MSA220/MVE440: Statistical Learning for Big Data
Exam Spring 2018

Wade Rosko 960111-T116
MPCAS Chalmers University of Technology

June 2018

Contents

1 Question 1: Mini Report

During the semester I completed two of the Minis on my own, and three with partners. Below I discuss my methods and findings for each of the Minis in chronological order.

1.1 Mini 1: How many clusters?

Clustering methods in statistical learning are entirely dependent upon the statistician and the data being utilized. In my case, I decided to investigate how some standard clustering methods perform on a dataset that has classes.

1.1.1 Data Set

For this task I used the Turkiye Student Evaluation Data Set (1). There were a total of 5820 instances in the data, each containing 33 attributes. This specific set consists of course evaluation data provided by students. While running clustering models on the data, I made note of the **Instructor** and **Class** attributes. I figured that if I were to cluster the data, I would either see three groups where they are aligned with the respective professors. I also thought that there may be around 13 clusters since there were a total of 13 different classes offered.

1.1.2 Simulation Setup

Initially, I needed to prepare the data for the simulations and models I ran. To prepare the data I normalized the values, and then I removed the **Instructor** and **Class** attributes. I did this so that we will be able to run the clustering methods without them recognizing the signals as cues to determine the number of classes. Additionally, I ran the experiments with many trials, on the whole set of data.

1.1.3 Methods and Models

To evaluate how different methods choose the number of clusters for this data, I decided to use the **factoextra** package (2), the **NbClust** package (3), and the **fpc** package (4).

With **factoextra** I use the total within sum square (wss) method, silhouette method, and the gap statistic method.

My results are given in Figures 1, 2, and 3. For the wss method I used the elbow method to determine the optimal number of clusters and found that $\mathbf{k} = 3$ is optimal. The silhouette method gives an optimal $\mathbf{k} = 2$, and the gap statistic gives an optimal cluster number of $\mathbf{k} = 14$.

Implementing the **NbClust** method using all of the different indices provided in the package with k-means as the method, there is a tie for votes between $\mathbf{k} = 2$ and $\mathbf{k} = 3$. The full results can be seen in Table 1.

For visualization purposes, I have included the results from the Hubert Index as well as the D Index. As the package documentation states (3), the Hubert and D Indices are graphical methods for determining the number of clusters. Both methods give votes for $\mathbf{k} = 4$ clusters as seen in Figures 4 and 5.

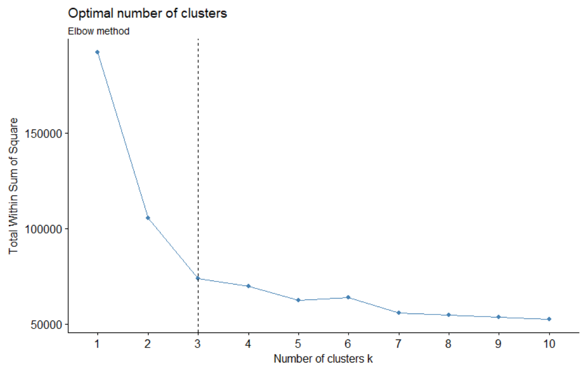


Figure 1: WSS method results for 10 clusters. Elbow occurs around **k** = 3 clusters.

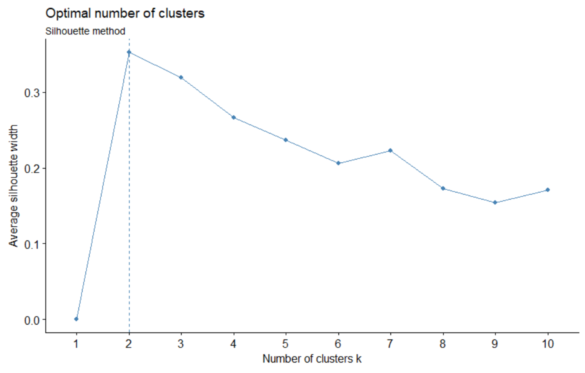


Figure 2: Silhouette method results for 10 clusters. Peak occurs at **k** = 2 clusters.

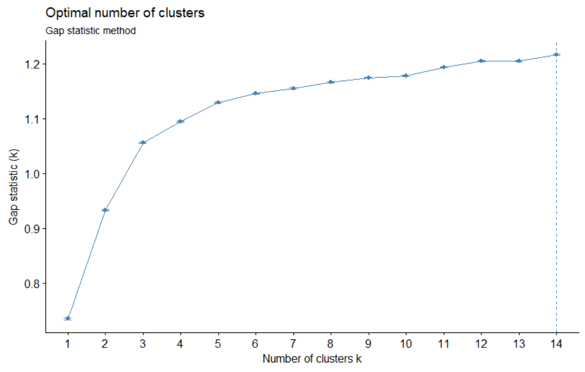


Figure 3: Gap statistic method results. The optimal cluster number is **k** = 14 clusters.

NbClust Votes	Number of Clusters
7	2
7	3
6	4
1	7
1	13
1	14

Table 1: Table containing the results from the **NbClust** function and respective package. There was a tie between 2 and 3 clusters as the optimum.

At the end I wanted to see how the data would be clustered if we were to apply a kmeans method with known cluster sizes. I did this with the **stats** package (5) and then visualized with **fpc** (4). The plots can be viewed in Figure 6 through Figure 9.

1.1.4 Discussion

Overall, my results were mixed. The different methods that I utilized all gave varied results. The main results gave votes for 2, 3, 4, and 14 clusters for this data set. As I stated before, I expected there to be a clear distinction between the three different

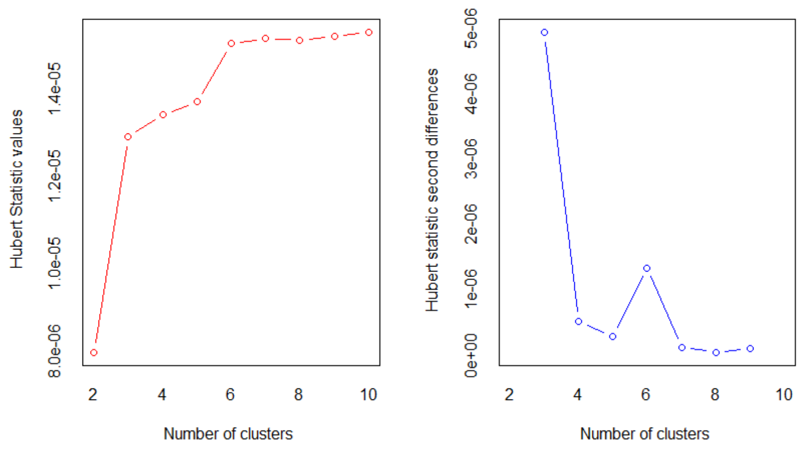


Figure 4: Hubert Index results with a significant knee in the statistic second difference at **k** = 4 clusters.

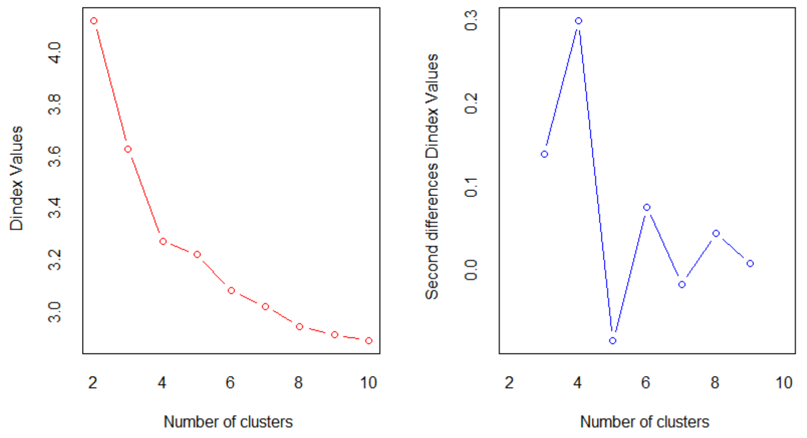


Figure 5: D Index results with a significant peak in the statistic second difference at **k** = 4 clusters.

teachers. With **factoextra**, wss was the only one that gave 3 clusters as the optimal, but **NbClust** gave a tie between 2 and 3 for the different indices. I believe that gap statistic was not a productive method to decide on the number of clusters because it continues to increase if I add additional k values. I could have set the max k value to be 13, and it would have aligned with the number of different classes that were evaluated by the students.

