# ChooseYourOwn

Anindita Mukherjee

6/8/2020

#Introduction

#Goal of the Project The goal of this project is to create a model that can predict the class of orthopedic patients. We want to classify the patients as belonging to one of the two categories, “Normal” and “Abnormal”. #The dataset and variables If a patient has either “Disk Hernia” or “Spondylolisthesis”, the patient is classified as “Abnormal” otherwise “Normal” I have used the dataset “Biomechanical Features of Orthopedic Patients” file “column_2C_weka.csv” from Kaggle on the site :

https://www.kaggle.com/uciml/biomechanical-features-of-orthopedic-patients

Data Download:

```
library(tidyverse)

## -- Attaching packages ---------------------------------------------------
----- tidyverse 1.3.0 --

## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts -----------------------------------------------------------
tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(caret)

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

library(data.table)

##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose

library(readr)
library(rpart)
library(ggplot2)
library(gam)

## Loading required package: splines

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loaded gam 1.16.1

library(dslabs)
library(knitr)
library(kernlab)

##
## Attaching package: 'kernlab'

## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     alpha

library(tinytex)

##download data to be used
dl <- tempfile()
download.file("https://raw.githubusercontent.com/AndyMukherjee/BioMed_feature_of_orthopedic_patients/master/datasets_2374_3987_column_2C_weka.csv",dl)
Bio <- read_csv(dl)

## Parsed with column specification:
## cols(
##   pelvic_incidence = col_double(),
##   `pelvic_tilt numeric` = col_double(),
```

```
##   lumbar_lordosis_angle = col_double(),
##   sacral_slope = col_double(),
##   pelvic_radius = col_double(),
##   degree_spondylolisthesis = col_double(),
##   class = col_character()
## )
```

I have downloaded the above data and put it in the dataframe “Bio”.

```
## View the data : data Visualization
head(Bio)

## # A tibble: 6 x 7
##   pelvic_incidence `pelvic_tilt nu~ lumbar_lordosis~ sacral_slope pelvic_r
adius
##              <dbl>            <dbl>            <dbl>        <dbl>
<dbl>
## 1             63.0             22.6             39.6         40.5
98.7
## 2             39.1             10.1             25.0         29.0
114.
## 3             68.8             22.2             50.1         46.6
106.
## 4             69.3             24.7             44.3         44.6
102.
## 5             49.7              9.65            28.3         40.1
108.
## 6             40.3             13.9             25.1         26.3
130.
## # ... with 2 more variables: degree_spondylolisthesis <dbl>, class <chr>

summary(Bio)

##  pelvic_incidence pelvic_tilt numeric lumbar_lordosis_angle  sacral_slope
##  Min.   : 26.15   Min.   :-6.555      Min.   : 14.00        Min.   : 13.37
##  1st Qu.: 46.43   1st Qu.:10.667      1st Qu.: 37.00        1st Qu.: 33.35
##  Median : 58.69   Median :16.358      Median : 49.56        Median : 42.40
##  Mean   : 60.50   Mean   :17.543      Mean   : 51.93        Mean   : 42.95
##  3rd Qu.: 72.88   3rd Qu.:22.120      3rd Qu.: 63.00        3rd Qu.: 52.70
##  Max.   :129.83   Max.   :49.432      Max.   :125.74        Max.   :121.43
##  pelvic_radius    degree_spondylolisthesis    class
##  Min.   : 70.08   Min.   :-11.058          Length:310
##  1st Qu.:110.71   1st Qu.:  1.604          Class :character
##  Median :118.27   Median : 11.768          Mode  :character
##  Mean   :117.92   Mean   : 26.297
##  3rd Qu.:125.47   3rd Qu.: 41.287
##  Max.   :163.07   Max.   :418.543

nrow(Bio) # 310 rows

## [1] 310
```

```r
ncol(Bio) # 7 columns
```

```
## [1] 7
```

```r
class(Bio)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Bio has 310 rows and 7 coulmns. Each patient is represented in the data by six biomechanical attributes derived from the shape and orientation of the pelvis and lumbar spine.Each of these in each column. The coulmns are: pelvic_incidence, pelvic_tilt numeric, lumbar_lordosis_angle, sacral_slope, pelvic_radius and degree_spondylolisthesis. The last column is "class" that we are going to determine in our model. The class is either "Normal" or "Abnormal".

#Key steps: 1. Visualizing the data, to see if a single attribute is responsible for the class determination 2.Divding the data into Train and Test sets to train and test the models 3.Determine the accuracy of each model 4.Compare the models 5.Get the best approach

#Method/Analysis

#Data visualization: Visualizing the data to determine if a single attribute is responsible to determine the class.

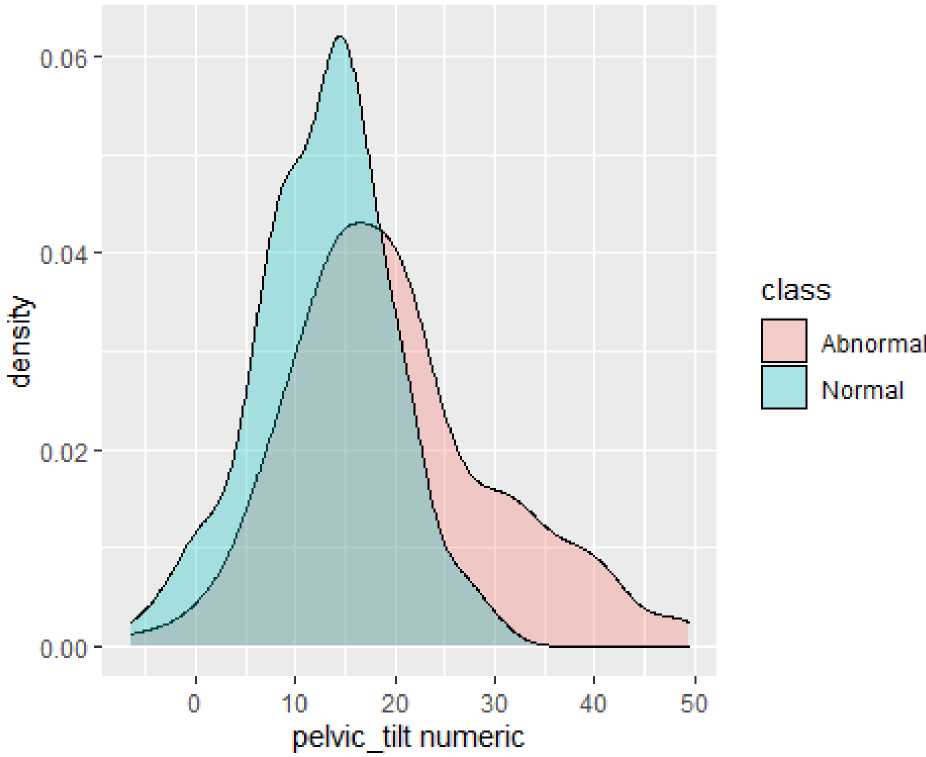
Checking the relationship of Class with pelvic_incidence

