Text Classification

Joseph Fallait

Classification Rule:

Given a population whose members each belong to one of a number of different sets or classes, a classification rule or classifier is a procedure by which the elements of the population set are each predicted to belong to one of the classes.[1] A perfect classification is one for which every element in the population is assigned to the class it really belongs to. An imperfect classification is one in which some errors appear, and then statistical analysis must be applied to analyse the classification.

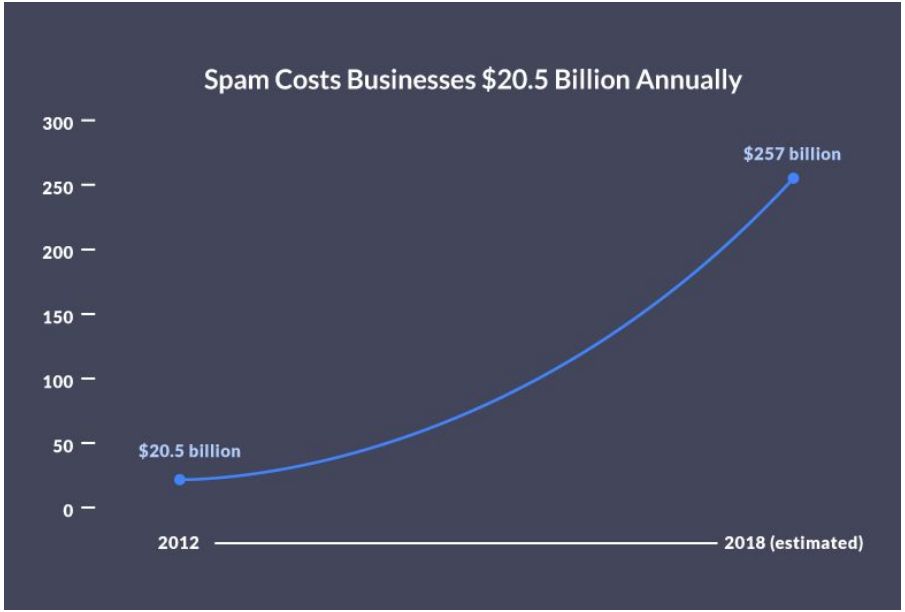
A special kind of classification rule is binary classification, for problems in which there are only two classes.

(https://en.wikipedia.org/wiki/Classification_rule)

Why Spam Classification?

According to a study done by Radicati Research Group Inc., email spam cost businesses $20.5 billion a year in 2012.

(https://www.propellercrm.com/blog/email-spam-statistics, https://www.radicati.com/?tag=spam)



My Project: Implementing and Comparing Naive Bayes Classifier and Support Vector Machine

Goals were to:

- Learn about good resources for datasets to find clean and helpful data
- Understand the underlying mechanisms of Naive Bayes and Support Vector Machine Classifiers
- Create a flexible set of functions that would allow me to reuse my functions for future data analysis
- Compare run-times and accuracy of the classifiers
- Optimize for training-set proportion

Naive Bayes Classifier

Naive Bayes Classifier

A simplistic approach to text classification. It is based entirely on Bayes' Theorem of Conditional Probability.

$$P(\mathrm{A}|\mathrm{B}) = \frac{P(\mathrm{B}|A) \cdot P(A)}{P(B)}$$

Basic Procedure of Naive Bayes

1. Separate data into a training set and a test set. A key aspect to statistical prediction is to **NOT** look at the test set until the time comes to predict it, so we should never access it.
2. Create probability scores for each word by:
    a. Track total number of Spam and Ham words
    b. Find the number of occurrences of each word in Spam and in Ham
    c. Calculate probability of SPAM given STRING/HAM given STRING for each potential string inputted using Bayes' Theorem
3. We can now multiply these probabilities to predict the probabilities that a potential message is either Spam or Ham.

NOT

What if the word “cat” appears in HAM but not SPAM?

