Assign05

Shivam

2023-04-04

-Loading the required packages for performing all the tasks.

```
if(!require(pastecs)){install.packages("pastecs")}
```

```
## Loading required package: pastecs
```

```
library("pastecs")

if(!require(lattice)){install.packages("lattice")}
```

```
## Loading required package: lattice
```

```
library("lattice")

if(!require(corrgram)){install.packages("corrgram")}
```

```
## Loading required package: corrgram
```

```
##
## Attaching package: 'corrgram'
```

```
## The following object is masked from 'package:lattice':
##
##     panel.fill
```

```
library("corrgram")

if(!require(tinytex)){install.packages("tinytex")}
```

```
## Loading required package: tinytex
```

```
library("tinytex")

if(!require(vcd)){install.packages("vcd")}
```

```
## Loading required package: vcd
```

```
## Loading required package: grid
```

```
library("vcd")

if(!require(HSAUR)){install.packages("HSAUR")}
```

```
## Loading required package: HSAUR
```

```
## Loading required package: tools
```

```
library("HSAUR")

if(!require(rmarkdown)){install.packages("rmarkdown")}
```

```
## Loading required package: rmarkdown
```

```
library("rmarkdown")

if(!require(ggplot2)){install.packages("ggplot2")}
```

```
## Loading required package: ggplot2
```

```
library("ggplot2")

if(!require(polycor)){install.packages("polycor")}
```

```
## Loading required package: polycor
```

```
library("polycor")


if(!require(klaR)){install.packages("klaR")}
```

```
## Loading required package: klaR
```

```
## Loading required package: MASS
```

```
library("klaR")

if(!require(MASS)){install.packages("MASS")}
library("MASS")

if(!require(partykit)){install.packages("partykit")}
```

```
## Loading required package: partykit
```

```
## Loading required package: libcoin
```

```
## Loading required package: mvtnorm
```

```
library("partykit")

if(!require(nnet)){install.packages("nnet")}
```

```
## Loading required package: nnet
```

```
library("nnet")
```

- Clearing the console and the whole workplace,to start with neat and clean workplace .

```
# Clear plots
if(!is.null(dev.list())) dev.off()
```

```
## null device
##           1
```

```
# Clear console
cat("\014")
```

```
# Clean workspace
rm(list=ls())
```

```
#Setting the work directory
setwd("C:/Users/holys/OneDrive/Desktop/Data Analytics,Mathamatics,Algor/Assign05")
```

1 Preliminary Data Preparation

```
dataset_SJ <- read.table("PROG8430_Assign04_23W.txt", header = TRUE, sep = ",")

dataset_SJ[0:15,]     #Printing first 5 data points from the database
```

```
##      DL  VN PG CS   ML DM HZ       CR  WT
## 1   8.1 324  5 13  313  C  N  Sup Del 216
## 2   8.4 135  2 13  830  I  N  Sup Del 160
## 3   8.6 391  3 12  304  C  N  Sup Del  25
## 4  11.3 245  6  7 1258  C  N  Sup Del  67
## 5   5.4 321  1  2  221  C  N Def Post  14
## 6   9.4 397  2  8 1002  I  N  Sup Del  47
## 7   8.2 390  6 13  655  C  N  Sup Del   7
## 8   9.4 252  2  8 1367  I  N  Sup Del   6
## 9   9.3 355  4  2  675  C  N  Sup Del  30
## 10  9.7 159  1 12  888  C  N  Sup Del 177
## 11  8.9 246  1  7  548  C  H Def Post  13
## 12  7.4 377  3 17  287  I  N Def Post  65
## 13  6.3 248  1  1 1286  C  N  Sup Del 137
## 14  8.2 368  3  7 1055  I  N Def Post 216
## 15  6.0 337  2  5  655  I  N Def Post 128
```

```
                      #to make sure it looks correct


str(dataset_SJ)
```

```
## 'data.frame':    487 obs. of  9 variables:
##  $ DL: num  8.1 8.4 8.6 11.3 5.4 9.4 8.2 9.4 9.3 9.7 ...
##  $ VN: int  324 135 391 245 321 397 390 252 355 159 ...
##  $ PG: int  5 2 3 6 1 2 6 2 4 1 ...
##  $ CS: int  13 13 12 7 2 8 13 8 2 12 ...
##  $ ML: int  313 830 304 1258 221 1002 655 1367 675 888 ...
##  $ DM: chr  "C" "I" "C" "C" ...
##  $ HZ: chr  "N" "N" "N" "N" ...
##  $ CR: chr  "Sup Del" "Sup Del" "Sup Del" "Sup Del" ...
##  $ WT: num  216 160 25 67 14 47 7 6 30 177 ...
```

1. Rename all variables with your initials appended (just as was done in previous assignments). Remember that any variables you subsequently create need to have your initials appended.

```
colnames(dataset_SJ) <- paste(colnames(dataset_SJ), "SJ", sep = "_")  # renaming all the
                                                                      # variables and adding
                                                                      # my initials SJ

  head(dataset_SJ)
```

```
##   DL_SJ VN_SJ PG_SJ CS_SJ ML_SJ DM_SJ HZ_SJ    CR_SJ WT_SJ
## 1   8.1   324     5    13   313     C     N  Sup Del   216
## 2   8.4   135     2    13   830     I     N  Sup Del   160
## 3   8.6   391     3    12   304     C     N  Sup Del    25
## 4  11.3   245     6     7  1258     C     N  Sup Del    67
## 5   5.4   321     1     2   221     C     N Def Post    14
## 6   9.4   397     2     8  1002     I     N  Sup Del    47
```

