Daniel Douglass

423276

Data Mining 514

HW 3

P1.

For Problem 1 we first fixed the missing value problem by using K-NN with k=7 and used the median of the 7 nearest neighbors to fill in our missing data. The K-NN were run on the ctrl and case groups separately. The goal of this problem was to use both agglomerative and divisive clustering methods to separate our data and evaluate the performance of each method. Below are the results our clustering models to evaluate the Alzheimer’s disease data set.

2. Out of 364 subjects, 148 were misclassified using agglomerative clustering. The data points that were misclassified to the wrong group by agglomerative clustering (bottom up) are: 'WGACON1', 'WGACON3', 'WGACON4', 'WGACON7', 'WGACON8', 'WGACON9', 'WGACON14', 'WGACON15', 'WGACON16', 'WGACON17', 'WGACON21', 'WGACON26', 'WGACON27', 'WGACON29', 'WGACON30', 'WGACON38', 'WGACON40', 'WGACON45', 'WGACON46', 'WGACON47', 'WGACON51', 'WGACON52', 'WGACON53', 'WGACON55', 'WGACON58', 'WGACON59', 'WGACON60', 'WGACON62', 'WGACON70', 'WGACON71', 'WGACON73', 'WGACON74', 'WGACON77', 'WGACON79', 'WGACON80', 'WGACON86', 'WGACON87', 'WGACON91', 'WGACON94', 'WGACON96', 'WGACON97', 'WGACON99', 'WGACON100', 'WGACON101', 'WGACON102', 'WGACON105', 'WGACON106', 'WGACON107', 'WGACON108', 'WGACON110', 'WGACON111', 'WGACON112', 'WGACON113', 'WGACON115', 'WGACON117', 'WGACON118', 'WGACON119', 'WGACON121', 'WGACON122', 'WGACON125', 'WGACON131', 'WGACON132', 'WGACON133', 'WGACON134', 'WGACON138', 'WGACON139', 'WGACON142', 'WGACON143', 'WGACON144', 'WGACON146', 'WGACON149', 'WGACON150', 'WGACON153', 'WGACON156', 'WGACON158', 'WGACON161', 'WGACON162', 'WGACON164', 'WGACON165', 'WGACON174', 'WGACON176', 'WGACON181', 'WGACON183', 'WGACON186', 'WGAAD2', 'WGAAD6', 'WGAAD7', 'WGAAD8', 'WGAAD9', 'WGAAD13', 'WGAAD14', 'WGAAD17', 'WGAAD20', 'WGAAD28', 'WGAAD32', 'WGAAD33', 'WGAAD38', 'WGAAD41', 'WGAAD42', 'WGAAD45', 'WGAAD46', 'WGAAD47', 'WGAAD48', 'WGAAD49', 'WGAAD50', 'WGAAD51', 'WGAAD60', 'WGAAD63', 'WGAAD64', 'WGAAD68', 'WGAAD72', 'WGAAD75', 'WGAAD78', 'WGAAD81', 'WGAAD82', 'WGAAD86', 'WGAAD90', 'WGAAD91', 'WGAAD92', 'WGAAD94', 'WGAAD95', 'WGAAD98', 'WGAAD108', 'WGAAD110', 'WGAAD113', 'WGAAD114', 'WGAAD115', 'WGAAD120', 'WGAAD123', 'WGAAD127', 'WGAAD132', 'WGAAD136', 'WGAAD145', 'WGAAD146', 'WGAAD148', 'WGAAD151', 'WGAAD154', 'WGAAD155', 'WGAAD156', 'WGAAD158', 'WGAAD162', 'WGAAD167', 'WGAAD168', 'WGAAD169', 'WGAAD170', 'WGAAD171', 'WGAAD175', 'WGAAD176'

3. Out of 364, 143 were misclassified using top down hierarchical clustering. The data points that were misclassified to the wrong group by divise clustering (top down) are: 'WGACON1', 'WGACON3', 'WGACON4', 'WGACON6', 'WGACON7', 'WGACON8', 'WGACON9', 'WGACON14', 'WGACON15', 'WGACON16', 'WGACON17', 'WGACON21', 'WGACON22', 'WGACON26', 'WGACON27', 'WGACON29', 'WGACON30', 'WGACON38', 'WGACON40', 'WGACON41', 'WGACON45', 'WGACON46', 'WGACON47', 'WGACON51', 'WGACON52', 'WGACON53', 'WGACON58', 'WGACON59', 'WGACON60', 'WGACON62', 'WGACON70', 'WGACON71', 'WGACON73', 'WGACON74', 'WGACON77', 'WGACON79', 'WGACON80', 'WGACON87', 'WGACON89', 'WGACON91', 'WGACON94', 'WGACON96', 'WGACON97', 'WGACON99', 'WGACON100', 'WGACON101', 'WGACON102', 'WGACON105', 'WGACON106', 'WGACON107', 'WGACON108', 'WGACON110', 'WGACON111', 'WGACON112', 'WGACON113', 'WGACON115', 'WGACON117', 'WGACON119', 'WGACON121', 'WGACON125', 'WGACON131', 'WGACON132', 'WGACON133', 'WGACON134', 'WGACON138', 'WGACON139', 'WGACON142', 'WGACON143', 'WGACON144', 'WGACON146', 'WGACON149', 'WGACON150', 'WGACON153', 'WGACON156', 'WGACON158', 'WGACON164', 'WGACON165', 'WGACON174', 'WGACON175', 'WGACON176', 'WGACON181', 'WGACON183', 'WGACON186', 'WGAAD2', 'WGAAD6', 'WGAAD7', 'WGAAD8', 'WGAAD9', 'WGAAD14', 'WGAAD17', 'WGAAD20', 'WGAAD21', 'WGAAD28', 'WGAAD38', 'WGAAD41', 'WGAAD42', 'WGAAD45', 'WGAAD46', 'WGAAD48', 'WGAAD49', 'WGAAD51', 'WGAAD63', 'WGAAD64', 'WGAAD68', 'WGAAD72', 'WGAAD74', 'WGAAD75', 'WGAAD78', 'WGAAD81', 'WGAAD82', 'WGAAD86', 'WGAAD91', 'WGAAD92', 'WGAAD94', 'WGAAD95', 'WGAAD97', 'WGAAD98', 'WGAAD108', 'WGAAD113', 'WGAAD114', 'WGAAD115', 'WGAAD120', 'WGAAD127', 'WGAAD132', 'WGAAD136', 'WGAAD138', 'WGAAD139', 'WGAAD145', 'WGAAD146', 'WGAAD148', 'WGAAD151', 'WGAAD153', 'WGAAD154', 'WGAAD155', 'WGAAD156', 'WGAAD158', 'WGAAD162', 'WGAAD167', 'WGAAD168', 'WGAAD169', 'WGAAD170', 'WGAAD171', 'WGAAD176'

3. In this section we will compare the results of Top Down recursive K-means and bottom up agglomerative clustering with 2,4 and 8 clusters.

