Predictive Classification Analysis with PCA, K-means, KNN, Random Forest and SVM.

Author: Deep Patel

Introduction

Obesity is a major disorder that involves excessive body fat which causes many serious health problems. This problem has become very common in various countries around the world, especially in Central America. This disorder usually has multiple causes. Although epidemiological analysis could be done for diseases with one cause, it takes much more time to do epidemiological analysis on diseases with multiple causes. Obesity is usually classified more accurately by a combination of dietary habits and physical activity habits. This dataset includes data of dietary and physical habits as well as few genetic factors. The use of automatic classifiers helps to detect patterns and effects of various features that ultimately leads to obesity. The use of classifiers using this dataset can also provide effective nutritional guidelines and potential effective treatments for obesity. This could help improve prediction of risk of obesity due to physical and dietary habits.

About Dataset

In the original selected dataset, the classes are divided into 7 total classes as shown in Figure 1. Class 1 includes the patients with underweight category. Class 2 includes patients with normal weight category. Class 3 and Class 4 includes patients with overweight I and overweight category II. Class 5, Class 6 and Class 7 includes patients with Obesity Level I, Obesity Level II, and Obesity Level III. The comparison of these classes with each other will help determine the key differences of dietary and physical patterns among different classes. In the original dataset, there are a total of 2,111 cases with each class having approximately 300 cases each.

In order to avoid different classes with similar characteristics and to make this analysis more interesting with appropriate sizes of the classes, the approach of regrouping and cloning classes was used. The classes of underweight and normal weight category are regrouped and named “Non-Obese” which resulted in total of 559 cases. The cases in Overweight I class and Overweight 2 category were combined through regrouping and the class was named “Overweight” which resulted in 580 cases. The cases in obesity level I, II and II were combined by a regrouping approach which resulted in 972 cases. This regrouped classes were assigned new class name “Obese” (CL3).

	Original Dataset	**After Data Warehousing**
Number of cases	2,111	3,250
Number of classes	7	3
Number of features	16	14 (29 after replacing features with binary features)
Size of each class	~300 in each class	559, 580, 972
Class Names	Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II, Obesity Type III	Non-Obese, Overweight, Obese

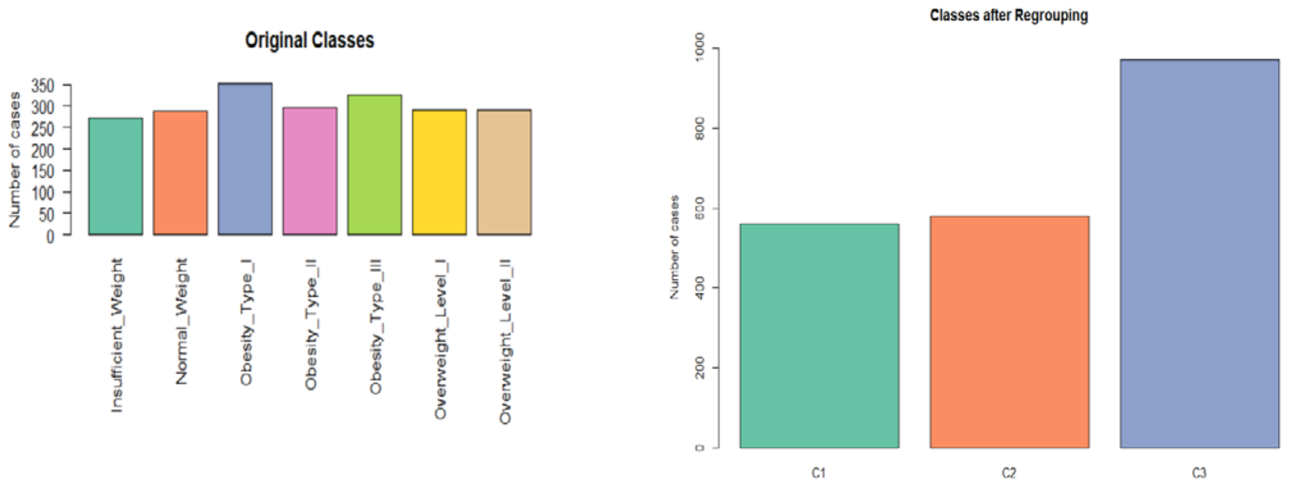


Figure 1. Distribution of classes before and after regrouping. *The left side of the figure shows the distribution of classes in the original dataset and on the right, the figure shows the distribution after regrouping similar classes where C1 is Non-Obese, C2 is Overweight and C3 is Obese.*

The features include data recorded from dietary habits, physical activity and genetic factors. Genetic factors include gender, age and family history. Physical activity features include Calories consumption monitoring (SCC), Physical activity frequency (FAF), Time using technology devices (TUE), and Transportation used (MTRANS). The features of dietary habits include data of Frequent consumption of high caloric food

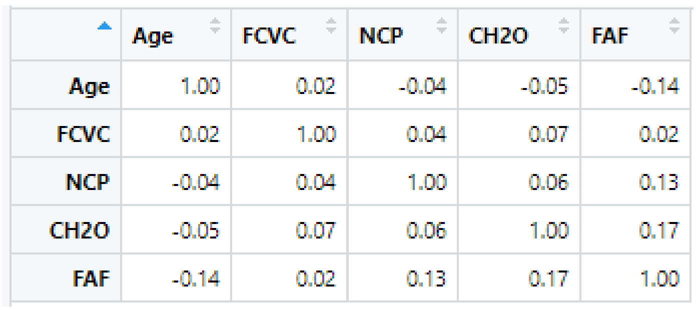
(FAVC), Frequency of consumption of vegetables (FCVC), Number of main meals (NCP), Consumption of food between meals (CAEC), Consumption of water daily (CH2O), Smoke, and Consumption of alcohol (CALC). Out of 14, six features are numerical and eight are categorical which were turned into numerical by using a binary system. Numerical features include Age, FCVC, NCP, CH2O, FAF and TUE. The categorical features Gender, Family history with overweight, FAVC, Smoke, SCC were replaced by two binary features each. The categorical features CAEC and CALC were replaced by four binary features each. Lastly, the categorical feature MTRANS were replaced by five binary features, resulting in a total of 29 features. As there are too many binary features, to avoid the issue of too many ties, small random perturbation was applied to all features.

PCA analysis with Standardized Dataset

All of the features in the dataset were standardized to avoid bias and to bring all values of the variables in common scale without distorting differences in the range of values. Standardization is important, as later in PCA and classification analysis the values of the features need to be in the same scale or else some features might be falsely weighted too high or too low when calculating distances.

PCA is used primarily for data compression by reducing the original dimension of the dataset. This transforms the original features onto a new feature subspace with a lower dimension. The main goal of Principal Component Analysis (PCA) algorithm is to improve classification analysis by reducing the number of features. The selected new features are used for data visualization. If time permits, the dataset with new features can also be used for comparative analysis with full dataset towards supervised learning techniques such as KNN, Random Forest classifier or K-means unsupervised learning.

Before implementing PCA, the correlation matrix was obtained for the full standardized dataset. The correlation matrix shows the relationship between each feature variable. The partial view of correlation matrix is shown below:

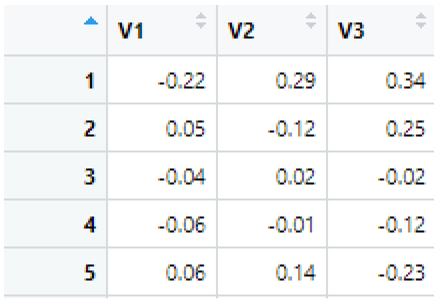


	Age	FCVC	NCP	CH2O	FAF
Age	1.00	0.02	-0.04	-0.05	-0.14
FCVC	0.02	1.00	0.04	0.07	0.02
NCP	-0.04	0.04	1.00	0.06	0.13
CH2O	-0.05	0.07	0.06	1.00	0.17
FAF	-0.14	0.02	0.13	0.17	1.00

Figure 2. Partial View of correlation matrix.

One of the ways Principal Component Analysis can be computed is by using eigenvalues and eigenvectors. Both, eigenvalues and eigenvectors, exist in pairs, meaning each eigenvector has a corresponding eigenvalue. In a p x p symmetric matrix A with scalar λ and nonzero vector V such that A*V=λ *V , λ is an eigenvalue and V is an eigenvector associated with an eigenvalue. An eigenvector is a vector whose direction remains unchanged when a linear transformation is applied to it. An eigenvalue tells us how much variance there is in the data in the direction of the eigenvector.

The eigenvalues and eigenvectors of the correlation matrix were obtained using the eigen() function in computational software R. The partial view of eigenvectors are shown in Figure 3. The eigenvalues are later displayed with Percentage of variance explained (PVE).



	V1	V2	V3
1	-0.22	0.29	0.34
2	0.05	-0.12	0.25
3	-0.04	0.02	-0.02
4	-0.06	-0.01	-0.12
5	0.06	0.14	-0.23

Figure 3. Partial View of Eigenvectors W

The eigenvalues, along with the percentage of variance explained is shown. The figure 4 shows the plot of the eigenvalues, Lr, vs r as a decreasing function where r corresponds to principal component number of the respective eigenvalue. By analyzing the plot, it is clear that the first values of r have the highest eigenvalues. The eigenvalues can also tell us about the distances of the projected data points on a one-dimensional line from the origin. The higher the sum of those distances, the greater the eigenvalue and the first principal components associated with those highest eigenvalues account for most of the variance in the data. It can also be observed that the values are decreasing and approaches zero around r=21. This is a sign that the eigenvalues which are almost zero or pretty much zero tell us that the features relating to those eigenvalues are related by a linear relation. The extremely low eigenvalues, also suggests that the data projected onto a one dimensional line through the origin have a low sum of distances of the projected data points on a one-dimensional line to the origin.

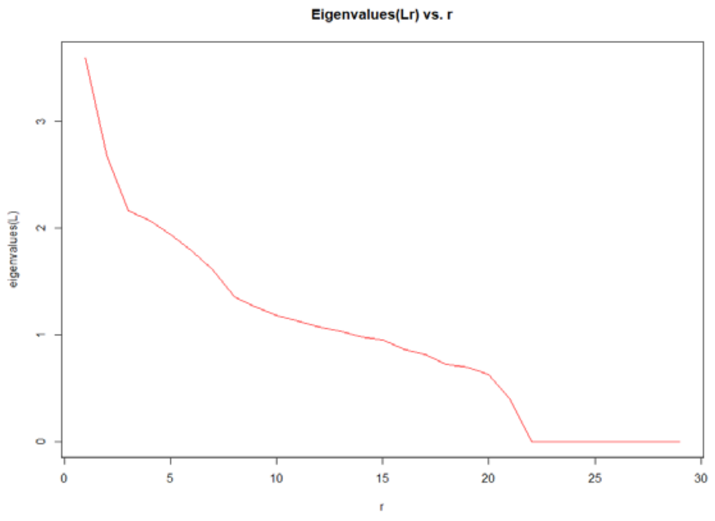
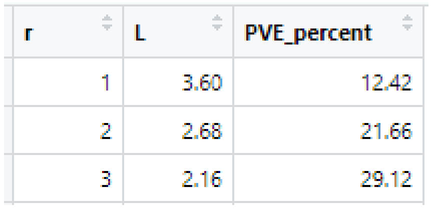


Figure 4. Eigenvalues(L) vs. r



r	L	PVE_percent
1	3.60	12.42
2	2.68	21.66
3	2.16	29.12

Figure 5. First 3 eigenvalues and percentage of variance explained

Furthermore, the percentage of variance was obtained from the eigenvalues. The first three eigenvalues(L) with percentage of variance explained is shown in Figure 5. Additionally, the plot of the Percentage of Variance Explained (PVE) against the r eigenvectors is shown in Figure 6. From the table and plot, we can see the percentage of variance for the first 3 features out of 29 features accounts for 30% variance. Also, notice the plot follows the inverted pattern of figure 4 as PVE is an increasing function with respect to r. To get the optimal value of eigenvectors and principal components, the PVE of 95 percent was selected which reduced the total number of features to 19. This indicates that from our original 29 features, a projected subspace of 19 new features accounts for 95 percent of the variability in our data. With an unavoidable loss of data with PCA, only five percent of the variability in our data is unexplained with r as 19. The new 19 features can keep the strong patterns in our dataset and simplify the required structure to represent the original data.

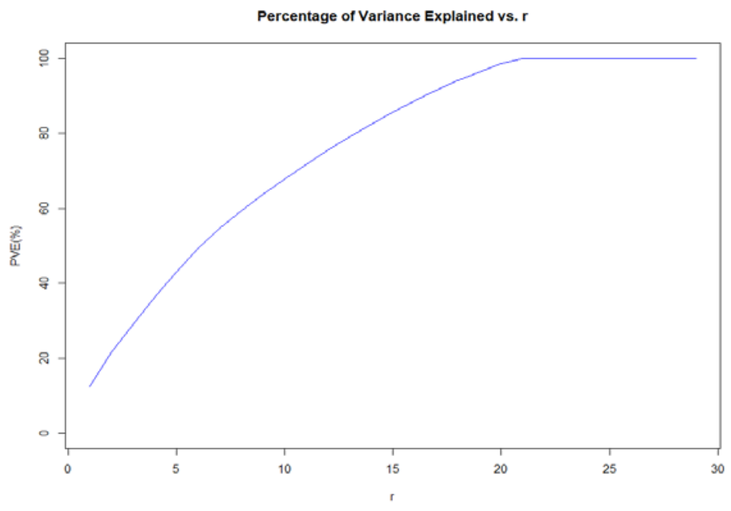
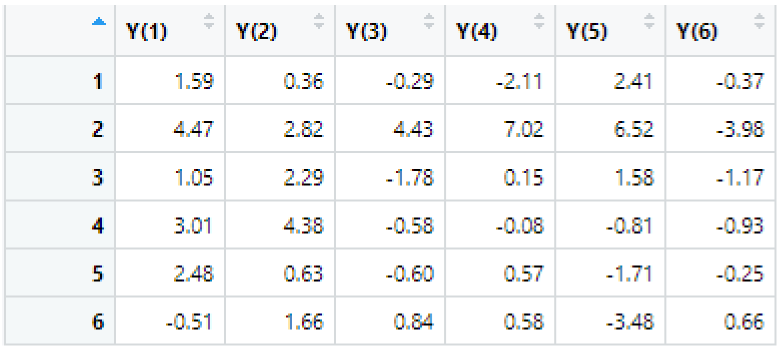


Figure 6. Plot of PVE(%) vs r eigenvectors

The matrix with new features was computed through transpose of the eigenvectors matrix W and the original standardized features as shown below.

$$\boldsymbol{Y(n)\ =\ W^{T} * \ X(n)}$$



	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	Y(6)
1	1.59	0.36	-0.29	-2.11	2.41	-0.37
2	4.47	2.82	4.43	7.02	6.52	-3.98
3	1.05	2.29	-1.78	0.15	1.58	-1.17
4	3.01	4.38	-0.58	-0.08	-0.81	-0.93
5	2.48	0.63	-0.60	0.57	-1.71	-0.25
6	-0.51	1.66	0.84	0.58	-3.48	0.66

Figure 7. Partial view of matrix with new features

Then, 19 new features were selected which are also known as principal components. Since PCA allows the visualization of high dimensional data, 3D scatter plots of the data separated in pairs of classes is plotted for the first three principal components.

