WAYNE STATE UNIVERSITY
COLLEGE OF ENGINEERING
COMPUTER SCIENCE DEPARTMENT

Course Project

CSC5800: Intelligent Systems

Parkinson's Disease Analysis

Supervised by

Dr. Suzan Arslantürk

Submitted by:

Debadeep Pharikal
dpharikal@wayne.edu
Access Id – gu5308
Cell# 248-882-5025

Table of Contents

ABSTRACT

This course projects includes analysis on Parkinson's disease which revolves around two telemonitoring datasets from UCI. The major goal of this course project is to experiment all the aspects covered under CSC 5800 - Intelligent Systems and work towards implementation of those knowledge to develop a near to perfect model in predicting Parkinson's disease. The more accurate the model are, more chances of artificial systems to predict if the person is having Parkinson's disease.

1. Introduction: Overview

Parkinson's disease is a neurodegenerative disorder that affects movement.

Approximately 60,000 Americans are diagnosed with Parkinson's disease each year, and this number does not reflect the thousands of cases that go undetected. More than 10 million people worldwide are living with PD.
[Source: https://www.parkinson.org/Understanding-Parkinsons/Statistics]

It develops over years and the symptoms varies from person to person owing to the diversity of the disease. The cause to this disease remains largely unknown. The usage of speech-based data in the classification of Parkinson disease (PD) has been shown to provide an effect, non-invasive mode of classification. This led to increased interest in speech analysis.

2. Data: Dataset Overview

The entire analysis revolves around two datasets obtained from UCI Machine Learning Repository

i) parkinsons.data and (will term as Parkinson's Dataset)
ii) parkinsons_updrs.data(will term as Parkinson's UPDRS Dataset)

Definitions of the dataset is as follows: [Source:UCI]
i) This dataset consists of a range of biomedical voice measurements from 31 individual, 23 with Parkinson's disease (PD). Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals. The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD.
ii)This dataset is based upon pro-longed experiment which consists of a range of biomedical voice measurements captured from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring. The recordings were automatically captured in the patient's homes.

The reason two datasets are chosen for this analysis will be elaborated later.

2.1 Attribute Information: *Following are the attribute details about Parkinson's Data and Parkinson's UPDRS data.*

parkinsons.data		
1	MDVP:Fo(Hz)	Average vocal fundamental frequency
2	MDVP:Fhi(Hz)	Maximum vocal fundamental frequency
3	MDVP:Flo(Hz)	Minimum vocal fundamental frequency
4	MDVP:Jitter(%)	Measures of variation in fundamental frequency

5	MDVP:Jitter(Abs)	Measures of variation in fundamental frequency
6	MDVP:RAP	Measures of variation in fundamental frequency
7	MDVP:PPQ	Measures of variation in fundamental frequency
8	Jitter:DDP	Measures of variation in fundamental frequency
9	MDVP:Shimmer	Measures of variation in amplitude
10	MDVP:Shimmer(dB)	Measures of variation in amplitude
11	Shimmer:APQ3	Measures of variation in amplitude
12	Shimmer:APQ5	Measures of variation in amplitude
13	MDVP:APQ	Measures of variation in amplitude
14	Shimmer:DDA	Measures of variation in amplitude
15	NHR	Measures of ratio of noise to tonal components in the voice
16	HNR	Measures of ratio of noise to tonal components in the voice
17	RPDE	Nonlinear dynamical complexity measures
18	D2	Nonlinear dynamical complexity measures
19	DFA	Signal fractal scaling exponent
20	spread1	Nonlinear measures of fundamental frequency variation
21	spread2	Nonlinear measures of fundamental frequency variation
22	PPE	Nonlinear measures of fundamental frequency variation
23	status	Health status of the subject(one)-> Parkinson's, (zero)-> Healthy

parkinsons.updrs.data		
1	subject	Integer that uniquely identifies each subject
2	age	Integer that depicts age of an individual
3	sex	gender '0' - male, '1' - female
4	test_time	Time since recruitment into the trial.
5	motor_updrs	Clinician's motor UPDRS score
6	total_updrs	Clinician's total UPDRS score, linearly interpolated
7	Jitter(%)	measure of variation in fundamental frequency
8	Jitter(Abs)	measure of variation in fundamental frequency
9	Jitter(RAP)	measure of variation in fundamental frequency
10	Jitter(PPQ5)	measure of variation in fundamental frequency
11	Jitter(DDP)	measure of variation in fundamental frequency
12	Shimmer	measure of variation in amplitude
13	Shimmer(dB)	measure of variation in amplitude
14	Shimmer:APQ3	measure of variation in amplitude
15	Shimmer:APQ5	measure of variation in amplitude

16	Shimmer:APQ11	measure of variation in amplitude
17	Shimmer:DDA	measure of variation in amplitude
18	NHR	measures of ratio of noise to tonal components in the voice
18	HNR	measures of ratio of noise to tonal components in the voice
20	RPDE	A nonlinear dynamical complexity measure
21	DFA	Signal fractal scaling exponent
22	PPE	A nonlinear measure of fundamental frequency variation

2.2 Data Exploratory Analysis

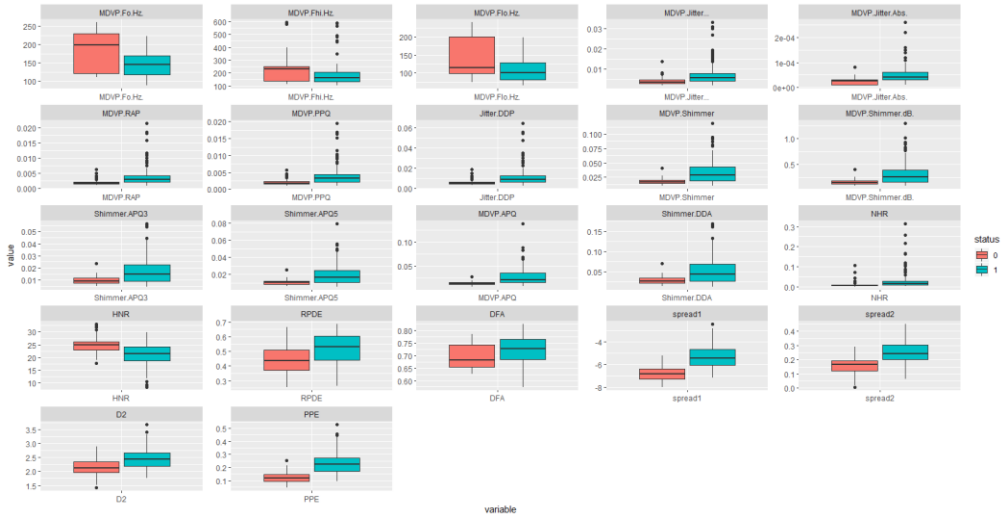
I plotted both the data on a histogram to understand its distribution. This step is essential fruitful analysis and understand the Parkinson's disease problem. This kind of analysis is also useful when we do not know what the right range of value is an attribute can hold especially for a dataset like this.

The *parkinsons.data* has a column called "status" which determines if a person is having Parkinson's disease. The status of the subject holds two values (one) - Parkinson's, (zero) – healthy. So, the attributes are plotted based on status which will clearly help us to understand a pattern how factors are contributing to Parkinson's.

Clearly since I already have the data which has status so I must develop a model based on this data which can predict if a person's speech symptoms can classify a Parkinson disease problem.

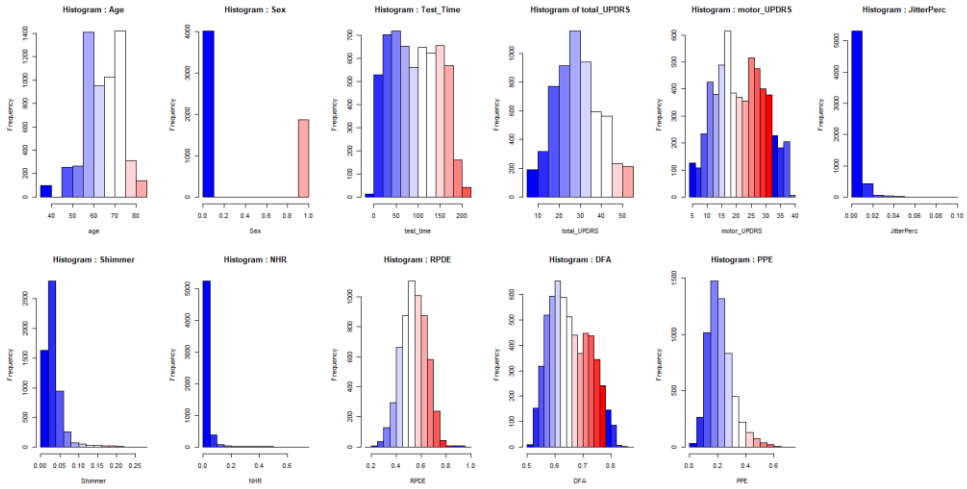
For simplicity I was able to plot the outliers in the same graph and observe that there are outliers also. The entire discussion regd. outlier its need for removal and procedure is provided in next section.

Fig 2.1 Histogram & Outlier Analysis: parkinson.data



Next histogram plot is plotted for telemonitoring data *parkinson.updrs.data* to understand sense of response variables and the co-variates and whether the data is normally distributed.

