Assignment 2

Ryan

December 14, 2020

Abstract

In this assignment, I attempt to code a spam filter using Naïve Bayes. I will include a list of my decisions and rationale, my results, and a final summary of what I have learned from this assignment.

Introduction

We were given a csv file containing 250 emails, each labeled as either spam or ham (ham being not spam). Our assignment is to split up these emails between training emails and testing emails, write an algorithm to learn from our training set, and to attempt to correctly guess the type of emails from our test set.

1 Initial Setup

For the initial setup, I downloaded the csv file from github. The URL was given to us in our assignment pdf but I will also post it here:

`https://github.com/tasdikrahman/datasets/tree/master/email/csv`

I also decided to use spyder to code and test my program. I chose spyder because it is an easy environment to code in and the background theme is easier for my eyes to handle over long periods of coding.

2 Decision Points

2.1 Decision 1

My first decision was how to split up the words in each email. I looked at my options using a delimiter and decided to first remove non-alphabetical symbols. I realize that some of these symbols might be indicators of spam but for now I want to just focus on each word in the emails. The characters I chose to delimit on are as follows:

[, > < + = [{ ( ) }] * & ^ % $. # @ ! \ : ; ' _ / | ?]

I used the regular expression (re) library in python to accomplish this delimiting and my results indicate it was successful. An example email with delimited words appears like this:

['-1', 'Email', 'marketing', 'works', 'There', 's', 'no', 'way', 'around', 'it', '\\nNo', 'other', 'medium', 'let' 's', 'you', 'share', 'your', 'offer', 'with', 'more', '\\npeople']

The -1 indicates that this email is spam and the \\n in two of the words comes from a new line call in the html code. I found no simple way to deal with this and from my testing I had greater accuracy leaving it in so I decided to leave it alone.

2.2 Decision 2

I initially thought about creating separate arrays for each email type, one for spam and the other for ham. I coded it out but realized that it would be a little bit messier when trying to separate the test set from the training set, so I decided to switch to a single array for all of the emails and then created two arrays to hold the training set and the test set.

I added a variable, percent_training, to set the percentage of training emails I wanted. I then used that number to separate out the training emails from the test emails. I also took this time to grab the number of training emails that contained spam and ham for later use.

2.3 Decision 3

Now that the emails have been separated into a training set and a test set, I needed to decide on how I was going to choose the words I wanted the algorithm to learn with. My first thought was to use a dictionary to hold the words and the number of times they occurred. I didn't have a clear idea of how the algorithm was going to learn from these words at this time so I only focused on trying to make it work with a dictionary.

The problem I ran into was needing to grab the words and their probabilites in a known order when performing my analysis of the test set. I could have grabbed the values and calculated the probabilities each time I needed them but that turned into a lot of messy code. After our professor provided more insight into the problem, I started work on a different solution. My first step was to grab all the words from every email. I created two dictionaries, one to hold all the words occurring in the spam emails and the other to hold all the words in the ham emails. Just to note, these words are from the emails in our training set. I then set conditions to only add words that contained alphabetical letters and words that are longer than two characters. My reasoning for these two conditions were to make sure that I didn't include numbers in my data set and to trim off the words that had two or less characters. These words are not very likely to be indicative of spam or ham emails so I wanted to exclude them early on.

My next step was to create two lists to hold my spam and ham words. I searched through the words in both my spam and ham dictionaries, added more conditions to further filter the words I wanted, and then added them to my lists. My first condition was to only add words that occurred in both spam and ham emails. I felt that by using words that occurred in both, I would be less likely to have some words heavily sway my results if they suddenly appeared in new emails of the opposite email type. This way I would also not have to worry about mathematical issues when using the adjusted naïve bayes algorithm provided by our instructor.

$$ln\frac{p(C_{spam} \mid x)}{p(C_{ham} \mid x)} = ln\frac{p(C_{spam})}{p(C_{ham})} + \sum_{i=1}^{n} ln\frac{p(x_i \mid C_{spam})}{p(x_i \mid C_{ham})}$$

By only using words that occur in both, I don't have to worry about division by zero if the word occurs zero times in ham emails or taking the log of zero if the word occurs zero times in spam emails.