According to PCA, PC1 and PC2 are the features which accounts for approximately 12% and 10% of the data whereas PC3 accounts for 8% variation of the data. This can be observed in the plots as PC1 and PC2 have slightly wider range than PC3.

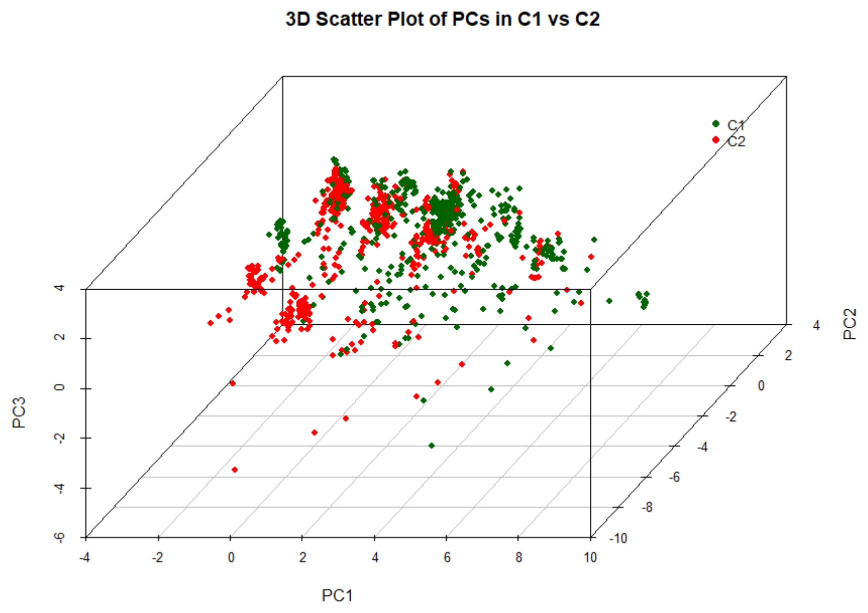


Figure 8. 3D Scatter Plot of first three principal components in C1 vs. C2

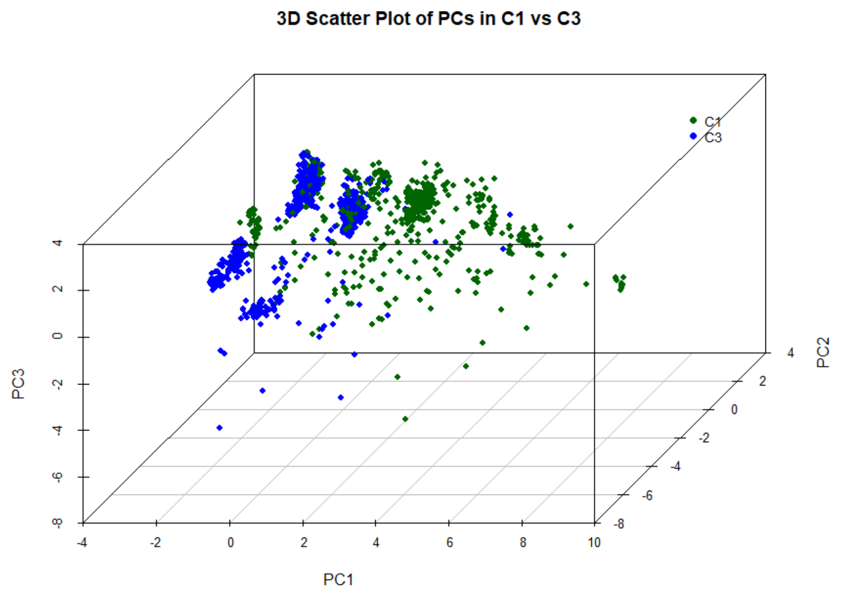


Figure 9. 3D Scatter Plot of first three principal components in C1 vs. C3

In terms of separability between pairs of the classes, 3D plots of C1 vs. C3 and C2 vs. C3 shows better separation than C1 vs. C2. This is likely due to the fact that C1 (Non-Obese) and C2 (Overweight) have approximately the same size of observations and may share few features that has similar contributing factor as opposed to C3 (Obese). However, the variation through separation can be clearly observed in each 3D plot. The plot with least overlap is C1 vs. C3 which makes sense because C1 represents Non-obese and C3 reperesents Obese. The features contributing to both of these classes should be significantly different as both classes are totally different.

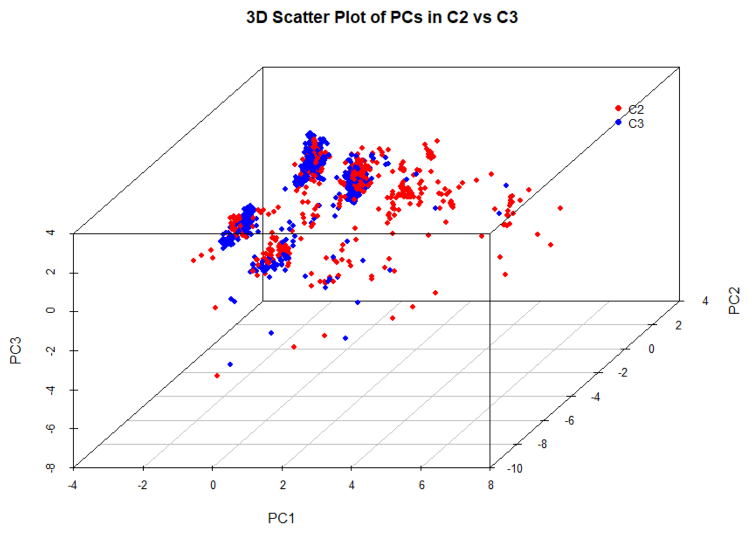


Figure 10. 3D Scatter Plot of first three principal components in C2 vs. C3

The 3D plots between each pair of classes helped to view the separability better because as shown in Figure 11, three classes on the same plot shows a lot of overlap which may falsely leave the impression that the variation between classes have insignificant variation in the first three principal components.

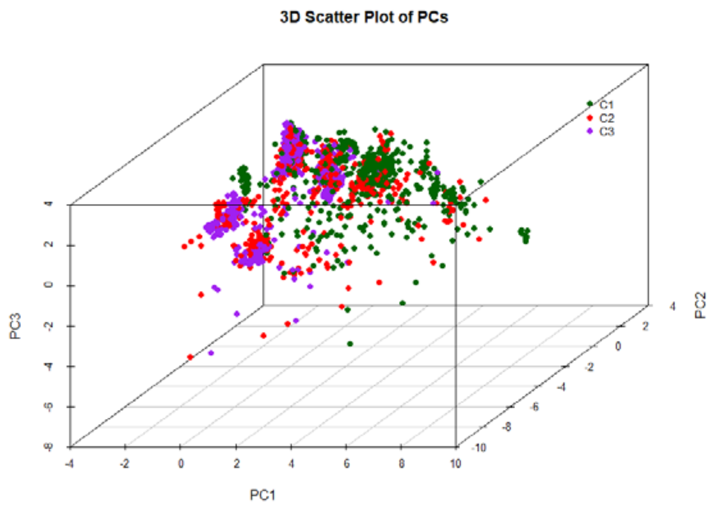


Figure 11. 3D Scatter Plot of first three principal components in C2 vs. C3

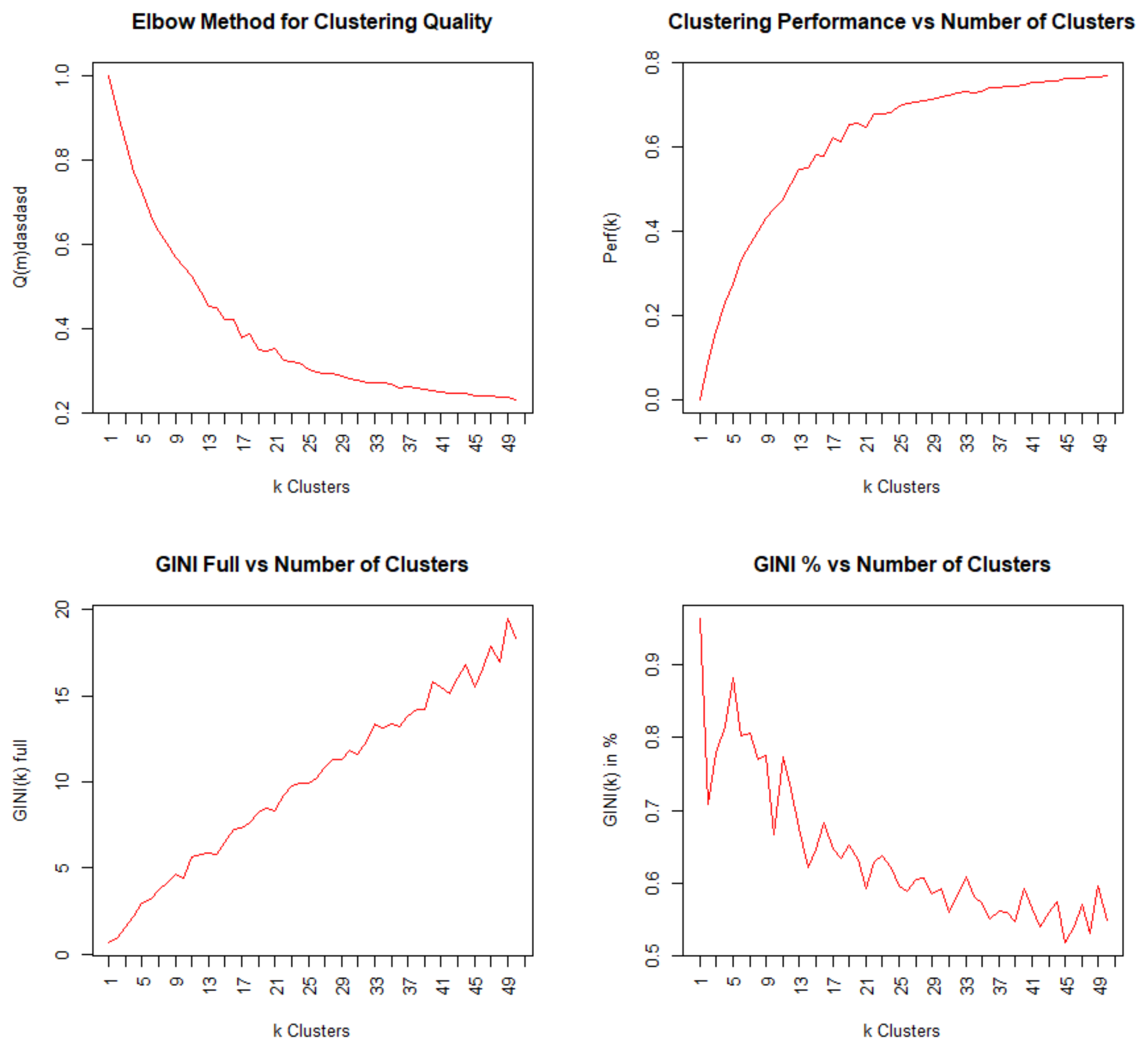
K-means Clustering Analysis

To see if this dataset works well with unsupervised model, Kmeans was used. As opposed to KNN, the K-means is an unsupervised learning algorithm. K-means clustering works by classifying a set of data into k distinct groups. K-means doesn't actually know the true number of classifications, but instead tries to find the true number by assuming different values of k and calculating the sum of the total variations of each cluster. Cluster variations are calculated by summing the Euclidean distances between cluster centers and each observation with the cluster. As k increases, the total dispersion decreases until k reaches n, where n is the number of observations in the data set, and the dispersion equals zero. K-best is then chosen by determining the value of k where the decreases in total dispersion per unit increase in k becomes marginal.

When K-means is run each observation is randomly assigned to a cluster. Then the algorithm goes through x iterations of a 2-step process. In the first step the cluster centers are determined by finding the point in n-dimensional space where the dispersions of the clusters are smallest. In the second step, each case is reassigned to the cluster to which it is closest (based on the Euclidean distance between the case and cluster center). With enough iterations the cluster centers and case assignments eventually stop changing.

The K-means algorithm was applied to the standardized data set with k=1 through 100. After repeated launchings, it was determined there is an insignificant rate of change after k=50. Therefore, only k=1 to 50 were plotted to determine the optimal k clusters. For each value of k the reduction of variance, perf(k), was calculated where:

$$\boldsymbol{perf(k) = 1 - \frac{total\ within\ cluster\ sum\ of\ squares}{total\ sum\ of\ squares}}$$



’***Figure 12. The number of k clusters plotted against Clustering Quality, Clustering performance, GINI Full Clustering Impurity and percentage of GINI impurity.***

Applying the elbow rule, k-best appears to be around 19. Choosing the elbow of the curve as shown in the top left plot in Figure 12 was a somewhat subjective task, so k-best was chosen based on the rate of change of Perf(k) falling below 0.02 after k = 19 as opposed to higher change of rate before k = 19. Notice that the highest performance with k = 50 clusters was around 80% and if the number of clusters k had kept increasing, the performance would be getting higher as a result of the decreasing total within cluster sum of squares. The other factors such as GINI Indexes, GINI Impurity of full clustering, and GINI percentage of Impurity were also taken into account. The bottom left plot shows the impurity of full clustering for each k-value from k=1 to 50. For 3 classes, the maximum impurity can be 1-(⅓)= ⅔ for each cluster. This plot on the bottom left follows a similar trend showing poor separation. Therefore, to better view how the number of k-clusters affects the GINI Impurity, the GINI Impurity of full clustering for each k-value was divided by maximum impurity to obtain a percentage of GINI impurity and then it was plotted against k-clusters which is shown on bottomleft plot of GINI impurity. The lower the GINI impurity %, the higher k-value. However, after evaluating and comparing all four plots, the optimal clusters seems to be k=19 as it falls on the vague elbow rule and flatter or minimal variance pattern can be observed after k=19 in clustering performance perfk plot and GINI Impurity percentage plot.

