Breast Cancer Analysis

Importing the data for Breast cancer analysis

```
#Loading the dataset in r
wisc_bc_df <- read.csv("D:/Fall 2018/ABI/Group Assignment/wisc_bc_data.csv")

#Printing the structure of the dataset
str(wisc_bc_df)

## 'data.frame':    569 obs. of  32 variables:
##  $ id               : int  87139402 8910251 905520 868871 9012568 906539
925291 87880 862989 89827 ...
##  $ diagnosis        : Factor w/ 2 levels "B","M": 1 1 1 1 1 1 1 2 1 1 ...
##  $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
##  $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
##  $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
##  $ area_mean        : num  464 346 373 385 712 ...
##  $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
##  $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
##  $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
##  $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
##  $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
##  $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
##  $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
##  $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
##  $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
##  $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
##  $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
##  $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
##  $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
##  $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
##  $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
##  $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
##  $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
##  $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
##  $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
##  $ area_worst       : num  549 425 471 434 819 ...
##  $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
##  $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
##  $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
##  $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
##  $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
##  $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

Since there are a lot of variables we cant really tell which ones are redundant. Thus we keep all the variables as of now. Also after looking at the data we can say that the second variable appears to be the classification.

Now we Organize the data

```
#Printing the value counts of 'B;' and 'M' from the column
table(wisc_bc_df$diagnosis)

##
##   B   M
## 357 212

#Checking if there are na values in the diagnosis column of the dataframe
sum(is.na(wisc_bc_df$diagnosis))

## [1] 0

#Renaming the labels pf the column from "B","M" to "benign","malugnant"
wisc_bc_df$diagnosis <- factor(wisc_bc_df$diagnosis, levels = c("B","M"),
labels = c("benign","malignant"))
```

If we take a look at the data we get to know that not all the parameters are on the same scale of the measurement hence we need to transform all variables to comparable scales.

We here define a finction to normalize the values

```
#defining the function and storing the function the variable normalize
normalize <- function(x){
  y <- (x-min(x))/(max(x)-min(x))
  y
}

#Applying Normalize function on columns 3 till 32 for all the rows
wbcd_n_L <- lapply(wisc_bc_df[ , 3:32], normalize)

#Converting the normalized datainto a dataframe and storing the dataframe in
the variable wbcd_n
wbcd_n <- data.frame(wbcd_n_L)

#Printing the first 3 rows and first 4 columns of the dataframe we just
normalzed
wbcd_n[1:3, 1:4]

##   radius_mean texture_mean perimeter_mean  area_mean
## 1   0.2526859    0.0906324      0.2422777 0.13599152
## 2   0.1712812    0.3124789      0.1761454 0.08606575
## 3   0.1921056    0.2407846      0.1874784 0.09743372

#Adding id labels as rownames to keep a track of the patient's data
rownames(wbcd_n) <- wisc_bc_df$id

#Isolating the class labels
BM_class <- wisc_bc_df[,2]
```

```r
#Setting the name for each object as per ids
names(BM_class) <- wisc_bc_df$id

#Printing the first three rows of the inter list
BM_class[1:3]

## 87139402  8910251   905520
##   benign   benign   benign
## Levels: benign malignant
```

Creating the training set and test validation datasets:

We need to split the data for training the model and for the verification of that model. A reasonabele balance is 2/3 for training and 1/3 for validation of the model we trained

```r
#Getting the count of the number of rows in the dataset
nrow(wisc_bc_df)

## [1] 569

#Randomly shuffling the indexes of the dataset
rand_permute <- sample(x=1:569, size = 569)

#Prining the first 5 elements of the variable we just created
rand_permute[1:5]

## [1]   9 125 416 352 225

#saving the random set of indexes so as to make sure each time we run it runs
using the same random vakues as now
#save(rand_permute, file = 'rand_permute.RData')

#Used for reloading the random values
#load("rand_permute.RData")

#Storing the id column of the dataframe as per the indexes of the rendom
number in the variable
all_id_random <- wisc_bc_df[rand_permute, "id"]

#Calculating 1/3rd of the data which comes around 189 which is the number we
will be using for spliting the data
569/3

## [1] 189.6667
```

Now we split the data in to two groups one of those is training and the other one is for validation purpose

```r
#Converting all the Ids till 189 in to a character
validate_id <- as.character(all_id_random[1:189])
```

```r
#Converting the remaining ids to character
training_id <- as.character(all_id_random[190:569])