My next condition was to only grab words that occurred more often in the current list's email type. For example, if I am adding a word to my spam words list, I only want words that occur more often in spam emails than ham emails. This way my list only contains words that are more likely to occur in the list's email type. My next condition was to reference a list of words I specifically wanted my algorithm to ignore and to exclude them from my spam or ham word list. This list was added near the end after I managed to get the rest of the program to work. By adding words to this list I was able to fine tune my algorithm and increase its accuracy by roughly 34%. This improvement came from removing outlier words and words that are unlikely to indicate spam or ham emails, such as you, our, the, and, but, etc. My last condition was to ignore words that occurred less than two times in the word's respective email type. This number was obtained from trial and error. It initially started out at twenty, but over time I adjusted and readjusted it until I obtained consistent and good results. This optimization added a roughly 15% increase in accuracy.

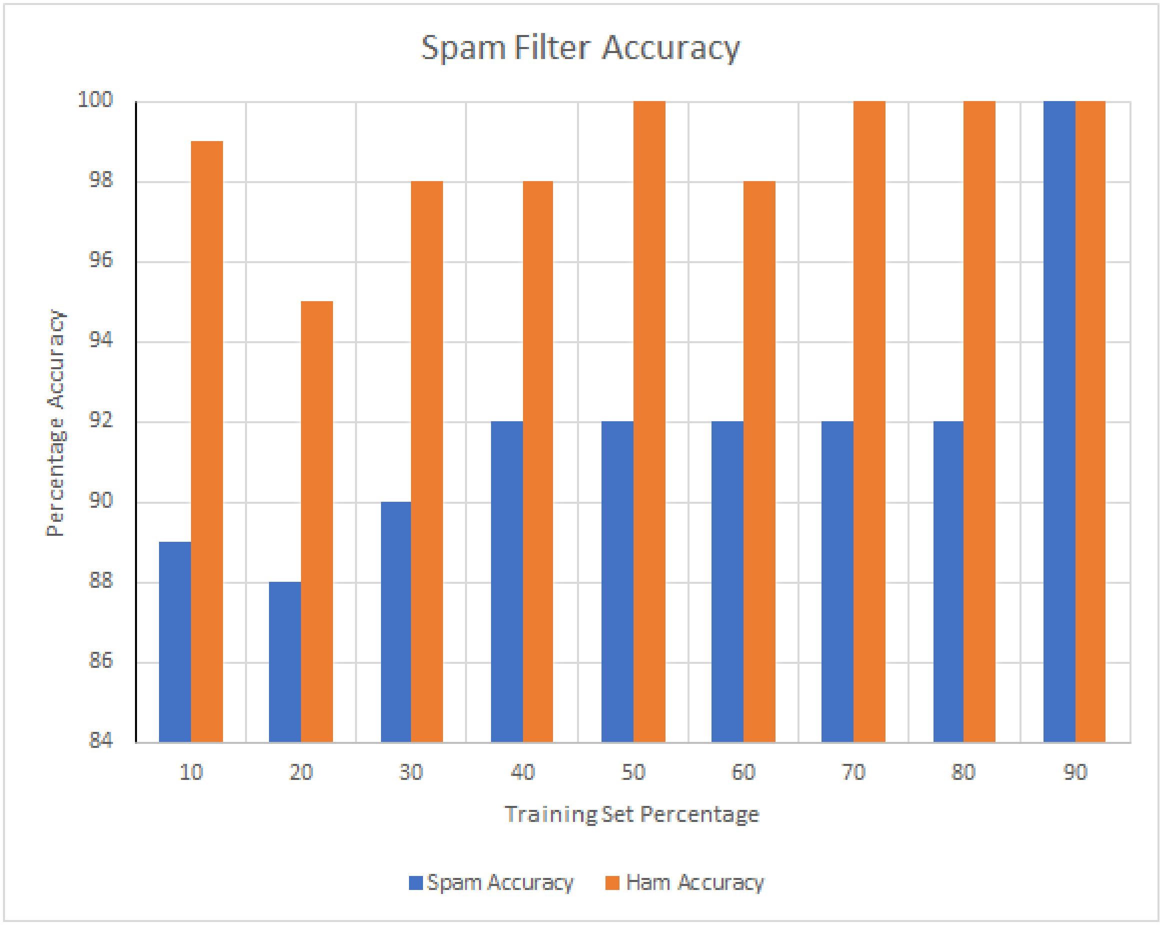
After running these filters, I took the length of the smallest list between my spam words list and ham words list and added that amount of words from each list to a final training words list. For example, if my spam words list was of length 400 and my ham words list was of length 300, I would grab the number 300 and choose the first 300 words from my spam words list and 300 words from my ham words list and add them both to a final list of training words. This would total up to 600 training words. I did this to ensure an equal number of more probable spam words and more probable ham words.