The votes for 2 clusters in the Silhouette method and **NbClust** are understandable since the majority of the data was from questions rated on a scale of 1-5 (poor to excellent). Thus, the results were more than likely clustered into two sides where the evaluation's average responses were positive or negative.

My experiments, coupled with the discussion in class tells me that we need to consider both the clustering index and method. We need to experiment with different methods

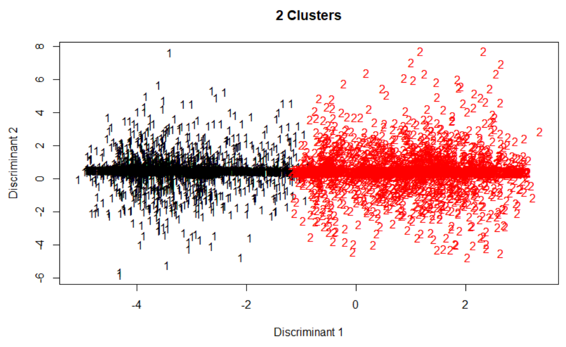


Figure 6: Visualization for 2 clusters.

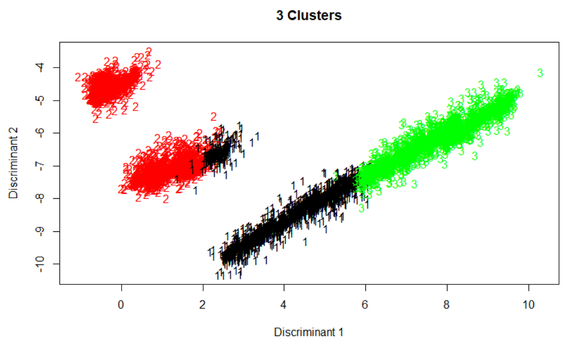


Figure 7: Visualization for 3 clusters.

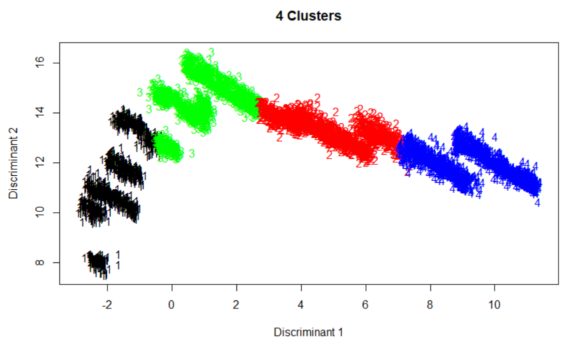


Figure 8: Visualization for 4 clusters.

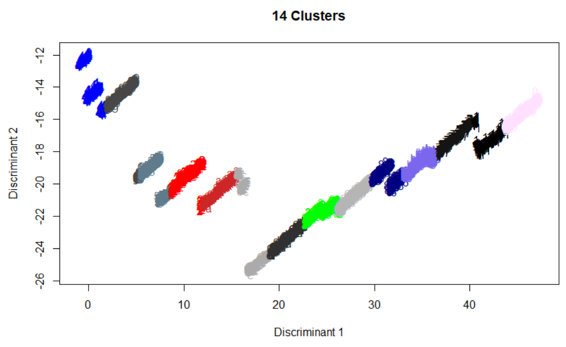


Figure 9: Visualization for 14 clusters.

for different data sets because some work better than others depending upon the type of data. This is a hard task to do because we need to take the time to various methods. We can couple the methods with sub-sampling and other techniques learned later in the course to speed up selection time. More experience with various data sets will allow me to gain a better understanding of which methods work best with the current data I am working with.

1.2 Mini 2: Classification Comparison

For Mini 2, the task was to choose 2 to 3 datasets and then compare kNN, logistic and multinomial regression, discriminant analysis, CART, RandomForest and other methods. We were then supposed to experiment with ensemble methods. For the assignment I looked at RandomForest, CART, kNN, logistic regression, Naive Bayes, penalized discriminant analysis (DA), quadratic DA, linear DA, bagged CART ensemble method, and stacked RandomForest.

1.2.1 Data Set

This task used the HTRU2 pulsar data set (6), and the Skin Segmentation data set (7). The pulsar data contains 17898 instances with 8 attributes and a defined class. It is unbalanced in its classes, so kappa values should be used to better estimate the strength of the models. The skin data set has 245057 instances with three variables (R,G,B image values) and a given binary class label. It too was unbalanced in its classes.

1.2.2 Simulation Setup

To setup the simulation I first needed to manipulate the data. For the pulsar data I used the whole data set, and split it by 80% for training and 20% for testing. For the skin data I used 40000 randomly selected instances, and then did a similar 80%-20% split. I then scaled and centered the data sets.

1.2.3 Methods and Models

For each of the datasets I ran the **caret** package (8), and included random forest, CART, knn, logistic regression, kNN, Naive Bayes, penalized DA, quadratic DA, and linear DA. I decided to include the different discriminant analyses because LDA and QDA are more flexible at classification. Each model was trained with 10 fold cross validation repeated 3 times. An ensemble method that could be implemented with **caret** was treebag. This is a bagging method that is applied with the CART technique. To explore another ensemble method, I utilized the **caretEnsemble** package (9) and ran a random forest ensemble on top of lda, qda, rpart, glm, knn, and svmradial models. The results for HTRU2 can be found in Table 2 and visualized in Figure 10 and the results for the Skin Segmentation can be found in 3 and visualized in Figure 11.

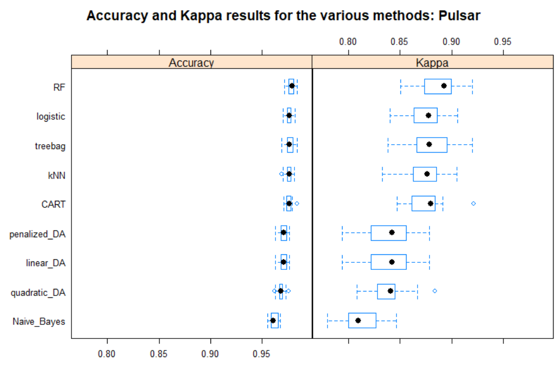


Figure 10: HTRU2 Pulsar results with **caret**.

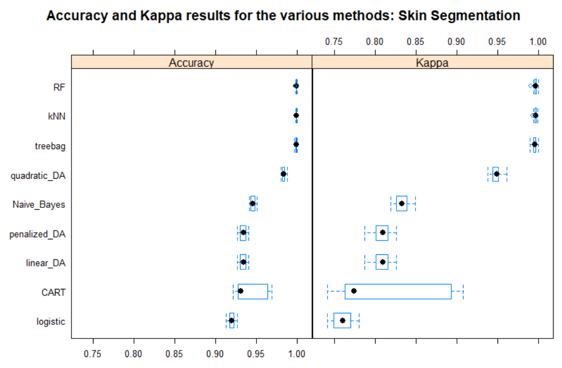


Figure 11: Skin Segmentation results with **caret**.

1.2.4 Discussion

Both sets of data, despite their difference in size and uneven class ratios, both saw random forest and the random forest ensemble return the best kappa results.

For the pulsar data, HTRU2, the best standalone method was random forest with mean accuracy and kappa of 97.88% and 88.80% respectively. The worst method was Naive Bayes with mean accuracy and kappa of 96.17% and 81.15% respectively. Treebag ensemble performed similarly to random forest which is expected because their algorithms differ in only in mtry which is the number of features to search over in a tree. It is interesting to note that QDA performed worse than LDA and PDA, this may be due to the equality of covariances in the predictor variables (QDA is able to work with data

HTRU2 Pulsar Data Accuracy and Kappa Results				
Method	Accuracy Mean	Accuracy Median	Kappa Mean	Kappa Median
Random Forest	0.9788157	0.9797415	0.8880336	0.89268
treebag (CART ensemble)	0.9775126	0.9776536	0.8813587	0.8812692
CART	0.9761853	0.9769392	0.8743934	0.8799119
kNN	0.9765343	0.9769473	0.8749744	0.8760936
Logistic	0.976721	0.9769553	0.8754723	0.8778923
Naive Bayes	0.9617523	0.9612296	0.8114976	0.8096356
Penalized DA	0.9711807	0.9713887	0.8403634	0.8423995
Quadratic DA	0.9682007	0.9685754	0.8378822	0.8410353
Linear DA	0.9711807	0.9713887	0.8403634	0.8423995
RF Ensemble	0.9823777	-	0.9073895	-

Table 2: Table containing the results from the **caret** package for the Pulsar Data set as well as the ensemble result.

