Naïve Bayes Spam Filter Report

Siddhant Shettiwar
1001879146

In this project we will build a spam classifier from scratch for SMS messages using Multinomial Naive Bayes which is known to suit situations where data can be turned into counts, such as word counts in text.

The dataset used for this project; **SMS Spam Collection Data Set** was downloaded from UCI Machine learning repository.
More details about dataset can be found at this link :
https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

-For this project I have used google Google Collaboratory.
Download the dataset from the above link and upload it to Google Collaboratory.

Steps:

1) Define libraries NumPy, pandas and import the dataset.
2) Now randomize the dataset so that the results can be reproducible, and split the dataset into 80:20 ratio, 80 for training and 20 for testing.
3) Next, we clean the dataset and created a vocabulary list. (all unique words present in the dataset). To do so I separated all the sentences with respect to word and used set() function to remove the duplicates.

   For cleaning I removed punctuations and converted the words to lowercase.(“APPLES” and “apples” should be considered as a same word). Using set function will ensure that no duplicates are appended to the vocabulary list

4) **DataFrame conversion:** Right now, our DataFrame is in the below format

	Label	SMS
0	Spam	SECRET PRIZE! CLAIM SECRET PRIZE NOW!!
1	Ham	Coming to my secret party?
2	Spam	Winner! Claim secret prize now!

We need to convert the above DataFrame format into the one which we can use as input. (given below)

	Label	Secret	Prize	Claim	Now	Coming	To	My	Party	winner
0	Spam	2	2	1	1	0	0	0	0	0
1	Ham	1	0	0	0	1	1	1	1	0
2	Spam	1	1	1	1	0	0	0	0	1

As we can see above:

Say for index 0 SMS: there is SECRET occurring twice and hence in then above table we have 2 under secret. Similarly other values represent the count of their occurrence in their respect SMS's

5) Now that we have a proper input format we will create a spam filter(Classifier).
   To classify whether the SMS is spam or ham we need to calculate the following probabilities mentioned below :

Classifier:

$P(Spam|w_1,w_2,w_3,....w_n) = P(Spam) . \prod^{n}_{i=1} P(w_i|Spam)$
$P(Ham|w_1,w_2,w_3,....w_n) = P(Ham) . \prod^{n}_{i=1} P(w_i|Ham)$

To calculate $P(w_i|Spam)$ and $P(w_i|Ham)$ for each SMS we use following equations:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + \alpha}{N_{Spam} + (\alpha \times N_{vocabulary})}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + \alpha}{N_{Ham} + (\alpha \times N_{vocabulary})}$$

here,
$N_{w_i|Spam}$ = the number of times the word w_i occurs in spam messages.
$N_{w_i|Ham}$ = the number of times the word w_i occurs in ham messages.
N_{Spam} = the total number of words in spam messages
N_{Ham} = the total number of words in ham messages
$N_{vocabulary}$ = the total number of words in the vocabulary.
α = 1 (Laplace smoothing parameter which is used to avoid zero-frequency problem)

We use the newly generated DataFrame in step 4 and isolate Spam and Ham messages after which,
we calculate constants:
P(Spam), P(Ham), N_{Spam}, N_{Ham}, $N_{vocabulary}$

Next, we calculate parameters: $P(w_i|Spam)$, $P(w_i|Ham)$

6) After calculating these prior probabilities, we define a classifier function which takes SMS as an input parameter.
Now to classify the SMS input we compare P (Spam|w_1, w_2, w_3,….w_n) and P(Ham|w_1, w_2,w_3,….w_n),

whichever comes out to be higher than the other one, that is the final class :

```
if p_ham_given_message > p_spam_given_message:
  return 'ham'
elif p_spam_given_message > p_ham_given_message:
  return 'spam'
else:
  return 'needs human classification'
```

Where “p_spam_given_message” and “p_ham_given_message” is P(Spam|w_1, w_2, w_3,….w_n) and P(Ham|w_1, w_2,w_3,….w_n) respectively.

7) Now to test our classifier we input a prelabelled spam SMS to check whether it is correctly classified by the classifier:

```
29 classify('Winner. Claim your free reward today. Winner')

P(Spam|message): 5.0394923646861 31e-21
P(Ham|message): 3.345594114328508e-28
Label: Spam
```

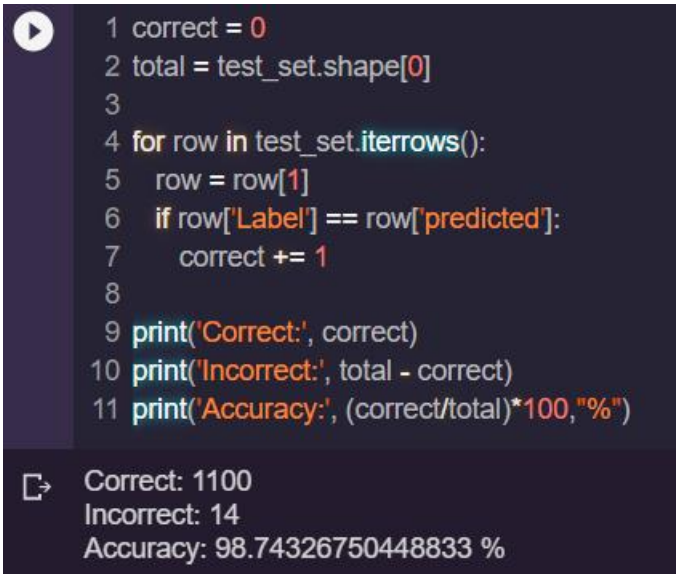
Above as we can see the SMS has higher Spam probability and hence was classified as Spam.

Similarly for a Ham SMS :

```
1 classify("Sounds good Rob, see you there tonight")

P(Spam|message): 4.799483208051815e-26
P(Ham|message): 1.2056621248378029e-21
Label: Ham
```

8) Now that these two messages were correctly classified, we proceed to classify remaining dataset , ie : Test dataset and calculate the correct and incorrect classification and ultimately the accuracy :



```
correct = 0
total = test_set.shape[0]

for row in test_set.iterrows():
    row = row[1]
    if row['Label'] == row['predicted']:
        correct += 1

print('Correct:', correct)
print('Incorrect:', total - correct)
print('Accuracy:', (correct/total)*100,"%")
```

```
Correct: 1100
Incorrect: 14
Accuracy: 98.74326750448833 %
```

As we can see above, we receive an accuracy of 98.74% with few incorrect classifications.