Bottom Up 1st Level

Cluster Sizes: The size of the cluster is 162 for the case group and 202 for the control group

Accuracy: 0.5934

True Positive Rate: 0.539773

False Positive Rate: 0.460227

Top Down 1st Level

Cluster Sizes: The size of the is 159 for the case group and 205 for the control group

Accuracy: 0.607143

True Positive Rate: 0.545455

False Positive Rate: 0.454545

Comparing the first levels of the two different clustering methods we see that top down provides better accuracy, true positive rate and a better false positive rate. If I were to recommend one of these algorithms to cluster the data, Top down would be my choice. However 60% accuracy is not an ideal accuracy as we are dealing with identifying Alzheimer’s disease for patients. A false positive is costly for our patients thus with such a high false positive rate in both cases I would not recommend using either clustering method to determine whether or not a patient has Alzheimer’s. In addition both clustering methods provided two clusters of very similar size.

The accuracy per clusters were calculated by classifying a cluster as either ctrl or case, this was done by taking the mode of true labels of these clusters groups.

Bottom Up 2nd Level

Size of Cluster Accuracy

'group0': [138, 0.58695652173913038],

'group1': [137, 0.61313868613138689],

'group2': [64, 0.625],

'group3': [25, 0.56000000000000005]

Top Down 2nd Level

Size of Cluster Accuracy

'group0': [46, 0.67391304347826086],

'group1': [113, 0.5752212389380531],

'group2': [138, 0.60869565217391308],

'group3': [67, 0.61194029850746268]}

Comparing the second level of the clustering methods we see that the bottom up method produced one cluster that is significantly small. This shows that these data points in the bottom up cluster consisting of 25 points may be considered outliers as they are far away from the other clusters. If we were to see the level below we might expect to find smaller sized clusters of potential outliers. In addition, we see on average the accuracy of the top down clustering performs better than the bottom up method.

Bottom Up 3rd Level

Cluster size accuracy

'group0': [52, 0.53846153846153844],

'group1': [11, 0.54545454545454541],

'group2': [19, 0.84210526315789469],

'group3': [139, 0.58273381294964022],

'group4': [117, 0.63247863247863245],

'group5': [9, 0.55555555555555558],

'group6': [12, 0.66666666666666674],

'group7': [5, 0.59999999999999998]