Since there is no trend separation in GINI full impurity plot as explained above, the rate of decreasing variance was taken into account when determining optimal k-value for k-means clustering along with GINI full Impurity percentage. In order to be sure to select the best k clusters based on elbow method, the 2 k-values above and below chosen optimal k-value (k*-/+1 and k*-/+2) were also tested. For each k=17,18,19,20, and 21, the k-means was launched 30 times separately. Each time the k-means was launched, the gini indices for each run was obtained and compared. Upon checking gini indices across different values of k, the trend was approximately similar, with k=17 having the lowest gini followed by k=19. Other k-values had gini indices of 0 which indicates maximum purity. However, upon checking the size of clusters, it was evident that clusters with gini of 0 contained only one case. Choosing this cluster for further classification analysis would be not ideal. Therefore, k=17 was chosen as the best k* it showed no unusual results and it had the cluster with lowest gini indices with appropriate class size distribution.

Using k = 17, the center of a cluster or centroid can be calculated by taking the average of all vectors over all cases belonging to the specific cluster. So, there would be 17 centroids for 17 clusters. The number of coordinates of one centroid is 29, as it is equal to the number of standardized features we used. The matrix generated to store the data, is a kxN matrix where N is the 29 coordinates. However, to visually plot the centroids, the matrix dimension must be reduced. To implement dimensionality reduction, the first three eigenvectors from the 29x29 correlation matrix were used to obtain the orthogonal projection of the centroids in 3 dimensional space. As a result, the kx3 matrix was obtained where k=17 clusters and 3 was the first three eigenvectors. Plotting these values allows us to visually see the distances between the centers shown in the following figure.

-

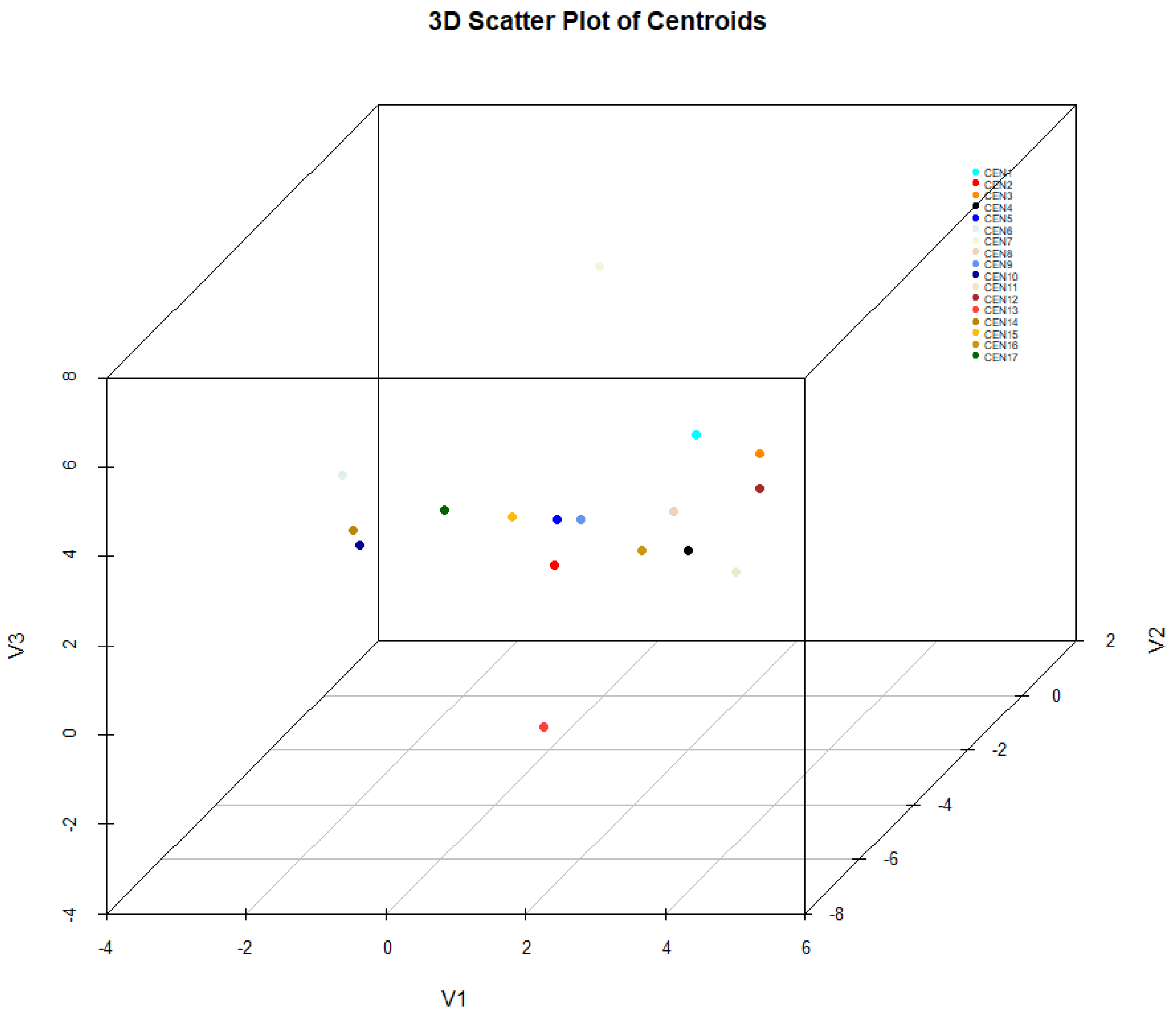
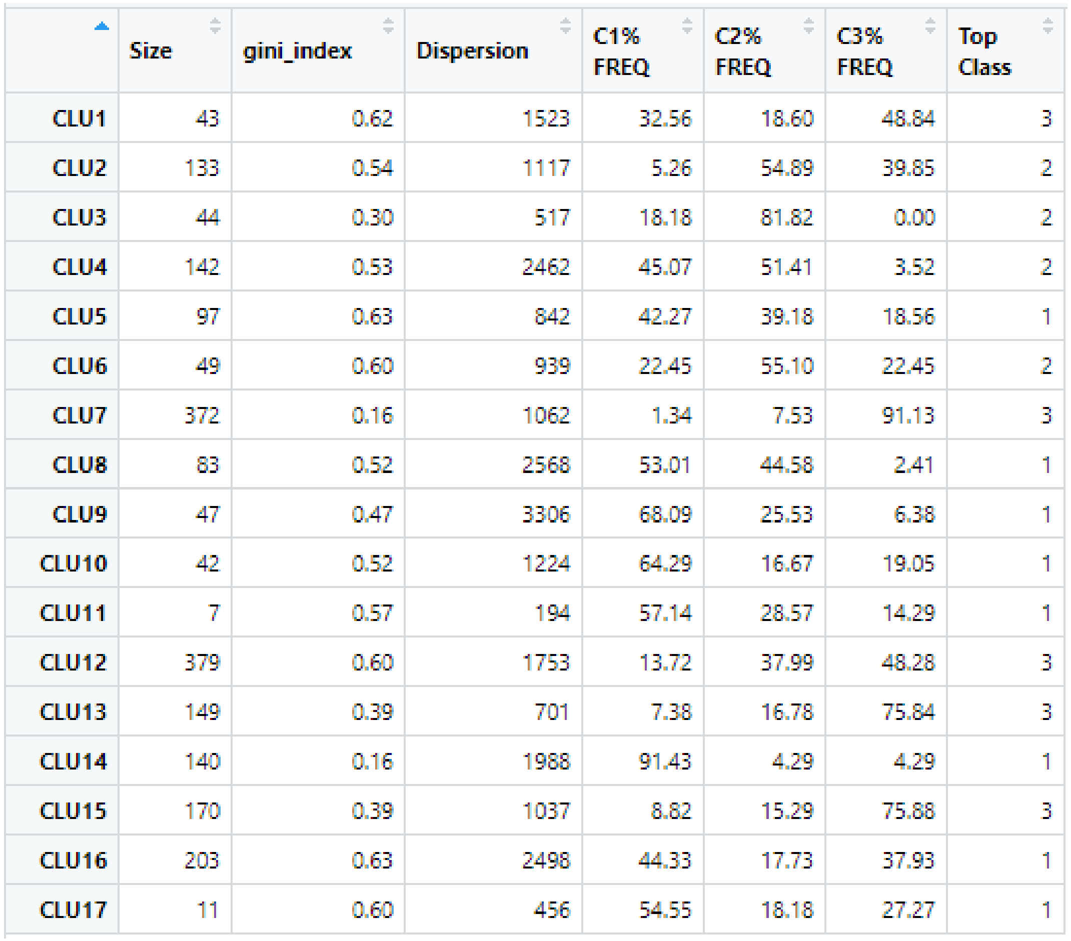


Figure 13. 3D Scatter Plots of Centers of Clusters.

With 3D Visualization, notice we can see the relative distances between the cluster centroids. Noticeably, centroid 2,6,10,and 16 seems to be further than the rest. In the table below, the relationship between the distances and cluster similarity is observed. In general, it is reasonable to suggest that a larger distance indicates a greater difference between the clusters. All of these four clusters have greater impurity ranging from 0.52 to 0.63. Notice, for Cluster 6 and 10, the cluster sizes are also extremely low.



	Size	gini_index	Dispersion	C1% FREQ	C2% FREQ	C3% FREQ	Top Class
CLU1	43	0.62	1523	32.56	18.60	48.84	3
CLU2	133	0.54	1117	5.26	54.89	39.85	2
CLU3	44	0.30	517	18.18	81.82	0.00	2
CLU4	142	0.53	2462	45.07	51.41	3.52	2
CLU5	97	0.63	842	42.27	39.18	18.56	1
CLU6	49	0.60	939	22.45	55.10	22.45	2
CLU7	372	0.16	1062	1.34	7.53	91.13	3
CLU8	83	0.52	2568	53.01	44.58	2.41	1
CLU9	47	0.47	3306	68.09	25.53	6.38	1
CLU10	42	0.52	1224	64.29	16.67	19.05	1
CLU11	7	0.57	194	57.14	28.57	14.29	1
CLU12	379	0.60	1753	13.72	37.99	48.28	3
CLU13	149	0.39	701	7.38	16.78	75.84	3
CLU14	140	0.16	1988	91.43	4.29	4.29	1
CLU15	170	0.39	1037	8.82	15.29	75.88	3
CLU16	203	0.63	2498	44.33	17.73	37.93	1
CLU17	11	0.60	456	54.55	18.18	27.27	1

Figure 14. Optimal k clusters size, dispersion, Gini Index, Probability Distribution, and Top Class

The largest cluster was cluster 12 with 379 cases but with a GINI index of 0.60 indicating high impurity. GINI indices of all clusters are below maximum possible impurity of 0.67 for three classes. A high purity of cluster is attained when the gini index is closest to zero. Purity is a measure of the extent to which clusters contain a single class. The equation to calculate the gini index is given below.

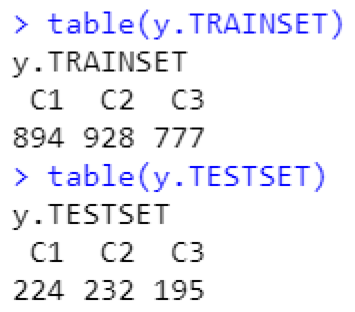
$$GINI\ index\ = p1 * (1 - p1)\ +\ p2 * (1 - p2)\ +\ p3 * (1 - p3)$$

(p_1,p_2,p_3) are the probability distribution (percentages) of C1,C2,C3 cases within a single cluster.

The cluster with lowest impurity was cluster 14 followed by cluster 7 which had nearly the same lowest gini indices. The cluster 14 had size of 140 whereas the cluster 7 had a size of 372. Both contain a majority class with 91% of the size. In cluster 14, class 1 was majority whereas in cluster 7, class 3 was majority. Both of these clusters will be later explored with the Random Forest classifier later in the report. The majority of the clusters among 17 clusters had top class 1 followed by class 3. There were only 4 clusters with top class 2. The full clustering impurity was 8.23 is considerably less than the maximum full clustering impurity for 17 clusters can be 11.39.

Training Set and Testing Set

The full dataset was split into train and test set using the classical 80-20 split approach. As the classes were not balanced, balancing was needed as it necessary to obtain reliable results using Random Forest classifier. Therefore, after train-test split, C1 and C2 were cloned to approximately balance the size of classes in train and test set respectively. The size distribution of each class within Trainset and Testset are shown below:


```
> table(y.TRAINSET)
y.TRAINSET
 C1  C2  C3 
894 928 777 
> table(y.TESTSET)
y.TESTSET
 C1  C2  C3 
224 232 195 
```

The k-nearest neighbors (KNN) is a supervised machine learning model which relies on labeled input data to learn a function that produces an appropriate output when given a new unlabeled data. KNN is used for classification and regression analysis but often more in classification problems. The principle of KNN assumes every data point falling in near to each other is falling in the same class which means similar things are near to each other. For example, consider Figure 8, suppose there are N training vectors and classes are ‘red circles’ and ‘yellow triangles’ in the model, KNN algorithm identifies k nearest neighbors of ‘blue smiley face’ regardless of labels. If k=4, it will identify 4 nearest neighbors of c using distance, Euclidean distance or proximity and choose the class with majority. In this case, when k=4, the k-nearest neighbors identifies 1 red circle and 3 yellow triangles in proximity. Based on the majority, it chooses the ‘yellow triangles’ as for a blue smiley face. The knn algorithm is often widely used as it is easy to understand and gives high accuracy which doesn’t necessarily mean it is a good algorithm as there are other good algorithms in supervised learning.

