The Honeybadger Field Guide to

Machine Learning With Ruby

By Starr Horne & Max Wilbert

Honeybadger.io - Exception, Uptime and Performance Monitoring for Ruby

In This Book

Introduction

At Honeybadger, we have a problem that is probably familiar to a lot of you. We have a lot of data.

Terabytes and terabytes of data – data about our own processes, data we collect on behalf of our customers, etc. For the most part, what we do with this data isn't very exciting. We grep it for errors, we count certain things, we perform other basic operations.

One of the main problems in building powerful applications is taking all of that raw data and translating it into knowledge that can increase our understanding of underlying trends and relationships.

We want to do more with our data, because what separates a great application from a mediocre one is the ability to make sense of data, which means looking into machine learning.

For Ruby developers, knowing the basics of machine learning can help your applications rise above the pack. We developers already have a specific skill set, so this e-book won't go into all the details of the complex math behind machine learning.

Instead, we're going to outline some of these basic ideas, so that maybe the next time you run into a situation where this sort of complex data analysis would be useful, you will know what search terms to plug into Google to help your apps.

This e-book will be an introduction to the ideas of machine learning – ideas that are used by companies like Google to take vast quantities of information and turn it into something useful, by companies like:

•Pandora to magically predict the musical preferences of their listeners
•Credit card companies to know when certain transactions are likely to be fraudulent
•Apple to implement their Siri voice recognition

This is the magic that turns data into understanding. Let's get to it – we've got a lot of ground to cover.

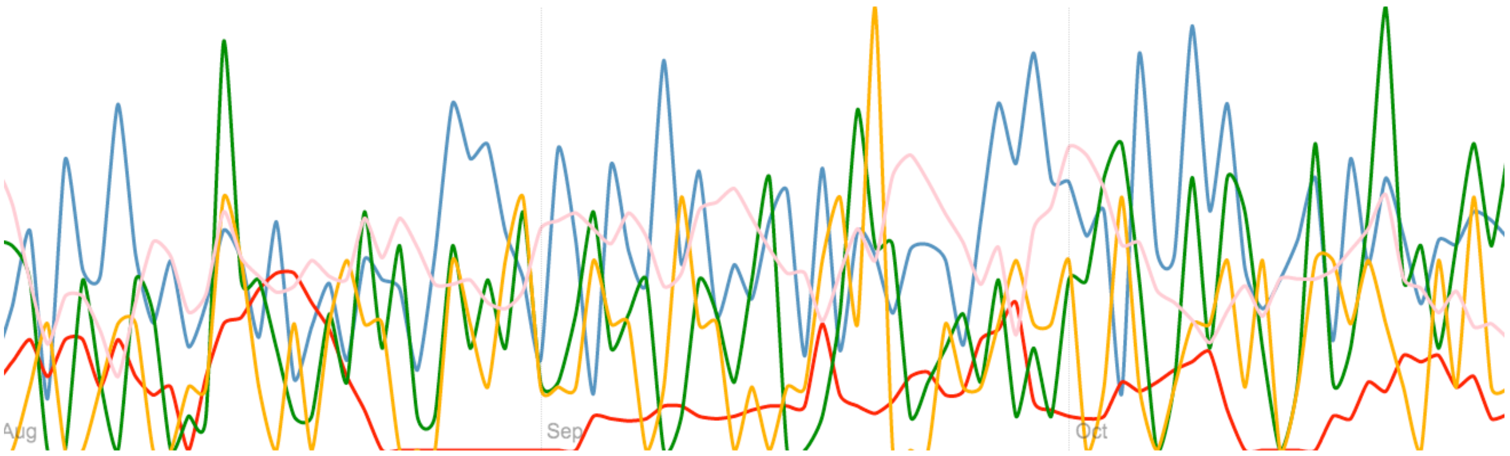
Machine Learning and Mathematics

The power of computers to understand raw data is based in mathematics. Over the last five years or so, this type of mathematics-based machine learning has taken over the web, especially on the most popular and powerful applications.

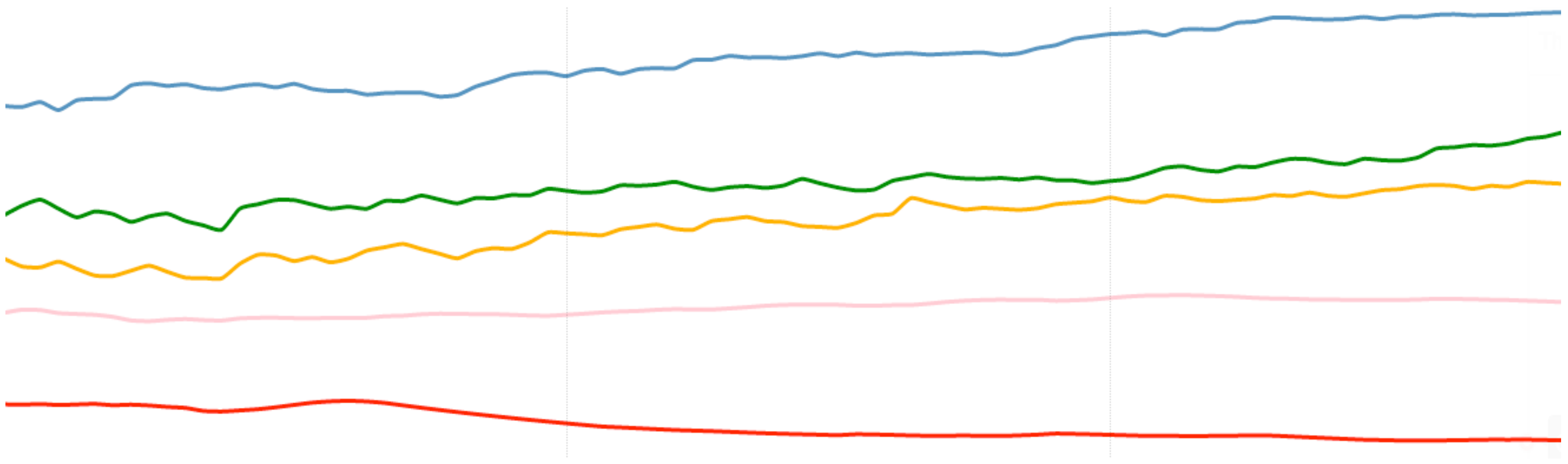
There are quite a few well-known solutions to data mining and computer learning problems that are relatively common across applications. The solutions are often complex, mathematically speaking, but are quite simple to implement once you know about them, read the basic information, and download the correct gems. Let’s get started with one of simpler operations.

LOESS for Data Smoothing

Let’s say that you have a simple data set that includes information for the amount of income your application is bringing in, as well as how many new customers are arriving and how many customers are canceling. It’s pretty simple to take this raw data and plot it, but unfortunately the resulting graph probably won’t be too useful. Unless you are a real wizard with making graphs, the data will just be too confusing to clearly observe trends.

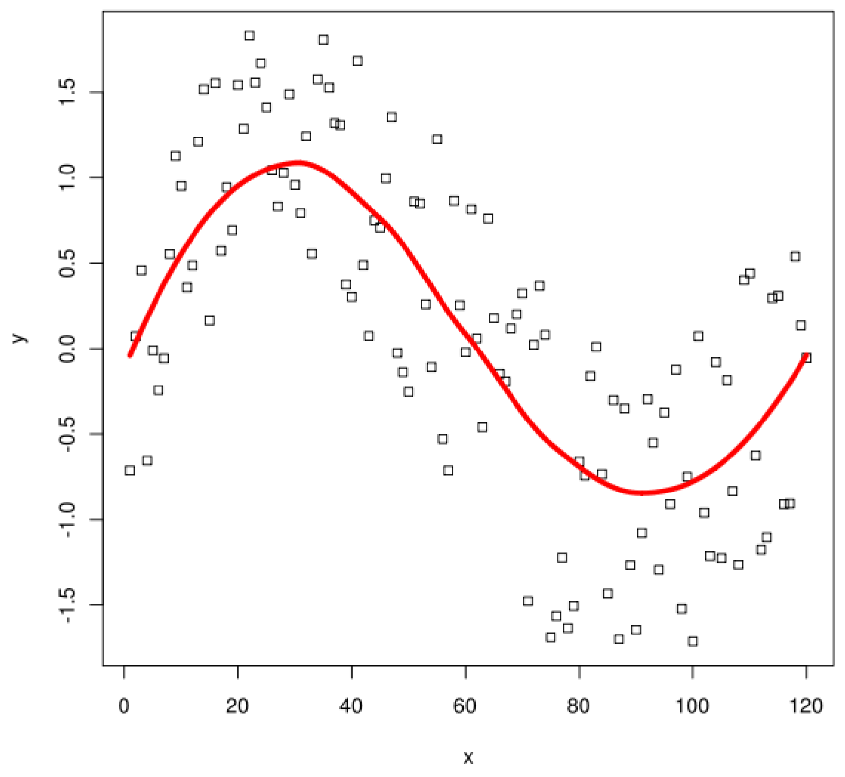


Raw
(not helpful)