Table 2.2 Histogram & Outlier Analysis: parkinson.updrs.data

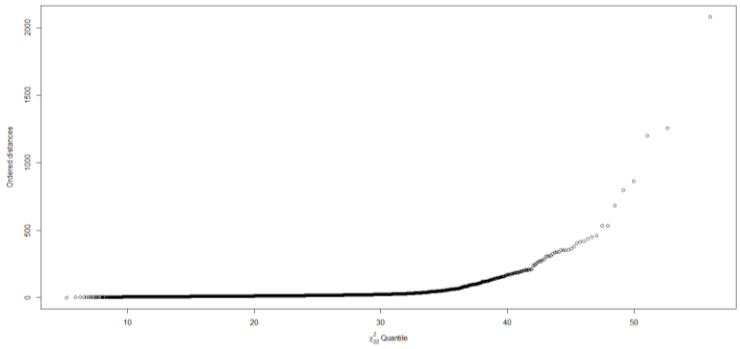


From all the observed histograms, it can be concluded that 'RPDE' and 'PPE' have the normalized data. 'Jitter (%)', 'Shimmer' and 'NHR' have very unnormalized data.

Used a technique called Chi-squareplot to understand the multivariate data furthermore. The function chisq.plot plots the ordered robust mahalanobis distances of the data against the quantiles of the Chi-squared distribution.

As we know for a normally distributed data a chi-square plot depicts the values approximately corresponds to each other and any outliers can be detected visually.

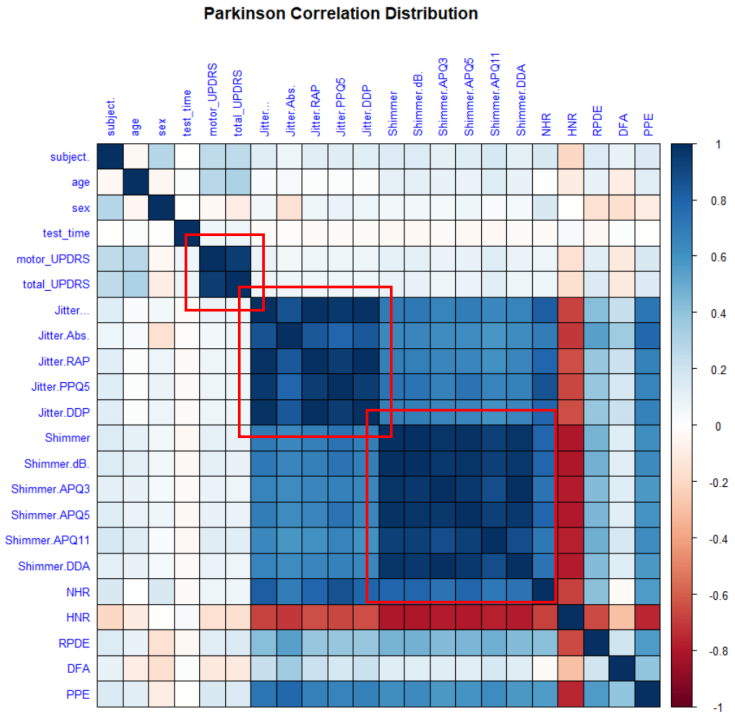
Table 2.3 Chi-square plot: Normality Test



From the above figure, I can see that my **multivariate data is not perfectly normally distributed. There may be some outliers in my dataset** and I need to remove them during pre-processing. Even though the data is not perfectly normally distributed I may have to proceed without normalization of the variables as it is out of scope of the analysis and I have taken this decision as I observed that the most important variable in the data containing UPDRS score(total_updrs) is almost normally distributed.

Before doing any pre-processing of data I also need to identify if there are attributes which are co-related.

Fig 2.4 Correlation Plot



Observation: Observed the following from the co-relation plot

- Jitter variables are highly correlated
- All the Shimmer variables are highly co-related
- Total UPDRS' has correlation with only Motor UPDRS

As I see both data are different and I have different problems to deal with. I have decided to provide results and analysis of Preprocessing, Feature Selection, Classification & Performance measures separately.

A. Parkinson UPDRS Dataset Analysis

3. Preprocessing :

As a part of pre-processing I am targeted to get rid of those variables which are null or non-numeric. Also, I plotted box plot of variables to observe if the data is having any outliers. Most parametric statistics which is used in my project like mean, correlations, and every statistic based on these, are highly sensitive to outliers.

3.1 Anomaly Detection

Firstly, my objective was to handle those data which have null value by removing them from the dataset as they do not lead to any conclusion and in a problem of disease prediction I must eliminate indecisiveness. So, I checked if the data has any missing values.

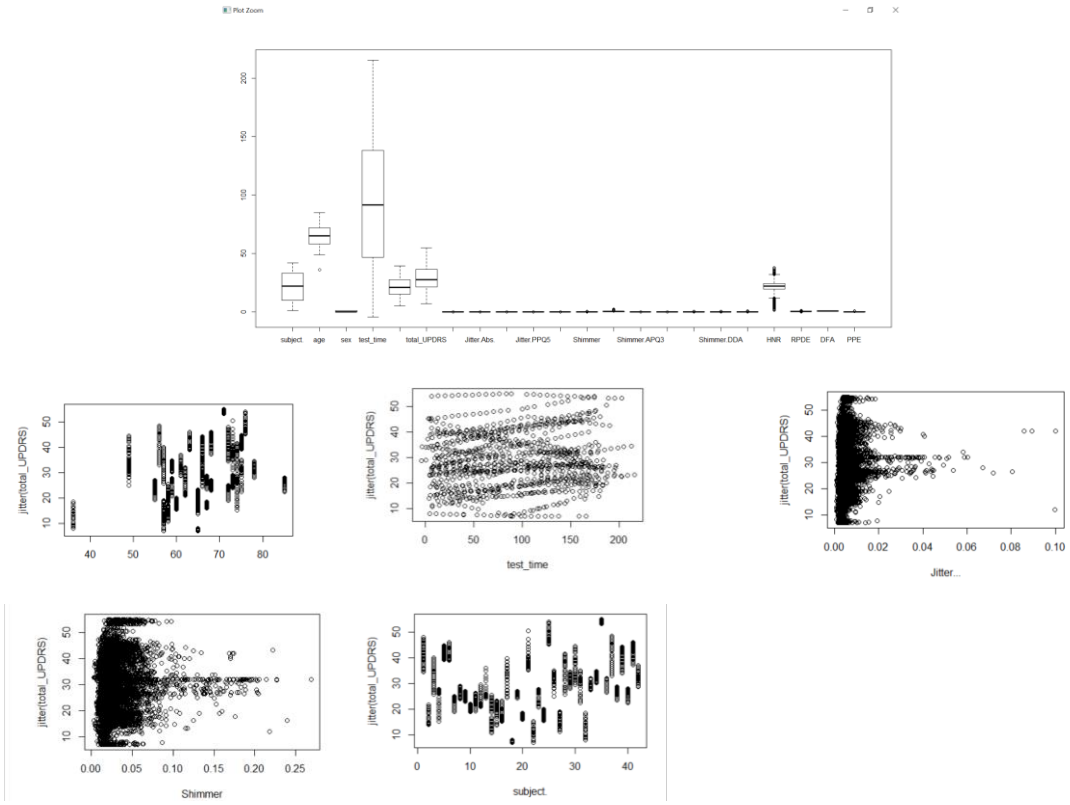
```
> na = colSums(is.na(park))
> na
            name      MDVP.Fo.Hz.     MDVP.Fhi.Hz.     MDVP.Flo.Hz.   MDVP.Jitter...
               0                0                0                0                0
MDVP.Jitter.Abs.      MDVP.RAP      MDVP.PPQ       Jitter.DDP     MDVP.Shimmer
               0                0                0                0                0
MDVP.Shimmer.dB.  Shimmer.APQ3  Shimmer.APQ5       MDVP.APQ      Shimmer.DDA
               0                0                0                0                0
             NHR              HNR           status             RPDE              DFA
               0                0                0                0                0
         spread1          spread2               D2              PPE
               0                0                0                0
>
```

The result shows there are no missing values in the dataset which is good for my analysis.