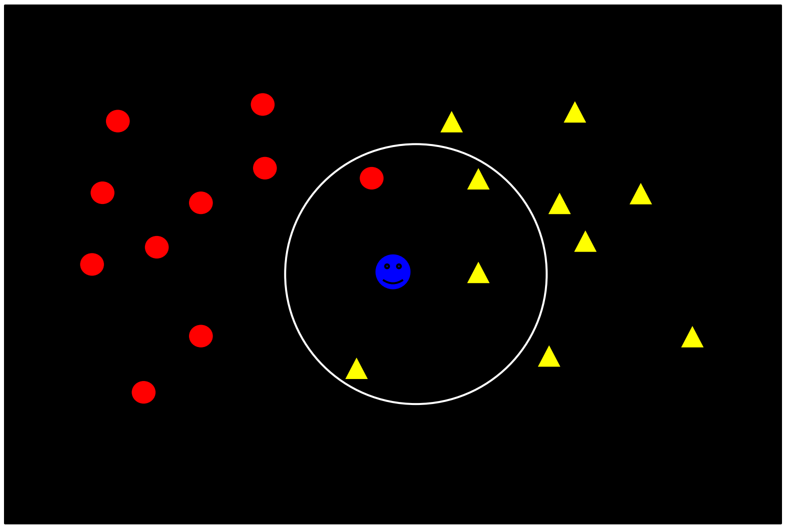
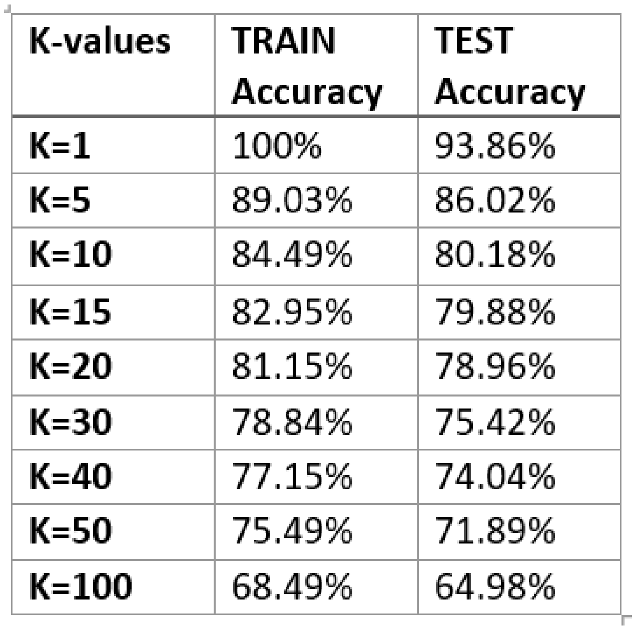


Figure 15. Simple example of KNN algorithm

The objective of using KNN method on obesity dataset is to evaluate the efficacy of KNN method to correctly classify the dietary and physical activities patterns of different classes of weight categories. Using the standardized dataset, the class sets labeled CL1 (Non-Obese), CL2 (Overweight) & CL3 (Obese) were each divided into two sets; the first being training sets with a random selection of 80% of the rows of CL1, CL2 & CL3, and the second being a testing set with the remaining 20% of rows of CL1, CL2 & CL3. The TRAIN SET consists of 2599 cases in total whereas the TESTSET consists of total 651 cases

The KNN method was applied to a range of k values starting from k=1 to k=100 in order to visually discern any trend between k value and testing/training accuracies. The accuracies of the some selected values are shown in Table 5. The accuracy plot of all 100 values is also shown in Figure 9.



K-values	TRAIN Accuracy	TEST Accuracy
K=1	100%	93.86%
K=5	89.03%	86.02%
K=10	84.49%	80.18%
K=15	82.95%	79.88%
K=20	81.15%	78.96%
K=30	78.84%	75.42%
K=40	77.15%	74.04%
K=50	75.49%	71.89%
K=100	68.49%	64.98%

Figure 16. Training accuracy and Test accuracy at different k-values

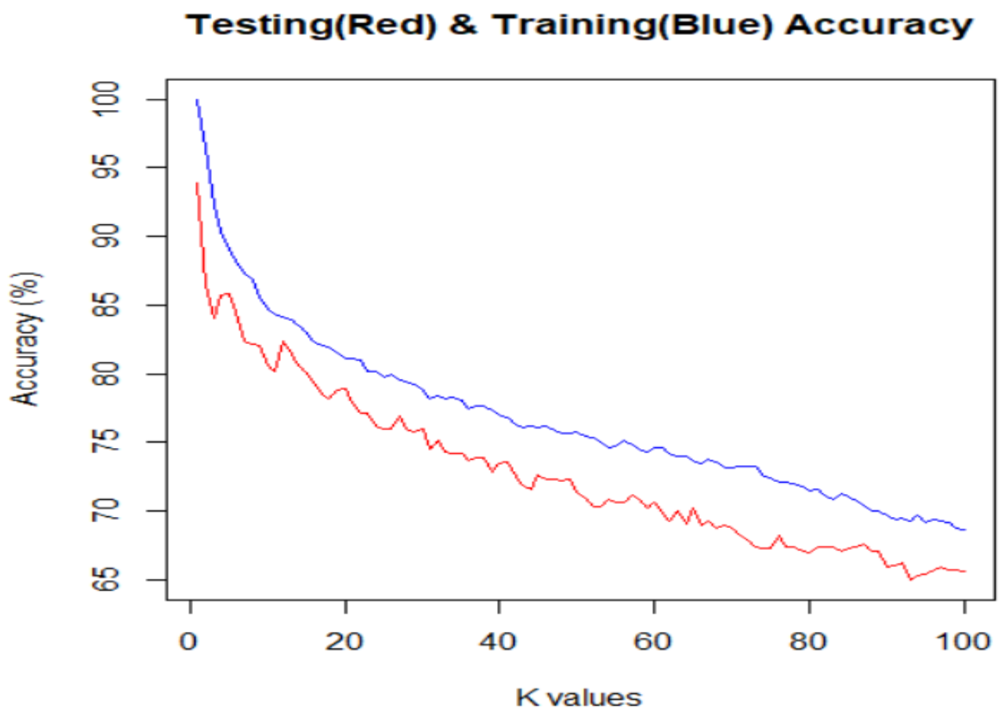


Figure 17: Accuracy curve of Training and Test set for k-values of 1 to 100

From the plot it appears that the best k values most likely lies between k= 1 & k= 30. The graph has a clear negative trend with increasing k so greater accuracies with larger k-values is not expected. Therefore, for a better zoomed view, another accuracy plot was generated for the k-values with the range of k=1 and k=30. Based on the plot, the best k-value for the testing set appears to be k=1.

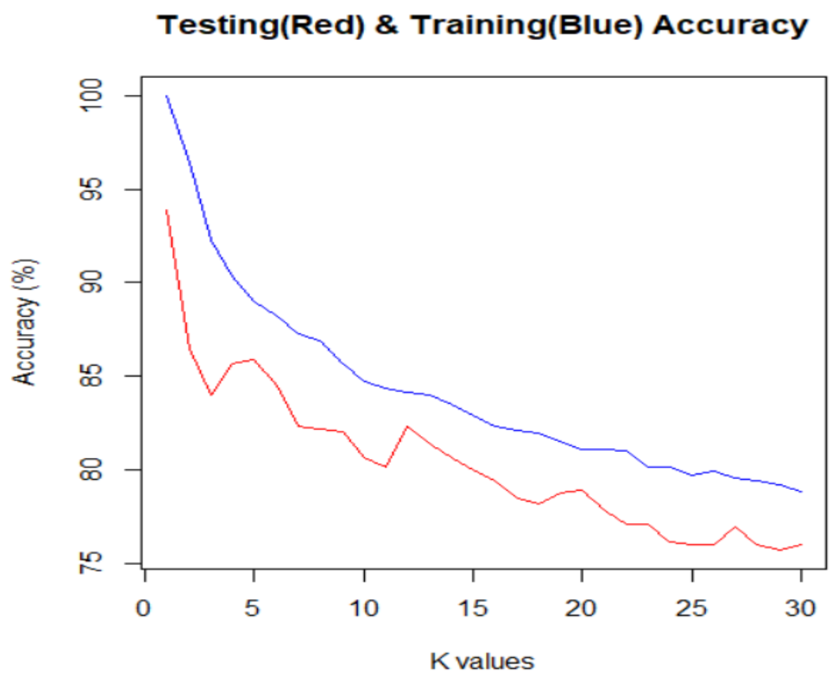


Figure 18: Accuracy curve of testing and training dataset for k-values of 1 to 30.

The accuracy values for training appear to be consistently about 5% higher than those for testing. As there is no overlap and there is discrepancy between training and testing data, this model may be overfitted which can be determined with the computation of confidence intervals. This can happen when the statistical model has less number of cases in the testing set. There are only 651 cases in the testing dataset. The error rate plots were also plotted against k-values as shown in Figure 11 to observe the differences between the error rates of training data and testing data. The error rate of testing data is higher than the error rate of the training data. It is evident that as accuracy decreases, the error rate increases.

Using kbest=1, the confusion matrix for test set (testconf) and train set (trainconf) was created as shown in Figure 5 on the next page.

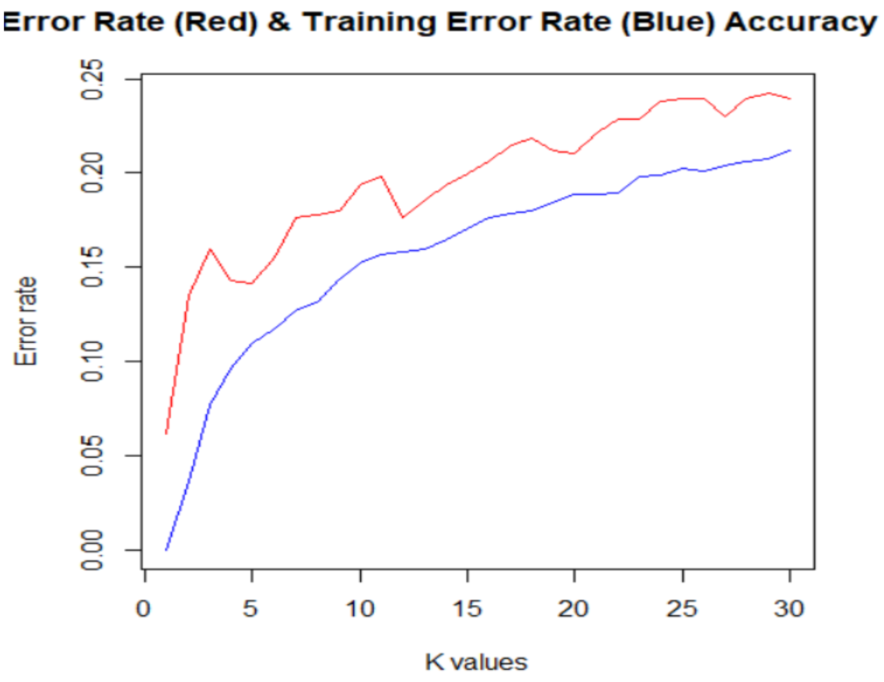


Figure 19: Error rate of Training (Blue) and Testing curve (Red).

a. Training confusion matrix (in percentage) **b. Testing confusion matrix (in percentage)**

```
                   True (Train)
Prediction (Train) CL1 CL2 CL3
               CL1 100   0   0
               CL2   0 100   0
               CL3   0   0 100
```

```
                   True (Test)
Prediction (Test)    CL1   CL2   CL3
              CL1  95.50  2.50  2.12
              CL2   1.80 92.50  7.41
              CL3   3.60  1.67 93.65
```

Figure 20. Training and Testing set confusion using kbest=1.

In the confusion matrices, we can see the high number of classes or percentage across the diagonals. This is because it depicts the true positive number of classes or percentage. Notice training confusion shows 100% for all classes. This is due to overfitting at k=1. Comparing both trainconf and testconf, the accuracies in testconf differs by 5 to 8% for each class. The global accuracy for trainconf is 100% whereas the global accuracy for testconf is 93.58%.

The 95% confidence intervals for each diagonal term was computed and compared for training confusion matrix and testing confusion matrix as shown in Table 6 and 7 on the next page.

When computing 95% confidence intervals of each diagonal term in the ***trainconf*** matrix , the accuracy would remain at 100% for all classes and there would be no interval. When computing 95% confidence intervals of each diagonal term in the ***testconf*** matrix, the confidence interval for CL1 and CL2 was narrower than the confidence interval of CL3. Comparing the confidence intervals for each diagonal terms of ***trainconf*** and ***testconf*** matrix shows that there is no overlap in the confidence interval accuracies and it is disjoint. Thus, it can be concluded that, q1>q2, meaning training set performance is better than testing performance.

TRAINSET	Lower Limit	Upper Limit
CL1 TRAIN	100%	100%
CL2 TRAIN	100%	100%
CL3 TRAIN	100%	100%

Figure 21: 95% confidence interval for TRAINSET.

As the accuracy itself is 100 for k=1, there would be no confidence interval for the train set but 100% can still be compared with the confidence intervals of the test set.

TESTSET	Lower Limit	Upper Limit
CL1 TEST	91.57%	97.5%
CL2 TEST	93.01%	98.28%
CL3 TEST	85.91%	94.18%

Figure 22: TESTSET 95% confidence interval

Although there is not a big gap between TRAIN SET and TESTSET accuracies which directly suggests overfitting, to verify that kbest=1 and not the second best k-value which is k=2, confidence intervals for k=2 were compared with the confidence intervals of k=1 which had an accuracy of 86.72%. Similar to the confidence interval for k=1, the confidence intervals for k=2 were computed as shown in Table 8 and Table 9. Comparing TRAIN SET and TESTSET, it is evident that there is no overlap and there is disjointness, verifying that training performance is better than the test performance. In the TESTSET of k=2, the confidence intervals for all classes are wider than the TESTSET confidence interval of k=1. Thus, k=1 is the best k value for testing performance.