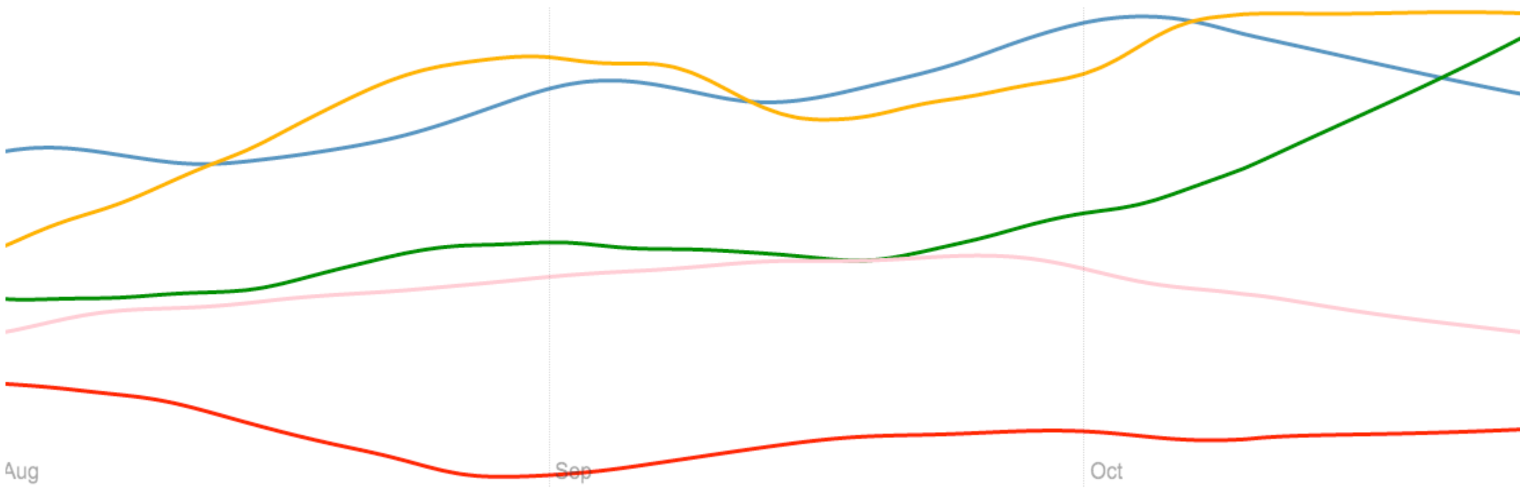
Rolling Average

The first thing that most people would do to try to make this graph more readable to humans would be apply a rolling average. With some data sets, this might work the first time around, but most times that graph isn't going to be very usable either. To make it usable, you'd have to tweak a whole bunch of variables for each data set, which is a lot of work. Most folks are too busy to deal with that.



LOESS

One great way to deal with this situation is to use a graph-fitting algorithm called “LOESS.” LOESS is pretty powerful and quite simple; you just plug your data into the equation and it will plot the graph automatically, applying the most useful regressions without any decisions from you.



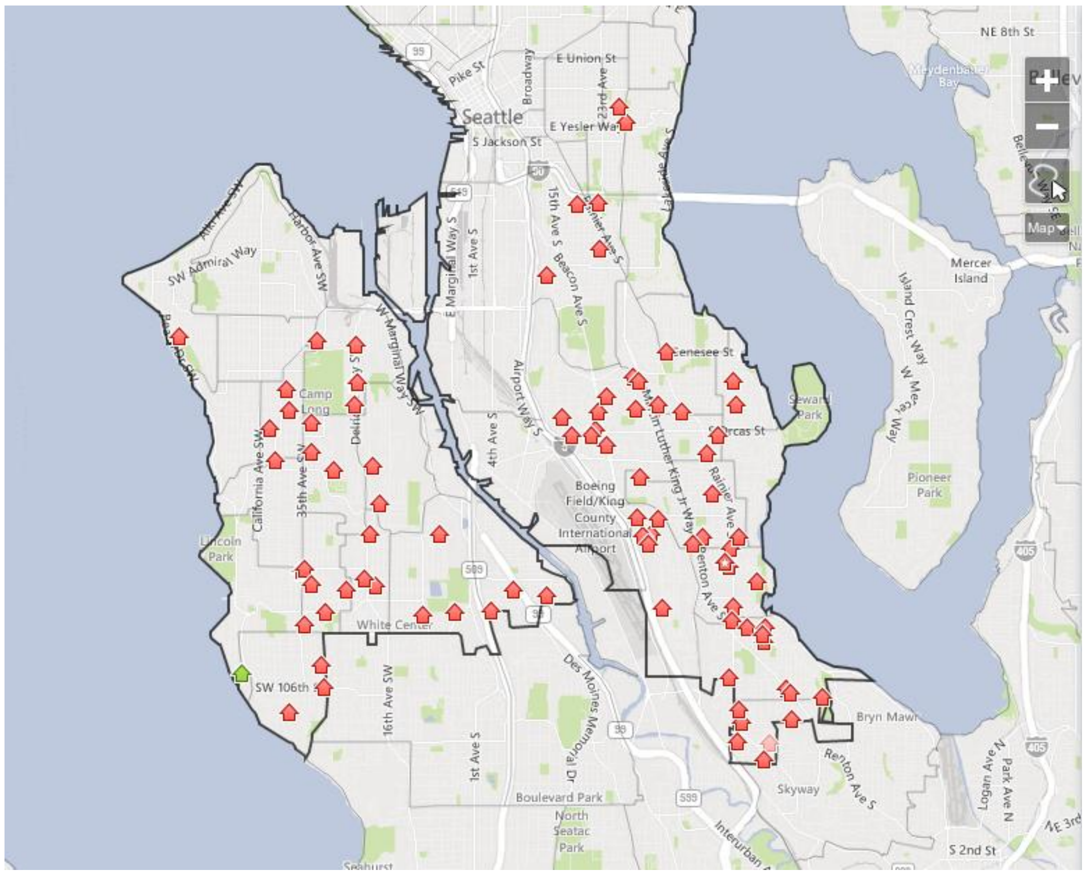
LOESS

LOESS-produced graphs clearly show inflection points and trends in your data set, without all the extra work that would otherwise be required.

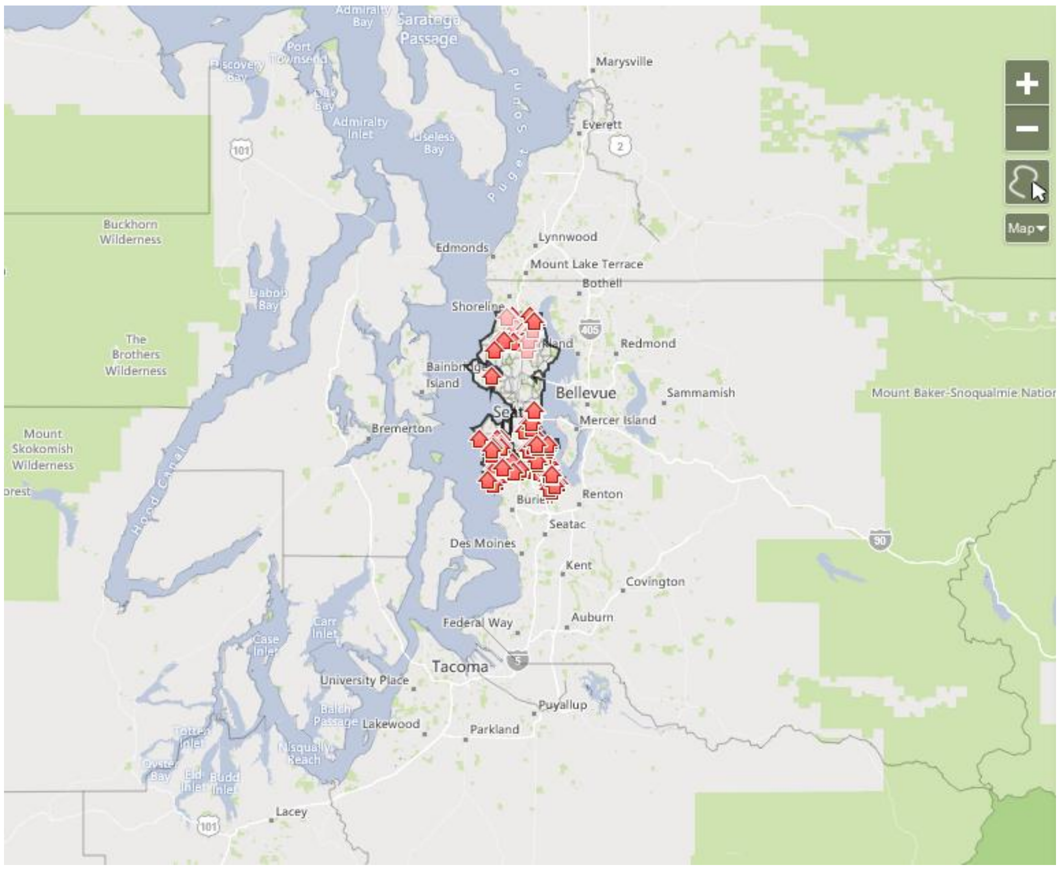
Pretty powerful, huh?