3.2 Outlier Detection & Removal:

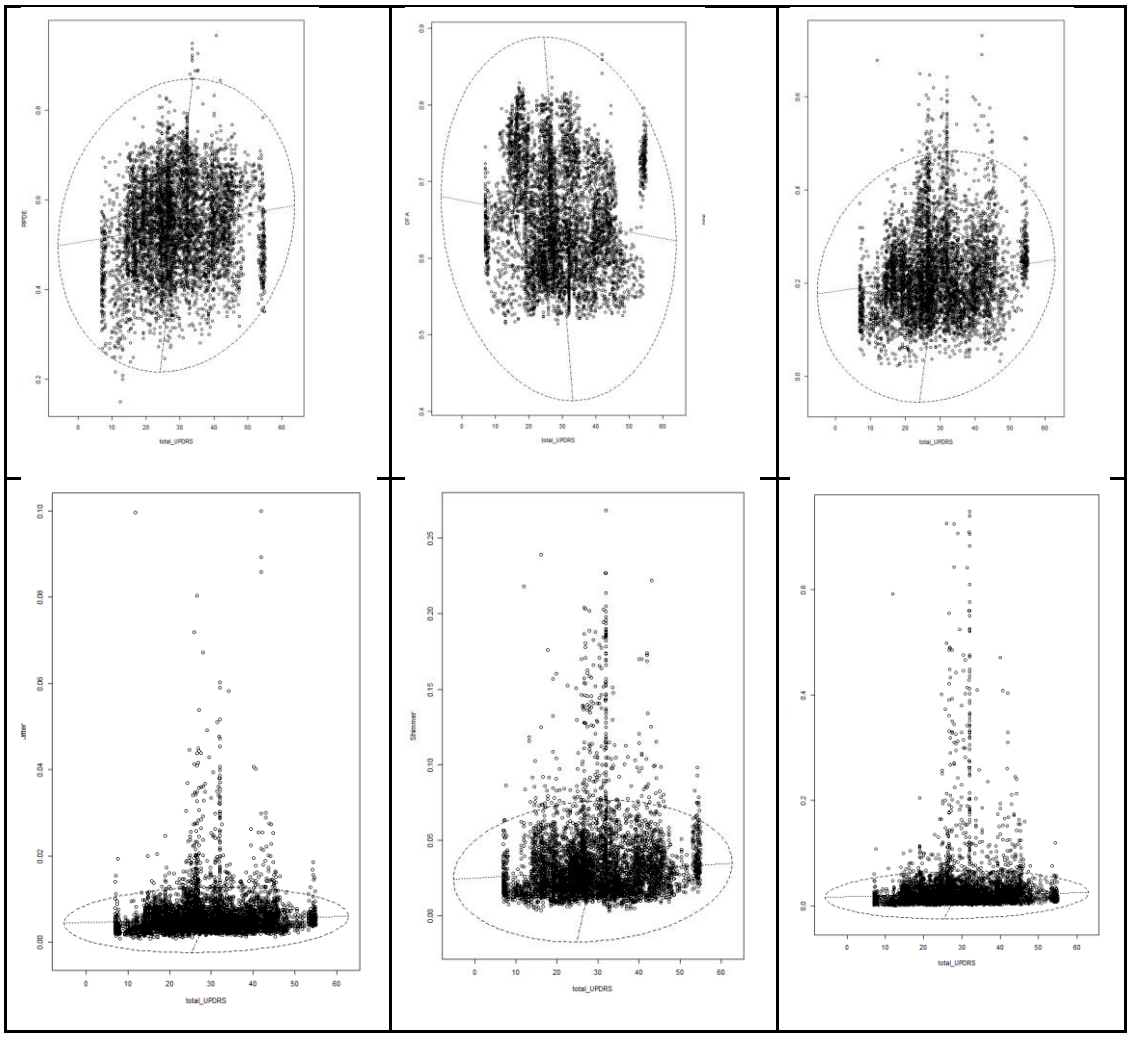
As I already know that there are outliers in dataset, and it seems it's appropriate to see each of the attributes for which the values are out of range.

Fig 3.5 Box Plot



Since these are multivariate data so I tried to plot those variables in a multi variate plot against "total_UPDRS".

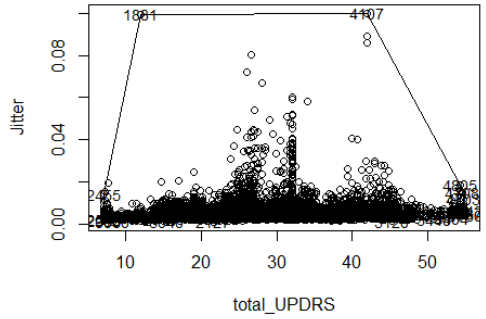
Fig 3.6 Bi-variate Box Plot



Outlier Reduction:

Next, I have removed outlier observations using Convex hull.

Fig 3.7 Bi-variate Box Plot(after Outlier Reduction)



3.3 Feature Selection:

I removed the feature "subject" to perform feature selection using Boruta since subject is a categorical attribute and has nothing to do with classification.

Boruta Algorithm:

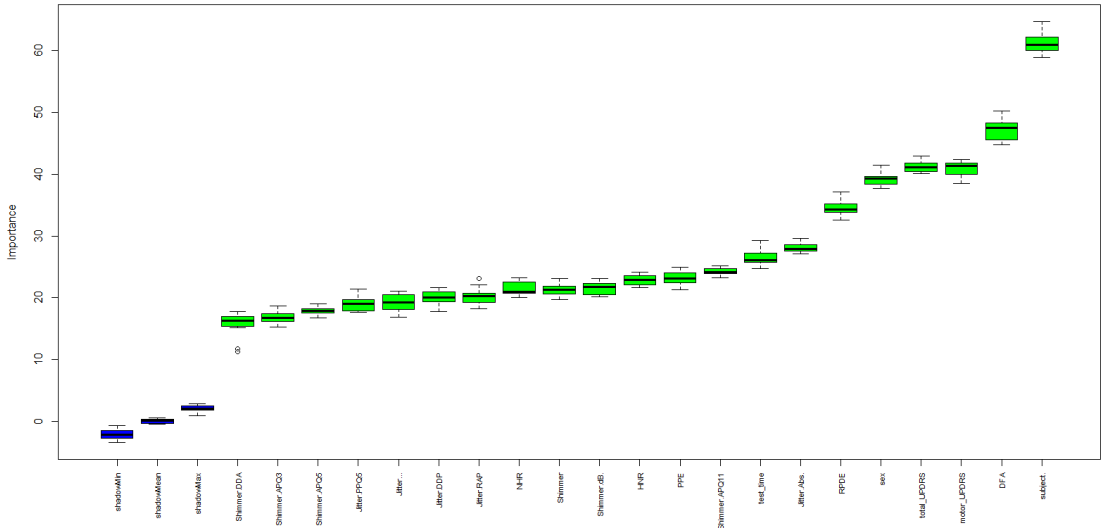
The Boruta algorithm is a wrapper encompasses around the random forest classification algorithm. It tends to capture all the vital, interesting features I might have in my dataset with respect to an outcome variable.

Output:

```
After 12 iterations, +1 mins:
 confirmed 21 attributes: DFA, HNR, Jitter..., Jitter.Abs., Jitter.DDP and 16 more;
 no more attributes left.

> print(Borutatrain)
Boruta performed 12 iterations in 1.044554 mins.
 21 attributes confirmed important: DFA, HNR, Jitter..., Jitter.Abs., Jitter.DDP and 16 more;
 No attributes deemed unimportant.
```

As per o/p I found **none** of the attributes can be removed. I plotted the importance of the attributes as follows:

Fig 3.1 Boruta Plot

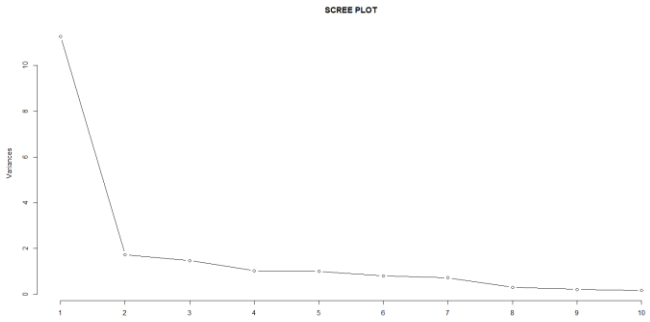


Even I could not find much help from Boruta algorithm, I tried to explore other options like Dimensionality reduction which is important technique in handling high dimensional dataset. However, the dataset like these are not the right choice for PCA. Following are the executions & observations:

3.4 Dimensionality Reduction:

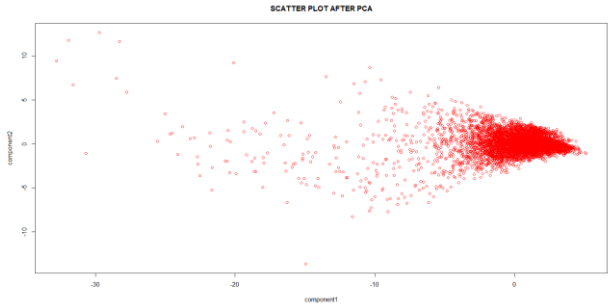
Principle Component Analysis: *Principal component analysis (PCA) is a statistical procedure which boils down to dimensionality reduction that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly non-correlated variables called principal components.[Source: Wikipedia]*

Fig 3.*2 Scree plot*



I have performed PCA on the UPDRS dataset and have used first two components of PCA for further analysis.

Fig 3.*3 Scatter plot after PCA*

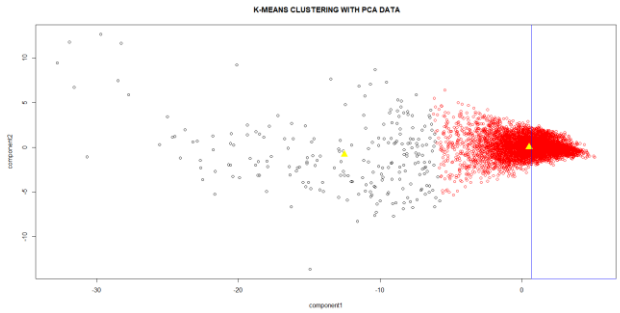


PCA is essentially a method that reduces the dimension of the feature space in such a way that new variables are orthogonal to each other (i.e. they are independent or not correlated).