Skin Segmentation Data Accuracy and Kappa Results				
Method	Accuracy Mean	Accuracy Median	Kappa Mean	Kappa Median
Random Forest	0.9986875	0.99875	0.9960452	0.9962327
treebag (CART ensemble)	0.9770937	0.9769553	0.8790357	0.8785567
CART	0.9422813	0.9303019	0.8166316	0.7740477
kNN	0.9987292	0.99875	0.9961742	0.9962368
Logistic	0.9193332	0.9190625	0.7593697	0.7596254
Naive Bayes	0.9455938	0.945625	0.8328882	0.8327285
Penalized DA	0.9332082	0.93375	0.8078688	0.8088006
Quadratic DA	0.9833958	0.9835938	0.9485612	0.9492064
Linear DA	0.9331977	0.93375	0.8078357	0.8088006
RF Ensemble	0.9995937	-	0.9987747	-

Table 3: Table containing the results from the **caret** package for the Skin Segmentation data set as well as the ensemble result.

that does not have good equality of covariances in the predictor variables). Also, logistic regression performs significantly better than LDA. The random forest ensemble method with stacking is the best performing model with an accuracy and kappa of 98.24% and 90.73% respectively.

The Skin Segmentation data had the kNN method perform the best with an accuracy and kappa of 99.87% and 99.62% respectively. This is interesting, but makes sense since the data is based upon RGB values for pictures of skin. Instances that are marked as skin would be grouped together based upon similar RGB values. The worst performing method was logistic regression with accuracy and kappa values of 91.93% and 75.94% respectively. Treebag has a good accuracy but loses runner-up status to random forest and QDA. Just like for HTRU2, the random forest ensemble performed the best with an accuracy and kappa of 99.96% and 99.88% respectively.

From my results I have found that the random forest ensemble method with stacking performs best when compared to all of the methods. It is useful for the HTRU2 data set because it increases kappa performance by a little over 1%, but it may not be useful for the Skin Segmentation data set because it only boosts kappa performance by 0.2%. Time can be saved by just using random forest or kNN for this set. The ensemble was selected through some trial and error, and I found that the models' correlations were all below 75% which is optimal for building an ensemble. These two sets of data were binary class problems, so it would become more difficult to find an ensemble method that is as efficient as the one used here (10).

1.3 Mini 3: Many Classes

Mini 3's task for Many Classes was to investigate how wrappers and filters work for feature selection. To do this I used the **caret** package (8), **mlr** package (11), and the **FSelector** package (12).

1.3.1 Data Set

For this Mini I decided to generate several artificial sets of data, and then perform feature selection on them. To do this I used a python script to generate 3000 samples of 5 with 5 features that contained 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, and 30 centers. These sets of data thus contained the respective number of classes. After this, I added an additional feature that was the sum of the 5 original features for each sample. I then followed the example code and added 10 additional noisy features that did not have any correlation to the sample's classification.

1.3.2 Simulation Setup

The simulation was setup by taking each of the sets of data and appending 10 unrelated features described before. The set was then standardized and then fed through the different models. Even though the number of samples did not change for each set of data, the time it took to run increased considerably between the set with 9 centers to the set with 30 centers.

1.3.3 Methods and Models

Many different methods were run for feature selection including: linear discriminant analysis (LDA), Naive Bayes (NB), random forest (RF), PAM (shrunken centroids - shrCent), sparse LDA (spLDA), sparse logistic regressions (spLogcaret, spLog1SE, spLog1min, spLogGroup1se, spLogGroupmin), LDA recursive feature elimination (LDA-rfe). The results for the kappa values for 3, 5, 10, 15, 20, and 30 centers can be found in Table 4 and in Figure 12.

Using the filter method to find the most important features from my data yields Table 5. I averaged all of the results from the filter method since they were all within 0.02 of each other.

1.3.4 Discussion

For this Mini I ran many different models on the data. From Table 4 and from Figure 12 it's visible that certain methods, such as shrunken centroids, and the sparse logistic regressions (except for **caret**'s) kappa accuracy deteriorates as I increase the number

Method	**Centers** 3	5	10	15	20	30
LDA	0.9979973	0.986664	0.997036	0.99857132	0.98922893	0.99218326
NB	0.9959945	0.9849972	0.9925898	0.99761887	0.986419	0.98436651
RF	0.9959945	0.989998	0.9940721	0.99428527	0.9859508	0.98850489
ShrCent	1	0.9766627	0.961479	0.99285662	0.94615506	0.88279943
spLDA	0.9959945	0.9866642	0.997036	0.99857132	0.98829241	0.990344
spLogcaret	0.9979973	0.9733284	0.997036	0.99857132	0.98829241	0.990344
spLog1SE	0.9979973	0.9766619	0.9866622	0.07522342	0.04898658	0.02807656
spLog1min	0.9959945	0.9749949	0.9881441	0.07474448	0.04851437	0.02716123
spLogGroup1se	0.9979973	0.979996	0.9940719	0.07379855	0.04709969	0.31022222
spLogGroupmin	0.9979973	0.979996	0.9940719	0.07379855	0.04709969	0.31022222
LDA-rfe	0.9919889	0.9766616	0.997036	0.9847606	0.97471154	0.96919368

Table 4: Table containing the kappa results for 3, 5, 10, 15, 20, and 30 centers

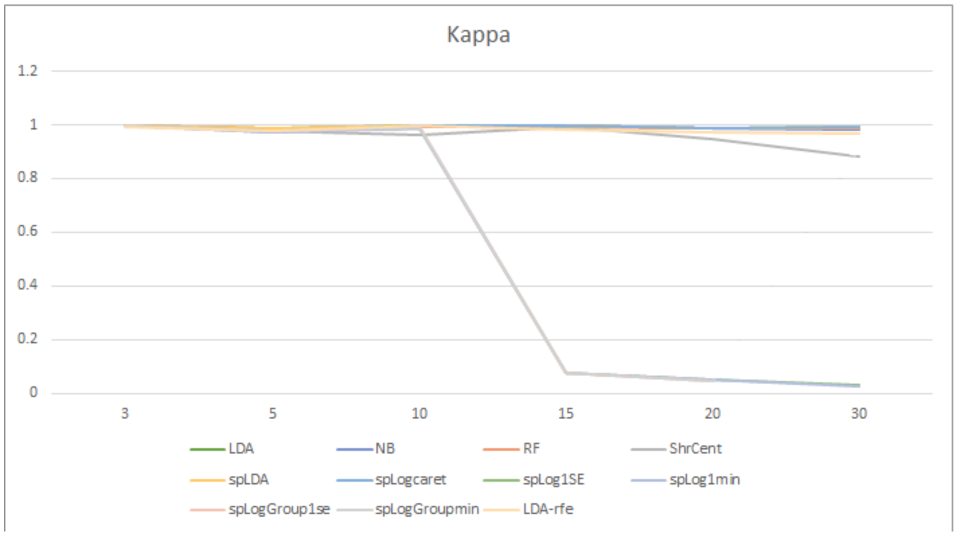


Figure 12: Kappa results for 3, 5, 10, 15, 20, 30 centers.

of centers/classes. The LDA recursive feature elimination is able to keep fairly good kappa while reducing the features used to the original 6 before the uncorrelated data was added. While it is not in the table, Naive Bayes recursive feature elimination was utilized as well. It returned similar results to the LDA. It is interesting to note that V6 had the most significant importance compared to all of the other original attributes. As stated previously, V6 was made by summing V1-V5 for each sample. I am unsure why this happened.

Overall, the takeaways from my experiment are that feature selection most probably helps for most data. In my case it did not perform as well as other techniques, but this was due to the data being so clean. It was an artificial data set that did not have much noise. More than likely the data was linearly separable. So filtering and wrapper methods can be useful depending on the type of data being used. Because of this it is important to run experiments to see how different methods run and to keep the computational power needed in mind.

