Machine Learning Project on Detection of Cancer Type Based on Gene Expression

RISHABH NARULA

4/1/2020

Data Acquisition

Setting up all the libraries and primary setup chunk for this project.

Data Acquisition

First step is Data acquisition in which we gather data from the web source, which is called "UCI Machine learning repository". This data set is present on this website as "Gene expression cancer RNA-Seq Data Set". Which is basically gene expression values from RNA-Seq (HiSeq) PANCAN data set. It is a random extraction of gene expression of patients having different type of tumor : BRCA, KIRC, COAD, LUAD and PRAD. This data set is good for classification models. It include 20531 genes in coloumns and 801 patients in rows. I will try to predict the type of cancer tumor from above classes based on gene expression values.

For further usage in this project I need to transpose the data from rows to coloumn and coloumns to rows so, I used t() function to transpose the data and also set the rows.names() as patients names. There is two files in this data set first file with all the data and second file with tumor type of each patient. So setting row.names as patients names in both files will help to merge files later.

```
# loading the data with setting row.names as patients names
data <- read.csv(file = "D:/Download/TCGA-PANCAN-HiSeq-801x20531/TCGA-PANCAN-HiSeq-801x20531/data.csv",

# transposing the data for further use
data <- as.data.frame(t(data))

# getting second file with specific tumore type to each patient
cancerMatrix <- read.csv(file = "D:/Download/TCGA-PANCAN-HiSeq-801x20531/TCGA-PANCAN-HiSeq-801x20531/la
```

Data Exploration

Exploratory data points

Exploration of data points using different graphs and analysis to check for data skewness and distribution. First we will start with Voom transformation. We have our data matrix, Voom will automatically adjust library sizes using the norm.factors. The Voom transformation uses experiment design matrix, and produce EList obejcts. It plots sqrt(standard deviation on Y-axis) and log2 of data on X-axis with Mean-Variance trend line along the data points.

For further exploration we divide the data as our data has 20000 coloumns and 800 rows, its really difficult to explore all the coloumns. So we sorted the data based largest sum of gene expression along with rows and coloumns and divided the top 50 genes and top 50 patients with largest sum in two different vectors.

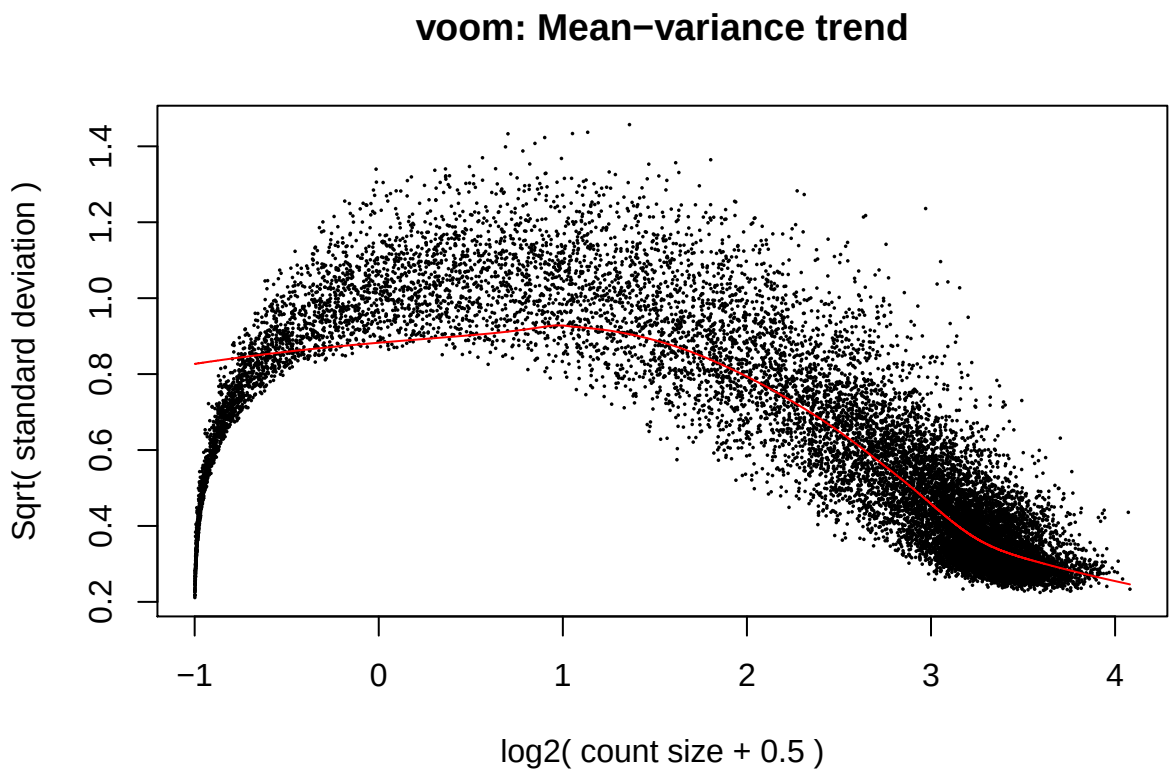
Then we plotted pheatmap for top 50 patients and top 50 genes, which represents top 50 patients in X-axis and top 50 genes in Y-axis, along with branched comparison, with colour scheme as red colour for high gene expression value and blue colour for low gene expression values. It signifies the massive distribution of gene expression values within one patient and one gene, as we can see in the pheatmap.

Next, we plotted box plot for similar top 50 genes and patients, to visualize the mean, median, and 75% quartile distribution along those gene expression values. As we can see median and quartile range also varries highly, within patients.