TRAINSET	Lower Limit	Upper Limit
CL1 TRAIN	95.45%	97.81%
CL2 TRAIN	92.36%	95.39%
CL3 TRAIN	96.80%	98.89%

Figure 23: 95% confidence Interval of k=2 for TRAIN SET.

TESTSET	Lower Limit	Upper Limit
CL1 TEST	83.81%	92.26%
CL2 TEST	76.50%	86.30%
CL3 TEST	81.21%	90.62%

Figure 24: 95% confidence interval of k=2 for TESTSET

Weighted KNN

One of the ways the accuracies of KNN test performance may be increased is by adding weight to the features. A unique weight can be calculated for each feature. However, a more efficient way would be to divide the features into 6 disjoint packs of features and find a weighted value for each group by applying KNN to each group individually. When assigning feature to each packs of the groups, discriminatory powers were taken into account to make sure features with weak discriminatory powers are not placed in same pack as it could affect the accuracy of the data.

After assigning features to six different packs of features, their accuracies for each weighted set was obtained using kbest=1. The results of each weights from each pack is shown below:

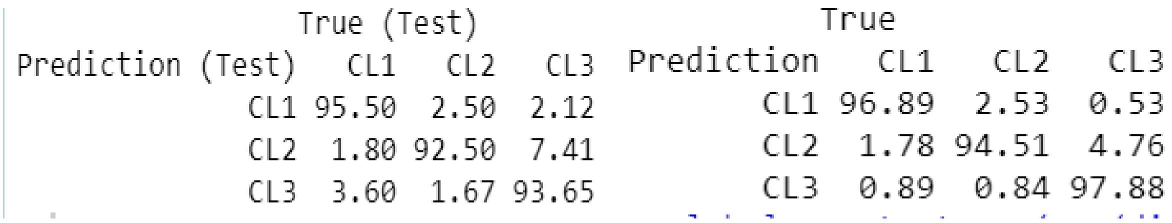
For w1=*testperfkbest* the accuracy was found to be 84.77%
For w2=*testperfkbest* the accuracy was found to be 77.11%
For w3=*testperfkbest* the accuracy was found to be 81.41%
For w4=*testperfkbest* the accuracy was found to be 83.87%
For w5=*testperfkbest* the accuracy was found to be 76.65%
For w6=*testperfkbest* the accuracy was found to be 78.64%

The particular weights accuracy applies to each feature in the pack of belonging weights. Considering the accuracies of all six w’s are close to each other, their weights may improve accuracy as irrelevant features or features with weak discriminatory power are given weight value close to zero which may result in improved accuracies.

The weights of each of the six packs of features were renormalized by the sum of their weights. Then w1 was assigned to all feature vectors in PACK2. Similarly, w2, w3, w4, w5 and w6 were assigned to the feature vectors in their respective PACKS. After all the weights were assigned, the packs were merged back to generate a dataset with weighted distance. The KNN method was then applied again with kbest=1 on the TESTSET.

The confusion matrix was generated for test data with weighted distance and compared with the confusion matrix generated earlier as shown in Figure 13. By looking at both confusion matrices, it appears that accuracies of each class with the testset improved with weighted distance. CL3 (Obese class) was the most changed whereas CL1(Non-Obese) and CL2(Overweight) change was minimal. Lastly, the global accuracy for test data with weighted distance of KNN was 96.3% and the global accuracy for test data un-weighted distance of KNN was 93.86%.

a. testconf matrix unweighted distance **b. Conftest.weight matrix in percentage**



```
                 True (Test)
Prediction (Test)   CL1   CL2   CL3
              CL1 95.50  2.50  2.12
              CL2  1.80 92.50  7.41
              CL3  3.60  1.67 93.65
```

```
             True
Prediction   CL1   CL2   CL3
       CL1 96.89  2.53  0.53
       CL2  1.78 94.51  4.76
       CL3  0.89  0.84 97.88
```

Figure 25. Confusion Matrices of weighted distance and ordinary distance. Part a shows the confusion matrix of ordinal distance obtained earlier in the analysis. Part b shows the confusion matrix of testset with applied weights.

Random Forest Classifier

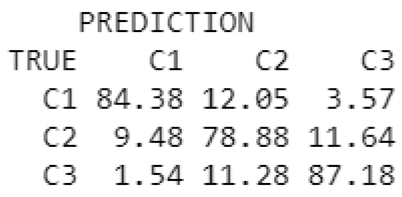
Random forest (RF) is a decision tree-based algorithm used for regression and classification problems. Decision trees (DT) work by continually segregating data into two groups at each node based on a single feature value at a time. Each new group is subsequently re-segregated until only groups of a single class remain. DTs are called such because each node at which data is split can be thought of as branches in a tree parting ways and the pure-class terminal groups can be thought of as the leaves.

RF is a unique DT method in that it builds many classifiers (“trees”) using this method. For each tree, RF first creates a “bootstrapped” data set from the supplied data set by randomly selecting one case at a time with replacement. Replacement means that the case is not removed from the selection pool after being selected and, hence, can be selected again. Any cases that are not included in the bootstrapped data set are called “out-of-bag.” Typically, each tree contains roughly two-thirds of the cases in the training set. Next, RF randomly chooses a number of features, mtry, where these features make the trees in the RF independent from each other. RF then determines which of those features is the best segregator of the data. Then the data is split using said feature with successive yes/no decisions. When splitting, if the feature is numerical, all data less than a certain value (threshold), V, are put into one group and all data greater than or equal to V are put into another group. For categorical features, the condition depends on the cases belonging to a particular value in that feature. As mentioned before, this happens continuously until all terminating groups are of a pure class. After a number, ntree, of trees are created each tree casts a set of classification votes for its out-of-bag cases. Each case ends up with a number of votes equal to the number of times it was out-of-bag and is then classified as the class which got the most votes. All of the decision trees together constitute the “random forest” and they are used collectively to predict classifications of new cases.

The randomForest function is accessible through R which is used for classifications. There are various arguments and parameters that can be used. The arguments of the randomForest function used in this analysis include data, mtry, ntree and importance. mtry is a number of variables randomly sampled as candidates at each split. For classifications problems this is usually the square-root of the number of data features. In our case, it will sqrt(29)= 5.38 which will be rounded to 5. ntree is the number of classifiers (aka “trees”) the RF function builds. Each tree votes for every case that is out-of-bag (not included in its bootstrapped data set). Importance is a boolean argument which tells the algorithm whether to assess the importance of the predictors. The predict function can be thought of as the second half of the randomForest function. It takes the test set and the classifier created by the randomForest function as arguments, then predicts the classes of each case in the test set. The resulting output can be used to compute confusion matrices and accuracies.

After dividing and balancing the dataset in TRAIN and TEST set , the random forest classifier was then applied to the train set with mtry set at square root of 29 which was rounded to 5 and the ntree was set at 100, 200, 300 and 400 for each different run of RF classifier. The parameter importance was also turned on, and will be looked into further detail later in the analysis. Since only the train set was provided to this classifier, the predict function was used to predict the classes for the test set. The confusion matrices along with its accuracies are shown below in figure 15 for each of those RFs. It is very apparent that the lower values of ntrees have lower global accuracy as expected. In general the more trees in a RF, the more robust the model will be and the better the predictions and accuracy. However, there is always a value of the number of trees in a RF where the global accuracy stabilizes, beyond which not much of a change in accuracy and prediction can be gained. As the accuracies seemed very stable for all those ntree 100, 200, 300 and 400, ntree 10 and ntree 50 was added to see when the change occurs in the accuracies. For RF with only 10 trees produced the lowest accuracy with 83.25%, while 400 trees produced the highest accuracy with 88.78%.

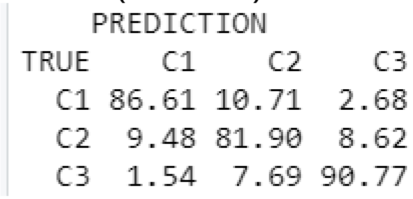
conf10 (ntree=10)



```
      PREDICTION
TRUE     C1    C2    C3
  C1 84.38 12.05  3.57
  C2  9.48 78.88 11.64
  C3  1.54 11.28 87.18
```

Global accuracy= 83.25%

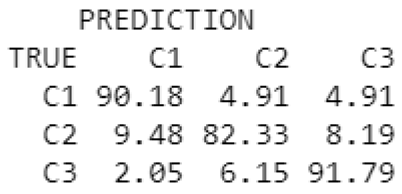
conf 50 (ntree=50)



```
      PREDICTION
TRUE     C1    C2    C3
  C1 86.61 10.71  2.68
  C2  9.48 81.90  8.62
  C3  1.54  7.69 90.77
```

Global accuracy= 86.17%

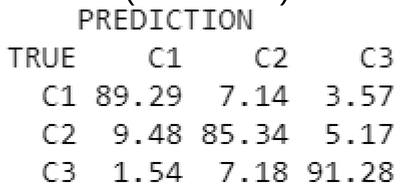
conf100 (ntree=100)



```
      PREDICTION
TRUE     C1    C2    C3
  C1 90.18  4.91  4.91
  C2  9.48 82.33  8.19
  C3  2.05  6.15 91.79
```

Global accuracy=87.86%

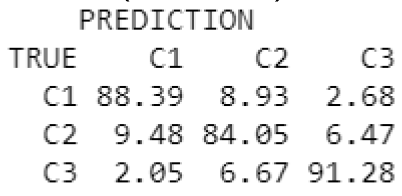
conf200 (ntree=200)



```
      PREDICTION
TRUE     C1    C2    C3
  C1 89.29  7.14  3.57
  C2  9.48 85.34  5.17
  C3  1.54  7.18 91.28
```

Global accuracy= 88.48%

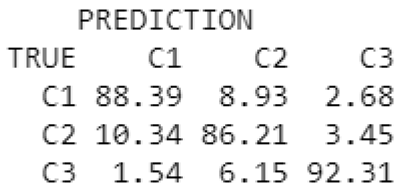
conf300 (ntree=300)



```
      PREDICTION
TRUE     C1    C2    C3
  C1 88.39  8.93  2.68
  C2  9.48 84.05  6.47
  C3  2.05  6.67 91.28
```

Global accuracy= 87.71%

conf400 (ntree=400)



```
      PREDICTION
TRUE     C1    C2    C3
  C1 88.39  8.93  2.68
  C2 10.34 86.21  3.45
  C3  1.54  6.15 92.31
```

Global accuracy= 88.78%

Figure 26. Confusion matrices and global accuracies for the testing set predicted by the random forest classifier with six different values of ntree with mtry staying at 10 for all. ntrees = {10,50,100,200,300,400}.

For visual representation and to ease the approach of selecting the best ntree, the global accuracies for train and test set were plotted against each ntree which is shown in Figure

16. In addition to that, another insightful plot is shown in figure 17 which shows a plot of all three individual class accuracies against the number of trees which shows some instability among class 1 and 2 which will also be explored later with more ntree to confirm there is no significant change in the accuracies of class 1 and class 2 with more number of ntrees.

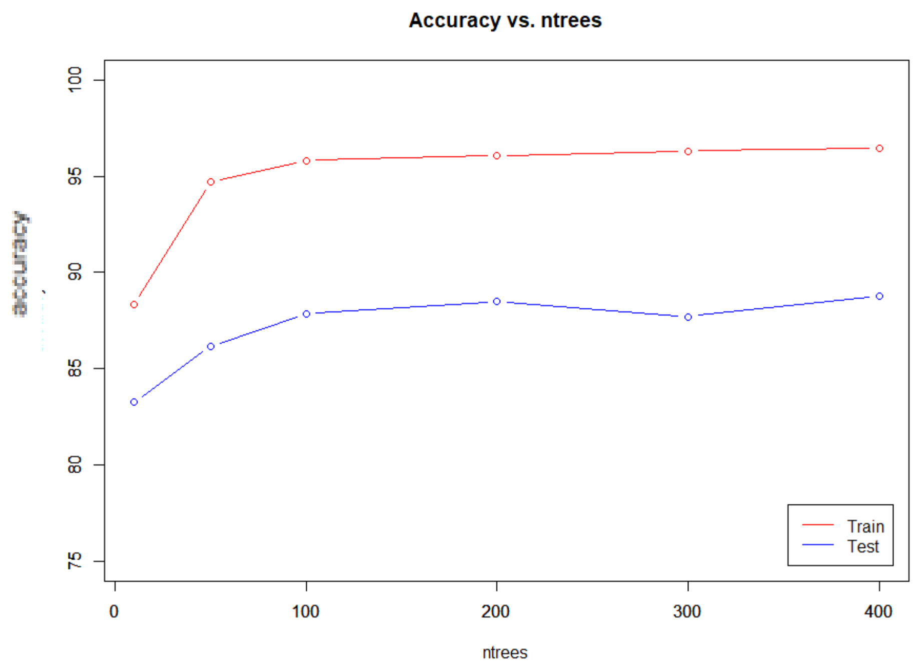


Figure 27. Global accuracy vs. ntrees. Accuracy appears to stabilize at ntree=200.