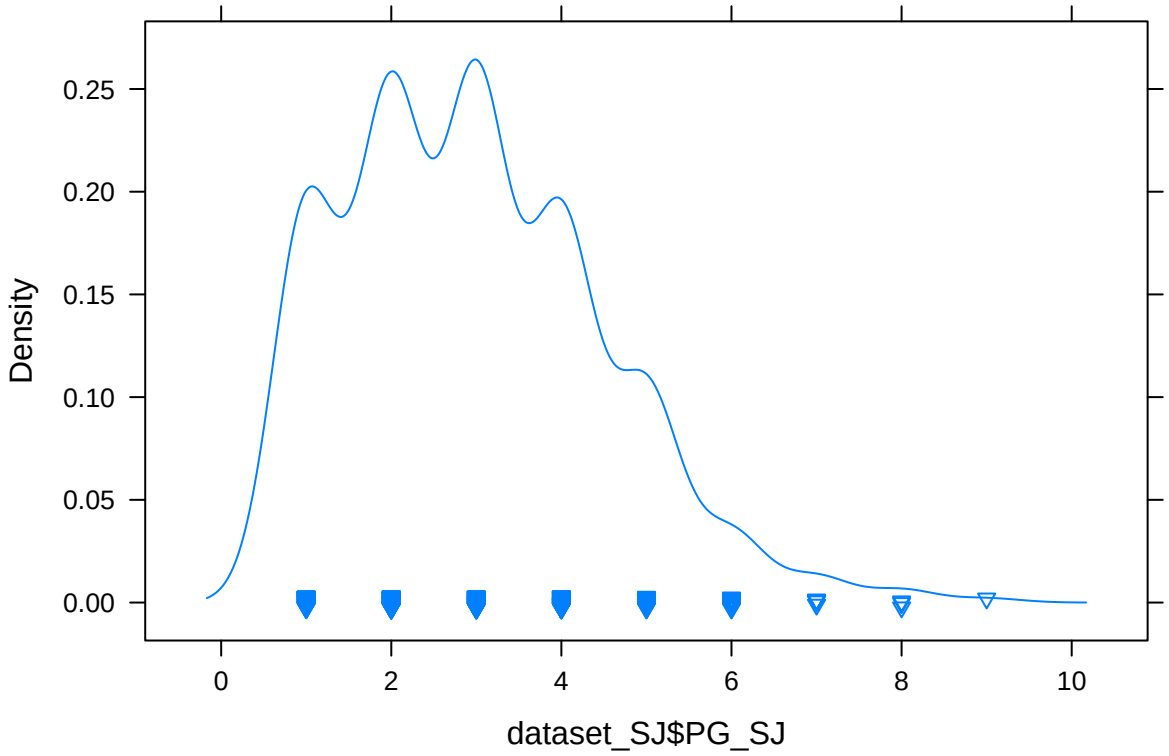
```
# Changed all the character variables to factors as to use them
#to plot and examine with the help of code given in the announcement
dataset_SJ <- as.data.frame(unclass(dataset_SJ), stringsAsFactors = TRUE)
```

2. Since this is the same dataset as used in Assignment 4, you do not need to do the regular descriptive analysis and outlier detection. As in Assignment 4, remember to delete the observation with PG <0 using the following code: ClssAssign <- ClssAssign[!ClssAssign$PG < 0,] NOTE – Your variable names will be different of course!

   -one data point which is negative in the field PG that is packages , which is impossible to have value of -2, it clearly signifies that something is fishy going there! PG refers to the number of packages of product that have been ordered, and that being negative is impossible unless it is a mistake or wrong interpretation. No product can have number of packages equal to -2. So, its a meaning less data point which should be removed.

```
# Removing the outlier

outlier_In_PG_SJ <- which(dataset_SJ$PG_SJ < 0) #Fond row number with PG
                                                #less then to 0
dataset_SJ <- dataset_SJ[-c(outlier_In_PG_SJ),] #deleted the row
densityplot( ~ dataset_SJ$PG_SJ, pch=6)
```



3. Create a new variable in the dataset called OT_[Intials] which will have a value of 1 if DL 8.5 and 0 otherwise. If you have forgotten how to do this, the code to accomplish it is included in the appendix. Remember to also delete the DL variable after this. - Creating new variable field called OT_SJ in the dataset as instructed. OT will have values 0 or one on the basis of the values of DL; if the DL value is less then equal to 8.5, then it will have OT value as 1 or less it will have 0. We will also delete the field DL as instructed.

```
dataset_SJ$OT_SJ <- as.factor(ifelse(dataset_SJ$DL_SJ <= 8.5, 1,0))
  head(dataset_SJ)
```

```
##   DL_SJ VN_SJ PG_SJ CS_SJ ML_SJ DM_SJ HZ_SJ    CR_SJ WT_SJ OT_SJ
## 1   8.1   324     5    13   313     C     N  Sup Del   216     1
## 2   8.4   135     2    13   830     I     N  Sup Del   160     1
## 3   8.6   391     3    12   304     C     N  Sup Del    25     0
## 4  11.3   245     6     7  1258     C     N  Sup Del    67     0
## 5   5.4   321     1     2   221     C     N Def Post    14     1
## 6   9.4   397     2     8  1002     I     N  Sup Del    47     0
```

```
dataset_SJ <- dataset_SJ[-c(1)]      #Removing the DL column as instructed
                                     #in the task

head(dataset_SJ)
```

```
##   VN_SJ PG_SJ CS_SJ ML_SJ DM_SJ HZ_SJ    CR_SJ WT_SJ OT_SJ
```

```
## 1   324     5    13   313     C     N  Sup Del   216     1
## 2   135     2    13   830     I     N  Sup Del   160     1
## 3   391     3    12   304     C     N  Sup Del    25     0
## 4   245     6     7  1258     C     N  Sup Del    67     0
## 5   321     1     2   221     C     N Def Post    14     1
## 6   397     2     8  1002     I     N  Sup Del    47     0
```

2 Exploratory Analysis

1. Correlations: Create correlations (as demonstrated) and comment on what you see. Are there co-linear variables?

- Splitting the data set into two parts as learned in the classes, one train set and t he other test set. I kept the sampling rate as 0.8, so splitting the data into 80/20 ratio. Also set the seed as 9647 according to the last four digits of my student number.

To get the coefficient matrix by using the hetcor function

```r
ht_Cor_SJ <- hetcor(train_SJ)
round(ht_Cor_SJ$correlations,2)
```

```
##       VN_SJ PG_SJ CS_SJ ML_SJ DM_SJ HZ_SJ CR_SJ WT_SJ OT_SJ
## VN_SJ  1.00 -0.01 -0.04 -0.02  0.12  0.01 -0.06  0.00  0.01
## PG_SJ -0.01  1.00  0.04  0.07  0.02  0.01 -0.07 -0.03 -0.49
## CS_SJ -0.04  0.04  1.00 -0.03  0.13 -0.04  0.02 -0.01 -0.04
## ML_SJ -0.02  0.07 -0.03  1.00 -0.04 -0.07  0.09 -0.02 -0.23
## DM_SJ  0.12  0.02  0.13 -0.04  1.00  0.05  0.06 -0.03 -0.09
## HZ_SJ  0.01  0.01 -0.04 -0.07  0.05  1.00  0.10  0.09  0.21
## CR_SJ -0.06 -0.07  0.02  0.09  0.06  0.10  1.00 -0.08 -0.36
## WT_SJ  0.00 -0.03 -0.01 -0.02 -0.03  0.09 -0.08  1.00  0.39
## OT_SJ  0.01 -0.49 -0.04 -0.23 -0.09  0.21 -0.36  0.39  1.00
```
+``

