Programming Assignment: Email Spam Naive Bayes

Overview/Task

The goal of this programming assignment is to build a naive bayes classifier from scratch that can determine whether email text should be labled spam or not spam based on its contents

Review

Remeber that a naive bayes classifier realizes the following probability:

$$P(Y|X_1, X_2, \ldots, X_n) \propto P(Y) * P(Y|X_1) * P(Y|X_2) * \ldots * P(Y|X_n)$$

Where Y is a binary class {0,1}

Where X_i is a feature of the input

The classifier will decide what class each input belongs to based on highest probability from the equation above

Reminders

Please remember that the classifier must be written from scratch; do NOT use any libraries that implement the classifier for you, such as but not limited to sklearn.

You CAN, however, use SKlearn to split up the dataset between testing and training.

Feel free to look up any tasks you are not familiar with, e.g. the function call to read a csv

Task list/Recommended Order

In order to provide some guidance, I am giving the following order/checklist to solve this task:

1. Compute the "prior": P(Y) for Y = 0 and Y = 1
2. Compute the "likelihood": $P(Y|X_n)$
3. Write code that uses the two items above to make a decision on whether or not an email is spam or ham (aka not spam)
4. Write code to evaluate your model. Test model on training data to debug
5. Test model on testing data to debug

In [8]:
```
#import cell
import numpy as np
import pandas as pd
import random
import csv
```

Function template

In [9]:

```
def prior(df):
    ham_prior = 0
    spam_prior =  0
    '''YOUR CODE HERE'''
    ham_num = df.loc[df['label'] == 'ham'].count().values[0]
    spam_num = df.loc[df['label'] == 'spam'].count().values[0]
    total = ham_num + spam_num
    ham_prior = 1.0*ham_num/total
    spam_prior = 1.0*spam_num/total
    '''END'''
    return ham_prior, spam_prior

def likelihood(df):
    ham_like_dict = {}
    spam_like_dict = {}
    '''YOUR CODE HERE'''
    for i, row in df.iterrows():
        text = set([i.strip("/.,:?!'\"") for i in row['text'].split()])
        label = row['label']
        if label == 'spam':
            for word in text:
                if word not in spam_like_dict:
                    spam_like_dict[word] = 0
                spam_like_dict[word] +=1
        else:
            for word in text:
                if word not in ham_like_dict:
                    ham_like_dict[word] = 0
                ham_like_dict[word] +=1

    '''END'''

    return ham_like_dict, spam_like_dict

def predict(ham_prior, spam_prior, ham_like_dict, spam_like_dict, text):
    '''
    prediction function that uses prior and likelihood structure to compute proportional posterior for a single line of
    '''
    #ham_spam_decision = 1 if classified as spam, 0 if classified as normal/ham
    ham_spam_decision = None

    '''YOUR CODE HERE'''
    #ham_posterior = posterior probability that the email is normal/ham
    ham_posterior = None

    #spam_posterior = posterior probability that the email is spam
    spam_posterior = None

    ham_posterior = ham_prior
    spam_posterior = spam_prior
    ham_num = df.loc[df['label'] == 'ham'].count().values[0]
    spam_num = df.loc[df['label'] == 'spam'].count().values[0]
    for word in [i.strip("/.,:?!'\"") for i in text.split()]:
        if word in spam_like_dict:
            spam_posterior = spam_posterior * ((spam_like_dict[word] +1) / spam_num)
        else:
            spam_posterior = spam_posterior * (1 / sum(spam_like_dict.values()))
        if word in ham_like_dict:
            ham_posterior = ham_posterior * ((ham_like_dict[word] +1) / ham_num)
        else:
            ham_posterior = ham_posterior * (1 / sum(ham_like_dict.values()))

    if ham_posterior >= spam_posterior:
        ham_spam_decision = 0
    else:
        ham_spam_decision = 1

    '''END'''
    return ham_spam_decision


def metrics(ham_prior, spam_prior, ham_dict, spam_dict, df):
    '''
    Calls "predict" function and report accuracy, precision, and recall of your prediction
    '''

    '''YOUR CODE HERE'''
    TP = 0
    TN = 0
    FP = 0
    FN = 0
    for i, row in df.iterrows():
        text = row['text']
        label = row['label']
        if label == 'spam':
```

```
            prediction = predict(ham_prior, spam_prior, ham_dict, spam_dict, text)
            if prediction ==1:
                TP +=1
            else:
                FN +=1
        else:
            prediction = predict(ham_prior, spam_prior, ham_dict, spam_dict, text)
            if prediction ==1:
                FP +=1
            else:
                TN +=1

    acc = (TP+TN)/(TP+TN+FP+FN)
    precision = TP/(TP+FP)
    recall = TP/(TP+FN)

    '''END'''
    return acc, precision, recall
```

Generate answers with your functions

In [10]:
```
#loading in the training data
train_df = pd.read_csv("./TRAIN_balanced_ham_spam.csv")
test_df = pd.read_csv("./TEST_balanced_ham_spam.csv")
df = train_df
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2398 entries, 0 to 2397
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Unnamed: 0    2398 non-null   int64 
 1   Unnamed: 0.1  2398 non-null   int64 
 2   label         2398 non-null   object
 3   text          2398 non-null   object
 4   label_num     2398 non-null   int64 
dtypes: int64(3), object(2)
memory usage: 93.8+ KB
```

In [11]:
```
#compute the prior

ham_prior, spam_prior = prior(df)

print(ham_prior, spam_prior)
```

```
0.5 0.5
```

In [12]:
```
# compute likelihood

ham_like_dict, spam_like_dict = likelihood(df)
```

In [13]:
```
# Test your predict function with some example TEXT

some_text_example = "write your test case here"
print(predict(ham_prior, spam_prior, ham_like_dict, spam_like_dict, some_text_example))
```

```
1
```

In [14]:
```
# Predict on test_df and compute metrics

df = test_df
acc, precision, recall = metrics(ham_prior, spam_prior, ham_like_dict, spam_like_dict, df)
print(acc, precision, recall)
```

```
0.9 0.9724409448818898 0.8233333333333334
```

In [ ]:
```

```