```r
##Check the relationship of Class with pelvic_incidence


Bio %>% ggplot(aes(`pelvic_incidence`, fill = class))+geom_density(alpha = 0.
3)
```


```
##The overlaping densities tell us that pelvic_incidence donot independently
determine the class
```

Check the relationship of Class with pelvic_tilt numeric:

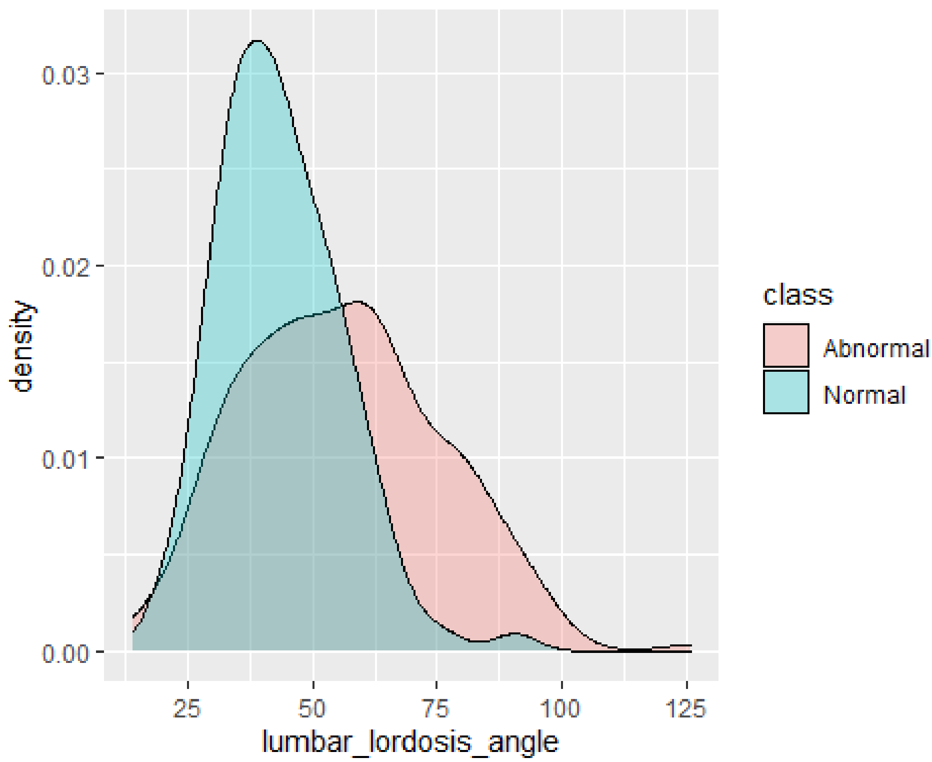
```
##Check the relationship of Class with pelvic_tilt numeric

Bio %>% ggplot(aes(`pelvic_tilt numeric`, fill = class))+geom_density(alpha =
0.3)
```




```
##The overlaping densities tell us that pelvic_tilt numeric donot independent
ly determine the class
```

Check the relationship of Class with lumbar_lordosis_angle

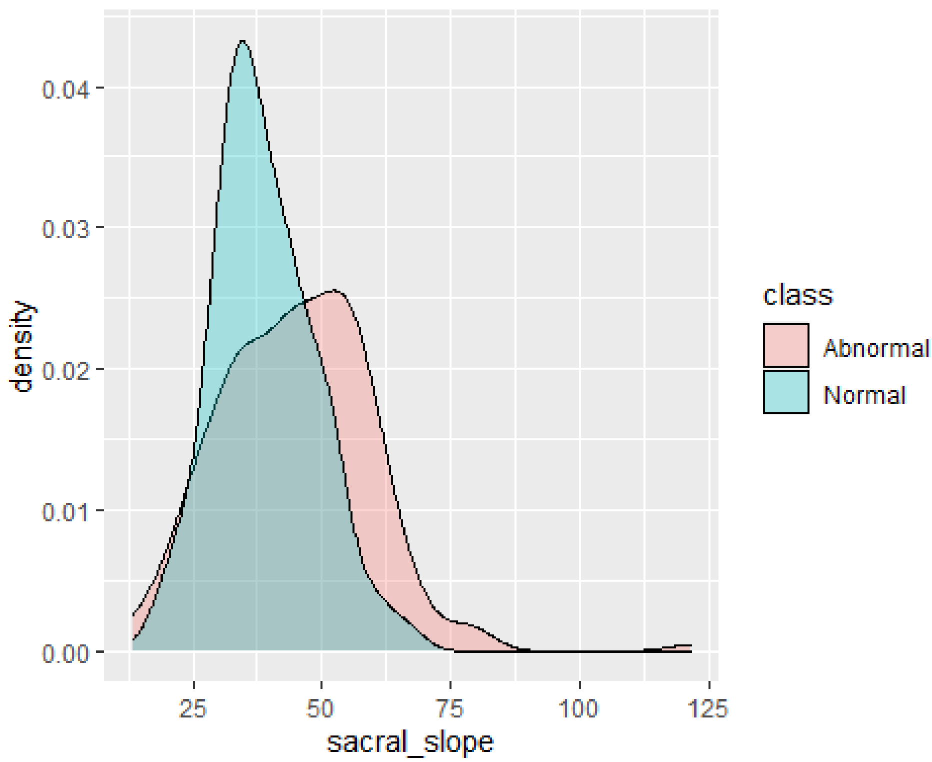
```
##Check the relationship of Class with lumbar_lordosis_angle

Bio %>% ggplot(aes(`lumbar_lordosis_angle`, fill = class))+geom_density(alpha
= 0.3)
```