3 Model Development As demonstrated in class, create two logistic regression models. 1. A full model using all of the variables. Lets create a full model, using all the variables of the dataset and OT as dependent variable.

```r
########################################
## Creating Baseline/Full Model       ##
########################################

glm.full_SJ <- glm( OT_SJ~ ., data=train_SJ,
                family="binomial", na.action=na.omit)
summary(glm.full_SJ)
```

```
##
```

```
## Call:
## glm(formula = OT_SJ ~ ., family = "binomial", data = train_SJ,
##     na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3450  -0.8416   0.3223   0.8125   2.5966
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.2508389  0.8133799   2.767  0.00565 **
## VN_SJ         0.0002311  0.0016767   0.138  0.89037
## PG_SJ        -0.7787848  0.1046241  -7.444 9.79e-14 ***
## CS_SJ        -0.0033440  0.0246415  -0.136  0.89205
## ML_SJ        -0.0009352  0.0003159  -2.960  0.00308 **
## DM_SJI       -0.2572698  0.2783113  -0.924  0.35528
## HZ_SJN        0.9331330  0.3760491   2.481  0.01309 *
## CR_SJSup Del -1.3399372  0.2688856  -4.983 6.25e-07 ***
## WT_SJ         0.0083193  0.0015421   5.395 6.85e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 535.56  on 387  degrees of freedom
## Residual deviance: 387.75  on 379  degrees of freedom
## AIC: 405.75
##
## Number of Fisher Scoring iterations: 5
```

2. An additional model using backward selection. - Lets make another model using Backward Selection

```
########################################
## Creating Backward Selection Model   ##
########################################
back_model_SJ = step(glm.full_SJ, direction="backward", details=TRUE)
```

```
## Start:  AIC=405.75
## OT_SJ ~ VN_SJ + PG_SJ + CS_SJ + ML_SJ + DM_SJ + HZ_SJ + CR_SJ +
##     WT_SJ
##
##         Df Deviance    AIC
## - CS_SJ  1   387.77 403.77
## - VN_SJ  1   387.77 403.77
## - DM_SJ  1   388.61 404.61
## <none>       387.75 405.75
## - HZ_SJ  1   394.06 410.06
## - ML_SJ  1   396.82 412.82
## - CR_SJ  1   414.91 430.91
## - WT_SJ  1   423.43 439.43
## - PG_SJ  1   462.63 478.63
##
## Step:  AIC=403.77
```

```
## OT_SJ ~ VN_SJ + PG_SJ + ML_SJ + DM_SJ + HZ_SJ + CR_SJ + WT_SJ
##
##         Df Deviance    AIC
## - VN_SJ  1   387.79 401.79
## - DM_SJ  1   388.65 402.65
## <none>       387.77 403.77
## - HZ_SJ  1   394.14 408.14
## - ML_SJ  1   396.82 410.82
## - CR_SJ  1   414.98 428.98
## - WT_SJ  1   423.47 437.47
## - PG_SJ  1   462.85 476.85
##
## Step:  AIC=401.79
## OT_SJ ~ PG_SJ + ML_SJ + DM_SJ + HZ_SJ + CR_SJ + WT_SJ
##
##         Df Deviance    AIC
## - DM_SJ  1   388.65 400.65
## <none>       387.79 401.79
## - HZ_SJ  1   394.15 406.15
## - ML_SJ  1   396.84 408.84
## - CR_SJ  1   415.09 427.09
## - WT_SJ  1   423.47 435.47
## - PG_SJ  1   462.88 474.88
##
## Step:  AIC=400.65
## OT_SJ ~ PG_SJ + ML_SJ + HZ_SJ + CR_SJ + WT_SJ
##
##         Df Deviance    AIC
## <none>       388.65 400.65
## - HZ_SJ  1   394.86 404.86
## - ML_SJ  1   397.56 407.56
## - CR_SJ  1   416.16 426.16
## - WT_SJ  1   424.69 434.69
## - PG_SJ  1   463.98 473.98
```

```
summary(back_model_SJ)
```

```
##
## Call:
## glm(formula = OT_SJ ~ PG_SJ + ML_SJ + HZ_SJ + CR_SJ + WT_SJ,
##     family = "binomial", data = train_SJ, na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3146  -0.8171   0.3198   0.8098   2.6271
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.2130636  0.5419303   4.084 4.43e-05 ***
## PG_SJ        -0.7774316  0.1041859  -7.462 8.52e-14 ***
## ML_SJ        -0.0009252  0.0003152  -2.935  0.00334 **
## HZ_SJN        0.9258039  0.3761665   2.461  0.01385 *
## CR_SJSup Del -1.3456127  0.2684283  -5.013 5.36e-07 ***
## WT_SJ         0.0083701  0.0015423   5.427 5.74e-08 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 535.56  on 387  degrees of freedom
## Residual deviance: 388.65  on 382  degrees of freedom
## AIC: 400.65
##
## Number of Fisher Scoring iterations: 5
```

For each model, interpret and comment on the main measures we discussed in class:

(1) Fisher iterations

- By comparing both the models , we can see that there is convergence by seeing number of fisher iterations, both of them have same number of iterations.

(2) AIC

- For full model, the AIC value is 405.75 and while using back model it is observed that AIC is equal; to 400.65. As we know lower the value of AIC, better the model.

(3) Residual Deviance

- Observing the Residual deviance, we saw that there is no major difference in comparing both the models, full model has deviance of near about 147.81 and it is 146.91 in the case of backward model. Higher the difference, better the model. Here, if we see the aspect residual deviance, Full model has a very little more difference.

(4) Residual symmetry

- Observing in both the models, in both the models the residuals are balanced as the median is near zero and the 1Q and 3Q are symmetrical and median is near

0.

