Project: **Machine Learning Programming Project 2 Part 2**
Team Members: **Debit Paudel, Kushal Dahal**
We have used github for the collaboration.
Github Link: https://github.com/debit7/Spam_Message_Detection

Starting to working on the dataset of messages where there were two attributes: Classifier, Messages. All of the messages are classified as spam and ham in the classifier. Initially we have read the data using open() and panda to build a dataframe.

In [161…

```
import pandas as pd
with open('spam.data') as f:
        lst = []
        for ele in f:
            line = ele.replace('\n','').split('\t')

            lst.append(line)
Headers=['Classifier','Messages']
df = pd.DataFrame(lst,columns =Headers)
df.head()
```

Out[161…

	Classifier	Messages
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

After building a dataframe, we splitted the dataset into training and testing dataset 75:25 ratio. Pre processing of training dataset plays a important role in the accuracy of the model. We started to remove the punctuations, created stop words dictionary and removed those words from our training dataset. Working on splitting the single message into words, we listed all the unique words in a list:wordlist.

In [162…

```
#randomly splitting data in training and testing data set
training = df.sample(frac=0.75)
testing = df.drop(training.index)
#print(training)
#print(test)
```

In [163…

```
#removing punctuations from the messages and converting the upper cases to lower case
punctuations='''€˜%^&"\,!*_~)(-[};:]{'<#£$>./?@+'''
stop_words=["i","da","we","ur" ,"u","am","me", "my", "myself", "we", "our", "ours", "ou
wordlist=[]
for message in training['Messages']:

    #print(message,'\n')
    for alpha in message:
        if alpha in punctuations:
            message = message.replace(alpha, " ")
        #for lower case
        message=message.lower()

        #for stop words
        message=message.split()
        #print(message)
        for word in message:
            if word in stop_words:
                #print(word,'\n')
                message.remove(word)
            wordlist.append(word)
    wordlist=list(set(wordlist))
    Vocabulary = len(wordlist)
```

With an completion of building a wordlist with distinct words from all the messages in training set, we built a new training table containing each word of the wordlist as an column resulting 6218 columns. After building this table structure, we counted the frequency of each word in each message for all the training rows and the frequency is stored in the word attribute column.

In [164…

```
#calculating frequency for each word in a message
word_counts_per_Messages=[
    [row[1].count(word) for word in wordlist]
    for _, row in training.iterrows()]
df_wordcount=pd.DataFrame(word_counts_per_Messages,columns=wordlist)
training = pd.concat([training.reset_index(), df_wordcount], axis=1).iloc[:,1:]
```

While working in the calculations for naive bayes, initially we calculated the classifiers probability based on training set. We created list for spam and ham classifiers. Calculated the total number of word positions for spam and ham classifiers. After the calculation of number of times word occurs in the particular classifier,total number of word positions in the particular classifier, we built different functions for each classifier to estimate the word occurrence of that particular message type. While building these functions, we felt there can be words that did not appear in training set but may appear in the test set. Our model may assign a probability of 0. To prevent this problem that may appear in future, we added 1 in the numerator and the length of the wordlist i.e |Vocabulary| in the denominator.

In [173…

```
spam_probability = training['Classifier'].value_counts()['spam'] / len(training)
ham_probability = training['Classifier'].value_counts()['ham'] / len(training)
spam_list=training['Classifier'] == 'spam'
ham_list=training['Classifier'] == 'ham'
spam = training[spam_list]
ham = training[ham_list]
spam_n = training.loc[spam_list, 'Messages'].apply(len).sum()
ham_n = training.loc[ham_list, 'Messages'].apply(len).sum()
```

In [166…

```
def probability_word_spam(word):
        return (spam[word].sum() + 1) / (spam_n + Vocabulary) if word in training.colum
```

```
def probability_word_ham(word):
        return (ham[word].sum() + 1) / (ham_n + Vocabulary) if word in training.column
```

With an aim to predict the classifier of all the test data sets, we created a dynamic function to classify the message as spam and ham. Here while we were multiplying each small probabilities, the calculation may result an arithmetic underflow. So to prevent this kind of issue, we implemented logarithms base 10 importing math library avoiding the underflow.