#Storing the training data in the variable
wbcd_train <- wbcd_n[training_id,]

#Storing the validation data in the variable
wbcd_val <- wbcd_n[validate_id,]

#Storing the diagnosis for training data
BM_class_train <- BM_class[training_id]

#Storing the diagnosis for Validation data
BM_class_val <- BM_class[validate_id]

#Getting the count of tumour in training dataset
table(BM_class_train)

## BM_class_train
##    benign malignant
##       240       140
```

Loading package class since KNN algorithm is implemented in that package

```r
#Importing the package class so as to implement KNN
library(class)
```

In order to calculate K We calculate the square root of the Training set

```r
#Calculating the square root of the training set
sqrt(nrow(wbcd_train))

## [1] 19.49359

#Thus we settle to a k value of 19
k <- 19

#Applying knn algorithm and storing the predicted results in variable
knn_predict <- knn(wbcd_train,wbcd_val,BM_class_train, k=19)

#Printing first 3 values of the predicted values
knn_predict[1:3]

## [1] benign    benign    malignant
## Levels: benign malignant

#Checking actual values with predicted values
table(knn_predict,BM_class_val)

##              BM_class_val
## knn_predict benign malignant
```

```
##   benign       115         7
##   malignant      2        65
```

```r
#Aligning the table with the probable values for every value=value/sumof values
prop.table(table(knn_predict, BM_class_val))
```

```
##              BM_class_val
## knn_predict      benign  malignant
##   benign    0.60846561 0.03703704
##   malignant 0.01058201 0.34391534
```

Testing the algorithm for different values of k

```r
#Verification for k=3,7,11,31
knn_predict_3 <- knn(wbcd_train, wbcd_val, BM_class_train, k = 3)
knn_predict_7 <- knn(wbcd_train, wbcd_val, BM_class_train, k = 7)
knn_predict_11 <- knn(wbcd_train, wbcd_val, BM_class_train, k = 11)
knn_predict_31 <- knn(wbcd_train, wbcd_val, BM_class_train, k = 31)

#Tabular format of actual vs predicted for 3,7,11,31
table(knn_predict_3,BM_class_val)
```

```
##                BM_class_val
## knn_predict_3 benign malignant
##     benign       117         4
##     malignant      0        68
```

```r
table(knn_predict_7,BM_class_val)
```

```
##                BM_class_val
## knn_predict_7 benign malignant
##     benign       117         3
##     malignant      0        69
```

```r
table(knn_predict_11,BM_class_val)
```

```
##                 BM_class_val
## knn_predict_11 benign malignant
##      benign       115         3
##      malignant      2        69
```

```r
table(knn_predict_31,BM_class_val)
```

```
##                 BM_class_val
## knn_predict_31 benign malignant
##      benign       115         7
##      malignant      2        65
```

The best we could get was with k=3.

Improving the analysis

```r
#Printing the names of all the columns in the dataframe
names(wbcd_train)

##  [1] "radius_mean"       "texture_mean"      "perimeter_mean"
##  [4] "area_mean"         "smoothness_mean"   "compactness_mean"
##  [7] "concavity_mean"    "points_mean"       "symmetry_mean"
## [10] "dimension_mean"    "radius_se"         "texture_se"
## [13] "perimeter_se"      "area_se"           "smoothness_se"
## [16] "compactness_se"    "concavity_se"      "points_se"
## [19] "symmetry_se"       "dimension_se"      "radius_worst"
## [22] "texture_worst"     "perimeter_worst"   "area_worst"
## [25] "smoothness_worst"  "compactness_worst" "concavity_worst"
## [28] "points_worst"      "symmetry_worst"    "dimension_worst"

#Applying linear regression on radius_mean against BM_class_train
lm_1 <- lm(radius_mean~BM_class_train, data = wbcd_train)

#Printing the summary of the model
summary(lm_1)

##
## Call:
## lm(formula = radius_mean ~ BM_class_train, data = wbcd_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.28284 -0.07636  0.00334  0.07193  0.47442
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.246183   0.007077   34.78   <2e-16 ***
## BM_class_trainmalignant 0.246745   0.011660   21.16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1096 on 378 degrees of freedom
## Multiple R-squared:  0.5423, Adjusted R-squared:  0.5411
## F-statistic: 447.8 on 1 and 378 DF,  p-value: < 2.2e-16

#Printing all the names of the columns in the dataframe
names(summary(lm_1))