Attribute	Importance
V1	0.36
V2	0.29
V3	0.19
V4	0.27
V5	0.31
V6	0.43
A.1	0.00
A.2	0.00
A.3	0.00
A.4	0.00
A.5	0.00
A.6	0.00
A.7	0.00
A.8	0.00
A.9	0.00
A.10	0.00

Table 5: Average attribute importance over all of the data sets. Variance was very low. V1 - V5 attributes were the 5d generated data, V6 was the sum of the V1-V5 attributes for each sample as described in the data set section. A.1-A.10 were noisy uncorrelated data.

1.4 Mini 4/5: Data Representation

For Mini 4/5 I looked at data representations and how these indicate clusters in the data. I looked at spectral clustering, then running PCA reduction and feeding the result into spectral clustering, tSNE data representation, tSNE then spectral clustering, and locally linear embedding then spectral clustering.

1.4.1 Data Set

For this Mini I used the standard Wine data set from UCI but accessed from **rattle** (13). I also used the Occupancy Detection data set (14). It is a set containing 20560 instances with 6 attributes and a binary class. I removed the date-time attribute because I did not think that it would be relevant for the methods and models I ran. For both of the sets I normalized and centered the data.

1.4.2 Simulation Setup

The simulation was setup by loading the respective data sets. The simulation was programmed so that the similarity matrices would be produced for the raw data, spectral clustering, tSNE, PCA, and LLE. Then the eigenvalues for the previous data after run through spectral clustering was programmed.

1.4.3 Methods and Models

As stated above, I found the similarity matrices for the wine and occupancy data. I first found the raw similarity matrix (Figures 13 and 17), then the tSNE similarity matrix (Figures 14 and 18), the PCA similarity matrix (Figures 15 and 19), and finally the LLE similarity matrix (Figures 16 and 20).

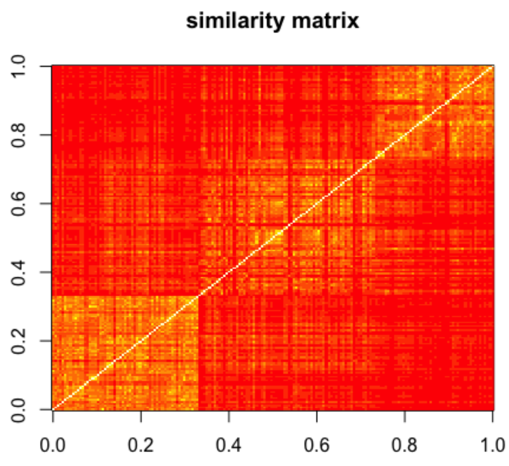


Figure 13: Raw wine similarity matrix

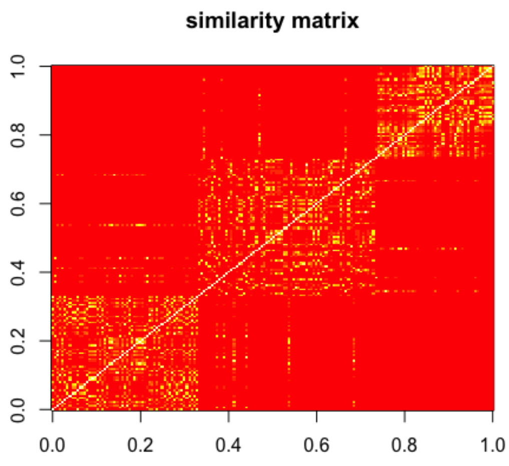


Figure 14: Similarity matrix for wine tSNE

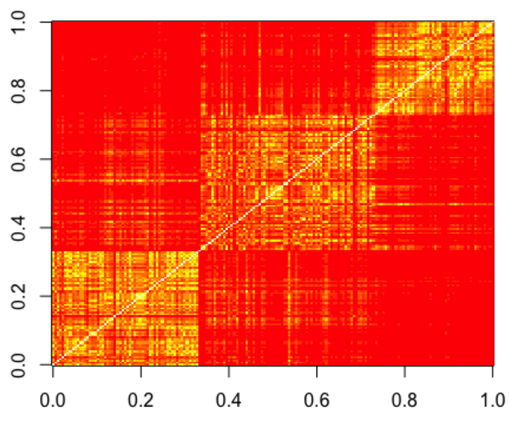


Figure 15: Similarity matrix for wine PCA

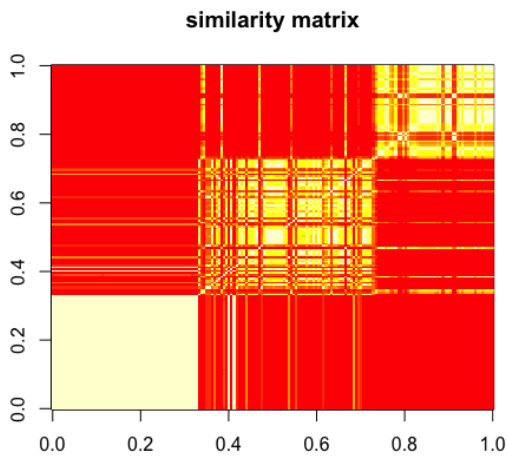


Figure 16: Similarity matrix for wine LLE

After this I found the visualization for tSNE for the respective data sets (Figures 21 and 22. Note how both visualizations form clusters for the respective actual number of classes in the data sets (3 for wine, 2 for occupancy).

From here I used the reduction that tSNE, PCA, and LLE give and then passed the reduced data through spectral clustering. The first 10 eigenvalues are given for the methods for wine in Figures 23 through 26 and similarly for occupancy in Figures 27 through 30.