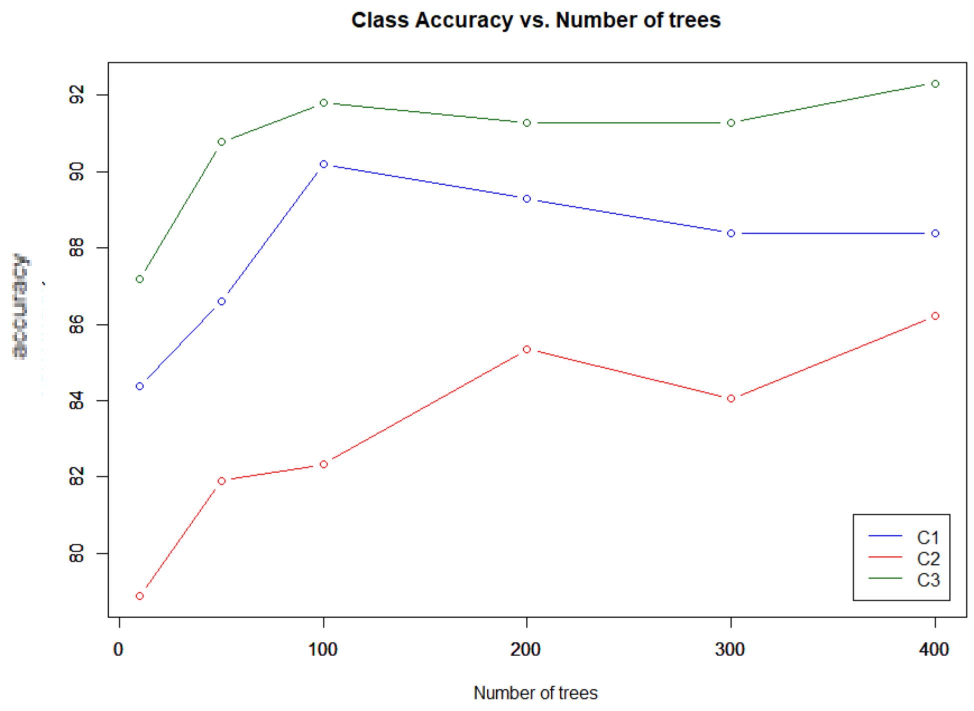
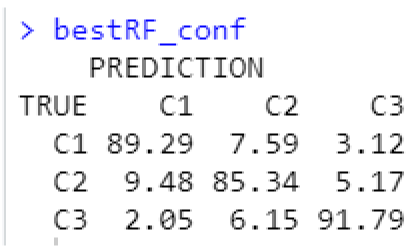


Figure 28. ***Class accuracy vs number of trees. Values extracted from the three diagonal coefficients of the confusion matrices.***

Based on the plots of accuracy vs. ntrees as shown in Figure 17, the best number of trees seems to be 400 as the curve seems to flatten out as the curve seems to be unstable for C1 and C2. C3 seems ntree=100 but showed slight uptick after ntree=300. The best number of trees 400 was selected as the accuracy curve seems to peak there for C1 and C3 and for C2 seems to flatten out after that and there is no significant improvement for C2 after ntree=400. Using ntree=400, a new RF classifier, bestRF will be implemented to explore feature importance. To check if the results are consistent if another train/test model is trained for RF classifier, another train/test set was created and the results were nearly similar showing no major differences, indicating the model was stable which could be also verified with confidence intervals. But since the model was run several times and the results were consistent, the need for it seemed redundant. If the accuracy seems to increase overall or for each class significantly with higher number of trees will be later explored in attempts of improvement.

Importance of features using best number of trees in Random Forest

The bestRF was re-run with a different train/test model to make sure the obtained results are consistent. The bestRF conf with another train/test model is shown below with global accuracy of 88.63% and if this is compared with the best ntree=400 confusion matrix in Figure 15, it is nearly similar.



```
> bestRF_conf
    PREDICTION
TRUE     C1    C2    C3
  C1 89.29  7.59  3.12
  C2  9.48 85.34  5.17
  C3  2.05  6.15 91.79
```

Figure 29. bestRF confusion matrices from another model

Setting importance equals True when running RF classifier, it gives the ability to retrieve the importances of features in a form of plot. The dotchart for mean decrease in accuracy and gini is shown for the all 29 features in Figure 19. The y axis shows the features with feature names. Nevertheless, in these dotcharts the importance value can be directly matched with its features. Both docharts are similar in terms of listing the importances. Age is classified as the most important feature whereas CALCAlways is classified as least important feature. It does not come to surprise that most important features are the numerical features whereas all binary features are of lesser importance. The Mean decrease accuracy shows the value of importance starting at 60 for feature Age then decreasing to around 50 for feature FCVC and at the end decreasing to nearly 0 for CALC Always feature.

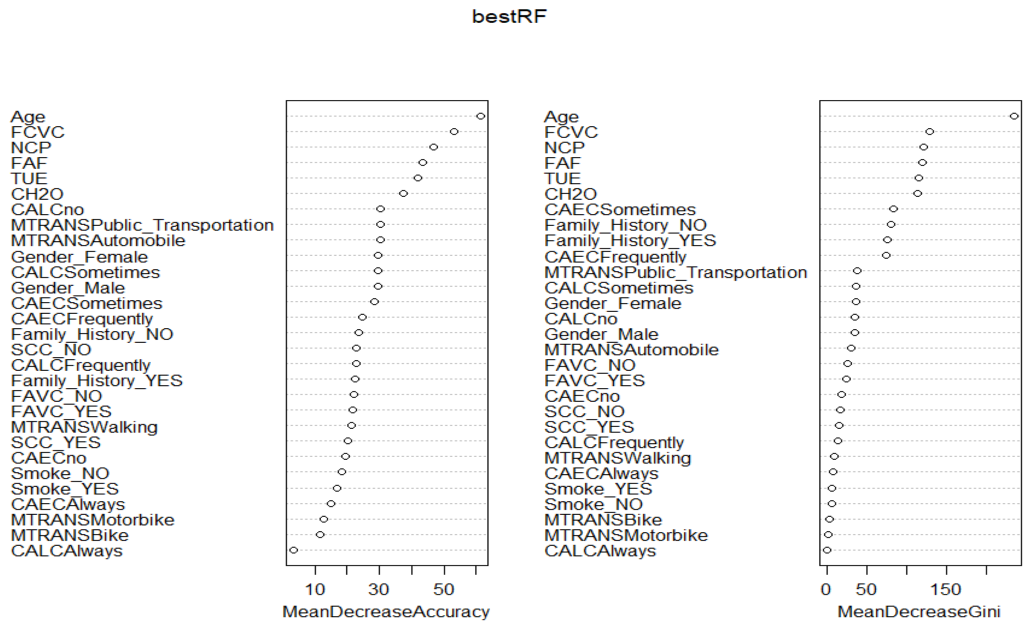


Figure 30. Feature importance, Variable Importance Plot. Shows the dotchart with MeanDecreaseAccuracy and MeanDecreaseGini as measured by the bestRF.

Moreover, to compare the distributions of the most important feature and least important feature, the histograms for those features were obtained and compared across different classes. The histogram below shows the most important feature Age. It is evident the distribution is quite different between different classes showing variance.

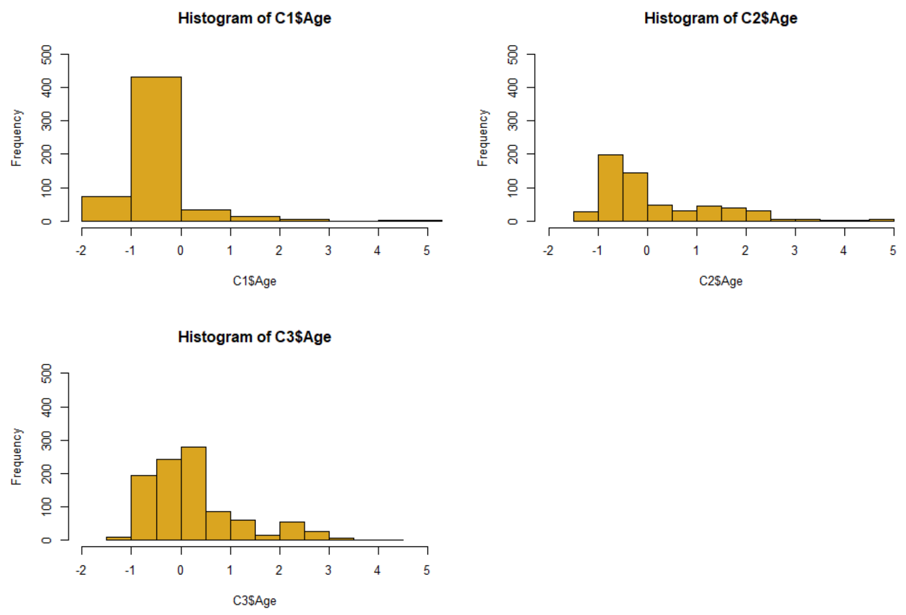


Figure 31. Histograms of most important feature- Age

On the other hand, the histograms of least important feature CALC Always shows almost no variance in distributions among all three classes. This is expected as it is a binary variable.

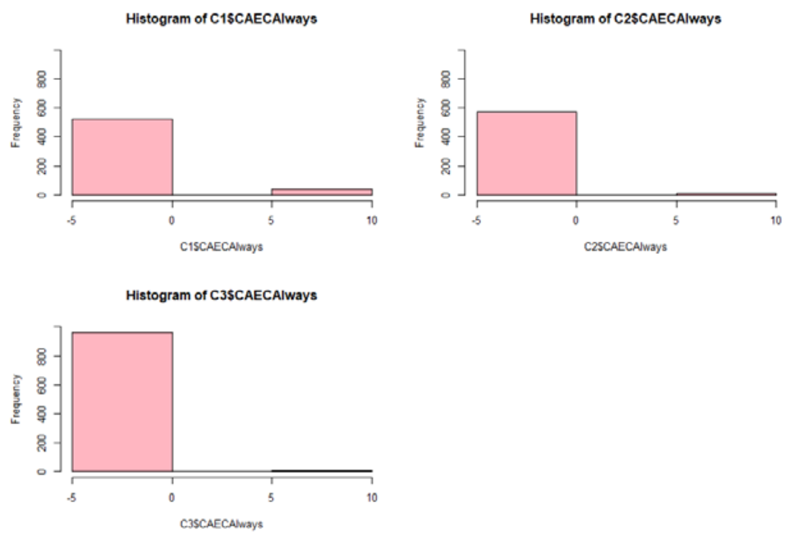
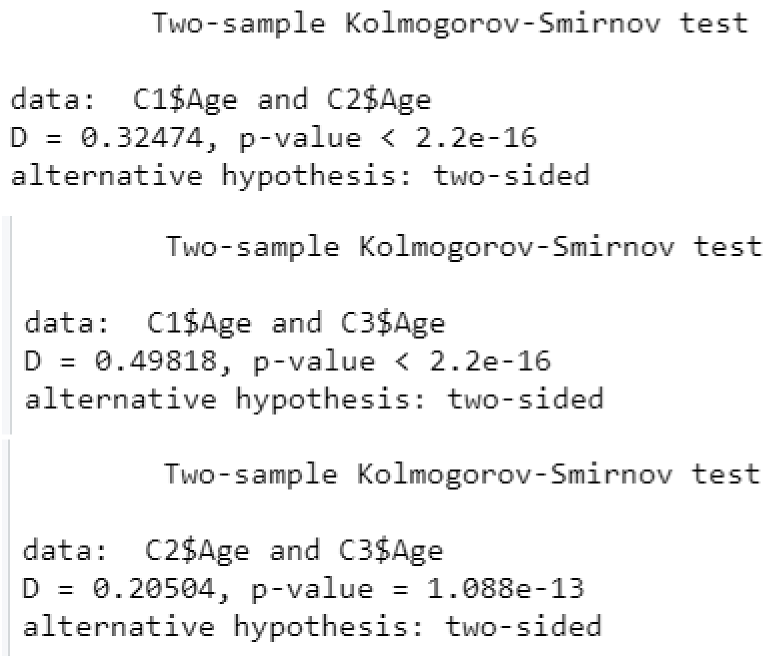


Figure 32. Histograms of least important feature- CALC Always

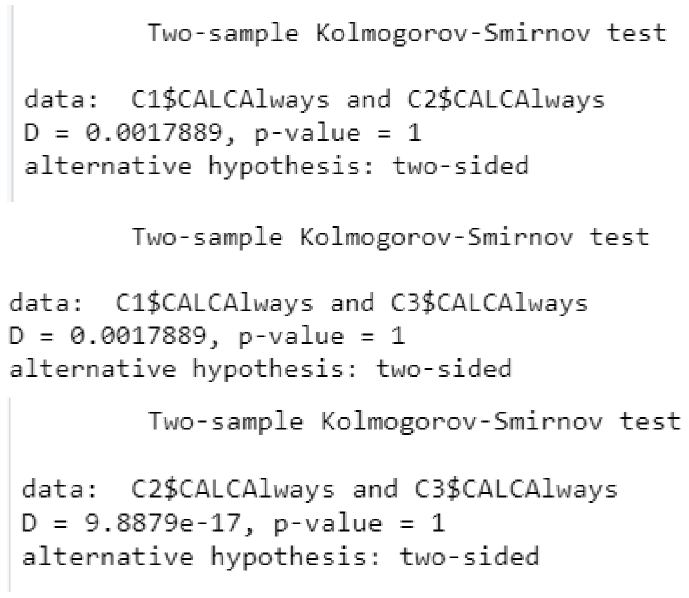
In order to be more accurate with variance among classes for the feature, the Kolmogorov-Smirov (KS) test was used which is a test for distributions adequacy or equality among features. The KS-test outputs were obtained from R which shows the p-value. Notice, the feature Age has very low p-value between all pairs of classes which suggests that Age has the variance among classes and passes as most important feature.



```
         Two-sample Kolmogorov-Smirnov test

data:  C1$Age and C2$Age
D = 0.32474, p-value < 2.2e-16
alternative hypothesis: two-sided

         Two-sample Kolmogorov-Smirnov test

data:  C1$Age and C3$Age
D = 0.49818, p-value < 2.2e-16
alternative hypothesis: two-sided

         Two-sample Kolmogorov-Smirnov test

data:  C2$Age and C3$Age
D = 0.20504, p-value = 1.088e-13
alternative hypothesis: two-sided
```

Figure 33. KS-Test Outputs for most important feature

On the other hand, the p-values between the pairs of classes for least important feature CALC Always were all close to approximately 1 which suggest that many of the observations are similar among different classes. This was expected as CALC is a binary variable and it takes 0 or 1 values among all classes.