##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"

#Printing the fstatistic of the model
summary(lm_1)$fstatistic

##    value    numdf    dendf
## 447.8369   1.0000 378.0000
```

```r
#Printing the fstatistics value
summary(lm_1)$fstatistic[1]

##    value
## 447.8369
```

In order to store the fstatistic value for all the 30 variables we need a vector

```r
#Creating a null numeric vector to store values for fstatistic for 30
variables
exp_var_fstat <- as.numeric(rep(NA, times=30))

#Assigning the names to the null vector as those of train dataframe
names(exp_var_fstat) <- names(wbcd_train)

#Storing the value of fstatistic in to the vector for radius_mean
exp_var_fstat["radius_mean"] <- summary(lm(radius_mean~BM_class_train,
data=wbcd_train))$fstatistic[1]

#Storing the value of fstatistic in to the vector for texture_mean
exp_var_fstat["texture_mean"] <- summary(lm(texture_mean ~ BM_class_train,
data = wbcd_train))$fstatistic[1]

#Storing the value of fstatistic in to the vector for perimeter_mean
exp_var_fstat["perimeter_mean"] <- summary(lm(perimeter_mean ~
BM_class_train, data = wbcd_train))$fstatistic[1]

#Printing the list to the console to check the values stored in it
exp_var_fstat

##       radius_mean      texture_mean    perimeter_mean         area_mean
##         447.83689          82.45248         483.79683                NA
##   smoothness_mean  compactness_mean    concavity_mean       points_mean
##                NA                NA                NA                NA
##     symmetry_mean    dimension_mean         radius_se        texture_se
##                NA                NA                NA                NA
##      perimeter_se           area_se     smoothness_se    compactness_se
##                NA                NA                NA                NA
##      concavity_se         points_se       symmetry_se      dimension_se
##                NA                NA                NA                NA
##      radius_worst     texture_worst   perimeter_worst        area_worst
##                NA                NA                NA                NA
##  smoothness_worst compactness_worst   concavity_worst      points_worst
##                NA                NA                NA                NA
##    symmetry_worst   dimension_worst
##                NA                NA
```

If we look above it seems cumbersome to calculate fstatistic for each and every variable instead we loop through variables and get the fstatistic value for all the variables.

```r
#Storing the names of the column in the variable
exp_vars <- names(wbcd_train)

#Storing numeric NA values in the variable
exp_var_fstat <- as.numeric(rep(NA,times=30))

#Setting the names of the object
names(exp_var_fstat) <- exp_vars

#for(j in 1:length(exp_vars)) {
  #Storing the value of fstatistic in vector
 # exp_var_fstat[exp_vars[j]] <-
summary(lm(exp_vars[j]~BM_class_train,data=wbcd_train))$fstatistic[1]
#}

#modifying the formula and running the loop
for (j in 1:length(exp_vars)) {
 exp_var_fstat[exp_vars[j]] <-
 summary(lm(as.formula(paste(exp_vars[j], " ~ BM_class_train")),data =
wbcd_train))$fstatistic[1]
}

#Printing the values of F statistics for each variable
exp_var_fstat

##       radius_mean      texture_mean    perimeter_mean         area_mean
##      447.83689012       82.45248345      483.79682725      408.60991943
##   smoothness_mean  compactness_mean    concavity_mean       points_mean
##       44.35789534      199.20199503      397.70754462      585.03725556
##     symmetry_mean    dimension_mean         radius_se        texture_se
##       49.37229799        0.00617121      208.82929366        0.11092834
##      perimeter_se           area_se     smoothness_se    compactness_se
##      202.70057735      190.11755234        3.72994592       43.44748749
##      concavity_se         points_se       symmetry_se      dimension_se
##       65.79806219      114.17389100        0.25606088        2.65063447
##      radius_worst     texture_worst   perimeter_worst        area_worst
##      594.03870393      106.65887206      610.51272899      456.84315676
##  smoothness_worst compactness_worst   concavity_worst      points_worst
##       65.01307321      195.74005256      314.49050519      658.69604540
##    symmetry_worst   dimension_worst
##       78.40777632       41.07914229

#Easier way of doing what is done above
exp_var_fstat2 <- sapply(exp_vars, function(x){
  summary(lm(as.formula(paste(x, "~BM_class_train")),data =
wbcd_train))$fstatistic[1]
})