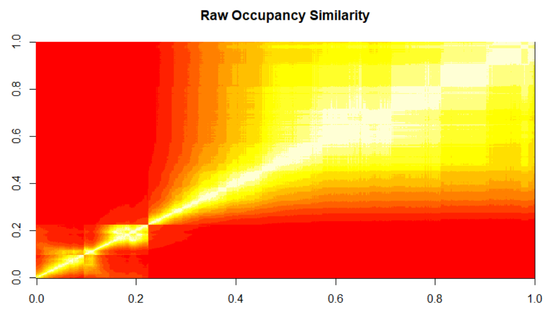


Figure 17: Raw Occupancy similarity matrix

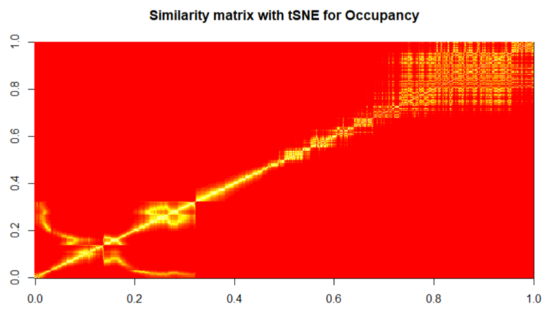


Figure 18: Similarity matrix for Occupancy tSNE

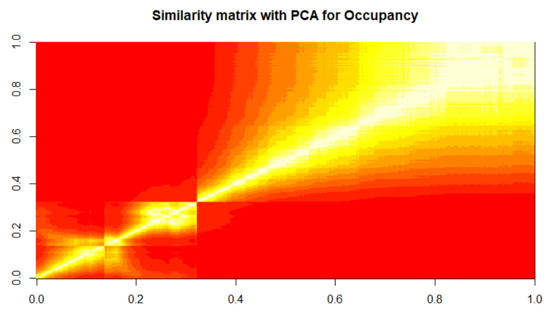


Figure 19: Similarity matrix for Occupancy PCA

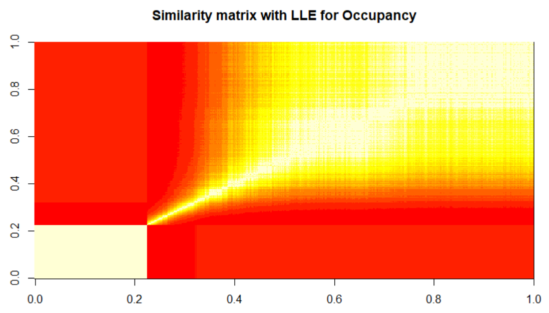


Figure 20: Similarity matrix for Occupancy LLE

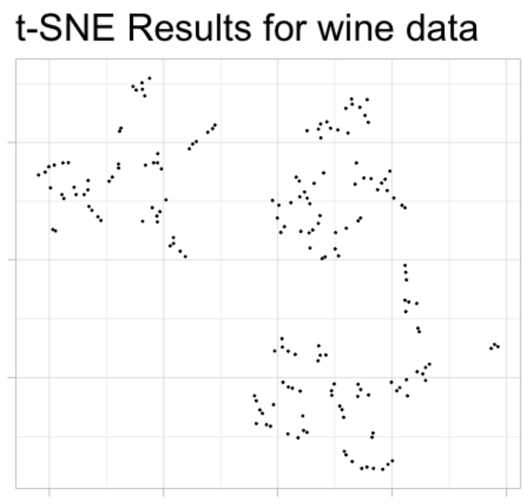


Figure 21: Wine tSNE visualization

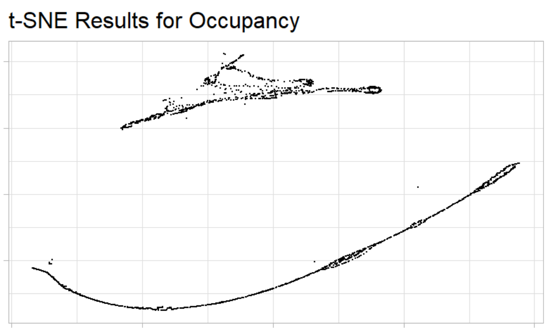


Figure 22: Occupancy tSNE visualization

1.4.4 Discussion

For these data sets it appears that reducing the data through different representations can both harm or help find the optimal number of clusters. Observing the wine data

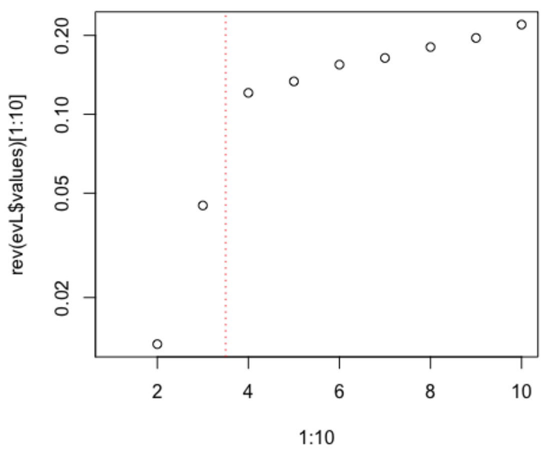


Figure 23: Eigenvalues for Wine spectral clustering

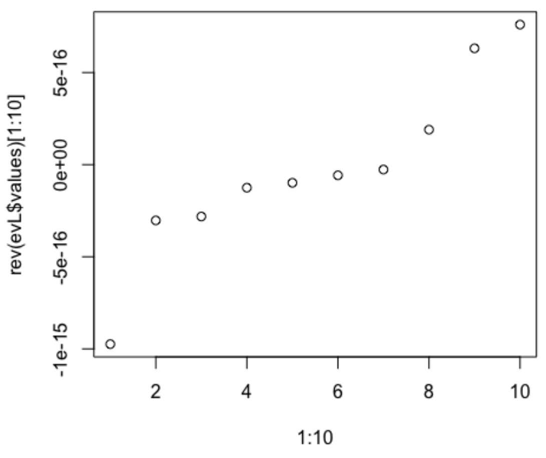


Figure 24: Eigenvalues for Wine tSNE then spectral clustering

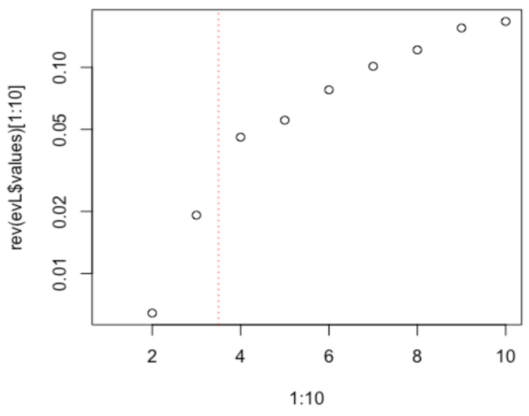


Figure 25: Eigenvalues for Wine PCA then spectral clustering

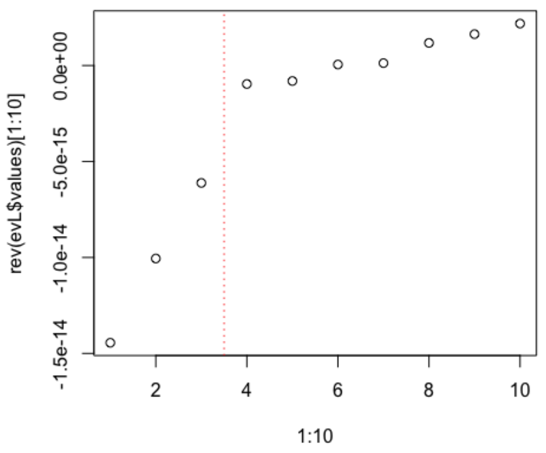


Figure 26: Eigenvalues for Wine LLE then spectral clustering

first, the raw similarity matrix shows outlines for 3 groups. The spectral clustering on this returns a plot of the eigenvalues where 3 or 4 eigenvalues would be optimal. If we go to tSNE next, the similarity matrix is more defined into three groups, but the spectral clustering on this data shows that two eigenvalues may be optimal. The tSNE representation clearly shows three clusters, so we would expect that the spectral clustering would only need 3 eigenvalues. Further, the PCA similarity matrix is also better defined than the raw data,and 3 or 4 eigenvalues are optimal. LLE gives an artifact in the similarity matrix, but passing it through spectral clustering yields optimal eigenvalues at 3 or 4.

Initially the raw data similarity matrix for room occupancy shows odd shaped clus-