Based off the basic method I just outlined, the SPAMSCORE for:

“press the link https://bostoncollege.instructure.com/ to transfer your ssn, cat” would be 0, while the HAMSCORE would be greater than 0 provided the other words appear in HAM texts that appear in the training set.

We would classify this message as HAM and not SPAM, which we know doesn’t make sense (Hopefully).

The solution is log smoothing: or taking the sum of the log of the wordscores + 1.

Sample Scoring

Say we have the sentence “big wire transfer”

Spamscore = -18.553682458368257

Hamscore = -21.244017628970745

(Scores were negated during the process so the greater score would correlate to the predicted classification)

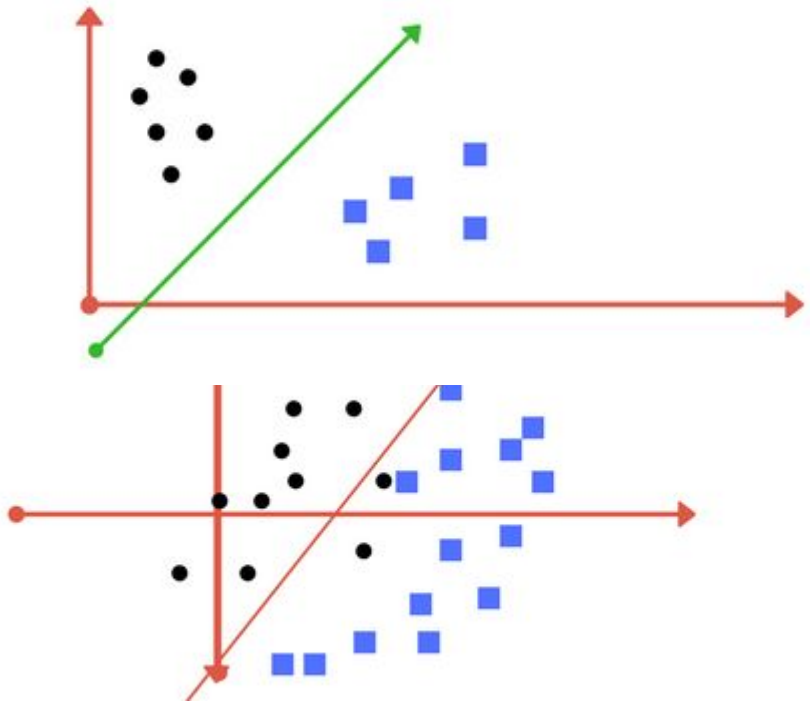
Spamscore > Hamscore, so we classify the sentence as SPAM.

Support Vector Machine

What is a Support Vector Machine?

A Support Vector Machine is another method of classification. Instead of compiling a list of scores and simply summing up or multiplying to equate a total probability it instead discriminates based off what is essentially a line of best fit.

Say we labeled spam as black points and ham as blue points, here are some situations we might find ourselves in.



https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

Kernels

There are a number of different ways we can draw this line of best fit, each requiring a different kernel.

“In machine learning, a “kernel” is usually used to refer to the kernel trick, a method of using a linear classifier to solve a non-linear problem. It entails transforming linearly inseparable data like (Fig. 3) to linearly separable ones (Fig. 2). The kernel function is what is applied on each data instance to map the original non-linear observations into a higher-dimensional space in which they become separable.” (https://towardsdatascience.com/kernel-function-6f1d2be6091)

For a two-dimensional analysis where we only have text messages and a category, it makes sense to use a linear kernel.

Predictions via Linear Kernel

The formula to predict a new point with a linear kernel is:

$$f(x) = B(0) + sum(ai * (x,xi))$$

Where:

- B(0) and ai are calculated via the learning algorithm (the best fit)
- x is the input value
- xi is the support vector
- (x, xi) is the dot product of these two

https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

Vectors with only Text?