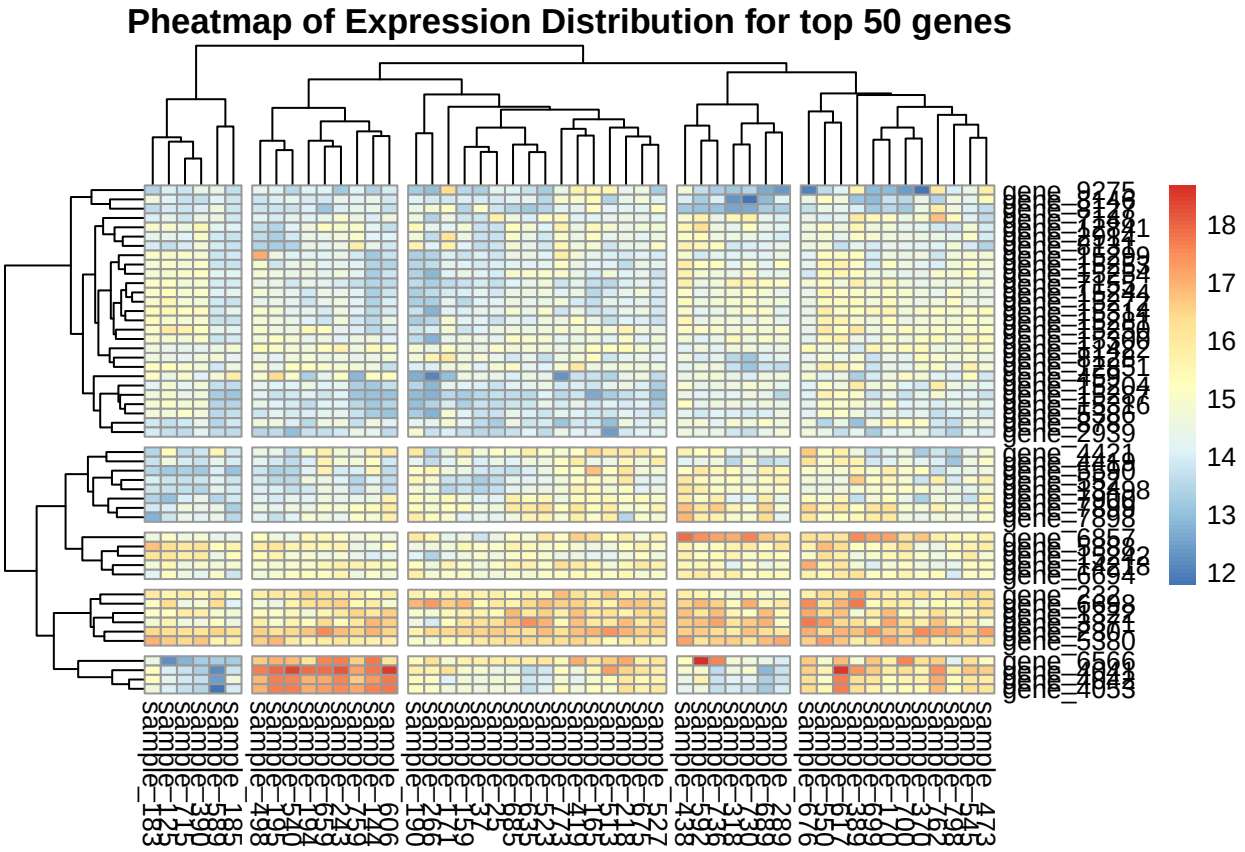
For our next analysis we plotted MDS chart with colour code according to different classes of cancer, to visualize potential clusters within the data. As we know for the fact that there can be 5 clusters, as there are 5 classes of tumor present in data. But due to high skewness of data, clsutering at this stage is not possible, so we used MDS plot which is basically MultiDimensional scaling plot. It is the analysis of principle component analysis to determine greatest sources of variation in data. The distinct far away points in plot can lead to variation in data.

In the end, we plotted simple histogram charts for top 5 patients, as data is highly right skewed, and does not follow normal distribution, we do need to transform the data for further classification ussage.

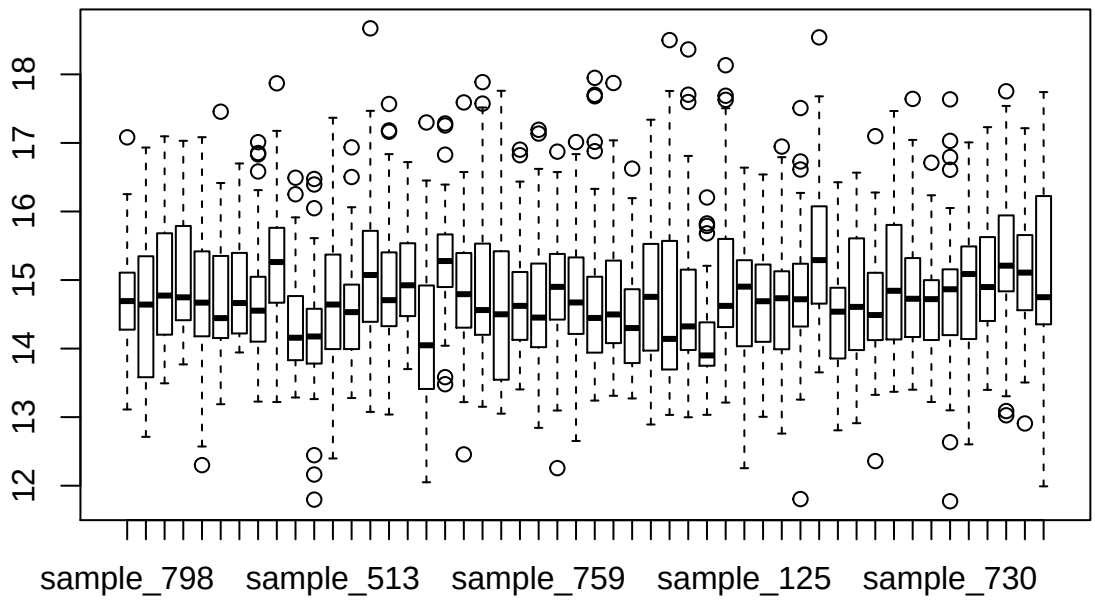
```
# Source code for some charts -> https://combine-australia.github.io/RNAseq-R/06-rnaseq-day1.html

# voom mean-variance plot
v <- voom(data,plot = TRUE)
```



```r
# getting largest gene expression genes
high.gene <- order(apply(data[1:nrow(data),1:ncol(data)], 1, sum), decreasing = TRUE)

# getting largest gene expression patients
high.patient <- order(apply(data[1:nrow(data),1:ncol(data)], 2, sum), decreasing = TRUE)

# plotting pheatmap
pheatmap(data[high.gene[1:50],high.patient[1:50]], cutree_rows = 5, cutree_cols = 5, main = "Pheatmap o
```



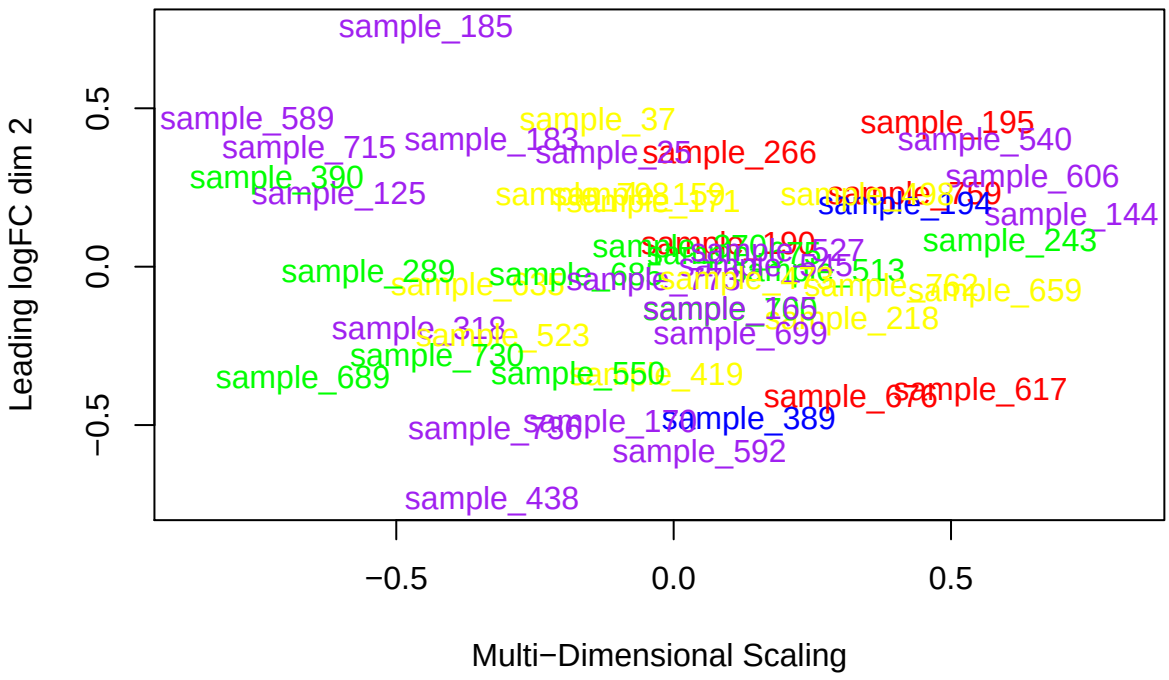

```r
# plotting boxplot
boxplot(data[high.gene[1:50],high.patient[1:50]], main = "Boxplot of Expression Distribution for top 50
```



