We focus on values of *x* that are located in a small symmetrical interval centered at 0. This interval is denoted as *S*. The estimated coefficients are denoted as a_k. We introduce the following notations:

$$f_n(x)=\sum_{k=0}^{n} a_k x^k, \text{ for } n\geq 0, \text{ with } f_{-1}(x)=0,$$

$$E_n=\int_S \Big(a_n x^n + f_{n-1}(x) - f(x)\Big)^2 dx, \text{ for } n\geq 0.$$

Here, *E*(*n*) is called the mean squared error after estimating *n* coefficients. It measures how well we are approaching the target function *f*(*x*) after *n* steps. The coefficient a_n (the estimated value of b_n) is chosen to minimize the mean squared error *E*(*n*) in the above formula. Note that the mean squared error is a decreasing function of *n*. We proceed iteratively starting with *n* = 0. As in the standard least square framework, take the derivative of *E*(*n*) with respect to a_n, and find its root, to determine a_n. The result is

$$a_n=\frac{\int_S x^n(f(x)-f_{n-1}(x))dx}{\int_S x^{2n}dx}.$$

Convergence theorem

We have the following remarkable result:

As the interval S, centered at 0, gets smaller and smaller and tends to S = {0}, the estimated coefficients a_k tend to the true value b_k.

For instance, if $f(x) = \exp(x)$, that is, if $b_k = 1/k!$, then the estimates a_k also tend to $1/k!$. Thus this framework provides an alternate way to compute the coefficients of a Taylor series, even when derivatives of $f(x)$ do not exist. It also means that step-wise regression, in this context, works just as well as a full-fledged regression, yet involves far fewer computations. A full-fledged regression would involve inverting an infinite matrix.

The proof of this theorem is quite simple, and proceeds by induction. First, check that $a_0 = b_0$. Then, if $a_k = b_k$ for all k less than or equal to n, we must also have $a_{n+1} = b_{n+1}$. In order to prove this, note that under this assumption, we have:

$$a_{n+1} = \sum_{k=0}^{\infty} b_{n+1+k} \cdot \frac{\int_S x^{2n+2+k} dx}{\int_S x^{2n+2} dx}.$$

As S tends to {0}, all terms except the first one (corresponding to $k = 0$) in the above series, vanish. Thus $a_{n+1} = b_{n+1}$, at the limit. Thus, all estimated coefficients match the true ones.

12.2. Application to Real Life Regression Models

The convergence theorem in the previous section seems to solve everything, even dealing with an infinite number of variables in a regression problem, and yet delivering a smooth, robust theoretical solution to a problem notoriously known for its over-fitting issues. It has to be too good to be true. While the theoretical result is correct, we explore in this section how it translates to applications such as fitting actual data with a polynomial model. It is not pretty, but also not as bad as you would think.

In real life applications, S is your set observations (the independent variable) rather than an interval, after re-scaling these observations so that they are centered at 0, and all very close to 0. You then replace the integral by a sum over all your re-scaled data points. Everything else stays the same.

I tested this using the same perfectly exponential response $f(x) = \exp(x)$, using m = 1,000,000 random deviates distributed on [$-1/m$, $-1/m$] to simulate a real data set. I then computed the estimates a_k. By design, according to the convergence theorem, they should all be close to $a_k = 1/k!$. This did not happen. Indeed it only worked for $k = 0$ and $k = 1$. By tweaking the parameters (the set S and m) I was able to get up to four correct coefficients. This is the best that I could get, due to machine precision. Now the good news: Even though higher order coefficients were all very wrong, the impact on interpolation / predictive power was minimal for four reasons:

- *S* was extremely concentrated around 0,
- As a result, higher terms (k = 3, 4, etc.) had almost no impact on the response,
- As a result, getting higher order estimates a_k, even if totally wrong, had no impact on the error $E(n)$ discussed in the first section,
- It was obvious that beyond k = 3 or k = 4, the error, very small, so small that it was smaller than machine precision, stabilized and did not go up, as expected.

The most surprising result is as follows, and I noticed this behavior in all my tests:

- Typical values: a_0 = 0.999999995416665, a_1 = 0.999999965249988, a_2 = 3.55555598019368. The correct ones are respectively 1, 1, and 1/2.
- Replace a($_0$) = 0.999999995416665 by a($_0$) = 1 in the regression model, now you get a_1 = 1, and a_2 =   0.50000086709697, very close to the theoretical value 1/2. How can such a small difference have such a big impact?

The answer is because machine precision in standard arithmetic is typically about 15 decimals, and the error $E(n)$ quickly drops to $1/10^{25}$, largely because the independent observations are highly concentrated so close to 0. One way to get around this is to use high precision computing. It allows you to work with thousands of correct decimals (or billions if you wanted to, but expect it to be very slow), rather than just 15.

Recommendations for practical model implementation

Here are some takeaways from my experiments:

- Polynomial regression, in general, should be avoided. If you want to do it, use step-wise polynomial regression as described in this article: it is more stable, and it leads to easier interpretation.
- Rescale your independent variable so that data points for this variable fit in [-1, 1], maybe even in [-0.01, 0.01], to get more robust results. Use for interpolation, not extrapolation.
- Avoid making predictions outside any for x outside [-1, 1].
- Start with a standard linear regression to get your first two estimated coefficients a_0 and a_1 as accurate as possible. Then do a stepwise polynomial regression to get the third and fourth coefficients in your polynomial model, with the first two coefficient estimates set to the value obtained in the linear regression.
- When the error $E(n)$ does not improve when adding new coefficients and increasing n, stop adding them. This should be your stopping point in your iterative step-wise polynomial regression algorithm, usually occurring at n = 3 or 4 unless you use the techniques (high precision computing and so on) described in this article.

Also, it is easy to compute confidence intervals for the coefficients in the step-wise linear regression, using Monte-Carlo simulations. Add random noise to your data, do it a thousand times with a different noise, and see how the estimated a_k's fluctuate based on the added noise: this will help you build confidence bands for your estimates.

The author of the article *Step-wise Polynomial Regression: Royal Road or Detour?* (here) comes to the same conclusion: it is an *ill-conditioned* n problem. The solution is to replace x^k in the Taylor series by a more general but smoother term $g_k(x)$ called a *spline*. The convergence theorem can easily be generalized.

13. Lifecycle of Data Science Projects

Data science projects can be broken down into 7 main phases. It is sometimes necessary to move back to a previous stage to fix previous business mistakes, and start over from there. Also, respect the 80/20 rule. Here are the 7 phases:

Identify the problem

- Identify baseline (doing nothing) and metrics used to measure success over baseline
- Identify type of problem: prototyping, proof of concept, root cause analysis, predictive analytics, prescriptive analytics, machine-to-machine implementation
- Identify key people within your organization and outside
- Get specifications, requirements, priorities, budgets
- How accurate the solution needs to be?
- Do we need all the data?
- Built internally versus using a vendor solution
- Vendor comparison, benchmarking

Identify available data sources

- Extract (or obtain) and check sample data (use sound sampling techniques); discuss fields to make sure data is understood by all stakeholders
- Perform EDA (exploratory analysis, data dictionary)
- Assess quality of data, and value available in data
- Identify data glitches, find workarounds
- Is data quality and fields populated consistent over time?
- Are some fields a blend of different stuff (example: keyword field, sometimes equal to user query, sometimes to advertiser keyword, with no way to know except via statistical analyses or by talking to business people)
- How to improve data quality moving forward?
- Do I need to create mini summary tables / database to store external data or perform local analyses more efficiently?
- Which tool do I need (R, Excel, Tableau, Python, Tableau, SAS and so on)

Identify if additional data sources are needed

- What fields should be captured?
- How granular?
- How much historical data do we need?

- Do we need real time data?
- How to store or access the data (NoSQL? Cloud?)
- Do we need experimental design?

Statistical Analyses

- Use imputation methods as needed
- Detect / remove / interpret outliers
- Selecting variables (also, variables reduction)
- Is the data censored (hidden data, as in survival analysis or time-to-crime statistics)
- Cross-correlation analysis
- Model selection (as needed, favor simple models)
- Sensitivity analysis
- Cross-validation, model fitting
- Measure accuracy, provide confidence intervals

Implementation, development

- FSSRR: Fast, simple, scalable, robust, re-usable
- How frequently do I need to update lookup tables, white lists, data uploads, and so on
- Debugging
- Need to create an API to communicate with other apps?

Communicate results

- Need to integrate results in dashboard? Need to create an email alert system?
- Decide on dashboard architecture, with business people
- Visualization
- Discuss potential improvements (with cost estimates)
- Provide training
- Commenting code, writing a technical report, explaining how your solution should be used, parameters fine-tuned, and results interpreted

Maintenance

- Test the model or implementation; stress tests
- Regular updates
- Final outsourcing to engineering and business people in your company, once solutions is stable
- Help move solution to new platform or vendor

Appendix A. Linear Algebra Revisited

This simple introduction to matrix theory offers a refreshing perspective on the subject. Using a basic concept that leads to a simple formula for the power of a matrix, we see how it can solve time series, Markov chains, linear regression, data reduction, principal components analysis (PCA) and other machine learning problems. These problems are usually solved with more advanced matrix calculus, including eigenvalues, diagonalization, generalized inverse matrices, and other types of matrix normalization. Our approach is more intuitive and thus appealing to professionals who do not have a strong mathematical background, or who have forgotten what they learned in math textbooks. It will also appeal to physicists and engineers. Finally, it leads to simple algorithms, for instance for matrix inversion. The classical statistician or data scientist will find our approach somewhat intriguing.

1. Power of a Matrix

For simplicity, we illustrate the methodology for a 2 x 2 matrix denoted as *A*. The generalization is straightforward. We provide a simple formula for the *n*-th power of *A*, where *n* is a positive integer. We then extend the formula to *n* = -1 (the most useful case) and to non-integer values of *n*.
Using the notation

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ and } A^n = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} a_{n-1} & b_{n-1} \\ c_{n-1} & d_{n-1} \end{pmatrix},$$

we obtain

$$\begin{cases} a_n = a \cdot a_{n-1} + b \cdot c_{n-1} \\ b_n = a \cdot b_{n-1} + b \cdot d_{n-1} \\ c_n = c \cdot a_{n-1} + d \cdot c_{n-1} \\ d_n = c \cdot b_{n-1} + d \cdot d_{n-1} \end{cases}$$

Using elementary substitutions, this leads to the following system:

$$\begin{cases} a_n = (a+d) \cdot a_{n-1} - (ad-bc) \cdot a_{n-2} \\ b_n = (a+d) \cdot b_{n-1} - (ad-bc) \cdot b_{n-2} \\ c_n = (a+d) \cdot c_{n-1} - (ad-bc) \cdot c_{n-2} \\ d_n = (a+d) \cdot d_{n-1} - (ad-bc) \cdot d_{n-2} \end{cases}$$

We are dealing with identical linear homogeneous recurrence relations. Only the initial conditions corresponding to *n* = 0 and *n* = 1, are different for these four equations. The solution to such equations is obtained as follows (see here for details.) First, solve the quadratic equation

$$x^2 - (a+d)x + (ad-bc) = 0.$$

The two solutions r_1, r_2 are

$$r_1 = \frac{1}{2}\left(a+d+\sqrt{(a+d)^2-4(ad-bc)}\right),$$
$$r_2 = \frac{1}{2}\left(a+d-\sqrt{(a+d)^2-4(ad-bc)}\right).$$

If the quantity under the square root is negative, then the roots are complex numbers. The final solution depends on whether the roots are distinct or not:

$$A^n = \begin{cases} r_1^n \cdot Q_1 + r_2^n \cdot Q_2, & \text{if } r_1 \neq r_2, \\ r_1^n \cdot Q_1 + n \cdot r_1^{n-1} \cdot Q_2, & \text{if } r_1 = r_2, \end{cases}$$

with

$$\begin{cases} Q_1 = (A - r_2 I)/(r_1 - r_2) & \text{if } r_1 \neq r_2, \\ Q_2 = (A - r_1 I)/(r_2 - r_1) & \text{if } r_1 \neq r_2, \\ Q_1 = I & \text{if } r_1 = r_2, \\ Q_2 = A - r_1 I & \text{if } r_1 = r_2. \end{cases}$$

Here the symbol ***I*** represents the 2 x 2 identity matrix. The last four relationships were obtained by applying the above formula for A^n, with n = 0 and n = 1. It is easy to prove (by recursion on n) that this is the correct solution.

If none of the roots is zero, then the formula is still valid for n = -1, and thus it can be used to compute the inverse of A.

2. Examples, Generalization, and Matrix Inversion

For a p x p matrix, the methodology generalizes as follows. The quadratic polynomial becomes a polynomial of degree p, known as the characteristic polynomial. If its roots are distinct, we have

$$A^n = \sum_{k=1}^{p} r_k^n \cdot Q_k, \text{ with } \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_p \end{pmatrix} = V^{-1} \begin{pmatrix} I \\ A \\ \vdots \\ A^{p-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ r_1 & r_2 & \cdots & r_p \\ \vdots & \vdots & & \vdots \\ r_1^{p-1} & r_2^{p-1} & \cdots & r_p^{p-1} \end{pmatrix}^{-1} \begin{pmatrix} I \\ A \\ \vdots \\ A^{p-1} \end{pmatrix}.$$

The matrix V is a Vandermonde matrix, so there is an explicit formula to compute its inverse, see here and here. A fast algorithm for the computation of its inverse is available here. The determinants of A and V are respectively equal to

$$|A| = (-1)^p r_1 r_2 \cdots r_p,$$
$$|V| = \prod_{1 \leq k < l \leq p} (r_k - r_l).$$

Note that the roots can be real or complex numbers, simple or multiple, or equal to zero. Usually the roots are ordered by decreasing modulus, that is

$$|r_1| \geq |r_2| \geq |r_3| \geq \cdots \geq |r_p|.$$

That way, a good approximation for A^n is obtained by using the first three or four roots if $n > 0$, and the last three or four roots if $n < 0$. In the context of linear regression (where the core of the problem consists of inverting a matrix, that is, using $n = -1$ in our general formula) this approximation is equivalent to performing a principal component analysis (PCA) as well as PCA-induced data reduction.

If some roots have a multiplicity higher than one, the formulas must be adjusted. The solution can be found by looking at how to solve an homogeneous linear recurrence equation, see theorem 4 in this document.

2.1. Example with a non-invertible matrix

Even if A is non-invertible, some useful quantities can still be computed when $n = -1$, not unlike using a pseudo-inverse matrix in the general linear model in regression analysis. Let's look at this example, using our own methodology:

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \Rightarrow A^n = 5^n \cdot \frac{1}{5}\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} + 0^n \cdot \frac{1}{5}\begin{pmatrix} 4 & -2 \\ -2 & 1 \end{pmatrix}.$$

The rightmost matrix attached to the second root 0 is of particular interest, and plays the role of a pseudo-inverse matrix for A. If that second root was very close to zero rather than exactly zero, then the term involving the rightmost matrix would largely dominate in the value of A^n, when $n = -1$. At the limit, some ratios involving the (non-existent!) inverse of A still make sense. For instance:

- The sum of the elements of the inverse of A, divided by its trace, is (4 - 2 - 2 + 1) / (4 + 1) = 1 / 5.
- The arithmetic mean divided by the geometric mean of its elements, is 1 / 2.