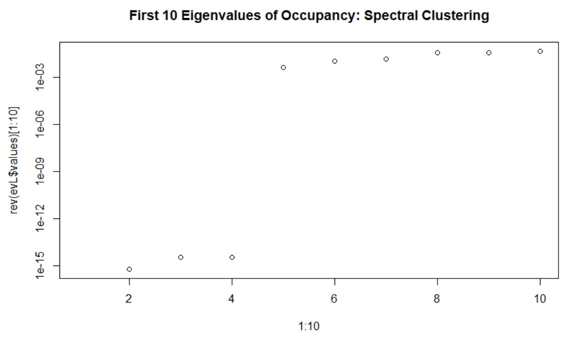


Figure 27: Eigenvalues for Occupancy spectral clustering

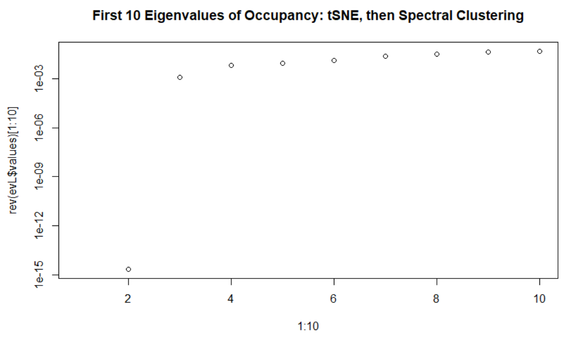


Figure 28: Eigenvalues for Occupancy tSNE then spectral clustering

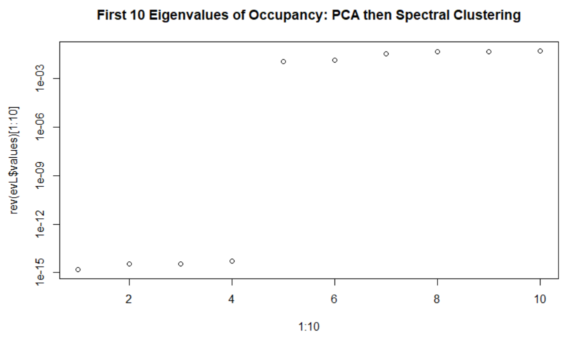


Figure 29: Eigenvalues for Occupancy PCA then spectral clustering

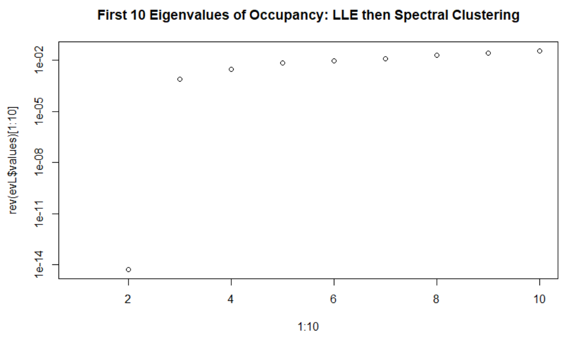


Figure 30: Eigenvalues for Occupancy LLE then spectral clustering

ters and the eigenvalues from the spectral clustering suggest that 4 or 5 eigenvalues are optimal. The tSNE representation reveals two types of clusters, and from here I expected that the similarity matrix would be more defined than the raw data. At first glance it appears less defined than the raw matrix. Passing it through spectral clustering reveals 3 eigenvalues to be optimal. After using PCA the similarity matrix appears to be convoluted and the optimal eigenvalue is around 5. This is a large jump, and we know that this method does not give a good representation of what I am looking for in the data. Finally with LLE there is also an artifact in the bottom left of the similarity matrix. Passing it through spectral clustering gives 3 eigenvalues as the optimum.

Comparing the two sets of data, I would conclude that using these graphical approaches is a nice way to provide an intuitive and useful interpretation of the data structure. Defining the affinity metric and parameters definitely needs to be examined on a case by case basis. The wine data set benefits more from the raw spectral clustering than the occupancy data. Spectral clustering is a computationally heavy method, and it was enlightening to see the decrease in run time after applying the other methods to reduce the data first. I believe that modifying some of the parameters could help obtain better results for the occupancy data.

1.5 Mini 6: Random Forest Variants

For Mini 6, I decided to investigate the random forest variants. For this, I looked at Hoeffding trees and their efficiency versus a random forest. We use both the Hoeffding method for online classification where we encounter a data stream.

1.5.1 Data Set

Once again, I used the Room Occupancy data set (14) with 20560 instances. Like before, I removed the dates that were in the original data. For the random forest itself, I did not modify the data except for the date removal. It was fed directly into the random forest function (15). For the Hoeffding tree, it is necessary to transform the data into a data stream to mimic online data.

1.5.2 Simulation Setup

To set up the simulation I loaded the occupancy data and then modified it as needed for the random forest algorithm and Hoeffding tree method. I then structured the program so that it would record run time and accuracy for the models being run with certain percentages of the data.

1.5.3 Methods and Models

Initially I ran the random forest algorithm on the whole set of data so that I could find the importance measure for all of the different attributes. The results are given in Table 6. From the table it appears the the only variables that are significantly important are

Variable	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
Temperature	26.93	31.80	39.54	902.84
Humidity	17.22	31.94	30.97	96.90
Light	92.26	291.24	200.74	3551.77
CO2	20.48	41.10	46.34	514.60
HumidityRatio	14.83	27.27	26.36	125.27

Table 6: The extractor function results for variable importance measures from **randomForest** (15).

Temperature, Light, and HumidityRatio. I initially took this information, and tested how the Hoeffding tree accuracy varied if I changed how the MOA model was fit by eliminating some of the variables. There was no significant change, at the most there were 1 or 2 extra instances that were classified correctly, so I kept all variables within the model due to the smaller complexity. The Hoeffding or Very Fast Decision Trees proposed by Domingos and Hulten (16), are able to learn from small sets of a large data stream. As data first comes in, the stream/algorithm picks the one-split tree (as described in class), Further data allows the next chunk to pick the next split. As the stream progresses, there may be additional splits added only when the algorithm deems it as a very good split.

After doing my initial test for the random forest and the variable selection for the Hoeffding tree, I then setup my program to record accuracy and time spent. The times for the two methods depending upon size of the dataset are given in Table 7 and Figure 31. The accuracy of the two methods are given in Table 8 and 32.

Fraction of Data	Hoeffding Tree Time (s)	Random Forest Time (s)
0.2	14.17013	13.48867
0.4	27.30032	65.82193
0.6	42.04664	145.19541
0.8	54.75511	272.52872
1	68.61288	438.4304

Table 7: Time comparison for Random Forest vs. Hoeffding Trees with certain fractions of data used

Fraction of Data	Hoeffding Tree Accuracy	Random Forest Accuracy
0.2	0.971060311	0.993190661
0.4	0.993555447	0.994892996
0.6	0.988326848	0.993433852
0.8	0.990211576	0.994102626
1	0.989153696	0.99348249

Table 8: Accuracy comparison for Random Forest vs. Hoeffding Trees with certain fractions of data used

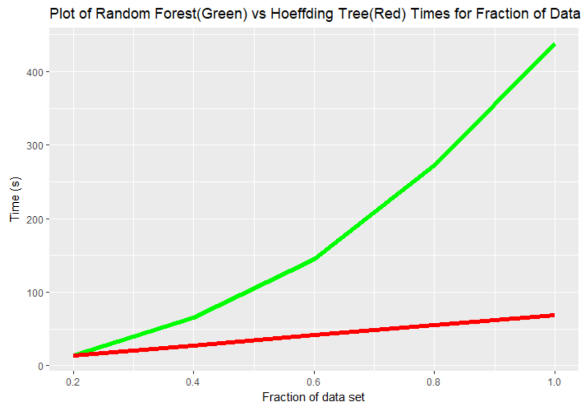


Figure 31: Time results for random forest and Hoeffding trees for given percentages of occupancy data

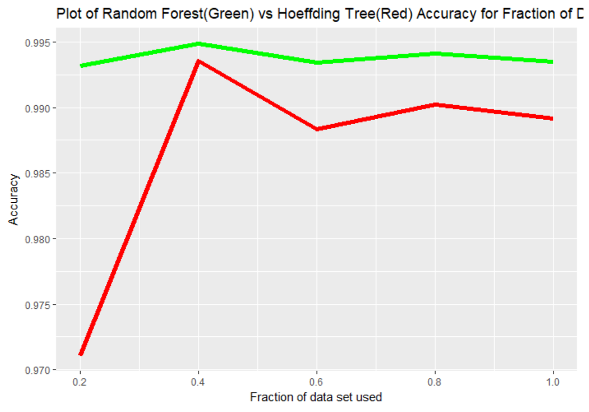


Figure 32: Time results for random forest and Hoeffding trees for given percentages of occupancy data

1.5.4 Discussion

Overall the results for random forest vs Hoeffding trees are promising. From 7 and Figure 31 it is evident that as we scale up the amount of data, the time increases in an exponential manner for the random forest, vs a linear increase for the Hoeffding tree. Also, the accuracy of the two is relatively close for 40% of the data used to the whole set. The average difference is roughly 0.5%. From here, we can conclude that the

Hoeffding tree has learned the proper splits without taking the time that the random forest required.

It makes sense that Hoeffding trees are used for online methods. In today's world organizations or individuals may need to deal with an immense amount of streaming data. By using the Hoeffding tree technique, these organizations can make predictions and classifications about chunks of new data without having to retrain the whole forest as you would in random forest. Recently there was a paper released that is a modification of the Very Fast Decision Tree, which tweaks the current Hoeffding model and is more accurate. It is called Extremely Fast Decision Tree (17), and while it is more accurate there is a slight increase in computational power needed to run the algorithm. It is faster than standard random forest.

2 Question 2: TCGA Mislabeled Data

The TCGA gene expression data mentioned in class, is a very large set of data containing 2887 instances with around 20,530 attributes (59.27 million values in the matrix). There are a total of six classes in the set of data, and they do not have an even distribution. See Table 9. In order to work with the data in a reasonable amount of time, I must employ a

Class	Count
BC	1215
GBM	172
KI	606
LU	571
OV	266
U	57

Table 9: Counts for the classes in the TCGA data.

type of reduction method that we discussed in class. The data set is simply too large for my computer or even a more powerful computer to apply expensive standard methods. I chose to use the **irlba** package (18) as demonstrated in class, because it performs a very fast singular value decomposition (SVD) for dense, large and sparse matrices. In my case, the TCGA data is a dense and large matrix so running this package helps me significantly reduce the size and still retain some of the important signals in the data.