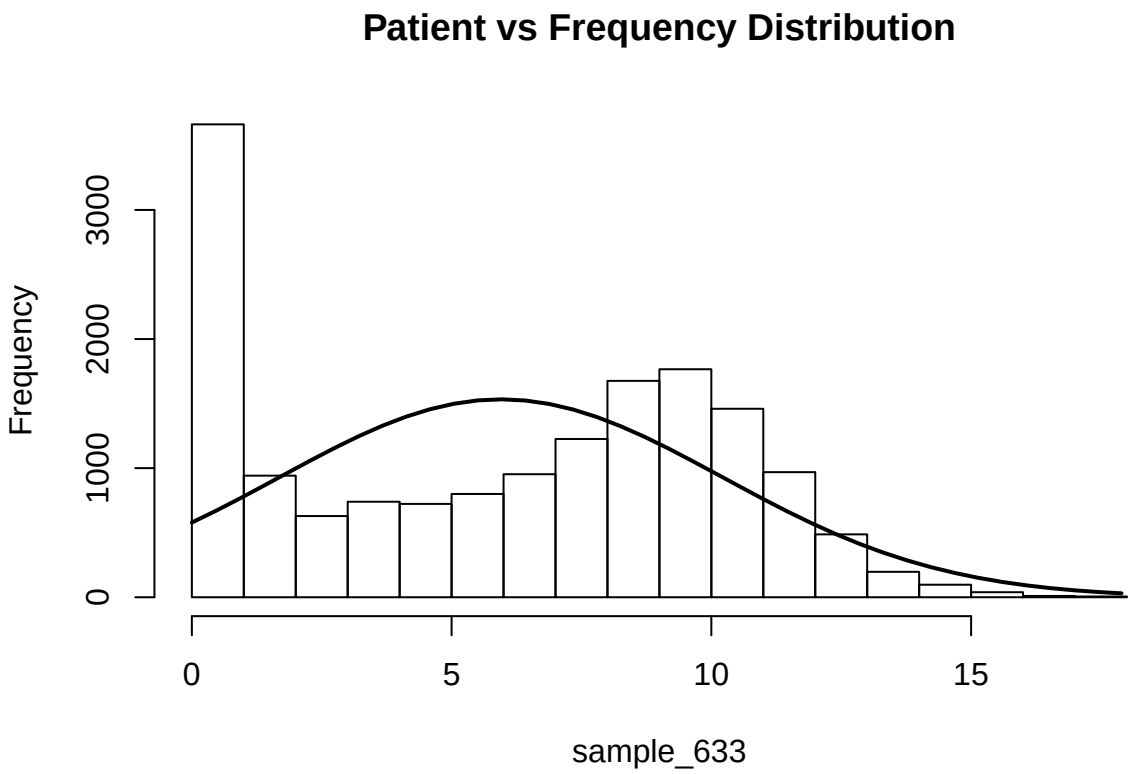

```
# plotting Md distribution according to colour code of different tumor
print("MultiDimensional Scaling Plot")
```

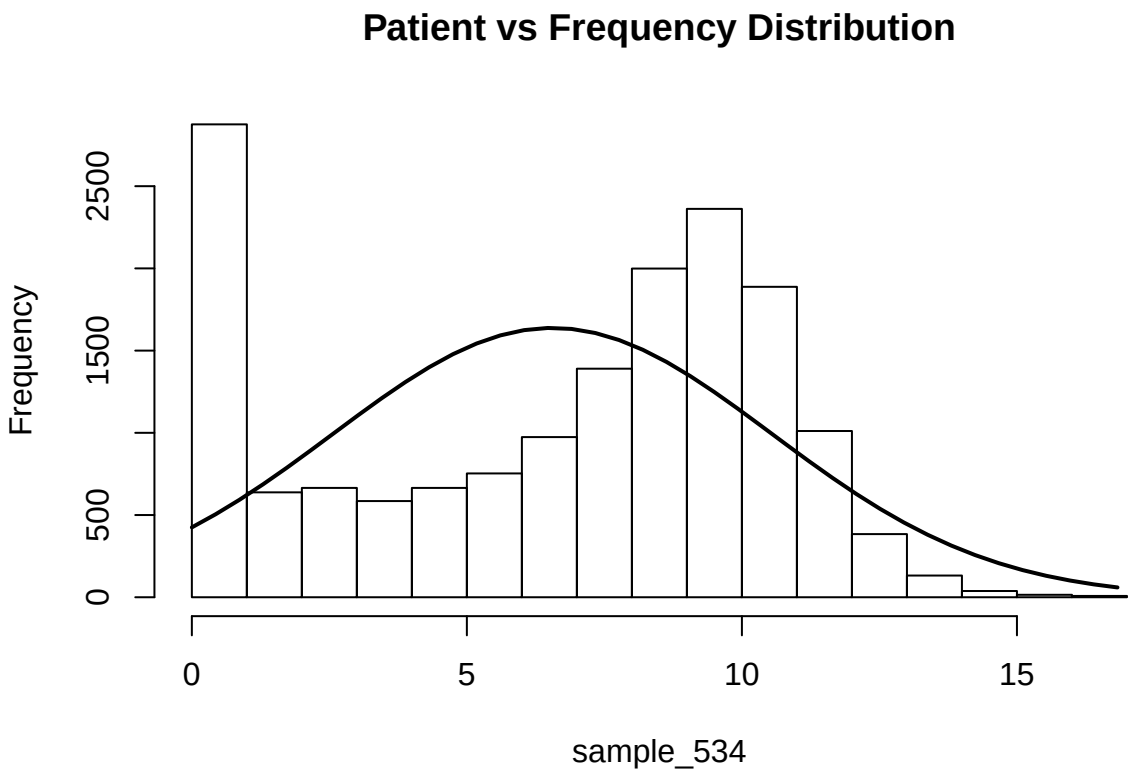
```
## [1] "MultiDimensional Scaling Plot"
```