Top Down 3rd Level

Cluster size Accuracy

'group0': [17 0.6470588235294118],

'group1': [29, 0.68965517241379315],

'group2': [46, 0.56521739130434778],

'group3': [67, 0.58208955223880599],

'group4': [61, 0.65573770491803285],

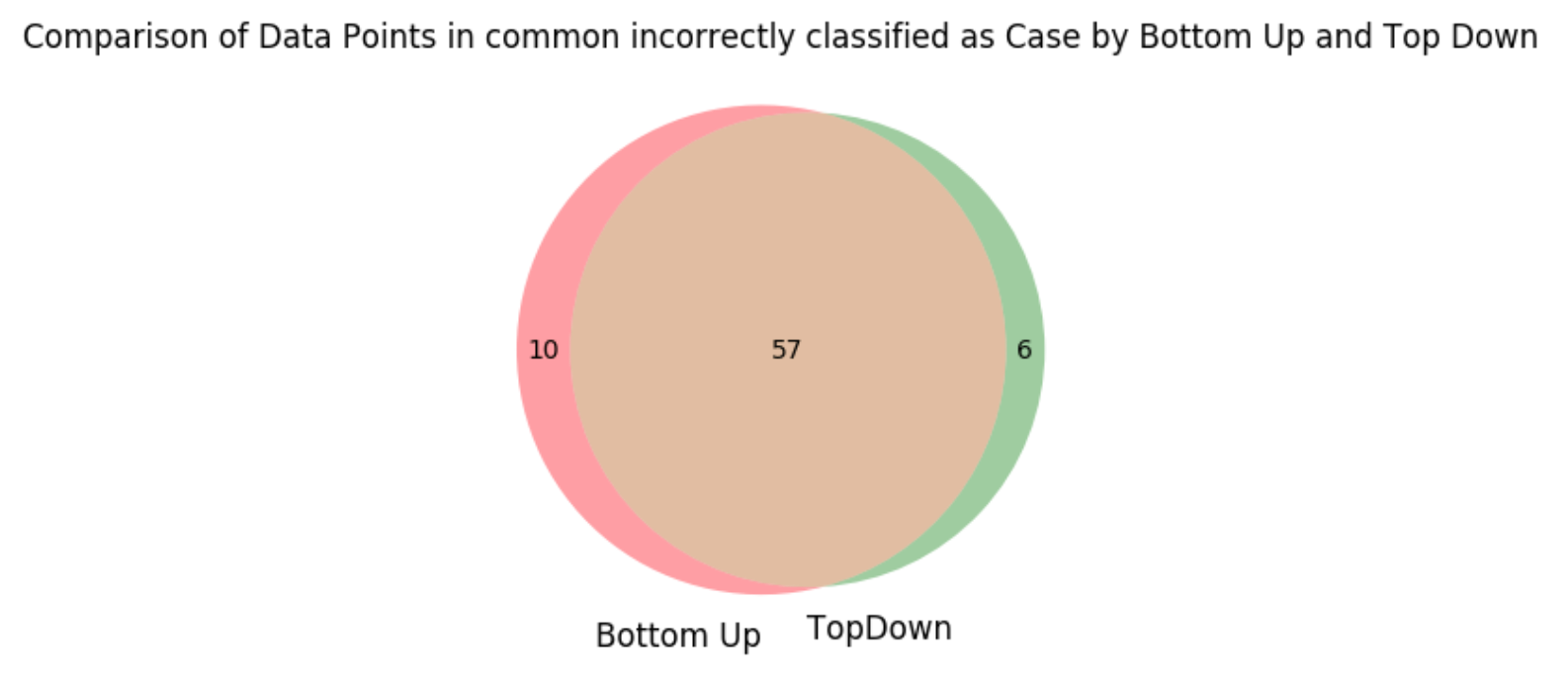
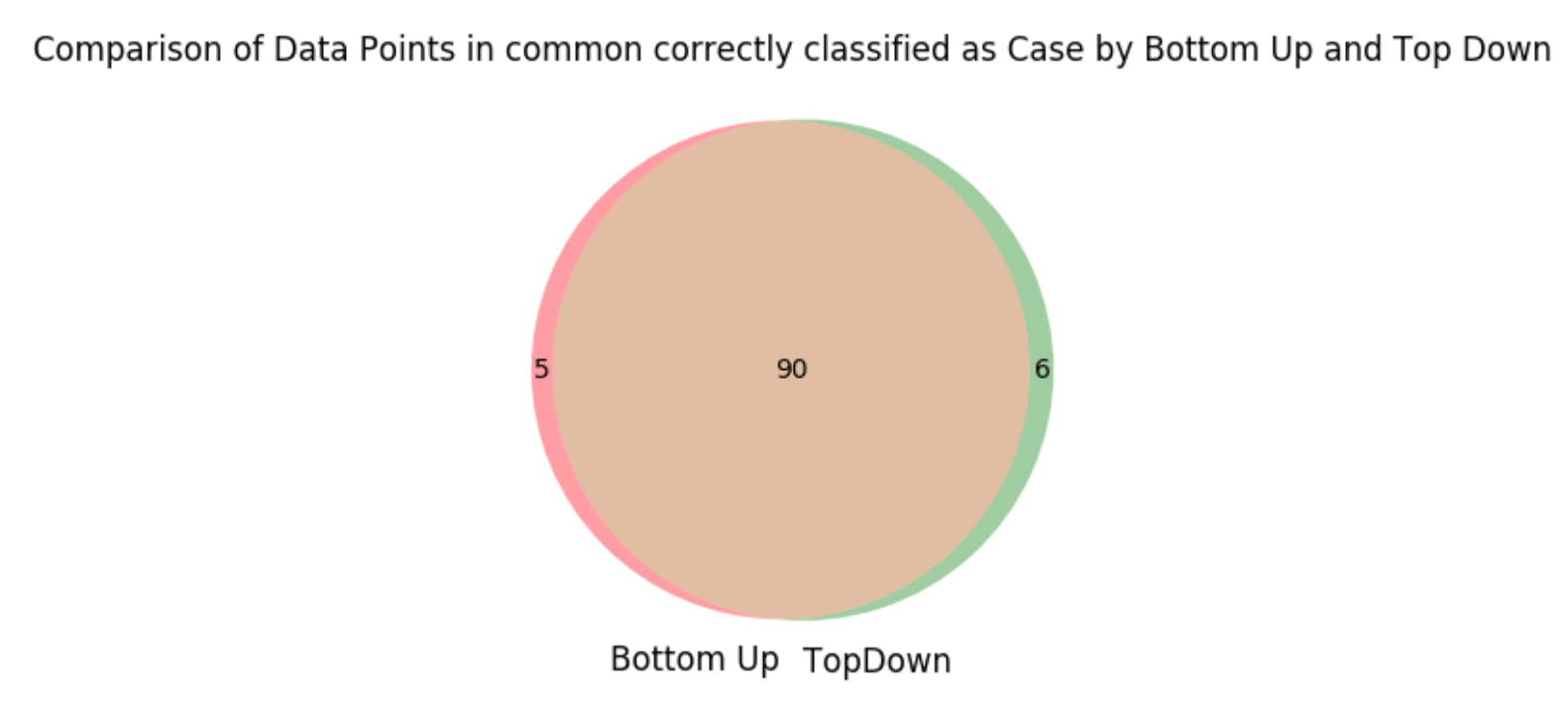
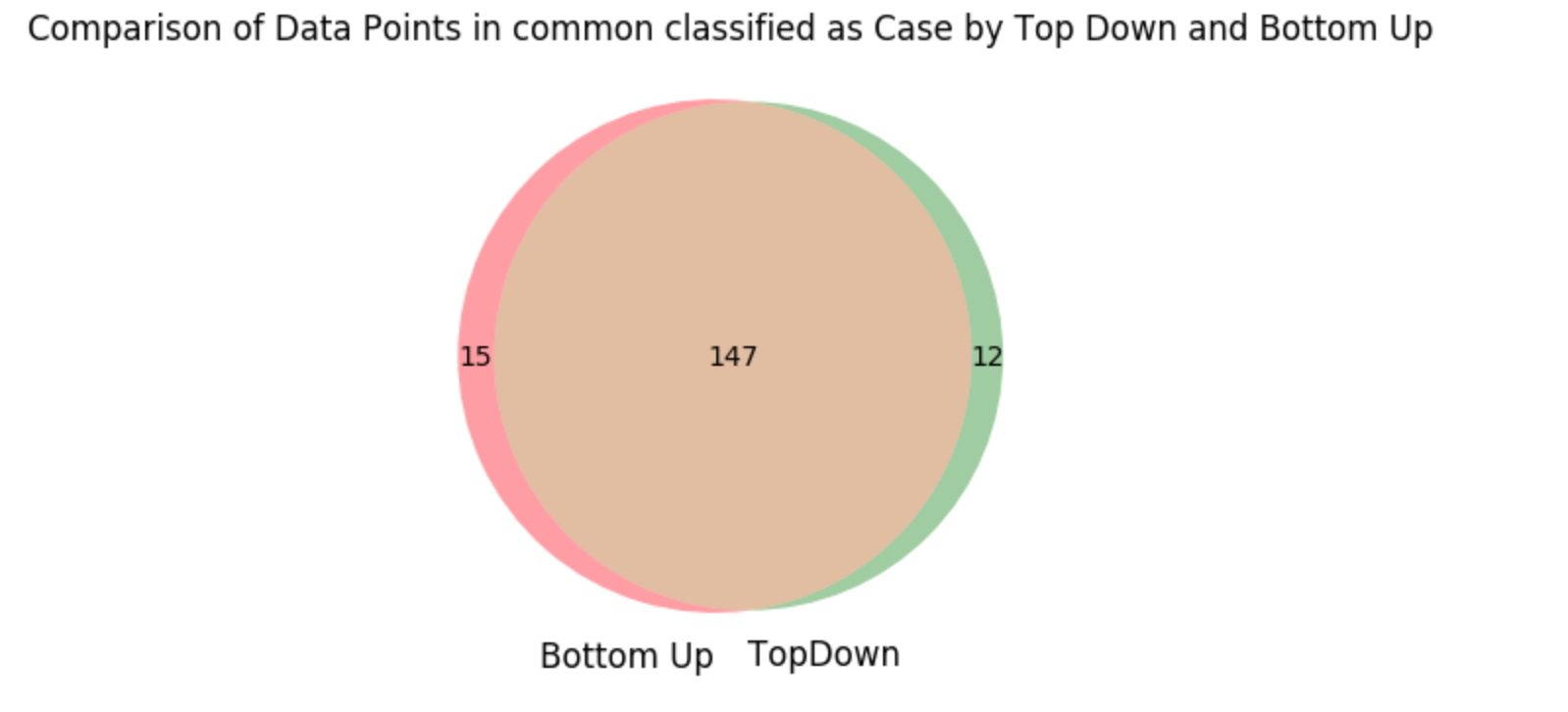
'group5': [77, 0.5714285714285714],

'group6': [27, 0.62962962962962965],

'group7': [40, 0.59999999999999998]

Comparing the 3rd level between bottom up and top down we see that bottom up produces two very large clusters of over 100 data points each, 5 very small clusters and one medium sized cluster. These smaller clusters are of interest as they may contain outliers in our data set. These points are very far the other clusters thus represent a minority in our data set. Top down in comparison has 6 medium sized clusters and 2 smaller clusters, but none as small as those contained in bottom up. Comparing these two methods at the 3rd level reveals that bottom up clustering may be better at detecting outliers in datasets as compared to top down k means clustering. But in comparison top down produces on average more accurate results. In conjunction these two clustering method may be used to provide accurate clusters and detect outliers for other data sets. In regards to the Alzheimer’s data these methods do not produce desirable results in which these algorithms can be used for practical use.

