from google.colab import drive

drive.mount('/content/gdrive')

!ls /content/gdrive/My\ Drive/Dora/Bilkent/CS464/HW1 # Use YOUR OWN DIRECTORY!!

import os

import csv

import math

import random

import operator

import pdb

import time

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import scipy as scp

from timeit import default\_timer as timer

from math import log, inf

np.random.seed(123)

tic = time.perf\_counter()

countfiles = 0

countspam = 0

root = '/content/gdrive/My Drive/Dora/Bilkent/CS464/HW1'

csv\_yTrain = os.path.join(root, 'y\_train.csv')

csv\_xTrain = os.path.join(root, 'x\_train.csv')

csv\_yTest = os.path.join(root, 'y\_test.csv')

csv\_xTest = os.path.join(root, 'x\_test.csv')

vocab = os.path.join(root, 'vocabulary.txt')

import numpy as np

import pandas as pd

#read the csv files ot extract the feature and labels from the test and

train\_feature = pd.read\_csv(csv\_xTrain, header=None)

train\_label = pd.read\_csv(csv\_yTrain, header=None)

test\_feature = pd.read\_csv(csv\_xTest, header=None)

test\_label = pd.read\_csv(csv\_yTest, header=None)

labels = ['0' , '1']

vocabcount = 44020 #since this quantity is given, it is directly used, however it could've also been easily calculated.

toc = time.perf\_counter()

print(f"Total time is {toc - tic:0.4f} seconds")

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force\_remount=True).

vocabulary.txt x\_test.csv x\_train.csv y\_test.csv y\_train.csv

Total time is 188.1832 seconds

tic = time.perf\_counter()

#generate the train\_multinomial concatanated matrix to assign training features to their respective labels.

train\_multinomial = pd.concat((train\_label[0].rename('label'), train\_feature), axis=1)

#count the number of spam and normal mails in the training dataset

label\_counts = train\_label[0].value\_counts()

normal\_mails = label\_counts[0]

spam\_mails = label\_counts[1]

total\_mails = normal\_mails + spam\_mails

#calculate the probabilities of choosing a normal or a spam email from the dataset.

prob\_normal = normal\_mails/total\_mails

prob\_spam = spam\_mails/total\_mails

toc = time.perf\_counter()

print(f"Total time is {toc - tic:0.4f} seconds")

Total time is 0.1258 seconds

tic = time.perf\_counter()

#find the number of occurances of a specific word in the spam and normal datasets, and store it in occurances.

occurances = train\_multinomial.groupby('label').sum()

#extract the total number of each word used in the normal and spam mails seperately.

normal\_total = occurances.sum(axis=1)[0]

spam\_total = occurances.sum(axis=1)[1]

#count the number of times that a particular word has occured. This will be used for the Bernoulli Model.

countoccurance = train\_multinomial.mask(train\_multinomial > 1, 1).groupby('label').sum()

toc = time.perf\_counter()

print(f"Total time is {toc - tic:0.4f} seconds")

Total time is 12.6460 seconds

def confusion\_results(prediction\_matrix):

    #this function is defined to compute the confusion matrix values for the ML models.

    true\_positive = 0

    true\_negative = 0

    false\_negative = 0

    false\_positive = 0

    for i in range(len(prediction\_matrix)):

        if prediction\_matrix[i] == 1 and prediction\_matrix[i] == test\_label[0][i] :

            true\_positive += 1

        elif prediction\_matrix[i] == 0 and prediction\_matrix[i] == test\_label[0][i] :

            true\_negative += 1

        elif prediction\_matrix[i] == 1 and prediction\_matrix[i] != test\_label[0][i] :

            false\_positive += 1

        elif prediction\_matrix[i] == 0 and prediction\_matrix[i] != test\_label[0][i] :

            false\_negative += 1

    accuracy = (true\_positive+true\_negative)/(true\_negative+true\_positive+false\_negative+false\_positive)\*100

    precision = true\_positive / (true\_positive + false\_positive)\*100

    print('Number of True Positives: ', true\_positive)

    print('Number of True Negatives: ', true\_negative)

    print('Number of False Positives: ', false\_positive)

    print('Number of False Negatives: ', false\_negative)

    print('Accuracy is ', accuracy, ' %')

    print('Precision is ', precision, ' %')

def MLE\_predictor(email):

    #this function is used to calculate the posteriors from the Multinomial MLE model, given its priors.