```
##The overlaping densities tell us that lumbar_lordosis_angle donot independe
ntly determine the class
```

Check the relationship of Class with sacral_slope

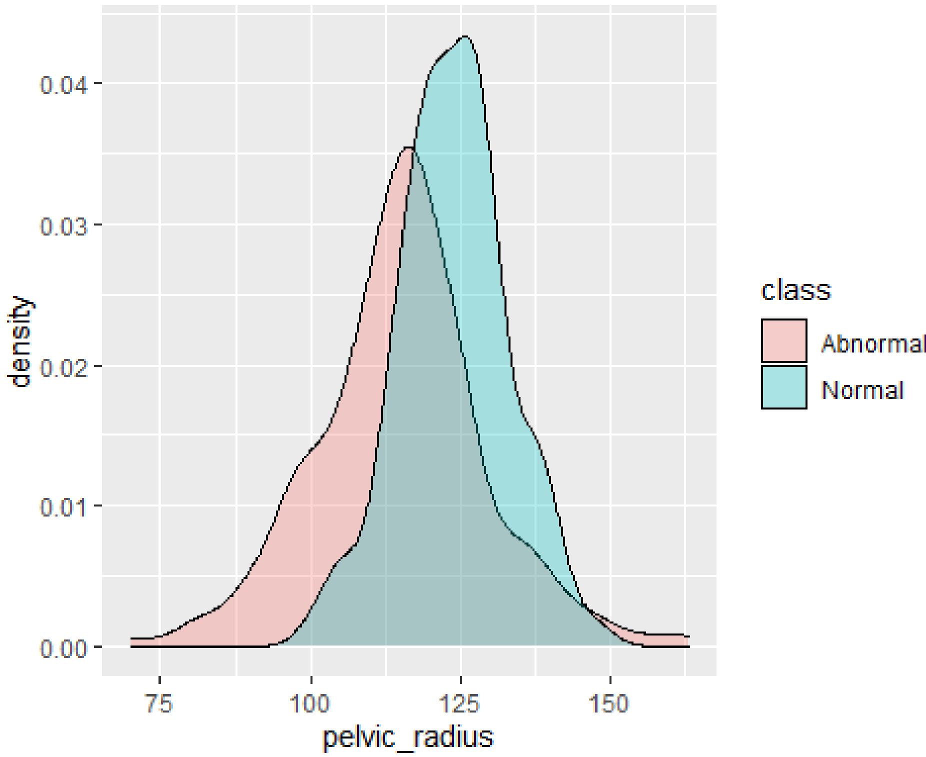
```
##Check the relationship of Class with sacral_slope

Bio %>% ggplot(aes(`sacral_slope`, fill = class))+geom_density(alpha = 0.3)
```




```
##The overlaping densities tell us that sacral_slope donot independently dete
rmine the class
```

Check the relationship of Class with pelvic_radius

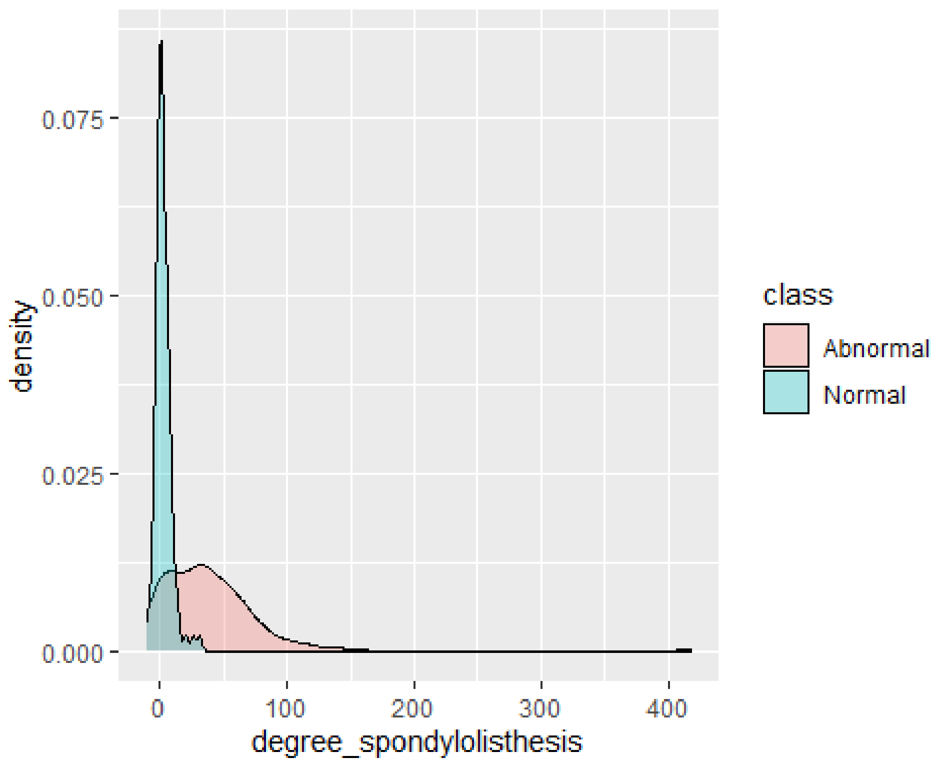
```
##Check the relationship of Class with pelvic_radius


Bio %>% ggplot(aes(`pelvic_radius`, fill = class))+geom_density(alpha = 0.3)
```




```
##The overlaping densities tell us that pelvic_radius donot independently det
ermine the class
```

Check the relationship of Class with degree_spondylolisthesis

```
##Check the relationship of Class with degree_spondylolisthesis

Bio %>% ggplot(aes(`degree_spondylolisthesis`, fill = class))+geom_density(al
pha = 0.3)
```




```
##The overlaping densities tell us that degree_spondylolisthesis donot indepe
ndently determine the class
```

The overlapping densities in all the above plots suggest that none of the attributes are individually responsible for determining the class.

#Data Cleansing We will now divide the Bio data into “train” and “test” sets, as 80 % and 20% respectively.

```
###################################################
#Training the models:

#Dividing the dataset of "Bio" into train and test sets

test_index <- createDataPartition(y = Bio$class, times = 1, p = 0.2, list = F
ALSE )
train <- Bio[-test_index,] #248 rows
test <- Bio[test_index,] #62 rows

## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

##creating x and y( subsets are x)
train_subset <- subset(train, select = -class)
test_subset <- subset(test, select = -class)
```

```r
train_y <- train$class
test_y <- test$class
```

We have further divided the Train and Test sets into train_subset as the x and train_y as the y for train set and test_subset as the x and test_y as the y for test, inorder to use them as “y ~ x” in some of the methods

#Modeling 1. Logistic Regression: Logistic regression is the method of fitting a regression curve y=f(x), where y is a categorical Value(NOrmal and Abnormal in this case), and x is a given set of predictors.