K-Means Clustering

Clustering similar data is a great way to expand the functionality of your application. One example is a real estate search engine like Zillow. Zillow allows users to search for homes with a certain number of bedrooms, bathrooms, within a price range, and more. Search results are shown as icons on the map.



If you’ve ever used a search engine like this, you’ve probably noticed that some of them have an annoying bug. As you zoom out, the scale of the map gets larger, but the icons representing each individual house stay the same size. When each icon is covering several square miles, it really reduces the functionality of the application.



K-Means clustering provides a solution to problems like this. As one of the most common clustering algorithms, it's really quite easy to explain. To use this algorithm, you first have to define a certain number of clusters. If you are still looking at the real estate map, you could, for example, define the center of each neighborhood as a cluster point, then tell each house to associate itself with the nearest cluster point.

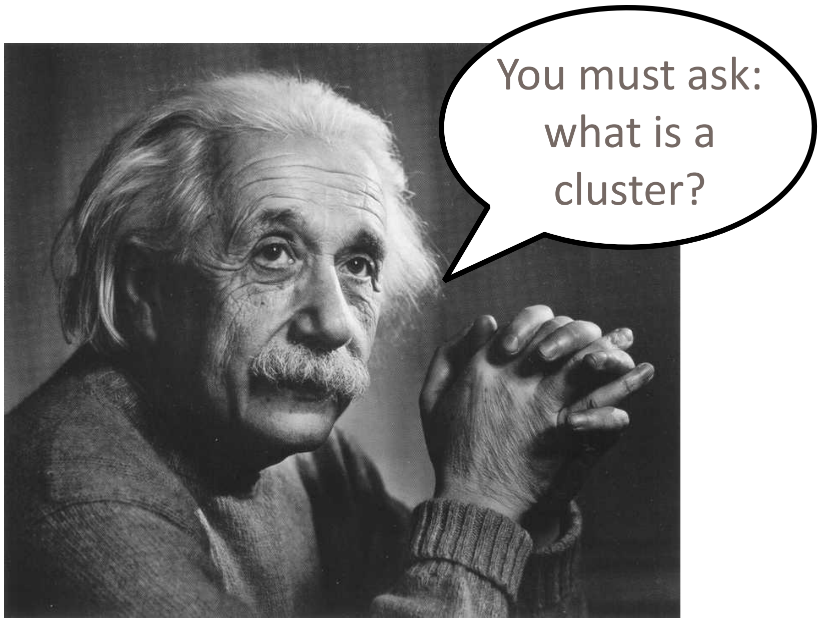
If you want the location of each cluster point to be determined by the average location of the nearest houses, the K-Means clustering algorithm can help with that too. After the initial clustering as described above, just iterate over those points, average the XY coordinates, and now you've got new cluster points.

Repeat this process, each time instructing each house to forget its previous cluster point and find it's new nearest cluster. When the houses stop changing clusters, you will have a map that uses clustering to accurately represent the average location of homes within each region.

The same operation can be used with any type of data you put on a graph. An image is a not so obvious example. Digital images are matrixes of pixels, but you can also represent them as graphs of colors in a color space. If you run a K-Means clustering algorithm on such a graph, you can find out the number of colors used to represent the full range of tones in the final image.

This is somewhat similar to the posterize filter that is common in Photoshop and other editing programs. Posterize reduces the number of colors, downsampling an image from, for example, 24-bit color to 8-bit color. This type of downsampling is the basis of lossy compression mechanisms used in audio and video codecs.

From Data to Clustering to Understanding



You might be asking, "I'm web developer. Why is this information useful to me?"

The answer to that question lies in the nature of the cluster. A cluster is a bunch of objects that you put together in geometric space in such a way that objects that are close together are similar in some way. We've looked at houses, which is an obvious example. We also look at colors, which is a little bit more abstract. But you can use the same method to look at ideas or any other sort of information.

For example, a Star Trek junkie might be especially interested in a character class graph. In this graph, lawful and unlawful are plotted on the y-axis and good and bad are plotted on the x-axis. So up in the left-hand corner, you've got Captain Picard who is good and always follows the rules. On the other hand, Captain Kirk is a good character but follows his own rules, so he falls further to the right.

This might seem like a silly example, but the lesson is important. We can get some great information from this graph that we might not otherwise have been able to extract from the background data. For example, Kirk would probably enjoy having lunch with Q more so than with Captain Picard.

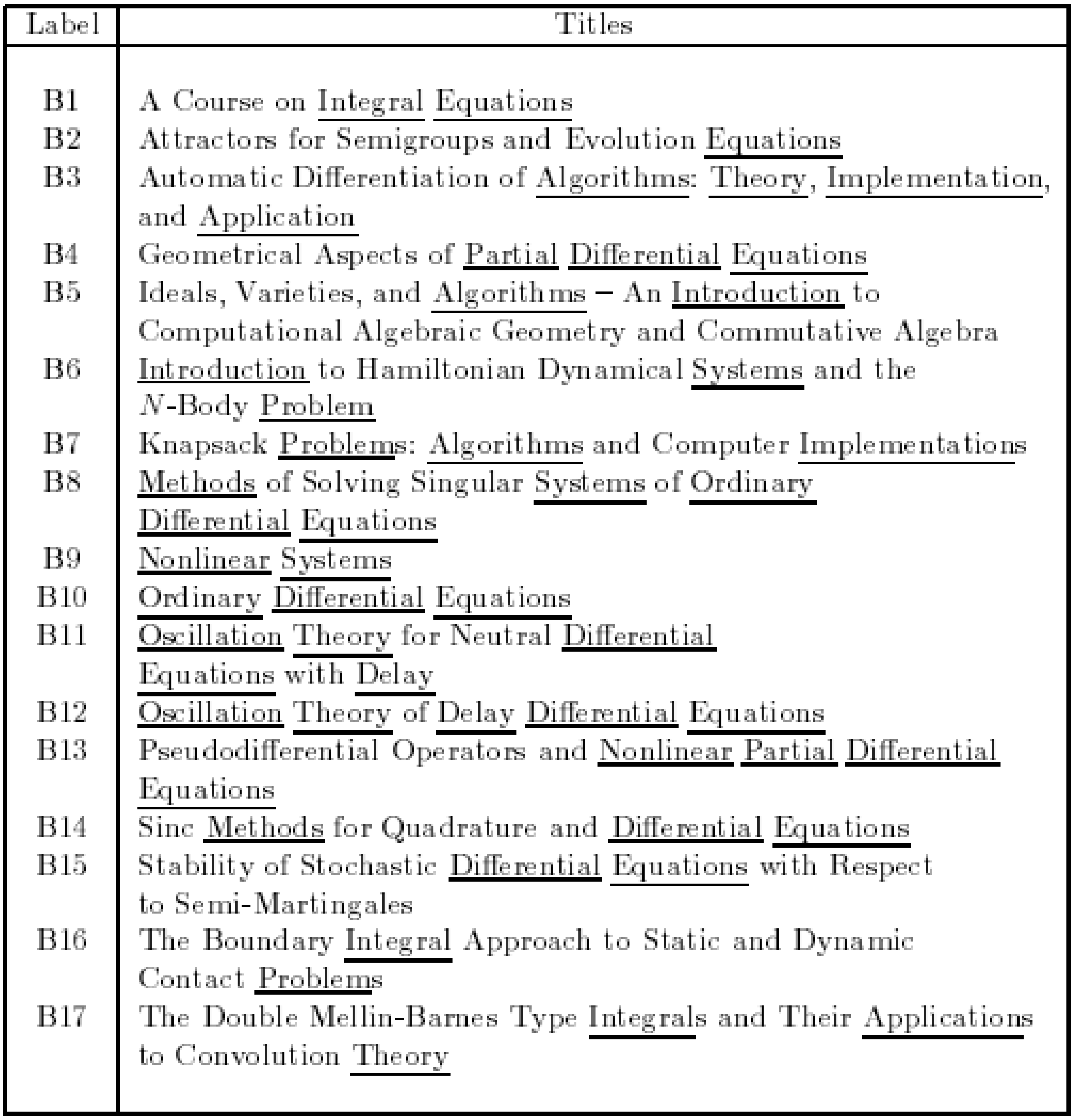
We’re starting to learn things from clustering data, but the next step is a bit less obvious. It's simple to do this as a human with a small data set like a few characters from a TV show, but how you get a computer to understand strings of data and plot meaning on a graph?