As mentioned in the previous slide, we can't actually compute a prediction for a point or even create a model without first converting our data into vectors. Sklearn provides a bunch of great tools to do so.

The CountVectorizer class converts the string: "recursion is all about recursion" into a matrix counting each occurrence of a word (the order is mixed up in the process). It would return [[1, 1, 1, 2]]

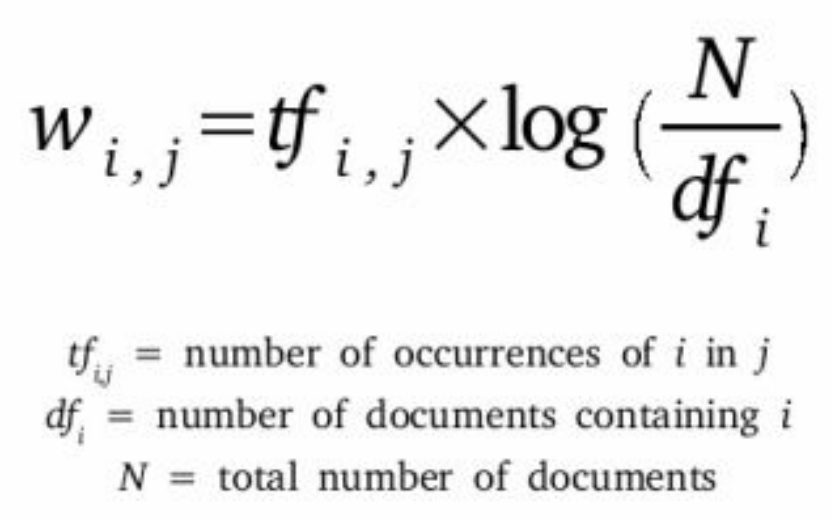
We then use a TfidfTransform, or Term-Frequency Inverse Document Frequency transformation.

https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

Term Frequency - Inverse Document

In order to further refine the results we receive, the SKlearn kit also provides a class called TfidfVectorizer.

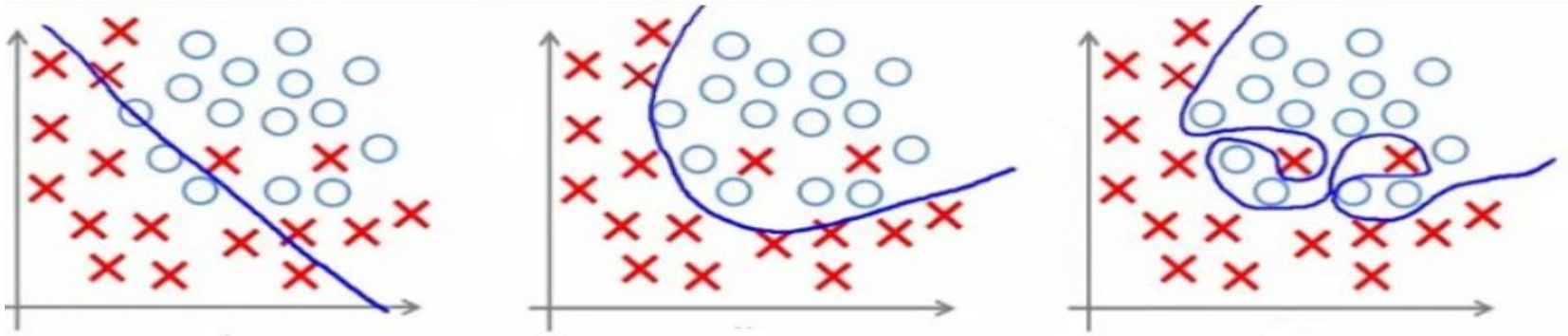
This class uses the formula to the right to penalize words that appear extremely frequently in the entire training set. This includes very frequently used words like “the”, or even other statistical noise relating to datasets, like the word “cancer” in a cancer dataset.



