Nadia Paquin
IST664 NLP
December 16, 2023

Final Project: Email Classification

Scope:

This project provided a dataset of 1500 spam emails and 3672 non-spam "ham" emails to use as training data in order to build a classifier to parse and detect spam. I used the NLTK package to pull in data, tokenize the corpora, build feature sets to train the Naive Bayes classifier, and cross-validate the results. I experimented with several different methods to build the classifier, including changing the vocabulary size, changing the corpus size, adding stop and negation words, POS tagging, using n-grams, and looking at the ratio of words to punctuation.

Results:

Given the input of 1000 corpora files (500 spam and 500 ham), I created a feature set on the top 2000 most frequent words of the corpus input. Initially I did not choose to filter the data for punctuation or stop words as I found that the repetitive occurrence of either of those is sometimes indicative of spam emails. Using this featureset, I used the NLTK Naive Bayes classifier to determine which sets of words occur more in either category and sort the corpora as either spam or ham. The mean accuracy on my first run was 95.4% with high precision on spam and less precision on ham (false negatives). This would result in a spam filter that occasionally (8% of the time) sends your real email to your spam folder. The recall values confirm that some ham emails end up in the spam folder. The F1 score, which indicates the harmonic mean of precision and recall, demonstrates the balance of precision and recall. With scores over 0.95, this classifier is robust. It might be beneficial however for the error to occur in high ham precision and lower spam precision, so as to avoid tossing any important emails aside at the risk of having a few spam emails enter your inbox.

No adjustments (2000 common words, 1000 files)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.96
        Precision       Recall          F1
ham           0.920      1.000      0.958
spam          1.000      0.926      0.962
Each fold size: 200
0 0.975
1 0.94
2 0.945
3 0.95
4 0.96
mean accuracy 0.954
usage: python classifySPAM.py <corpus-dir> <limit>
```

The following three results are caused by experimentation with the vocabulary size. When creating the feature set, I use NLTK's FreqDist command to identify the top most frequent words in a corpus. In the previous example, I began with 2000 words. The next three decrease to 1000, 500, and 100 words. The mean accuracies are 95.3%, 95.6%, and 95.4%. Accuracy overall remains steady despite narrowing down the words. The precision and recall however begin to level out as the number of words shrink. 2000 and 1000 words yield almost identical results but the 500 and 100 words' precision and recall appear to become more evenly split.

Reduced freqdist (1000 common, 1000 files)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.96
        Precision       Recall          F1
spam          1.000      0.921      0.959
ham           0.925      1.000      0.961
Each fold size: 200
0 0.965
1 0.96
2 0.935
3 0.945
4 0.96
mean accuracy 0.953
usage: python classifySPAM.py <corpus-dir> <limit>
```

Reduced freqdist (500 common, 1000 files)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.94
        Precision       Recall          F1
ham           0.892      0.989      0.938
spam          0.990      0.898      0.942
Each fold size: 200
0 0.95
1 0.97
2 0.955
3 0.965
4 0.94
mean accuracy 0.9559999999999998
usage: python classifySPAM.py <corpus-dir> <limit>
```

Most reduced freqdist (100 common, 1000 files)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.955
        Precision       Recall          F1
ham           0.915      0.989      0.950
spam          0.991      0.929      0.959
Each fold size: 200
0 0.945
1 0.955
2 0.955
3 0.955
4 0.955
mean accuracy 0.953
usage: python classifySPAM.py <corpus-dir> <limit>
```

The following four results are caused by a decrease in files fed into the classifier. Beginning with 500 files per group, I dropped the number to 250 each, then 100 each, 25 each, and 5 each. Naturally, the decrease in files resulted in a decrease in overall accuracy.

Reduced Corpora (2000 common, 500 files)

```
Number of spam files: 250
Number of ham files: 250
Accuracy: 0.94
        Precision       Recall          F1
spam          1.000      0.880      0.936
ham           0.893      1.000      0.943
Each fold size: 100
0 0.96
1 0.92
2 0.93
3 0.96
4 0.94
mean accuracy 0.942
usage: python classifySPAM.py <corpus-dir> <limit>
```

Reduced Corpora (2000 common, 200 files)

```
Number of spam files: 100
Number of ham files: 100
Accuracy: 0.775
        Precision       Recall          F1
ham           0.941      0.667      0.780
spam          0.652      0.938      0.769
Each fold size: 40
0 0.85
1 0.875
2 0.925
3 0.975
4 0.775
mean accuracy 0.8800000000000001
usage: python classifySPAM.py <corpus-dir> <limit>
```