4.

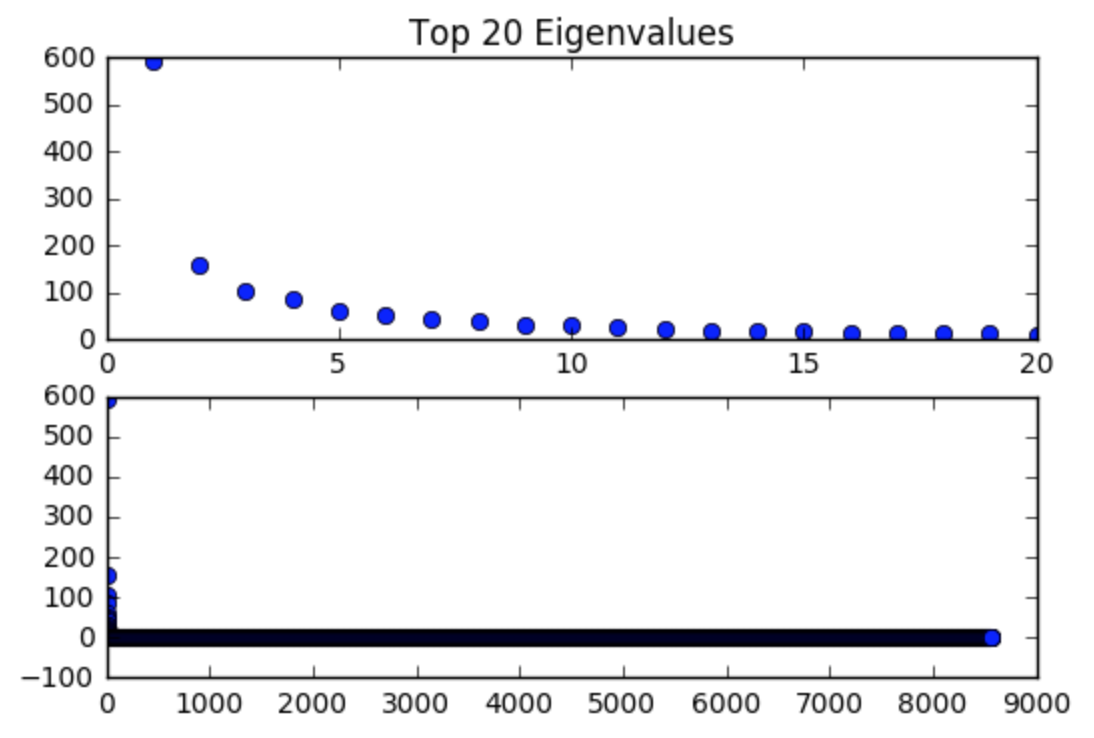
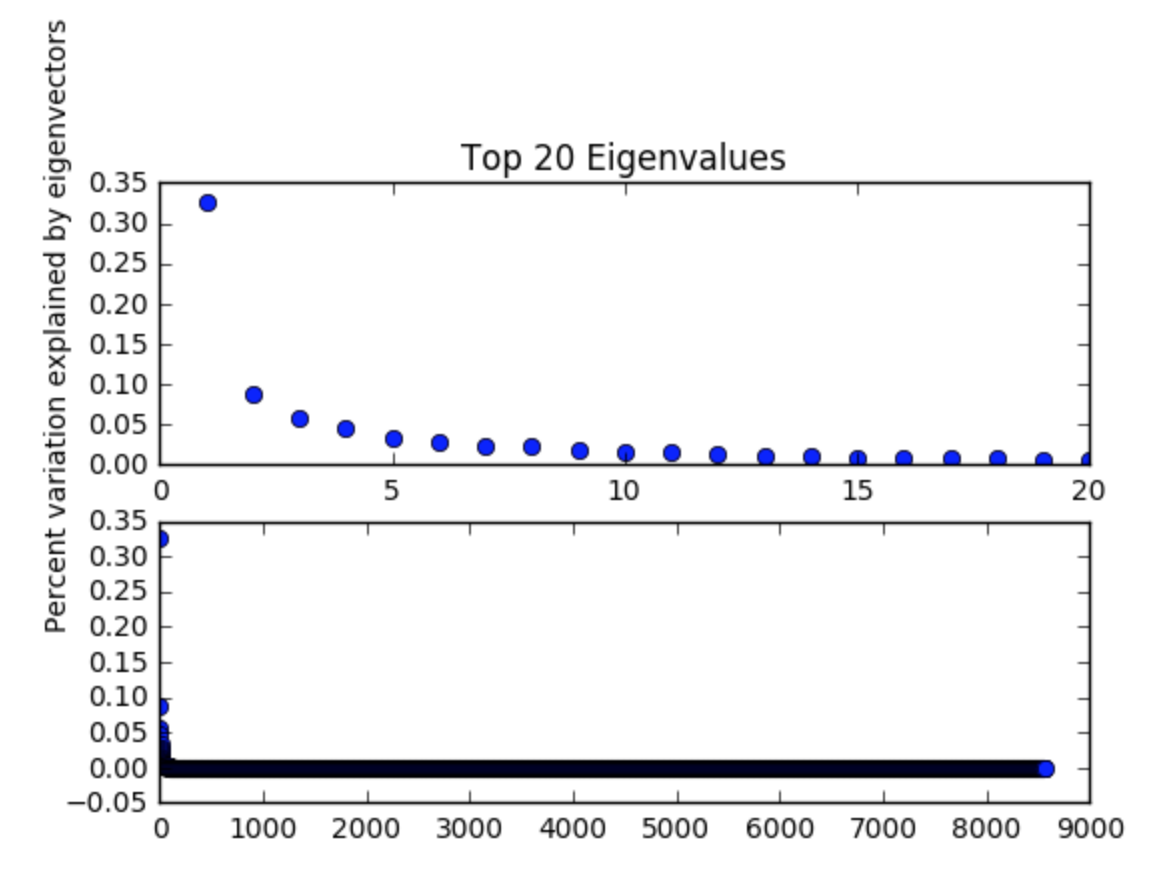


The true positive rate for the Top Down clustering method into 2 clusters is .5454 and the true positive rate for the bottom up clustering method into 2 clusters is .539. Between pB and pT, 147 of those examples where in both pB and pT. This shows how similar bottom up and top down were in correctly classifying the case groups. When looking at the False positive of the two groups we see that the bottom up incorrectly classified 4 more examples than did the top down approach. Because a false positive is not ideal in the case of diagnosing someone with Alzheimer’s we see that the top down approach produces a better model, but only marginally.

5. To detect real outliers I would use a partitioning cluster based method. For example I would use bottom up k means clustering. This clustering outlier detection model can determine systematic outliers if a data either falls into a very small cluster, does not belong to a cluster or if there is a large distance between the data point and the closest cluster. From the previous part of this problem we see that bottom up clustering into 8 clusters produces some clusters that have a very small amount of data points. These small clusters may contain outliers in the data.