Latent Semantic Indexing and Primary Component Analysis

Two great examples of how to use this method to power your applications are Latent Semantic Indexing (LSI) and Primary Component Analysis (PCA). These mathematical tools allow you to cluster data and analyze those clusters to determine similarities between various data points.

Let’s look at an example. Say you have an online bookstore, and you want to provide recommendations based on the past purchasing habits of your customers. One way to get accurate recommendations for your customers is to use keywords.

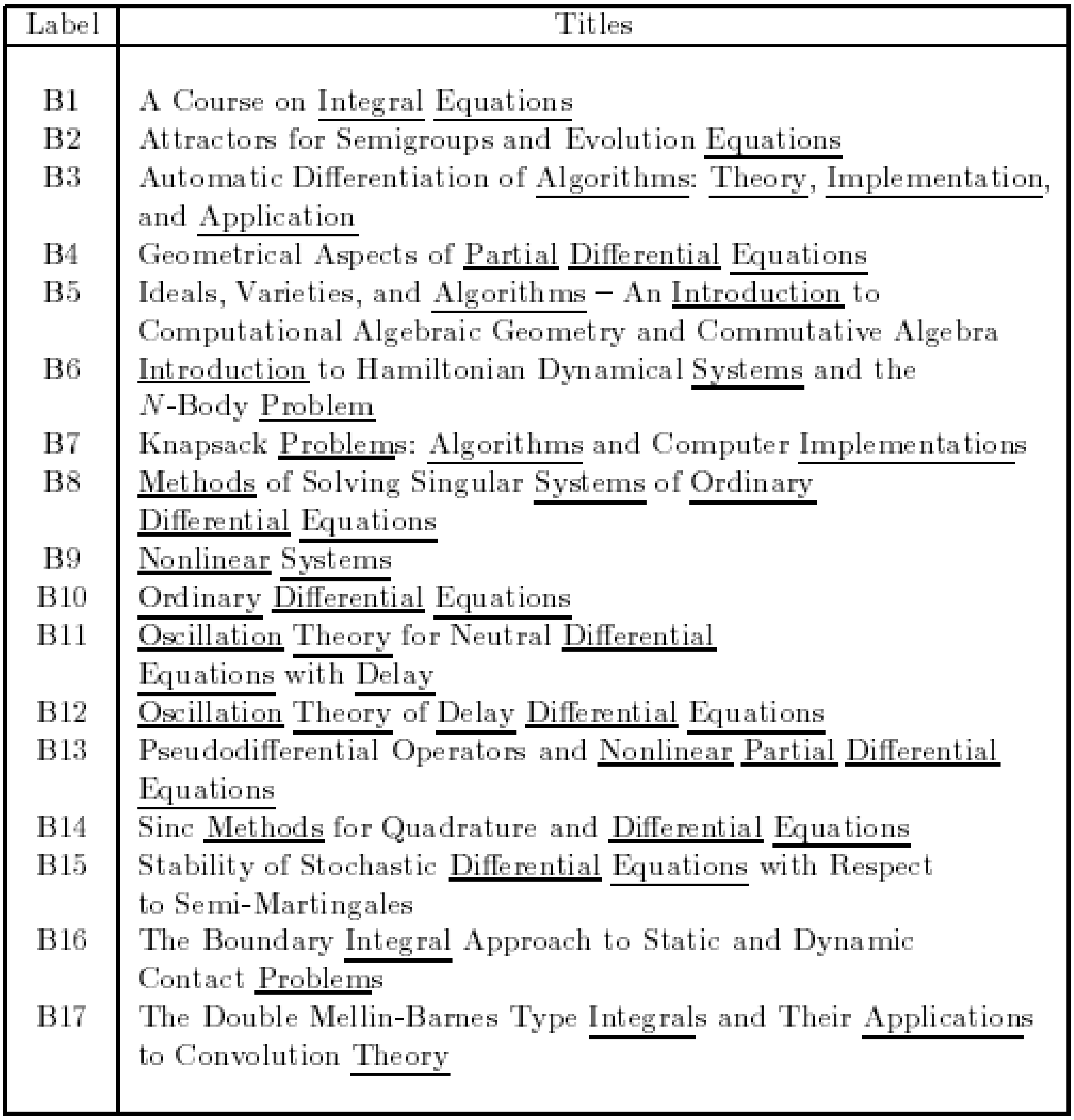
First, you have to analyze the text of every book in the store.



Label	Titles
B1	A Course on Integral Equations
B2	Attractors for Semigroups and Evolution Equations
B3	Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4	Geometrical Aspects of Partial Differential Equations
B5	Ideals, Varieties, and Algorithms – An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6	Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7	Knapsack Problems: Algorithms and Computer Implementations
B8	Methods of Solving Singular Systems of Ordinary Differential Equations
B9	Nonlinear Systems
B10	Ordinary Differential Equations
B11	Oscillation Theory for Neutral Differential Equations with Delay
B12	Oscillation Theory of Delay Differential Equations
B13	Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14	Sinc Methods for Quadrature and Differential Equations
B15	Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16	The Boundary Integral Approach to Static and Dynamic Contact Problems
B17	The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

Source: http://www2.denizyuret.com/ref/berry/berry95using.pdf

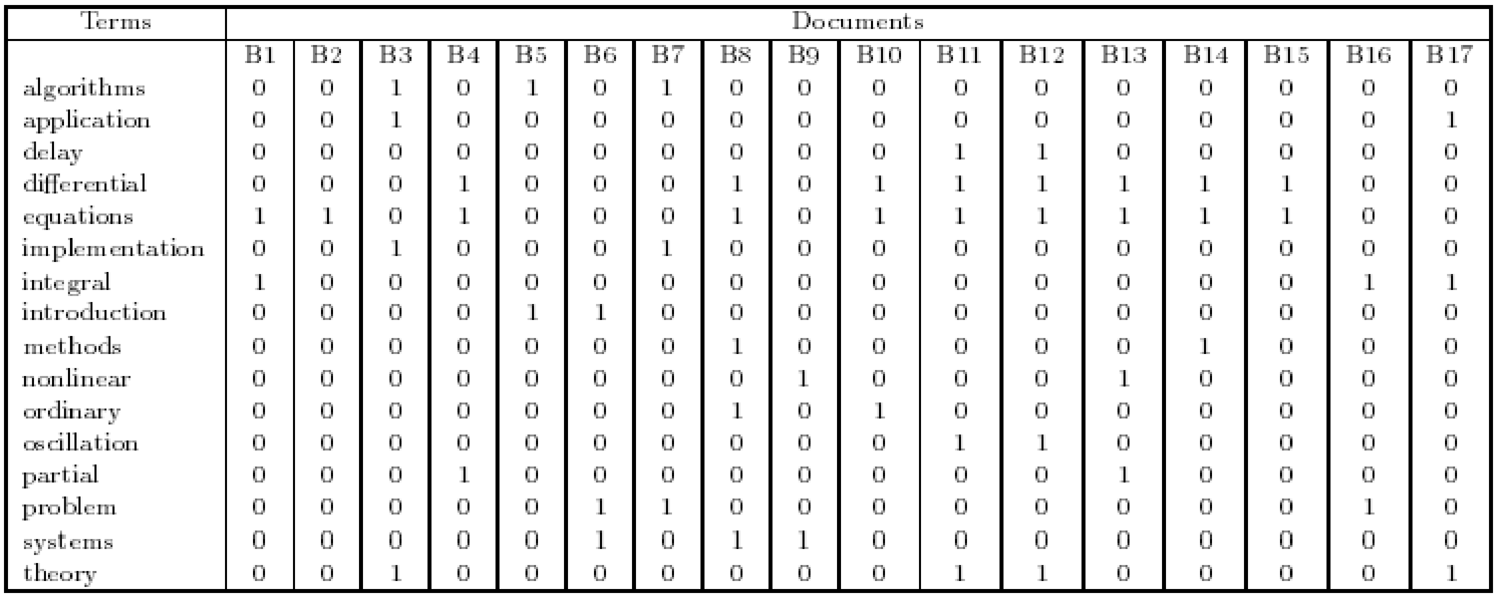
First, you have to analyze the text of every book in the store.



