Sammy Pardes
IST 664
Final Project
3/30/2021

Introduction

For this assignment, I chose to investigate the Spam/Ham email corpus. Spam filters remove unwanted messages from our inboxes while preserving important information. Spam filters are something that most of us use in our daily communication, although many people may not even realize it. In fact, if your email provider's spam filter is working correctly, it's likely to go unnoticed.

Step 1: Data Processing and Filtering

I ran the processspamham() function to retrieve 1,500 "spam" emails and 1,500 emails classified as "ham". I chose this number because I wanted to analyze an equal amount of spam and ham emails. 1,500 was the maximum amount of spam available to retrieve from the corpus.

The only update I made to the processspamham() file, was adding the line "global emaildocs". I wanted to be sure I could analyze the list of emails outside of the original function, so I set "emaildocs" to be a global variable.

After taking a look at a few unprocessed emails, I decided to filter out some words from the data. I first stored the NLTK default list of English stopwords in the "stopwords" variable. Next, I created a list called "mystop" containing punctuation that I noticed appeared frequently within the emails. I also added a list of first names to "mystop" that I assumed were organization-specific to the email sample. I didn't want the first names of employees to be the main determiner in whether or not an email was considered spam. I added stopwords and mystop together in a new variable called "all_stop". Additionally, I removed the word "subject", since it will likely appear in each email, and "enron" because that is the name of the organization the emails are from. I also only included words at least three characters in length to account for any punctuation I may have missed.

Once I gathered the stop words list, I created a function called filter_stopwords() to remove any word in the stop words list from any email in the emaildocs list. I appended the spam/ham classifier to the tokenized email without the stop words. I stored the filtered email list as a global variable to be analyzed in other cells.

Here is an example of one emails from the corpus, filtered and unfiltered:

Filtered email: (['prozacs', 'meds', 'million', 'people', 'rely', 'wonder', 'drug', 'prozac', 'relax', 'enjoy', 'life', 'wish', 'receive', 'emails', 'please', 'click', 'honor', 'unsubscribe', 'requests', 'immediatly'], 'spam')

Unfiltered email: (['Subject', ':', 'prozacs', 'meds', '30', 'million', 'people', 'now', 'rely', 'on', 'this', 'wonder', 'drug', '!', 'get', 'prozac', 'now', 'relax', 'and', 'enjoy', 'life', '!', '!', '!', '!', '!', '!', 'if', 'you', 'do', 'not', 'wish', 'to', 'receive', 'further', 'emails', 'please', 'click', 'here', '.', 'we', 'honor', 'all', 'unsubscribe', 'requests', 'immediatly', '.'], 'spam')

Step 2: Produce Features

After obtaining the list of filtered emails, I captured only the words from the filtered emails using another for loop. I captured the frequency distribution of the filtered words using the FreqDist() function in NLTK.

Next, I got the top 100 most common words from the filtered emails using most_common(100) on the frequency distribution. I thought 100 words would be sufficient to create word features. In a new variable called freq_words_only, I stored the 100 most frequently occurring filtered words without their exact counts.

Now that I had a list of frequent words, I was able to create a feature function based on frequency, called freq_features(). For each email in the filtered email list, I tokenized the email words and added them to a dictionary called "features". Then, I checked to see if any of the frequent words were in the email with a True/False statement.

Below is a portion of the frequency features of one email after I ran the freq_features() function. Each word contains "is_freq_" followed by the frequent word and then the True/False indicator depending on whether or not the word is in the given email. The spam or ham class was kept as the second element for each email tuple in the email list.

```
({'is_freq_2000': False,
  'is_freq_please': False,
  'is_freq_meter': False,
  'is_freq_deal': False,
  'is_freq_corp': False,
  'is_freq_http': False,
  'is_freq_company': False,
  'is_freq_thanks': False,
  ...
  'is_freq_office': False,
  'is_freq_based': False},
 'spam')
```

I divided the data and created a training set with 70% of the data, and a testing set with the remaining 30%. I ran the NaiveBayesClassifier() function on the training data and used nltk.classify.accuracy() with the aforementioned classifier and the test data. This yielded an accuracy of 91.3%.

Using the confusion matrix and evaluation metrics functions we utilized in labs, I created a confusion matrix and gathered the precision, recall, and f1 of the frequency classifier. This classifier over-predicted spam but, overall, did a good job predicting both spam and ham. I think this would be considered a great spam filter because the accuracy is high and it over-predicts "ham" rather than "spam". This means it's more likely to send spam to an inbox than legitimate emails to the spam folder.