2.2. Fast computations

If n is large, one way to efficiently compute A^n is as follows. Let's say that $n = 100$. Do the following computations:

$$A^2 = A \cdot A, A^4 = A^2 \cdot A^2, A^8 = A^4 \cdot A^4, A^{16} = A^8 \cdot A^8,$$
$$A^{32} = A^{16} \cdot A^{16}, A^{64} = A^{32} \cdot A^{32}, A^{100} = A^{64} \cdot A^{32} \cdot A^4.$$

This can be useful to quickly get an approximation of the largest root of the characteristic polynomial, by eliminating all but the first root in the formula for A^n, and using $n = 100$. Once the first root has been found, it is easy to also get an approximation for the second one, and then for the third one.

If instead, you are interested in approximating the smallest roots, you can proceed the other way around, by using the formula for A^n, with $n = -100$ this time.

3. Application to Machine Learning Problems

We have discussed principal component analysis, data reduction, and pseudo-inverse matrices in section 2. Here we focus on applications to time series, Markov chains, and linear regression.

3.1. Markov chains

A Markov chain is a particular type of time series or stochastic process. At iteration or time *n*, a system is in a particular state *s* with probability *P*(*s* | *n*). The probability to move from state *s* at time *n*, to state *t* at time *n* + 1 is called a transition probability, and does not depend on *n*, but only on *s* and *t*. The Markov chain is governed by its initial conditions (at *n* = 0) and the transition probability matrix denoted as *A*. The size of the transition matrix is *p* x *p*, where *p* is the number of potential states that the system can evolve to. As *n* tends to infinity A^n and the whole system reaches an equilibrium distribution. This is because

- The characteristic polynomial attached to *A* has a root equal to 1.
- The absolute value of any root is less than or equal to 1.

3.2. AR processes

Auto-regressive (AR) processes represent another basic type of time series. Unlike Markov chains, the number of potential states is infinite and forms a continuum. Yet the time is still discrete. Time-continuous AR processes such as Gaussian processes, are not included in this discussion. An AR(*p*) process is defined as follows:

$$X_n = a_1 X_{n-1} + \cdots + a_p X_{n-p} + e_n.$$

Its characteristic polynomial is

$$x^p = a_1 x^{p-1} + a_2 x^{p-2} + \cdots + a_{p-1} x + a_p.$$

Here { e_n } is a white noise process (typically uncorrelated Gaussian variables with same variance) and we can assume that all expectations are zero. We are dealing here with a *non-homogeneous* linear (stochastic) recurrence relation. The most interesting case is when all the roots of the characteristic polynomial have absolute value less than 1. Processes satisfying this condition are called stationary. In that case, the auto-correlations are decaying exponentially fast.

The lag-*k* covariances satisfy the relation

$$\gamma(k) = \text{Cov}[X_n, X_{n-k}] = \begin{cases} a_1\gamma(k-1) + \cdots a_p\gamma(k-p) & \text{if } k \neq 0 \\ a_1\gamma(k-1) + \cdots a_p\gamma(k-p) + \sigma^2 & \text{if } k = 0 \end{cases}$$

with

$$\sigma^2 = \mathrm{Var}[e_n], \quad \mathrm{Var}[X_n] = \gamma(0), \quad \rho(k) = \mathrm{Corr}[X_n, X_{n-k}] = \gamma(k)/\gamma(0).$$

Thus the auto-correlations can be explicitly computed, and are also related to the characteristic polynomial. This fact can be used for model fitting, as the auto-correlation structure uniquely characterizes the (stationary) time series. Note that if the white noise is Gaussian, then the X_n's are also Gaussian.

The results about the auto-correlation structure can be found in this document, pages 98 and 106, originally posted here. See also this this document (pages 112 and 113) originally posted here, or the whole book (especially chapter 6) available here. See also section 4 (Appendix.)

3.3. Linear regression

Linear regression problems can be solved using the OLS (ordinary least squares) method, see here. The framework involves a response ***y***, a data set ***X*** consisting of *p* features or variables and *m* observations, and *p* regression coefficients (to be determined) stored in a vector ***b***. In matrix notation, the problem consists of finding ***b*** that minimizes the distance **||*y*** - ***Xb*||** between ***y*** and ***Xb***. The solution is

$$\hat{b} = A^{-1}X^Ty, \text{ with } A = X^TX.$$

The techniques discussed in this appendix can be used to compute the inverse of *A*, either exactly using all the roots of its characteristic polynomial, or approximately using the last few roots with the lowest moduli, as if performing a principal component analysis. If *A* is not invertible, the methodology described in section 2.1 can be useful: it amounts to working with a pseudo inverse of *A*. Note that *A* is a *p* x *p* matrix as in section 2.

Questions regarding confidence intervals (for instance, for the coefficients) can be addressed using the model-free re-sampling techniques discussed in chapter 16.

4. Appendix

Here we connect the dots between the auto-regressive time series described in section 3.2., and the material in section 2. For the AR(*p*) process in section 3.2., we have

$$X_n = g(e_p, e_{p+1}, \cdots, e_n) + \sum_{k=1}^{p} r_k^n \cdot Q_k, \text{ with } \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_p \end{pmatrix} = V^{-1} \begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_{p-1} \end{pmatrix}$$

where V is the same matrix as in section 2, the r_k's are the roots of the characteristic polynomial (assumed distinct here), and g is a linear function of e_p, e_{p+1}, ..., e_n. For instance, if p = 1, we have

$$g(e_p, e_{p+1}, \cdots, e_n) = \sum_{k=0}^{n-p} a_1^k \cdot e_{n-k}.$$

This allows you to compute Var[X_n] and Cov[X_n, X_{n-k}], conditionally to X_0, ..., X_{p-1}. The limit, when n tends to infinity, allows you to compute the unconditional variance and auto-correlations attached to the process, in the stationary case. For instance, if p = 1, we have

$$\text{Var}[X_n] = \lim_{n\to\infty} \sum_{k=0}^{n-p} a_1^{2k} \cdot \text{Var}[e_{n-k}] = \sigma^2 \sum_{k=0}^{\infty} a_1^2 = \sum \frac{\sigma^2}{1-a_1^2},$$

where σ^2 is the variance of the white noise, and $|a_1| < 1$ because we assumed stationarity.

For the general case (any p) the formula, if n is larger than or equal to p, is

$$g(e_p, e_{p+1}, \cdots, e_n) = \sum_{k=0}^{n-p} A_k e_{n-k}, \text{ with } A_k = \sum_{t=1}^{p} a_t A_{k-t}, \text{ and}$$
$$A_0 = 1, A_k = 0 \text{ if } k < 0.$$

The initial conditions for the coefficients A_k correspond to k = 0, -1, -2, ..., -(p -1), as listed above. The recurrence relation for A_k, besides the initial conditions, is identical to the previous one and thus can be solved with the same p x p matrix V and the same roots. If two time series models, say an ARMA and an AR models, have the same variance and covariance structure, they are actually identical.

Appendix B. Organized Chaos

I decided to add this appendix as it is a nice introduction to stochastic processes, time series and dynamical systems, with applications in experimental mathematics, Fintech, cryptography and Blockchain. Also it discusses new statistical tests (the Brownian test) and especially, several examples discussed in this book are based on the processes investigated in this appendix.

I present here some innovative results from my most recent research on stochastic processes. chaos modeling, and dynamical systems, with applications to Fintech, cryptography, number theory, and random number generators. While covering advanced topics, this appendix is accessible to professionals with limited knowledge in statistical or mathematical theory. It introduces new material not covered in my recent book (available here) on applied stochastic processes. You don't need to read my book to understand this article, but the book is a nice complement and introduction to the concepts discussed here.

None of the material presented here is covered in standard textbooks on stochastic processes or dynamical systems. In particular, it has nothing to do with the classical logistic map or Brownian motions, though the systems investigated here exhibit very similar behaviors and are related to the classical models. This cross-disciplinary appendix is targeted to professionals with interests in statistics, probability, mathematics, machine learning, simulations, signal processing, operations research, computer science, pattern recognition, and physics. Because of its tutorial style, it should also appeal to beginners learning about Markov processes, time series, and data science techniques in general, offering fresh, off-the-beaten-path content not found anywhere else, contrasting with the material covered again and again in countless, identical books, websites, and classes catering to students and researchers alike.

Some problems discussed here could be used by college professors in the classroom, or as original exam questions, while others are extremely challenging questions that could be the subject of a PhD thesis or even well beyond that level. This appendix constitutes (along with my book) a stepping stone in my endeavor to solve one of the biggest mysteries in the universe: are the digits of mathematical constants such as Pi, evenly distributed? To this day, no one knows if these digits even have a distribution to start with, let alone whether that distribution is uniform or not. Part of the discussion is about statistical properties of numeration systems in a non-integer base (such as the golden ratio base) and its applications. All systems investigated here, whether deterministic or not, are treated as stochastic processes, including the digits in question. They all exhibit strong chaos, albeit easily manageable due to their ergodicity.

Interesting connections to the golden ratio, Fibonacci numbers, Pisano periods, special polynomials, Brownian motions, and other special mathematical constants, are discussed throughout the article. All the analyses were done in Excel. You can download my spreadsheets from this appendix; all the results are replicable. Also, numerous illustrations are provided.

Content

1. General framework, notations and terminology

We are dealing here with sequences { x_n }, sometimes denoted as { $x(n)$ }, starting with $n = 1$, recursively defined by an iterative formula $x_{n+1} = g(x_n)$. We will explore various functions *g* in the next sections. Typically, x_n is a real number in [0, 1], and *g* is a mapping such that $x_{n+1} = g(x_n)$ is also in [0, 1]. The first, value, x_1, is called the *seed*. In short, { x_n } is a time series or stochastic process, and the index *n* denotes the (discrete) time or iteration.

Typically, the values x_n appear to be distributed somewhat randomly, according to some statistical distribution called the *equilibrium distribution*, and generally, the x_n's are auto-correlated. So x_n can be seen as a realization or observation of a random variable *X*, whose distribution is the equilibrium distribution. That is, the empirical distribution of the x_n's, when computed on a large number of terms, tends to the theoretical equilibrium distribution in question. Also, in practice, the vast majority of seeds yield the same exact equilibrium distribution. Such seeds are known as *good seeds*, the other ones are called *bad seeds*.

1.1. Finding the equilibrium distribution

The equilibrium distribution can be obtained by solving the equation P(X < y) = P(g(X) < y) with y in [0, 1]. This is actually a stochastic integral equation: the probability distribution P is the solution, and corresponds to the distribution of X. This distribution is sometimes denoted as F. Whether the equilibrium distribution exists or not, and whether it is unique or not (for good seeds), is not discussed here. However, we will provide several examples with unique equilibrium distribution, throughout this article, including how to solve the stochastic integral equation. The density attached to the equilibrium distribution is called the *equilibrium density* and is denoted as f.

1.2. Auto-correlation and spectral analysis

The theoretical auto-correlation between successive values of x(n) can be computed as follows:

$$E[X \cdot g(X)] = \int_0^1 y \cdot g(y) \cdot f_X(y) dy,$$

$$\rho(X, g(X)) = \frac{E[X \cdot g(X)] - E^2[X]}{E[X^2] - E^2[X]}.$$

Once computed, it is interesting to compare its value to the observed auto-correlation measured on the first few thousand terms of the sequence { x_n }. Longer-term auto-correlations (lag-2, lag-3 and so on) can be computed using the same principle, either theoretically or empirically (on data). The entire auto-correlation structure, given the equilibrium distribution, uniquely characterizes the stochastic process. The study of these auto-correlations is called *spectral analysis*.

1.3. Ergodicity, convergence, and attractors

So far we have looked at one instance or realization { x_n } of the underlying process, characterized by a mapping g and a seed x_1. This provides enough information to determine the auto-correlation structure and equilibrium distribution, which do not depend on the good seed.

There is another way to look at things. You can simulate m deviates of a random variable Z_1 with any pre-specified distribution, say uniform on [0, 1]. Then apply the mapping g to each of these deviates, to obtain another set of m values. These new values are m deviates of a random variable denoted as Z_2, also with known statistical distribution. Repeat this step over and over, to obtain Z_3, Z_4, and so on. For large n, Z_n converges in distribution to the equilibrium distribution, regardless of the initial distribution chosen for Z_1. We illustrate how it works on an example, later in this article. Because convergence to the same equilibrium distribution occurs regardless of the initial distribution, the equilibrium distribution, in the language of dynamical systems, is called an *attractor distribution*. The method described here can be used to identify these attractors.

A stochastic process where the equilibrium distribution does not depend on Z_1 nor on the good seed, is called *ergodic*. All the processes studied here are ergodic. An example of non-ergodic process can be found here.

1.4. Space state, time state, and Markov chain approximations

The *space state* is the space where { x_n } takes its values; here it is [0, 1] and is thus continuous. The *time space* is attached to the index n, and it is discrete here. However, in some of our examples, x_n can be written explicitly as a function of n, thus it can easily be extended to real values of n, making it a time-continuous process.

We can also divide the continuous space state [0, 1] into a finite number of disjoint intervals $S(1)$, $S(2)$, ..., $S(k)$. Rather than studying the full equilibrium distribution, we could compute the probabilities

$$p(i) = P(X \in S(i)),\ p_{ij} = P(g(X) \in S(j) | X \in S(i)),\ 1 \leq i, j \leq k.$$

Then we are dealing with a state-discrete Markov chain with k states, and $p(i, j)$ represents the transition probability for moving from state i to state j, estimated on n observations x_1, ..., x_n. One can compute the steady state probability vector, by solving a linear system of k equations; it represents the stable state of the system. As k and n tend to infinity and the intervals $S(1)$, ..., $S(k)$ become infinitesimal, the steady state probability vector converges to the equilibrium density. This is another way to approximate the equilibrium distribution.

1.5. Examples

The most well-known examples of { x_n } are random walks (n discrete) and Brownian motions (n continuous). However, since the space state considered in this appendix is [0, 1], a better suited example would be a random walk constrained to stay within [0, 1]. Such processes are discussed in my previous book, in chapter 3. Other less well known examples, but more relevant to this article, are also discussed in my previous book (chapters 7 to 12), and here, including the logistic map $x_{n+1} = 4\, x_n(1 - x_n)$. Also:

- $x_{n+1} = (\sin(\pi x_n))^b$, where b is a parameter between 0 and 1
- $x_{n+1} = |\log x_n|$, known as the logarithmic map, see here
- $x_{n+1} = 1 / |1 - x_n|^b$, where b is a parameter between 0 and 1 (typically b = 0.5)

Let us now introduce the example that we will discuss in detail throughout the remaining of this article. It is defined with the following mapping:

$$x(n+1) = g(x(n)) = \{bx(n)\} = bx(n) - \lfloor bx(n) \rfloor.$$

Here the curly brackets represent the fractional part function, also denoted as FRAC. The straight brackets on the right-hand side represent the integer part or floor function, also denoted as INT. Since the seed is between 0 and 1, we also have this interesting property: INT(bx_n) is the n^{th} digit of the seed x_1, in base b. Thus the parameter b is called the *base*, and it can be any real number greater than 1. In the next section, we consider the case where b is in]1, 2]. Non-integer bases are also discussed here and here, and also extensively in my previous book. While this process looks very peculiar, there is a mapping between the base-2 system, and the very popular logistic map system: see chapter 10 in my previous book for details, or here for a summary.

