Jack Wagner

11/17/19

COSC 16

Richard Granger

SMS Spam Detection

I used the Naive Bayes Classifier to detect whether SMS messages were considered to be *ham* (i.e. legit messages) or *spam*. I used the "SMS Spam Collection Data Set" which consisted of 5574 different SMS messages and whether they were spam or ham. I used 4676 of these messages to train my program then the remaining 898 messages to test my program.

The first step of my implementation was taking the training file line by line and splitting apart the different words. Then I made a dictionary of all the words in the training file. Each word was a key and pointed to a 2-item list in which the first item was the number of times where that word appeared in spam messages, and the second item was the number of times where that word appeared in ham messages.

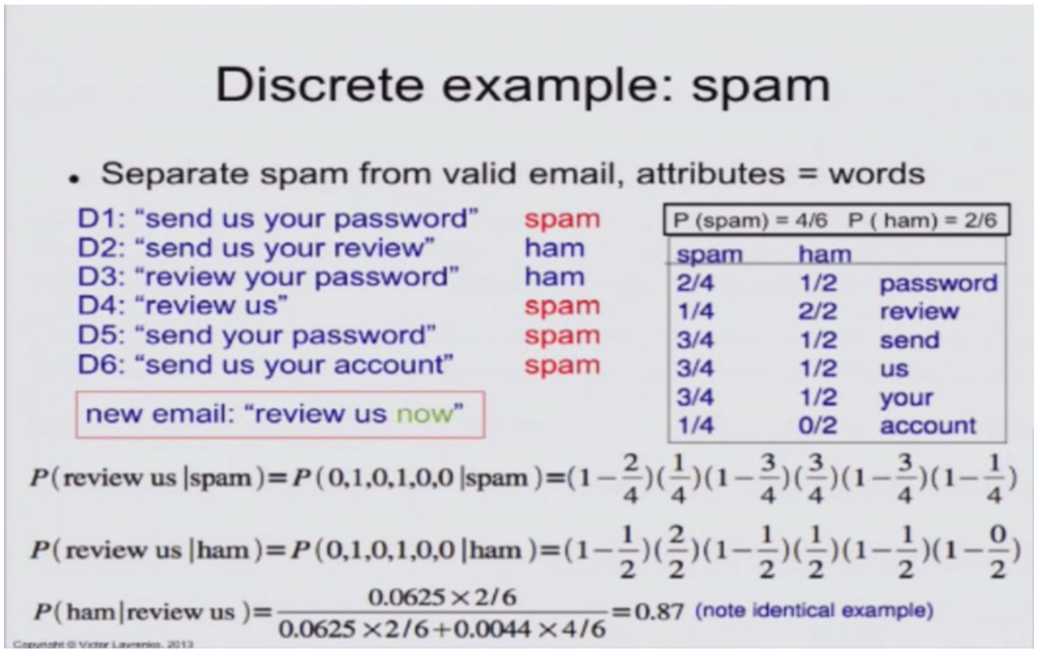
Next, to test against the training data and dictionary I created, I took the testing file and made a dictionary of each line's words. I then called my main function, *find_prob_of_input(dictionary)* on each line. If the output of that function matched the expected output of the training line (whether it was supposed to be ham or spam) then that line was considered to be interpreted correctly. Using 898 test lines, my testing predicted 338 correct, at a 37.6% success rate. I will discuss different possibilities of error in a moment.

The function *find_prob_of_input(dictionary)* is what did the Naive Bayes Classification. The following formula was used:

$$Pr[ham|\text{specific words in test line}] = \frac{Pr[ham \cap \text{specific words in test line}]}{Pr[\text{specific words in test line}]}$$

$$Pr[spam|\text{specific words in test line}] = \frac{Pr[spam \cap \text{specific words in test line}]}{Pr[\text{specific words in test line}]}$$

To do this, I had to consider every word from the training dictionary. If a word in the training dictionary did not appear in the test line, then its complement probability was used. However, if the word in the training dictionary *did* appear in the test line, then its corresponding spam or ham probability was used. The sample problem I used to teach myself this concept is from https://www.youtube.com/watch?v=8aZNAmWKGfs and appears here:



Discrete example: spam

- Separate spam from valid email, attributes = words

D1: "send us your password"	spam
D2: "send us your review"	ham
D3: "review your password"	ham
D4: "review us"	spam
D5: "send your password"	spam
D6: "send us your account"	spam

new email: "review us now"

P (spam) = 4/6   P (ham) = 2/6

spam	ham	
2/4	1/2	password
1/4	2/2	review
3/4	1/2	send
3/4	1/2	us
3/4	1/2	your
1/4	0/2	account

$$P(\text{review us}\,|\,\text{spam}) = P(0,1,0,1,0,0\,|\,\text{spam}) = (1-\frac{2}{4})(\frac{1}{4})(1-\frac{3}{4})(\frac{3}{4})(1-\frac{3}{4})(1-\frac{1}{4})$$