2.4 Decision 4

After obtaining all of my training words, I then needed to decide how to approach the algorithm to calculate the probability of each email type from my test set. I wasn't as clear on this to begin with but our instructor gave us more guidance on this which helped me to better understand where to go. I initially thought I needed to determine the percentage that a word occurred in spam and ham out of all occurences of the word, but after class I learned I needed to determine the percentage of how often each word occurred in their respective email types. After that, it was just a matter of coding the algorithm and testing my approach.

3 Results

My initial results were not stellar but after fine tuning and adjusting certain values I was able to obtain an accuracy of 92% with a training size of 40-80% of the emails. I decided to test my algorithm on different sized training sets starting from 10% all the way up to 90% training sizes. The results are given in the graph below:



The accuracy of the program increased as the training size increased though there was not much change between 40% and 80%. I feel that there are further optimizations that can be made but overall I am happy with the results.

Confusion Matrices: (Training set size given in percent)

$$\begin{bmatrix} TrueSpam & FalseSpam \\ FalseHam & TrueHam \end{bmatrix}$$

$$10\% : \begin{bmatrix} 111 & 13 \\ 1 & 100 \end{bmatrix} \quad 20\% : \begin{bmatrix} 96 & 13 \\ 4 & 87 \end{bmatrix} \quad 30\% : \begin{bmatrix} 85 & 9 \\ 1 & 80 \end{bmatrix}$$

$$40\% : \begin{bmatrix} 72 & 6 \\ 1 & 71 \end{bmatrix} \quad 50\% : \begin{bmatrix} 61 & 5 \\ 0 & 59 \end{bmatrix} \quad 60\% : \begin{bmatrix} 46 & 4 \\ 1 & 49 \end{bmatrix}$$

$$70\% : \begin{bmatrix} 35 & 3 \\ 0 & 37 \end{bmatrix} \quad 80\% : \begin{bmatrix} 24 & 2 \\ 0 & 24 \end{bmatrix} \quad 90\% : \begin{bmatrix} 12 & 0 \\ 0 & 13 \end{bmatrix}$$

Though not perfect, the naiv̈e baye's approach is surprisingly effective at spotting spam emails with the effectiveness increasing as our training set size increases.

4 What I learned

I learned how to create a spam filter using naiv̈e bayes and how to write a basic report using Latex. I have also increased my knowledge of python, mainly in the area of list comprehension and data structures. I have also gained a better understanding of machine learning by putting into practice some of the ideas and techniques we have covered in class such as probability analysis, classifiers, and the stretching of my thinking in order to come up with solutions to a problem. I also learned that it is very important to keep a running log of my thought process and decisions in order to accurately recount my decision points. This is something that I will keep in mind for future projects.