```r
##Using Logistic Regression to fit a model, to find the combined effect of al
l the columns
##together on the class, and find the accuracy of this.

  fit <- train(train_subset, train_y, method = "glm")

  fit$results

##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.8455197 0.6454607  0.0336843 0.07122199

## Accuracy of the logistic regression model is : 0.850605
```

Accuracy of the logistic regression model is : 0.850605

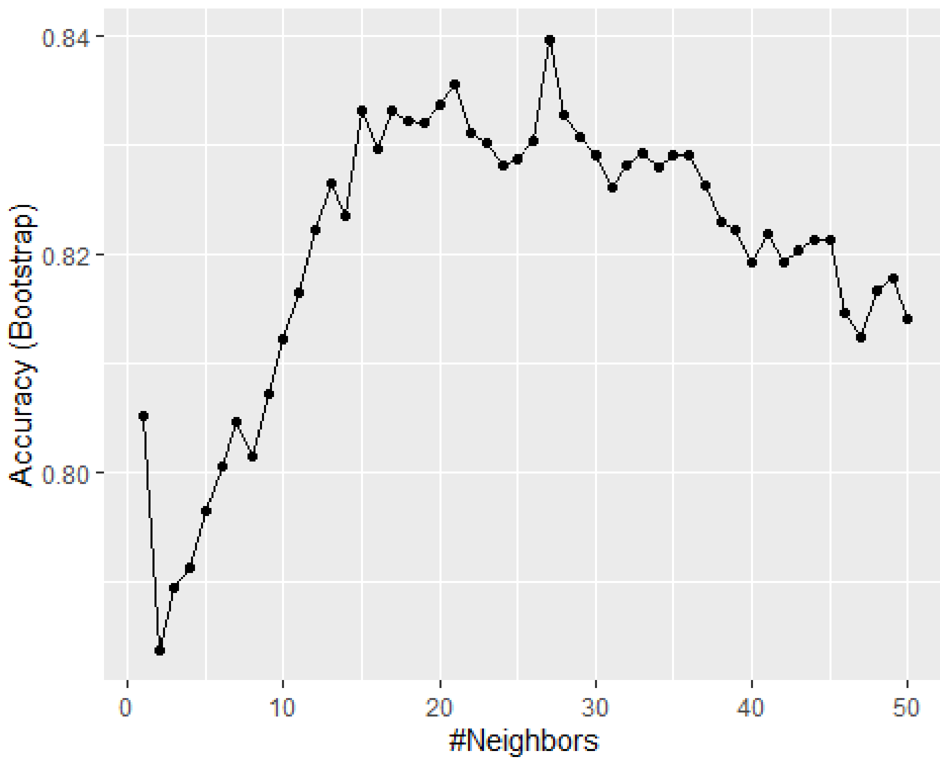
2.LDA and QDA: Linear Discriminant Analysis or LDA is similar to Logistic regression or glm method where y depends of the values of a set of predictors that is x and the distribution of x is normal. Quadratic Discriminant Analysis or QDA is similar to LDA , but covarience matrix is not common.

```r
##Using the LDA and QDA methods:

##LDA
fit_lda <- train(class ~ ., method = "lda", data = train)

fit_lda$results["Accuracy"] # 0.8408057

##    Accuracy
## 1 0.8189324

##QDA

# fit_qda <- train(class ~ ., method = "qda", data = train)
# fit_qda$results#["Accuracy"] #no results
## need to comment this out, else the code gives error
```

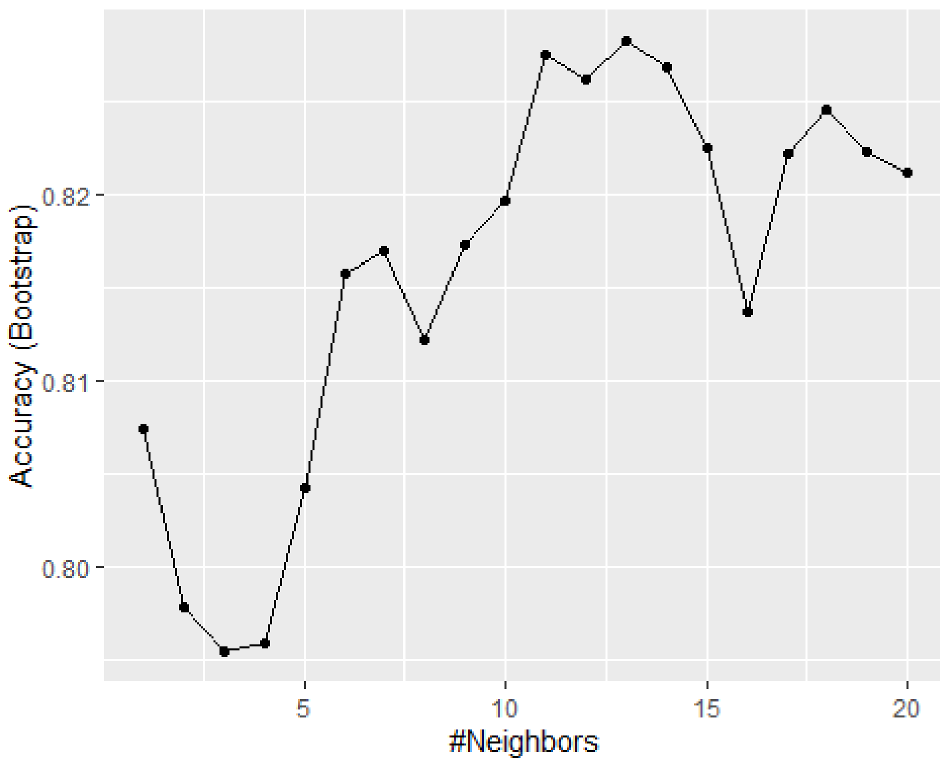
In the above models only LDA gave results and the Accuracy of the model is 0.8408057

3.KNN K Nearest Neighbor is a Parametric algorithm. Here we use a K number that is the tuning parameter to determine the number of nearest neighbors that gives the best Accuracy results.