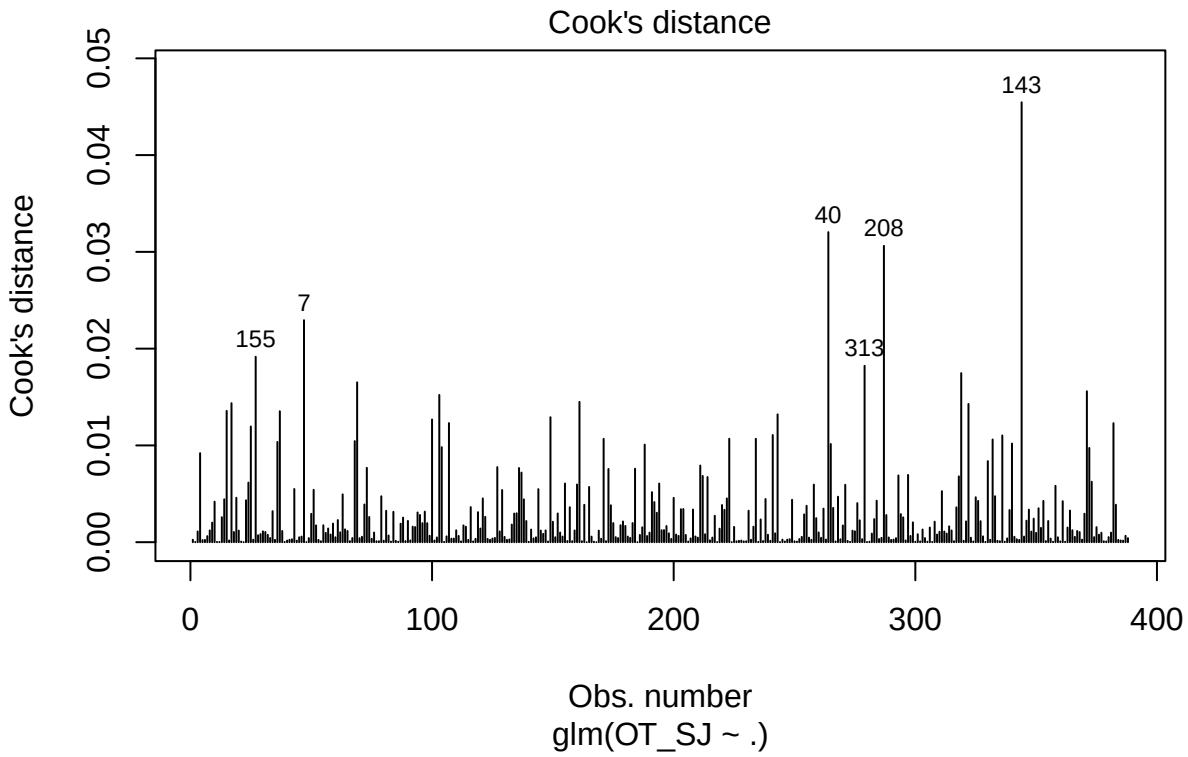
(5) z-values As comparing the p values of Z-test, we saw that all the variable passed the test in the case of Back model as all the p-values are less then 0.05 , and in the case of full model, six out of eight passed the test, according to that pack model is better. Three of the variables and not passing the test in the case of full model.

(6) Parameter Co-Efficients As comparing the coefficients, with the correlation matrix from the hetcor function, all the coefficients matches as all the coefficients that were obtained negative value had negative values in the matrix also fro both the models that is back and full.

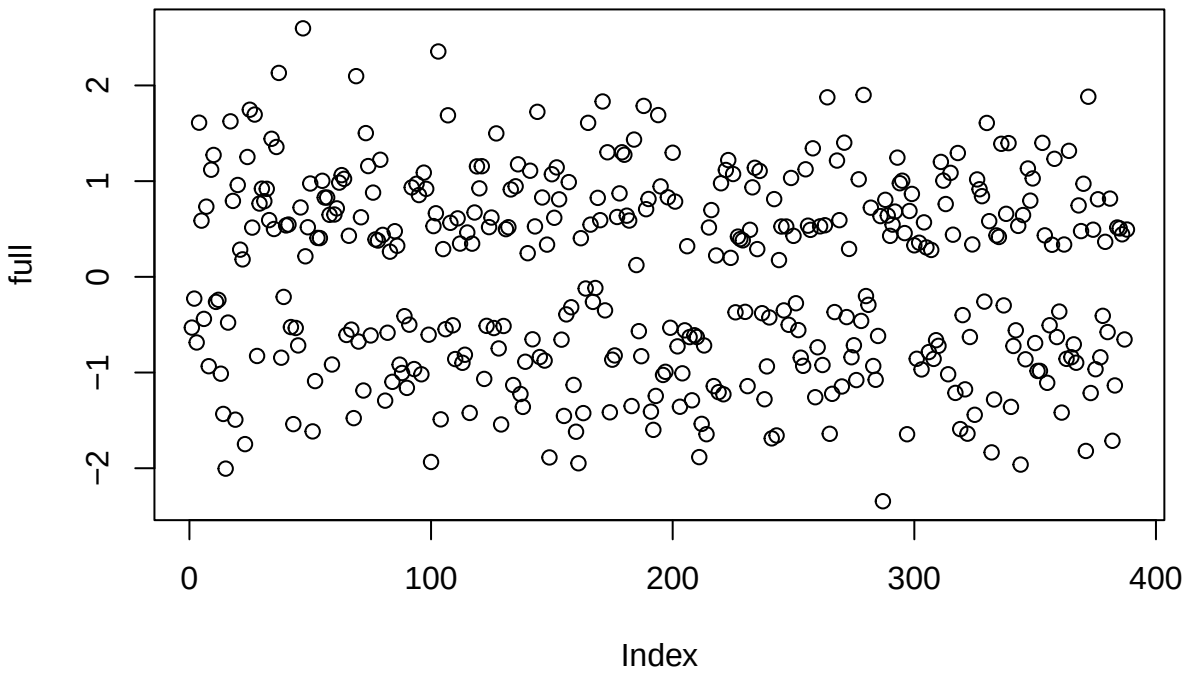
For both models created we will also check the assumptions, that is, for logistic regression, all the points should be centered around 0 and cooks distance should be in the range. So, for that lest plot graphs for checking both the assumptions, to see its datapoints and the cook's distance.

Lets plot the graphs to check that the full model fulfill the assumptions.

```
plot(glm.full_SJ, which=4, id.n=6)
```



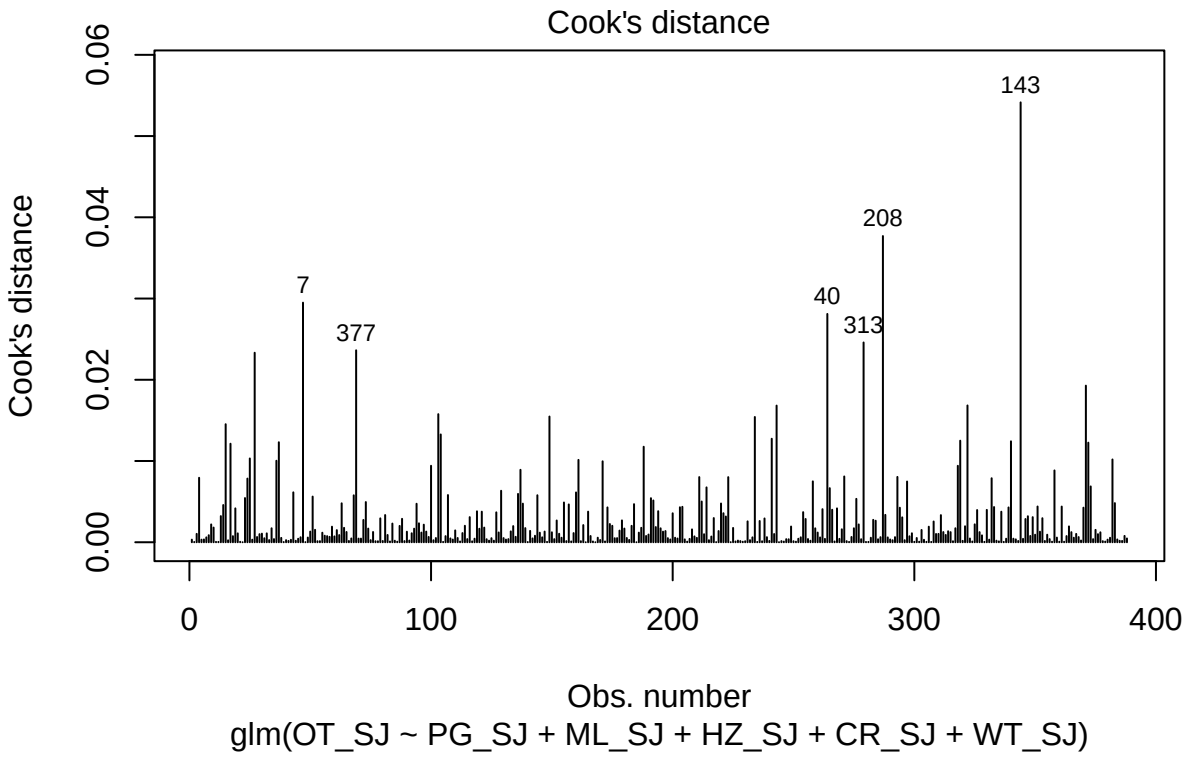

```
full <- residuals(glm.full_SJ)
plot(full)
```