It makes sense to call this process the *base-b process*. If b is an integer, its equilibrium distribution is uniform on [0, 1] assuming you use a good seed. Also, if b is an integer, the auto-correlation between successive values of x_n is $1/b$. This fact was probably mentioned for the first time in my previous book. It was never proved, but assumed based on simulations and a lot of data crunching. The proof is actually quite simple and worth reading; it shows how to compute such auto-correlations, and constitutes an interesting classroom problem or exam question. You will find it in this appendix.

Pretty much all numbers in [0, 1] are good seeds for the b-process. However, there are infinitely many exceptions: in particular, none of the rational numbers is a good seed. Identifying the class of good seeds is an incredibly complicated problem, still unsolved today. If we knew which numbers are good seeds, we would know whether or not the digits of π or any popular mathematical constant, are evenly distributed. Another question is whether or not a good seed is just a normal number, and conversely. The two concepts are closely related, and possibly identical. Later in this appendix, we will discuss a stochastic process where all seeds are good seeds.

Finally, the most interesting values of b are those that are less than two. In some ways, the associated stochastic processes are also much easier to study. But most interestingly, the similarities between these b-processes and stochastic dynamical systems are easier to grasp, for instance regarding branching behavior, and attractors. This is the subject of the next section. The second fundamental theorem in the next section is one of the fascinating results published here for the first time, and still a work in progress.

Note that if b is an integer, it is easy to turn the time-discrete b-process into a time-continuous one. We have

$$x(n+1) = b^n x - \lfloor b^n x \rfloor.$$

Thus the formula can be extended to values of n that are not integers.

2. Case study

In this section, we consider the b-process introduced as an example in section 1.5, with b in]1, 2]. Thus, $x_{n+1} = g(x_n)$, with $g(x) = bx$ - INT(bx), and x_1 is the seed. We now jump right away to the two fundamental theorems, and cool applications will follow afterwards.

2.1. First fundamental theorem

Let *Z* be a random variable with an arbitrary distribution *F*, admitting a density function *f* on [0, 1]. Let *Y* = *g*(*Z*) be the fractional part of *bZ*, and *b* in]1, 2]. Then we have:

$$\text{If } 0 \leq y \leq b-1, \text{ then } f_Y(y) = \frac{1}{b} f_Z\Big(\frac{1+y}{b}\Big) + \frac{1}{b} f_Z\Big(\frac{y}{b}\Big),$$

$$\text{If } b-1 < y \leq 1, \text{ then } f_Y(y) = \frac{1}{b} f_Z\Big(\frac{y}{b}\Big).$$

This result is easy to obtain and constitutes an interesting classroom problem, or exam question. The proof is in the appendix. This theorem allows you to design a simple iterative algorithm to approximate the equilibrium distribution, and to assess how fast it converges to the solution. The result is valid even if the density function of *Z* has an infinite but countable number of discontinuities. This will be the case in the examples discussed here, in which a uniform distribution on [0, 1] is chosen for *Z*.

The algorithm, already discussed in the first section (see the ergodicity, convergence and attractors subsection), consists in iteratively computing the distribution of g(Z), *g*(*g*(*Z*)), *g*(*g*(*g*(*Z*))), and so on, until the difference between two successive iterates is small enough. Here, the difference is measured as the distance between two distributions, using one of the many distance metrics discussed in the literature (see here.)

The next theorem tells you in more details what happens if you choose a uniform distribution on [0, 1], for *Z.* This was our favorite choice in most of our simulations.

2.2. Second fundamental theorem

We use the same assumptions as in the first theorem, but here *Z* has a uniform distribution on [0, 1]. The following theorem can be used to find the equilibrium density, as illustrated in the appendix, using the supergolden ratio constant for *b*:

Let $Z_1 = Z$, and $Z_{n+1} = g(Z_n)$. Then Z_{n+1} has a piece-wise uniform distribution, more precisely, a mixture (see chapter 11) of *n*+1 uniform distributions on *n*+1 intervals. These intervals are denoted as

$$[0, c_1[, \quad [c_1, c_2[, \quad [c_2, c_3[, \ldots, [c_n, 1],$$

and the constant value of the density of Z_{n+1} on the k^{th} interval (*k* = 1, ..., *n*+1) is denoted as d_k. The distribution of Z_{n+1} has the following features:

- Sometimes $c_{k-1} = c_k$ depending on *b*, *k*, and *n*
- $b^n d_k$ is an integer and $\{ d_k \}$ is a decreasing sequence
- c_k is a polynomial of degree *n* in *b*, with coefficients equal to 0, 1, or -1
- Only the dominant coefficient of this polynomial is equal to 1

It is convenient to use the notation $c_0 = 0$ and $c_{n+1} = 1$. The c_k's, for $k = 1, ..., n$ are called the *change points*. A change point is thus a discontinuity in the density function. One of these change points is always equal to $b - 1$.

I haven't completed the proof of the second theorem yet, and the theorem itself can probably be further refined. However, using the first fundamental theorem, it is easy to see that when moving from iteration n to $n+1$, we observe the following:

- Because b is smaller than 2 and Z_{n+1} takes on value between 0 and 1, it is clear that Z_{n+1}, the fractional part of bZ_n, takes more frequently on smaller values (closer to 0) than on larger ones (closer to 1.) Thus the interval densities d_k are highest next to zero, and lowest next to 1, and decreasing in between. This explains why $\{ d_k \}$ is a decreasing sequence.
- The densities are also constant on each interval, as we are only dealing with uniform densities, throughout the iterations. Also $b^k d_k$ must be an integer, as the formula in the first fundamental theorem only involves adding integers (divided by b). This is easy to prove by recursion.
- Finally, at iteration $n = 2$, we have a single change point $c_1 = b - 1$, and two intervals. Any new iteration, because of the formula in the first fundamental theorem, creates a whole new set of new change points, each one either equal to cb or $c(b - 1)$, where c is any change point from the current iteration. This explains the special polynomial expression for c_k.

For any value of n, the exact distribution of Z_{n+1} can be computed explicitly. The computation is elementary, but becomes incredibly tedious (and should be automated using some software) as soon as n is larger than 5: this is a combinatorial problem. But particular results are easier to obtain. The simplest case is as follows:

$$\text{If } b^{n-1}(b-1) < 1 \text{ then } d_k = (n-k+2)/b^n, c_k = b^{n-k+1}(b-1), \text{ for } k = 1, \cdots, n+1.$$

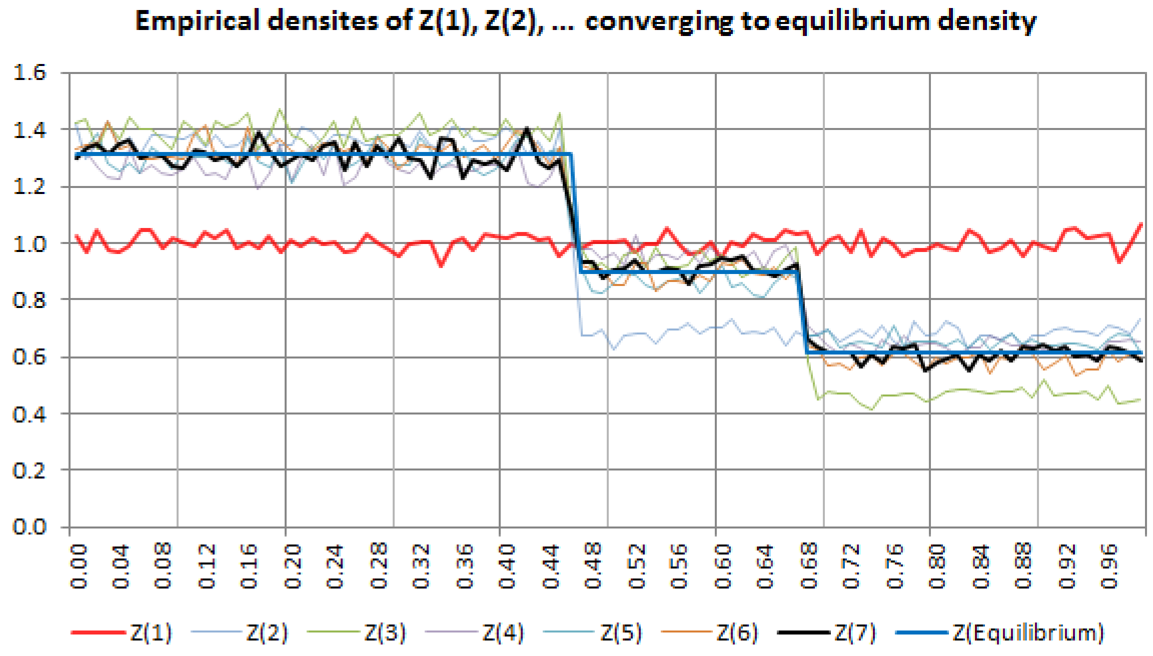
Exercise: prove that this is actually a density, that is,

$$\sum_{k=1}^{n+1} d_k(c_k - c_{k-1}) = 1.$$

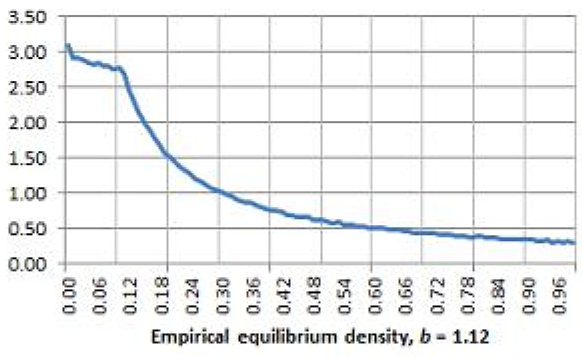
Other easy cases include the full solution (for any value of b between 1 and 2) when n = 2 or n =3. This is left as an exercise. Note that if for some finite value of n, Z_n has the equilibrium density, then Z_{n+1}, Z_{n+2} and so on also have that exact same density. This is why the equilibrium distribution is called an attractor.

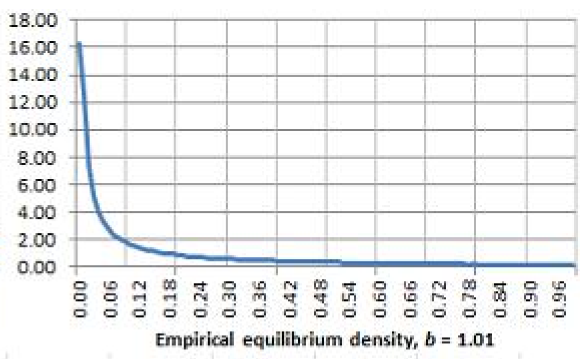
2.3. Convergence to equilibrium: illustration

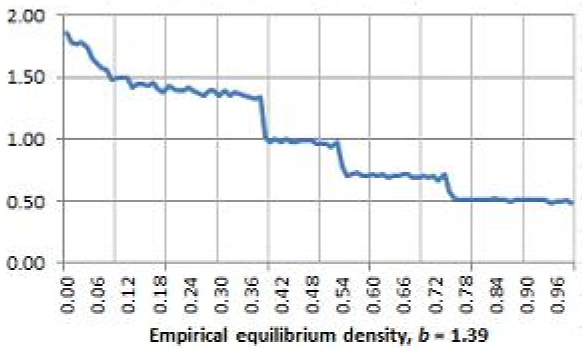
The first picture below illustrates the convergence of the empirical equilibrium densities to the theoretical solution, starting with a simulated uniform density on [0, 1] for Z_1, and computing the empirical densities for Z_2, Z_3, and so on, up to Z_7. You can check out the computations in this spreadsheet. The parameter b used here is the supergolden ratio constant (see next section) and we used 100,000 observations to estimate each density.

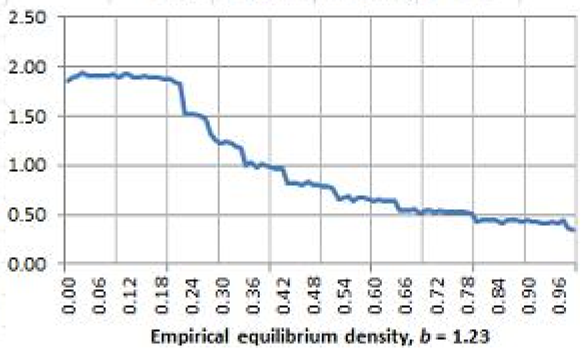


Below are a few equilibrium densities (approximated using the empirical density) for various values of *b*.









The spreadsheet used to produce the 4 above charts, with detailed computations, is available here. Some exact solutions (where the theoretical equilibrium density is easy to compute) are provided in the next section and in the appendix, with a short tutorial on how to discover these solutions and to apply the methodology to the general case (see appendix.).

3. Applications

In this section, we discuss applications, still focusing for the most part on *b*-processes with *b* smaller than 2. But we also discuss other stochastic processes.

3.1. Potential application domains

Stochastic processes or time series models are routinely used in Fintech to model the price of some commodities. Thus, *b*-processes offer a new tool for quants, behaving quite differently than traditional processes and time series. The variety in these *b*-processes is such, and the behavior so unique depending on *b*, that it allows the data scientist to attach a unique number to an observed time series: its estimated parameter *b*. Two different values of *b* provide two wildly different types of patterns, as is usually the case with all chaotic dynamical systems, for instance, with the logistic map. Whether in finance or other fields, these processes model situations in which an auto-correlated system evolves chaotically over time, with sharp drops every now and then in the equilibrium density, occurring at what we defined earlier as change points. Depending on *b*, the number of change points in [0, 1] can be 2, 3, 4, and so on, up to values so large that the equilibrium density looks perfectly smooth (this is the case, for instance if *b* is very close to 1.) Thus the parameter *b* can be chosen to fit with a wide array of change point locations, as well as various downward trends and gaps, in the equilibrium density. As discussed in section 2, the *b*-process can be seen as an infinite mixture of uniform distributions on infinitesimal intervals, or finite mixture on larger intervals, depending on *b*.

Other specific applications include:

- Generation of non-periodic, continuous, replicable **pseudo-random numbers**. By far, the largest class of pseudo random number generators currently in use is made of periodic, discrete generators, though the period is extremely large in modern generators. And random numbers produced using physical devices are typically not replicable. To get a good generator, one would have to use a value of *b* resulting in a known equilibrium distribution, start with a good seed, and map the sequence { x_n } so that the distribution of the mapped x_n's (each one representing a random number) becomes uniform, with no auto-correlation. How to do this is described in the next sub-section about the golden ratio process.

- Thus, with a careful choice of *b* and proper mapping, *b*-processes can be used in **cryptographic systems** and Blockchain. Digits produced by *b*-processes, and defined as a_n = INT(bx_n), have the following particular property. The digits are binary (equal to 0 or 1), so each digit can be called a *bit*, using the language of information theory. Indeed, when *b* = 2, this is just the standard base-2 numeration system that we are all familiar with. However, when b is smaller than 2, each digit carries an amount of information smaller than the standard bit. Indeed, that amount is equal to the logarithm of b in base 2 (and of course, equal to 0 if b = 1.) So not only we invented a unit of information that is smaller than the smallest unit currently in use, but it allows you to create encryption systems filled with some amount of natural blurring, which may or may not be useful depending on the purpose.