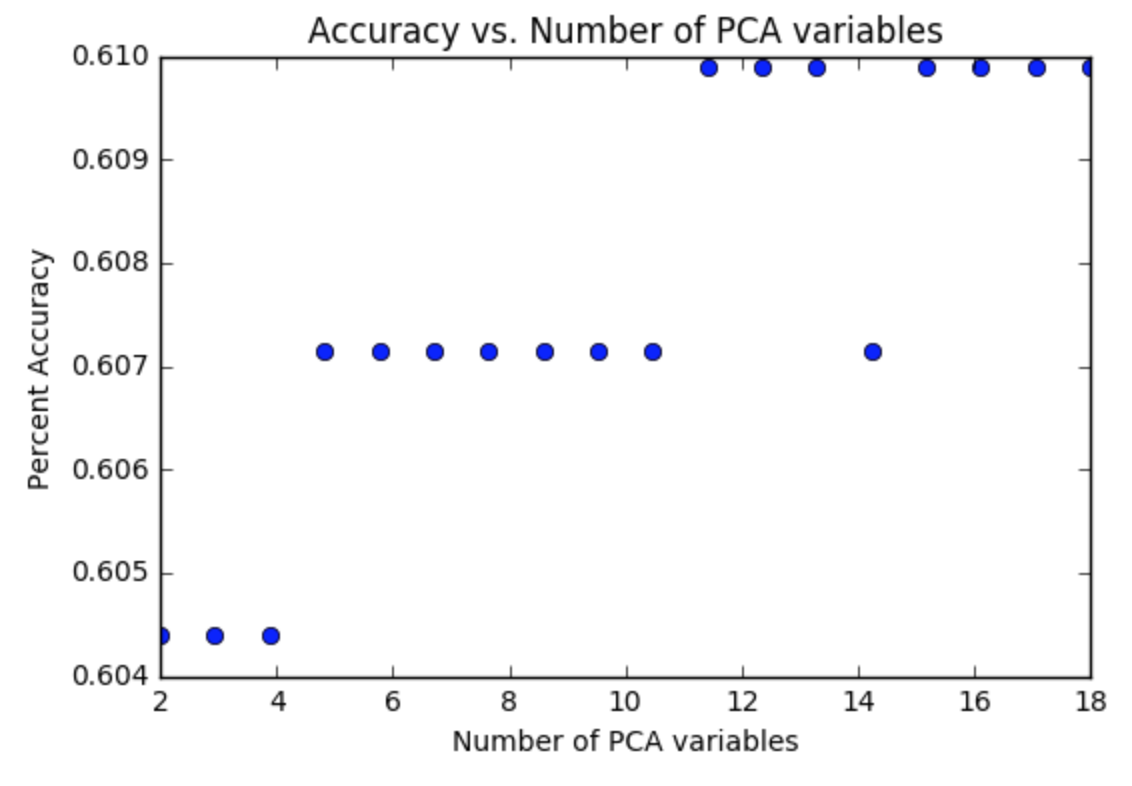
P2.

1.



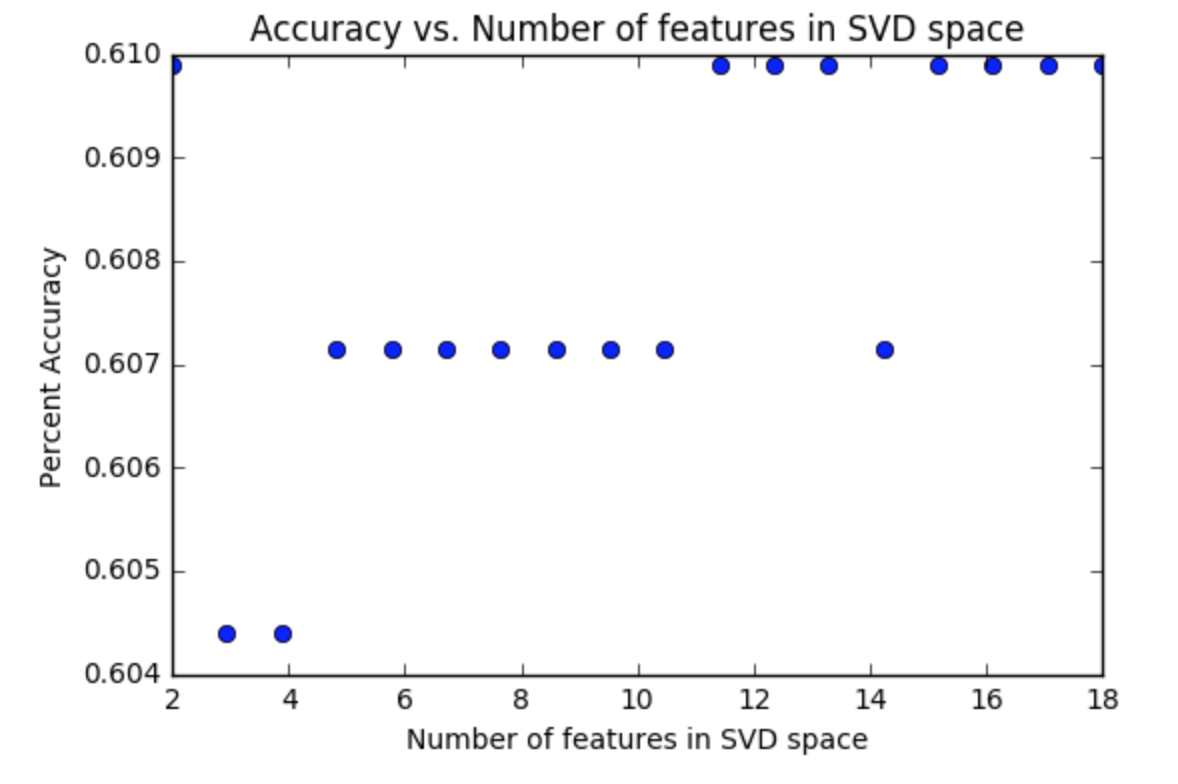
Above are graphs showing the top 20 eigenvalues ordered from largest to smallest and then all eigenvalues from the covariance matrix ordered. The graphs on the left rank the eigenvalues based on the percent variance explained by the respective eigenvector and on the right the value of the eigenvalue. The graph on the left shows the two biggest eigenvalues having eigenvectors that explain ~35% and ~10% of the variance in the data respectively. In addition, the top 20 eigenvalues can explain approximately ~65% of the variation in our data. This is significant in the fact that our datasets can be reduced from ~8000 features to 2-20 features that are transformed by PCA, while preserving 45%-65% of variation in the dataset. This shows how PCA is effective in reducing the dimensionality of a high dimensional data set.

2.



After running k-means from for the number of features used from 2 to 20 we see that using more than 4 features that are transformed by eigenvectors result in a consistent accuracy of at least .605. We can see that our accuracy does not vary very much between the numbers of features used. It shows that with the addition of another feature the accuracy of our clustering model may increase very slightly but by not much. It also shows that after a certain amount of eigenvectors used to transform our data there is not much improvement in the accuracy of our clustering.

p3.