```r
##Using the KNN method:
fit <- train(train_subset, train_y, method = "knn",tuneGrid = data.frame(k =
seq(1,50,1)))

ggplot(fit)
```




```r
##Looking at the graph since there is a steady decrease in accuracy from 20,

##I will narrow the k value from 1 to 20, to have a clearer view

fit_knn <- train(train_subset, train_y, method = "knn",tuneGrid = data.frame(
k = seq(1,20,1)))

ggplot(fit_knn)
```

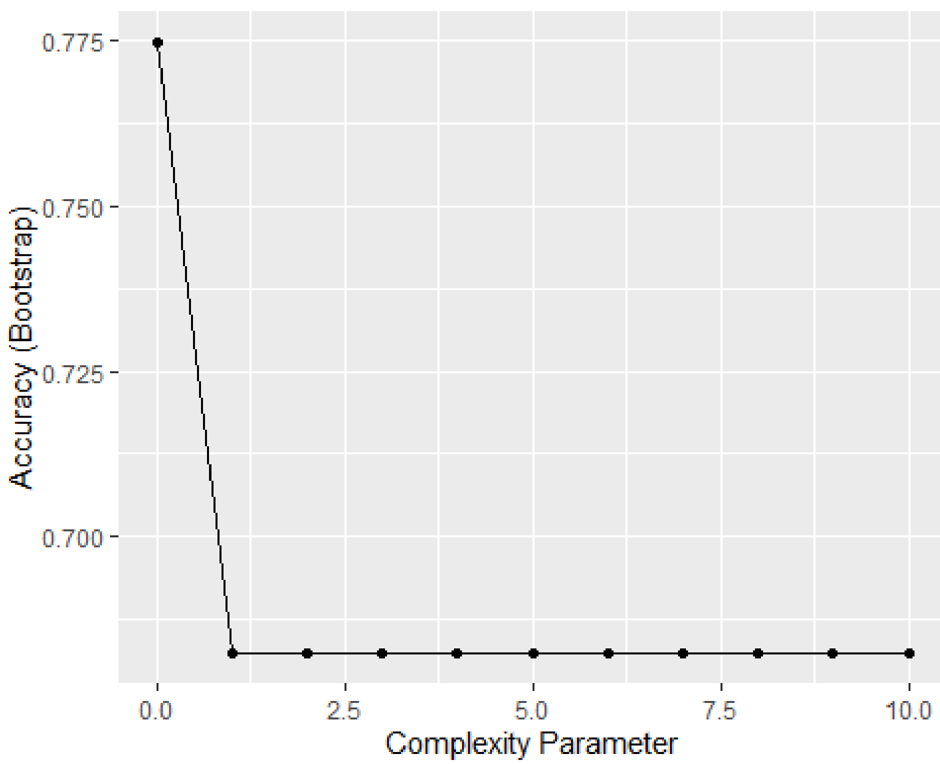



```
fit_knn$results

##     k  Accuracy     Kappa AccuracySD    KappaSD
## 1   1 0.8073670 0.5635852 0.03952289 0.08670518
## 2   2 0.7978030 0.5396731 0.04222529 0.09701320
## 3   3 0.7954735 0.5344783 0.04744142 0.10054160
## 4   4 0.7959066 0.5351543 0.04046741 0.08730172
## 5   5 0.8042218 0.5510111 0.03962272 0.08590465
## 6   6 0.8157515 0.5784875 0.03801158 0.08528277
## 7   7 0.8169467 0.5799379 0.03794379 0.08334547
## 8   8 0.8122045 0.5696033 0.04460732 0.09389319
## 9   9 0.8172378 0.5824070 0.04118363 0.08649974
## 10 10 0.8196553 0.5890771 0.03562705 0.07687608
## 11 11 0.8274505 0.6098315 0.03190611 0.06725981
## 12 12 0.8261751 0.6065296 0.03699680 0.07831534
## 13 13 0.8282213 0.6110438 0.03684085 0.07924945
## 14 14 0.8267664 0.6061294 0.03814322 0.08006176
## 15 15 0.8224560 0.5975303 0.04452781 0.09375730
## 16 16 0.8136315 0.5766370 0.03883120 0.08096832
## 17 17 0.8220921 0.6003201 0.04360192 0.08622723
## 18 18 0.8245302 0.6066494 0.04229025 0.08174049
## 19 19 0.8221988 0.6004802 0.04187214 0.08445952
## 20 20 0.8212068 0.5997291 0.03697478 0.07257044

## so K value of 5 has the heighest accuracy : 0.8333217
## but this is still less than the glm method
```

The Accuracy of the above model is 0.8333217

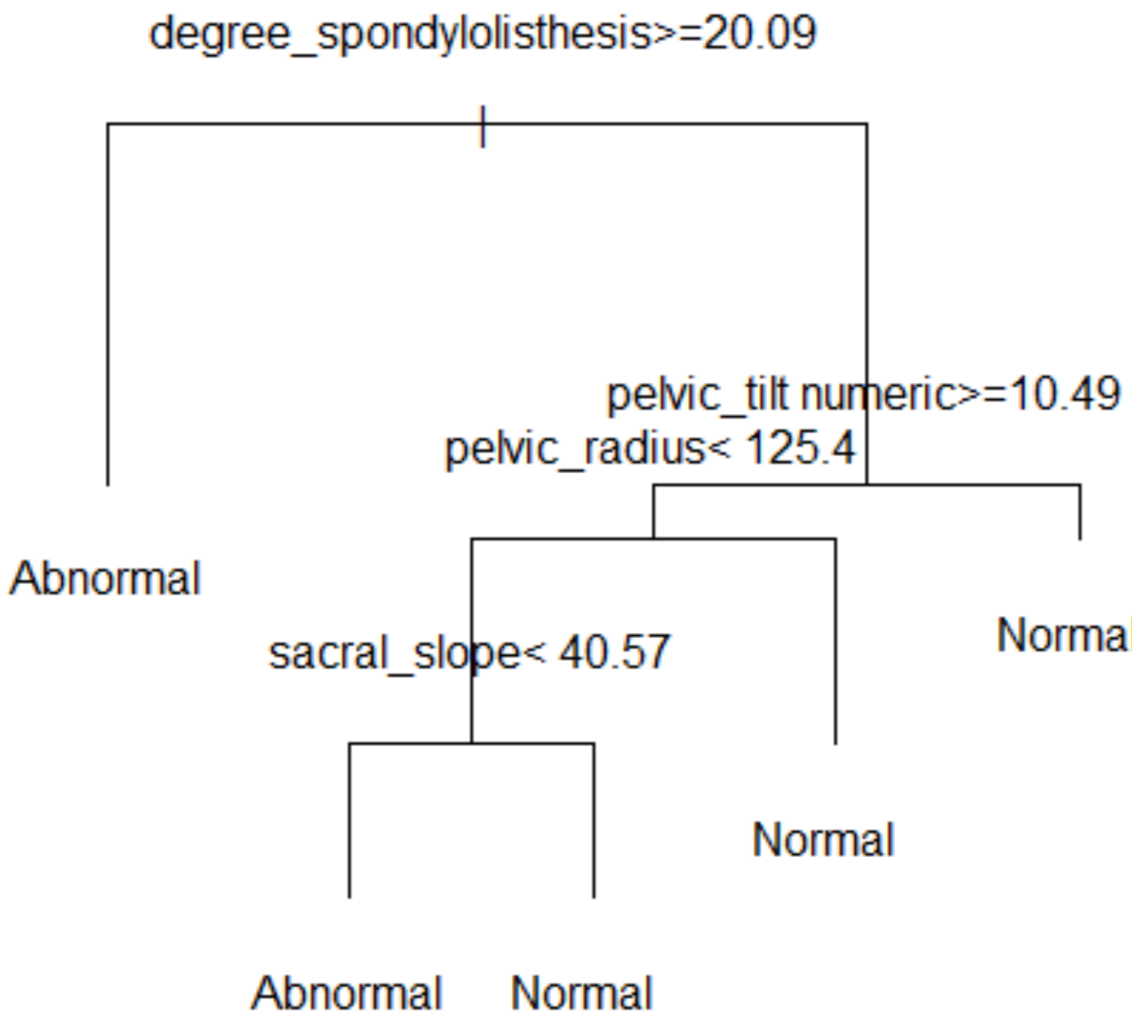
4.Rpart Recurssive Partitioning helps us explore thestructure of a dataset in a visual discision tree outcome. This is a tree based model.