https://medium.com/@imamun/creating-a-tf-idf-in-python-e43f05e4d424

Evaluating The Classifiers

Optimizing via Training Set Size:



Under-fitting

(too simple to explain the variance)

Appropriate-fitting

Over-fitting

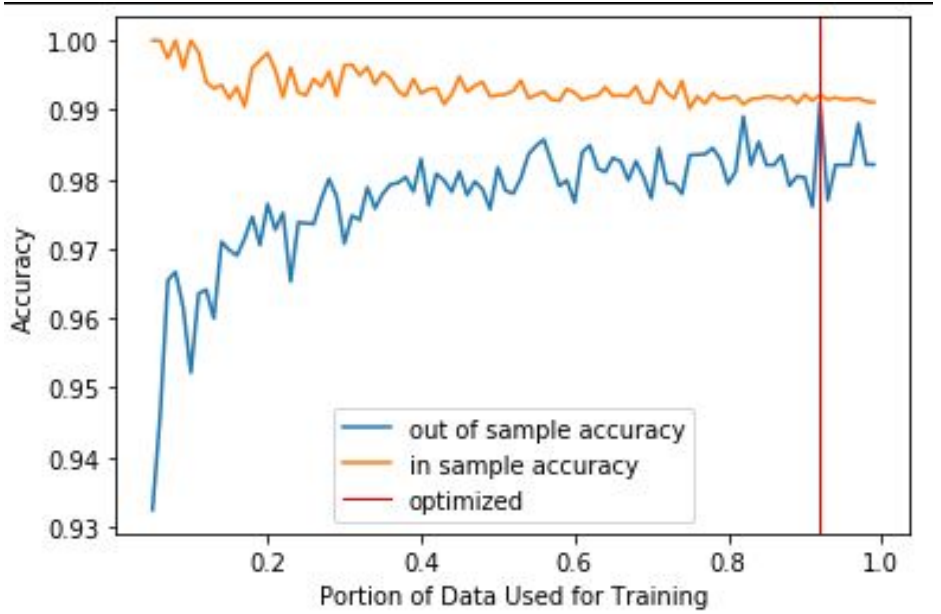
(forcefitting -- too good to be true)

In creating a classifier, the goal is to have it perform well in and out of sample. Underfitting is not explanatory enough, and over-fitting lets in a lot of statistical noise.

Optimizing the Naive Bayes Classifier Model

The Naive Bayes Classifier runs a lot faster than Support Vector Machines, because SVM requires a number of different lines to be fit and tested to determine the B(0) and ai coefficients from the linear kernel formula.

Because NVM is cheap, we can run it a bunch of times with different training set portions to find this point. Over about 50 trials and 2 minutes of run-time, the average optimized point I found was about .86-.88 training data.



Results of SVM

- Using the same portion size as I found was optimal in the Naive Bayes model, SVM performs similarly well as Naive Bayes if not a little bit better.
- On an average run, the Naive Bayes Classifier takes less than .1 seconds, hovering around .07 seconds. A highly optimized SVM, however, even one provided by sklearn, still hovers around 1 second for the exact same provided data.
- So the takeaway should be that Naive Bayes Classifier does the same job in much less time right?

Not necessarily.

No Free Lunch

- Naive Bayes excels in circumstances with fewer instances in comparison to SVM. The dataset used for this Spam/Ham classifier used around 5500 text messages, which by classifier standards is relatively small.
- Naive Bayes treats every datapoint observed as a being independent, but this isn't always the case. Especially in datasets where there are a wider variety of classes with a bunch of overlap between the words defining the classes, SVM might be more effective.
- According to the **No Free Lunch** theorem "any elevated performance over one class of problems is exactly paid for in performance over another class." (http://www.no-free-lunch.org/) The moral of this story is that you need to tailor your classifier algorithm to the problem you're trying to solve, in this situation a simpler Naive Bayes classifier was just as strong as SVM.