- Another application is to **benchmark computer systems**, testing for accuracy when performing heavy computations that require a large number of decimals. If you compute the successive values of x_1, x_2 and so on up to x_n, all your numbers will be completely wrong once n is larger than 45. You may not notice it initially, but try in Excel with a base b that is an even integer: it will become very obvious! Sometimes it does not matter (for instance when studying asymptotic properties such as auto-correlations or the equilibrium distribution) because b-processes are ergodic, and sometimes it matters. This is discussed in detail in my previous book, available here: see the chapters about high precision computing, or read this article.

- Also, b-process can be used to **benchmark and test the power of statistical tests**, the sample size needed, and other statistical procedures. Since the "observations" { x_n } have a known statistical distribution (depending on b, see next subsection about the golden ratio process) -- a property never found in actual, real life data sets -- you can test a number of hypotheses (for instance about the value of some auto-correlation), and check when your statistical tests provide the right or wrong answer. Here, the right answer is known in advance!

- Another application is to **design a lottery**, where the winning number is a sequence of digits generated after re-mapping a b-process so that its associated digits are uniformly distributed, and with no auto-correlation. See the golden ratio process below about how to do this re-mapping. Use digits in positions n = 1,001 to 1,020 the first week, 1,021 to 1,040 the second week, and so on. The advantage of such a lottery is that the winning numbers are known in advance yet unpredictable unless you know the secret base b, the secret seed x_1, and the secret starting point n = 1,001. So it can be labeled as a game of skills rather than a game of chance, and not subject to lottery laws. Another example of such a "lottery" (number guessing) played with real money and cryptocurrency is described in chapter 18.

3.2. Example: the golden ratio process

The golden ratio process, as its name indicates, corresponds to $b = (1 + 5^{1/2})/2$. Its associated numeration system, with INT(bx_n) representing the n^{th} digit of the seed $x(1)$ in base b, has been studied in some details, see here. This b-process is the best behaved one, and the easiest to study, besides b = 2. It is probably the only b-process with exactly one change point, and its equilibrium distribution is known (probably published here for the first time.) Thus, it is a good candidate for applications such as encryption, random number generation, model fitting, testing statistical tests, or Blockchain.

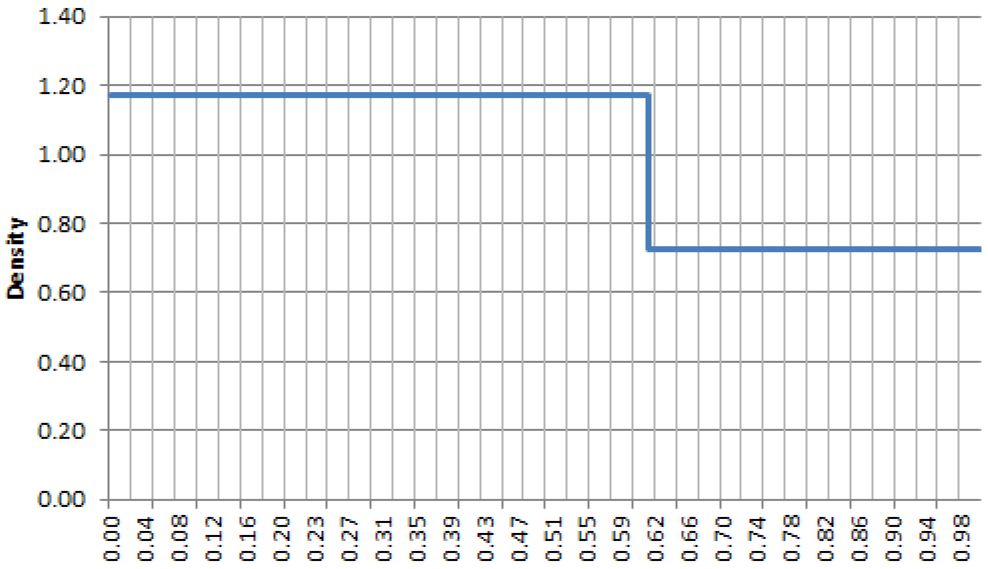
(a) Properties and use in cryptography

Using the notations introduced in section 2, this process has the following features:

- The unique change point is $c_1 = b - 1$
- The equilibrium distribution is a mixture of two uniform distributions: one on $[0, c_1[$, and one on $[c_1, 1[$
- At equilibrium, the two respective densities are $d_1 = (5 + 3*5^{1/2})/10$ and $d_2 = (5 + 5^{1/2})/10$.

Below is a picture of the equilibrium density associated with this process:

Equilbrium density for the golden ratio process



In order to make this process suitable for use in cryptography, one has to map { x_n } onto a new sequence { y_n }, so that the new equilibrium density becomes uniform on [0, 1]. This is achieved as follows:

If $x_n < b$ -1, then $y_n = x_n/(b - 1)$ else $y_n = (x_n - (b - 1))/(2 - b)$.

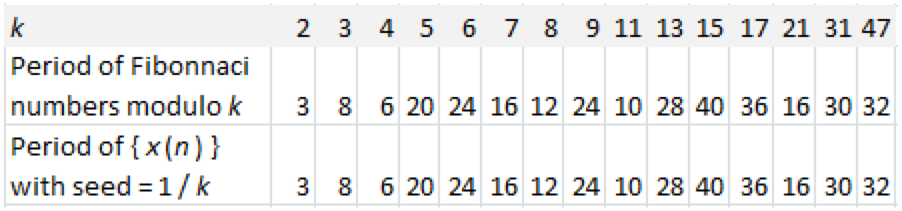
Now the { y_n } sequence has a uniform equilibrium distribution on [0, 1]. However, this new sequence has a major problem: high auto-correlations, and frequently, two or three successive values that are identical (this would not happen with a random *b*, but here *b* is the golden ratio -- a very special value -- and this is what is causing the problem.)

A workaround is to ignore all values of x_n that are larger than b - 1, that is, discarding y_n if x_n is larger than b -1. This is really a magic trick. Now, not only the lag-1 auto-correlation in the remaining { y_n } sequence is equal to 1/2, the same value as for the full { x_n } sequence with b = 2, but the lag-1 auto-correlation in the remaining sequence of binary digits (digits are defined as INT(by_n) is also equal to zero, just like for ordinary digits in base 2! The proof of these facts is left as an exercise.

(b) Bad seeds and connection to Fibonacci numbers

Fibonacci numbers are defined by the recursion $F_{n+1} = F_n + F_{n-1}$, with $F_1 = F_2 = 1$. Also, F_n = INT(xb^n) +1 if n is odd, and F_n = INT(xb^n) otherwise, with $x = 5^{-1/2}$. Thus they are related to the golden ratio process with $b = (1 + 5^{1/2})/2$ and the seed $x = x_1 = 5^{-1/2}$. That seed is actually a bad seed, resulting in periodicity: $x_{n+4} = x_n$.

But the link to bad seeds of the golden ratio process does not stop here. All fractions 1/k, with k a positive integer, are also bad seeds, resulting in periodicity in { x_n }. Here we compare these periods with those (well-known) of Fibonacci numbers modulo k, known as Pisano periods. If you are not familiar with elementary modulo arithmetic, you can check the Wikipedia page on the subject, here.



k	2	3	4	5	6	7	8	9	11	13	15	17	21	31	47
Period of Fibonnaci numbers modulo *k*	3	8	6	20	24	16	12	24	10	28	40	36	16	30	32
Period of { *x* (*n*) } with seed = 1 / *k*	3	8	6	20	24	16	12	24	10	28	40	36	16	30	32

I only tested a few values of k, but in all cases, both periods were identical.

3.3. Finding other useful *b*-processes

There are different ways to compute the equilibrium distribution of a b-process when you have 3 change points or less. Finding the change points is easy: one of them is always b-1, and the other ones can be any of these:

$$b^3 - 1, b^3 - b, b^3 - b^2, b^3 - b^2 - b, b^3 - b^2 - 1, b^3 - b - 1, b^2 - b, b^2 - 1.$$

You can identify them by visual inspection of the empirical equilibrium density. And among the 8 potential change points listed above, you must ignore those below 0 or above 1. Note that the golden ratio process actually has two change points: $b^2 - 1$ and b - 1. But $b^2 - b = 1$ in this case, thus the first one is not a real change point. If you try with b = 1.61 (very close to the golden ratio) this ghost change point is now visible, and it is very close to 1. If you try b = 1.60, you now have 3 change points. And with b = 1.59, the empirical equilibrium density looks entirely different, possibly with a lot of change points and no visible drop (just a linear, bumpy downward trend), though it is hard to tell. In some cases, a change point can be double (or triple) for instance if $b^3 - b^2$

- $b = b$ - 1. It typically results in an equilibrium density with very few change points. Once the change points are known, the densities can be computed using the first fundamental theorem (see section 2 in this article) and solving a linear system of equations. This is illustrated in the appendix.

Interestingly, the *b*-processes most likely to have a simple equilibrium density with very few change points correspond to *b*'s for which two of the above polynomials have the exact same value, or one is equal to 0 or 1, when evaluated for the *b* in question. This was the case for the golden ratio process. Below are other examples:

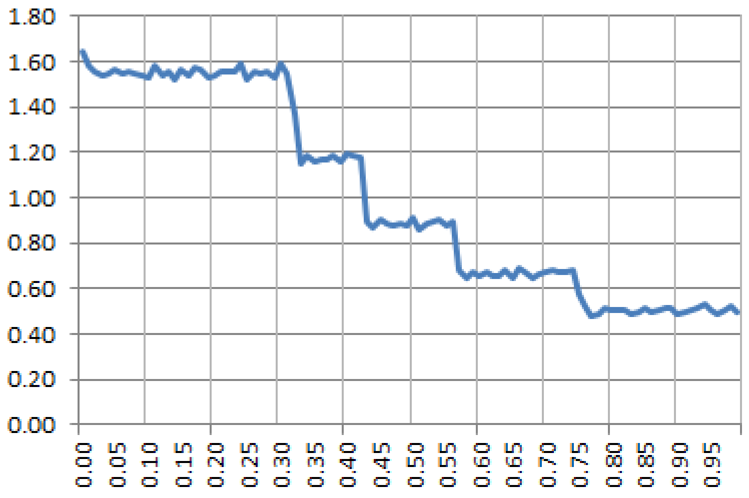
- $b^3 - b^2 = 1$ yields b = 1.4655712318767... (3 change points, *b* is the supergolden ratio)
- $b^3 - b^2 - b = b - 1$ yields b = 1.8019377358048... (3 change points)
- $b^5 - b^4 = 1$ or $b^3 - b = 1$ yields b = 1.32471795724475... (4 change points, *b* is the plastic number)

You can find more about these three special constants, in this article. The exact values, respectively for the supergolden ratio and the plastic number, are

$$b = \frac{1}{3}\left\{1 + \left(\frac{29 - 3\sqrt{93}}{2}\right)^{1/3} + \left(\frac{29 + 3\sqrt{93}}{2}\right)^{1/3}\right\},$$

$$b = \frac{(9-\sqrt{69})^{1/3} + (9+\sqrt{69})^{1/3}}{2^{3/2}\cdot 3^{2/3}}$$

To get an approximation of the equilibrium distribution and see the change points, start with a good seed x_1. For whatever reasons $x_1 = 2^{1/2}/2$ works better than $\pi/4$. Then compute x_n up to N = 1,000,000, then plot the empirical equilibrium density computed on the x_n's. This is illustrated in my spreadsheet, available here. See also the picture below based on values of x_n for n =1, ..., 300,000, with *b* being the plastic number.



Empirical equilibrium density, *b* = plastic number

As a general rule, the lower the value of b, the more change points, on average. Also, most values of b (whether special or not) always produce a few major change points (and frequently a large number of minor ones), with big drops in the density function occurring at the major change points. Analyzing the polynomials discussed in the second fundamental theorem, can help you identify these major change points.
In the appendix, we completely solve the case where b is the supergolden ratio.

4. Additional research topics

Here we discuss three potential topics for future research: stochastic processes free of bad seeds, the asymptotic properties of attractors and the construction of a table of attractors summarizing their features, and finally, some applications of b-processes to probabilistic and experimental number theory, including the discussion of some special integrals.

4.1. Perfect stochastic processes and Brownian motion

The b-process, defined by $g(x) = bx$ - INT(bx), has bad seeds, as discussed earlier. For a b-process, the vast majority of seeds are good seeds (the set of bad seeds actually has Lebesgue measure zero), but nobody knows if mathematical constants such as π or $2^{1/2}$ are good or bad seeds. Are there any stochastic processes free of bad seeds? Such processes can have some benefits (but mostly drawbacks!) and are called *perfect processes*, until someone comes up with a better word. The term *universally good averaging sequence* is sometimes used. One example is the following.

The process defined by $g(x) = x + b$ - INT($x + b$), where b is a positive irrational number, fits the bill. Since by definition, $x_{n+1} = g(x_n)$, it is easy to see that

$$x_n = (n - 1)b + x_1 - \text{INT}((n - 1)b + x_1).$$

The fact that there is no bad seed is guaranteed by the equidistribution theorem.
Even $x_1 = 0$ is a good seed.

(a) Properties of perfect processes

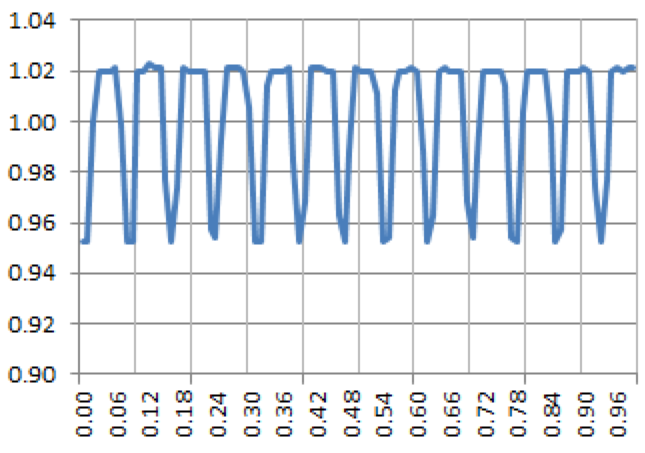
This process is investigated in chapter 11 in my previous book, available here (see page 70.) The n^{th} binary digit is defined as INT($2x_n$), and these digits carry even less information than those generated by b-processes with b between 1 and 2. If b = log 2, the first few digits of the seed $x_1 = 0$ are as follows:

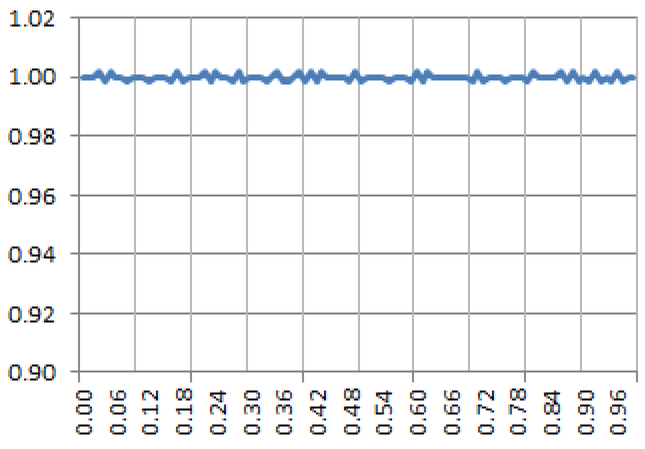
0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0

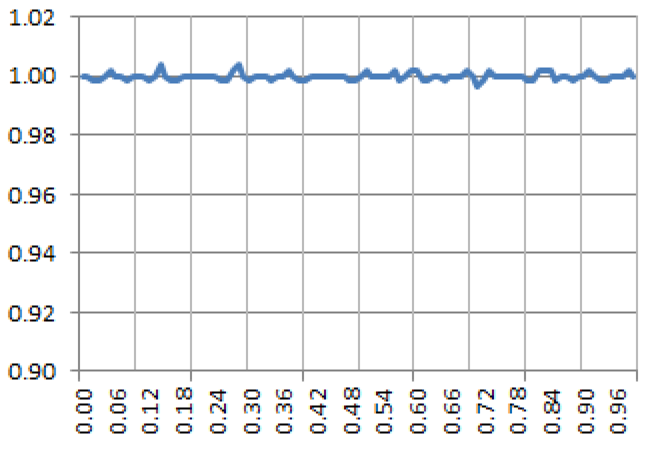
In contrast to b-processes, all seeds (regardless of b) have 50% of digits equal to 0, and 50% equal to 1. This process is related to integral C defined later in this appendix, in the sub-section *Probabilistic calculus and number theory, special integrals*.

