Foundations of Machine Learning

DA5400 – Assignment II – BE21B037

SPAM or HAM Classifier:

Goal : Given a test folder containing a list of text files named “emails1.txt”, “emails2.txt” … predict if each email is a spam email or not (ham) .

Algorithm Used: Naive Bayes algorithm with laplacian smoothing.

Code Workflow:

1) Import Libraries
2) Input Training Data
3) Model Training
4) Sample Testing and error valuation
5) Function/procedure to test emails

Libraries Used:

```
# Import libraries
import pandas as pd
import numpy as np
import os
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

The **pandas** and **Numpy** libraries are standard libraries used for data tabulation and manipulation . The **os** library is to access folders and read text files as and when needed. The **CountVectorizer** is used to vectorize a given text file based on the words that occurred. And the **train_test_split** allows us to split the data according to a ratio for training and testing purposes.

Training Dataset/Description:

The dataset used was imported from the publicly available source from Kaggle. The downloaded zip file is extracted, and it contains a csv file named emails.csv, which is our dataset.

The dataset was then split 70–30, stratified over the number of spam/not_spam emails so as to maintain an equal balance in test and training data.

Information about Dataset:
Shape = 5728 rows × 2 columns
Two columns being "text" which contains the emails (including subject line) and "Spam" which contains binary values {0,1}. Marked 1 when the email is spam, and 0 when it is not (ham)
The original dataset is named **df**
Training Dataset is **df1**
Testing Dataset is **df2**

To see the ratio of spam to ham in **df1 (train)**

```
df1.groupby('spam').describe()
```

Output of which is :

	text			
	count	unique	top	freq
spam				
0	3052	3033	Subject: re : pserc industrial advisory board ...	2
1	957	957	Subject: top - level logo and business identit...	1

Hence the Train Dataset contains 3033 unique emails that are not spam, and 957 emails that are spam.

Similarly in **df2 (test)**

```
df2.groupby('spam').describe()
```

The output of which is :

	text			
	count	unique	top	freq
spam				
0	1308	1307	Subject: re : grades pam , the students rese...	2
1	411	411	Subject: 100 % free hardcore megasite ! ! 100...	1

Hence the Test Dataset contains 1307 unique emails that are not spam, and 411 emails that are spam.

Model Training :

Naive Bayes Method:
The ideology is that a particular email can either be a spam or not spam email.
The probability an email is spam = p_spam
The probability an email is ham = p_ham

Once an email has been decided as spam or ham. Then each word has a particular probability of appearing in that email. Its probability varies depending on whether the email is spam or not.
We create a dictionary of **d** words.
For spam the probability for each of the d words will be = [$p_1^1, p_2^1, p_3^1 \dots p_{d-1}^1, p_d^1$]
For Ham the probability for each of the d words will be = [$p_1^0, p_2^0, p_3^0 \dots p_{d-1}^0, p_d^0$]

Hence the features here are the words themselves, so a total of 2d–1 parameters must be estimated.
The feature vector for one email = [$f_1 , f_2 , f_3 \dots f_{d-1} , f_d$]
Where each f_i is either 1 or 0 depending on if the word exists in the email or not.

Assumption 1: Class conditional independence. The presence of one word does not affect the probability that another word appears.

Assumption 2: If there is a new word that is not present in our dictionary, it could have equal probability of coming in a spam or non-spam email. Hence the new word will not be counted in the dictionary.

We also add two new data points to the dataset, two feature vectors with all ones. One of them is marked as spam and the other is marked as ham. This is to take care of an edge case if any of the spam or ham probabilities is zero

So the parameter estimation is as follows (Training)
p_spam = number of spam emails / Total number of emails
p_ham = number of ham emails / Total number of emails

P_i^1 = the probability that a word(i) comes in spam emails
Hence P_i^1 = # of spam emails that contain the word(i) / # of spam emails [# - number of]

P_i^0 = the probability that a word(i) comes in ham emails
Hence P_i^0 = # of ham emails that contain the word(i) / # of ham emails

Now given an **x_test = [$f_1 , f_2 , f_3 \dots f_{d-1} , f_d$] ,**

y_test = 1 if

$$\frac{P(Y_{test}=1 \mid X_{test})}{P(Y_{test}=0 \mid X_{test})} \geq 0$$

⇒

$$log\left(\frac{P(Y_{test}=1 \mid X_{test})}{P(Y_{test}=0 \mid X_{test})}\right) \geq 0$$

⇒

$$log\left(\frac{P(X_{test} \mid Y_{test}=1) \times P(Y_{test}=1) / P(X_{test})}{P(X_{test} \mid Y_{test}=0) \times P(Y_{test}=0) / P(X_{test})}\right) \geq 0$$

Which upon simplification gives us

$$log(\frac{P_{spam}}{P_{ham}}) + \sum_{i=1}^{d} f_i \, log\,(\frac{P_i^{\,1} \times (1-P_i^{\,0})}{P_i^{\,0} \times (1-P_i^{\,1})}) + log\,(\frac{1-P_i^{\,1}}{1-P_i^{\,0}}) \geq 0$$

Otherwise y_test = 0

Training

Step 1: Each email is vectorised. Either a particular word is present (1) or not present (0) in an email