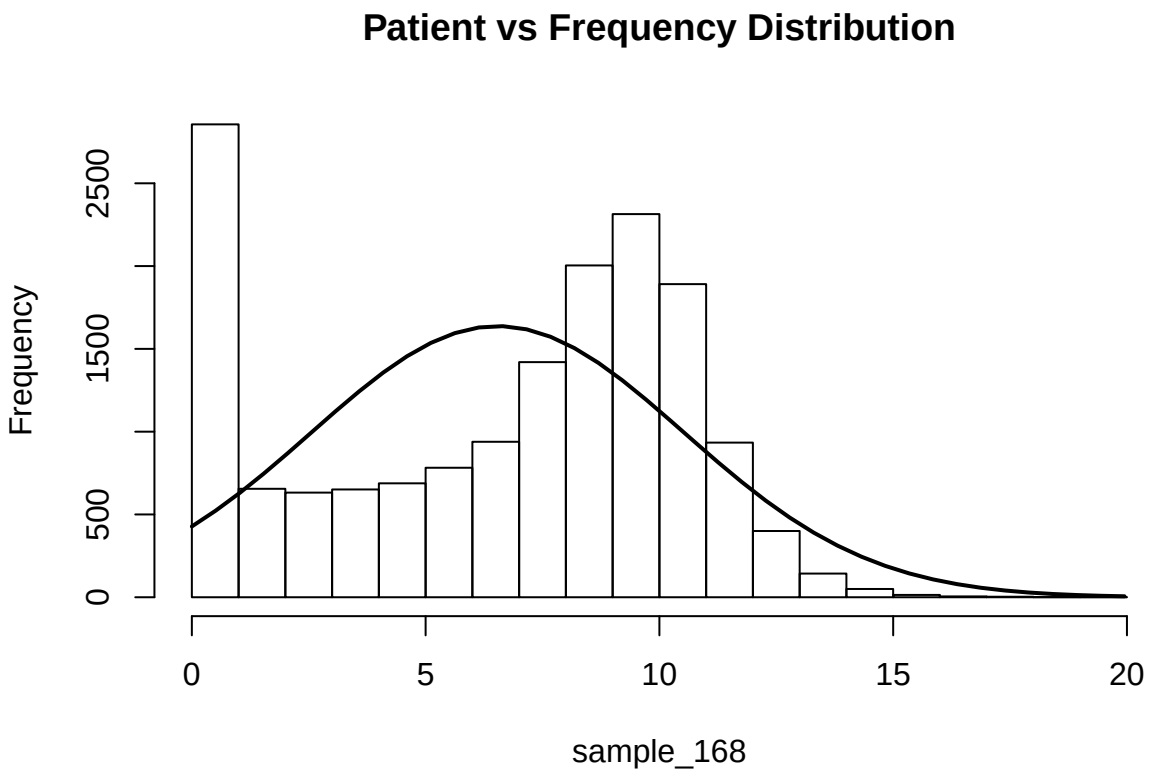
```
col.cell <- c("purple","blue","green","red","yellow","black")[cancerMatrix$Class]
plotMDS(data[high.gene[1:50],high.patient[1:50]], col=col.cell, xlab = "Multi-Dimensional Scaling")
```

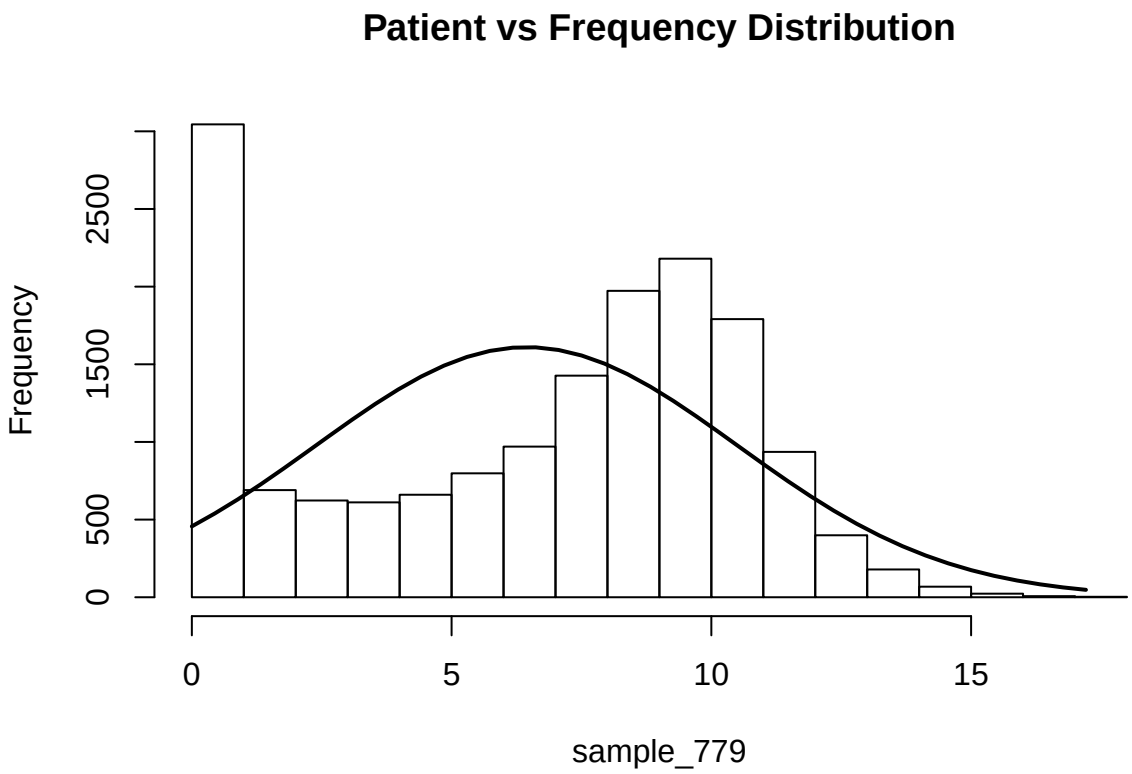


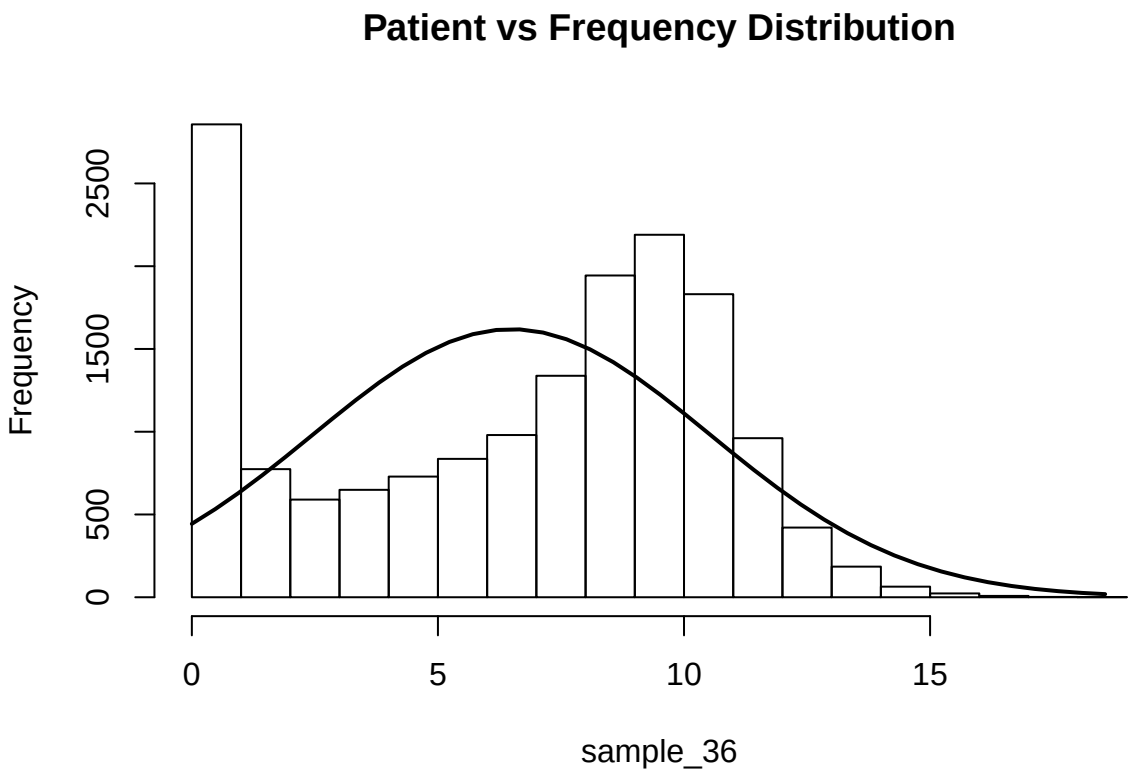

```r
# plotting histogram for checking skewness in data along with normal curve line
for (i in high.patient[1:5]){
  h <- hist(data[,high.patient[i]], xlab = colnames(data[high.patient[i]]), main="Patient vs Frequency
  xfit <- seq(min(data[,high.patient[i]]), max(data[,high.patient[i]]), length = 40)
  yfit <- dnorm(xfit, mean = mean(data[,high.patient[i]]), sd = sd(data[,high.patient[i]]))
  yfit <- yfit * diff(h$mids[1:2]) * length(data[,high.patient[i]])
  lines(xfit, yfit, col = "black", lwd = 2)
}
```