3.5 K-means Clustering

This clustering is a method of vector quantization popular for cluster analysis in data mining. K-means begins with a randomly selected centroids and works iteratively to find optimized centroids. Here I found two distinct centroids. **Plotted with k=2**

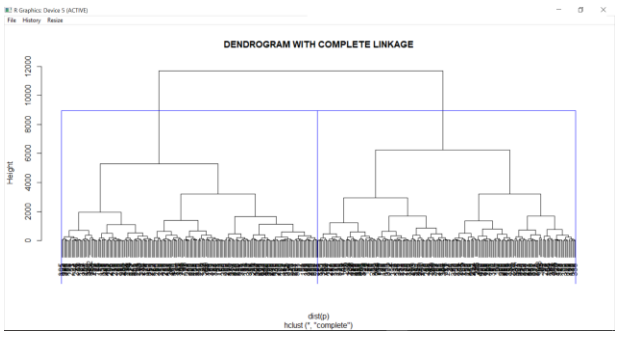
***Fig** 3.4 K-means Clustering with PCA data*



3.6 Hierarchical Clustering with Complete Linkage

Complete-linkage clustering is one of several methods of agglomerative hierarchical clustering.(k=2)

***Fig** 3.5 Complete Linkage with PCA data*



Observation: Dendrogram

After Principal Component Analysis for a random sample of 400 observations I get two solid clusters that captures "motor_UPDRS" and "total_UPDRS".(as described above)

Observation after Dimensionality Reduction:

However, I am dropping this inference as I cannot classify the data based on only just two factors because **If I build a model using these two determining variables, then it will have an inherent bias induced into the estimates of coefficients and variance of predicted response which is not intended. Hence even after all those feature selection techniques I will build the model with** "total_UPDRS" as response variables and rest others as predictor variables.

4. Classification: Algorithm Selection

Random Forest Classifier:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes my model's prediction.[Source: https://towardsdatascience.com/understanding-random-forest-58381e0602d2]

Steps of Classification

Set up training and test data sets:

```
#Training a model on the data
##Set up trainning and test data sets:
set.seed(350)
indx = sample(1:nrow(p), as.integer(0.9*nrow(p)))
indx[1:10]
```

The output shows that the random forest contains 500 trees and tried 6 variables at each split, 96.79% of the variation can be explained by my model.

```
> #Training a model on the data
> ##Set up trainning and test data sets:
> set.seed(350)
> indx = sample(1:nrow(p), as.integer(0.9*nrow(p)))
> indx[1:10]
 [1]  573  947 4734 1754 4206 4175 1275 2328 5666 3104
> p_train = p[indx,]
> p_test = p[-indx,]
> library(randomForest)
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
> rf<- randomForest(total_UPDRS~ ., data =p_train )
> rf

Call:
 randomForest(formula = total_UPDRS ~ ., data = p_train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 3.456223
                    % Var explained: 96.79
```

Importance of each predictor:

```
> importance(rf)
              IncNodePurity
subject.          205857.296
age               160154.612
sex                18096.551
test_time          21677.738
Jitter...           4918.665
Jitter.Abs.        13711.500
Jitter.RAP          4080.539
Jitter.PPQ5         4960.149
Jitter.DDP          4384.442
Shimmer             4792.227
Shimmer.dB.         4278.941
Shimmer.APQ3        5272.660
Shimmer.APQ5        5954.384
Shimmer.APQ11       6061.087
Shimmer.DDA         5472.053
NHR                 7015.334
HNR                13981.443
RPDE               13737.868
DFA                41690.173
PPE                10877.285
```

Plot of Importance of each variables explained as below

```
> #Plot of Important Variables
> varImpPlot(rf)
> #Evaluating Model Performance
> pred<-predict(rf,p_test,type='response')
> head(pred)
      21       47       53       65       68       73
44.15631 43.89570 36.18615 41.39274 43.41590 40.09386
> summary(pred)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  8.719  21.562  27.864  28.934  36.490  54.314
> summary(p$total_UPDRS)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   7.00   21.54   27.52   28.95   36.02   54.87
> cor(pred,p_test$total_UPDRS)
[1] 0.990016
```

The correlation between predicted and actual total UPDR is 99.001%.

Correlation only measures how correctly the predictions are related to the true value. However it is does not measure how far off the predictions were from the true values.

Another way to depict model's performance is to consider how far, on average, its prediction was from the true value which is given by **mean absolute error (MAE).** The equation for MAE is as follows, where *n* indicates the number of predictions and e_i indicates the error for prediction i: Function to calculate the mean absolute error:

The mean absolute error is given by:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$ [1]

Console Snippet

```
> MAE <- function(actual, predicted) {
+   mean(abs(actual - predicted))
+ }
> MAE(pred, p_test$total_UPDRS)
[1] 1.311008
```

This shows that on an average, the difference between my model's predictions and the true total UPDRS score was about 1.31.

On a scale from 0 - 10, this seems to suggest that my model is doing fairly well.

Improve model performance:

Though the model is doing well I tried to increase the randomly selected features at each split to 10 from previous 6. It shows that % var is increased from 96.79% to 99.16%

```
Call:
 randomForest(formula = total_UPDRS ~ ., data = p_train, mtry = 10)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 10

          Mean of squared residuals: 0.9053813
                    % Var explained: 99.16
>
```

The correlation between predicted and actual total UPDR is 99.695%.

```
> summary(pred1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.399  21.121  27.235  28.965  37.249  54.523
> summary(p$total_UPDRS)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   7.00   21.54   27.52   28.95   36.02   54.87
> cor(pred1,p_test$total_UPDRS)
[1] 0.9969521
```

Mean absolute error is given by 0.5894

```
> cor(pred1,p_test$total_UPDRS)
[1] 0.9969521
> MAE(pred1, p_test$total_UPDRS)
[1] 0.5894105
```

MAE decreased from 1.31 to 0.58

Classification Model: SVM

*A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.*

This method is an extension of the support vector classifier that results from enlarging the feature space using kernels. Where K is some non-linear kernel function that quantifies the similarity of two observations. Essentially, we fit a support vector classifier in a higher-dimensional space using a polynomial kernel of d-dimensions to create a support vector machine.

[Source: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72*]*

With a smaller value of cost I obtain a margin number of support vectors via the larger margin. The e1071 library has a built-in tune() function that performs 10-fold cross-validation on the models.

Linear Kernel:

Tuning Parameters: cost=c(0.001, 0.01, 0.1, 1,5,10), epsilon=c(0.1,0.5,1)))

```
Console  Terminal
~/
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost epsilon
  0.1     0.5

- best performance: 82.42757
```

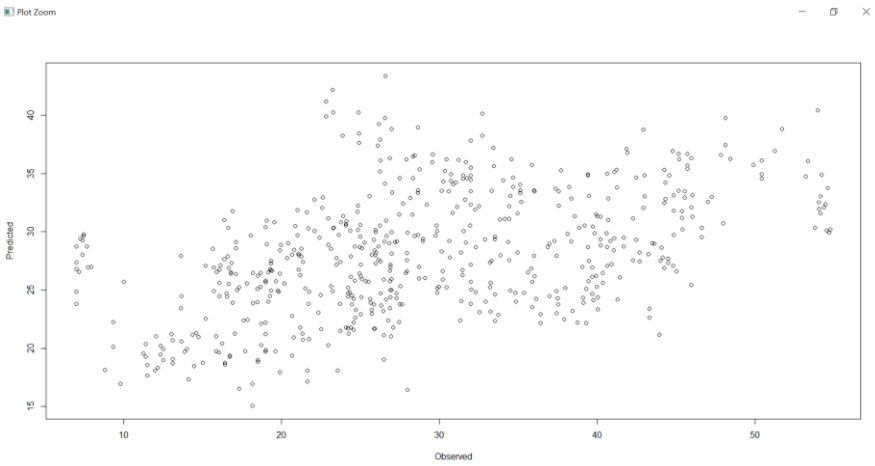
Best Performance = 82.42%

```
Console  Terminal
~/
> bestmod=tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = total_UPDRS ~ ., data = p_train, ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5,
    10), epsilon = c(0.1, 0.5, 1)), kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.05
    epsilon:  0.5


Number of Support Vectors:  3072
```

Root Mean Square Error:

```
> svmRMSE2 <- sqrt(mean((predict2-p_test$total_UPDRS)^2))
> svmRMSE2
[1] 9.543529
```

Plot:



Radial Kernel:

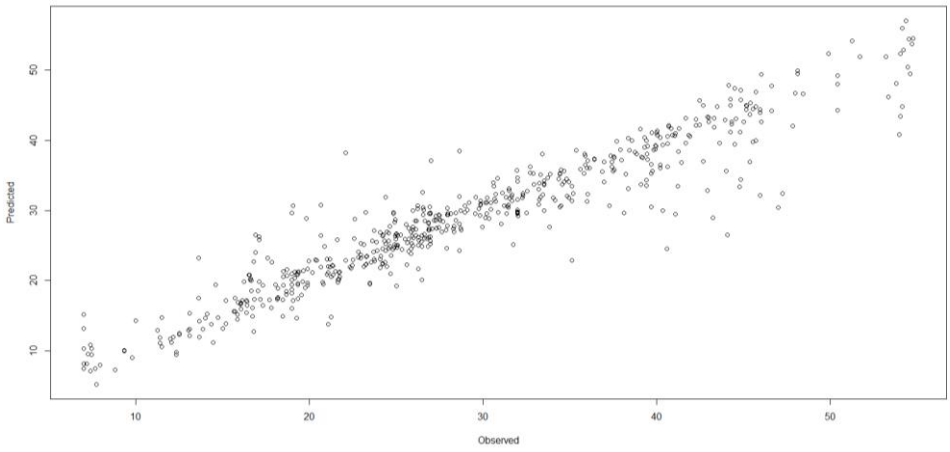
For ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10),gamma=c(0.1,0.3,0.4,0.5,1,2)))

```
Console  Terminal

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10   0.3

- best performance: 11.98956
```

```
Call:
best.tune(method = svm, train.x = total_UPDRS ~ ., data = p_train, ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5,
    10), gamma = c(0.1, 0.3, 0.4, 0.5, 1, 2)), kernel = "radial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.3
    epsilon:  0.1


Number of Support Vectors:  3073
```

After applying the model in the test set I am get following Root Mean Square Error

```
> predict2 <- predict(bestmod,p_test, type= 'response')
> svmRMSE2 <- sqrt(mean((predict2-p_test$total_UPDRS)^2))
> svmRMSE2
[1] 3.533158
```



SVM(Polynomial Kernel)

For tuning parameter : ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10), degree=c(2,3,4)))

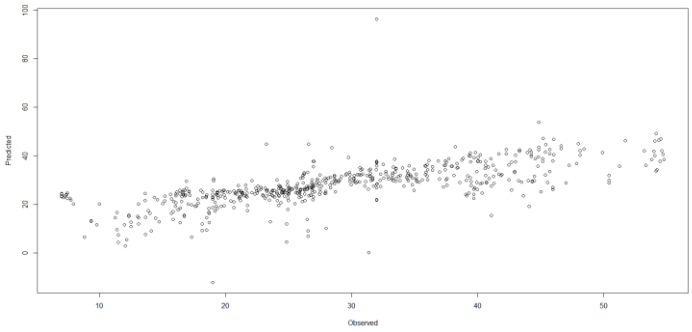
```
> #poly
> tune.out=tune(svm,total_UPDRS~.,data=p_train,kernel="polynomial",
+               ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10),
+                           degree=c(2,3,4)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
    5      3

- best performance: 61.19011
```

Best Performance: 61.19%

```
> bestmod=tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = total_UPDRS ~ ., data = p_train, ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5,
    10), degree = c(2, 3, 4)), kernel = "polynomial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  polynomial
       cost:  5
     degree:  3
      gamma:  0.05
    coef.0:  0
    epsilon:  0.1


Number of Support Vectors:  4226
```

```
> predict2 <- predict(bestmod,p_test, type= 'response')
> svmRMSE2 <- sqrt(mean((predict2-p_test$total_UPDRS)^2))
> svmRMSE2
[1] 7.871141
> plot(p_test$total_UPDRS,predict2, xlab = "Observed",ylab = "Predicted")
```

Plot:



So the best model is linear kernel model of **SVM giving best performance as it has higher RMSE and also RMSE should be more useful when large errors are particularly undesirable for the cases of disease identification problem,**

Now as I have shared the details of comparison of two classifiers, I would like to share why I reached this path of choosing the dataset. I have explored all the other algorithms with UPDRS data. I will report all the details here henceforth.

B. Parkinson Data Analysis

3. Preprocessing :

3.1 *Anomaly Detection:*

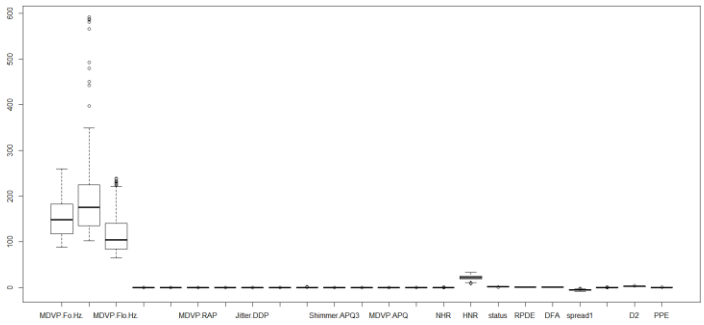
The dataset doesn't have any null values.

```
Console  Terminal
> na = colSums(is.na(park))
> na
            name      MDVP.Fo.Hz.     MDVP.Fhi.Hz.     MDVP.Flo.Hz.   MDVP.Jitter...  MDVP.Jitter.Abs.
               0                0                0                0               0                 0
        MDVP.RAP        MDVP.PPQ      Jitter.DDP     MDVP.Shimmer MDVP.Shimmer.dB.      Shimmer.APQ3
               0                0                0                0               0                 0
    Shimmer.APQ5        MDVP.APQ     Shimmer.DDA              NHR             HNR            status
               0                0                0                0               0                 0
            RPDE              DFA         spread1          spread2              D2               PPE
               0                0                0                0               0                 0
```

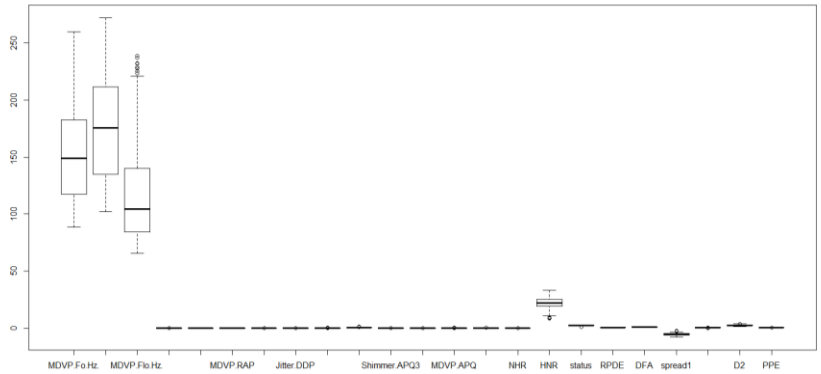
3.2 *Outlier detection &removal:*

I used boxplots to identify if the data has any outlier and removed them

Before Outlier Reduction:



After Outlier Reduction:

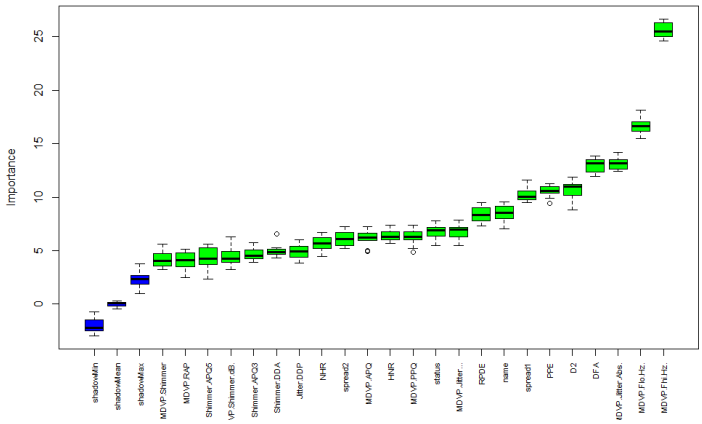


3.3 ***Feature selection:***

Boruta Algorithm: No features seem deemed unimportant after Boruta plot

```
> # Borutatrain <- Boruta(name ~ ., data = pdata, doTrace = 2)
> print(Borutatrain)
Boruta performed 16 iterations in 1.434128 secs.
 23 attributes confirmed important: D2, DFA, HNR, Jitter.DDP, MDVP.APQ and 18 more;
 No attributes deemed unimportant.
```

Boruta Plot



4. Classification: Algorithm Selection

Logical Regression: *Logistic regression is a technique that is well suited for examining the relationship between a categorical response variable and one or more categorical or continuous predictor variables. [Source:Google]*

I split the data into 80% and 20% ratio and will use Logical Regression for classification.

```
Call:  glm(formula = status ~ MDVP.Fhi.Hz. + MDVP.Jitter... + MDVP.RAP +
    MDVP.PPQ + Jitter.DDP + MDVP.APQ + Shimmer.DDA + RPDE + spread2 +
    PPE, family = "binomial", data = train[, -1])

Coefficients:
   (Intercept)    MDVP.Fhi.Hz.  MDVP.Jitter...        MDVP.RAP        MDVP.PPQ      J
itter.DDP
        3.7668         -0.6426         -3.6377          1.1025         -2.9152
4.8655
      MDVP.APQ     Shimmer.DDA            RPDE         spread2             PPE
        7.0288         -3.9273         -0.7536          1.5137          2.6336

Degrees of Freedom: 155 Total (i.e. Null);  145 Residual
Null Deviance:       173.2
Residual Deviance: 74.04   AIC: 96.04
```

Now I will choose the one the with lowest AIC.