#Printing the values in the variable
exp_var_fstat2
```

```
##       radius_mean.value      texture_mean.value    perimeter_mean.value 
##            447.83689012             82.45248345            483.79682725 
##         area_mean.value   smoothness_mean.value  compactness_mean.value 
##            408.60991943             44.35789534            199.20199503 
##     concavity_mean.value       points_mean.value     symmetry_mean.value 
##            397.70754462            585.03725556             49.37229799 
##     dimension_mean.value         radius_se.value        texture_se.value 
##              0.00617121            208.82929366              0.11092834 
##       perimeter_se.value           area_se.value     smoothness_se.value 
##            202.70057735            190.11755234              3.72994592 
##     compactness_se.value      concavity_se.value          points_se.value 
##             43.44748749             65.79806219            114.17389100 
##        symmetry_se.value      dimension_se.value      radius_worst.value 
##              0.25606088              2.65063447            594.03870393 
##      texture_worst.value   perimeter_worst.value        area_worst.value 
##            106.65887206            610.51272899            456.84315676 
##   smoothness_worst.value compactness_worst.value   concavity_worst.value 
##             65.01307321            195.74005256            314.49050519 
##       points_worst.value     symmetry_worst.value   dimension_worst.value 
##            658.69604540             78.40777632             41.07914229

#Assigning names to the vector
names(exp_var_fstat2) <- exp_vars

#Stores a list of dataframes for a particular variable
wbcd_df_L <- lapply(exp_vars, function(x) {
 df <- data.frame(sample = rownames(wbcd_train),
                  variable = x,
                  value = wbcd_train[,x],
                  class = BM_class_train)
 df
})

#Printing the head of the
head(wbcd_df_L[[1]])

##          sample    variable     value     class
## 9112367 9112367 radius_mean 0.2948081    benign
## 877486   877486 radius_mean 0.5773581 malignant
## 871201   871201 radius_mean 0.5967627 malignant
## 8810436 8810436 radius_mean 0.3923044    benign
## 906024   906024 radius_mean 0.2706706    benign
## 9113239 9113239 radius_mean 0.2962279    benign

# Assigning names to the dataframe
names(wbcd_df_L) <- exp_vars
```

Using laply function from plyr library to get fstatistic value

```
#importing the required libraries
library(plyr)
```

```r
#applying linear regression on each df ans storing the values of those in a variable
var_sig_fstats <- laply(wbcd_df_L, function(df){
  fit <- lm(value~class, data=df)
  f <- summary(fit)$fstatistic[1]
  f
})

#Assigning the names to the variable
names(var_sig_fstats) <- names(wbcd_df_L)

#Printing the first 3 values from the list
var_sig_fstats[1:3]

##    radius_mean   texture_mean perimeter_mean
##      447.83689       82.45248      483.79683

#Storing the values according to descending value of fstatistics
most_sig_stats <- sort(var_sig_fstats, decreasing=T)

#Printing the first 5 values of the list
most_sig_stats[1:5]

##    points_worst perimeter_worst    radius_worst     points_mean
##        658.6960        610.5127        594.0387        585.0373
##  perimeter_mean
##        483.7968

#Printing the last 5 values of the list
most_sig_stats[25:30]

## dimension_worst   smoothness_se    dimension_se     symmetry_se
##     41.07914229      3.72994592      2.65063447      0.25606088
##      texture_se  dimension_mean
##      0.11092834      0.00617121
```

We can conclude from the above that the last variables in the list arent significant on their own. Adding them to the model will only increase the variance.

Thus we reorder the dataset as per the fstatistic values we found

```r
#Reordering the train dataframe
wbcd_train_ord <- wbcd_train[, names(most_sig_stats)]
```

Now since we have ordered the data as per the fstatistic value of the linear model, the next step is to access how many variables we shoukd consider to give the best fit hence we perform cross validation

Further now we divide the training set into size 2/3

```r
#Printing the length of the training_id
length(training_id)

## [1] 380

#Printing 2/3rd of the length
(2/3) * length(training_id)

## [1] 253.3333

#subtracting 253 from length of the ids
length(training_id)-253

## [1] 127

#creating 1000 samples of random 253 values from set and storing it in the
variable
training_family_L <- lapply(1:1000, function(j) {
 perm <- sample(1:380, size = 380, replace = F)
 shuffle <- training_id[perm]
 trn <- shuffle[1:253]
 trn
})

#Saving the randomvalues we just generated
#save(training_family_L, file='training_family_L.RData')