In [167…

```
import math as mt
def naive_classifier(Message):
    prob_spam = spam_probability
    prob_ham = ham_probability
    for word in Message:
        prob_spam += mt.log(probability_word_spam(word),10)
        prob_ham += mt.log(probability_word_ham(word),10)
    prob_spam=10**prob_spam
    prob_ham=10**prob_ham
    return 'ham' if prob_ham > prob_spam else 'spam'
```

In [168…

```
testing['Predicted_label'] = testing['Messages'].apply(naive_classifier)
Accuracy = (testing['Predicted_label'] == testing['Classifier']).sum() / len(testing) *
print("Accuracy of this model:",Accuracy)
```

```
Accuracy of this model: 96.12625538020086
```

While predicting the messages creating new column: Predicted_label, we also calculated the accuracy of our model based on the prediction of testing dataset and got the accuracy of 96%.

Approach :

We followed the approach of bayes algorithm considering each word in the distinct wordlist of training set as an attribute and estimated the word occurrence of that particular message type.

Problems

We faced several problems while working on this project. We faced problems to create attributes for each word in the training set and to calculate the frequency. We tried nested loop to do this task. But it was time consuming. So we found out that inline forloop and pandas iterrows() can do the same task in short about of time.

Experiment

Experiment 1:

While experimenting the classifying function without implementing logarithm base 10, multiplying each small probabilities result an arithmetic underflow and the accuracy was 94%. But after the implementation of logarithm base 10 the accuracy increased upto 96%

Experiment 2:

Another experiment we conducted was we did not remove punctuations and stop words intially. The accuracy was not good i.e 88%. So we removed punctuations and stop words. The accuracy was raised to 96%.

Experiment 3:

Another experiment we conducted was we separated training and testing datasets as 60:40 proportion and got the accuracy of 94%. While comparing the accuracy with the experiment where the dataset was classified into 75-25 for training and testing sets,the accuracy was raised to 96%.

In [189…

```
#Experiment 1: without using logarithms
training = df.sample(frac=0.75)
testing = df.drop(training.index)
wordlist=[]
for message in training['Messages']:

    for alpha in message:
        if alpha in punctuations:
            message = message.replace(alpha, " ")
    #for lower case
    message=message.lower()

    #for stop words
    message=message.split()
    #print(message)
    for word in message:
        if word in stop_words:
            #print(word,'\n')
            message.remove(word)
        wordlist.append(word)
wordlist=list(set(wordlist))
Vocabulary = len(wordlist)
word_counts_per_Messages=[
    [row[1].count(word) for word in wordlist]
    for _, row in training.iterrows()]
df_wordcount=pd.DataFrame(word_counts_per_Messages,columns=wordlist)
training = pd.concat([training.reset_index(), df_wordcount], axis=1).iloc[:,1:]
spam_probability = training['Classifier'].value_counts()['spam'] / len(training)
ham_probability = training['Classifier'].value_counts()['ham'] / len(training)
spam_list=training['Classifier'] == 'spam'
ham_list=training['Classifier'] == 'ham'
spam = training[spam_list]
ham = training[ham_list]
spam_n = training.loc[spam_list, 'Messages'].apply(len).sum()
ham_n = training.loc[ham_list, 'Messages'].apply(len).sum()
def probability_word_spam(word):
        return (spam[word].sum() + 1) / (spam_n + Vocabulary) if word in training.colum

def probability_word_ham(word):
        return (ham[word].sum() + 1) / (ham_n + Vocabulary) if word in training.columns

def naive_classifier(Message):
    prob_spam = spam_probability
    prob_ham = ham_probability
    for word in Message:
        prob_spam *= probability_word_spam(word)
        prob_ham *=probability_word_ham(word)
    return 'ham' if prob_ham > prob_spam else 'spam'
testing['Predicted_label'] = testing['Messages'].apply(naive_classifier)
Accuracy = (testing['Predicted_label'] == testing['Classifier']).sum() / len(testing) *
print("Accuracy of this model without using Logarithm:",Accuracy)
```

```
Accuracy of this model without using Logarithm: 94.33285509325682
```