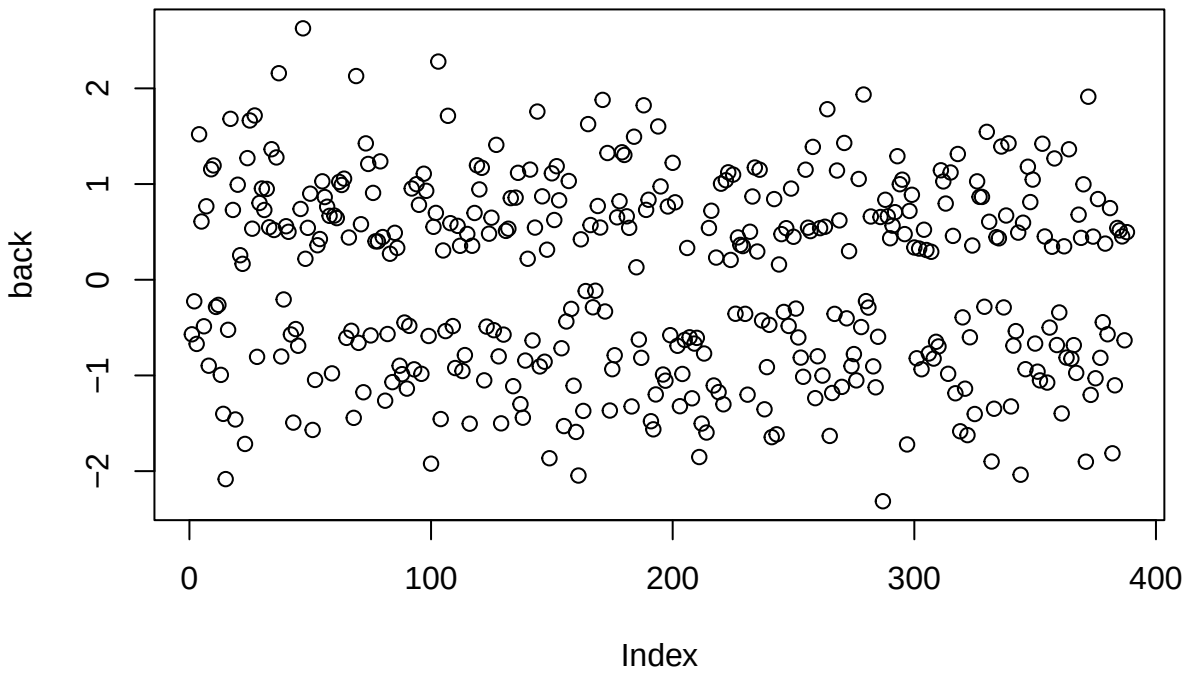
- we saw that the datapoints from the full model are centered around zero and the cook's distance is in the range as we saw from the graph above. As any of the cook's distance is not greater then 0.5,Both the assumptions are fulfilled so our model is good.

Checking the assumptions for the backward model

```
plot(back_model_SJ, which=4, id.n=6)
```




```
back <- residuals(back_model_SJ)
plot(back)
```



For the back model also both the assumptions are fulfilled as observed from the graph that all the datapoints are near about zero and cook's distance in the range , that is; As any of the cook's distance is not greater then 0.5,there is no influence on any particular data point in both the models.

3. As demonstrated in class, analyze the output for any significantly influential datapoints. Are there any?

   There are no points having high influence in both the models, for both the cases there are no points out of the cook's distance, both the models don't have any datapoints which are highly influencing the regression line.

4. Based on your preceding analysis, recommend which model should be selected and explain why.

   According to me, back model is better as both the AICs are very near to each other of both the models , only in the full model, three of the variables are not passing the z-test as their p values are not less then 0.05 and on the other hand all the variables passes the z-test in Back model, so it is better.

PART B **********

1 Logistic Regression – stepwise 1. As above, use the step option in the glm function to fit the model (using stepwise selection).

```
start_time_SJ <- Sys.time()

glm.mod_SJ = glm(OT_SJ ~ .    ,
              family="binomial", data=train_SJ, na.action=na.omit)
```

```r
stp.glm_SJ <- step(glm.mod_SJ, trace=FALSE)

end_time_SJ <- Sys.time()

GLM_Time_SJ <- end_time_SJ - start_time_SJ

summary(stp.glm_SJ)
```

```
##
## Call:
## glm(formula = OT_SJ ~ PG_SJ + ML_SJ + HZ_SJ + CR_SJ + WT_SJ,
##     family = "binomial", data = train_SJ, na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3146  -0.8171   0.3198   0.8098   2.6271
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.2130636  0.5419303   4.084 4.43e-05 ***
## PG_SJ        -0.7774316  0.1041859  -7.462 8.52e-14 ***
## ML_SJ        -0.0009252  0.0003152  -2.935  0.00334 **
## HZ_SJN        0.9258039  0.3761665   2.461  0.01385 *
## CR_SJSup Del -1.3456127  0.2684283  -5.013 5.36e-07 ***
## WT_SJ         0.0083701  0.0015423   5.427 5.74e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 535.56  on 387  degrees of freedom
## Residual deviance: 388.65  on 382  degrees of freedom
## AIC: 400.65
##
## Number of Fisher Scoring iterations: 5
```