```
##Rpart
fit_rpart <- train(train_subset, train_y, method = "rpart",
             tuneGrid = data.frame(cp = seq(0, 10, 1)))

ggplot(fit_rpart)
```




```
confusionMatrix(fit_rpart)

## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal     56.8   11.0
##   Normal       11.5   20.7
##
##  Accuracy (average) : 0.7748

## Accuracy is : 0.7997
```

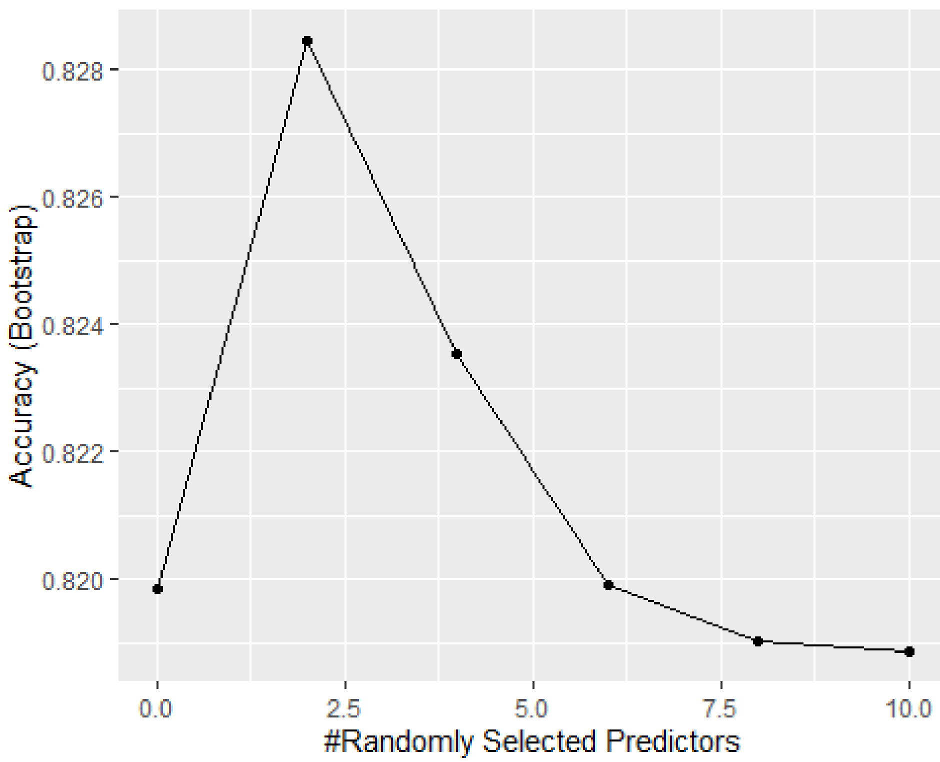
```
## Ploting the final Model
plot(fit_rpart$finalModel, margin = 0.1)
text(fit_rpart$finalModel)
```




The accuracy of the above model is 0.7997

5.Random Forest The random forest of the RF model is also a Tree bases model. This model reduces instability by averaging multiple dicision trees. It is a forest of trees constructed with randomness

```
##Using random forest method :
fit_rf <- train(train_subset, train_y, method = "rf",
                nodesize = 1,
                tuneGrid = data.frame(mtry = seq(0, 10, 2)))

ggplot(fit_rf)
```




```
confusionMatrix(fit_rf)  #Accuracy=  0.8391

## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##             Reference
## Prediction Abnormal Normal
##   Abnormal     60.1    9.4
##   Normal        7.7   22.8
##
##  Accuracy (average) : 0.8285

fit_rf$bestTune #mtry = 2

##   mtry
## 2    2
```

The Accuracy is 0.8391

6.Multinorm Multinorm is used to calculate a multivariate normal distribution

```
##Multinorm Method:
fit_multi <- train(class ~ . , method = "multinom", data = train)

## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 72.289497
```

```
## iter  20 value 71.168986
## final  value 71.162628
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.279091
## iter  20 value 78.444166
## final  value 78.444165
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 72.316004
## iter  20 value 71.194816
## final  value 71.188610
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.469851
## iter  20 value 65.028766
## final  value 65.007908
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.159104
## iter  20 value 73.090583
## iter  20 value 73.090582
## final  value 73.090569
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.499012
## iter  20 value 65.056392
## final  value 65.035971
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.484223
## iter  20 value 64.107770
## final  value 64.025908
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 74.945283
## iter  20 value 70.697870
## iter  20 value 70.697870
## final  value 70.697864
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.506724
```

```
## iter  20 value 64.128355
## final  value 64.048203
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.880315
## iter  20 value 66.322371
## final  value 66.317671
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.248769
## final  value 73.762257
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.907562
## iter  20 value 66.351017
## final  value 66.346301
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 77.580089
## iter  20 value 75.932150
## final  value 75.928776
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.052121
## iter  20 value 81.957874
## iter  20 value 81.957874
## final  value 81.957870
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 77.598349
## iter  20 value 75.948761
## final  value 75.945465
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.740532
## iter  20 value 69.395171
## final  value 69.347868
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.475924
## iter  20 value 80.934209
## iter  20 value 80.934209
```

```
## iter  20 value 80.934209
## final  value 80.934209
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.776838
## iter  20 value 69.437773
## final  value 69.391137
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 70.168532
## iter  20 value 69.149217
## final  value 69.146633
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.114217
## iter  20 value 76.397709
## iter  20 value 76.397709
## final  value 76.397696
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 70.193605
## iter  20 value 69.171565
## final  value 69.169043
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.364510
## iter  20 value 77.595575
## final  value 77.592400
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 90.680960
## iter  20 value 88.752902
## iter  20 value 88.752901
## iter  20 value 88.752901
## final  value 88.752901
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.399956
## iter  20 value 77.634867
## final  value 77.631696
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
```