Label	Titles
B1	A Course on Integral Equations
B2	Attractors for Semigroups and Evolution Equations
B3	Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4	Geometrical Aspects of Partial Differential Equations
B5	Ideals, Varieties, and Algorithms – An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6	Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7	Knapsack Problems: Algorithms and Computer Implementations
B8	Methods of Solving Singular Systems of Ordinary Differential Equations
B9	Nonlinear Systems
B10	Ordinary Differential Equations
B11	Oscillation Theory for Neutral Differential Equations with Delay
B12	Oscillation Theory of Delay Differential Equations
B13	Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14	Sinc Methods for Quadrature and Differential Equations
B15	Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16	The Boundary Integral Approach to Static and Dynamic Contact Problems
B17	The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

Source: http://www2.denizyuret.com/ref/berry/berry95using.pdf

Next, generate a matrix of how common each word is in each book.

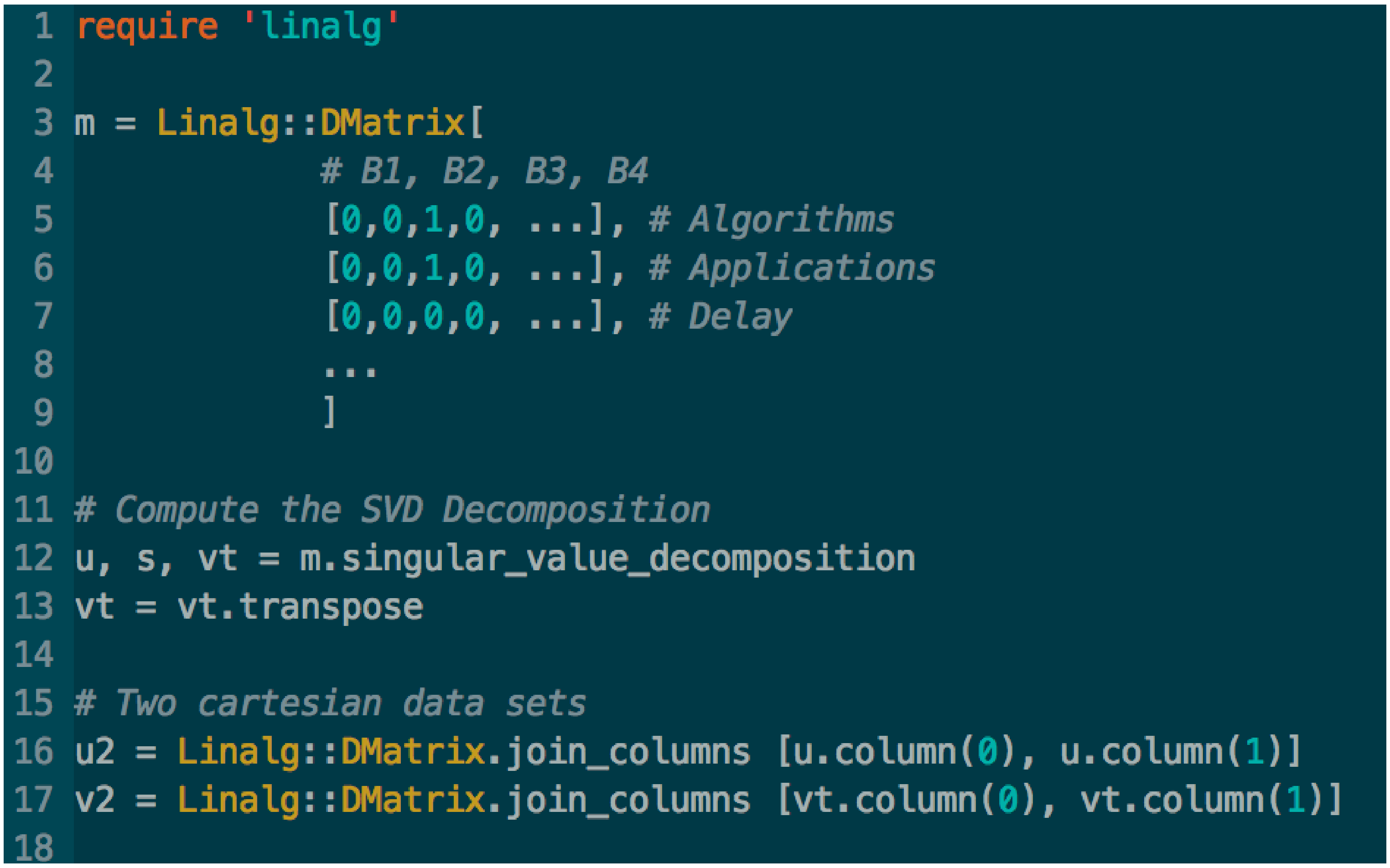


Terms	Documents																
	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17
algorithms	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
application	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
delay	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
differential	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
equations	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
implementation	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
integral	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
introduction	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
methods	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
nonlinear	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
ordinary	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
oscillation	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
partial	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
problem	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
systems	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
theory	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

Source: http://www2.denizyuret.com/ref/berry/berry95using.pdf

Matrixify It

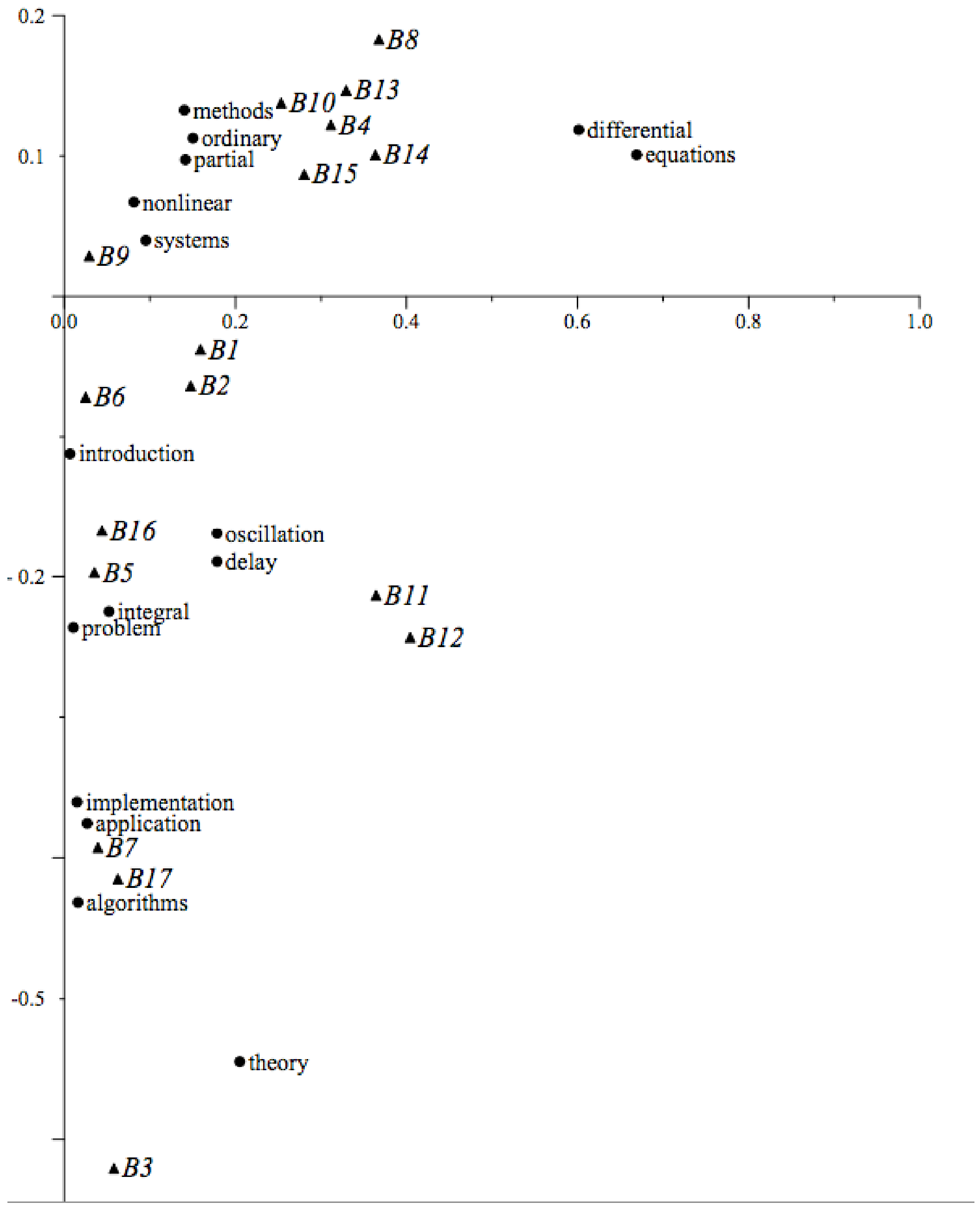
Then, using the matrix, you can plot two graphs - the first relating to the words, the second to the books - with simple linear algebra.



```ruby
require 'linalg'

m = Linalg::DMatrix[
            # B1, B2, B3, B4
            [0,0,1,0, ...], # Algorithms
            [0,0,1,0, ...], # Applications
            [0,0,0,0, ...], # Delay
            ...
            ]

# Compute the SVD Decomposition
u, s, vt = m.singular_value_decomposition
vt = vt.transpose

# Two cartesian data sets
u2 = Linalg::DMatrix.join_columns [u.column(0), u.column(1)]
v2 = Linalg::DMatrix.join_columns [vt.column(0), vt.column(1)]
```