After running k-means from for the number of features used from 2 to 20 we see that using more than 4 features that are transformed by eigenvectors result in a consistent accuracy of at least .605. We can see that our accuracy does not vary very much between the numbers of features used. It shows that with the addition of another feature the accuracy of our clustering model may increase very slightly but by not much. It also shows that after a certain amount of eigenvectors used to transform our data there is not much improvement in the accuracy of our clustering. Our results from SVD and PCA are very similar. This is expected because SVD computes the right singular vectors of the data matrix X that are the same as the eigenvectors for the covariance matrix that PCA generates. This is because we center our data around our means in PCA while in SVD we do not.

Problem 1 Script:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **knnimpute** **as** **knnImp**

**from** **sklearn.cluster** **import** AgglomerativeClustering

**from** **sklearn.cluster** **import** KMeans

In [ ]:

caseDf = pd.read\_csv('/Users/Daniel/Documents/Data Mining 514/HW3/case.csv',low\_memory=**False**)

ctrlDf = pd.read\_csv('/Users/Daniel/Documents/Data Mining 514/HW1/problem5/ctrl.csv',low\_memory=**False**)

patientLabels = ctrlDf['Arrays']

patientLabels = patientLabels.append(caseDf['Arrays']).reset\_index(drop=**True**)

In [ ]:

*#imputes median value from 7-nn for missing values ctrl group*

ctrlData = ctrlDf.drop('Arrays',axis=1)

features = ctrlData.columns

ctrlDataImputed = knnImp.knn\_impute\_few\_observed(ctrlData.as\_matrix(),ctrlData.isnull().as\_matrix(),7)

ctrlDf = pd.DataFrame(data = ctrlDataImputed, columns = features)

In [ ]:

*#imputes median value from 7-nn for missing values case group*

caseData = caseDf.drop('Arrays',axis=1)

features = caseData.columns

caseDataImputed = knnImp.knn\_impute\_few\_observed(caseData.as\_matrix(),caseData.isnull().as\_matrix(),7)

caseDf =pd.DataFrame(data = caseDataImputed, columns = features)

In [ ]:

allPatientsKnnImputedData = pd.concat([ctrlDf,caseDf])

*#allPatientsKnnImputedData.to\_csv('/Users/Daniel/Documents/Data Mining 514/HW3/allPatientsKnnImputedData.csv',low\_memory=False)*

In [ ]:

*#gets data and sets up Patients and Normal indices, X and Y for estimator and validation*

allPatientsKnnImputedData = pd.read\_csv('/Users/Daniel/Documents/Data Mining 514/HW3/allPatientsKnnImputedData.csv',low\_memory=**False**)

P = np.linspace(0,175,num=176)*#indices of normal ctrl*

N = np.linspace(176,363,num=188)*#indices of case patients*

Y = allPatientsKnnImputedData['Classes']

X = allPatientsKnnImputedData.drop('Classes',axis=1)

In [ ]:

*#fits estimator bottom up for 2 clusters*

estimator = AgglomerativeClustering(n\_clusters=2, affinity='cosine', linkage = 'complete')

estimator.fit(X.sub(X.mean(axis=0), axis=1))

bottomUp2ClusterLabels = estimator.labels\_

*#determines the ctrl and case labels*

results = abs((estimator.labels\_+1)- Y.as\_matrix())

**if** sum(results)/len(results) > .5:

misclassifiedBottomUp = np.nonzero(results-1)

pB = np.where(estimator.labels\_ ==0)

nB = np.where(estimator.labels\_ ==1)

**else**:

misclassifiedBottomUp = np.nonzero((results))

pB = np.where(estimator.labels\_ ==1)

nB = np.where(estimator.labels\_ ==0)

max(sum(abs((estimator.labels\_+1)- Y.as\_matrix()))/len(Y),1-sum(abs((estimator.labels\_+1)- Y.as\_matrix()))/len(Y))

In [ ]:

TPR = len(np.intersect1d(pB[0][pB[0]<176],P))/len(P)

FPR = 1 - len(np.intersect1d(pB[0][pB[0]<176],P))/len(P)

print("The true Positie rate is **%f** and the False Positive rate it **%f**"%(TPR,FPR) )

print("The size of the clusters are **%d** for the positve group and **%d** for the negative group"%(len(pB[0]),len(nB[0])))

patientList = []

**for** pat **in** patientLabels[misclassifiedBottomUp[0]]:

patientList.append(pat)

print(patientList)

patientList = []

*#Bottom up 4 groups clustering*

estimator = AgglomerativeClustering(n\_clusters=4, affinity='cosine', linkage = 'complete')

estimator.fit(X.sub(X.mean(axis=0), axis=1))

bottomUp4ClusterLabels = estimator.labels\_

bottomUp2ndLevelDict = {'group0':[],

'group1':[],

'group2':[],

'group3':[]

}

bottomUp2ndLevelStats = {'group0':0,

'group1':0,

'group2':0,

'group3':0

}

totalMisclassified4Clusters=0

**for** label **in** np.unique(bottomUp4ClusterLabels):

group = 'group**{0}**'.format(label)

bottomUp2ndLevelDict[group] = X.iloc[np.where(bottomUp4ClusterLabels==label)]

tempAccuracy = sum(bottomUp2ndLevelDict[group].index<176)/len(bottomUp2ndLevelDict[group])

**if** tempAccuracy>.5:

bottomUp2ndLevelStats[group] = [len(bottomUp2ndLevelDict[group]),tempAccuracy]

**else**:

bottomUp2ndLevelStats[group] = [len(bottomUp2ndLevelDict[group]),1-tempAccuracy]

totalMisclassified4Clusters = totalMisclassified4Clusters + bottomUp2ndLevelStats[group][0]

In [ ]:

bottomUp2ndLevelStats

In [ ]:

*#Bottom up 8 groups clustering*

estimator = AgglomerativeClustering(n\_clusters=8, affinity='cosine', linkage = 'average')

estimator.fit(X.sub(X.mean(axis=0), axis=1))

bottomUp8ClusterLabels = estimator.labels\_

bottomUp3rdLevelDict = { 'group0':[],

'group1':[],

'group2':[],

'group3':[],

'group4':[],

'group5':[],

'group6':[],

'group7':[]

}

bottomUp3rdLevelStats = { 'group0':0,

'group1':0,

'group2':0,

'group3':0,

'group4':0,

'group5':0,

'group6':0,

'group7':0

}

totalMisclassified8Clusters = 0

*#get indices for each group and calcualte misclassification number and accuracy based on majority of case or ctrl in group*

**for** label **in** np.unique(bottomUp8ClusterLabels):

group = 'group**{0}**'.format(label)

bottomUp3rdLevelDict[group] = X.iloc[np.where(bottomUp8ClusterLabels==label)]

tempAccuracy = sum(bottomUp3rdLevelDict[group].index<176)/len(bottomUp3rdLevelDict[group])

clusterSize = len(bottomUp3rdLevelDict[group])