2. Summarize the results in Confusion Matrices for both train and test sets. Lets first create the confusion matrices for both test and train sets

```r
# Create Confusion Matrix for train

resp_glm_step_SJ <- predict(stp.glm_SJ, newdata=train_SJ, type="response")
Class_glm_step_SJ <- ifelse(resp_glm_step_SJ > 0.5,"Y","N")
CF_GLM_step_train_SJ <- table(train_SJ$OT_SJ, Class_glm_step_SJ,
                dnn=list("Act OT","Predicted") )
CF_GLM_step_train_SJ
```

```
##       Predicted
## Act OT   N   Y
##      0 128  51
##      1  43 166
```

```r
# Create Confusion Matrix for test

resp_glm_step_test_SJ <- predict(stp.glm_SJ, newdata=test_SJ, type="response")
Class_glm_step_test_SJ <- ifelse(resp_glm_step_test_SJ > 0.5,"Y","N")
CF_GLM_step_test_SJ <- table(test_SJ$OT_SJ, Class_glm_step_test_SJ,
                dnn=list("Act OT","Predicted") )

CF_GLM_step_test_SJ
```

```
##       Predicted
## Act OT  N  Y
##      0 44 10
##      1 12 32
```

```r
GLM_Time_SJ
```

```
## Time difference of 0.09830499 secs
```

```r
#For summarizing confusion matrix of train set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_glm.mod_SJ <- (CF_GLM_step_train_SJ[2,2] + CF_GLM_step_train_SJ[1,1])/sum(CF_GLM_step_train_SJ
paste('Accuracy is :TP+TN/Total =',round(Accuracy_glm.mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.76"
```

```r
#Sensitivity= TP/(TP+FN)
Sensitivity_glm.mod_SJ <- CF_GLM_step_train_SJ[2,2] / (CF_GLM_step_train_SJ[2,2]+ CF_GLM_step_train_SJ[1
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_glm.mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.79"
```

```r
#Specificity= TN/(TN+FP)
Specificity_glm.mod_SJ <- CF_GLM_step_train_SJ[1,1]/(CF_GLM_step_train_SJ[1,1]+ CF_GLM_step_train_SJ[1,2
paste('Specificity is :TN/(TN+FP) =',round(Specificity_glm.mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.72"
```

```r
#Precision = TP/(TP+FP)
Precision_glm.mod_SJ <- CF_GLM_step_test_SJ[2,2]/(CF_GLM_step_test_SJ[2,2] + CF_GLM_step_test_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_glm.mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.76"
```

Calculating the same measures for the test set as well:

```r
#For summarizing confusion matrix of test set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_glm.mod_SJ <- (CF_GLM_step_test_SJ[2,2] + CF_GLM_step_test_SJ[1,1])/sum(CF_GLM_step_test_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_glm.mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.78"
```

```r
#Sensitivity= TP/(TP+FN)
Sensitivity_glm.mod_SJ <- CF_GLM_step_test_SJ[2,2] / (CF_GLM_step_test_SJ[2,2]+ CF_GLM_step_test_SJ[2,1]
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_glm.mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.73"
```

```r
#Specificity= TN/(TN+FP)
Specificity_glm.mod_SJ <- CF_GLM_step_test_SJ[1,1]/(CF_GLM_step_test_SJ[1,1]+ CF_GLM_step_test_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_glm.mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.81"
```

```r
#Precision = TP/(TP+FP)
Precision_glm.mod_SJ <- CF_GLM_step_test_SJ[2,2]/(CF_GLM_step_test_SJ[2,2] + CF_GLM_step_test_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_glm.mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.76"
```

```
As we can observe that as we trained the dataset, we got 0.76 accuracy on
the train set,and we got 0.78 as accuracy, that shows that we had a good
model giving better results.We can also observe that all the train dataset
have similar precision and sensitivity with the test dataset outcomes. We
can also see that we got 0.81 Specificity on the test set by our model.
```

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary.
   As I also calculated above the timings for running both the models,

```r
GLM_Time_SJ
```

```
## Time difference of 0.09830499 secs
```

It took about 0.124357 secs to run the step wise model. It takes the longest times of all the models, but it gives highest accuracy, highest precision , good sensitivity and specificity. ~~~~~~

2 Naïve-Bayes Classification

1. Use all the variables in the dataset to fit a Naïve-Bayesian classification model.

```r
start_time <- Sys.time()

NB.mod_SJ <- NaiveBayes(OT_SJ ~ . ,
                     data = train_SJ, na.action=na.omit)

end_time <- Sys.time()

NB_Time <- end_time - start_time
NB_Time
```

```
## Time difference of 0.004684925 secs
```

2. Summarize the results in Confusion Matrices for both train and test sets.

```
# Create Confusion Matrix for train
pred_NB <- predict(NB.mod_SJ,newdata=train_SJ)
CF_NB_train_SJ <- table(Actual=train_SJ$OT_SJ, Predicted=pred_NB$class)
CF_NB_train_SJ
```

```
##       Predicted
## Actual   0   1
##      0 125  54
##      1  41 168
```

```
# Create Confusion Matrix for test
pred_NB <- predict(NB.mod_SJ,newdata=test_SJ)
CF_NB_test_SJ <- table(Actual=test_SJ$OT_SJ, Predicted=pred_NB$class)
CF_NB_test_SJ
```

```
##       Predicted
## Actual  0  1
##      0 41 13
##      1 17 27
```

```
#For summarizing confusion matrix of train set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_NB.mod_SJ <- (CF_NB_train_SJ[2,2] + CF_NB_train_SJ[1,1])/sum(CF_NB_train_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_NB.mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.76"
```

```
#Sensitivity= TP/(TP+FN)
Sensitivity_NB.mod_SJ <- CF_NB_train_SJ[2,2] / (CF_NB_train_SJ[2,2]+ CF_NB_train_SJ[2,1])
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_NB.mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.8"
```

```
#Specificity= TN/(TN+FP)
Specificity_NB.mod_SJ <- CF_NB_train_SJ[1,1]/(CF_GLM_step_train_SJ[1,1]+ CF_NB_train_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_NB.mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.69"
```

```
#Precision = TP/(TP+FP)
Precision_NB.mod_SJ <- CF_NB_train_SJ[2,2]/(CF_NB_train_SJ[2,2] + CF_NB_train_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_NB.mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.76"
```

```r
#For summarizing confusion matrix of test set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_NB.mod_SJ <- (CF_NB_test_SJ[2,2] + CF_NB_test_SJ[1,1])/sum(CF_NB_test_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_NB.mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.69"
```

```r
#Sensitivity= TP/(TP+FN)
Sensitivity_NB.mod_SJ <- CF_NB_test_SJ[2,2] / (CF_NB_test_SJ[2,2]+ CF_NB_test_SJ[2,1])
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_NB.mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.61"
```

```r
#Specificity= TN/(TN+FP)
Specificity_NB.mod_SJ <- CF_NB_test_SJ[1,1]/(CF_GLM_step_test_SJ[1,1]+ CF_NB_test_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_NB.mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.72"
```

```r
#Precision = TP/(TP+FP)
Precision_NB.mod_SJ <- CF_NB_test_SJ[2,2]/(CF_NB_test_SJ[2,2] + CF_NB_test_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_NB.mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.68"
```