2.0.1 Part A.

I first ran **irlba** to figure out how many of the principle components contribute significantly to the data. The resulting graph can be seen in Figure 33. From this plot I decided to use the elbow method to determine where I can take the cutoff principal component (PC). I chose the PC at 25, because the data forms an elbow here, and the rest of the PCs slowly taper off from this point. With this metric in hand, I am able to then run several different classification algorithms on the reduced SVD data. I decided to investigate linear discriminant analysis (LDA), random forest (RF), shrunken centroids (or PAM), and Naive Bayes (NB).

For this part, the goal was to investigate how the methods work for mislabeled data. For the structured data, I replaced the BC classes with GBM. I did this for varying percentages of the BC classes, starting with 0% replaced and increasing by 10% each time. Figure 34 displays the accuracy measurements for this method and Figure 35 displays the kappa measurements. It is interesting to note that the metrics both decrease to around 60%. This is because the BC classes make up roughly 40% of the data, so we should see this much error if the methods cannot figure out how the two classes can be similar in any way.

For the randomly mislabelled data I once again did it in increments of 10% of the total data. The results obtained are what we expect. The accuracy is displayed in Figure 36 and the kappa is displayed in Figure 37. In both of these the decrease in accuracy is fairly linear. For accuracy all four methods appear to behave similarly towards the end, but in kappa the LDA and RF do worse when the whole set is mislabelled.

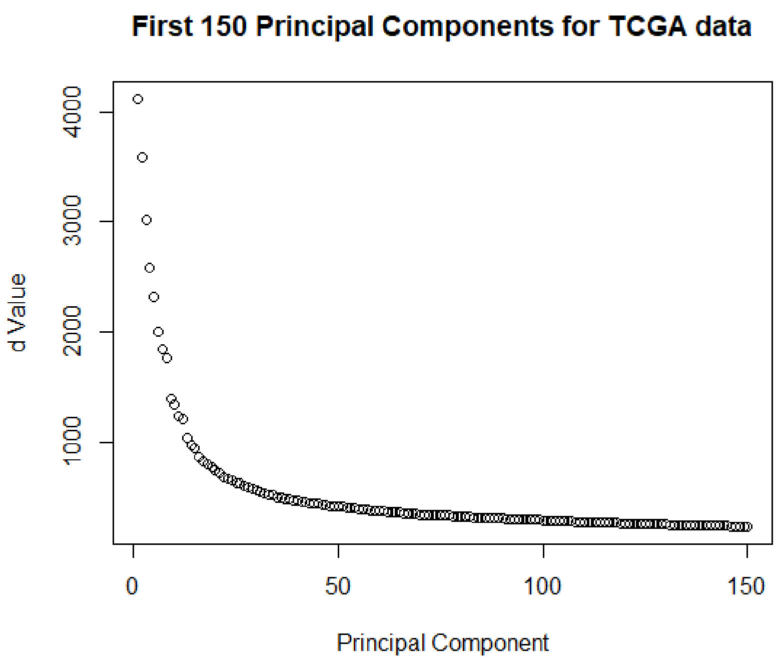


Figure 33: First 150 Principal Components of TCGA

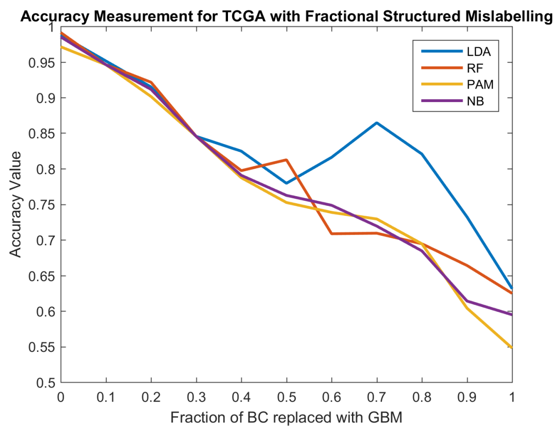


Figure 34: Accuracy plot for the methods for the structured mislabeled data. BC classes are replaced with GBM until GBM is no longer present as a class.

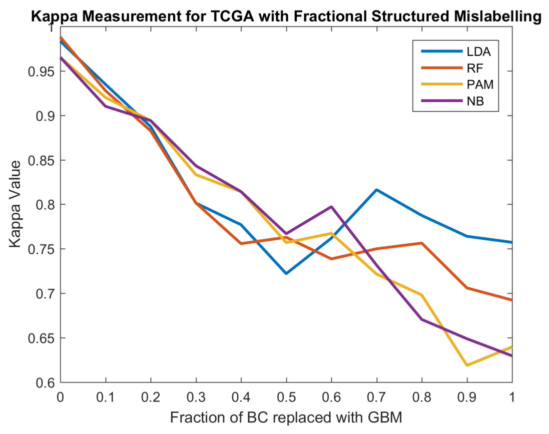
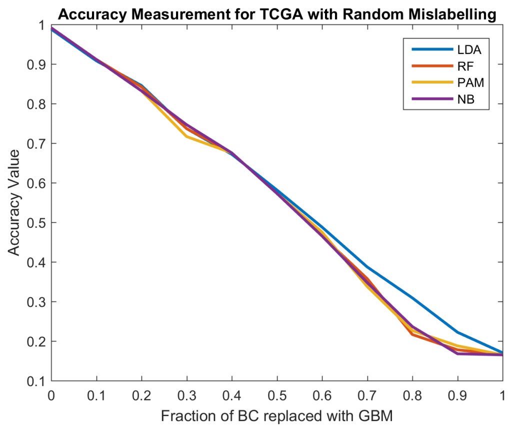


Figure 35: Kappa plot for the methods for the structured mislabeled data. BC classes are replaced with GBM until GBM is no longer present as a class.

Overall, mislabelling can have a dramatic impact on the accuracy and kappa metrics for these models. It is then important to make sure that the data one uses is as accurate as possible. If it is not, and it is difficult to obtain desirable classification results we can try a clustering method proposed for part b.



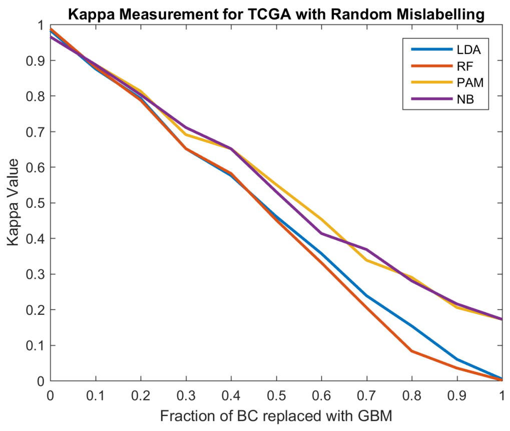


Figure 36: Accuracy plot for the methods for the randomly mislabeled data.

Figure 37: Kappa plot for the methods for the randomly mislabeled data.

2.0.2 Part B.

For this part, I defined a moderate degree of mislabeling as 40% mislabelled. After reading the article about noise filters for mislabelled data (19), I decided to install the **NoiseFiltersR** package (20). This package contains different types of noise identification and noise handling functions that are designed to fix mislabelled data. To test it out, I made the mislabelled data set and ran the same methods that I ran in Part A. After this, I first ran the **edgeBoostFilter** from the **NoiseFiltersR** package. This gave poor results as seen in Table 10. There was hardly any increase in performance

	Mislabelled Accuracy	Edge Boost Accuracy	Mislabelled Kappa	Edge Boost Kappa
PCA	0.6927083	0.7034991	0.6005908	0.6136307
RF	0.6892361	0.6942910	0.5960761	0.6023312
PAM	0.6650890	0.6840199	0.6440972	0.6703499
NB	0.6470962	0.7056192	0.6354090	0.6617781

Table 10: Accuracy and Kappa results for the 40% mislabelled data and edge boost filtered data.

with this filter. So then after consulting the paper (19), I ran the **hybridRepairFilter** from the **NoiseFiltersR** package and then took the data and ran it through the original methods. My results are in the following table, Table 11: It is apparent from Table 11

	Mislabelled Accuracy	Hybrid Repair Accuracy	Mislabelled Kappa	Hybrid Repair Kappa
PCA	0.6927083	0.9866667	0.6005908	0.9816493
RF	0.6892361	0.9765661	0.5960255	0.9926440
PAM	0.6561728	0.9459867	0.6440972	0.9813333
NB	0.6356089	0.9946667	0.6202064	0.9798632

Table 11: Accuracy and Kappa results for the 40% mislabelled data and hybrid repaired data.

that the hybrid repair filter worked extremely well. We see 30% or more increases in the metrics when the hybrid filter is applied. I believe that this function can be very useful for future work, especially with data that is often mislabelled because it is an ensemble method for repairing mislabelled data.