Source: www.igvita.com

Now you have data that you can cluster. If a user likes book #8, then they might also like book #13, #4, and #15, since they contain similar keywords.



Source: http://www2.denizyuret.com/ref/berry/berry95using.pdf

This is a powerful tool that can extend way beyond this example. You could also use it for search, since you can map your queries into the graph(s) that you have created.

Some people speculate that Google is doing things like this in order to separate between similar search terms (for example the Amazon River versus Amazon.com) that fall into widely different clusters of meaning despite the overlapping terminology.

Sentiment Analysis with Naive Bayesian Classifiers

A similar technique can be applied to other situations. To continue with the bookstore example, imagine that you want to classify reviews into two categories – positive and negative – based on contents of the review. This can make it easy for you to sort reviews not only by the number of "stars,“ but also by the actual content.

This is an example of something called sentiment analysis, and to do this we are going to use something called Naive Bayesian Classifiers. Naive Bayesian Classifiers are often used in classifying spam email. There are a great tool: they're fast, easy to implement, simple to understand, and useful for a wide variety of problems.

In order to classify reviews, you first have to gather some data, and this requires human input. So to start a system like this, you hire somebody to look through a good number of reviews – say a thousand, or a couple thousand – and classify them into positive and negative categories.

Now that you have a reliable human classification, you can teach your system to automatically categorize reviews based on the probability of certain words or phrases occurring in a negative review vs. a positive review.

```
{
    best:      { pos: 1, neg: 0 },
    bleed:     { pos: 0, neg: 1 },
    forbidden: { pos: 1, neg: 1 },
    ...
}
```

It’s well known that Naive Bayesian Classifiers have their issues. Since the probabilities are based on rough inferences made by computers, they can be wrong. Who hasn’t had an important email classified as spam? Nonetheless, this is still a really powerful tool that you can use to take your data and understand it.

“I will love this book forever. It’s the bloody best.”

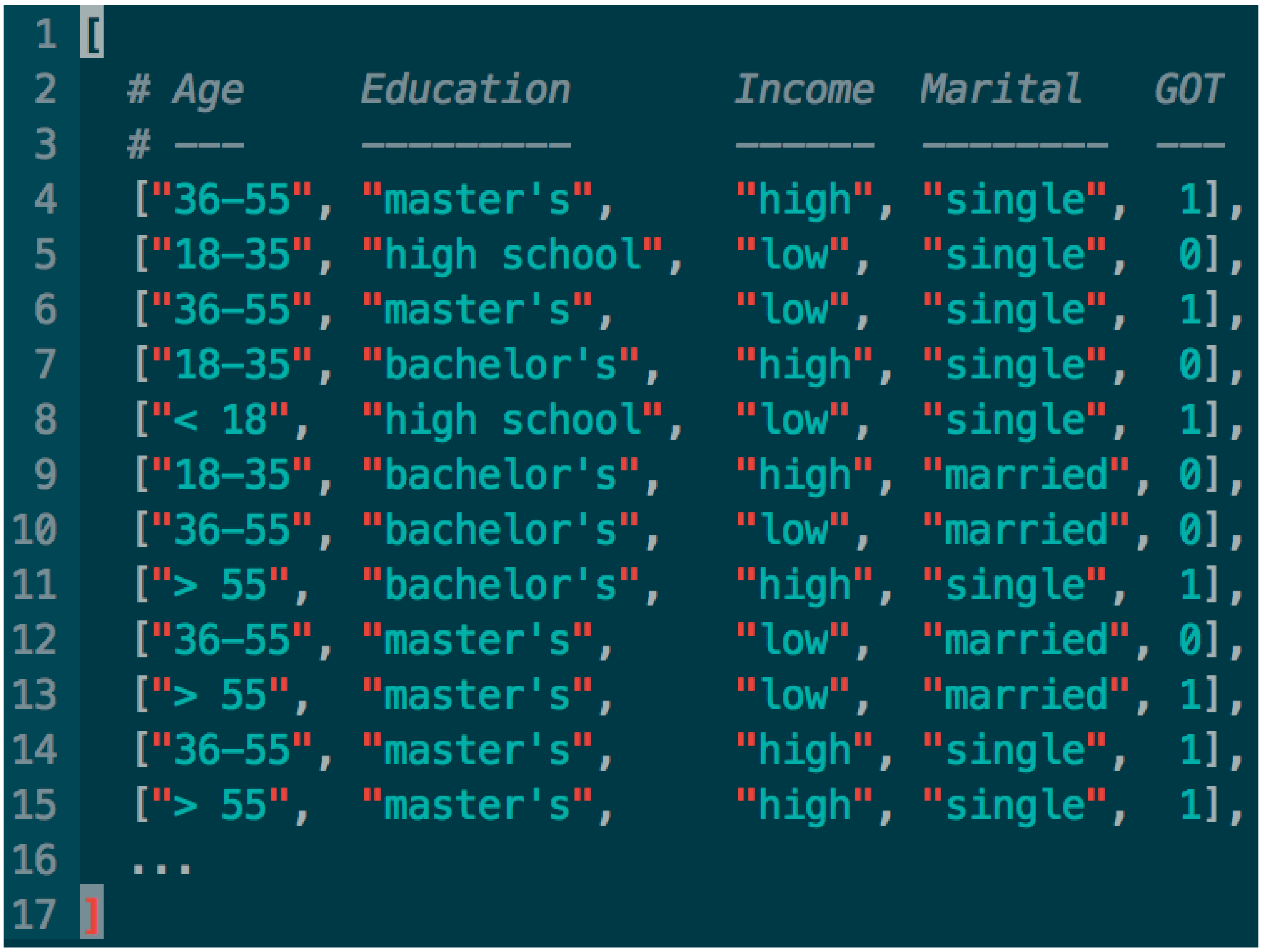
Pos: 59% > Neg: 38%

Decision Trees

Here is another incredibly powerful mathematical tool. Imagine that you've got a bunch of customers at your bookstore, and a new book is coming out. Let's say it's a big release, like the new Game of Thrones book, and you want to notify customers about this.