In [177…

```
#Experiment 2: Without cleaning the punctuation and stop words
training = df.sample(frac=0.75)
testing = df.drop(training.index)
wordlist=[]
for message in training['Messages']:
    message=message.split()
    for word in message:
        wordlist.append(word)
wordlist=list(set(wordlist))
Vocabulary = len(wordlist)
word_counts_per_Messages=[
    [row[1].count(word) for word in wordlist]
    for _, row in training.iterrows()]
df_wordcount=pd.DataFrame(word_counts_per_Messages,columns=wordlist)
training = pd.concat([training.reset_index(), df_wordcount], axis=1).iloc[:,1:]
spam_probability = training['Classifier'].value_counts()['spam'] / len(training)
ham_probability = training['Classifier'].value_counts()['ham'] / len(training)
spam_list=training['Classifier'] == 'spam'
ham_list=training['Classifier'] == 'ham'
spam = training[spam_list]
ham = training[ham_list]
spam_n = training.loc[spam_list, 'Messages'].apply(len).sum()
ham_n = training.loc[ham_list, 'Messages'].apply(len).sum()
def probability_word_spam(word):
        return (spam[word].sum() + 1) / (spam_n + Vocabulary) if word in training.colum

def probability_word_ham(word):
        return (ham[word].sum() + 1) / (ham_n + Vocabulary) if word in training.columns
import math as mt
def naive_classifier(Message):
    prob_spam = spam_probability
    prob_ham = ham_probability
    for word in Message:
        prob_spam += mt.log(probability_word_spam(word),10)
        prob_ham += mt.log(probability_word_ham(word),10)
    prob_spam=10**prob_spam
    prob_ham=10**prob_ham
    return 'ham' if prob_ham > prob_spam else 'spam'
testing['Predicted_label'] = testing['Messages'].apply(naive_classifier)
Accuracy = (testing['Predicted_label'] == testing['Classifier']).sum() / len(testing) *
print("Accuracy of this model:",Accuracy)
```

```
Accuracy of this model: 88.30703012912483
```

In [188…

```
#Experiment 3: partitioning training and testing data sets as 60:40
training = df.sample(frac=0.6)
testing = df.drop(training.index)
punctuations='''€˜%^&"\,!*_~)(-[};:]{'<#£$>./?@+'''
stop_words=["i","da","we","ur" ,"u","am","me", "my", "myself", "we", "our", "ours", "ou
wordlist=[]
for message in training['Messages']:

    for alpha in message:
        if alpha in punctuations:
            message = message.replace(alpha, " ")
    #for lower case
    message=message.lower()

    #for stop words
    message=message.split()
    #print(message)
    for word in message:
        if word in stop_words:
            #print(word,'\n')
            message.remove(word)
        wordlist.append(word)
wordlist=list(set(wordlist))
Vocabulary = len(wordlist)
word_counts_per_Messag=[
    [row[1].count(word) for word in wordlist]
    for _, row in training.iterrows()]
df_wordcount=pd.DataFrame(word_counts_per_Messag,columns=wordlist)
training = pd.concat([training.reset_index(), df_wordcount], axis=1).iloc[:,1:]
spam_probability = training['Classifier'].value_counts()['spam'] / len(training)
ham_probability = training['Classifier'].value_counts()['ham'] / len(training)
spam_list=training['Classifier'] == 'spam'
ham_list=training['Classifier'] == 'ham'
spam = training[spam_list]
ham = training[ham_list]
spam_n = training.loc[spam_list, 'Messages'].apply(len).sum()
ham_n = training.loc[ham_list, 'Messages'].apply(len).sum()
def probability_word_spam(word):
        return (spam[word].sum() + 1) / (spam_n + Vocabulary) if word in training.colum

def probability_word_ham(word):
        return (ham[word].sum() + 1) / (ham_n + Vocabulary) if word in training.columns

def naive_classifier(Message):
    prob_spam = spam_probability
    prob_ham = ham_probability
    for word in Message:
        prob_spam *= probability_word_spam(word)
        prob_ham *=probability_word_ham(word)
    return 'ham' if prob_ham > prob_spam else 'spam'
testing['Predicted_label'] = testing['Messages'].apply(naive_classifier)

Accuracy = (testing['Predicted_label'] == testing['Classifier']).sum() / len(testing) *
print("Accuracy of this model implementing 60:40 training:testing proportion:",Accuracy
```

```
Accuracy of this model implementing 60:40 training:testing proportion: 94.84304932735425
```

Effectiveness of our classifier comparing with random guessing:

In our case, there are two classifiers for the messages spam,ham. Even a random guess choosing out of these two classes has the probability 1/2 to choose correct class.With an simple logic we can say that the expected accuracy is 50%. Looking at this accuracy, we know that our classifier is trained well to predict the class with 96% accuracy. Again when we consider the basic two hypothesis for the random guess, one is the best case and other is the worst case. For the best case of the random choice, it may get 100 % accuracy with a very low probability whereas our classifier will never reach upto 100 %. But while considering the worst hypothesis, the random guess can have 0 % accuracy and classify all of the messages wrong with certainity. This can never happen with our classifier whose accuracy is 96%.

So while concluding, though our classifier may not get 100 % accuracy on both best and worst case but it may never get an accuracy of 0% wrongly classifying the messages with certainity.