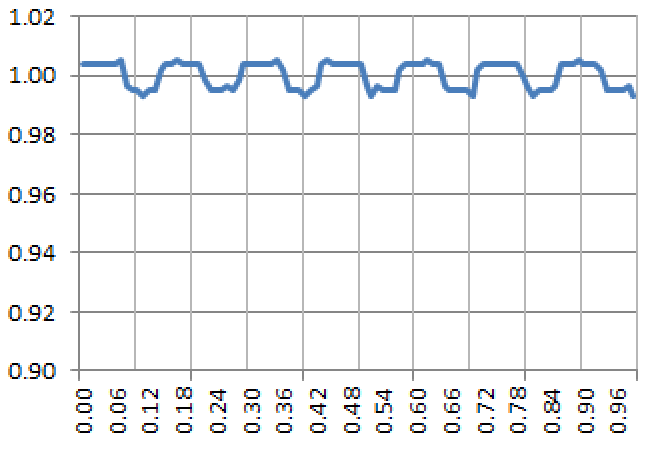
The equilibrium distribution is always uniform if *b* is irrational, thus it is possible to compute the theoretical lag-1 auto-correlation of the sequence { x_n } (using the first formula in section 1) and search for the *b*'s that minimize, in absolute value, that auto-correlation. See appendix for a detailed solution. The empirical equilibrium distribution converges much faster to the theoretical one, than with *b*-processes. However, I've found a striking, unusual pattern for $b = \pi$ and $b = \exp(\pi)$.

The empirical density, computed on $x_1, ..., x_n$ and binned into *N* intervals, shows strong periodic bumps that other irrational *b* do not produce, not even $b = \pi - 0.00001$. It occurs even with the seed $x_1 = 0$, with specific values of *n* and *N*, for instance $n = 10{,}000$ and $N = 100$, or $n = 1{,}000{,}000$ and $n = 100$, but not with $N = 100{,}000$ and $N = 100$. These bumps decrease as *n* increases, and convergence to uniform [0, 1] still occurs for $b = \pi$ and $b = \exp(\pi)$. Initially, I thought it was an issue with my internal machine arithmetic, but both my Perl and Excel implementations reproduce the same patterns. The Perl code is available here. The pattern is illustrated in the figure below.


Empirical equilibrium density for *b* = Pi


Empirical equilibrium density for *b* = SQRT(10)

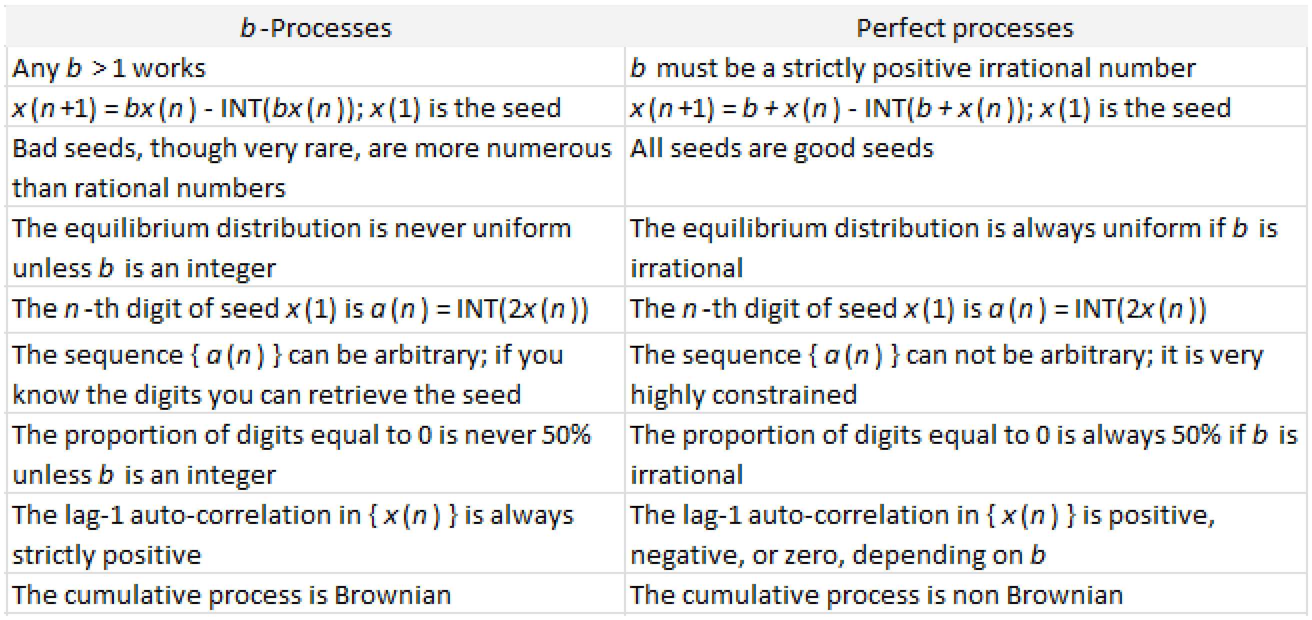

Empirical equilibrium density, *b* = Pi - 0.00001


Empirical equilibrium density, *b* = log(63)

The above picture shows the empirical density, with $n = 56{,}000$ observations and $N = 100$ bins, for four values of *b*. It is extremely close to the theoretical uniform equilibrium on [0, 1]. I truncated the *Y*-axis to visually amplify the pattern. The spreadsheet is available here.

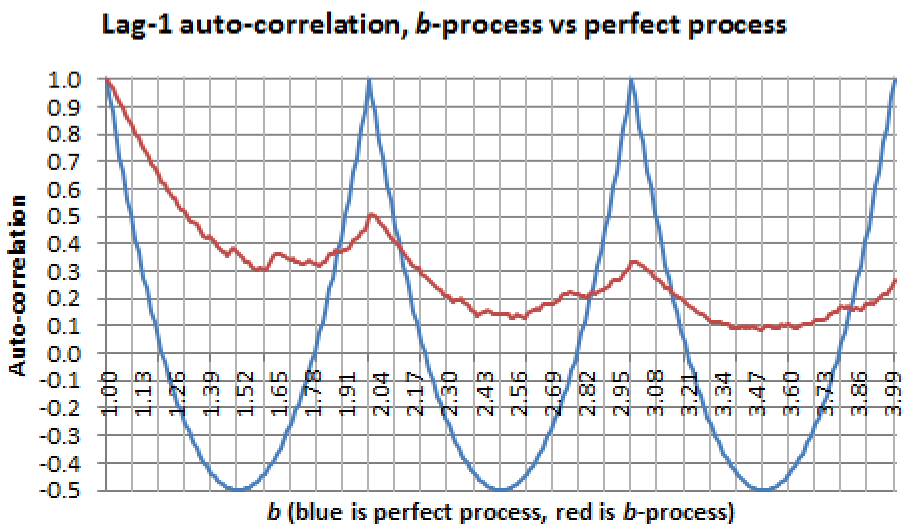
(b) Comparison with *b*-processes

Below we contrast some of the properties of *b*-processes, with those of perfect processes.



b-Processes	Perfect processes
Any $b > 1$ works	b must be a strictly positive irrational number
$x(n+1) = bx(n) - \text{INT}(bx(n))$; $x(1)$ is the seed	$x(n+1) = b + x(n) - \text{INT}(b + x(n))$; $x(1)$ is the seed
Bad seeds, though very rare, are more numerous than rational numbers	All seeds are good seeds
The equilibrium distribution is never uniform unless b is an integer	The equilibrium distribution is always uniform if b is irrational
The n-th digit of seed $x(1)$ is $a(n) = \text{INT}(2x(n))$	The n-th digit of seed $x(1)$ is $a(n) = \text{INT}(2x(n))$
The sequence $\{ a(n) \}$ can be arbitrary; if you know the digits you can retrieve the seed	The sequence $\{ a(n) \}$ can not be arbitrary; it is very highly constrained
The proportion of digits equal to 0 is never 50% unless b is an integer	The proportion of digits equal to 0 is always 50% if b is irrational
The lag-1 auto-correlation in $\{ x(n) \}$ is always strictly positive	The lag-1 auto-correlation in $\{ x(n) \}$ is positive, negative, or zero, depending on b
The cumulative process is Brownian	The cumulative process is non Brownian

Below is a chart comparing the auto-correlation of *b*-processes with that of perfect processes. The red curve was computed empirically (based on simulations) while the blue curve represents the exact values. The small bumps in the red curve are real; they are not caused by a small sample size. Note that *b* is in [1, 4] here. While in this appendix we focused on *b* between 1 and 2 for *b*-processes, it can easily be extended to any *b* larger than 2.



Auto-correlations and cross-correlations in multivariate processes are studied in chapter 13. For *b*-processes, the lag-*k* auto-correlation in base *b* is equal to the lag-1 auto-correlation in base b^k. For perfect processes, the lag-*k* auto-correlation in base *b* is equal to the lag-1 auto-correlation in base b^k. These results are valid for any good seed.

(c) Connection with Brownian motions

Now, let us investigate the connection with Brownian motions. It seems that few of the perfect processes investigated here can emulate Brownian motions, because they are usually too strongly auto-correlated. But there are a few exceptions. The opposite is true for *b*-processes.

Let us define

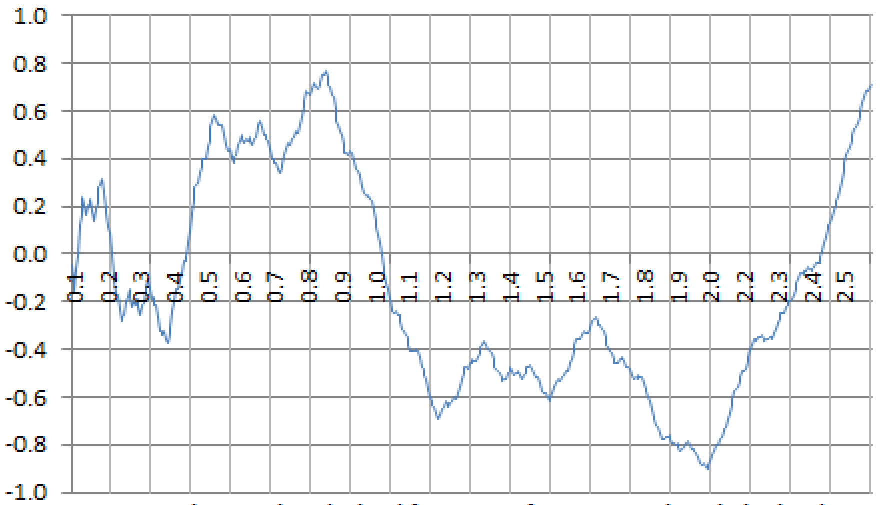
$$y(n) = \frac{1}{u(n)} \sum_{k=1}^{n} \Big(x(k) - E\Big), \; z(n) = \frac{1}{v(n)} \sum_{k=1}^{n} u(k) \Big(y(k) - w(n)\Big).$$

where *E* is the expected value of { x_n }, and u_n, v_n and w_n are functions chosen to stabilize the variances of { y_n } and { z_n } respectively, as *n* becomes large. For details about stabilizing the variance to turn time-discrete processes such as { y_n } or { z_n } into time-continuous processes such as Brownian motions (the time is the index *n*), see chapter 1 and 2 in my previous book, here. In short, it consists of re-scaling both the *Y*-axis (values) and *X*-axis (time, or *n*) to make sure that variances stay finite and non-zero as *n* tends to infinity. As time increments become infinitesimal and *n* tends to infinity, convergence to a continuous process is obtained. The textbook example is the transformation of a random walk into a Brownian motion. Here we only provide two examples, and technical details are omitted.

The first example is the *b*-process with *b* an integer larger than 1. In this case, with x_1 a good seed thus $E = 1/2$,with $u_n = n^{1/2}$, $v_n = n^{3/2}$, and $w_n = 0$, after re-scaling the time axis,

{ y_n } becomes a Brownian motion, while { z_n } becomes an integrated Brownian motion. There is nothing new here. What is new though, is the fact that this works too even if b is not an integer.

The second example involves a perfect process, with the seed $x_1 = 1$, $b = 2^{1/2} - 1$, $E = 1/2$, $u_n = 1$, $v_n = n^{1/2}$, and $w_n = n/2$. After re-scaling, this time { z_n }, not { y_n }, looks like a Brownian motion, see picture below.



Brownian motion derived from a perfect process (X-axis is time)

If you try the perfect process with a different parameter b and a different seed, the result will usually be very different, sometimes unexpectedly beautiful with many smooth bumps, if your sample is large enough, but it won't look at all like a Brownian motion. All the computations are in my spreadsheet, available here. You can play with it to see the variety of patterns that it can produce. Here only the first 500 values of z_n have been used. If you try with the first 50,000 values instead, it still looks Brownian, and indeed, strikingly similar due to the fractal nature of Brownian motions (when you zoom in or out, the randomness patterns stay the same.) A statistical test to assess the Brownian character of this time series would probably conclude that it is likely to be Brownian. One such test was designed by Grzegorz Sikora, see here; his article was submitted for publication in 2018.

One of the issues with very large n is machine precision. As a test, replicate this example with a large sample, using only 8 correct decimals in your computation, rather than the 15 that Excel offers by default. Is the resulting chart still the same? If you want to replicate the example with the b-process and b an integer, avoid $b = 2$, try $b = 3$ instead, because programming languages and Excel rely on base-32 or base-64 arithmetic, and assign the value 0 to x_n when $n > 55$ or so. A workaround is to use high precision computing, or use $b = 1.9999999$ (1.999 won't work, and indeed theoretically, it is not supposed to work.)