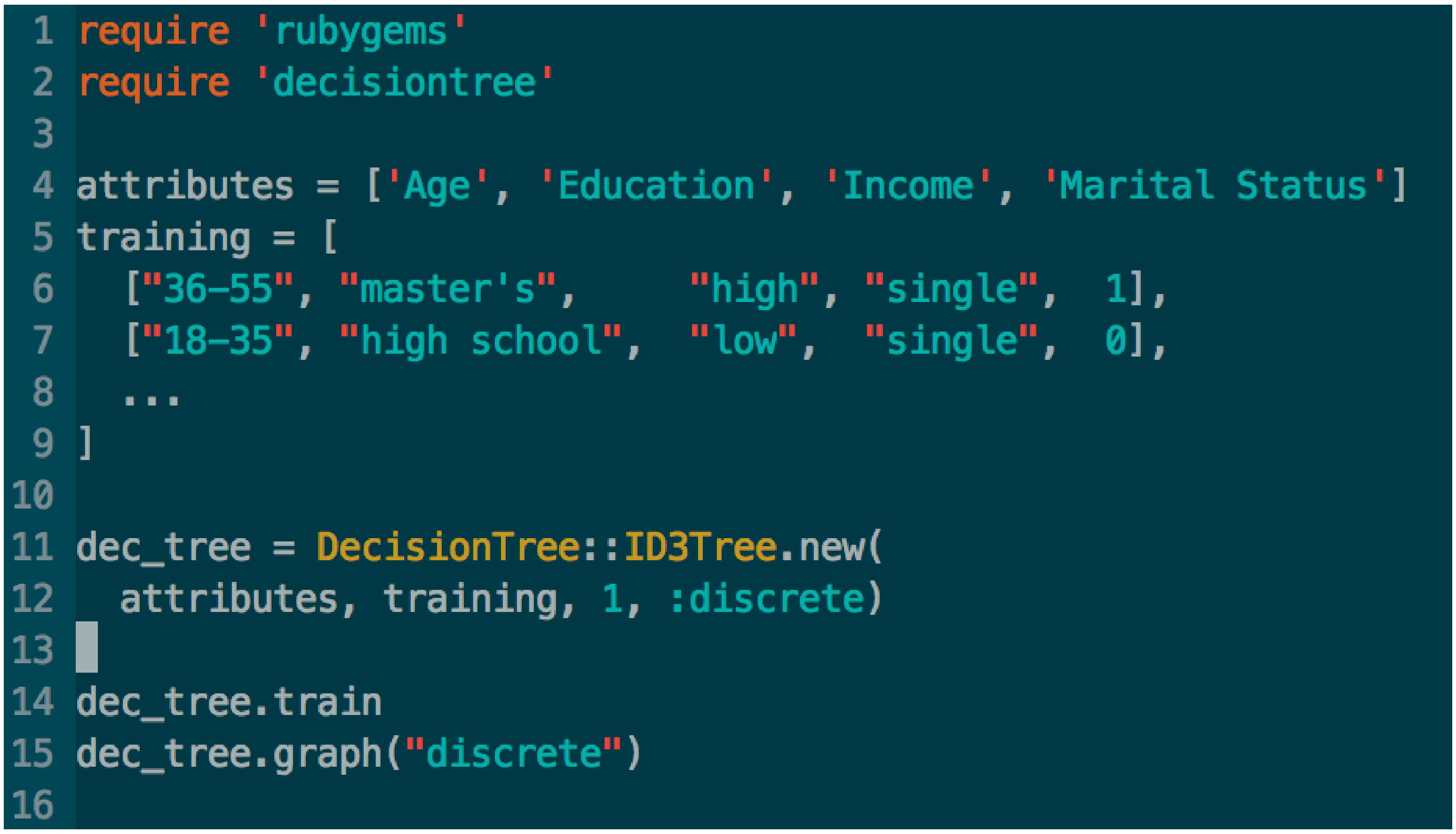
But the last thing you want is to spam people who don't care for fantasy. How do you classify your customers into those who are or are not likely to want a Game of Thrones book?

Decision tree makers allow you take a bunch of data, like demographic information about your customers, and run it through the algorithms to generate a graph with probabilities of what sort of customer is most likely to want Game of Thrones. This type of data can be really useful for analyzing your market.

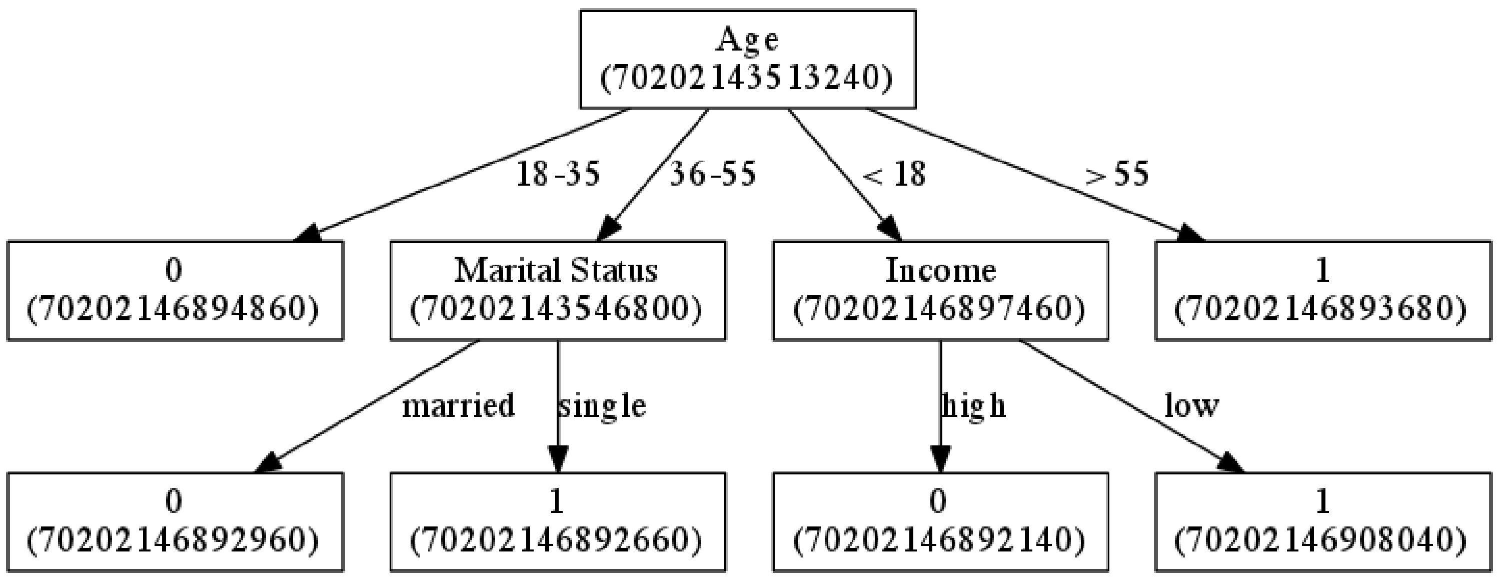


```
[
  # Age     Education          Income  Marital    GOT
  # ---     ---------          ------  -------    ---
  ["36-55", "master's",        "high", "single",   1],
  ["18-35", "high school",     "low",  "single",   0],
  ["36-55", "master's",        "low",  "single",   1],
  ["18-35", "bachelor's",      "high", "single",   0],
  ["< 18",  "high school",     "low",  "single",   1],
  ["18-35", "bachelor's",      "high", "married",  0],
  ["36-55", "bachelor's",      "low",  "married",  0],
  ["> 55",  "bachelor's",      "high", "single",   1],
  ["36-55", "master's",        "low",  "married",  0],
  ["> 55",  "master's",        "low",  "married",  1],
  ["36-55", "master's",        "high", "single",   1],
  ["> 55",  "master's",        "high", "single",   1],
  ...
]
```

There is some fascinating math going on in the background here - stuff that was originally developed in the 1950s for signal analysis work - that allows the computer to predict a certain outcome using the fewest number of steps. In short, it involves determining which types of data (age, sex, income, etc.) have the greatest impact on the outcome (liking or not liking Game of Thrones).



```ruby
require 'rubygems'
require 'decisiontree'

attributes = ['Age', 'Education', 'Income', 'Marital Status']
training = [
  ["36-55", "master's",     "high", "single",  1],
  ["18-35", "high school",  "low",  "single",  0],
  ...
]

dec_tree = DecisionTree::ID3Tree.new(
  attributes, training, 1, :discrete)

dec_tree.train
dec_tree.graph("discrete")

