Zequn Yu

CSE151

PA 5

Q1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | t = 3 | t = 7 | t = 10 | t = 15 | t = 20 |
| Training error | 0.064 | 0.029 | 0.016 | 0.000 | 0.000 |
| Testing error | 0.039 | 0.031 | 0.039 | 0.023 | 0.023 |

Q2.

The first ten rounds are:

remove

language

free

university

money

linguistic

click

fax

want

de

*# Author: Zequn Yu  
# PID: A14712777***from** \_\_future\_\_ **import** division  
*#import pandas as pd***import** numpy  
**import** math  
  
*# function to help add data to array***def** add\_data(line, data, file):  
 **while** (line):  
 data.append(line.split())  
 line = file.readline()  
  
*# read data from files  
# to save data from files*train\_data = []  
test\_data = []  
*# to open files*train\_file = open(**"pa5train.txt"**, **"r"**)  
test\_file = open(**"pa5test.txt"**, **"r"**)  
*# to read each line from files*train\_line = train\_file.readline()  
test\_line = test\_file.readline()  
*# set label*label\_idx = len(train\_data[0]) - 1;  
*# call function to read*add\_data(train\_line, train\_data, train\_file)  
add\_data(test\_line, test\_data, test\_file)  
*# test read  
# print("train: ", train\_data)  
# print("test:", test\_data  
  
  
# implement given weak learners:  
# hi,+(x) = 1, if word i occurs in email x  
# = −1, otherwise***def** classifier\_h(data, idx, sign):  
 *# h+ case  
 # print(" Now the sign is: ", sign)* **if**(sign == **"+"**):  
 **if** (data[idx] == 1):  
 **return** 1  
 **else**:  
 **return** -1  
 *# h- case* **else**:  
 **if** (data[idx] == 0):  
 **return** 1  
 **else**:  
 **return** -1  
  
*# function to get error***def** cal\_error(list):  
 tol\_err = 0  
 *# check each e-mail feature read in* **for** email **in** list:  
 *# check each feature  
 # for feature in range(0, label\_idx):* train\_l = email[label\_idx]  
 should\_l = classifier\_h(email, label\_idx, **"+"**)  
 **if**(train\_l == should\_l):  
 tol\_err = tol\_err + 1  
 **return** tol\_err  
  
*# function to change alpha***def** update\_alpha(b\_err):  
 c\_list = ()  
 alpha = .5 \* numpy.log((1 - b\_err.e) / b\_err.e)  
 c\_list.append((alpha, b\_err.h, b\_err.word))  
 **return** c\_list  
  
*# the main part of bossting***def** boost(data):  
 class\_tuple = ()  
 *# for rounds test* boost = [3, 7, 10, 15, 20]  
 *# check different boost* **for** b **in** boost:  
 *# print("Now boost is: ", b )* cur\_err = 100.0  
 cur\_feature = -1  
 cur\_label = 2  
 weight = [1/len(data)] \* len(data)  
 *# run for rounds of range  
 # update feature, label and error* rounds = len(data[0]) - 1  
 **for** r **in** range(rounds):  
 *# calculate the number of error* tmp\_err = cal\_error(r)  
 *# check if need to update error* **if**(tmp\_err < cur\_err):  
 cur\_err = tmp\_err  
 cur\_feature = r  
 cur\_label = 1  
 *# check if flip label* **elif**(1 - tmp\_err < cur\_err):  
 cur\_err = 1 - tmp\_err  
 cur\_feature = 1  
 cur\_label = -1  
 *# update alpha* class\_tuple = update\_alpha(cur\_err)  
 *# update* **for** i **in** range(len(data)):  
 d = [weight[i] \* numpy.e \*\* (-class\_tuple[0] \* y[i] \* cur\_err.h(x.iloc[i, :], cur\_err.word))]  
 *# set normalize* d = 0  
 sum\_d = sum(d)  
 update\_d = [(i / sum\_d) **for** i **in** d]  
 *#return class\_tuple* print\_stat(train\_data, test\_data, tuple)  
  
*# function to get label***def** cal\_label(feature, list):  
 cur\_l = 0  
 **for** l **in** list:  
 **if** l[1] == 1:  
 **if** feature[l[0]] == 1:  
 cur\_l += l[2]  
 **else**:  
 cur\_l -= l[2]  
 **else**:  
 **if** feature[l[0]] == 0:  
 cur\_l += l[2]  
 **else**:  
 cur\_l -= l[2]  
 **return** (cur\_l / math.fabs(cur\_l))  
  
*# predict***def** predict (data, classif):  
 pred = data.apply(**lambda** x: classify(x, classif), axis=1)  
 **return** pred  
  
*# set final classify***def** classify(x, classifiers):  
 *#print("classify turn")* total = 0  
 **for** c **in** classifiers:  
 *# update* alpha = c[0]  
 h = c[1]  
 word = c[2]  
 total += (alpha \* h(x, word))  
 *# print(total)* **return** numpy.sign(total)  
  
*# cal train error***def** t\_err(t\_data, tuple):  
 err = 0  
 **for** t **in** t\_data:  
 currLabel = cal\_label(t, tuple)  
 **if** currLabel != t[-1]:  
 err += 1  
 err = (err / float(len(train\_data)))  
 **return** err  
*# cal test error***def** e\_err(e\_data, tuple):  
 err = 0  
 **for** t **in** e\_data:  
 currLabel = cal\_label(t, tuple)  
 **if** currLabel != t[-1]:  
 err += 1  
 err = err / float(len(e\_data))  
 **return** err  
  
**def** print\_stat(t\_data, e\_data, tuple):  
 *# print("start training data:")* t = boost(t\_data)  
 *# print("run predict: ")* p = predict(t\_data, t)  
 tr\_err = t\_err(t\_data, tuple)  
 *#print("Now the data err is: ", tr\_err)* e = boost(e\_data)  
 *# peinr("run predict: ")* p2 = predict(e\_data, e)  
 te\_err = e\_err(e\_data, tuple)  
  
**'''  
rounds = [3,4,7,10,15,20]  
for r in rounds:  
 for t in dataList:  
 curr\_l = calLabel(t, tuple)  
 if curLabel != t[-1]:  
 errorCount += 1  
 readFile("..\data\pa5test.txt")  
 errorCount = 0  
  
 for t in dataList:  
 curr\_l = calLabel(t, tuple)  
 if curr\_l != t[-1]:  
 errorCount += 1  
  
'''**