```
## iter  10 value 72.386789
## iter  20 value 71.582792
## final  value 71.573332
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.436065
## iter  20 value 76.976616
## iter  20 value 76.976615
## iter  20 value 76.976615
## final  value 76.976615
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 72.404813
## iter  20 value 71.598528
## final  value 71.589218
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.045185
## iter  20 value 66.202401
## final  value 66.197564
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.963530
## iter  20 value 77.790352
## iter  20 value 77.790352
## iter  20 value 77.790351
## final  value 77.790351
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.086129
## iter  20 value 66.250246
## final  value 66.245294
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 60.518368
## iter  20 value 58.259600
## final  value 58.131088
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 69.056911
## iter  20 value 64.576793
## final  value 64.575864
## converged
```

```
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 60.546999
## iter  20 value 58.291269
## final  value 58.161757
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 65.440026
## iter  20 value 63.479997
## final  value 63.478659
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 76.944201
## iter  20 value 73.677077
## iter  20 value 73.677077
## final  value 73.677063
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 65.476334
## iter  20 value 63.538068
## final  value 63.537132
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.285923
## iter  20 value 80.208633
## final  value 80.207589
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 83.612240
## final  value 82.262642
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.293802
## iter  20 value 80.214076
## final  value 80.213044
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.803524
## iter  20 value 75.770010
## final  value 75.769935
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
```

```
## iter  10 value 84.486570
## iter  20 value 83.722536
## iter  20 value 83.722536
## final  value 83.722532
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.829756
## iter  20 value 75.797381
## final  value 75.797322
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.191559
## iter  20 value 69.907631
## final  value 69.865526
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.588112
## final  value 76.118841
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.214238
## iter  20 value 69.927408
## final  value 69.885662
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.270749
## iter  20 value 79.097032
## final  value 79.094766
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.774135
## final  value 82.221257
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.282958
## iter  20 value 79.105279
## final  value 79.103038
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.825582
## iter  20 value 62.432785
## final  value 62.247503
```

```
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 70.538192
## iter  20 value 66.276991
## final  value 66.276709
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.935996
## iter  20 value 62.415907
## final  value 62.260502
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.594214
## iter  20 value 78.481922
## final  value 78.479529
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.428405
## final  value 82.112966
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.608080
## iter  20 value 78.493062
## final  value 78.490726
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.295644
## iter  20 value 80.015143
## final  value 80.014603
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 87.783624
## final  value 86.950475
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.315624
## iter  20 value 80.036237
## iter  20 value 80.036236
## final  value 80.035962
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
```

```
## iter  10 value 78.409383
## iter  20 value 77.326705
## final  value 77.324651
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 86.847405
## iter  20 value 84.229452
## iter  20 value 84.229452
## iter  20 value 84.229451
## final  value 84.229451
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.431193
## iter  20 value 77.345591
## final  value 77.343582
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 76.298872
## iter  20 value 74.767342
## final  value 74.761537
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 83.328130
## iter  20 value 80.951033
## iter  20 value 80.951032
## iter  20 value 80.951032
## final  value 80.951032
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 76.318116
## iter  20 value 74.784356
## final  value 74.778629
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 61.364025
## iter  20 value 59.425988
## final  value 59.406794
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.659858
## iter  20 value 74.840404
## iter  20 value 74.840404
## iter  20 value 74.840403
```

```
## final  value 74.840403
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 61.419258
## iter  20 value 59.499026
## final  value 59.478236
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 65.793884
## iter  20 value 64.942304
## final  value 64.941712
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.935229
## iter  20 value 73.707708
## iter  20 value 73.707708
## iter  20 value 73.707707
## final  value 73.707707
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 65.823878
## iter  20 value 64.978310
## final  value 64.977554
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.764189
## iter  20 value 69.488916
## final  value 69.481198
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 74.700472
## iter  20 value 71.774458
## iter  20 value 71.774458
## iter  20 value 71.774457
## final  value 71.774457
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.776018
## iter  20 value 69.495613
## final  value 69.488010
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
```

```
## iter  10 value 57.689830
## iter  20 value 56.369296
## final  value 56.353259
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 67.137135
## iter  20 value 64.627154
## final  value 64.627073
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 57.727665
## iter  20 value 56.398069
## final  value 56.383863
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 76.620202
## iter  20 value 75.502375
## final  value 75.499120
## converged
```

```
confusionMatrix(fit_multi) ## Accuracy = 0.847
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal     58.3    7.1
##   Normal        9.1   25.5
##
##  Accuracy (average) : 0.8383
```

The Accuracy of this model is 0.847

7.SVM Support Vector Machine is used for both regression and classification

```
##SVM Linear model:
 fit_svm <- train(class ~ . , method = "svmLinear", data = train)
 confusionMatrix(fit_svm) ## Accuracy = 0.8408
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal     59.0    8.0
```

```
##   Normal        8.0   25.1
##
##  Accuracy (average) : 0.8403
```

The Accuracy of this method is 0.8408

8.All together Compairing all the methods together, to get which method gives the best accuracy

```
## Joining all the models together to get the accuracy of each model to compa
re :

model <- c("glm", "lda", "knn", "rf", "svmLinear",  "multinom")

fits <- lapply(model, function(model){
  #print(model)
  train(class ~ . , method = model, data = train)
})
```

```
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.467823
## iter  20 value 77.554469
## final  value 77.548135
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 85.805573
## iter  20 value 83.863120
## iter  20 value 83.863120
## final  value 83.863113
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 78.488144
## iter  20 value 77.575026
## final  value 77.568798
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.759816
## iter  20 value 65.108070
## final  value 65.065588
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 74.848669
## iter  20 value 71.997594
## iter  20 value 71.997594
## iter  20 value 71.997593
```

```
## final  value 71.997593
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.783512
## iter  20 value 65.132915
## final  value 65.090674
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.552401
## iter  20 value 80.466268
## final  value 80.465742
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 86.864135
## final  value 86.630646
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.571594
## iter  20 value 80.486834
## iter  20 value 80.486834
## final  value 80.486775
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.104783
## iter  20 value 78.865250
## final  value 78.864549
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.706098
## final  value 83.639376
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.121569
## iter  20 value 78.881462
## final  value 78.880775
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.797086
## iter  20 value 69.459370
## final  value 69.426510
## converged
## # weights:  8 (7 variable)
```