So, my final model will be:

status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP + MDVP.Shimmer + RPDE + spread2 + D2 + PPE

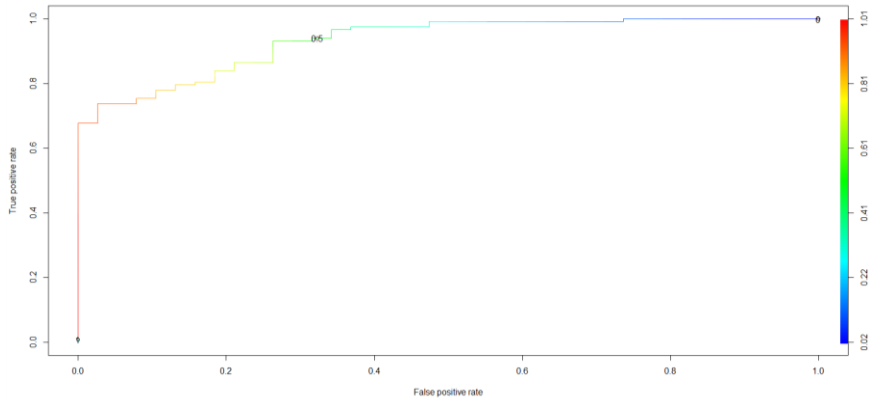
I will check my model's accuracy now. First, I will use my training data.

```
> pre <- as.numeric(predict(final.model,type="response")>0.45)
> confusionMatrix(table(pre,train$status))
Confusion Matrix and Statistics
pre   0   1
  0  25   7
  1  13 111


               Accuracy : 0.8718
                 95% CI : (0.809, 0.9199)
    No Information Rate : 0.7564
    P-Value [Acc > NIR] : 0.0002606

                  Kappa : 0.6324

 Mcnemar's Test P-Value : 0.2635525

            Sensitivity : 0.6579
            Specificity : 0.9407
         Pos Pred Value : 0.7813
         Neg Pred Value : 0.8952
             Prevalence : 0.2436
         Detection Rate : 0.1603
   Detection Prevalence : 0.2051
      Balanced Accuracy : 0.7993

       'Positive' Class : 0
```

I will test the same thing on a test data set.

```
> pre <- as.numeric(predict(final.model,newdata=test[,-1],type="response")>0.5)
> confusionMatrix(table(pre,test$status))
Confusion Matrix and Statistics


pre  0  1
  0  9  4
  1  1 25

               Accuracy : 0.8718
                 95% CI : (0.7257, 0.957)
    No Information Rate : 0.7436
    P-Value [Acc > NIR] : 0.04267

                  Kappa : 0.6939

 Mcnemar's Test P-Value : 0.37109

            Sensitivity : 0.9000
            Specificity : 0.8621
         Pos Pred Value : 0.6923
         Neg Pred Value : 0.9615
             Prevalence : 0.2564
         Detection Rate : 0.2308
   Detection Prevalence : 0.3333
      Balanced Accuracy : 0.8810
       'Positive' Class : 0
```

ROC Curve:



SVM: A Support Vector Machine (SVM) is a discriminative classifier formally defined by a sep arating hyperplane. (Linear)

Output:

```
Call:
svm(formula = status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP + MDVP.Shimmer +
    Shimmer.DDA + RPDE + spread2 + PPE, data = train[, -1])


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  77

 ( 47 30 )


Number of Classes:  2

Levels:
 0 1

> #Predict Output
> predicted=predict(fit.svm,test)
> table(predicted)
predicted
 0  1
 9 30
> table_mat <- table(test$status, predicted)
> accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 0.923076923076923"
```

Decision Tree Algorithm: Decision Tree Algorithm is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Output:

```
> #Decision tree algorithm
> set.seed(7)
> library(rpart)
> library(rpart.plot)
```

```
> fit.ds <- rpart(status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP +
+                   MDVP.Shimmer + Shimmer.DDA + RPDE + spread2 + PPE, data = train[, -1], method
= 'class')
> #rpart.plot(fit.ds)
> predict_unseen <-predict(fit.ds,test, type = 'class')
> table_mat <- table(test$status, predict_unseen)
> accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 0.846153846153846"
```

Naïve Bayes:

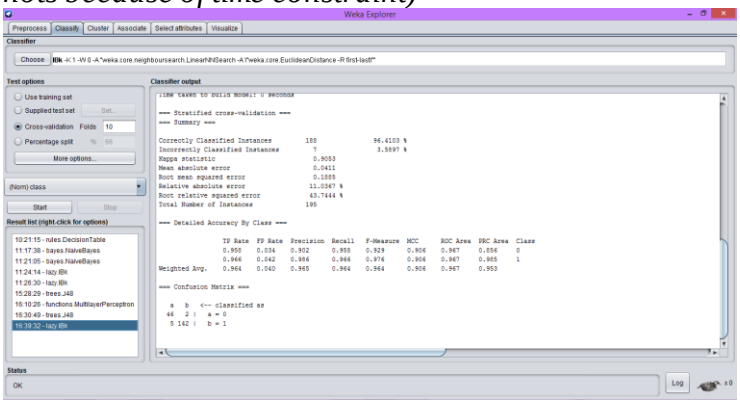
Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

Output:

```
> summary(fit.nb)
          Length Class  Mode
apriori   2      table  numeric
tables    8      -none- list
levels    2      -none- character
isnumeric 8      -none- logical
call      4      -none- call
>
>
> #Predict Output
> predicted= predict(fit.nb,test)
> table_mat <- table(test$status, predicted)
> accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 0.743589743589744"
```

KNN:

The k-nearest neighbor algorithm classifies objects on closest training examples in the feature space. It is also known as instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification.*(Provided Weka screens hots because of time constraint)*

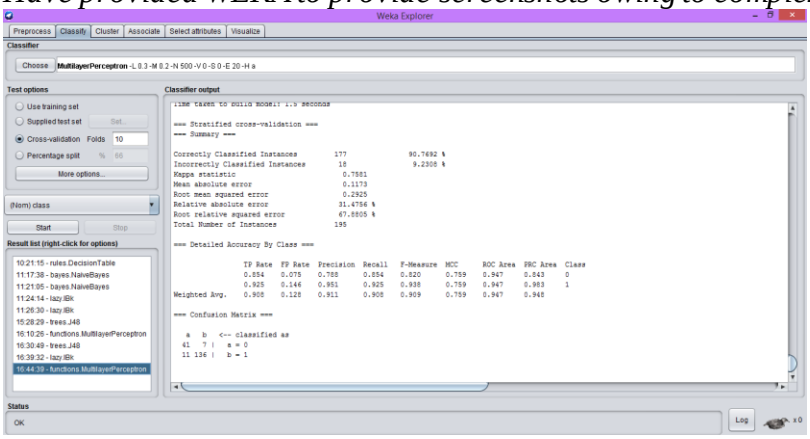


Multi-Layer Perceptron:

*A **multilayer perceptron** (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to refer to any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptron (with threshold activation)*

[Source: Wikipedia]

Have provided WEKA to provide screenshots owing to complexity.



5. Overall Analysis Results and Comparison(combined)

Parkinsons.data

	No.	Algorithm	Accuracy	Tool Used
	1	Naïve Bayes	74.35%	R
	2	Decision Tree Algorithm	84.61%	R
	3	Logical Regression	87.18%	R
	4	Multi Layer Perceptron	90.77%	Weka
	5	SVM	92.30%	R
	6	KNN Algorithm	96.41%	Weka

Initially I started this project using Weka and all the algorithms gave close to 92% Accuracy (Two of the screenshots are provided in support of that work). I assumed that overfitting is the major cause of that much level of accuracy hence as suggested by professor, I shifted the entire work and used R so that I can handle the data manually and remove the biasing of cross-folds.
The ranking of the algorithm is mentioned above and KNN gave 96.41% of accuracy (included Weka) Otherwise when I used manual test-train split SVM gave the maximum accuracy of 92.3%. Owing to overfittings my analysis shows SVM has better accuracy especially for the dataset – parkinsons.data.

Parkinson.updrs.data

Also as I have also explored the telemonitoring dataset following is my comparison.
I find Random Forest is giving maximum correlation between predicted and actual total UPDR and it have very minimal MAE. However I calculated SVN based on RMSE perfo rmance measuring technique which also give good RMSE for SVM(Linear). But if I have to compare between two I find **Random Forest is better since it has highest co-rel ation between actual and predicted UPDRS score(~99%).**

No.	Algorithm	Metric	Error	Tool Used	Comments
1	SVM(radial)	RMSE	3.53	R	NA
2	SVM(poly)	RMSE	7.87	R	NA
3	SVM(Linear)	RMSE	9.54	R	(Greater RMSE → better)
4	Random Forest Algorithm	MAE	0.58	R	(Lesser MAE → better)

6. Challenges:

- Feature selection was challenging and may need improved techniques in both the Parkinson dataset as for all the algorithm used none of the feature seems un-important.
- Jitter and Shimmer have high-correlation which caused multi collinearity problems.
- There is tendency of inherent biasing owing to UPDRS score variables (motor &total) for which I cannot use dimensionality reduction. However, much analysis needed for that and there may be possible solution.
- Since the data is not normally distributed much effort will be required for normalization which was excluded from this analysis
- Cross folds in Weka caused overfitting. Accuracy reduced used manual split.
- SVM ran for almost an hour owing to tuning method used and multiple repeation

7. Conclusion:

- Through Random Forest algorithm I came to know age, sex, test-time and DFA are the top-most important factor contributing Parkinson's disease and I find that is the best algorithm to analyze Parkinson's disease.
- UPDRS dataset is more reliable as it provided better accuracy score and tests are done over a larger span of time and have multiple attributes. So, any algorithm when used have higher reliability over the other dataset.