```
# Training Data
x_train = df1['text']
y_train = df1['spam']

# Binary Count Matrix
cv = CountVectorizer()
count_matrix = cv.fit_transform(x_train.values)
encoded_df = pd.DataFrame(count_matrix.toarray(), columns=cv.get_feature_names_out())
# Count matrix stored as DataFrame
```

Step 2: Laplacian smoothing to avoid zero probabilities

```
# Create Laplacian smoothing datapoints
Laplacian_spam = pd.DataFrame([[1] * encoded_df.shape[1]],
columns=encoded_df.columns)
Laplacian_ham = pd.DataFrame([[1] * encoded_df.shape[1]], columns=encoded_df.columns)
y_train =  pd.concat([y_train, pd.DataFrame([[0], [1]], columns=[0])],
ignore_index=True)

# Concatenate Smoothing datapoints and training data
encoded_df = pd.concat([encoded_df, Laplacian_spam], ignore_index=True)
encoded_df = pd.concat([encoded_df, Laplacian_ham], ignore_index=True)
```

Step 3: Parameter Estimation

```
# Split Training data based on spam or ham
encoded_df['Binary'] = y_train
spam_df = encoded_df[encoded_df['Binary'] == 1]
ham_df = encoded_df[encoded_df['Binary'] == 0]

# Spam and Ham Inputs
spam_x = spam_df.iloc[:,:-1]
ham_x = ham_df.iloc[:,:-1]

# Maximum likelihood estimation calculation for each feature
```

```
spam_sum = pd.DataFrame(spam_x.sum()).T
ham_sum = pd.DataFrame(ham_x.sum()).T
spam_prob = spam_sum.div(len(spam_x),axis=1)
ham_prob = ham_sum.div(len(ham_x),axis = 1)
p_spam = len(spam_x)/(len(spam_x)+len(ham_x))
p_ham = len(ham_x)/(len(spam_x)+len(ham_x))
```

Step 4: Prediction function for a new email text

```
# Prediction (Given text is spam or ham)
def test_email(text, cv, spam_prob, ham_prob, p_spam, p_ham):
    l = spam_prob.shape[1]
    test_wordcount = cv.transform([text]) # Any new word that is not present in the
training dictionary is ignored.
    test_df = pd.DataFrame(test_wordcount.toarray(),
columns=cv.get_feature_names_out())

    new_df = test_df * np.log((spam_prob * (1 - ham_prob))/(ham_prob * (1 -
spam_prob))) + np.log((1-spam_prob)/(1-ham_prob))
    sum = new_df.sum().sum()
    sum = sum + (np.log(p_spam/p_ham))

    if sum >= 0:
        return 1 # Spam
    else :
        return 0 # Ham
```

Sample Testing

We test with the 30% dataset **(df2).**

```
# Testing
[TP,TN,FN,FP] = [0,0,0,0]

for index,row in df2.iterrows():
    text = row['text']
    actual = row['spam']
    predicted = test_email(text, cv, spam_prob, ham_prob, p_spam, p_ham)

    if actual == predicted == 1:
        TP +=1
    elif actual == predicted == 0:
        TN += 1
    elif actual == 1 and predicted == 0:
        FN += 1
    elif actual == 0 and predicted == 1:
```

```
        FP += 1

Accuracy = (TP + TN)/(TP + TN + FP + FN)
Recall = TP/ (TP + FN)
Precision = TP/ (TP + FP)
F1_score = TP / (TP + 0.5*(FP + FN))
F1 = 2*(Precision*Recall)/(Recall+Precision)

print(f'Accuracy = {Accuracy}')
print(f'Recall = {Recall}')
print(f'Precision = {Precision}')
print(f'F1_score = {F1_score}')
```

The output of which is

```
Accuracy = 0.983129726585224
Recall = 0.9902676399026764
Precision = 0.9421296296296297
F1_score = 0.9655990510083037
```

We observe a F1_score of 96% percentage which tells us that model has an overall effectiveness, both in sensitivity and precision.

Function/Procedure to Test:

Assumption:
Current directory contains a folder called Test, which contains multiple text files named “email1.txt”, “email2.txt” … and so on .

Input to this section of code will be the file path to the Test folder
The output to this section will be of the format

email1.txt is a spam email

---

email2.txt is not a spam email

And so on…

The code will also save this data in a csv file named **'SpamHam.csv'** in the current directory.

```
# Reading the test data
folder_path = "/content/test" # input path of Test Folder
```

```
check_df = pd.DataFrame(columns=["email", "spam/ham"])

for filename in os.listdir(folder_path):
    if filename.endswith(".txt"):
        file_path = os.path.join(folder_path, filename)

        with open(file_path, 'r') as file:
            file_content = file.read()

        x = test_email(file_content, cv, spam_prob, ham_prob, p_spam, p_ham)
        check_df.loc[len(check_df)] = [filename, x]

        if x == 1:
            print(f"{filename} is a spam email")
        else:
            print(f"{filename} is not a spam email")
        print("\n" + "-"*40 + "\n")

# print(check_df)
check_df.to_csv('SpamHam.csv', index=False)
```