Patient vs Frequency Distribution
Frequency
0
1000
2000
3000
0
5
10
15
sample_633


Patient vs Frequency Distribution
Frequency
0
500
1500
2500
0
5
10
15
sample_534


Patient vs Frequency Distribution
Frequency
0
500
1500
2500
0
5
10
15
20
sample_168


Patient vs Frequency Distribution
Frequency
0
500
1500
2500
0
5
10
15
sample_779



Data Exploration

Detection of Outliers

Next step towards data exploration is outlier detection, for outlier detection we used z-score method, that is any point beyond plus-minus two standard deviation is considered as outlier. We are going to print the outlier row number and total count of rows of outliers for first 10 coloumns out of 800. We are detecting outliers in this step, but we are not going to remove them right now as for data cleaning process, we are going to use Differential Expression method in Bioinformatics.

```
print("Number of outliers in first 10 coloumns")
```

```
## [1] "Number of outliers in first 10 coloumns"
```

```
# creating for loop to loop over all coloumns
for (i in 1:ncol(data)){
  # getting mean
  meanCom <- mean(data[,i])
  sdCom <- sd(data[,i])
  # getting 2 * standard devaition
  sdCom <- sdCom * 2

  # printing first 10 coloumns
  if(i < 10 ){
```

```
    print(colnames(data[i]))
    # printing row numbers which include outliers in each column
    print(length(which(data[,i] > meanCom + sdCom | data[,i] < meanCom - sdCom)))
  }

}
```

```
## [1] "sample_0"
## [1] 93
## [1] "sample_1"
## [1] 53
## [1] "sample_2"
## [1] 52
## [1] "sample_3"
## [1] 57
## [1] "sample_4"
## [1] 56
## [1] "sample_5"
## [1] 92
## [1] "sample_6"
## [1] 99
## [1] "sample_7"
## [1] 64
## [1] "sample_8"
## [1] 44
```

Data Exploration

Correlation Analysis

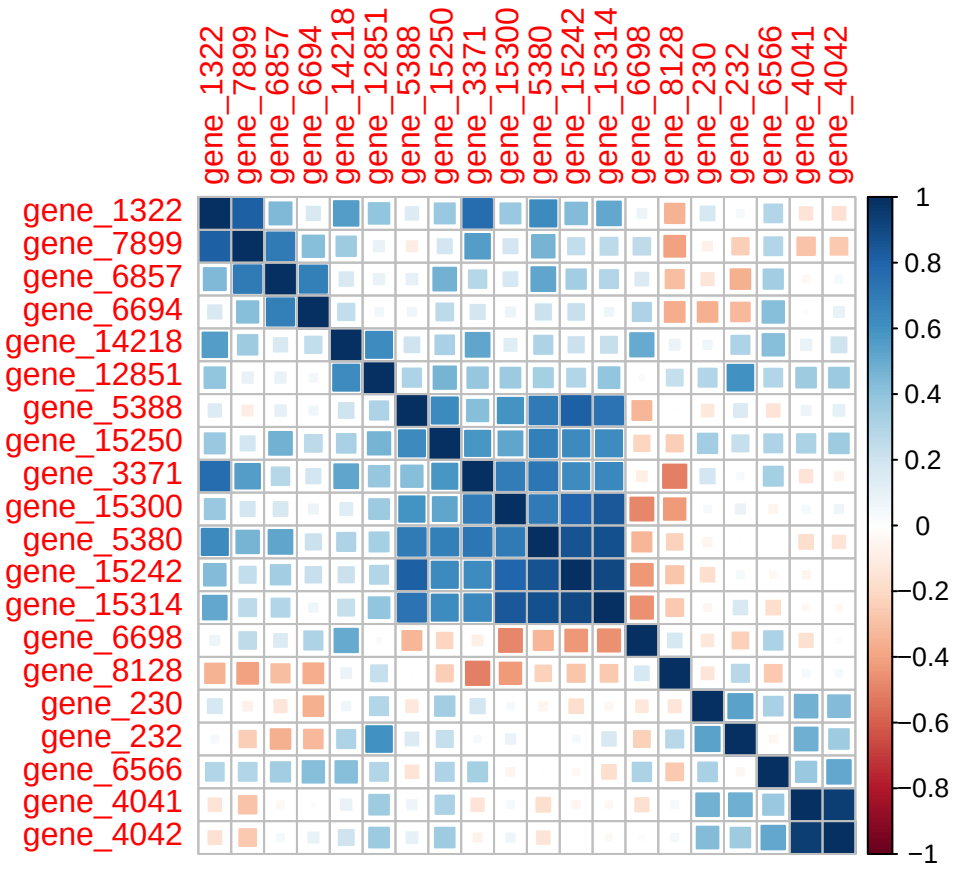
For correlation plotting, we used corrplot() function to plot the correaltion between top 20 genes and top 20 patients, with pairwise comparison. It creates a square matrix with colour code as blue colour refers to highly correlated, as 1 and red colour as negatively correlated. And we can see in graph below, there are many genes and patients which have high correlation in the matrix.