**if** tempAccuracy>.5:

bottomUp3rdLevelStats[group] = [clusterSize,sum(bottomUp3rdLevelDict[group].index>176), tempAccuracy]

**else**:

bottomUp3rdLevelStats[group] = [clusterSize,sum(bottomUp3rdLevelDict[group].index<176),1-tempAccuracy]

totalMisclassified8Clusters = totalMisclassified8Clusters + bottomUp3rdLevelStats[group][0]

In [ ]:

bottomUp3rdLevelStats

In [ ]:

**def** new\_euclidean\_distances(X, Y=**None**, Y\_norm\_squared=**None**, squared=**False**):

**return** cosine\_similarity(X,Y)

*# monkey patch (ensure cosine dist function is used)*

KMeans.euclidean\_distances = new\_euclidean\_distances

estimator = KMeans(n\_clusters = 2)

estimator.fit(X)

max(sum(abs((estimator.labels\_+1)- Y.as\_matrix()))/len(Y),1-sum(abs((estimator.labels\_+1)- Y.as\_matrix()))/len(Y))

results = abs((estimator.labels\_+1)- Y.as\_matrix())

**if** sum(results)/len(results) > .5:

misclassifiedTopDown = np.nonzero(results-1)

pT = np.where(estimator.labels\_ ==0)

nT = np.where(estimator.labels\_ ==1)

**else**:

misclassifiedTopDown = np.nonzero((results))

pT = np.where(estimator.labels\_ ==1)

nT = np.where(estimator.labels\_ ==0)

patientList = []

**for** pat **in** patientLabels[misclassifiedTopDown[0]]:

patientList.append(pat)

print("The accuracy is **%f**"%(1-len(misclassifiedTopDown[0])/364))

print(patientList)

In [ ]:

TPR = len(np.intersect1d(pT[0][pT[0]<176],P))/len(P)

FPR = 1 - len(np.intersect1d(pT[0][pT[0]<176],P))/len(P)

print("The true Positie rate is **%f** and the False Positive rate it **%f**"%(TPR,FPR) )

print("The size of the clusters are **%d** for the positve group and **%d** for the negative group"%(len(pT[0]),len(nT[0])))

In [ ]:

len(misclassifiedTopDown[0])

In [ ]:

**import** **matplotlib\_venn** **as** **v**

**from** **matplotlib** **import** pyplot **as** plt

pBset = set(pB[0])

pTset = set(pT[0])

common = pBset & pTset

v.venn2([pBset,pTset],set\_labels = ('Bottom Up','TopDown'))

plt.title('Comparison of Data Points in common classified as Case by Top Down and Bottom Up')

plt.show()

pBCorrectset = set(pB[0][pB[0]<176])

pTCorrectset = set(pT[0][pT[0]<176])

v.venn2([pBCorrectset,pTCorrectset],set\_labels = ('Bottom Up','TopDown'))

plt.title('Comparison of Data Points in common correctly classified as Case by Bottom Up and Top Down')

plt.show()

pBIncorrectset = set(pB[0][pB[0]>175])

pTIncorrectset = set(pT[0][pT[0]>175])

v.venn2([pBIncorrectset,pTIncorrectset],set\_labels = ('Bottom Up','TopDown'))

plt.title('Comparison of Data Points in common incorrectly classified as Case by Bottom Up and Top Down')

plt.show()

In [ ]:

len(np.intersect1d(pB[0][pB[0]<176],P))/len(P)

In [ ]:

*#second iteration*

group0 = np.where(estimator.labels\_==0)[0]

group1 = np.where(estimator.labels\_==1)[0]

group0X = X.iloc[group0]

group1X = X.iloc[group1]

estimator = KMeans(n\_clusters = 2)

estimator.fit(group0X)

group0ClustersubsetLabels = estimator.labels\_

estimator = KMeans(n\_clusters = 2)

estimator.fit(group1X)

group1ClustersubsetLabels = estimator.labels\_

In [ ]:

group0 = np.where(group0ClustersubsetLabels==0)[0]

group1 = np.where(group0ClustersubsetLabels==1)[0]

group2 = np.where(group1ClustersubsetLabels==0)[0]

group3 = np.where(group1ClustersubsetLabels==1)[0]

*#split into four groups*

group2level0X = group0X.iloc[group0]

group2level1X = group0X.iloc[group1]

group2level2X = group1X.iloc[group2]

group2level3X = group1X.iloc[group3]

topDown2ndLevelDict = {'group0':group2level0X,

'group1':group2level1X,

'group2':group2level2X,

'group3':group2level3X

}

topDown2ndLevelStats = {'group0':0,

'group1':0,

'group2':0,

'group3':0

}

**for** i **in** range(0,4):

group = 'group**{0}**'.format(i)

tempAccuracy = sum(topDown2ndLevelDict[group].index<176)/len(topDown2ndLevelDict[group])

clusterSize = len(topDown2ndLevelDict[group])

**if** tempAccuracy>.5:

topDown2ndLevelStats[group] = [clusterSize, tempAccuracy]

**else**:

topDown2ndLevelStats[group] = [clusterSize,1-tempAccuracy]

topDown2ndLevelStats

In [ ]:

*#estimators of the 4 nodes*

estimator = KMeans(n\_clusters = 2)

estimator.fit(group2level0X)

group0level2ClustersubsetLabels = estimator.labels\_

estimator = KMeans(n\_clusters = 2)

estimator.fit(group2level1X)

group1level2ClustersubsetLabels = estimator.labels\_

estimator = KMeans(n\_clusters = 2)

estimator.fit(group2level2X)

group2level2ClustersubsetLabels = estimator.labels\_

estimator = KMeans(n\_clusters = 2)

estimator.fit(group2level3X)

group3level2ClustersubsetLabels = estimator.labels\_

group0 = np.where(group0level2ClustersubsetLabels==0)[0]

group1 = np.where(group0level2ClustersubsetLabels==1)[0]

group2 = np.where(group1level2ClustersubsetLabels==0)[0]

group3 = np.where(group1level2ClustersubsetLabels==1)[0]

group4 = np.where(group2level2ClustersubsetLabels==0)[0]

group5 = np.where(group2level2ClustersubsetLabels==1)[0]

group6 = np.where(group3level2ClustersubsetLabels==0)[0]

group7 = np.where(group3level2ClustersubsetLabels==1)[0]

group3level0X = group2level0X.iloc[group0]

group3level1X = group2level0X.iloc[group1]

group3level2X = group2level1X.iloc[group2]

group3level3X = group2level1X.iloc[group3]

group3level4X = group2level2X.iloc[group4]

group3level5X = group2level2X.iloc[group5]

group3level6X = group2level3X.iloc[group6]

group3level7X = group2level3X.iloc[group7]

topDown3rdLevelDict = { 'group0':group3level0X,

'group1':group3level1X,

'group2':group3level2X,

'group3':group3level3X,

'group4':group3level4X,

'group5':group3level5X,

'group6':group3level6X,

'group7':group3level7X

}

topDown3rdLevelStats = { 'group0':0,

'group1':0,

'group2':0,

'group3':0,

'group4':0,

'group5':0,

'group6':0,

'group7':0

}

**for** i **in** range(0,8):

group = 'group**{0}**'.format(i)

tempAccuracy = sum(topDown3rdLevelDict[group].index<176)/len(topDown3rdLevelDict[group])

clusterSize = len(topDown3rdLevelDict[group])

