**Comp-6915 machine learning**

**Assignment2**

1. **Short description**
2. **collect data: using the data in X\_train.txt and store it in a list.**
3. **In the program, we select 3 values of K. For each k we selected, do as follows**
4. **We use grid-search 10-fold CV, split the data into 10 parts. Select one of part to be test set, and merge the rest into one list. There are 10 situations, because the list split into 10 parts.**
5. **For each situation in above, we can use assignment1 to work out a prediction list of the test set from. Compare the prediction value and true value search from Y\_train. We can work out the distance, here we have two method to figure out distance, one is Manhattan distance D2 and the other is Euclidean distance D1. Use each of the distance to work out spearman correlation for D1 and D2, we also can work out the standard division for each distance. That is for one situation. We have 10 situations, so we can work out 20 spearman correlation numbers. 10 spearman correlation numbers for D1, and the others for D2. Then figure out the average of spearman correlation numbers for each distance. In this step we can have two average spearman correlation numbers. In the one of the case K, one for D1 and the other for D2.**
6. **Repeat the above steps three times for 3 different K which means find the nearest K neighbours. We can get three sets of numbers, each set include two spearman correlations, one for D1 and the other for D2.**
7. **The last step is output the result as a excel file to show the result.**

**1.pseudo-code of KNN implementation:****def** inputsequence(filename): #open input file and store the file content as a list。X\_train.txt and X.unseen.txt will use this function  
 **with** open(filename) **as** f:  
 **for e**achline **in** f.readlines():  
 tf\_sq\_sp = tf\_sq.split(**'\t'**) #use tab to split the list  
 tf.append(tf\_sq\_sp) # store in a list named tf  
 **return** tf

**def** splitDataSet(trainset,split\_size):# for K-fold CV, this function can split it into K parts  
 each\_size = len(trainset)/split\_size #figure out how many elements in each parts  
 **fo**r each **in** dataset: #traversal all elements in dataset  
 each\_split.append(i) #add each elements into a list until the capacity is full  
 count\_num = count\_num + 1  
 **if** the part is full: #if the part is full, store it in a another list  
 split\_all.append(each\_split)  
 each\_split = []  
 count\_num = 0  
 split\_all.append(each\_split) #the list have 10 subsets  
 r**eturn** split\_all

**def** denerageDataSet(splitData,j): *#generate test set and train set*  **for** i **in** splitData: #traversal all 10 subsets, the number i as test set, the others merge as one train set  
 **if** i **is** splitData[j]:  
 testset = i  
 **else**:  
 trainset = trainset + i  
 **return** testset,trainset

**def** performanceeachtrainset(k,doneset,outputset,points): #figure out the spearman correlation for each situation  
 c = comparetf(doneset[0],doneset[1],points) *#*donset[0] is testset，doneset[1]is trainset  
e = classify(int(k),outputset,c,doneset[0])  
 d1 = 0 *#*Manhattan distance  
d2 = 0 #Euclidean distance  
 **while** w < len(e): #traversal all the name in the prediction set  
**while** i < len(outputset[0]): *#*traversal all the name in the Y\_train  
**if** the prediction name == the name in the Y\_train: **while** j < len(outputset):

workd out the Manhattan distance  
woked out the Euclidean distane

dsquaresum1 = dsquaresum1 + (d1 \* d1)  
 dsquaresum2 = dsquaresum2 + (d2 \* d2)

j = j + 1  
 spearman1 = 1 - 6 \* dsquaresum1 / ((len(outputset) -1 ) \* (len(outputset) - 1) \* (len(outputset) - 1) - (len(outputset) - 1)) #using the forum work out the spearman correlation for Manhattan distance  
 spearman2 = 1 - 6 \* dsquaresum2 / ((len(outputset) - 1) \* (len(outputset) - 1) \* (len(outputset) - 1) - (len(outputset) - 1)) #using the forum work out the spearman correlation for Euclidean distance

j = 1

i = i + 1

i = 0  
 spearmansum1 = spearmansum1 + spearman1  
 spearmansum2 = spearmansum2 + spearman2  
 w = w + 1

allspearmanmean1 = spearmansum1/len(e) #accordingD1 figure out the mean of the spearman correlation   
 allspearmanmean2 = spearmansum2 / len(e) #accordingD2 figure out the mean of the spearman correlation  
 **return** allspearmanmean1,allspearmanmean2

**def** performancealltrainset(k,splitDate,outputset,points): #figure out the average spearman correlation for one case K