For collinearity we used pairs.panel, to create collinearity matrix for top 10 genes and patients, we can see there is no high collinearity between top 10 genes and top 10 patients expression.

```
# corrplot with multiple correlation plotting, using square method and order hclust
print("Correlation Matrix")
```

```
## [1] "Correlation Matrix"
```

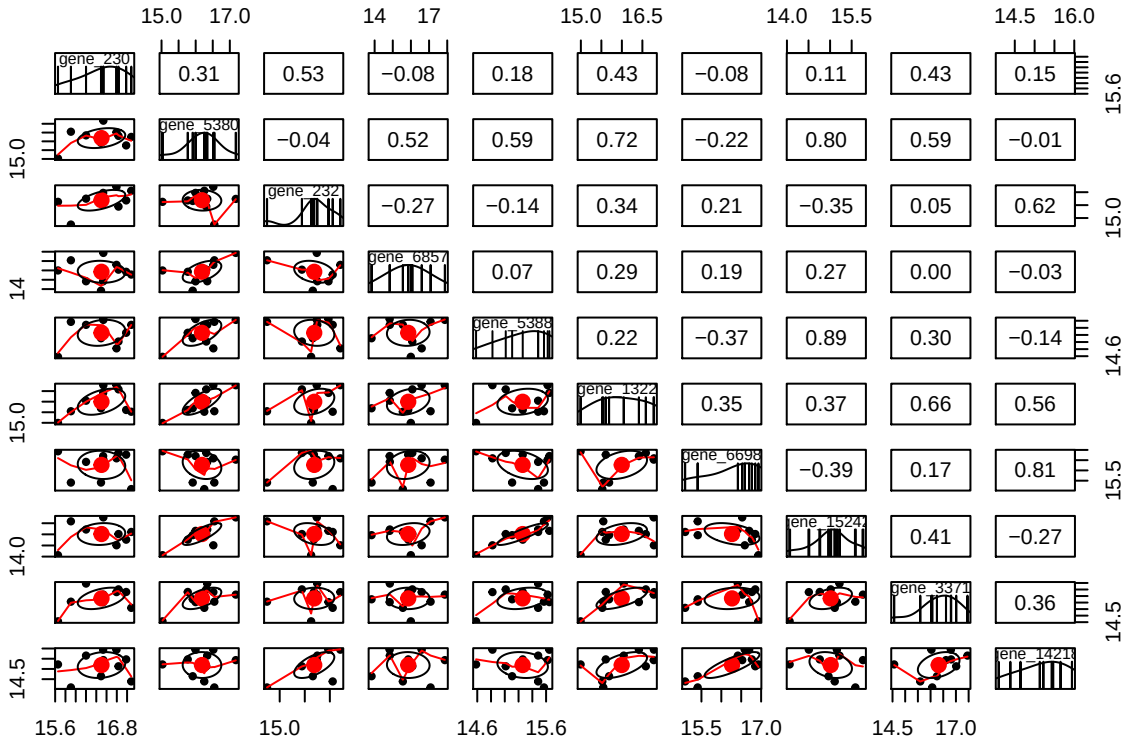
```
corrplot(cor(as.data.frame(t(data[high.gene[1:20],high.patient[1:20]])), use = "pairwise.complete.obs")
```




```
# collinearity matrix with pairs.panel for top 10 genes and patients
print("Collinearity Matrix")
```

```
## [1] "Collinearity Matrix"
```

```
pairs.panels(as.data.frame(t(data[high.gene[1:10],high.patient[1:10]])))
```