#Loading the values
#load("training_family_L.RData")

#Creating Validation set for each training set
validation_family_L <- lapply(training_family_L,function(x)
  setdiff(training_id,x))
```

We are all set with the requirements and will now find the optimal set of variables and optimal value for k

```r
#Creating a sequence for variables values from 3 to 29 with stepsize of 2
N <- seq(from = 3, to=29, by=2)

#Finding the square root of the length of the dataframe
sqrt(length(training_family_L[[1]]))

## [1] 15.90597

#We will vary our k from 3 to 19
K <- seq(from=3 , to=19, by=2)

#Number of choices we will validate for KNN
1000*length(N)*length(K)

## [1] 126000
```

```r
#Creating a dataframe for errors
paramter_errors_df <- data.frame(mc_index = as.integer(rep(NA,times =
126000)),
                                 var_num = as.integer(rep(NA, times =
126000)),
                                 k =as.integer(rep(NA, times = 126000)),
                                 error = as.numeric(rep(NA, times = 126000)))

#Writing test for the first 5 variables we found according to fstatistics and
with k=7
knn_test <- knn(train = wbcd_train_ord[training_family_L[[1]],1:5],
                test = wbcd_train_ord[validation_family_L[[1]], 1:5],
                cl = BM_class_train[training_family_L[[1]]], k = 7)

#Printing the first 3 values of the test we ran above
knn_test[1:3]

## [1] benign    benign    malignant
## Levels: benign malignant

#Storing acutal vs predicted in variable tbl_test
tbl_test <- table(knn_test,BM_class_train[validation_family_L[[1]]])

#Printing the result
tbl_test

##
## knn_test    benign malignant
##   benign        83         5
##   malignant      1        38

#Calculating total error and dividing it with the total length of the
Validation family
err_rate <- (tbl_test[1, 2] + tbl_test[2,1])/length(validation_family_L[[1]])
err_rate

## [1] 0.04724409

# j = index, n = length of range of variables, k=k
# Creating a function for j,n,k
core_knn <- function(j, n, k) {
 knn_predict <- knn(train =wbcd_train_ord[training_family_L[[j]], 1:n],
                    test = wbcd_train_ord[validation_family_L[[j]], 1:n],
                    cl=BM_class_train[training_family_L[[j]]],
                    k = k)
 tbl <- table(knn_predict,BM_class_train[validation_family_L[[j]]])
 err <- (tbl[1, 2] + tbl[2, 1])/length(validation_family_L[[j]])
 err
}
```

```r
#Running a sample on the function we just created
core_knn(1, 5, 7)

## [1] 0.04724409

#to keep the track of what loop we are in
iter <- 1

#Storing start time of the system
str_time <- Sys.time()

#Looping for all 126000 values of combinations
for (j in 1:1000) {
 for (n in 1:length(N)) {
   for (m in 1:length(K)) {
     err <- core_knn(j, N[n], K[m])
     paramter_errors_df[iter, ] <- c(j, N[n], K[m], err)
     iter <- iter + 1
    }
  }
}
#Calculating total time required for running the loop
time_lapsed_for <- Sys.time() - str_time

#Saving the paramter for errors
save(paramter_errors_df, time_lapsed_for, file
="for_loop_paramter_errors.RData")

#Loading the parameter
load("for_loop_paramter_errors.RData")

#Printing the time for which the loop was running
time_lapsed_for

## Time difference of 7.024795 mins

#Merging combination of 1000 random draws with number of variables
param_df1 <- merge(data.frame(mc_index = 1:1000),data.frame(var_num = N))

#Merging the above combination with k values
param_df <- merge(param_df1, data.frame(k = K))

#We get combination of all the values
str(param_df)

## 'data.frame':    126000 obs. of  3 variables:
##  $ mc_index: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ var_num : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ k       : num  3 3 3 3 3 3 3 3 3 3 ...
```

```r
#For first 20 values
knn_err_est_df_test <- ddply(param_df[1:20, ], .(mc_index, var_num,k),
function(df) {
 err <- core_knn(df$mc_index[1], df$var_num[1], df$k[1])
 err
})
#Printing head of the error values
head(knn_err_est_df_test)

##   mc_index var_num k         V1
## 1        1       3 3 0.04724409
## 2        2       3 3 0.06299213
## 3        3       3 3 0.04724409
## 4        4       3 3 0.03937008
## 5        5       3 3 0.07086614
## 6        6       3 3 0.04724409