Citation

[1] A Tsanas, MA Little, PE McSharry, LO Ramig (2009)
'Accurate telemonitoring of Parkinson.s disease progression by non-invasive speech tests.

[2] A Survey of Machine Learning Based Approaches for Parkinson Disease Prediction. Shubham Bind1 , Arvind Kumar Tiwari2 , Anil Kumar Sahani3

[3] Feature Relevance Analysis and Classification of Parkinson Disease Tele-Monitoring Data Through Data Mining. Dr.R.Geetha Ramani. G. Sivagami. Shomona Gracia Jacob
[https://pdfs.semanticscholar.org/291e/c451967e81db4f6b2d08090f435990d24a24.pdf]

[4] Rpubs.com

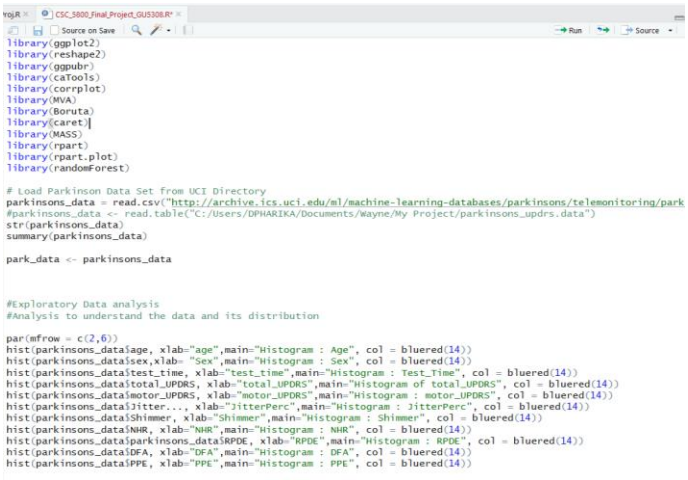
[5] Conf Proc IEEE Eng Med Biol Soc. 2016 Aug;2016:6389-6392. doi: 10.1109/EMBC.2016.7592190.
Support vector machine classification of Parkinson's disease and essential tremor subjects based on temporal fluctuation.
https://www.ncbi.nlm.nih.gov/pubmed/28269710

[6] Random Forest-Based Prediction of Parkinson's Disease Progression Using

Acoustic, ASR and Intelligibility Features. Alexander Zlotnik, Juan M. Montero, Ruben San-Segundo, Ascension Gallardo-Antol]
[7] https://machinelearningmastery.com/machine-learning-in-r-step-by-step/
[8] Wikipedia
[9] Youtube

Code:

(For handling parkinsons.updrs.data)



```
library(ggplot2)
library(reshape2)
library(ggpubr)
library(caTools)
library(corrplot)
library(MVA)
library(Boruta)
library(caret)
library(MASS)
library(rpart)
library(rpart.plot)
library(randomForest)

# Load Parkinson Data Set from UCI Directory
parkinsons_data = read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/park
#parkinsons_data <- read.table("C:/Users/DPHARIKA/Documents/Wayne/My Project/parkinsons_updrs.data")
str(parkinsons_data)
summary(parkinsons_data)

park_data <- parkinsons_data


#Exploratory Data analysis
#Analysis to understand the data and its distribution

par(mfrow = c(2,6))
hist(parkinsons_data$age, xlab="age",main="Histogram : Age", col = bluered(14))
hist(parkinsons_data$sex,xlab= "Sex",main="Histogram : Sex", col = bluered(14))
hist(parkinsons_data$test_time, xlab="test_time",main="Histogram : Test_Time", col = bluered(14))
hist(parkinsons_data$total_UPDRS, xlab="total_UPDRS",main="Histogram of total_UPDRS", col = bluered(14))
hist(parkinsons_data$motor_UPDRS, xlab="motor_UPDRS",main="Histogram : motor_UPDRS", col = bluered(14))
hist(parkinsons_data$Jitter..., xlab="JitterPerc",main="Histogram : JitterPerc", col = bluered(14))
hist(parkinsons_data$Shimmer, xlab="Shimmer",main="Histogram : Shimmer", col = bluered(14))
hist(parkinsons_data$NHR, xlab="NHR",main="Histogram : NHR", col = bluered(14))
hist(parkinsons_data$parkinsons_data$RPDE, xlab="RPDE",main="Histogram : RPDE", col = bluered(14))
hist(parkinsons_data$DFA, xlab="DFA",main="Histogram : DFA", col = bluered(14))
hist(parkinsons_data$PPE, xlab="PPE",main="Histogram : PPE", col = bluered(14))
```

```
#To check if the data is normally distributed:  Chi-Square plot:
------------------------------------------------------------------------
library(mvnormtest)
data <- parkinsons_data
cm <- colMeans(data)
S <- cov(data)
d <- apply(data, 1, function(data) t(data - cm) %*% solve(S) %*% (data - cm))

# Chi-Square plot:
plot(qchisq((1:nrow(data) - 1/2) / nrow(data), df = ncol(data)),
sort(d),
xlab = expression(paste(chi[22]^2, " Quantile")),
ylab = "Ordered distances")


#Data Cleaning:

#Anomaly detection and treatment:
na = colSums(is.na(parkinsons_data))
na


#### Collinear Distribution

library(corrplot)
corrplot(cor(parkinsons_data), type="full", method ="color",
         title = "Parkinson Correlation Distribution",
         mar=c(0,0,1,0), tl.cex= 0.8, outline= T, tl.col="blue")
#Outlier Detection & Removal:
#Scattered plot to look into data distribution

par(mfrow = c(1,3))
windows()
plot(jitter(total_UPDRS)~., parkinsons_data)


#To Draw boxplots for understanding the outliers of the data

boxplot(parkinsons_data)

#To Draw Bivariate Boxplot against total_UPDRS
  library(MVA)
  par(mfrow = c(1,3))
  bvbox(parkinsons_data[,6:7],     xlab = "total_UPDRS", ylab = "Jitter")
  bvbox(parkinsons_data[,c(6,12)], xlab = "total_UPDRS", ylab = "Shimmer")
  bvbox(parkinsons_data[,c(6,18)], xlab = "total_UPDRS", ylab = "NHR")
  bvbox(parkinsons_data[,c(6,20)], xlab = "total_UPDRS", ylab = "RPDE")
  bvbox(parkinsons_data[,c(6,21)], xlab = "total_UPDRS", ylab = "DFA")
  bvbox(parkinsons_data[,c(6,22)], xlab = "total_UPDRS", ylab = "PPE")

#Convex hull method

hull1 <- chull(parkinsons_data[,6:7])
parkhull <- match(lab <- rownames(parkinsons_data[hull1,]),rownames(parkinsons_data))
plot(parkinsons_data[,6:7], xlab = "total_UPDRS", ylab = "Jitter")
polygon(parkinsons_data$Jitter...[hull1]~parkinsons_data$total_UPDRS[hull1])
text(parkinsons_data[parkhull,6:7], labels = lab, pch=".", cex = 0.9)
```

```r
#Outlier Reduction

outlier <- parkinsons_data[-hull1,]
dim(outlier)
dim(parkinsons_data)

hull2 <- chull(outlier[,c(6,12)])
parkinsons_data <- outlier[-hull2,]

hull3 <- chull(parkinsons_data[,c(6,18)])
outlier <- parkinsons_data[-hull3,]

hull4 <- chull(outlier[,c(6,20)])
parkinsons_data <- outlier[-hull4,]

hull5 <- chull(parkinsons_data[,c(6,21)])
outlier <- parkinsons_data[-hull5,]

hull6 <- chull(outlier[,c(6,22)])
parkinsons_data <- outlier[-hull6,]

dim(parkinsons_data)
summary(parkinsons_data)

#FEATURE SELECTION:
# Preprocessing: Using Boruta to identify important Attributes

library(Boruta)
Borutatrain <- Boruta(age ~ ., data = parkinsons_data, doTrace = 2)
print(Borutatrain)
```

```r
# Plot Important Attributes " No unimpotant attribute detected"

plot(Borutatrain, xlab = "", xaxt = "n")
lz<-lapply(1:ncol(Borutatrain$ImpHistory),function(i)
  Borutatrain$ImpHistory[is.finite(Borutatrain$ImpHistory[,i]),i])
names(lz) <- colnames(Borutatrain$ImpHistory)
Labels <- sort(sapply(lz,median))
axis(side = 1,las=2,labels = names(Labels),
     at = 1:ncol(Borutatrain$ImpHistory), cex.axis = 0.7)


#Dimensionality Reduction: PCA
pca_parkinsons_data<-park_data
pca_parkinsons_data[["subject."]]= NULL
pca_parkinsons_data[["motor_UPDRS"]]=NULL
pca_parkinsons_data[["total_UPDRS"]]=NULL


pc=prcomp(pca_parkinsons_data,center = TRUE,scale. = TRUE)
plot(pc,type="l",main = "PCA:SCREE PLOT")
component1=pc$x[,1]
component2=pc$x[,2]
pc_data =cbind(component1,component2)
plot(component1,component2,main = "SCATTER PLOT AFTER PCA",col = "red")

#K-MEANS CLUSTERING
Parkinsons_kmeans = kmeans(pc_data,2)
plot(pc_data, col = Parkinsons_kmeans$cluster, main = " K-MEANS CLUSTERING WITH PCA DATA")
points(Parkinsons_kmeans$centers, col = c("yellow"), pch = 17, cex= 2)
rect.hclust(pk_den,k=2,border = "blue")
```

```r
#HIERARCHIAL CLUSTERING
set.seed(666)
library (cluster)
p = sample(length(pc_data),400)
pk_den = hclust(dist(p),method = "complete")
cut_clust=cutree(pk_den,k=2)
windows()
plot(pk_den,main = "DENDROGRAM WITH COMPLETE LINKAGE")
rect.hclust(pk_den,k=2,border = "blue")
```

```r
#CLASSIFICATION: RANDOM FOREST

# remove motor UPDRS to make the data single variate regression
p<-data.frame(parkinsons_data[,-5])
str(p)

#Training a model on the data
##Set up trainning and test data sets:
set.seed(350)
indx = sample(1:nrow(p), as.integer(0.9*nrow(p)))
indx[1:10]

p_train = p[indx,]
p_test = p[-indx,]

library(randomForest)
rf<- randomForest(total_UPDRS~ ., data =p_train )
rf

#Check importance of each predictor:

library(randomForest)
library(ggplot2)
importance(rf)

#Plot of Important Variables
varImpPlot(rf)


#Evaluating Model Performance
pred<-predict(rf,p_test,type='response')
head(pred)
summary(pred)
summary(p$total_UPDRS)

cor(pred,p_test$total_UPDRS)

MAE <- function(actual, predicted) {
  mean(abs(actual - predicted))
}
```

```
#To Improve Model Performance
library(randomForest)
rf1<- randomForest(total_UPDRS~ ., data =p_train,mtry=10)
rf1


importance(rf1)
varImpPlot(rf1)
pred1<-predict(rf1,p_test,type='response')
head(pred1)
head(p_test$total_UPDRS)
summary(pred1)
summary(p$total_UPDRS)
#Compare the correlation between predicted and actual total UPDRS.

cor(pred1,p_test$total_UPDRS)

#Mean absolute error between predicted and actual values:

MAE(pred1, p_test$total_UPDRS)
```

```
#CLASSIFICATION: SVM


library(e1071)
#linear
tune.out=tune(svm,total_UPDRS~.,data=p_train,kernel="linear",
              ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10),
                          epsilon=c(0.1,0.5,1)))
summary(tune.out)
bestmod=tune.out$best.model
summary(bestmod)
predict2 <- predict(bestmod,p_test, type= 'response')
svmRMSE2 <- sqrt(mean((predict2-p_test$total_UPDRS)^2))
svmRMSE2

plot(p_test$total_UPDRS,predict2, xlab = "Observed",ylab = "Predicted")

#radial
tune.out=tune(svm,total_UPDRS~.,data=p_train,kernel="radial",
              ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10),
                          gamma=c(0.1,0.3,0.4,0.5,1,2)))
summary(tune.out)
bestmod=tune.out$best.model
summary(bestmod)
predict2 <- predict(bestmod,p_test, type= 'response')
svmRMSE2 <- sqrt(mean((predict2-p_test$total_UPDRS)^2))
svmRMSE2

plot(p_test$total_UPDRS,predict2, xlab = "Observed",ylab = "Predicted")


#poly
tune.out=tune(svm,total_UPDRS~.,data=p_train,kernel="polynomial",
              ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10),
                          degree=c(2,3,4)))
summary(tune.out)
bestmod=tune.out$best.model
summary(bestmod)
predict2 <- predict(bestmod,p_test, type= 'response')
svmRMSE2 <- sqrt(mean((predict2-p_test$total_UPDRS)^2))
svmRMSE2
plot(p_test$total_UPDRS,predict2, xlab = "Observed",ylab = "Predicted")
```