I ran the cross_validation_accuracy function with 10 folds (300 emails per fold) to get an average accuracy of 91.6%. To further examine the output of the classifier, I ran the show_most_informative_features() function.

Interestingly, the word "forwarded" is an excellent indicator of whether or not an email is considered spam. If the frequent word "forwarded" is included in the email, it is 229:1 more likely to be ham. I think this makes a lot of sense because people are usually not forwarding spam to one another. The word "attached" is also an indicator of ham. Typically, spam does not include attachments and people tend to call out when they are attaching a document to an email.

```
Most Informative Features
is_freq_forwarded = True ham : spam = 229.8 : 1.0
is_freq_farmer = True ham : spam = 49.2 : 1.0
is_freq_attached = True ham : spam = 23.7 : 1.0
```

Next, I defined a function to create word features based on parts of speech. For each email, I initialized a counter variable at 0 for the different parts of speech and added +1 for each occurrence of type in the email. This kept track of the number of nouns, verbs, adjectives in every email. I created a True/False feature for each POS, depending on whether or not the total count was over 20.

Just like with the frequency feature, I split the data, ran the Naive Bayes classifier on the training data, and performed 10-fold cross-validation. I received a mean accuracy of 53.4%, significantly lower than with the frequency features. In this iteration, 359 "ham" emails were incorrectly classified as "spam". The POS features alone would make an awful spam filter.

Looking at the most informative features, it seems an email with over 20 adverbs or adjectives is slightly more likely spam, but not by a significant ratio.

```
Most Informative Features
      20+_adjectives = True           spam : ham    =      1.9 : 1.0
         20+_adverbs = True           spam : ham    =      1.9 : 1.0
```

Since the unigram frequency features gave great results, I decided to investigate bigram and trigram frequency. After gathering the bigrams with NLTK's bigrams() function, I extracted the top 100 bigrams from the filtered emails.

Replicating the classifier and cross-validation steps from earlier, the bigrams features yielded a mean accuracy of 57.9%. This model seriously over-predicted spam and the important features did not produce any meaningful results. I would certainly not recommend using this as a spam filter either.

The trigram features function also did not produce great results. The average accuracy was only 47.1%. In this instance, it predicted every email as "ham", which would be a terrible choice in a filter since it would keep all the emails in the inbox.

Finally, I decided to combine the unigram frequency, part of speech, and bigram features into one feature function called all_features(). The mean accuracy was very similar to accuracy of the frequency feature set at nearly 92%. This makes a lot of sense because after combining the features, all of the most important were the frequency features.

Step 3: Classification Experiments

To experiment, I decided to run the unigram frequency feature function on the unfiltered emaildocs list to see if the accuracy would improve. I obtained the unfiltered words from the unfiltered emails and ran the freq_features() function once more. The mean accuracy decreased slightly to about 88.6% and the most important features were different. Here, "hou", "cc", and "ect" were strong features of "ham", while punctuation like "!", "%", and "|" were indicators of "spam". I think this makes sense because people often call out when they cc others and the spam messages seemed to contain a lot of extraneous punctuation marks.

```
Most Informative Features
         is_freq_hou = True            ham : spam   =    211.1 : 1.0
         is_freq_ect = True            ham : spam   =    133.1 : 1.0
          is_freq_cc = True            ham : spam   =    123.2 : 1.0
          is_freq_pm = True            ham : spam   =     30.9 : 1.0
           is_freq_| = True           spam : ham    =     25.7 : 1.0
```

Additionally, I ran the NaiveBayes classifier from SciKit on the frequency feature function. The weighted average of all metrics went down slightly compared to the NLTK classifier at 88%.

Conclusion

It’s more important to correctly capture ham compared to spam. If some spam emails get through, you can easily delete the email. However, If ham is falsely flagged as spam, the user may miss something important. It seems the best and most accurate way to determine whether or not an email is spam through unigram frequency features. Although, 92% is a solid number, this classifier would not work in the real world. Misidentifying 8% of emails could lead to a user missing a very important message or having a cluttered inbox.

If I were to do this assignment again, I would like to experiment with even more features to see if I could get the accuracy above 95%. It would be interesting to try using regular expressions to see if the amount of punctuation could be a useful feature. I would also like to try my current frequency classifier on a different set of emails. “Forwarded” and “attached” are strong identifiers for this set of emails, but I would like to know if that is a universal observation.