#Storing the start time
str_time <- Sys.time()

#Applying KNN for calculating error
knn_err_est_df <- ddply(param_df, .(mc_index, var_num, k), function(df) {
 err <- core_knn(df$mc_index[1], df$var_num[1], df$k[1])
 err
})

#Calculating the lapsed time
time_lapsed <- Sys.time() - str_time

#Storing the lapsed time
save(knn_err_est_df, time_lapsed, file = "knn_err_est_df.RData")

#Loading the Time lapsed
load("knn_err_est_df.RData")

#Printing the time lapsed on the console
time_lapsed

## Time difference of 4.18361 mins

#Printing the head of the KNN Error Estimate
head(knn_err_est_df)

##   mc_index var_num  k         V1
## 1        1       3  3 0.04724409
## 2        1       3  5 0.04724409
## 3        1       3  7 0.04724409
## 4        1       3  9 0.04724409
## 5        1       3 11 0.05511811
## 6        1       3 13 0.06299213
```

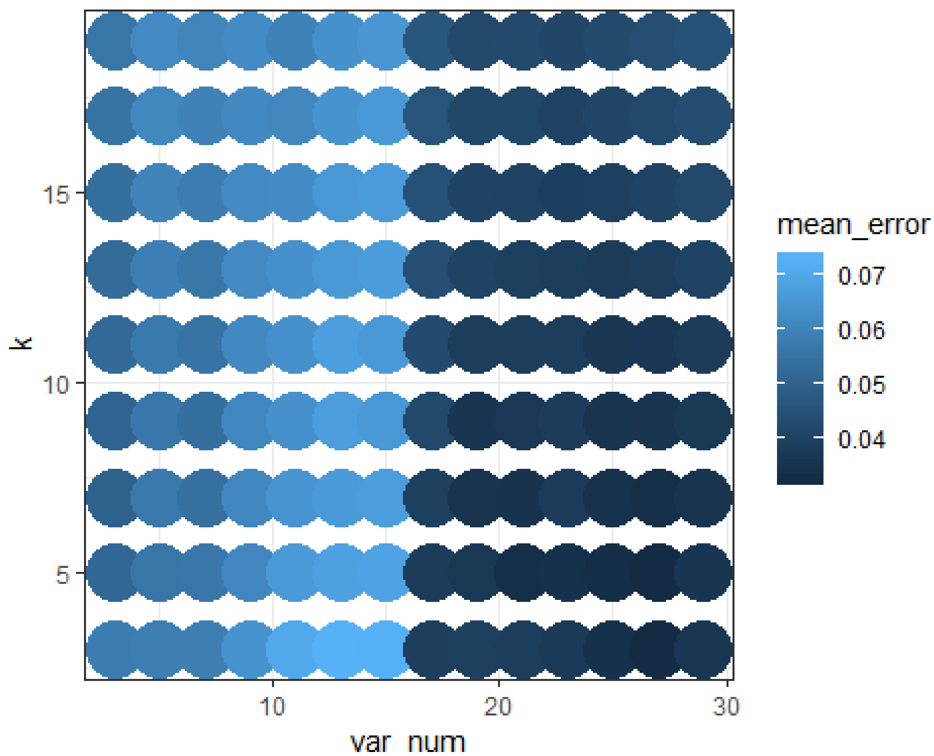
```r
#Renaming column 4 to Error
names(knn_err_est_df)[4] <- "error"
```

Now we will Get the summary performance of the statistics

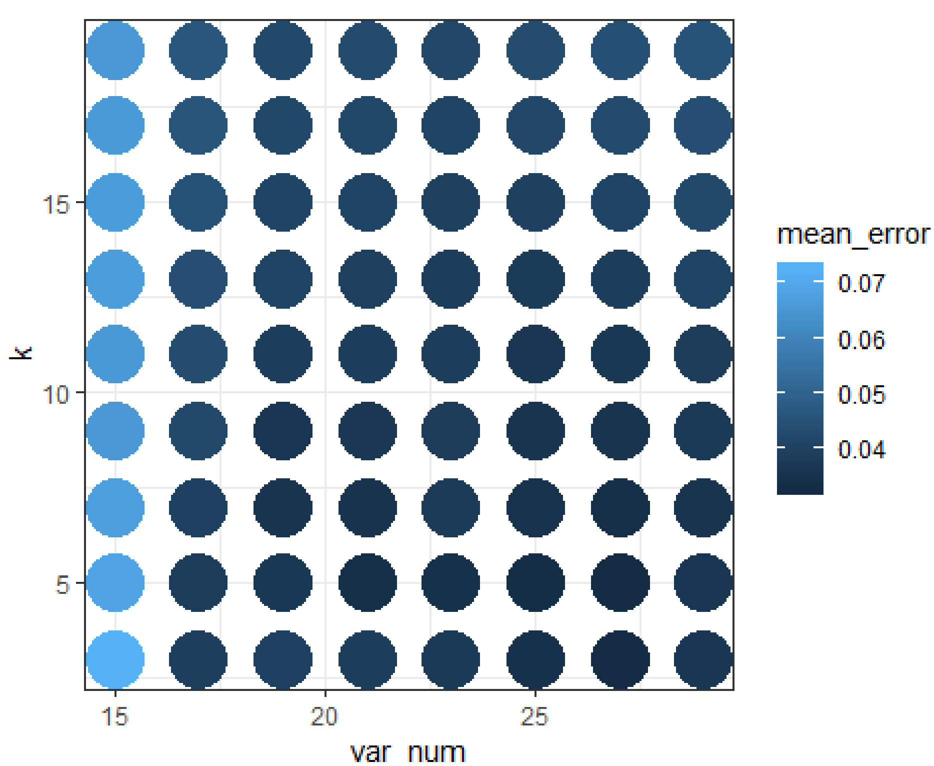
```r
#Creating subset with var num 5 and 7
mean_ex_df <- subset(knn_err_est_df, var_num == 5 & k == 7)