The connection to Brownian motions (including the smoothness of b-processes versus perfect processes) is further studied in chapter 12.

4.2. Characterization of equilibrium distributions (the attractors)

Here, we focus again on *b*-processes with *b* in]1, 2]. Another interesting research topic is about characterizing the class of attractors. That is, what kind of distribution is an equilibrium distribution? What makes them peculiar compared to other distributions? Another question is about how the number of attractors grows as the number of change points increases. Is there an asymptotic relationship between the number of change points (say *N*), and the number of attractors that have *N* change points?

Number of change points (or discontinuities in the equilibrium density function)	Known attractors (or equilibrium densities)	Corresponding values of *b*
0	1	*b* = 2
1	1	*b* = 1.618... (golden ratio)
2	2	*b* = 1.465... (surpergolden ratio), *b* = 1.801...
3	1	*b* = 1.324... (plastic number)

It is not even known if the number of attractors with a finite number of change points, is finite or infinite. Surely, there are more than two attractors with two change points, and much more than one attractor with three change points. The ones listed in the above table are only those that I have studied. So this table is a work in progress.

4.3. Probabilistic calculus and number theory, special integrals

When I first started this research project a while back, the initial purpose was to study the behavior of the digits of numbers such as π. In fact, in this article, INT(bx_n) represents the n^{th} digit of the seed x_1 in base *b*, whether *b* is an integer, a real number between 1 and 2, or any real number above 1. My previous book *Applied Stochastic Processes, Chaos Modeling, and Probabilistic Properties of Numeration Systems* published in June 2018 (see here) was the first milestone: developing a general framework to study this kind of problems. Since then, I have had new ideas. Here, I present some of them that I am still pursuing today.

In this subsection, the notation { *x* } represents the fractional part of the number *x*, in contrast to the remaining of this appendix, where $\{ x_n \}$ represents the entire sequence x_1, x_2, and so on. Here we will only consider the case $b = 2$. Also, the seed x_1 is denoted as *x*.

If *b* is an integer and if the seed $x = x_1$ is in [0, 1], we have, for *k* larger or equal to 0:

$$x(k+1) = \{b^k x(1)\} = \{b^k x\}.$$

(a) Interesting series, limits, and integrals

One of the promising starting points is the following result. The proportion of digits of *x* equal to 0 in base 2 is 50% if and only if the series below, with *b* = 2, converges:

$$h(x,b)=\sum_{k=1}^{\infty}\frac{\{b^kx\}-\frac{1}{2}}{k}.$$

In particular, it always converges if *x* is a good seed in base 2. It would be interesting to study the wildly erratic behavior of this function, which is not only discontinuous everywhere, but admits a dense set of singularities (where it does not converge.) Note that if we replace b^k by *k* in the above series, it always converges whenever *x* is an irrational number, a consequence of the equidistribution theorem. What happens if we replace b^k by k^2 *or* k^b?

A related quantity is the following:

$$M(b)=\lim_{n\to\infty}\sum_{k=n+1}^{2n}\frac{\{b^kx\}}{k}.$$

If *b* is an integer, *M*(*b*) = (log 2)/2 and does not depend on *x*, assuming *x* is a good seed. If *b* is not an integer, *M*(*b*) is smaller than (log 2)/2. More precisely, *M*(*b*) = *E*(*b*) log(2), where *E*(*b*) is the expected value of the equilibrium distribution of a *b*-process. If *b* is between 1 and 2, then *E*(*b*) is approximately equal to the proportion of binary digits in base *b*, that are equal to 1, for a good seed *x*; it does not depend on x. For instance, based on results established in section 3.2.(a), we have:

$$M\left(\frac{1+\sqrt{5}}{2}\right)=\frac{\sqrt{5}}{5}\cdot\log 2.$$

Finally, let us consider the following integrals:

$$A=\int_0^{\infty}\frac{\{b^{2y}x\}-\{b^yx\}}{y}\,dy,\ B=\int_0^{\infty}\frac{\{y^2x\}-\{yx\}}{y\log y}\,dy,\ C=\int_0^{\infty}\frac{\{2yx\}-\{yx\}}{y}\,dy.$$

It seems that *A* is related to *M*(*b*). After a change of variable that makes the parameter *b* disappear, *A* = *B*, so *A* does not depend on *b*. One can prove that *B* = (log 2)/2, see here for details. What about *C*? That one is also equal to (log 2)/2, as one would expect, so *A* = *B* = *C* = (log 2)/2. Other similar integrals, known as Frullani integrals, can be found in section 10 in chapter 28.

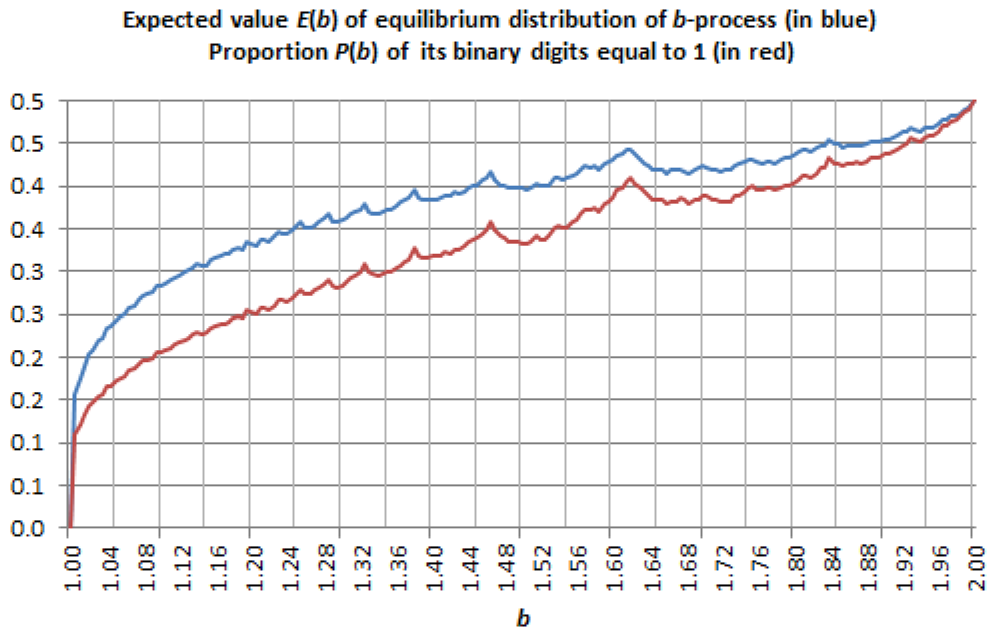
Integral A is associated with *b*-processes, which have bad seeds, and are sometimes called *universally bad averaging sequences* for that reason. Integral C is associated with a process with no bad seed, defined at the beginning of section 4, see *Perfect stochastic processes* in this appendix. The integrals *A* and *C* are associated with time-continuous versions of these processes, respectively for *b*-processes and perfect processes.

(b) Some expected values, distribution of binary digits

Let *E*(*b*) be the expected value of the equilibrium distribution of a *b*-process. What is the average value of *E*(*b*) between 1 and 2? That is,

$$\int_1^2 E(b)db = \frac{1}{\log 2}\int_1^2 M(b)db = \;??$$

The value is around 0.38. See references at the end of this section, for a tentative solution. Interestingly, the exact value of *E*(*b*) is not even known for most *b*'s. The figure below shows *E*(*b*), as well as the proportion *P*(*b*) of digits equal to 1 for a *b*-process. The largest peak takes place at $b = (1 + 5^{1/2})/2$, the golden ratio. The case $b = 2$ corresponds to the standard base-2 numeration system. The *n*-th digit of the seed $x = x_1$ is INT($2x_n$), and it is equal to either 0 or 1 depending on *x*. It is assumed that *x* is a good seed in base *b*, thus *P*(*b*) and *E*(*b*) do not depend on *x*. Also, *b* is in]1, 2].



(c) References

- StackExhange: Computation of *B*
- StackExhange: Average value of *M*(*b*) on [1, 2]
- Research paper: Arithmetics on number systems with irrational bases, by P. Ambroz, *et al*.
- Wikipedia: Golden ration numeration system

5. Appendix

Here we dive into more technical details, regarding four problems discussed in the article.

5.1. Computing the auto-correlation at equilibrium

We consider a *b*-process where *b* is an integer, so the equilibrium distribution is "known" to be uniform on [0, 1]. This fact has been taken for granted probably for more than a thousand years (and that's why people believe that the digits of π and other mathematical constants, are uniformly distributed), but it would be nice (and easy) to prove it, if the seed is a good seed. This is left as an exercise. It is not true usually if the seed is a bad seed. π is believed to be a good seed, but no one has ever managed to prove it: it is one of the biggest mathematical challenges of all times.

Note that at equilibrium, both *X* and *g*(X) have the same distribution, so their mean and variance are identical. So if *b* is an integer and the seed x_1 is a good seed, the only challenge in the auto-correlation formula mentioned in the first section, is the computation of *E*[*Xg*(*X*)].

We have:

$$E(X\cdot\lfloor bX\rfloor)=\sum_{k=0}^{b-1}E(kX\mid\lfloor bX\rfloor=k)P(\lfloor bX\rfloor=k)=\frac{1}{b}\sum_{k=0}^{b-1}k\cdot E\Big(X\mid X\in\Big[\frac{k}{b},\frac{k+1}{b}\Big[\Big)$$

$$=\frac{1}{b}\sum_{k=0}^{b-1}k\cdot\frac{k+\frac{1}{2}}{b}=\frac{1}{b^2}\Big(\sum_{k=0}^{b-1}k^2+\frac{1}{2}\sum_{k=0}^{b-1}k\Big)=\frac{b-1}{b}\cdot\frac{4b+1}{12}.$$

By definition, *g*(*X*) = *bX* - INT(*bX*). Thus,

$$E(Xg(X))=bE(X^2)-E(X\lfloor bX\rfloor)=\frac{b}{3}-\frac{b-1}{b}\cdot\frac{4b+1}{12}=\frac{1}{4}+\frac{1}{12b}.$$

Combined with the fact that *E*(*X*) = *E*(*g*(*X*)) = 1/2 and Var(*X*) = Var(*g*(*X*)) = 1/12, as the equilibrium distribution is uniform on [0, 1], we obtain the final result: the correlation between *X* and *g*(*X*), that is, the theoretical auto-correlation between two successive values of x_n, is equal to 1/*b*. It is easy to check this result by computing the estimated value of the lag-1 auto-correlation on x_1, ..., x_n, with *n* = 1,000,000: this test provides a very good approximation.

5.2. Proof of the first fundamental theorem

Here *b* is in]1, 2]. If *Y* = *g*(*X*), we have, for *y* in [0, 1]:

$$\begin{aligned}P(Y<y) &= P(bx-\lfloor bX\rfloor<y)=P(bX<y \text{ and } \lfloor bX\rfloor=0)+P(bX-1<y \text{ and } \lfloor bX\rfloor=1)\\ &= P(X<y/b \text{ and } X<1/b)+P(X<(1+y)/b \text{ and } 1/b\leq X\leq 1)\\ &= P(X<y/b)+P(1/b\leq X<\min(1,(1+y)/b)).\end{aligned}$$

Thus, using the notation *F* for the probability distribution function (and *f* for the density) we have:

$$\text{If } (1+y)/b>1, \text{ then } F_Y(y)=F_X(y/b)+1-F_X(1/b), \text{ else}$$
$$F_Y(y)=F_X(y/b)+F_X((1+y)/b)-F_X(1/b).$$

Taking the derivative with respect to *y* on both sides of the equation, we obtain the final result:

$$f_Y(y)=\frac{1}{b}f_X(y/b) \text{ if } y>b-1,$$

$$f_Y(y)=\frac{1}{b}\Big(f_X(y/b)+f_X((1+y)/b)\Big) \text{ otherwise.}$$

5.3. How to find the exact equilibrium distribution

We focus on *b*-processes with *b* in]1, 2]. Finding the equilibrium distribution (actually, its density) is accomplished in two steps.

First, compute *P*(*b*) for all the polynomials *P* mentioned in the second fundamental theorem. Any value of *P*(*b*) between 0 and 1 corresponds to a potential change point. By looking at the empirical equilibrium density, computed on 100,000 values of { x_n }, you can find the approximate value of the major change points: they correspond to points of discontinuity in the density function. For instance, if *b* = 1.4656... (the supergolden ratio), there is clearly a change point around 0.46, see the first picture in section 2. There is another one around 0.68. The exact values c_1 and c_2 of the two change points are derived from two of these polynomials:

$$c_1 = b - 1, \text{ and } c_2 = b^2 - b,$$

because no other polynomial (in the small list that you have to check) gets so close to 0.46 and 0.68 respectively, when evaluated at *b*.

Then, once the change points are identified, take three different values -- one in each interval -- for instance

0.25 in $S(1) = [0, c_1[$, 0.50 in $S(2) = [c_1, c_2[$, and 0.75 in $S(3) = [c_2, 1]$.

This assumes that you have three intervals, but you can easily generalize if you have more. Now apply the first fundamental theorem with *y* = 0.25, *y* = 0.50, and *y* = 0.75 respectively. You get:

$$f_Y(0.25) = (f_X(0.853...) + f_X(0.171...))/b, \text{ as } 0.25 < b - 1,$$
$$f_Y(0.50) = (f_X(0.341...))/b, \text{ as } 0.50 \geq b - 1,$$
$$f_Y(0.75) = (f_X(0.512...))/b, \text{ as } 0.75 \geq b - 1.$$

Note that

- 0.853... = (1 + 0.25)/*b*, and it is in *S*(3), and 0.171... = 0.25/*b*, and it is in *S*(1)
- 0.341... = 0.50/*b*, and it is in *S*(1)
- 0.512... = 0.75/*b*, and it is in *S*(2)

At equilibrium, the density functions of *X* and *Y* are identical. Thus, if d_1, d_2 and d_3 denote the density values in each of the 3 intervals, we end up with the following linear system, where d_1, d_2 and d_3 are the unknowns:

$$d_1 = (d_3 + d_1)/b$$
$$d_2 = d_1/b$$
$$d_3 = d_2/b$$

It has an infinite number of solutions, and you need to add one constraint, the fact that the total density sums to 1, to be able to solve it. That constraint is

$$d_1c_1 + d_2(c_2 - c_1) + d_3(1 - c_2) = 1,$$

that is,

$$d_1(b - 1) + d_2(b^2 - 2b + 1) + d_3(1 - b^2 + b) = 1.$$

Finally, the solution, in this case, is

$$d_1 = b^2/(2b^3 - 4b^2 + 2b + 1),\ d_2 = d_1/b, \text{ and } d_3 = d_1/b^2.$$

If you cannot easily determine which polynomials yield the change points or you want to automate the method, you may as well try any two combinations of the potential polynomials (assuming you have two change points), and for each pair of polynomials (that that is, for each pair of change point candidates) solve a similar linear system. You then plug the tentative equilibrium densities obtained for each pair of polynomials, into the formula in the first fundamental theorem. Only one of them will satisfy the fact that *X* and *Y* have the same density everywhere on [0, 1], and that is the solution.