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary. - It takes the least time , that is; it is the fastest of all models,

```r
NB_Time
```

```
## Time difference of 0.004684925 secs
```

~~~~~~

3 Recursive Partitioning Analysis 1. Use all the variables in the dataset to fit an recursive partitioning classification model.

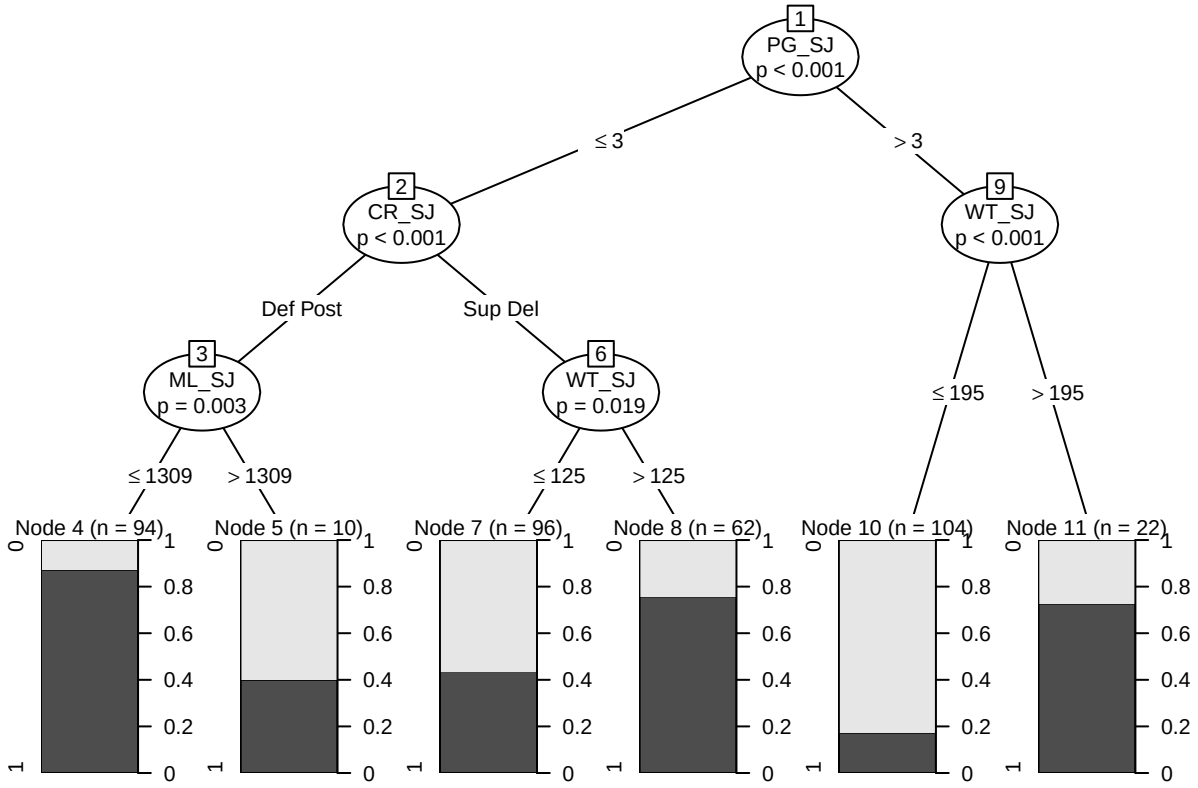
```r
start_time <- Sys.time()

RP.mod_SJ <- ctree(OT_SJ ~ ., data=train_SJ)

end_time <- Sys.time()

RP_Time <- end_time - start_time

plot(RP.mod_SJ, gp=gpar(fontsize=8))
```
~~~~~~



2. Summarize the results in Confusion Matrices for both train and test sets.

```
# Create Confusion Matrix for train
pred.RP <- predict(RP.mod_SJ, newdata=train_SJ)
CF_RP_train_SJ <- table(Actual=train_SJ$OT_SJ, Predicted=pred.RP)
CF_RP_train_SJ
```

```
##       Predicted
## Actual   0   1
##      0 146  33
##      1  64 145
```

```
# Create Confusion Matrix for test
pred.RP_test <- predict(RP.mod_SJ, newdata=test_SJ)
CF_RP_test_SJ <- table(Actual=test_SJ$OT_SJ, Predicted=pred.RP_test)
CF_RP_train_SJ
```

```
##       Predicted
## Actual   0   1
##      0 146  33
##      1  64 145
```

```
#For summarizing confusion matrix of train set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_RP_mod_SJ <- (CF_RP_train_SJ[2,2] + CF_RP_train_SJ[1,1])/sum(CF_RP_train_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_RP_mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.75"
```

```
#Sensitivity= TP/(TP+FN)
Sensitivity_RP_mod_SJ  <- CF_RP_train_SJ[2,2] / (CF_RP_train_SJ[2,2]+ CF_RP_train_SJ[2,1])
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_RP_mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.69"
```

```
#Specificity= TN/(TN+FP)
Specificity_RP_mod_SJ <- CF_RP_train_SJ[1,1]/(CF_RP_train_SJ[1,1]+ CF_RP_train_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_RP_mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.82"
```

```
#Precision = TP/(TP+FP)
Precision_RP_mod_SJ <- CF_RP_train_SJ[2,2]/(CF_RP_train_SJ[2,2] + CF_RP_train_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_RP_mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.81"
```

```
#For summarizing confusion matrix of test set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_RP_mod_SJ <- (CF_RP_test_SJ[2,2] + CF_RP_test_SJ[1,1])/sum(CF_RP_test_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_RP_mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.69"
```

```
#Sensitivity= TP/(TP+FN)
Sensitivity_RP_mod_SJ <- CF_RP_test_SJ[2,2] / (CF_RP_test_SJ[2,2]+ CF_RP_test_SJ[2,1])
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_RP_mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.52"
```

```
#Specificity= TN/(TN+FP)
Specificity_RP_mod_SJ <- CF_RP_test_SJ[1,1]/(CF_RP_test_SJ[1,1]+ CF_RP_test_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_RP_mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.83"
```

```
#Precision = TP/(TP+FP)
Precision_RP_mod_SJ <- CF_RP_test_SJ[2,2]/(CF_RP_test_SJ[2,2] + CF_RP_test_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_RP_mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.72"
```

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary.

```
RP_Time
```

```
## Time difference of 0.03340602 secs
```

- RP takes about 0.05 seconds that is good enough. ++ ## 3a BONUS SECTION ##

This section is bonus marks. There is no need to complete this, but successful completion of this section will be worth 3 bonus marks.

