CS5691 Assignment3: Spam v. Ham

Kartik Bharadwaj (CS20S020)

December 31, 2020

Note: I have attached a Colab notebook file **Assignment3.ipynb** containing my code. This file can be directly run on a Google Colab session using a Gmail account. A Colab notebook is used due to large number of datasets involved and higher CPU compute requirements. It is recommended that the Colab notebook be used for easy evaluation of the code. Please note to add **stopwords.txt** and **frequency_dictionary_en_82_765.txt** while running the session. The code will create a 'text' folder where test emails can be dragged and dropped in the same session.

In addition, I have also attached the python file for the same. If you plan to use the python file, additional python libraries have to be installed. These libraries are mentioned in the initial lines of **Assignment3.py**. Dataset can be found here.

1 Introduction

The goal of this assignment is to design a spam classifier from scratch. Any publicly available training data which doesn't have any copyright restrictions can be used. For this binary classification, any set of feature extraction can be performed to increase the model's performance. Since I don't have access to the test set (except for the two emails) or its distribution, I have to carefully choose our preprocessing, feature extraction, and algorithms.

2 Data & Preprocessing

There are plenty of datasets consisting of spam and ham emails. In particular, I have chosen three of them to train the spam classifier: Enron spam dataset, Ling spam dataset, and Spam assassin datasets. After removing duplicate emails, the count is as follows:

Datasets		
	#Ham Emails	#Spam Emails
Enron	15910	14584
Lingspam	2408	468
Spamassassin	6747	2284

All the 3 datasets contain raw emails. Specifically, in the Spamassassin dataset, some emails are in raw html format. To get the main text and body of an email from Enron and Lingspam dataset, I use regex expressions. To process Spamassassin dataset, I use regex expressions and the **email** and **html** library available in python. Few of the preprocessing steps to clean the email include converting all:

1. http/https phrases to 'httplink'.
2. email address phrases to 'emailaddr'.

3. IP addresses to 'ipaddr'.
4. image extensions to 'imgext'.
5. extra spaces to a single space.

Note that such preprocessing steps are common to all emails (train and test).

3 Feature Extraction

To train a spam classifier, an input dataset is formed through the bag-of-words (BoW) approach. The steps are as follows:

1. Clean the emails as mentioned in the pre-processing steps.
2. Remove all stopwords such as: $\{$the, an, you, have, etc.$\}$. Not all emails will be cleaned 100%. After cleanup, there can be words having length less than 2 or greater than 15 which makes no sense. Hence, such words are removed.
3. Using **symspellpy** library, spell check is performed only on ham emails and not on spam emails. This is done to ensure that spam words don't convert into ham words. However, spell check is performed on ham emails as it is less likely for them to get converted spam words.
4. Using Snowball stemmer available in **nltk** library, words are stemmed in all of our emails. This process will give the root form of each word. Next, stemmed words are lemmatized to fix incorrectly stemmed word.
5. Next, we created two bag-of-words, one for spam emails, another for non-spam emails. We take the top $\frac{k}{2}$ words from each bag and combine them into a single bag of k words. This is to ensure that we have an equal number of spam and non-spam words. If we directly take the top k words from our emails, it may happen that our BoW has more non-spam words than spam words.
6. Using this combined BoW, pre-processed input emails are tokenized. Laplace smoothing is performed by adding two emails which contain all words, one for spam label and one for ham label.
7. For test emails, cleanup is done, stopwords are removed, and tokenization is performed using train BoW.

4 Experiments

In ML, the basic idea is to collect all data, split into train and test set, run the algorithm on train set, and predict on the test set to get the actual performance. Most often, spam classifiers are designed using this idea. However, my hypothesis is that spam classifiers can suffer from concept drift. For instance, the distribution of all spam emails will most often contain topics such as 'money', 'insurance', 'lottery', etc. In this case, test spam emails will most likely come from this distribution. However, for non-spam (or ham) emails, this is not the case. This is because distribution of non-spam emails can range from college ham emails to health insurance company's ham emails.

Here, we are facing a similar problem. We don't know how our test distribution looks like. To design a classifier using the 'basic idea' mentioned above would be sub-optimal. Hence, we take emails from Enron and Spamassassin dataset as our training set and emails from Lingspam as our test set. This makes our evaluation criteria more fairer.

A number of experiments were run to find the best algorithm. Parameters on which experiments were run are as follows:

1. No spell check
2. Word stemming
3. Word Lemming
4. Remove low frequency words

Experiments were run based on a combination of these parameters. After running the experiments, we find that Bernoulli Naive Bayes performs significantly better than Logistic Regression, and LinearSVM. Hence, Bernoulli Naive Bayes is used.