5.4. Perfect process with no auto-correlation

We compute the covariance *E*(*X*, *g*(*X*)) between successive observations x_n in a perfect process. These processes were introduced in subsection 4.1 and are defined by $g(x) = x + b - \text{INT}(x + b)$. We show that the covariance between *X* and *g*(*X*), and thus the lag-1 auto-correlation, is zero if and only if the fractional part of *b* is equal to $(3 + 3^{1/2})/6$ or $(3 - 3^{1/2})/6$. Also, the lag-1 auto-correlation is minimum, and equal to -1/2, when the fractional part of *b* tends to 1/2.

Here, *b* is any positive irrational number. The seed x_1 can be any real number in [0, 1], even $x_1 = 0$, since perfect processes don't have bad seeds. Also, as usual and by definition, $x_{n+1} = g(x_n)$.

To prove the result, we start with the fact that since *X* is in [0, 1], we have:

INT(*X* + *b*) = INT(*b*) if *X* < INT(*b* + 1) - *b*, otherwise INT(*X* + *b*) = INT(*b* + 1).

The equilibrium distribution being uniform on [0, 1], and using the brackets to represent the INT function, we thus have:

$$E(X\cdot\lfloor X+b\rfloor)=\int_0^1 x\lfloor x+b\rfloor f_X(x)dx=\int_0^{\lfloor b+1\rfloor-b} x\lfloor b\rfloor dx\ +\int_{\lfloor b+1\rfloor-b}^1 x\lfloor b+1\rfloor dx$$
$$=\lfloor b\rfloor\int_0^{\lfloor b+1\rfloor-b} xdx\ +\lfloor b+1\rfloor\int_{\lfloor b+1\rfloor-b}^1 xdx$$
$$=\frac{1}{2}\Big(\lfloor b+1\rfloor-(\lfloor b+1\rfloor-b)^2\Big).$$

At equilibrium, we have *E*(*X*) = *E*(*g*(*X*)) = 1/2, and $E(X^2) = 1/3$ since the distribution is uniform. Thus,

$$\text{Cov}(X,g(X))= E(X\cdot(X+b-\lfloor X+b\rfloor))=E(X^2)+bE(X)-E(X\lfloor X+b\rfloor)-\frac{1}{4}$$
$$=\frac{1}{3}+\frac{b}{2}-E(X\lfloor X+b\rfloor)-\frac{1}{4}$$
$$=\frac{1}{2}\Big(b^2+(1-2\lfloor b+1\rfloor)b+\lfloor b+1\rfloor^2-\lfloor b+1\rfloor+\frac{1}{6}\Big).$$

With the notation *k* = INT(*b*), *A* = 1, *B* = 1 - 2(*k* + 1), and $C = (k + 1)^2 - (k + 1) + 1/6$, finding the values of *b* that yield Cov(*X*, *g*(*X*)) = 0, consists in solving the quadratic equation $Ab^2 + Bb + C = 0$. There may be a solution for each *k* = 0, 1, 2, and so on. Note that the discriminant of the quadratic equation does not depend on *k*:

$$B^2-4AC=(1-2(k+1))^2-4(k+1)^2+4(k+1)-\frac{4}{6}=\frac{1}{3}.$$

Thus the solutions are $b = k + (3 + 3^{1/2})/6$ and $b = k + (3 - 3^{1/2})/6$, for *k* = 0, 1, and so on. Note that *b* must be irrational, otherwise the equilibrium distribution may not exist or may not be uniform.

Appendix C. Cheat Sheet

Here is all you need to get started from scratch, including sample projects, data sets, and references.

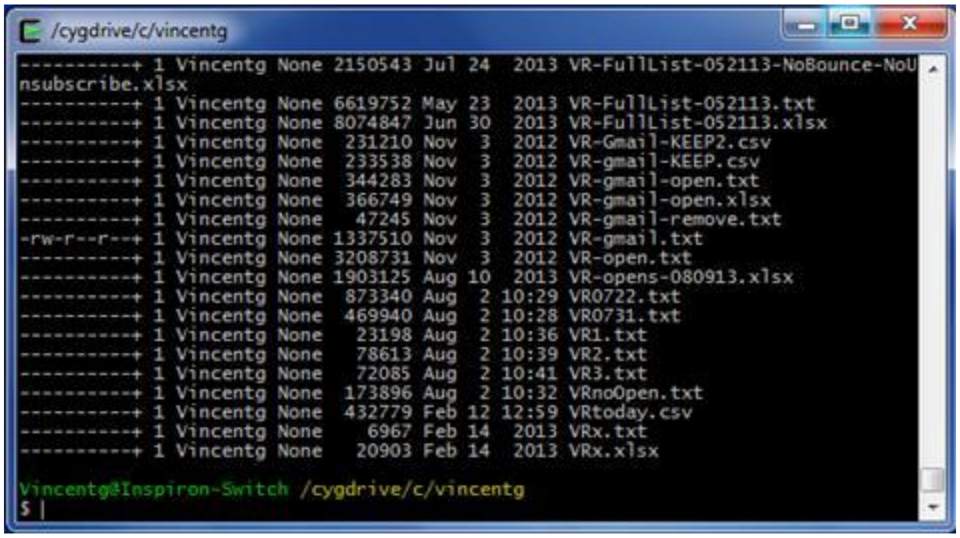
1. Hardware

A laptop is the ideal device. I've been using Windows laptops for years, and I always installed a Linux layer (acting as an operating system on top of Windows), known as Cygwin. This way, you get the benefits of having Windows (Excel, Word, compatibility with clients and employers, many apps such as FileZilla) together with the flexibility and pleasure of working with Linux. Note that Linux is a particular version of UNIX. So the first recommended step (to start your data science journey) is to get a modern Windows laptop and install Cygwin.

Even if you work heavily on the cloud (AWS, or in my case, access to a few remote servers mostly to store data, receive data from clients and backups), your laptop is you core device to connect to all external services (via the Internet). Don't forget to do regular backups of important files, using services such as DropBox.

2. Linux environment on Windows laptop

Once you installed Cygwin, you can type commands or execute programs in the Cygwin console. Here's how the console looks like on my laptop:



```
----------+ 1 Vincentg None 2150543 Jul 24  2013 VR-FullList-052113-NoBounce-NoU
nsubscribe.xlsx
----------+ 1 Vincentg None 6619752 May 23  2013 VR-FullList-052113.txt
----------+ 1 Vincentg None 8074847 Jun 30  2013 VR-FullList-052113.xlsx
----------+ 1 Vincentg None  231210 Nov  3  2012 VR-Gmail-KEEP2.csv
----------+ 1 Vincentg None  233538 Nov  3  2012 VR-gmail-KEEP.csv
----------+ 1 Vincentg None  344283 Nov  3  2012 VR-gmail-open.txt
----------+ 1 Vincentg None  366749 Nov  3  2012 VR-gmail-open.xlsx
----------+ 1 Vincentg None   47245 Nov  3  2012 VR-gmail-remove.txt
-rw-r--r--+ 1 Vincentg None 1337510 Nov  3  2012 VR-gmail.txt
----------+ 1 Vincentg None 3208731 Nov  3  2012 VR-open.txt
----------+ 1 Vincentg None 1903125 Aug 10  2013 VR-opens-080913.xlsx
----------+ 1 Vincentg None  873340 Aug  2 10:29 VR0722.txt
----------+ 1 Vincentg None  469940 Aug  2 10:28 VR0731.txt
----------+ 1 Vincentg None   23198 Aug  2 10:36 VR1.txt
----------+ 1 Vincentg None   78613 Aug  2 10:39 VR2.txt
----------+ 1 Vincentg None   72085 Aug  2 10:41 VR3.txt
----------+ 1 Vincentg None  173896 Aug  2 10:32 VRnoOpen.txt
----------+ 1 Vincentg None  432779 Feb 12 12:59 VRtoday.csv
----------+ 1 Vincentg None    6967 Feb 14  2013 VRx.txt
----------+ 1 Vincentg None   20903 Feb 14  2013 VRx.xlsx

Vincentg@Inspiron-Switch /cygdrive/c/vincentg
$ |
```

Figure 1: *Cygwin (Linux) console on Windows laptop*

You can open multiple Cygwin windows on your screen(s).

To connect to an external server for file transfers, I use the Windows FileZilla freeware rather than the command-line ftp offered by Cygwin. If you need full privileges on the remote machine, use Putty instead (for Telnet/SSH sessions).
You can run commands in the background using the & operator. For instance,

```
$ notepad VR3.txt &
```

will launch Notepad (the standard Windows text editor) from the Cygwin console, into another window, and open the file `VR3.txt` located in your local directory (if this file exists in that directory). Note the `$` symbol preceding any command (see Figure 1). In addition, the console also displays the username (`Vincent@Inspiron-Switch` in my case) as well as the directory I'm in `(/cygdrive/c/vincentg/` in Linux, corresponding to the `c://vincentg/` pathname under windows).

Basic operations:

- Changing directory is performed with the command cd (examples: `cd subfolder/sub-subfolder`, `cd ..` to go one level above, `cd .` to go to your home directory)
- Listing content of directory is done with command `ls -l` (note that `-l` is a command argument used to specify that want a full, detailed listing; without this option, the listing shown in Figure 1 would be far less detailed).
- If you don't know your local directory, type in the command pwd, it will tell you your location (path)

So far you've learned the following Linux concepts: command line and symbol `$` (sometimes replaced by `>` depending on the Linux version), operator `&` (for background processing), paths, commands `cd`, `pwd`, and `ls`, command options (`-l` for `ls`) and shortcuts (`.` and `..` for the `cd` command).

A few more things about files

Files have an extension that indicates what kind of file it is (text, image,spreadsheet) and what software can open and process them. In Figure 1, `VR3.txt` has the `.txt` extension, meaning it's a text file - the most widespread type of data file. There are two types of files: binary (used by various programs; compressed/encrypted format) and text (can be processed easily by any program or editor). It is important to know the distinction when doing FTP file transfers (FTP clients allow you to specify the type of file, though it's automated and transparent to the user with FileZilla).

Other extensions include

- `.csv` (comma-separated text file that you can open with Excel or Notepad; it can have more than 1 million rows),
- `.xlsx` (Excel files limited to 1 million rows, this is a binary file),
- `.gz` (compressed files, thus binary files),

- `.png` (best image format, other image formats include `.gif`, `.jpg`, `.jpeg`, and `.bmp`; these are binary files),
- `.docx` (Word documents; binary),
- `.html` (text files representing source code of a web page),
- `.sql` (text file used to store an SQL query, used as input for some database clients such as Brio),
- `.php` (PHP code, text format),
- `.pl` (Perl code, text format),
- `.js` (Javascript code, text format),
- `.r` (R code, text format),
- `.py` (Python code, text format),
- `.c` (C code, text format),
- `.exe` (Windows executable),
- `.xml` (XML, text format for name-value pairs)

Files are not stored exactly the same way in Windows and UNIX. Also, some systems use UNICODE for file encoding, which takes much more space but allow you e.g. to work with Chinese characters (stored using two bytes per character). When processing such a file (they are rather rare fortunately), you'll first need to clean it and standardize it to traditional ASCII (one byte = one character).

Finally, the best text format that you can use is tab separated: each column or field is separated by a TAB, an invisible char represented by `\t` in some programming languages. The reason is that some fields contain commas, and thus using csv (comma-separated text files) results in broken fields and data that looks like garbage, and is hard to process (requiring a laborious cleaning step first, or talking to your client to receive tab-separated format instead).

When processing data, the first step is to produce a data dictionary (see section 8 in chapter 25). It is easily done using a scripting language.

File management

Filenames should be designed carefully (no space or special char in a filename), especially when you have thousands or millions of files across thousands of directories and sub-directories, and across dozens of servers (the cloud). One of the two core components of Hadoop is actually its file management system, known as HDFS (the other component being the distributed *Map-Reduce* architecture to process tasks).
It's always a good idea to always have a time stamp embedded into the file name, representing the creation date. Note that in Figure 1, the files all start with VR, an abbreviation for Vertical Response, as these files are coming or related to our email service provider, called Vertical Response. File names should be very detailed: keep in mind that sooner rather than later, you might run scripts to process millions of them. Without proper naming conventions, this task will be impossible.

A final word, if you look at Figure 1, the first column indicates who can read (`r`), re-write (`w`) or execute (`x`) these files, besides me. It's never been an issue on Windows for me, but on a true UNIX operating system (not Cygwin), you might want to set the right protections: for example Perl scripts (despite being text) must be set to Executable, with the UNIX command `chmod 755 filename.pl`, where `filename.pl` is your Perl script. File protections (and locks) are important for organizations where files can be shared by many users, sometimes simultaneously.

3. Basic UNIX commands

You don't need to spend hours learning UNIX and buy 800-page books on the subject. The following commands will get you started, once you have your Cygwin console:

- `cd`, `pwd`, `ls` (see section 2)
- `tail -100, head -150` to extract the last 100 or first 150 rows of a file
- `cp`, `mv`, `mkdir`, `rmdir` respectively copy a file to another location, rename a file, create a new directory or delete a directory (you need to erase all files first)
- sort, uniq respectively sort a file and remove duplicate entries (you can sort alphabetically or numerically depending on the option; default is alphabetical order)
- `gzip`: compress/un-compress files
- `wc`: count number of rows and words in a text file
- `grep`: identify all rows containing a specific string in a text file (it helps to be familiar with regular expression)
- `cat`: display content of text file on your screen
- `chmod`: change file protections, see section 2
- `history`: lists the last commands you used, as it is very common to re-use the same commands all the time.
- `cron`, `crontab`: to automatically schedule tasks (running an executable once a day)

Operators include `>` (to save output to a new file), `>>` to append output to an existing file, `|` (the pipe operator, see examples), `&` (see section 2, used for background or batch mode when executing a command), `*` (see examples) and `!`(see examples.)

Examples

- `sort filename.txt | uni -c > results.txt` (sort `filename.txt` alphabetically - not numerically - then remove duplicates, and for each final entry count number of duplicates with option `-c`; store results in `results.txt`)
- `rm -i test*.txt` (remove all files starting with test and with extension `.txt`; the extension `-i` is to request manual confirmation before each file gets deleted)
- `grep 'abc' test.txt | wc` (extract all rows containing ‘abc’ in `test.txt`, then count these rows with `wc`)
- `!545` (run command #545, after you run the command history to get the lists of previously entered commands)

Check out this reference for more details (exact syntax and options).

Miscellaneous

Shell scripts (or batch files) are small programs that execute a list of commands, and can be run in batch mode. For regular expressions, see section 4.

4. Scripting languages

You can get started in data science wth just a few Unix commands, a tool for statistical analyses such as R (unless you write your own algorithms to get more robust and simple tools) and a scripting programming language such as Perl or Python. Python (together with Pandas libraries) is the most popular language for data science. Python and machine learning resources are provided later in this article. This article is a good introduction on Python for data science. This reference has tons of resources about Python for data science.

Here I describe fundamental features of Perl, but they apply to all scripting languages. You can download Perl from ActiveState. Numerous short programs (Perl, but also R), easy to read and understand, can be found here. Perl scripts are text files with a `.pl` extension (say `myprogram.pl`) that you can execute in the Cygwin console with the command line `perl myprogram.pl` once you have installed Perl on your laptop.