Reduced Corpora (2000 common, 50 files)

```
Number of spam files: 25
Number of ham files: 25
Accuracy: 0.9
        Precision       Recall          F1
spam          1.000      0.800      0.889
ham           0.833      1.000      0.909
Each fold size: 10
0 0.8
1 0.7
2 0.4
3 0.8
4 0.9
mean accuracy 0.72
usage: python classifySPAM.py <corpus-dir> <limit>
```

Reduced Corpora (2000 common, 10 files)

```
Number of spam files: 5
Number of ham files: 5
Accuracy: 1.0
        Precision       Recall          F1
ham           1.000      1.000      1.000
Each fold size: 2
0 0.0
1 0.5
2 0.0
3 0.5
4 1.0
mean accuracy 0.4
usage: python classifySPAM.py <corpus-dir> <limit>
```

I changed the number of files back to 500 with the top 2000 frequent words. This time I filtered the feature set to remove any stopwords and included negation, both of which had little to no effect on the accuracy.

NLTK stopwords (2000 common, 500 files)

```
Number of spam files: 500
Number of ham files: 500
0.95
        Precision       Recall          F1
spam          1.000      0.896      0.945
ham           0.912      1.000      0.954
usage: python classifySPAM.py <corpus-dir> <limit>
```

Negation (2000 common, 500 files)

```
Number of spam files: 500
Number of ham files: 500
0.945
        Precision       Recall          F1
spam          1.000      0.897      0.946
ham           0.894      1.000      0.944
usage: python classifySPAM.py <corpus-dir> <limit>
```

I used NLTK's bigram finder to identify the top 500 bigrams (using chi squared) and underwent the same process to train a classifier. This yielded a wider range of precision and recall but overall did not affect accuracy.

Bigrams (top 500 chi squared, 500 files)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.92
        Precision       Recall          F1
ham           0.845      1.000      0.916
spam          1.000      0.858      0.924
Each fold size: 200
0 0.955
1 0.975
2 0.95
3 0.95
4 0.92
mean accuracy 0.95
usage: python classifySPAM.py <corpus-dir> <limit>
```

Below I tested my theory that spam emails often throw in excessive punctuation. I wrote a function to identify word punctuation ratio per piece of text and trained the classifier on that. The results were notably worse than any previous experiment, with an accuracy of around 50%.

Using only the ratio of words to punctuation (500 files) **New Feature**

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.52
        Precision       Recall          F1
ham           0.130      0.591      0.213
spam          0.910      0.511      0.655
Each fold size: 200
0 0.49
1 0.575
2 0.475
3 0.585
4 0.52
mean accuracy 0.529
```

I used part of speech tagging to do the same process. Again, this yielded very little if any overall change in the accuracy.

POS Tagging (500 files)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.945
        Precision       Recall          F1
ham           0.888      1.000      0.941
spam          1.000      0.903      0.949
Each fold size: 200
0 0.955
1 0.945
2 0.96
3 0.96
4 0.945
mean accuracy 0.953
usage: python classifySPAM.py <corpus-dir> <limit>
```

I removed stopwords from the text in addition to using bigrams instead of word frequencies to train the data. Unsurprisingly the accuracy is still steady at 95%.

Stopwords and bigrams combined (2000 common, 500 files, 500 chi squared)

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.95
        Precision       Recall          F1
ham           0.899      1.000      0.947
spam          1.000      0.910      0.953
Each fold size: 200
0 0.96
1 0.945
2 0.945
3 0.96
4 0.95
mean accuracy 0.952
usage: python classifySPAM.py <corpus-dir> <limit>
```

Last, I tried using the word frequency instead of the boolean value to train and test the data. Although this was a successful method, it still was unable to bring the accuracy over 95%.

```
Number of spam files: 500
Number of ham files: 500
Accuracy: 0.945
        Precision       Recall          F1
spam          1.000      0.890      0.942
ham           0.901      1.000      0.948
Each fold size: 200
0 0.965
1 0.935
2 0.95
3 0.945
4 0.945
mean accuracy 0.9479999999999998
usage: python classifySPAM.py <corpus-dir> <limit>
```