3 Question 3: Investigating when $K > L$ and when $L > K$

This question asks me to investigate how some methods perform when I vary the ratio of K clusters and L classes. To address the two parts A and B, I decided that I would compare the results to each other over a varying range of K to L ratios. The K and

K Clusters	L Classes
2	20
4	18
5	15
8	12
12	8
15	5
18	4
20	2

Table 12: Combinations of K and L values.

L values I chose are in Table 12. I avoided the ratio of the two being equal to 50% in order to just explore the behaviors when $K > L$ or $K < L$. To create my data I used Python's **sklearn** library (21), and used the **makeclassification** function to generate my data. I made the data with a low class separation parameter so that it is slightly noisy with overlapping classes. Overall, the data had the following features for each pair noted in Table 12:

- Sample size of 3000,
- K cluster, L classes
- 23 features: 9 informative, 5 redundant, and 9 noisy and random,
- class separation of 0.25.

I use the same methods as in the 2nd question to classify/cluster the data. Once again, they are:

- Linear Discriminant Analysis (LDA),
- Random Forest (RF),
- Shrunken Centroids/Partition Around Medoids (PAM),
- Naive Bayes (NB).

I chose these methods because they are standard approaches that are all slightly different and they are easily callable within R.

After running the four methods on each K and L pair, I found that the RF and NB methods took a significant amount of time when $K > L$. The plot results for accuracy are in Figure 38 and the kappa results are in 39. According to the results, accuracy is

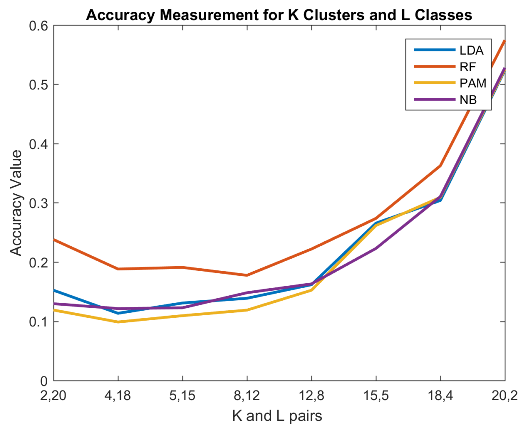


Figure 38: Accuracy plot for the methods for K clusters and L classes.

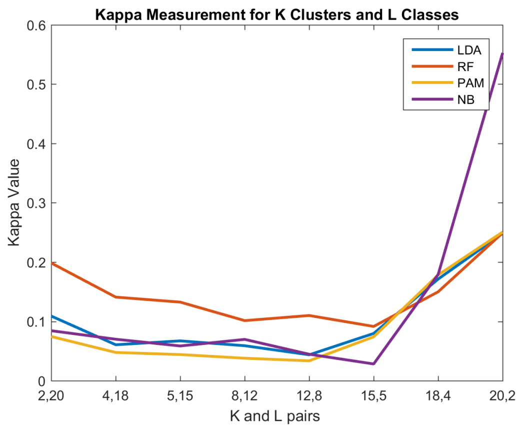


Figure 39: Kappa plot for the methods for K clusters and L classes.

low and kappa is very low when $K < L$. For this data set, when L becomes greater than K, the accuracy and kappa both increase. These results are not surprising given the structure of the data because the class separation is very low so there is a great deal of overlap. In addition, when there are very few clusters and many classes it is difficult for some statistical learning methods to be able to classify the clusters appropriately. In this case, some type of artificial neural network may be more accurate at detecting the classes. When we increase the number of clusters compared to classes, the performance of accuracy and kappa both rise. I would have expected this number to rise toward 80 or 90% for both cases. I think that the data set hindered the methods' abilities to accurately classify the instances because of the amount of noise in the data. There were 9 noisy variables, and these could have contributed negatively to the performance of the models. In a future experiment, feature selection may be important to reduce noisy variables in the data.

4 Summary

In summary, I thought that this course was a nice introduction to statistical learning techniques. I definitely think that I would have benefited from taking the regression course previously, but I was able to use some of my physics knowledge for following some of the techniques. In all, my largest takeaways are that it is very important to explore my data first, and try to figure out what I want to get out of it. In every single case we saw that results are dependent upon what you want to fins. This is a blessing and a curse because of how tricky it can be sometimes to get an unbiased result. I plan to build upon my knowledge of these techniques, and figuring out ways to create ensembles for the data I work with in the future.

References

[1] N. Gunduz and E. Fokoue, "Uci machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/turkiye+student+evaluation

[2] A. Kassambara and F. Mundt, *factoextra: Extract and Visualize the Results of Multivariate Data Analyses*, 2017, r package version 1.0.5. [Online]. Available: https://CRAN.R-project.org/package=factoextra

[3] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs, "NbClust: An R package for determining the relevant number of clusters in a data set," *Journal of Statistical Software*, vol. 61, no. 6, pp. 1–36, 2014. [Online]. Available: http://www.jstatsoft.org/v61/i06/

[4] C. Hennig, *fpc: Flexible Procedures for Clustering*, 2018, r package version 2.1-11. [Online]. Available: https://CRAN.R-project.org/package=fpc

[5] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2018. [Online]. Available: https://www.R-project.org/

[6] R. Lyon, "Htru2," Mar 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/HTRU2

[7] R. Bhatt and A. Dhall, "Skin segmentation," 2009. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/skin+segmentation

[8] M. K. C. from Jed Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, and T. Hunt., *caret: Classification and Regression Training*, 2018, r package version 6.0-79. [Online]. Available: https://CRAN.R-project.org/package=caret

[9] Z. A. Deane-Mayer and J. E. Knowles, *caretEnsemble: Ensembles of Caret Models*, 2016, r package version 2.0.0. [Online]. Available: https://CRAN.R-project.org/package=caretEnsemble

[10] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *In Proceedings of the 21st International Conference on Machine Learning*. ACM Press, 2004, pp. 137–144.

[11] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones, "mlr: Machine learning in r," *Journal of Machine Learning Research*, vol. 17, no. 170, pp. 1–5, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-066.html

[12] P. Romanski and L. Kotthoff, *FSelector: Selecting Attributes*, 2016, r package version 0.21. [Online]. Available: https://CRAN.R-project.org/package=FSelector

[13] G. J. Williams, *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*, ser. Use R! Springer, 2011. [Online]. Available: http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896

[14] L. Candanedo and V. Feldheim, "Occupancy detection data set," 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[15] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: https://CRAN.R-project.org/doc/Rnews/

[16] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 71–80. [Online]. Available: http://doi.acm.org/10.1145/347090.347107

[17] C. Manapragada, G. I. Webb, and M. Salehi, "Extremely fast decision tree," *CoRR*, vol. abs/1802.08780, 2018.

[18] J. Baglama, L. Reichel, and B. W. Lewis, *irlba: Fast Truncated Singular Value Decomposition and Principal Components Analysis for Large Dense and Sparse Matrices*, 2018, r package version 2.3.2. [Online]. Available: https://CRAN.R-project.org/package=irlba

[19] P. Morales, J. Luengo, L. P. Garcia, A. C. Lorena, A. C. de Carvalho, and F. Herrera, "The NoiseFiltersR Package: Label Noise Preprocessing in R," *The R Journal*, 2017, accepted, may change after copy-editing. [Online]. Available: https://journal.r-project.org/archive/2017/RJ-2017-027/index.html

[20] P. Morales, J. Luengo, L. P. Garcia, A. C. Lorena, A. C. de Carvalho, and F. Herrera, *NoiseFiltersR: Label Noise Filters for Data Preprocessing in Classification*, 2016, r package version 0.1.0. [Online]. Available: https://CRAN.R-project.org/package=NoiseFiltersR

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.