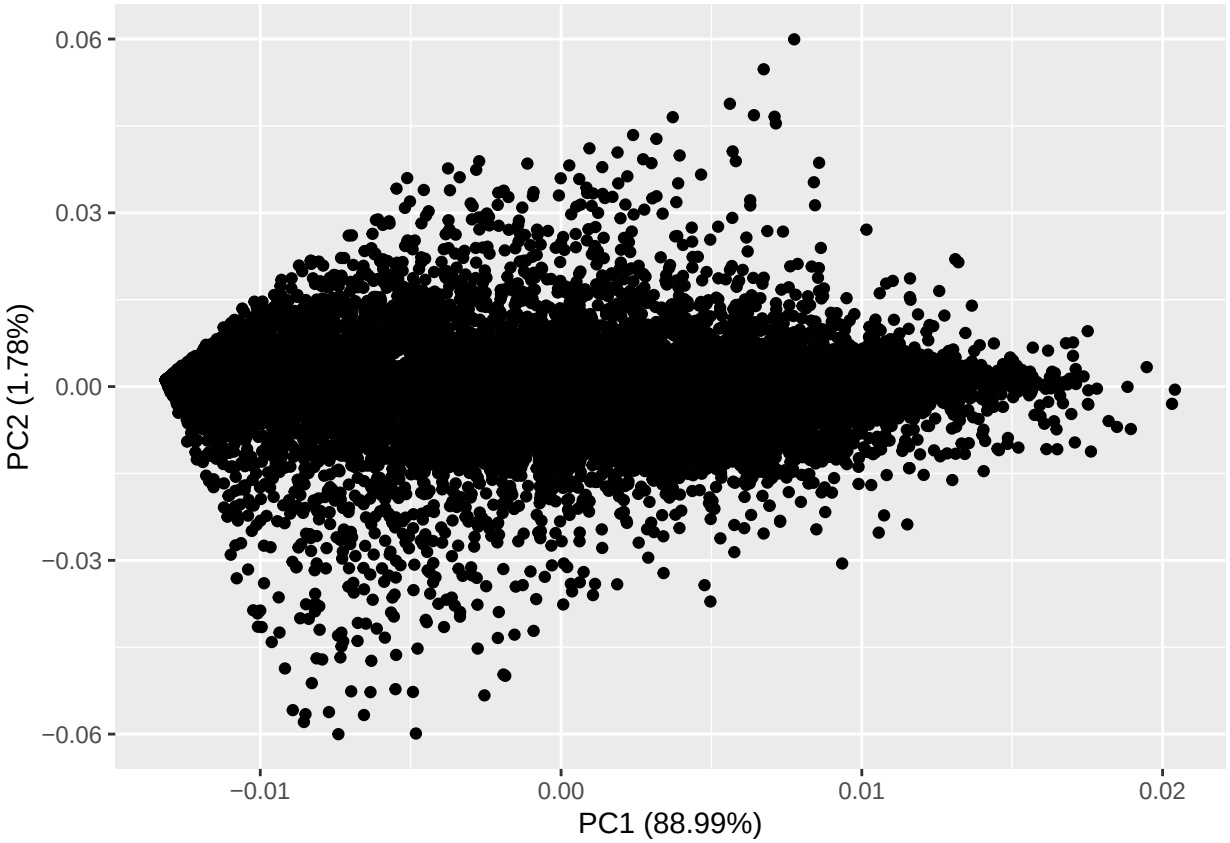
Data Cleaning and shaping

PCA

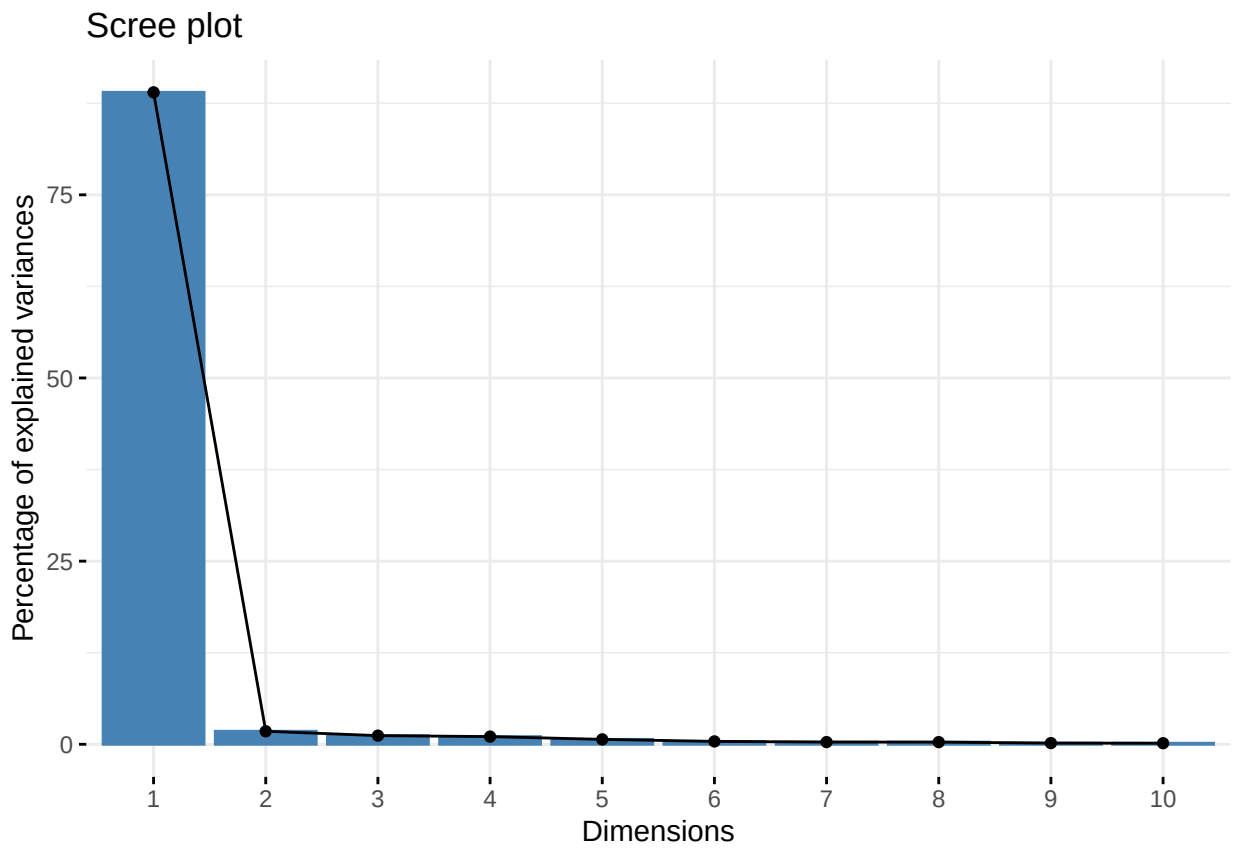
For data cleaning first we start with analysing data points using Principle component analysis, to get an idea of out of 20000 genes, how many of them actually relevant. We used prcomp() function on whole dataset, and then we plotted the pca, using autoplot().

After plotting PCA we can see data is highly skewed and inbalanced, with lot of excess of features, so we need to clean the data, as the data points are totally clustered without any distribution, and Scree-plot is also imbalanced to analyse any further. So, first we try to select important genes out of 20000, using Differentail Expression tools.

```
# pca of the whole dataset
pca.data <- prcomp(data)

# plotting pca
autoplot(pca.data, data = data)
```




```
# plotting scree-plot
fviz_eig(pca.data)
```



Data Cleaning and shaping

Data Cleaning

Applying Differential expression along the genes for data cleaning, to select the relevant genes whose expression actually affects the factor of disease. We are using EdgeR library for this. First we created a factor list of all classes of cancer along with patients ratio.

Then we applied **model.matrix feature for dummy coding** the factors of tumor classes and storing them in matrix format. We used DGEList feature to provide data and align the tumor type by groups. Then we used estimateGLMTrend feature to evaluate different GLM possibilities in dataset. We passed this variable to estimateGLMTagDisp function to tag all the stages of GLM with respect to different groups.

After this we used glmFit feature to create glmFit variable. Then we passed fir variable along in glmRT with coef=2, following the process by using topTags function to extract topTags from glmRT variable. We stored this variable in "fdr" with "n = Inf" flag. We extacted table from fdr topTags variable, which contains the pValue of all the genes in accending order along with gene names, LogFC value, logCPM value, LR value, FDR value and P-value. We are interested in p-value of Logistic regression we applied.

Then using elimination method based on P-value significance of genes which we got by GLM Fit model, We select genes with significant p-value which is less than 0.05 and we can remove all other genes from our dataset as, they are not as relevant feature on disease prediction.We write a table to store it, and then read it as read.table() function. Then we selected the genes on based on p-value less than 0.05, and extracted the row names of that genes only, to create a new data set called data.clean, which only include relevant genes for analysis.

We got approx 7500 genes out of 20000 genes which are relevant and plays significant role in affects tumor type by gene expression values.

```
# craeting factor of 5 tumor type
typ <- factor(cancerMatrix$Class)

# dummy coding tumor classes
design <- model.matrix(~typ)

# head of dummy table
print("Dummy code table for Tumor Classes")
```

```
## [1] "Dummy code table for Tumor Classes"
```

```
head(design)
```

```
##   (Intercept) typCOAD typKIRC typLUAD typPRAD
## 1           1       0       0       0       1
## 2           1       0       0       1       0
## 3           1       0       0       0       1
## 4           1       0       0       0       1
## 5           1       0       0       0       0
## 6           1       0       0       0       1
```

```
# Listing patients and genes along the grouping by tumor type
n <- DGEList(counts = data, group = typ)

# estimating GLM trend display
n <- estimateGLMTrendedDisp(n, design)

# estimating GLM tagwise display
n <- estimateGLMTagwiseDisp(n, design)

# creating glm fit model
fit <- glmFit(n,design)

# creating glmRT model
lrt <- glmLRT(fit,coef=2)

# applying topTag function
#topTags(lrt)

# storing topTag function
fdr <- topTags(lrt, n=Inf)

# writing the table in file
write.table(fdr$table, file = "table.txt", sep="\t", quote=FALSE)

# rading the written table
table <- read.table("table.txt", sep = "\t")

# getting head of table
print("Table with GLM fit results along with p-value of each gene")
```

```
## [1] "Table with GLM fit results along with p-value of each gene"
```

```
head(table)
```

```
##               logFC   logCPM       LR PValue FDR
## gene_11059 5.682410 5.113288 1777.850      0   0
## gene_5829  5.657307 5.200944 2105.596      0   0
## gene_2318  5.301042 5.504794 1823.629      0   0
## gene_30    4.993431 5.294807 1512.158      0   0
## gene_3524  4.750612 5.336074 2273.008      0   0
## gene_7965  4.721491 5.775332 2318.970      0   0
```

```
# extracting rows which have p-value less than 0.05
num <- which(table$PValue<0.05)

# selecting these genes only from data
table <- table[num,]

# nrow pf genes selected
print(paste0("Number of genes selected in Table is : " ,nrow(table)))
```

```
## [1] "Number of genes selected in Table is : 7628"
```

```
# creating new data set with selected genes only
data.clean <- data[c(rownames(table)),]

# printing nrow in new data to get number of genes selected
print(paste0("Number of genes selected in New data set is : " ,nrow(data.clean)))
```

```
## [1] "Number of genes selected in New data set is : 7628"
```