```



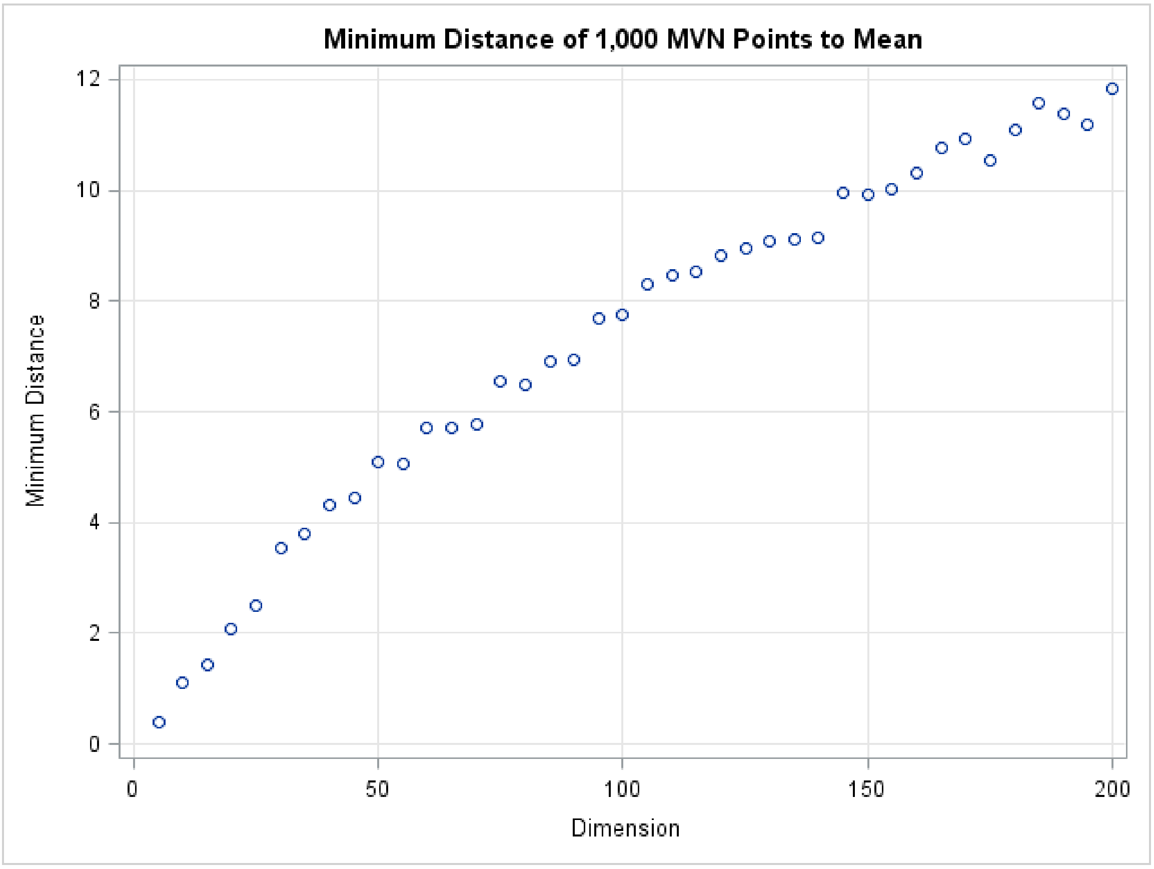
Decision Tree Graph

Beware The Curse of Dimensionality

Let’s look at a problem that can arise. When you are dealing with large datasets with a ton of different variables, one issue that can arise is the dramatically-named “Curse of Dimensionality.” Don’t worry though - it’s a lot simpler and less scary than the name makes it seem.

The Curse of Dimensionality basically means that if you have a dataset, and that dataset is along one dimension (a line), as you increase the number of dimensions in that dataset, the data points get farther and farther apart.

It's easy to think about if you think of two points on a line. This is just one dimension, but now, you add a second axis - height. For any value of height except for zero, the distance between any two points gets bigger. If you wanted to add a third dimension, it's the same thing.



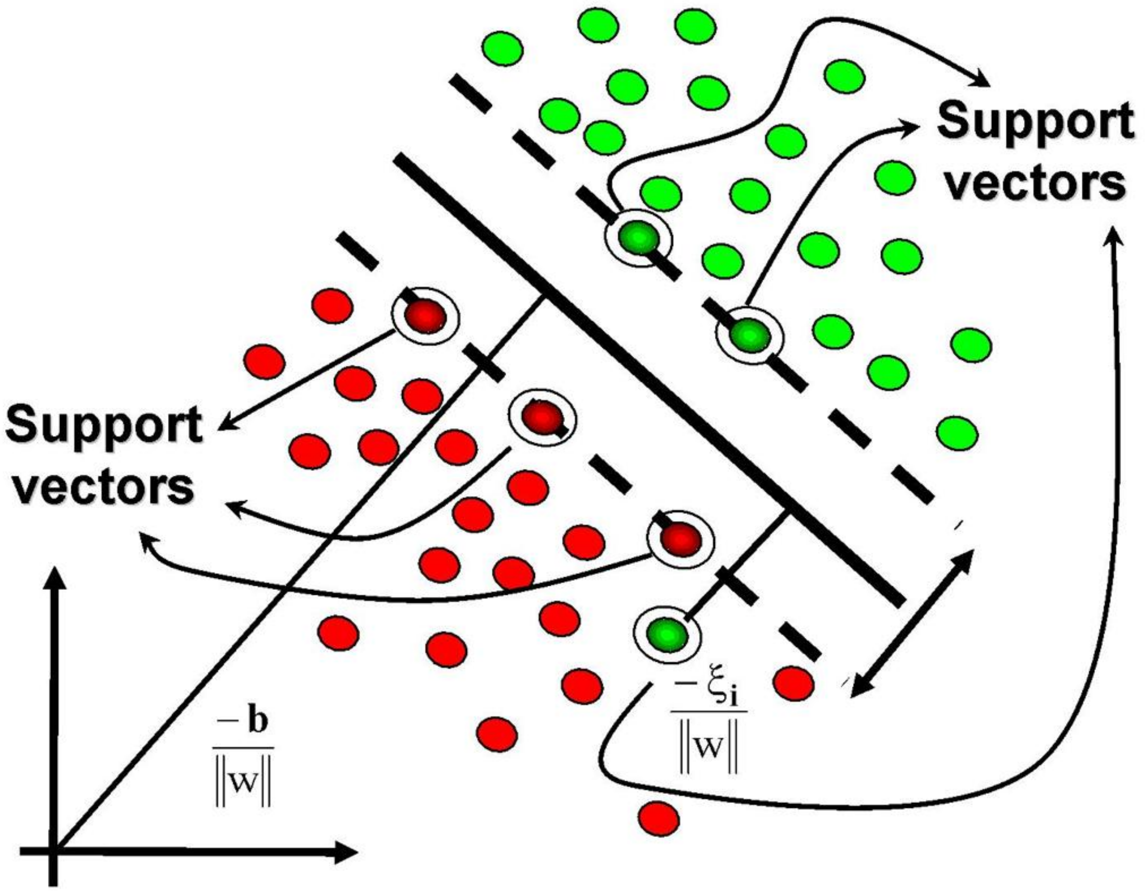
Source: http://blogs.sas.com/content/iml/2012/03/23/the-curse-of-dimensionality/

This is important because as you add dimensions - as your data gets more and more spread apart - a lot of machine learning models tend to get less effective. But there are some tools out there that deal well with highly-dimensional data, and one of those is Support Vector Machines.

Optical Character Recognition (OCR) with Support Vector Machines

Support Vector Machines are one type of classifier. The highly-simplified explanation of why they foil the Curse of Dimensionality is that they're not actually looking for clumps of data. They're actually trying to find lines or planes that give you the greatest separation between data, so you're actually looking for the space in between your data points.

Another advantage of Support Vector Machines is their support for nonlinear boundaries. In most of these machine learning techniques, data is classified based on a line in the sand: either a review is good or it is bad. With this technique, your lines don’t have to be so clear cut, and this is really useful when it comes to something like handwriting recognition, or Optical Character Recognition (OCR).



The reason for this is that OCR deals with images. Say you’ve got a set of 32x32 bitmaps for each character. If you want to create an OCR system based on these bitmaps, you first need to turn them into a 1,024-dimension matrix. Yes, it sounds crazy, but once you do that, you can run some linear algebra classifier tricks on the matrix and end up with a powerful handwriting recognition system.

And that is amazing stuff.

Conclusion: The Importance of These Methods

We've gone over quite a few methods here. We started with the basics of Machine Learning and glossed over LOESS, K-Means Clustering, Latent Semantic Indexing and Primary Component Analysis, Naive Bayesian Filers, Decision Trees, got a bit of warning about the Curse of Dimensionality, then learned one method to combat it with Support Vector Machines.

These gems work great, although for big datasets and real-time user interfaces these operations might be best written in C.

The great news for developers is that there are Ruby libraries for these operations.

Another great resource, which provided a lot (ahem, all) of the code examples from this e-book is Ilya Grigorik's blog.

http://www.igvita.com/

If you want to learn more about these subjects check out our blog or Twitter:

http://honeybadger.io/blog

@honeybadgerapp