#Printing head of the dataframe
head(mean_ex_df)

##     mc_index var_num k      error
## 12         1       5 7 0.04724409
## 138        2       5 7 0.06299213
## 264        3       5 7 0.03937008
## 390        4       5 7 0.03149606
## 516        5       5 7 0.07874016
## 642        6       5 7 0.03937008

#Calculating mean error
mean(mean_ex_df$error)

## [1] 0.05710236

#Calculating errors for all the number of parameters and k values
mean_errs_df <- ddply(knn_err_est_df, .(var_num, k),
function(df)mean(df$error))

#Printing the head of the vector
head(mean_errs_df)

##   var_num  k         V1
## 1       3  3 0.05839370
## 2       3  5 0.05180315
## 3       3  7 0.04996063
## 4       3  9 0.05103937
## 5       3 11 0.05235433
## 6       3 13 0.05293701

#Renaming the last column to mean_error
names(mean_errs_df)[3] <- "mean_error"
```

Visualizing all the parameters performances

```r
#importing the required libraries
library(ggplot2)

#Plotting var_num against k as per mean_error
ggplot(data = mean_errs_df, aes(x = var_num, y = k, color = mean_error))+
geom_point(size = 10) + theme_bw()
```




```
#Plotting the same plot as above for k values from 15 to 29
ggplot(data = subset(mean_errs_df, var_num >= 15), aes(x = var_num,y = k,
color = mean_error)) + geom_point(size = 10) + theme_bw()
```



After looking at the plots we figure that with 19 variables and low k value the algorithm seems to work best. Thus we explore the mean_error of variables 17,19,21,25.

```
#Extracting subset with first 17 variable  for all the k values which shows mean_error
subset(mean_errs_df, var_num == 17)

##    var_num  k mean_error
## 64      17  3 0.03884252
## 65      17  5 0.03833858
## 66      17  7 0.03959843
## 67      17  9 0.04215748
## 68      17 11 0.04325197
## 69      17 13 0.04399213
## 70      17 15 0.04507087
## 71      17 17 0.04592126
## 72      17 19 0.04648819

#Extracting subset with first 19 variable  for all the k values which shows mean_error
subset(mean_errs_df, var_num == 19)

##    var_num  k mean_error
## 73      19  3 0.03959843
## 74      19  5 0.03680315
## 75      19  7 0.03555118
## 76      19  9 0.03607087
## 77      19 11 0.03872441
## 78      19 13 0.04045669
## 79      19 15 0.04086614
## 80      19 17 0.04191339
## 81      19 19 0.04255906

#Extracting subset with first 21 variable  for all the k values which shows mean_error
subset(mean_errs_df, var_num == 21)

