Programming Assignment 4
Hetsvi Navnitlal

1.

For 4 rounds:
Train – 0.051111111111111114
Test – 0.03875968992248062

Training errors:
For 3 rounds:
Train: 0.06444444444444444

For 7 rounds:
Train: 0.028888888888888888

For 10 rounds:
Train: 0.015555555555555555

For 15 rounds:
Train: 0.0

For 20 rounds:
Train: 0.0

Test Errors:
For 3 rounds:
Test: 0.03875968992248062

For 7 rounds:
Test: 0.031007751937984496

For 10 rounds:
Test: 0.03875968992248062

For 15 rounds:
Test: 0.023255813953488372

For 20 rounds:
Test: 0.023255813953488372

Overview of Training and Test Errors for each iteration

Iteration	Train	Test
4	0.051111111111111114	0.03875968992248062
3	0.06444444444444444	0.03875968992248062
7	0.028888888888888888	0.031007751937984496
10	0.015555555555555555	0.03875968992248062
15	0.0	0.023255813953488372
20	0.0	0.023255813953488372

2.

The top words after 10 iterations are:
1 - Remove
2 - Language
3 - Free
4 - University
5 - Money
6 - Linguistic
7 - Click
8 - Fax
9 - Want
10 - De

Code:

```
# imports
import numpy as np
import pandas as pd
import random
from collections import Counter
import math

# loading files
text = np.loadtxt('./pa4train.txt')
other = np.loadtxt('./pa4test.txt')
words = open('./pa4dictionary.txt')
word = words.readlines()

class boosting(object):  # boosting class
```

```
def __init__(self):   # distribution over training
   self.distribution = (len(text))*[1/(len(text))]

def alpha(self, error): # calculating the alpha
   return 1/2*(np.log((1-error)/error))

def h(self, training, clas, hc): # using the classifier getting value
   if(hc == '-'):
      if(training[clas] == 0):
         return 1
      else:
         return -1
   if(hc == '+'):
      if(training[clas] == 1):
         return 1
      else:
         return -1

def boosting(self, training,t): # the boosting training
   alph = []
   main_min_errors = []
   for i in range(t):
      min_error = []
      for j in range(len(training[0])-1):
         error = [0, 0 ,'+']
         error_one = [0, 0, '-']
         for k in range(len(training)):
            if (training[k][j] == 1):
               if training[k][-1] != 1:
                  error[0] += self.distribution[k]
                  error[1] = j
            else:
               if training[k][-1] != -1:
                  error[0] += self.distribution[k]
                  error[1] = j

            if (training[k][j] == 0):
               if training[k][-1] != 1:
                  error_one[0] += self.distribution[k]
                  error_one[1] = j
            else:
               if training[k][-1] != -1:
                  error_one[0] += self.distribution[k]
                  error_one[1] = j
```

```
            if error[0] < error_one[0]:

                min_error.append(error)
            else:

                min_error.append(error_one)

        min_index = np.argmin(min_error, axis = 0)
        main_min_error = min_error[min_index[0]]
        alphaa = self.alpha(main_min_error[0])

        alph.append(alphaa)
        main_min_errors.append(main_min_error)

        z = 0
        numnerators = []
        for v in range(len(training)):
            z += self.distribution[v]*math.exp(-alphaa*training[v][-1]*self.h(training[v],
main_min_error[1], main_min_error[2]))
            numnerators.append(self.distribution[v]*math.exp(-alphaa*training[v][-
1]*self.h(training[v], main_min_error[1], main_min_error[2])))

        d = []
        for t in numnerators:
            d.append(t/z)

        for s in range(len(training)):
            self.distribution[s] = d[s]

    print(main_min_errors)

    return main_min_errors, alph

  def cl(self, alpha, cls, t, training): # from training get classifier using that to get value
    sum =0
    for i in range(t):
      sum += alpha[i]*self.h(training, cls[i][1], cls[i][2])
    return np.sign(sum)

  def predict(self, training, other, t): # getting predictions
    predictions = []
    a,b = self.boosting(training, t)
    for i in range(len(other)):
```

```
            predictions.append(self.cl(b, a , t, other[i]))
        return predictions

    def rates(self,train, other, t): # error rates
        count = 0
        for i in range(len(other)):
            if other[i] != train[i][-1]:
                count = count +1
        return count / len(other)

    def words(self, word, training, t): # words from classifier
        a, b = self.boosting(training, t)
        words = []
        for i in a:
            words.append(word[i[1]])

        return words

# gives error
boostingg = boosting()
predict = boostingg.predict(text, other, 10)
boostingg.rates(other,predict, 10)

# gives words
boost = boosting()
boost.words(word, text, 10)
```