For handling parkinsons.data:

(except for classification part,code is almost similar)

```
library(ggplot2)
library(reshape2)
library(ggpubr)
library(caTools)

park=read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data",
                sep = ",", header = TRUE, stringsAsFactors = FALSE)
head(park,6)
dim(park)
str(park)


park$status=as.factor(park$status)
summary(park)

#Anomaly detection and treatment:
na = colSums(is.na(park))
na

#Exploratory Data Analysis & Outlier
park.m=melt(park[,-1],id.vars = "status")
p <- ggplot(data = park.m, aes(x=variable, y=value)) + geom_boxplot(aes(fill=status))
p + facet_wrap( ~ variable, scales="free")

names(park)

#Outlier Removal
boxplot(park)
outlier_remove=function(x){
  quantiles <- quantile( x, c(.25, .75 ) )
  iqr=IQR(x)
  med=median(x)
  x[ x < quantiles[1] -iqr ] <- med
  x[ x > quantiles[2]+ iqr] <- med
  x
}


#Replacing the outliers of various variables :

park$MDVP.Fhi.Hz.= outlier_remove(park$MDVP.Fhi.Hz.)
park$MDVP.Jitter...= outlier_remove(park$MDVP.Jitter...)
park$MDVP.Jitter.Abs.= outlier_remove(park$MDVP.Jitter.Abs.)
park$MDVP.RAP= outlier_remove(park$MDVP.RAP)
park$MDVP.PPQ= outlier_remove(park$MDVP.PPQ)
park$Jitter.DDP= outlier_remove(park$Jitter.DDP)
park$NHR= outlier_remove(park$NHR)

#Feature Selection

# Preprocessing: Using Boruta to identify important Attributes

library(Boruta)
Borutatrain <- Boruta(MDVP.Fo.Hz. ~ ., data = park, doTrace = 2)
# Borutatrain <- Boruta(name ~ ., data = pdata, doTrace = 2)
print(Borutatrain)

# Plot Important Attributes " No unimpotant attribute detected"

plot(Borutatrain, xlab = "", xaxt = "n")
lz<-lapply(1:ncol(Borutatrain$ImpHistory),function(i)
  Borutatrain$ImpHistory[is.finite(Borutatrain$ImpHistory[,i]),i])
names(lz) <- colnames(Borutatrain$ImpHistory)
Labels <- sort(sapply(lz,median))
axis(side = 1,las=2,labels = names(Labels),
     at = 1:ncol(Borutatrain$ImpHistory), cex.axis = 0.7)


#Classification
#Logical Regression

park[,-c(1,18)]=scale(park[,-c(1,18)])
split=sample.split(park$status,SplitRatio = .80)
train=subset(park,split==T)
test=subset(park,split==F)
```

```r
library(caret)
library(MASS)
model.lg=glm(data=train[,-1],status~.,family = "binomial")
summary(model.lg)
step(model.lg,direction="backward")

final.model =glm(status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP +
                   MDVP.Shimmer + Shimmer.DDA + RPDE + spread2 + PPE, family = "binomial",
                 data = train[, -1])
summary(final.model)
str(final.model)
#We will check our model's accuracy now. First we will use our training data. Let's see:

pre <- as.numeric(predict(final.model,type="response")>0.45)
confusionMatrix(table(pre,train$status))

#We will test our model on the basis of test data set.
pre <- as.numeric(predict(final.model,newdata=test[,-1],type="response")>0.5)
confusionMatrix(table(pre,test$status))

#ROC

prob_train <- predict(final.model, type = "response")
library(ROCR)
pred <- prediction(prob_train, train$status)
perf <- performance(pred, measure="tpr", x.measure="fpr")
plot(perf, col=rainbow(10), colorize=T, print.cutoffs.at=seq(0,1,0.50))

perf_auc <- performance(pred, measure="auc")
auc <- perf_auc@y.values[[1]]
print(auc)


#SVM
set.seed(7)
library(e1071)
# Fitting model
fit.svm <-svm(status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP +
                MDVP.Shimmer + Shimmer.DDA + RPDE + spread2 + PPE, data = train[, -1])
summary(fit.svm)
#Predict Output
predicted=predict(fit.svm,test)
table(predicted)
table_mat <- table(test$status, predicted)
accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test', accuracy_Test))



#Decision tree algorithm
set.seed(7)
library(rpart)
library(rpart.plot)
fit.ds <- rpart(status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP +
                  MDVP.Shimmer + Shimmer.DDA + RPDE + spread2 + PPE, data = train[, -1], method = 'class')
#rpart.plot(fit.ds)
predict_unseen <-predict(fit.ds,test, type = 'class')
table_mat <- table(test$status, predict_unseen)
accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test', accuracy_Test))



#Naive Bayes
library(e1071 )
set.seed(7)
# Fitting model
fit.nb <-naiveBayes(status ~ MDVP.Fhi.Hz. + MDVP.Jitter.Abs. + MDVP.RAP +
                      MDVP.Shimmer + Shimmer.DDA + RPDE + spread2 + PPE, data = train[, -1])
summary(fit.nb)


#Predict Output
predicted= predict(fit.nb,test)
table_mat <- table(test$status, predicted)
accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test', accuracy_Test))
```