##    var_num  k mean_error
## 82      21  3 0.03863780
## 83      21  5 0.03381890
## 84      21  7 0.03534646
## 85      21  9 0.03703150
## 86      21 11 0.03847244
## 87      21 13 0.03947244
## 88      21 15 0.04058268
## 89      21 17 0.04181102
## 90      21 19 0.04289764

#Extracting subset with first 25 variable  for all the k values which shows mean_error
subset(mean_errs_df, var_num == 25)
```

```
##     var_num  k mean_error
## 100      25  3 0.03459055
## 101      25  5 0.03340157
## 102      25  7 0.03494488
## 103      25  9 0.03577165
## 104      25 11 0.03653543
## 105      25 13 0.03807874
## 106      25 15 0.03970866
## 107      25 17 0.04150394
## 108      25 19 0.04322047
```

```r
#Printing the row with minimum mean error value
mean_errs_df[which.min(mean_errs_df$mean_error), ]
```

```
##     var_num k mean_error
## 110      27 5 0.03246457
```

By looking at the above data we can infer that the best is with 27 variables with k=3.

```r
#Printing the variables name to the console as per their fstatistics value
names(wbcd_train_ord)
```

```
##  [1] "points_worst"      "perimeter_worst"   "radius_worst"
##  [4] "points_mean"       "perimeter_mean"    "area_worst"
##  [7] "radius_mean"       "area_mean"         "concavity_mean"
## [10] "concavity_worst"   "radius_se"         "perimeter_se"
## [13] "compactness_mean"  "compactness_worst" "area_se"
## [16] "points_se"         "texture_worst"     "texture_mean"
## [19] "symmetry_worst"    "concavity_se"      "smoothness_worst"
## [22] "symmetry_mean"     "smoothness_mean"   "compactness_se"
## [25] "dimension_worst"   "smoothness_se"     "dimension_se"
## [28] "symmetry_se"       "texture_se"        "dimension_mean"
```

Validation of the final test

```r
#Sorting the variables as per how we arranged the data above
wbcd_val_ord <- wbcd_val[, names(wbcd_train_ord)]

#Applying knn algorithm for optimal values of varables and k which we found
to be 27 and 3 respectively to predict the validation set
bm_val_pred <- knn(train = wbcd_train_ord[, 1:27], wbcd_val_ord[,1:27],
BM_class_train, k = 3)

#Storing predicted values and actual values for validation set in tabular
format
tbl_bm_val <- table(bm_val_pred, BM_class_val)

#Printing the above table to console
tbl_bm_val
```

```
##             BM_class_val
## bm_val_pred benign malignant
##   benign       116         6
##   malignant      1        66
```

```r
#Calculating standard error which is summation of error values divided by
total number of values
(val_error <- tbl_bm_val[1, 2] + tbl_bm_val[2,1])/length(BM_class_val)
```

```
## [1] 0.03703704
```

Speeding up the KNN algorithm

```r
#Installing the required packages
#install.packages("doParallel")
#install.packages("doSNOW")

#Importing the required libraries
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```r
library(doSNOW)
```

```
## Loading required package: snow

##
## Attaching package: 'snow'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, clusterSplit, makeCluster,
##     parApply, parCapply, parLapply, parRapply, parSapply,
##     splitIndices, stopCluster
```

```r
#register the parallel backend with the foreach package
registerDoParallel()

#Printing the number of cores we are using
getDoParWorkers()
```

```
## [1] 3
```

```r
#Storing sys.time
str_time <- Sys.time()

#Running knn test parallely on the number of cores we found above
knn_err_est_df_par <- ddply(param_df, .(mc_index, var_num, k), function(df) {
```

```r
 err <- tryCatch({core_knn(df$mc_index[1], df$var_num[1], df$k[1])},
error=function(e) {
              class(e) <- class(simpleWarning(''))
              warning(e)
              NULL})
 err
}, .parallel = TRUE)
```

```
## Warning: <anonymous>: ... may be used in an incorrect context:
'.fun(piece, ...)'

## Warning: <anonymous>: ... may be used in an incorrect context:
'.fun(piece, ...)'
```

```r
#Storing the time lapsed i.e time for actual run
time_lapsed_par <- Sys.time() - str_time

#Saving the value of the run
save(knn_err_est_df_par, time_lapsed_par, file ="knn_err_est_df_par.RData")

#Loading the data
load("knn_err_est_df_par.RData")

#Printing the optimized time on console
time_lapsed_par
```

```
## Time difference of 54.51312 secs
```