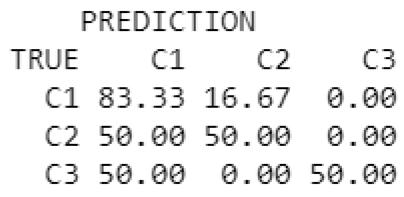


```
         Two-sample Kolmogorov-Smirnov test

data:  C1$CALCAlways and C2$CALCAlways
D = 0.0017889, p-value = 1
alternative hypothesis: two-sided

         Two-sample Kolmogorov-Smirnov test

data:  C1$CALCAlways and C3$CALCAlways
D = 0.0017889, p-value = 1
alternative hypothesis: two-sided

         Two-sample Kolmogorov-Smirnov test

data:  C2$CALCAlways and C3$CALCAlways
D = 9.8879e-17, p-value = 1
alternative hypothesis: two-sided
```

Figure 34. KS-Test Outputs for least important feature

K-means Small Gini w/ RF

From the results obtained from K-means of 17 clusters, cluster 17 had the lowest gini of 0.1604 which had 140 cases with the split of 128 in C1, 6 in C2, 6 in C3. This indicates the cluster is almost pure. As per the directions, the subset of cases for this cluster was divided into train and test sets and then 80-20 split and then C1 had to be extremely undersampled in order to balance the number of cases between all classes. C2 and C3 were also cloned to three times creating the distribution to 12 cases in each class of new TRAIN SET and 6 cases in each class of new TESTSET, totalling 36 cases in new TRAIN set and 18 cases in new TEST set. Of course, this total number of cases is extremely low due to poor separation between classes and pure cluster. After creating the train test set, RF function was applied with new training data and observations were predicted using the predict function for the test set. The confusion matrix below shows C1 being most dominant and other two classes are hugely misclassified despite the efforts of balancing the classes. This is probably due to using the pure cluster which generally would predict C1 for all cases if the classes had not been balanced which means extreme balancing in the cases of using pure clusters which have a high majority of only one class. The efforts of doing different train/test splits also resulted in poor test performance among C2 & C3. SepRF1

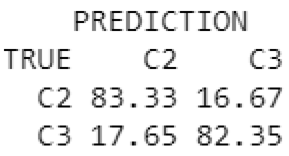


```
     PREDICTION
 TRUE    C1    C2    C3
   C1 83.33 16.67  0.00
   C2 50.00 50.00  0.00
   C3 50.00  0.00 50.00
```

Global accuracy = 61.11%

In another attempt to deal with a small number of classes, one of the ideas was to remove a class with an excessively lower number of cases for more accurate prediction of other two classes. This was done using cluster 7 which also had a gini index of 0.164 nearly the same GINI index as the lowest gini. This cluster had a distribution of 5 cases in C1, 28 in C2 and 339 in C3. By removing the C1 cases, C2 was cloned to triple the size for train and test set separately and C3 was undersampled by ¼ factor, resulting in a total 133 cases in trainset, with 66 belonging to C2 and 67 belonging to C3. In testset, the total was 36 cases with C2 having 18 cases and C3 having 17 cases. By training this train/test split in separate RF and predicting, the confusion matrix of the testset was obtained as shown below which shows better accuracy levels than it was compared with 3 classes. The global accuracies and class accuracies of C2 and C3 were close to best ntree=400 but best ntree=400 still had slightly higher accuracy than the newly trained model SepRF2.

SepRF2

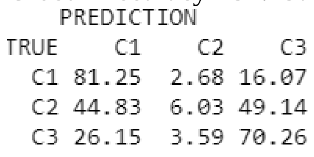


```
    PREDICTION
TRUE    C2    C3
  C2 83.33 16.67
  C3 17.65 82.35
```

Global accuracy = 82.86%

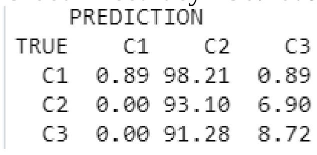
Due to the overall low number of cases in each cluster and based on the results obtained SepRF1 and SepRF2 for smallest gini indices, I do not think there is an advantage to training each cluster separately as doing so won't lead to any improvisation. However, to be confident and to cross-validate my prediction, each cluster was used to train SepRFj where j is cluster 1 to 17. As random Forest can also be used without balance of classes, compromising performance little bit, cases within each cluster was used to train separate RF for each cluster and the exhausting poor results are shown below which shows the better the separation between classes, the higher the accuracy but no cluster had an overall higher accuracy. In short, the reasons for poor performance may include low number of overall cases, poor separation of classes due to GINI purity close to zero.

CLU 1
Global Accuracy= 51.15%



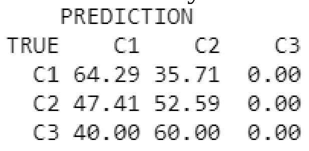
```
    PREDICTION
TRUE    C1    C2    C3
  C1 81.25  2.68 16.07
  C2 44.83  6.03 49.14
  C3 26.15  3.59 70.26
```

CLU 2
Global Accuracy= 36.10%



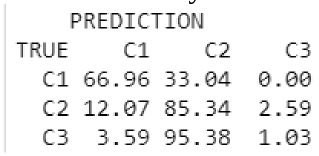
```
    PREDICTION
TRUE    C1    C2    C3
  C1  0.89 98.21  0.89
  C2  0.00 93.10  6.90
  C3  0.00 91.28  8.72
```

CLU 3
Global Accuracy= 40.86%



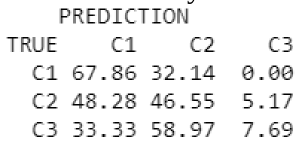
```
    PREDICTION
TRUE    C1    C2    C3
  C1 64.29 35.71  0.00
  C2 47.41 52.59  0.00
  C3 40.00 60.00  0.00
```

CLU 4
Global Accuracy= 53.76%



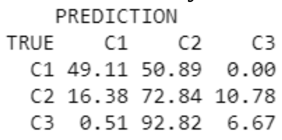
```
    PREDICTION
TRUE    C1    C2    C3
  C1 66.96 33.04  0.00
  C2 12.07 85.34  2.59
  C3  3.59 95.38  1.03
```

CLU 5
Global Accuracy= 42.24%



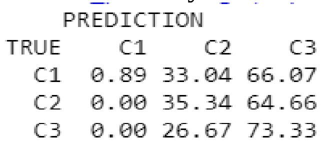
```
    PREDICTION
TRUE    C1    C2    C3
  C1 67.86 32.14  0.00
  C2 48.28 46.55  5.17
  C3 33.33 58.97  7.69
```

CLU 6
Global Accuracy= 44.85%



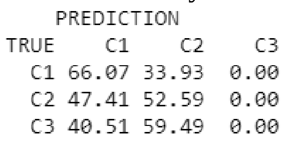
```
    PREDICTION
TRUE    C1    C2    C3
  C1 49.11 50.89  0.00
  C2 16.38 72.84 10.78
  C3  0.51 92.82  6.67
```

CLU 7
Global Accuracy= 34.87%



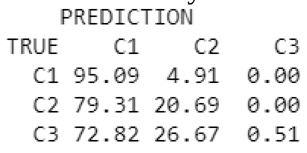
```
    PREDICTION
TRUE    C1    C2    C3
  C1  0.89 33.04 66.07
  C2  0.00 35.34 64.66
  C3  0.00 26.67 73.33
```

CLU 8
Global Accuracy= 41.47%



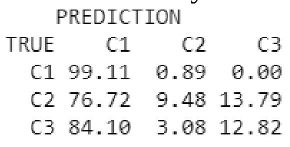
```
    PREDICTION
TRUE    C1    C2    C3
  C1 66.07 33.93  0.00
  C2 47.41 52.59  0.00
  C3 40.51 59.49  0.00
```

CLU 9
Global Accuracy= 40.23%



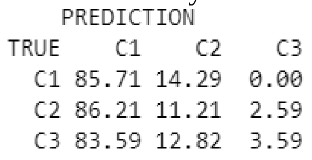
```
    PREDICTION
TRUE    C1    C2    C3
  C1 95.09  4.91  0.00
  C2 79.31 20.69  0.00
  C3 72.82 26.67  0.51
```

CLU 10
Global Accuracy= 41.32%



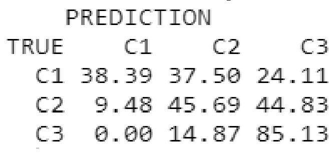
```
     PREDICTION
TRUE     C1    C2    C3
  C1 99.11  0.89  0.00
  C2 76.72  9.48 13.79
  C3 84.10  3.08 12.82
```

CLU 11
Global Accuracy= 34.56%



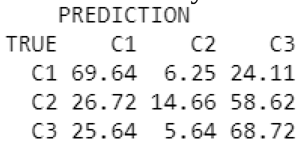
```
     PREDICTION
TRUE     C1    C2    C3
  C1 85.71 14.29  0.00
  C2 86.21 11.21  2.59
  C3 83.59 12.82  3.59
```

CLU 12
Global Accuracy= 54.99%



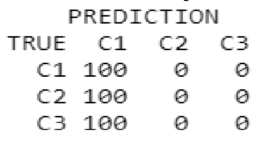
```
     PREDICTION
TRUE     C1    C2    C3
  C1 38.39 37.50 24.11
  C2  9.48 45.69 44.83
  C3  0.00 14.87 85.13
```

CLU 13
Global Accuracy= 49.77%



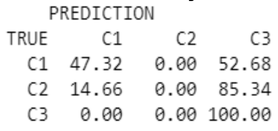
```
     PREDICTION
TRUE     C1    C2    C3
  C1 69.64  6.25 24.11
  C2 26.72 14.66 58.62
  C3 25.64  5.64 68.72
```

CLU 14
Global Accuracy= 34.41%



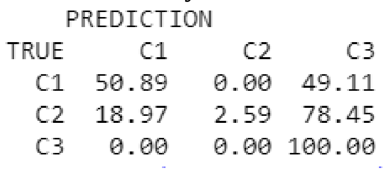
```
     PREDICTION
TRUE   C1  C2  C3
  C1 100   0   0
  C2 100   0   0
  C3 100   0   0
```

CLU 15
Global Accuracy= 46.24%



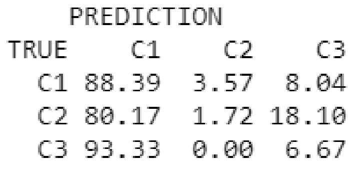
```
     PREDICTION
TRUE     C1     C2     C3
  C1  47.32   0.00  52.68
  C2  14.66   0.00  85.34
  C3   0.00   0.00 100.00
```

CLU 16
Global Accuracy= 48.39%



```
     PREDICTION
TRUE      C1     C2     C3
  C1   50.89   0.00  49.11
  C2   18.97   2.59  78.45
  C3    0.00   0.00 100.00
```

CLU 17
Global Accuracy= 33.03%



```
     PREDICTION
TRUE     C1    C2    C3
  C1 88.39  3.57  8.04
  C2 80.17  1.72 18.10
  C3 93.33  0.00  6.67
```

Support Vector Machine (SVM) Analysis
Yet another model of supervised machine learning is SVM (Support Vector Machine). SVM uses classification algorithms for two-group classification problems. It can be also used for regression analysis, although mainly used for classification. It is also used for optimization to find solutions (hyperplanes) with few errors and seek large margin separators to improve generalization. SVM looks at the training data and sports it into one of two categories. The distance between support vectors and hyperplanes should be as far as possible.

The SVM analysis was performed using the train set and testset created earlier in Question 5. However, only two classes were used. Since C1 and C3 used to have higher performance than C2, C1 and C3 were chosen for SVM classification analysis. In the train set, C1 had 894 cases and C3 had 777 cases. In the testset, C1 had 224 cases and C3 had 195 cases.

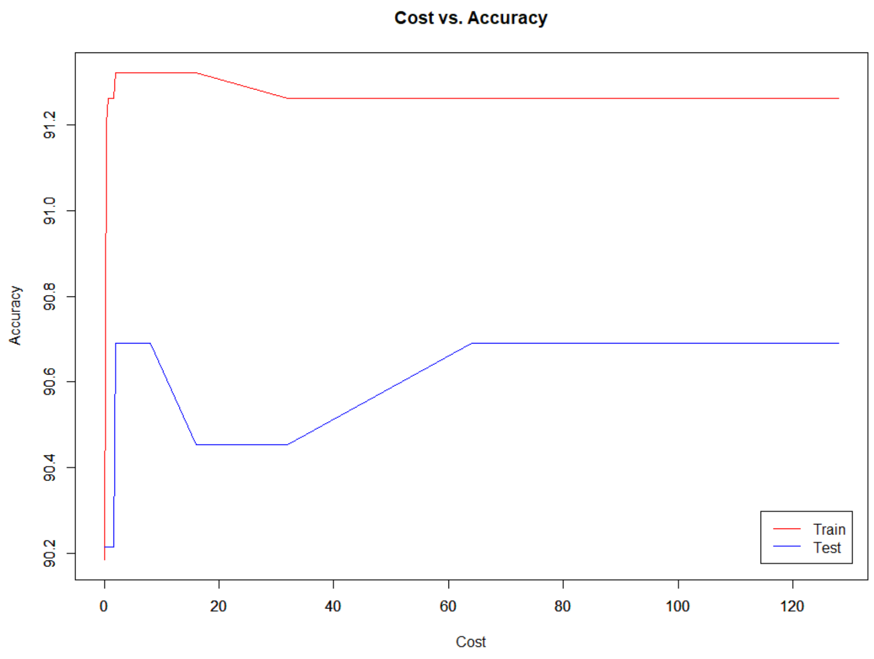


Figure 35: Cost vs. Accuracy for Training and Test data

SVM linear analysis were performed using existing svm functions in R. First, cost was tuned from several distinct values of cost c ranging from 0.01 to 128. The cost vs. accuracy plot for training data and testing data was plotted as shown in Figure 24. The clear pattern is observed when cost rapidly goes up at 2 and then declines slightly eventually flattening completely. The test accuracy showed patterns of increasing first, then decreasing, stable for while and then increasing again eventually flattening completely. There was not a big difference between the accuracies of train and test. The best cost for both was determined to be 2 after testing various cost c values. The output showed the number of support vectors for cost=2 is 379 in which C1 had 194 support vectors and C3 had 185 support vectors

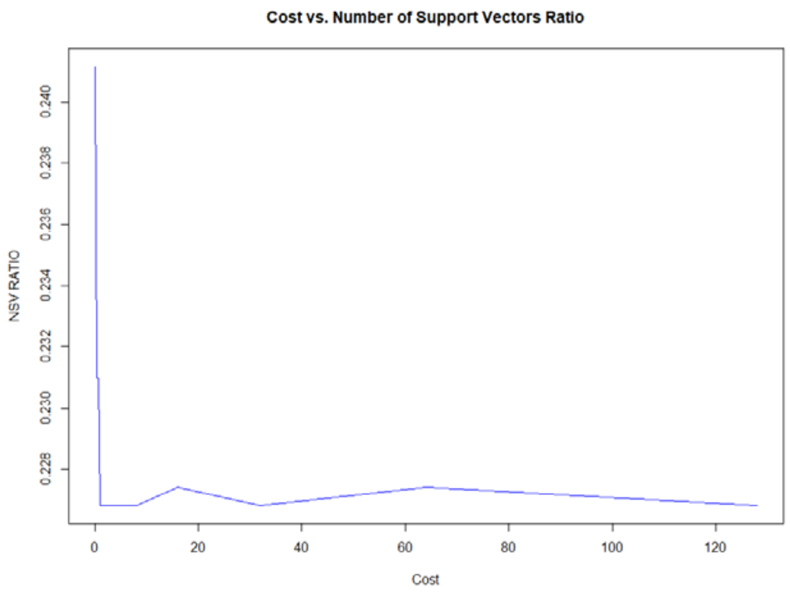


Figure 36. Cost vs. Ratio of Number of Support Vector over cases in training set

Additionally, to confirm the best cost c, the ratio of number of support vectors to the size of the training data set was obtained and plotted against each cost c. This showed the rapidly declining curve resulting in low ratio at cost=2. The value of the best cost c mendered from 2 to 7 in multiple launchings of the svm function. As the best cost c=2 matches for both plots, best c=2 was chosen to make predictions using predict function in R. The resulting confusion matrix is shown which shows the global accuracy of 91.89 for the test set.

```
     Prediction
TRUE    C1    C3
  C1 88.39 11.61
  C3  4.10 95.90
```

Global Accuracy= 91.89%

Attempts at Improvement

PCA

The percentage of variance explained was also tested for 90% instead of 95% which reduced 2 more features, making it 17 new features instead of 19 features. However, it did not change the visualization plots and did not change potentially due to only the first three principal components being used for visualization purposes.