    #the posteriror for spam and normal will be calculated and compared to reach to a prediction for a particular email document.

    yi\_normaldf = (email \* np.log((occurances.iloc[0])/(normal\_total))).fillna(0)

    yi\_spamdf = (email \* np.log((occurances.iloc[1])/(spam\_total))).fillna(0)

    yi\_normal = yi\_normaldf.values.sum() + math.log(prob\_normal)

    yi\_spam = yi\_spamdf.values.sum() + math.log(prob\_spam)

    if yi\_normal > yi\_spam:

        prediction = 0

    elif yi\_normal < yi\_spam:

        prediction = 1

    elif yi\_normal == yi\_spam:

        prediction = 0

    return prediction

def MAP\_predictor(email):

    #this function is used to calculate the posteriors from the MAP model, given its priors.

    #the posteriror for spam and normal will be calculated and compared to reach to a prediction for a particular email document.

    yi\_normaldf = (email \* np.log((occurances.iloc[0]+1)/(normal\_total+44020))).fillna(0)

    yi\_spamdf = (email \* np.log((occurances.iloc[1]+1)/(spam\_total+44020))).fillna(0)

    yi\_normal = yi\_normaldf.values.sum() + math.log(prob\_normal)

    yi\_spam = yi\_spamdf.values.sum() + math.log(prob\_spam)

    if yi\_normal > yi\_spam:

        prediction = 0

    elif yi\_normal < yi\_spam:

        prediction = 1

    elif yi\_normal == yi\_spam:

        prediction = 0

    return prediction

def Bernoulli\_predictor(email):

    #this function is used to calculate the posteriors from the Bernoulli model, given its priors.

    #the posteriror for spam and normal will be calculated and compared to reach to a prediction for a particular email document.

    newmail = email.mask(email > 1, 1)

    compnewmail = 1-newmail

    yi\_normaldf = (newmail \* np.log((countoccurance.iloc[0])/normal\_mails)).fillna(0) + ((compnewmail)\*(np.log(1-(countoccurance.iloc[0])/normal\_mails))).fillna(0)

    yi\_spamdf = (newmail \* np.log((countoccurance.iloc[1])/(spam\_mails))).fillna(0) + ((compnewmail)\*(np.log(1-(countoccurance.iloc[1])/spam\_mails))).fillna(0)

    yi\_normal = yi\_normaldf.values.sum() + math.log(prob\_normal)

    yi\_spam = yi\_spamdf.values.sum() + math.log(prob\_spam)

    if yi\_normal > yi\_spam:

        prediction = 0

    elif yi\_normal < yi\_spam:

        prediction = 1

    elif yi\_normal == yi\_spam:

        prediction = 0

    return prediction

prediction\_matrix\_MLE = []

np.warnings.filterwarnings('ignore')

tic = time.perf\_counter()

#the prediction for every document will be conducted from the test features, using the corresponding model.

#predictions will be stored in its correspondig matrix, each index referring to the respective document.

for i in range(len(test\_label)):

    prediction\_matrix\_MLE.append(MLE\_predictor(test\_feature.iloc[i]))

confusion\_results(prediction\_matrix\_MLE)

toc = time.perf\_counter()

print(f"Total time is {toc - tic:0.4f} seconds")

Number of True Positives: 611

Number of True Negatives: 317

Number of False Positives: 8

Number of False Negatives: 150

Accuracy is 85.451197053407 %

Precision is 98.70759289176091 %

Total time is 16.1942 seconds

prediction\_matrix\_MAP = []

tic = time.perf\_counter()

#the prediction for every document will be conducted from the test features, using the corresponding model.

#predictions will be stored in its correspondig matrix, each index referring to the respective document.

for i in range(len(test\_label)):

    prediction\_matrix\_MAP.append(MAP\_predictor(test\_feature.iloc[i]))

confusion\_results(prediction\_matrix\_MAP)

toc = time.perf\_counter()

print(f"Total time is {toc - tic:0.4f} seconds")

Number of True Positives: 751

Number of True Negatives: 315

Number of False Positives: 10

Number of False Negatives: 10

Accuracy is 98.15837937384899 %

Precision is 98.68593955321944 %

Total time is 17.4232 seconds

prediction\_matrix\_Bernoulli = []

np.warnings.filterwarnings('ignore')

tic = time.perf\_counter()

#the prediction for every document will be conducted from the test features, using the corresponding model.

#predictions will be stored in its correspondig matrix, each index referring to the respective document.

for i in range(len(test\_label)):

    prediction\_matrix\_Bernoulli.append(Bernoulli\_predictor(test\_feature.iloc[i]))

confusion\_results(prediction\_matrix\_Bernoulli)

toc = time.perf\_counter()

print(f"Total time is {toc - tic:0.4f} seconds")

Number of True Positives: 613

Number of True Negatives: 300

Number of False Positives: 25

Number of False Negatives: 148

Accuracy is 84.06998158379373 %

Precision is 96.08150470219435 %

Total time is 33.8550 seconds