Neural Network

1. Use all the variables in the dataset to fit a Neural Network classification model. Set the seed to 8430, the size to 3, rang=0.1 and maxit=1200.

```
start_time <- Sys.time()

set.seed(8430)
nn.mod_SJ <- nnet(OT_SJ ~ .,
          data=train_SJ,
          size=3,
          rang=0.1,
          maxit=1200,
          trace=FALSE)

end_time <- Sys.time()

NN_Time <- end_time - start_time

pred.nn <- predict(nn.mod_SJ, newdata=train_SJ, type="class")

NN_Time
```

```
## Time difference of 0.02465391 secs
```

2. Summarize the results in Confusion Matrices for both train and test sets.

```
# Create Confusion Matrix for train
pred.nn_train_SJ <- predict(nn.mod_SJ, newdata=train_SJ, type="class")
CF_NN_train_SJ <- table(Actual=train_SJ$OT_SJ, Predicted=pred.nn_train_SJ)
CF_NN_train_SJ
```

```
##       Predicted
## Actual   0   1
##      0 144  35
##      1 114  95
```

```
# Create Confusion Matrix for test
pred.nn_test_SJ <- predict(nn.mod_SJ, newdata=test_SJ, type="class")
CF_NN_test_SJ <- table(Actual=test_SJ$OT_SJ, Predicted=pred.nn_test_SJ)
CF_NN_test_SJ
```

```
##       Predicted
## Actual  0  1
##      0 46  8
##      1 31 13
```

```
#For summarizing confusion matrix of train set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_NN_mod_SJ <- (CF_NN_train_SJ[2,2] + CF_NN_train_SJ[1,1])/sum(CF_NN_train_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_NN_mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.62"
```

```
#Sensitivity= TP/(TP+FN)
Sensitivity_NN_mod_SJ  <- CF_NN_train_SJ[2,2] / (CF_NN_train_SJ[2,2]+ CF_NN_train_SJ[2,1])
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_NN_mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.45"
```

```
#Specificity= TN/(TN+FP)
Specificity_NN_mod_SJ <- CF_NN_train_SJ[1,1]/(CF_NN_train_SJ[1,1]+ CF_NN_train_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_NN_mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.8"
```

```
#Precision = TP/(TP+FP)
Precision_NN_mod_SJ <- CF_NN_train_SJ[2,2]/(CF_NN_train_SJ[2,2] + CF_NN_train_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_NN_mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.73"
```

```
#For summarizing confusion matrix of test set
#As we have summarize both the confusion matrices, lets calculate the accuracy that is (TP+TN/Total)
Accuracy_NN_mod_SJ <- (CF_NN_test_SJ[2,2] + CF_NN_test_SJ[1,1])/sum(CF_NN_test_SJ)
paste('Accuracy is :TP+TN/Total =',round(Accuracy_NN_mod_SJ,2))
```

```
## [1] "Accuracy is :TP+TN/Total = 0.6"
```

```
#Sensitivity= TP/(TP+FN)
Sensitivity_NN_mod_SJ  <- CF_NN_test_SJ[2,2] / (CF_NN_test_SJ[2,2]+ CF_NN_test_SJ[2,1])
paste('Sensitivity is :TP/(TP+FN) =',round(Sensitivity_NN_mod_SJ,2))
```

```
## [1] "Sensitivity is :TP/(TP+FN) = 0.3"
```

```
#Specificity= TN/(TN+FP)
Specificity_NN_mod_SJ <- CF_NN_test_SJ[1,1]/(CF_NN_test_SJ[1,1]+ CF_NN_test_SJ[1,2])
paste('Specificity is :TN/(TN+FP) =',round(Specificity_NN_mod_SJ,2))
```

```
## [1] "Specificity is :TN/(TN+FP) = 0.85"
```

```
#Precision = TP/(TP+FP)
Precision_NN_mod_SJ <- CF_NN_test_SJ[2,2]/(CF_NN_test_SJ[2,2] + CF_NN_test_SJ[1,2])
paste('Precision is :TP/(TP+FP) =',round(Precision_NN_mod_SJ,2))
```

```
## [1] "Precision is :TP/(TP+FP) = 0.62"
```

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary.

++
##

4 Compare All Classifiers For all questions below please provide evidence.

1. Which classifier is most accurate?

- Going through the accuracies of all the four models, I can observe that the most accurate result is given by the Stepwise Logistic Regression model, as it gives the exact result of 0.76 in test set as that of train set, if we compare the other models, in naive baye's Model, it chages form 0.76 to 0.69, in the case of Recurssive Partitioning,the accuracy duiffers from 0.75 in train to 0.69 in test, and for Neural netwrok, it changes from 0.62 to 0.60 which is good enough.

2. Which classifier seems most consistent (think train and test)?
- Again going trough all ther four aspects, accuracy, sensitivity,specificity and precision, the most consistant is Logistic Regression model (stepwise), which gives the most consistant result, precision is exact same for both test and train set, the accuracy is also very near, that is 0.78 from 0.76, sensitivity is consistant with a very little difference of about 7%. Specificity is having difference of 0.03 which is very less. Hence the most consistant model is LR.

3. Which classifier is most suitable when processing speed is most important?
   - Going throyugh the time taken for all the models to run , the most time is taken by Stepwise Logistic Regression Model.Neural Network takes significant time.Naive Baye's and Recurssive Partitioning
   models almost take similar time, but the fastest of all is Naive-Bayes classifier.

4. Which classifier minimizes false positives?
- It can be observed that Neural Network has the least of False Positives that is 8, from the test data it was 35 and in test , it was reduced to 8, which is excellent.

5. In your opinion, classifier is best overall?

- Going through all the specifics and comparing the four major aspects that is Accuracy, Sensitivity, Specificity and Precision we can observe that in the Logistic Regression has Highest Accuracy of 0.78 that is about 78%, Sensitivity is 0.73, which isalso very near to that iof train set, while comparing specificity it give a little high value of 0.81 as compared to

0.72, which is okay! not that bad! Talking about the precision, it is exact the dsame value of 0.76m, same as I obtained in train set, that is the highest Precision out of all models,If we compare the Naive Bayes Model, the accuracy is reduced by about 7% , sensitivity has a difference of about 20 % which is very bad. Comparing other two aspects thast is specificityu and precision, also have significant difference. Comparing the Recursive Partitioning model, it gives similar specificity of 0of about 0.83 which is very near to that in trainset. Neural Network gives the worse of sensitivity as as it was 0.45 on train set and 0.50 which have difference of about 15% . Comparing all the aspects, I can say that Logistic Regression gives best results although it takes a little bit more time but gives high accuracy, high precision and also High sensitivity that is number of true positives,is also better. Therefore I will choose and prefer that the stepwise Logistic Regression is the best Classifier.
Logistic Regression is the best Classifier as it is most accurate and most precise although it takes a little more time .

**********Thank you **********