K-means Clustering

In order to find the optimal number of k-clusters, the model was ran many times to make for a particular number of values in order to ensure the quality of the results. When clusters were used for Random Forest classification, several methods such as removing a class with extremely low number cases, comparing one class with the other two, etc.. were done.

Random Forest

In the initially chosen set of numbers of trees which included ntree of {10,50,100,200,300,400}, it was noticeable that the accuracy curve was stable after ntree=200. However, in the plot of three diagonal terms which shows the accuracy for each class at different numbers of trees, the accuracy for class 2 seemed to decline slightly after 400 number of trees and for class 1 and class 3, it seemed to increase slightly insignificantly. Thus, exploration with more number of trees was done to make sure there is no significant improvement with larger number trees. I explored 500, 750, 1000 numbers of trees in addition to the initially chosen set of ntrees. As it can be seen in Figure 26 and 27 , the best number of trees appears to be ntree=500 as the accuracy vs. ntrees plot appears to peak there for test curve and the class accuracy vs. ntrees plot shows that class 1 and class 2 accuracy increases at ntree=500 and then there is no significant improvement afterwards. This can be also observed across the confusion matrix and accuracy of different numbers of trees as shown in Figure 28. Even though the accuracies improved at ntree=500, the difference was not significant from ntree=400 which would not change the results dramatically.

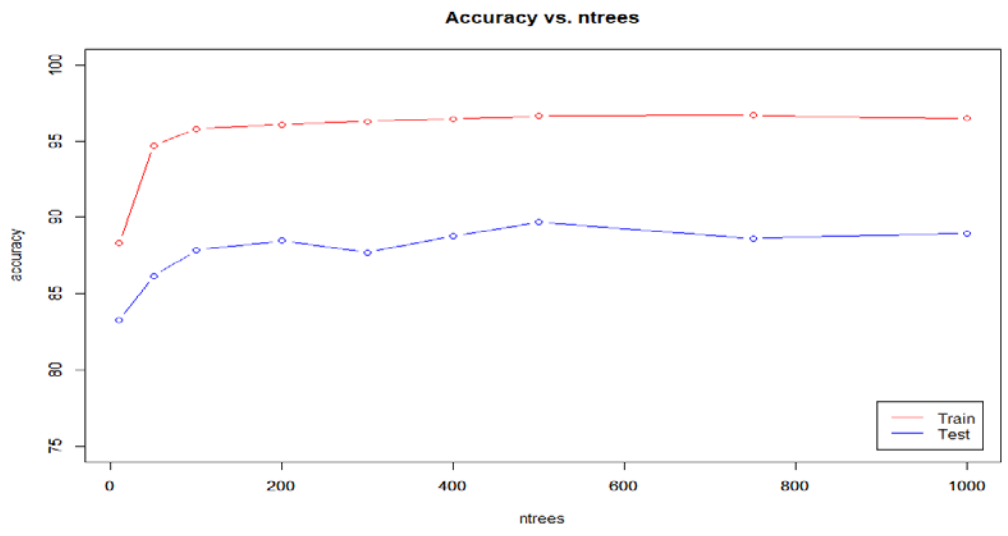


Figure 37. Accuracy vs. larger number of ntrees

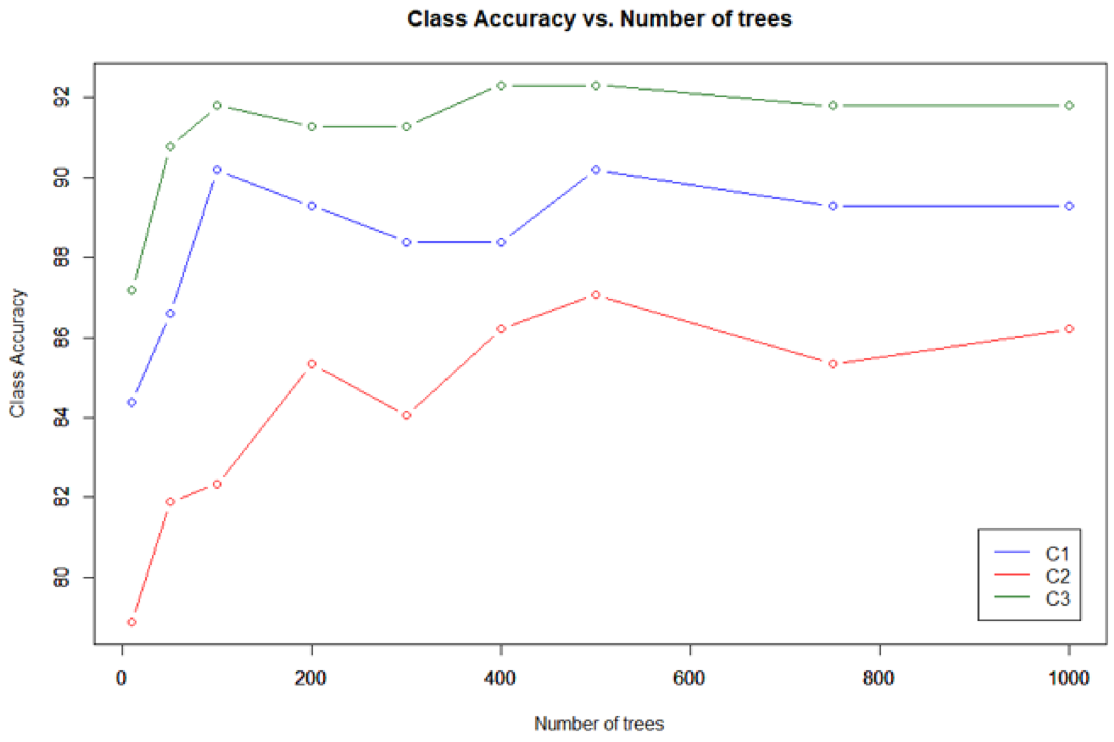
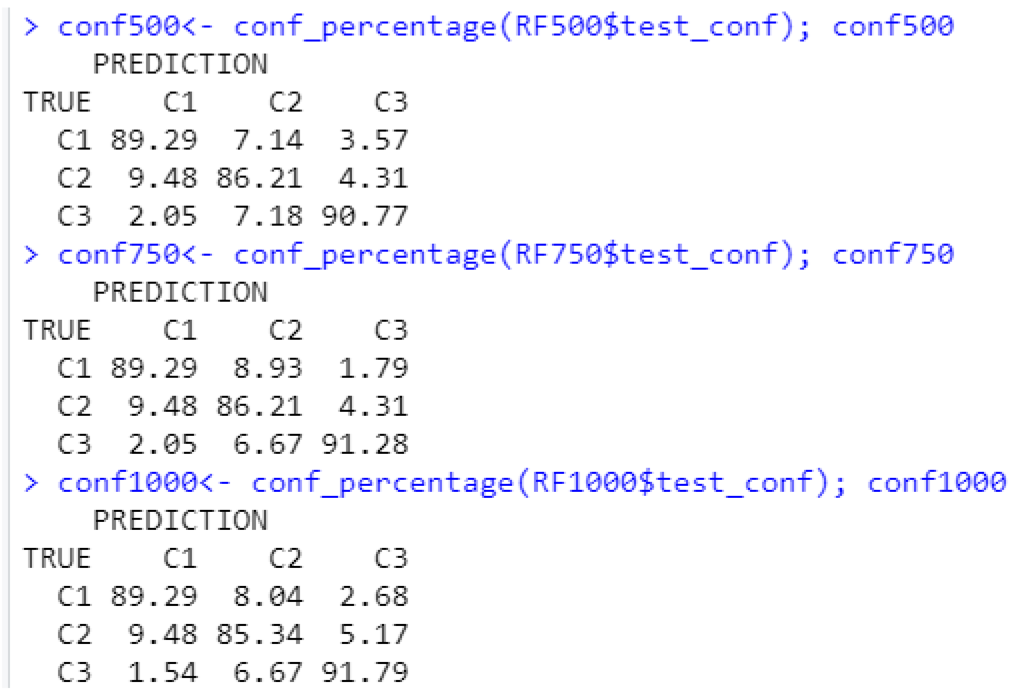


Figure 38. Class accuracies vs. larger number of trees.



```
> conf500<- conf_percentage(RF500$test_conf); conf500
    PREDICTION
TRUE    C1    C2    C3
  C1 89.29  7.14  3.57
  C2  9.48 86.21  4.31
  C3  2.05  7.18 90.77
> conf750<- conf_percentage(RF750$test_conf); conf750
    PREDICTION
TRUE    C1    C2    C3
  C1 89.29  8.93  1.79
  C2  9.48 86.21  4.31
  C3  2.05  6.67 91.28
> conf1000<- conf_percentage(RF1000$test_conf); conf1000
    PREDICTION
TRUE    C1    C2    C3
  C1 89.29  8.04  2.68
  C2  9.48 85.34  5.17
  C3  1.54  6.67 91.79
```

Figure 39. Confusion matrices of larger number of ntrees.

Conclusion

Utilizing PCA analysis reduced unimportant features, keeping the number of features to 19 with 95% PVE. If there was more time available, performing analysis using this dataset might have been beneficial for this dataset as it has a lot of binary features and PCA reduction gets rid of many binary features which could optimize the performance for other classifiers. Visualizing and comparing only two classes on the 3D Scatterplot helped to point out the variation between classes.

The goal of using K-means clustering was to find good unsupervised clustering for which the within-cluster variation is as small as possible and to see if it can produce better classification prediction accuracy when used with Random Forest classifier. However, K-means clustering requires us to pre-specify the number of clusters using a specific k from the elbow method which is not precise as it is mostly chosen based on visual representation which can be detrimental to prediction accuracy. As an alternative, GINI indexes, impurity and performance curve were also compared but it was still not as precise a method as a lot of computing time was wasted in trial and error. Of course, in reality clusters of users are nebulous and will certainly overlap, however this also makes it such that K-means doesn't need to find a true k, because no true k exists.

Comparison of Automatic Classifiers

	KNN	RF	SVM
C1 accuracy	94%	89%	88%
C2 accuracy	88%	85%	-
C3 accuracy	92%	91%	95%
Global accuracy	91%	89%	92%
Computing Time min:sec	5:22	0:35 (for ntree=400)	4:29

There was no classification analysis done for predicting accuracy with K-means but the average computing time for K-means was 2 minutes 57 secs

K-means clustering did not produce quality results with this data, but it is probably due to too many binary variables in the data and K-means is known to not work well with binary data. K-means might really shine in a context like targeted advertising, where enormous amounts of data are collected on customers such as categories of items they like to buy and their spending habits.

In this analysis of obesity dataset, binary encoding of categorical features improved the prediction ability of the study. Several features were compared on histograms and discriminating power was calculated for features based on t-test on three different groups of classes. I believe discriminating power played an essential role in improving accuracies for weighted KNN as it allowed us to visualize and place features with weaker discriminatory power in separate packs. KNN analysis resulted in higher testset accuracy which implies greater ability to predict the risk of obesity based on features. However, with low k-value being the best, the uncertainty of overfitting remains due to low number of testset cases. The bigger dataset with more cases could have been helpful to avoid the overfitting situation and the more the cases in testset, the better it is for the KNN algorithm.

Random forest achieved slightly higher prediction accuracy than KNN. Random forest also had the fastest computing time than any other algorithm we have learned this semester. On our typical personal computers, the average computational time for the random forest with 100 trees and mtry set at 10 was around 8.48 seconds. The average run time for the set of random forests with {10,50,100,200,300,400} respective trees from part 3 ran together took around 1 minute 10 seconds. For the bestRF classifier with 300 trees, the average run time was 23.42 seconds. Lastly, from the attempted improvements, we tried RF with 1000 trees, and the average run time for that classifier was 56.87

seconds. Ultimately, with these computational times on an ordinary computer, it can be said that random forest is very efficient to train on a normal sized dataset, especially when compared to the previous algorithms such as KNN and K-means clustering.

Support Vector Machine was the latest algorithm implemented to predict two-group classifications. The Support Vector was computationally expensive as the average run time for 100 distinct cost values was around 6 minutes 23 seconds. The SVM had the highest global accuracy among all classifiers.

Overall, this dataset seems like a good fit for supervised learning but does not seem to work well with unsupervised learning. However, it may be just that too many binary features affecting the performance of the model.