```
## initial  value 171.900501
## iter  10 value 79.739378
## iter  20 value 76.838176
## iter  20 value 76.838176
## iter  20 value 76.838176
## final  value 76.838176
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.819991
## iter  20 value 69.481528
## final  value 69.449028
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 85.331014
## iter  20 value 82.338206
## final  value 82.334305
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 89.310865
## iter  20 value 85.819313
## iter  20 value 85.819313
## final  value 85.819304
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 85.342832
## iter  20 value 82.347322
## final  value 82.343477
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 69.236588
## iter  20 value 67.604794
## iter  20 value 67.604794
## iter  20 value 67.604794
## final  value 67.604794
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 82.423385
## iter  20 value 80.757583
## iter  20 value 80.757583
## iter  20 value 80.757583
## final  value 80.757583
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
```

```
## iter  10 value 69.276595
## iter  20 value 67.662611
## final  value 67.662485
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 87.172250
## iter  20 value 87.081900
## iter  20 value 87.081899
## final  value 87.081847
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 92.702957
## final  value 92.048131
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 87.186126
## iter  20 value 87.097400
## iter  20 value 87.097399
## final  value 87.097349
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.101078
## iter  20 value 73.945308
## final  value 73.934586
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.621393
## iter  20 value 81.963595
## final  value 81.963590
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.129332
## iter  20 value 73.973832
## final  value 73.963285
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.624546
## iter  20 value 78.088806
## iter  20 value 78.088806
## final  value 78.088689
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
```

```
## iter  10 value 85.959744
## iter  20 value 83.227678
## iter  20 value 83.227678
## final  value 83.227671
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.640576
## iter  20 value 78.101789
## iter  20 value 78.101788
## final  value 78.101747
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.868962
## iter  20 value 79.659630
## final  value 79.658932
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 89.765832
## final  value 88.799299
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 79.897839
## iter  20 value 79.692422
## final  value 79.691712
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.330239
## iter  20 value 72.795609
## final  value 72.788402
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.389087
## final  value 77.699509
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.346698
## iter  20 value 72.807573
## final  value 72.800508
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.060596
## iter  20 value 79.314112
```

```
## iter  20 value 79.314112
## iter  20 value 79.314112
## final  value 79.314112
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 91.287286
## final  value 90.098173
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.092192
## final  value 79.353083
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 56.191585
## iter  20 value 55.899744
## iter  20 value 55.899744
## final  value 55.899708
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.099742
## iter  20 value 68.871237
## iter  20 value 68.871237
## iter  20 value 68.871236
## final  value 68.871236
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 56.240089
## iter  20 value 55.953474
## final  value 55.953303
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 76.990750
## iter  20 value 76.596525
## final  value 76.595467
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 87.145264
## final  value 86.561996
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 77.022838
## iter  20 value 76.636462
```

```
## final  value 76.635418
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.970175
## iter  20 value 81.908853
## final  value 81.908688
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 89.710408
## final  value 89.322937
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 81.989734
## iter  20 value 81.929802
## final  value 81.929642
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 61.407635
## iter  20 value 59.785150
## final  value 59.751252
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 69.824894
## iter  20 value 68.573239
## final  value 68.573077
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 61.436013
## iter  20 value 59.815460
## final  value 59.781983
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 77.309374
## iter  20 value 72.453425
## final  value 72.361354
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 84.845851
## iter  20 value 78.968616
## final  value 78.968220
## converged
## # weights:  8 (7 variable)
```

```
## initial  value 171.900501
## iter  10 value 77.332456
## iter  20 value 72.471681
## final  value 72.380764
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.571397
## iter  20 value 80.357601
## iter  20 value 80.357601
## iter  30 value 80.357477
## iter  30 value 80.357477
## iter  30 value 80.357476
## final  value 80.357476
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 86.129234
## final  value 85.422015
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 80.585934
## iter  20 value 80.373202
## final  value 80.372793
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.446197
## iter  20 value 66.153013
## final  value 66.152871
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 73.273115
## final  value 72.913934
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.467010
## iter  20 value 66.171888
## final  value 66.171743
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 70.308700
## iter  20 value 68.585173
## final  value 68.534232
## converged
## # weights:  8 (7 variable)
```

```
## initial  value 171.900501
## iter  10 value 76.961839
## iter  20 value 73.853393
## final  value 73.852831
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 70.329153
## iter  20 value 68.605004
## final  value 68.554403
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 64.143640
## iter  20 value 61.497961
## final  value 61.477521
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 71.640717
## final  value 69.284968
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 64.168522
## iter  20 value 61.523970
## final  value 61.503978
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.689445
## iter  20 value 65.399276
## final  value 65.386134
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 75.058072
## iter  20 value 72.976894
## iter  20 value 72.976894
## final  value 72.976888
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 66.713129
## iter  20 value 65.423588
## final  value 65.410758
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 82.315474
```

```
## iter  20 value 81.674798
## final  value 81.674076
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 88.049932
## final  value 87.029325
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 82.330223
## iter  20 value 81.688527
## final  value 81.687815
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 83.805133
## iter  20 value 83.530280
## final  value 83.529841
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 90.728105
## final  value 89.799733
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 83.824946
## iter  20 value 83.551075
## final  value 83.550602
## converged
## # weights:  8 (7 variable)
## initial  value 171.900501
## iter  10 value 76.620202
## iter  20 value 75.502375
## final  value 75.499120
## converged

pred <- sapply(fits, function(x){
  predict(x, newdata = test)
})

## get the Accuracy for each method:

colMeans(pred == test$class)

## [1] 0.8870968 0.8548387 0.9032258 0.8709677 0.8870968 0.9032258

##0.8709677 0.8548387 0.8548387 0.8387097 0.8709677 0.8870968
##According to the above result the best accuracy is from multinom model
```

The above results suggests that the best accuracy is in the multinorm model : 0.8870968

9.Ensemble This method takes the average of all the methods

```
##Ensemble Method:
En <- rowMeans(pred == "Normal")
y_hat <- ifelse(En > 0.5, "Normal", "Abnormal")
mean(y_hat == test$class)

## [1] 0.9032258

##Accuracy: 0.8709677
```

The accuracy is 0.8709677

#Results Comparing all models together we see that the best accuracy is from the Multinorm Model:0.8870968. Ensembling all models together provided the accuracy of 0.8709677 that is the same as glm and svmLinear methods.

#Conclusion I have used 9 methods to determine the best suited model that can predict the class of a patient to be either Normal or Abnormal. Out of these The multinorm method gave the best results and the ensemble, glm and SVMLinear gave the average results. The Potential impact of this research is to get the best model that in this case is the multinorm model. Limitations of my work is that the QDA and the GamLoess method did not give any results Future work on this is that we can use a few more alogorithms to test which could perform better than the Multinorm model.