**if** tempAccuracy>.5:

topDown3rdLevelStats[group] = [clusterSize,sum(topDown3rdLevelDict[group].index>176), tempAccuracy]

**else**:

topDown3rdLevelStats[group] = [clusterSize,sum(bottomUp3rdLevelDict[group].index<176),1-tempAccuracy]

topDown3rdLevelStats

Problem 2 Script:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.preprocessing** **import** Imputer

**from** **sklearn.cluster** **import** KMeans

**from** **scipy.stats.stats** **import** pearsonr

In [ ]:

df = pd.read\_csv('/Users/Daniel/Documents/Data Mining 514/HW3/allPatientsKnnImputedData.csv',low\_memory=**False**)

X = df.drop('Classes',axis=1)

features = X.columns

*#X.replace(to\_replace='?', value = 'NaN', inplace=True)*

*#imp = Imputer(missing\_values='NaN', strategy='mean', axis=0)*

*#X=imp.fit\_transform(X)*

Y = df['Classes']

X = pd.DataFrame(X,columns=features)

In [ ]:

*#get covariance matrix, eigenvalues and eigenvectors*

covMatrix = X.cov()

w,v = np.linalg.eig(covMatrix)

wOrdered = sorted(w,reverse=**True**)

In [ ]:

*#plot eigenvalues*

f, axarr = plt.subplots(2, sharex=**False**)

axarr[0].plot(np.linspace(1,20,20),wOrdered[:20],'o')

axarr[0].set\_title('Top 20 Eigenvalues')

*#axarr[1].set\_title('All Eigenvalues')*

axarr[1].plot(np.linspace(1,len(wOrdered),len(wOrdered)),wOrdered,'o')

plt.show()

In [ ]:

wOrderedPercent =wOrdered/sum(wOrdered)

f, axarr = plt.subplots(2, sharex=**False**)

axarr[0].plot(np.linspace(1,20,20),wOrderedPercent[:20],'o')

axarr[0].set\_title('Top 20 Eigenvalues')

axarr[0].set\_ylabel('Percent variation explained by eigenvectors')

*#axarr[1].set\_ylabel('Percent variation explained by eigenvectors')*

*#axarr[1].set\_title('All Eigenvalues')*

axarr[1].plot(np.linspace(1,len(wOrderedPercent),len(wOrderedPercent)),wOrderedPercent,'o')

plt.show()

In [ ]:

eig\_pairs = [(np.abs(w[i]), v[:,i]) **for** i **in** range(len(w))]

eig\_pairs.sort(key=**lambda** x: x[0], reverse=**True**)

eigenvectorsSorted = [x[1] **for** x **in** eig\_pairs]

eigenvectorsSorted[:4]

In [ ]:

**def** new\_euclidean\_distances(X, Y=**None**, Y\_norm\_squared=**None**, squared=**False**):

**return** pearsonr(X,Y)

KMeans.euclidean\_distances = new\_euclidean\_distances

results = np.zeros(18)

**for** i **in** range(0,18):

eig = eigenvectorsSorted[:(i+2)]

estimators = KMeans(n\_clusters=2).fit(X.dot(np.asarray(eig).T))

results[i] = max(sum(abs((estimators.labels\_+1)- Y.as\_matrix()))/len(Y),1-sum(abs((estimators.labels\_+1)- Y.as\_matrix()))/len(Y))

In [ ]:

plt.plot(np.linspace(2,18,18),results,'o')

plt.ylabel('Percent Accuracy')

plt.xlabel('Number of PCA variables')

plt.title('Accuracy vs. Number of PCA variables')

In [ ]:

plt.show()

In [ ]:

**from** **sklearn.decomposition** **import** PCA **as** sklearnPCA

results = np.zeros(18)

**for** i **in** range(0,18):

sklearn\_pca = sklearnPCA(n\_components=(i+2))

sklearn\_transf = sklearn\_pca.fit\_transform(X)

estimators = KMeans(n\_clusters=2).fit(sklearn\_transf)

results[i] = max(sum(abs((estimators.labels\_+1)- Y.as\_matrix()))/len(Y),1-sum(abs((estimators.labels\_+1)- Y.as\_matrix()))/len(Y))

In [ ]:

plt.plot(np.linspace(2,18,18),results,'o')

plt.ylabel('Percent Accuracy')

plt.xlabel('Number of PCA variables')

plt.title('Accuracy vs. Number of PCA variables')

plt.show()

Problem 3 Script:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.decomposition** **import** TruncatedSVD **as** sklearnSVD

**from** **sklearn.cluster** **import** KMeans

**from** **scipy.stats.stats** **import** pearsonr

**def** new\_euclidean\_distances(X, Y=**None**, Y\_norm\_squared=**None**, squared=**False**):

**return** pearsonr(X,Y)

KMeans.euclidean\_distances = new\_euclidean\_distances

In [ ]:

df = pd.read\_csv('/Users/Daniel/Documents/Data Mining 514/HW3/allPatientsKnnImputedData.csv',low\_memory=**False**)

X = df.drop('Classes',axis=1)

features = X.columns

*#X.replace(to\_replace='?', value = 'NaN', inplace=True)*

*#imp = Imputer(missing\_values='NaN', strategy='mean', axis=0)*

*#X=imp.fit\_transform(X)*

Y = df['Classes']

X = pd.DataFrame(X,columns=features)

In [ ]:

results = np.zeros(18)

**for** i **in** range(0,18):

sklearn\_SVD = sklearnSVD(n\_components=(i+2))

sklearn\_transf = sklearn\_SVD.fit\_transform(X)

estimators = KMeans(n\_clusters=2).fit(sklearn\_transf)

results[i] = max(sum(abs((estimators.labels\_+1)- Y.as\_matrix()))/len(Y),1-sum(abs((estimators.labels\_+1)- Y.as\_matrix()))/len(Y))

In [ ]:

plt.plot(np.linspace(2,18,18),results,'o')

plt.ylabel('Percent Accuracy')

plt.xlabel('Number of features in SVD space')

plt.title('Accuracy vs. Number of features in SVD space')

plt.show()

In [ ]:

plt.plot(np.linspace(1,len(estimators.explained\_variance\_),len(sklearnSVD.fit(X).explained\_variance\_)),estimators.explained\_variance\_,'o')

In [ ]:

sklearn\_SVD = sklearnSVD(n\_components = 40)

sklearn\_SVD.fit(X)

sklearn\_SVD.explained\_variance\_

plt.plot(np.linspace(1,len(sklearn\_SVD.explained\_variance\_),len(sklearn\_SVD.explained\_variance\_)),sklearn\_SVD.explained\_variance\_,'o')

In [ ]:

plt.show()