Spencer Walker
0715530
March 29, 2016
CIS*3700

Assignment 5: Bayesian Spam Filtering

Step 1. Collect a dataset of ham emails and spam emails

For my data collection I used two separate contest sites which were very similar. Both offered a folder full of “training” emails with a label file used to identify which files were ham/spam. Both also came with a “testing” folder, but for the sake of their competition, this folder did not come with a label file. Because of this, I used the training folders but used the smaller of the two for testing because I could test my results against the label file.

Training: http://csmining.org/index.php/spam-email-datasets-.html
- The training files from this source can be found in my a5/training

Testing: https://inclass.kaggle.com/c/adcg-ss14-challenge-02-spam-mails-detection/data
- The testing files from this source can be found in my a5/testing folder. Because of the reasons explained above, all of these files have the word “TRAIN” in them

Step 2. Tokenize your datasets into words choosing suitable delimiters

Before tokenizing my datasets, I used the tool ExtractContent.py which was included in the competition folder. This removes the <body> from the email which reduces file formatting that needs to be omitted. I then parsed the files and used the function *re.findall(r"[\w']+", string)* function to extract all “substrings consisting of a sequence of word characters”.

Step 3. Build a dictionary listing the words and their number of occurrences

My program contains a function called generateDictionaries() that determines which files are ham or spam by comparing all files in the */training* directory to the label file */training/SPAMTrain.label*. The tokens from each class (ham/spam) are then saved in their respective categories as tuples with their corresponding number of occurrences.

These collections are saved as HAM_DICTIONARY and SPAM_DICTIONARY, so this function should only be called when creating a new dictionary.

Step 4. Calculate $P(\ word \mid ham\ )$ and $P(word \mid spam)$ using these frequencies

$P(word \mid spam) = occurrences/numFiles$

Example
word: free
- appears 43 times in 1077 emails
- $P(free \mid ham) = \ .039 = 4\%$

- appears 391 times in 1077 emails

- $P(free \mid spam) = .363 = 36\%$

This means there is a 4% chance a given ham message contains the word “free” and a 36% chance that a given spam message contains the word

Step 5. Estimate $P(spam)$ and $P(ham)$

Estimating P(ham) and P(spam) having always had a spam filter installed is challenging. With the modern intelligence and sheer quantity of email bots out there, it seems probable that the vast majority of received email is spam. I would estimate P(ham) = .3 and P(spam) = .7. This study shows that upwards of 80% of email is spam, so P(spam) ≈ .8.

The Wikipedia article on Bayesian Spam Filtering states that most filters use a non-biased approach with no a priori reason assuming p(ham) = p(spam). In my implementation I base the dictionary off of an equal number of ham and spam emails in the learning phase to conform to this 50% rule.

Step 6. Use Bayes’ Theorem to calculate $P(ham \mid word)$ and $P(spam \mid word)$ for each word

$P(spam \mid word)$ is the probability that a message is spam, given that it contains a word. Because we only want important keywords that can tell us useful information, only words that occur ≥ 5 times in all files during the learning phase are considered for this calculation.

$$P(spam \mid word) = \frac{P(word \mid spam)P(spam)}{P(word \mid spam)P(spam) + P(word \mid ham)P(ham)}$$

Because I use the non-biased approach, P(ham) and P(spam) can be omitted in this calculation. It can then be calculated as follows:

$$P(spam \mid word) = \frac{P(word \mid spam)}{P(word \mid spam) + P(word \mid ham)}$$

Continuing the previous example, if we have $P(free \mid ham) = .039 = 4\%$
and $P(free \mid spam) = .363 = 36\%$ then:

$$P(spam \mid free) = \frac{P(free \mid spam)}{P(free \mid spam) + P(free \mid ham)}$$

$$= \frac{.363}{.363 + .039} = .902 = 90\%$$

Based on this single event, it would be 90% likely that the email is spam given that it contains “free”. A single word is obviously not enough to be 90% certain that an email is spam, so we need to factor in the probabilities from more events to increase accuracy (next step).

Step 7. Compute $P(spam \mid word1\ and\ word2\ and\ word3)$

Calculating the chance the message is spam using multiple independent events requires the following, modified formula:

$$\frac{P(spam\mid word1)P(spam\mid word2)P(spam\mid word3)}{P(spam\mid word1)P(spam\mid word2)P(spam\mid word3)+(1-P(spam\mid word1))(1-\ P(spam\mid word2))(1-P(spam\mid word3))}$$

By factoring in each independent event, the new formula can provide a more accurate estimate of how likely the message is to be spam.

Step 8. Build a Bayesian spam filter to classify email as spam or ham

See README for instructions on how to run the program.

Step 9. Adjust filter to find a balance between false positives and false negatives

I tested about 50-100 of the files from the *testing/* folder and comparing them against the answer key in *testing/spam-mail.tr.label*. I found that the most accurate threshold for classifying something as spam was 70% (after the calculation in step 7). I also chose only the 10 words with the greatest absolute value $\left|0.5 - P(ham \mid word)\right|$ and $\left|0.5 - P(spam \mid word)\right|$ as to reduce the number of neutral words which don't give much information.

In addition to this, I observed the most frequently occurring words that were being chosen as the 10 keywords. I selectively made a list of words that are considered neutral, as well as common HTML attributes and CSS properties. In a future version of the program I would parse these out ahead of time.

Step 10. Evaluate the accuracy of your filter based on the same dataset used in learning phase

The output for these test cases can be found in the *step10tests/* directory.

I tested the last 10 files rather than the first 10, because in the process of equalizing the dataset (numHamFiles = numSpamFiles) I deleted the first thousand ham files in the folder. Because of this, the first 10 files would have all been spam.

File	Class	My output
training/TRAIN_03401.eml	spam	spam
training/TRAIN_03402.eml	ham	ham
training/TRAIN_03403.eml	ham	ham
training/TRAIN_03404.eml	ham	ham
training/TRAIN_03405.eml	spam	spam
training/TRAIN_03406.eml	ham	ham

training/TRAIN_03407.eml	spam	spam
training/TRAIN_03408.eml	spam	spam
training/TRAIN_03409.eml	ham	ham
training/TRAIN_03410.eml	spam	spam

With testing these last 10 files which were used as part of the "learning" phase, my program actually gets 100% success. That is, there was not a single false positive or false negative in these test cases.

Step 11. Evaluate the accuracy of your filter based on a new dataset

The output for these test cases can be found in the *step11tests* directory

File	**Class**	**My output**
testing/TRAIN_1.eml	spam	spam
testing/TRAIN_2.eml	spam	spam
testing/TRAIN_3.eml	ham	ham
testing/TRAIN_4.eml	spam	spam
testing/TRAIN_5.eml	spam	spam
testing/TRAIN_6.eml	ham	ham
testing/TRAIN_7.eml	ham	ham
testing/TRAIN_8.eml	ham	spam
testing/TRAIN_9.eml	ham	spam
testing/TRAIN_10.eml	ham	ham

Testing the first 10 files from the new directory gave decent results. I had a total of two false positives which are less favorable than false negatives. From examining the output, it is clear that no informative keywords were caught to suggest a high probability of ham. The word "because" was chosen as one of the 10 keywords, scoring a .498 score. This word could be safely added to my list of neutral words, but since my program prioritizes scores furthest away from .5 it is surprising that this word was chosen.

Overall I am happy with the performance of my naïve spam filter. This basic algorithm can be quite effective. It would be much more powerful if I had a larger dataset for the learning phase. In addition, more heuristics for sequential words would have a great effect on the accuracy.