Core elements of scripting languages

Some basic stuff that is used in pretty much any programs include

- **Hash tables** are lists of name-value pairs, where insertion or deletion of an element is very fast. They can be descibed as arrays indexed by strings, and constitute a powerful, fundamental data structure. They can be used to produce efficient joins. See data dictionary (section 8 in chapter 25) for a simple illustration. Hash tables store data using a syntax such as `$myhash{"Vincent Granville|Data Scientist"} = "yes";` In this case the index is bi-dimensional and is made up of the name and job title; the value is "yes" or "no". If the name or job title is not in your data, no entry is created (that's why this data structure produces efficient joins). See also chapter 5 on feature selection, for a more sophisticated application.

- **Associative arrays** are just hash tables: arrays indexed by strings rather than integers. In Perl, they are declared using `%myhash=()` while regular arrays are declared using `@myarray=()`. Memory allocation for hash tables is automated in Perl. However, you should create a variable `$myhashsize` that is incremented by 1 each time an entry is added to `%myhash` (or decremented by 1 in case of deletion). This way, you know how big your hash tables grow. If your program displays (on the screen) the exact time every 300,000 newly created hash entries, you'll have an idea when you run out of memory: at that moment, your Perl script suddenly starts running 20 times slower. When this happens, it's time to think about optimization using Hadoop or Map-Reduce (distributed architecture.)

- String processing and regular expressions: the sample code below contains basic strings substitution including special characters `(\n`, `\:`). Many substitutions can be performed in just one tiny bit of code using **regular expressions**, click here or here for details. One of the most widespread operations is to split a text `$text` into elements stored in an array `@myarray`; the syntax is `@myarray = split(/\t/,$text)`. Here we assume that text elements are separated by TABs (the special character `\t`). The number of text elements is stored in the variable `$#myarray`.

The easiest way to learn how to code is to look at simple, well written sample programs of increasing complexity, and become an expert in Google search to find solutions to coding questions - many answers can be found on StackOverflow. I have learned R, SQL C, C++ and Perl that way, without attending any training. If you need training, check out this list of courses. The following are good examples of code to get you started.

Sample scripts to get you started

Here is some sample code.

- Code to run SQL queries 10 times faster than Brio, Toad etc.
- Source code for our Big Data keyword correlation API (Perl API)
- Source code to compute N-grams (NLP)
- Simulation of stochastic processes
- Simple source code to simulate nice cluster structures
- Ridge regression with bootstrap
- Basic web crawler
- Model-free confidence intervals (see section 2, subsection "Perl code")

Below is a simple script that performs automated dns lookups to extract domain names associated with IP addresses. The input file is a list of IP addresses (`ips.txt`) and the output file is a text file `outip.txt` with two fields, tab-separated: IP address and domain name. A temporary file `titi.txt` is created each time we call the external Cygwin command `'nslookup'`. Note that `$` is used for variables. There's some basic string processing here, for instance: `$ip=~s/\n//g` substitutes each carriage return / line feed (special character `\n`) by nothing (empty) in the variable `$ip`. Note that the symbol `#` means that what follows (in the line in question) is a comment, not code.

```
`rm titi.txt`;
# $ip="107.2.111.109";
```

```
open(IN,"<ips.txt");
open (OUT,">outip.txt");
while ($lu=<IN>) {
  $ip=$lu;
  $n++;
  $ip=~s/\n//g;
  if ($ip eq "") { $ip="na"; }
  `nslookup $ip | grep Name > titi.txt`;
  open(TMP,"<titi.txt");
  $x="n/a";
  while ($i=<TMP>) {
    $n++;
    $i=~s/\n//g;
    $i=~s/Name\://g;
    $x=$i;
  }
  close(TMP);
  print OUT "$ip\t$x\n";
  print "$n> $ip | $x\n";
  sleep(0);
}
close(OUT);
close(IN);
```

Now, you can download logfiles (see free data sets in section 6), extract IP addresses and traffic statistics per IP address, and run the above script (using a distributed architecture, with 20 copies of your script running on your laptop) to extract domain names attached to IP addresses. Then you can write a program to map each IP address to an IP category using the technique described in my article Internet Topology Mapping. And finally, sell or license the final data to clients.

A few more useful concepts:

- Functions in Perl are declared using the `subroutine` reserved keyword. A few examples are found in the sample scripts. Learn how to pass an argument that is a variable, an array or an hash table. Subroutines can return more than one value. Use of global variables is discouraged, but with proper care (naming conventions), you can do it without risks.
- You can write programs that accept command-line arguments. Google 'command-line arguments' for details.
- Libraries (home-made or external) require an inclusion directive, such as require `LWP::UserAgent;` in the web robot sample code (see link above) that uses the LWP library. If a library is not available in your Perl distribution, you can download and add it using the ppm command, or even manually (see my Wiley book page 138, where I discuss how to manually install the library `Permutor.pl`).
- Scripts can be automatically run according to a pre-established schedule, say once a day. Google 'cron jobs' for details, and check this article for running cron jobs on Cygwin.

Exercise

Write a script that accesses all the text files on your laptop using two steps:

- recursively using the `ls-l > dir.txt` Cygwin command from within Perl to create directory listings (one for each folder / subfolder) saved as text files and named `dir.txt`
- accessing each text file from each of these automatically created directory listings dir.txt in each directory

Then count the number of occurrences for each word (broken down per file creation year) across these files, using a hash table. Purpose: identify keyword trends in your data.

5. Python, R, Hadoop, SQL, DataViz

R is a popular language to perform statistical analyses or nice graphics. I would not use it for black-box applications. Large applications such as text clustering involving 20 million keywords are performed in Python or Perl, known as scripting languages. Python libraries for data analysis and machine learning are widely available and discussed in a few O'Reilly books: they offer an alternative to R, for big data processing. Note that R does have an extensive collection of sophisticated statistical functions, too many in my opinion. Finally, R is currently used for exploratory data analysis rather than production-mode development. For more info, read R versus SAS versus Python or R versus Python.

R programming

You can download the open-source R package from The R Project. Installation and running R programs via the GUI, on a Windows laptop, is straightforward. Memory limitations can be bypassed using multiple copies of R on multiples machines, some R packages, or using RHadoop (R + Hadoop). R programs are text files with an `.r` extension. Useful links:

- Producing videos with R
- R libraries
- R cheat sheets
- Sample R code

Also, see here for more references.

Python programming

Check out these articles about Python for data science, the preferred full-fledged programming language for data scientists. Sample code, cheat sheets, and machine learning libraries for Python, can be found on GitHub.com and in the following articles:

- Python cheat sheets
- Python libraries
- Sample Python code

Hadoop

Hadoop is a file management system used to perform tasks in a distributed environment, across multiple servers if necessary, by spitting files into sub-files, performing the analysis on each sub-file separately, and summarizing the results (by collecting the various outputs associated with each file, and putting it together). This environment uses redundancy to easily and transparently recover from server crashes. It works well for web crawling projects, even for sorting, but not so much for graph databases or real time data (except as a back-end platform). For an example, read this article. For an example of what Map-Reduce (the distributed architecture supporting Hadoop) can't do, follow this link.

SQL

Finally, don't forget that SQL is still a widely used language. Learn at least the basics, including to joining multiple tables efficiently, and playing with indexes and keys. The book SQL Essentials is a good starting point. Also, search this PDF document for the keyword *fuzzy joins*. NoSQL databases also exist, in particular graph databases.

Excel

Excel has advanced functions such as Linest (linear regression), Vlookup, percentiles, rank statistics, random numbers, or index (indirect cell referencing.) A large collection of Excel spreadsheets featuring advanced machine learning techniques can be found as attachments in the articles posted here or within this book (search for the keyword *Excel* or *spreadsheet* in this PDF document.) In particular:

- Advanced Machine Learning with Basic Excel (chapter 3)
- Black box confidence intervals - See section 2, Source Code
- Model-free confidence intervals

Also, a list of articles about data science with Excel can be found here.

Visualization

Many visualizations can be performed with R (see section on R in this article), Excel, or Python libraries. Specific types of charts (graphs or decision trees) require special functions. The most popular software is Tableau. Birt (by Accenture) is popular for dashboards and Visio for diagrams. Most tools allow you to produce maps, scatterplots and various types of visualizations. For a reference, follow this link or search for visualization cheat sheet. See also here (dataviz with R.) Also, search the web to learn how to avoid creating bad charts.

6. Machine Learning

To understand the difference between machine learning and data science, read this article. A large list of machine learning references can be found here. It covers the following domains:

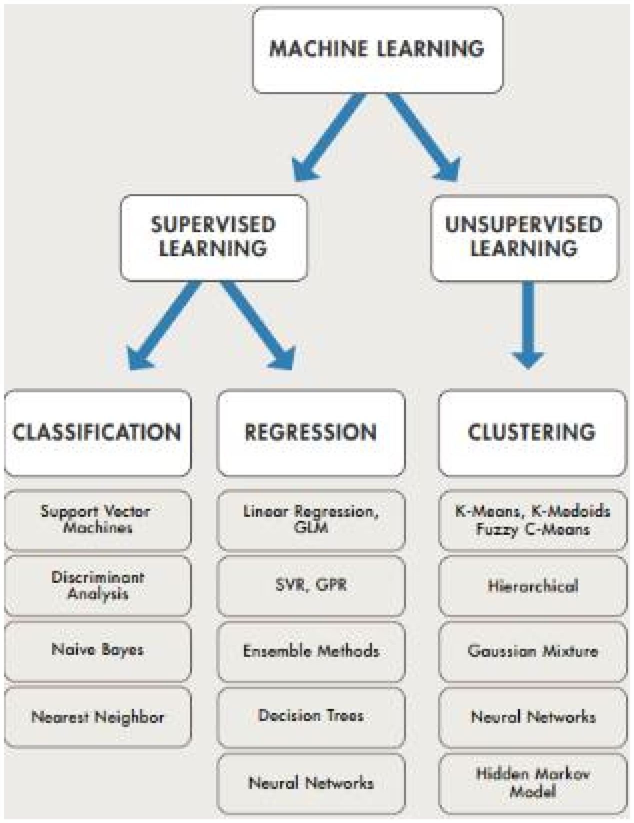
- Support Vector Machines
- Clustering
- Dimensionality Reduction
- Anomaly Detection
- Recommender Systems
- Collaborative Filtering
- Large Scale Machine Learning
- Deep Learning
- Sparse Coding

Also check out our resources section. Some of these resources include:

- 34 Great Articles and Tutorials on Clustering
- 22 Great Articles and Tutorials on Classification Methods
- 13 Great Articles and Tutorials about Correlation
- 26 Great Articles and Tutorials about Regression Analysis
- 15 Great Articles About Decision Trees
- 27 Great Resources About Logistic Regression
- Four Great Pictures Illustrating Machine Learning Concepts
- 11 Great Hadoop, Spark and Map-Reduce Articles
- 20 Cheat Sheets: Python, ML, Data Science, R, and More
- 25 Great Articles About SQL and NoSQL
- 15 Great Articles about Bayesian Methods and Networks
- 22 Great Articles About Neural Networks
- 21 Great Articles and Tutorials on Time Series
- 15 Deep Learning Tutorials
- 11 Great Articles About Natural Language Processing (NLP)
- Statistical Concepts Explained in Simple English
- Machine Learning Concepts Explained in One Picture

Algorithms

The picture below summarizes the main types of algorithms used in machine learning (ML). Source: MathWorks. See also this list and for more details, follow this link. Deep learning (neural networks with several intermediate layers) is getting more and more popular: click here for a starting point.



Getting started

A great ML cheat sheet can be found here. For a search engine focusing exclusively on ML and related topics, click here. For competitions, visit Kaggle.com. For job interview questions and answers, follow this link or download this document. Glossaries can be found here. For a great, interactive tutorial, check out Ajit Jaokar's book series, here. Many ML projects (and hopefully yours in the future) are hosted on Github.com. To ask questions, one of the best platforms is Stackexchange: see here. For more articles from the author of this book, visit this web page.

ML Applications

The following articles illustrate how ML is used in business.

- Unusual ML application: gaming technology
- 22 tips for better data science

- 21 data science systems used by Amazon to operate its business
- 40 Techniques Used by Data Scientists
- Designing better algorithms: 5 case studies
- 33 unusual applications of machine learning
- Architecture of Data Science Projects
- 24 Uses of Statistical Modeling (Part II) | (Part I)

Data sets and sample projects

Open source data sets can be found here and here. Here is another list featuring 100 data sets. KDNuggets.com also maintains a fairly comprehensive list of data sets. The following articles also feature interesting data sets:

- Source code for our Big Data keyword correlation API
- Great statistical analysis: forecasting meteorite hits
- Fast clustering algorithms for massive datasets
- 53.5 billion clicks dataset available for benchmarking and testing
- Over 5,000,000 financial, economic and social datasets
- New pattern to predict stock prices, multiplies return by factor 5
- 3.5 billion web pages
- Another large data set - 250 million data points - available for do...
- 125 Years of Public Health Data Available for Download
- From the trenches: real data science project (Google Analytics)

You can also start working on the following projects:

- Analyzing 40,000 web pages to **optimize content**: see here. Work on the data to identify the types of articles and other metrics associated with success (and how do you measure success in the first place?), such as identifying great content for our audience, forecasting articles' lifetime and page views based on subject line or category, assessing impact of re-tweets, likes, and sharing on traffic, and detecting factors impacting Google organic traffic. Also, designing a tool to identify new trends and hot keywords. See also chapter 3 for a related NLP project. Chapters 19 to 23 are also good starting points.

- **Categorization** of data scientists. Also, create a list of top 500 data scientists using public data such as Twitter, and rate them based on number of followers or better criteria. Also identify new stars and trends - note that new stars have fewer followers even though they might be more popular, as it takes time to build a list of followers. Classify top practitioners into a number of categories (unsupervised clustering) based on their expertise (identified by keywords or hashtags in their postings or LinkedIn profile). Filter out automated from real tweets. Finally, create a taxonomy of data scientists: see here for a starting point.

- **Spurious correlations** in big data, how to detect and fix it. You have n = 5,000 variables uniformly distributed on [0,1]. What is the expected number m of correlations that are above p = 0.95? Perform simulations or find theoretical solution. Try with various values of *n* (from 5,000 to 100,000) and *p* (from 0.80 to 0.99) and obtain confidence intervals for *m* (*m* is a function of *n* and *p*). Identify better indicators than correlation to measure whether two time series are really related. The purpose here is twofold: (1) to show that with big data, your strongest correlations are likely to be spurious, and (2) to identify better metrics than correlation in this context. A starting point is my article about the curse of big data, also in my Wiley book pages 41-45. Or read chapter 27 in this book.

- Perform **simulations** to assess the probability of some extreme events (useful in fraud detection problems, to detect fake or shared profiles or fake reviews). See here, also here (are there too many twin points in this dataset?) and here. Simulations are useful in pattern detection problems. For number theory applications (experimental mathematics involving chaotic sequences) - with several statistical tests being used to assess departure from randomness - check out Appendix B in this book, or in my book on stochastic processes, here.