**for** j **in** range(len(splitDate)): #traversal 10 situations  
 a = denerageDataSet(splitDate, j) #generate the test set and train set for one situation  
 spearman = performanceeachtrainset(k,a,outputset,points) *#figure out the spearman correlation* spearmanset1.append(spearman[0]) *#add the spearman number in a list, all numbers for Manhattan distance.*

spearmanset2.append(spearman[1]) *#add the spearman number in a list, all numbers for Euclidean distance*

allspearmansum1 = allspearmansum1 + spearman[0] *#figure out the sum. Manhattan distance* allspearmansum2 = allspearmansum2 + spearman[1]  *#figure out the sum. Euclidean distance* allspearmanmean1 = allspearmansum1 / len(splitDate) *# figure out the average number. Manhattan distance* allspearmanmean2 = allspearmansum2 / len(splitDate) *#figure out the average number. Euclidean distance* **for** i **in** spearmanset1:   
 squaresum1 = squaresum1 + (i-allspearmanmean1)\*(i-allspearmanmean1) sme1 = squaresum1\*\*0.5 *# figure out the standard deviation. Manhattan distance*  
 **for** w **in** spearmanset2:  
 squaresum2 = squaresum2 + (i-allspearmanmean2)\*(i-allspearmanmean2) sme2 = squaresum2\*\*0.5 *#figure out the standard deviation. Euclidean distance*  
 **return** allspearmanmean1,sme1,allspearmanmean2,sme2

**def** circileperformance(): #in this function can loop exeution for each K  
 k = [2,3,5] #we define 3 K numbers, the find the nearest K neighbours

**for** i **in** k: #traversal each k   
 row.append(**"K="**)  
 row.append(i)

a = performancealltrainset(i, f, outputset,j) #figure out two spearman correlation numbers for D1 and D2  
 **if** spearman correlation number is the largest:  
 bestk = i # store the number of K  
 store the result in one list and have three list for 3 different K

rowall.append(row)

final.append(**"model chosen:"**)  
 output the best choice who have the largest spearman correlation numbers

**def** classify(k,outputset,nameset,tf\_test\_all): #according to the X\_unseen set to find the distance between each factor in X\_unseen set and each factor in train set. And then ccording to K, we can find the k nearest neighbors. Figure out the average of neighbors' output.  
 **while** w <= (len(nameset)-1): #find form the first unseen factor**while** j <= k - 1: #find k nieghbors  
**while** i <= (len(outputset[0])-1): #find the nearst neighbor name in the Y\_train.txt  
**if** the neighbor name == name in Y\_output:

record the name’s location   
 averageset.append(tf\_test\_all[w][0]) #record the name of a unseen factor  
 **while** z <= (len(outputset) - 1): *#*calculate the distance

**for** x **in** target:  
 sum = sum + float(outputset[z][x])  
 average = sum/len(target)  
 averageset.append(average)#record the average distance  
 averagesetall.append(averageset)#record all the distance in a list  
 **return** averagesetall  
**def** comparetf(tf\_test\_all,tf\_train\_all): # figure out the distance of testset and trainset  
 **while** i<=(len(tf\_test\_all)-1): #traversal all the X\_train  
**while** j<=(len(tf\_train\_all)-1): #traversal all the Y\_unseen  
**for** x **in** [2,4,5,24,30,43,45,46,47]: #set monitoring point  
**if** point in X\_train not the same as point X\_unseen: distance add one  
 put the name and distance together and sotre in a list  
 namedistanceset = []  
 distanceset = sorted(distanceset, key=**lambda** x: x[0])#sort by the distance  
 distancesetall.append(distanceset) #put one test factor distance in the set  
 **return** distancesetall

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
file = sys.argv[1] *#"X\_train.txt"*

file2 = sys.argv[2] *#"Y\_train.txt"*filename3 = **"** **model\_selection\_table.txt"**

trainset = inputsequence(file) *#a*

outputset = outputsequence(file2) *#d*

f = splitDataSet(trainset,10)

result = circileperformance()

output = open(**' model\_selection\_table.txt'**,**'w'**,encoding=**'gbk'**)  
output.write(**'\t\t D1Euclidean Distance\t\t\t\tD2Manhattan Distance\n'**)  
print(**"the result is in model\_selection\_table.txt"**)

the result is as below graph:

图片包含 屏幕截图

描述已自动生成

**2.** **description of distance functions assessed,：**

In the code, we use two distance functions which are Manhattan distance and Euclidean distance. According to the result of the program, the better is Euclidean distance function which make the spearman correlation is bigger than Manhattan distance.

**3.performance of the KNN model chosen in comparison with alternative models considered**

There are 3 values of K (2,3,5) and two distance function one is Euclidean distance and the other is Manhattan distance.Each containing average spearman correlation plus/minus standard deviation. The last sentence is the best model which the program chosen from 6 sets of models.The performance is the best when the K=5 and distance function is Euclidean distance function

图片包含 屏幕截图

描述已自动生成