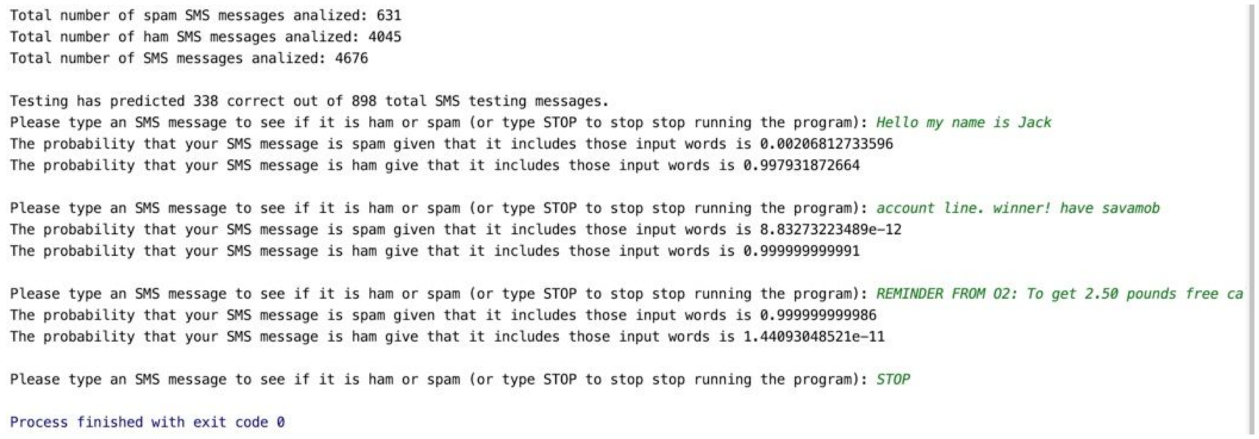
$$P(\text{review us}\,|\,\text{ham}) = P(0,1,0,1,0,0\,|\,\text{ham}) = (1-\frac{1}{2})(\frac{2}{2})(1-\frac{1}{2})(\frac{1}{2})(1-\frac{1}{2})(1-\frac{0}{2})$$

$$P(\text{ham}\,|\,\text{review us}) = \frac{0.0625 \times 2/6}{0.0625 \times 2/6 + 0.0044 \times 4/6} = 0.87$$ (note identical example)

*** The 1s mean that "review" and "us" are in the training set* <u>*and*</u> *in the test sentence.. The 0s mean that "password", "send", "your", and "account" did* <u>*not*</u> *appear in the test sentence.***

One of the things that I debated heavily when creating my program was the use of punctuation. When I split apart the lines in both training and testing, I ultimately decided to keep the punctuation because spam and ham messages followed different trends in their punctuation. For example, as seen from the *SMSSpamCollection* data file and also my output *words_in_spam_SMS*, punctuation such as "=", "!", and "?" had unique roles in spam and ham messages. I decided to keep the punctuation in my training because of this.

I think that one of the main problems with this method was indeed the punctuation. The punctuation varied widely in the spam messages, but I did not want to sacrifice the uniqueness of words with their punctuation in spam messages to make my program more general. Also, however, the length of the messages also played a role in the output. I created a method in which the user can type their own string to see if that would be considered ham or spam based off my training, and I noticed from many of my own testing messages that shorter messages had a higher probability of being considered ham messages. Even if I typed an input string of multiple words words that appeared in spam messages, the string was still considered to be spam. Some example outputs are shown here:



```
Total number of spam SMS messages analized: 631
Total number of ham SMS messages analized: 4045
Total number of SMS messages analized: 4676

Testing has predicted 338 correct out of 898 total SMS testing messages.
Please type an SMS message to see if it is ham or spam (or type STOP to stop stop running the program): Hello my name is Jack
The probability that your SMS message is spam given that it includes those input words is 0.00206812733596
The probability that your SMS message is ham give that it includes those input words is 0.997931872664

Please type an SMS message to see if it is ham or spam (or type STOP to stop stop running the program): account line. winner! have savamob
The probability that your SMS message is spam given that it includes those input words is 8.83273223489e-12
The probability that your SMS message is ham give that it includes those input words is 0.999999999991

Please type an SMS message to see if it is ham or spam (or type STOP to stop stop running the program): REMINDER FROM O2: To get 2.50 pounds free ca
The probability that your SMS message is spam given that it includes those input words is 0.999999999986
The probability that your SMS message is ham give that it includes those input words is 1.44093048521e-11

Please type an SMS message to see if it is ham or spam (or type STOP to stop stop running the program): STOP

Process finished with exit code 0
```

The last message I inputted is a direct copy from line 893 of my testing file. It was correctly identified as a spam SMS message.

I do believe that this could be improved if a smaller dataset was used and if the problem with the punctuation could be handled more carefully. I think that, if done correctly, it would make it more accurate without losing the unique punctuation from some spam messages. This program could also be used on other data as well. For example, if there was a dataset of words spoken from two different people, then this program could use the frequency of words between the two people to predict if a certain testing phrase was said by one or the other.

RESOURCES

"How to Clean Text for Machine Learning in Python?":
https://machinelearningmastery.com/clean-text-machine-learning-python/

Naive Bayes for Spam Detection YouTube:
https://www.youtube.com/watch?v=8aZNAmWKGfs

"Python Numbers, Type Conversion, and Mathematics":
https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection#

SMS Spam Collection Data Set: https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection#

"Why Doesn't This Division Work in Python?":
https://stackoverflow.com/questions/1787249/why-doesnt-this-division-work-in-python

"3 Ways to Write Text to a File in Python?":
https://cmdlinetips.com/2012/09/three-ways-to-write-text-to-a-file-in-python/