Data Cleaning and shaping

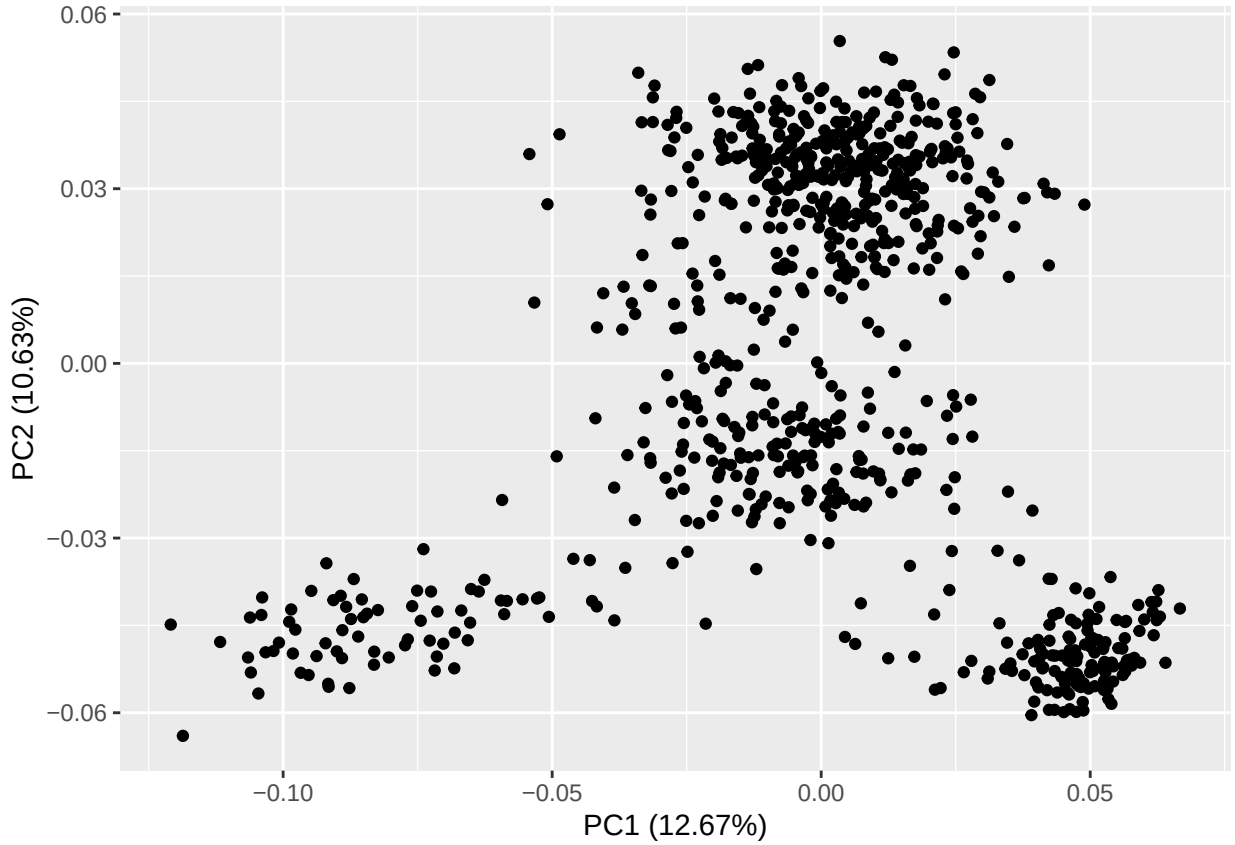
Normalization of feature Values

Now we got the relevant genes from data, next step is to normalize these gene expression values, as data is still highly skewed. For normalizing we are using Z-score normalization. It gives us values with zero mean and 1 standard deviation.

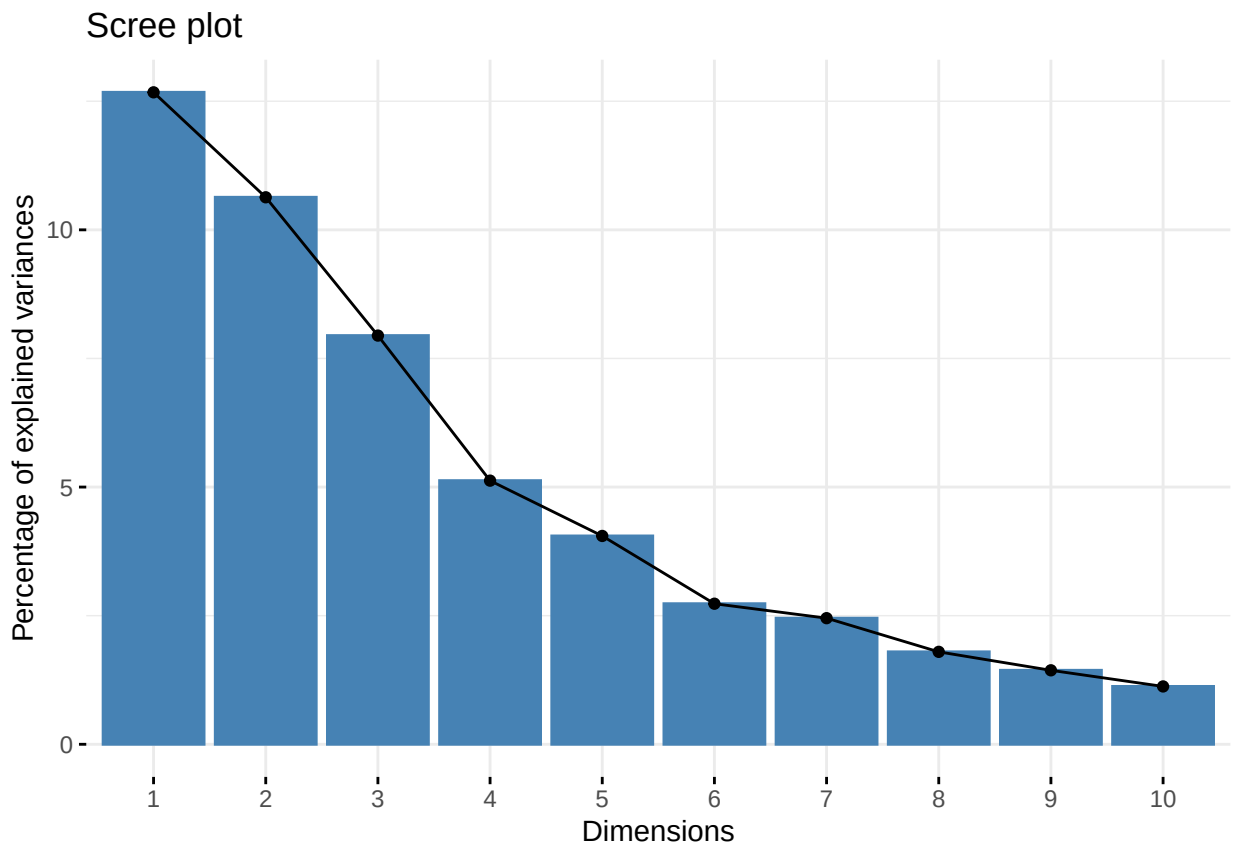
After normalizing we again tried pca and this time we can clearly see 5 clusters within data points, which indicates presence of 5 type of tumors in data and scree-plot also look more readable now.

```
# making copy of original data
data.clean.copy <- data.clean

# transposing data for further classification as patients in rows and genes in coloumns
data.clean <- as.data.frame(t(data.clean))

# normailization function
normalize <- function(x){
  return ((x - mean(x))/ sd(x))
```

```
}

# applying normalization
data.clean <- as.data.frame(apply(data.clean[1:ncol(data.clean)], 2, normalize))

# pca plot after cleaning
pca.data <- prcomp(data.clean)

# pca plot
autoplot(pca.data, data = data.clean)
```




```
# scree plot
fviz_eig(pca.data)
```



Data Cleaning and shaping

New Features

To create and use prediction algorithms we need to include tumor type within data as coloumn which assigns number 1 to 5, each represents one class of tumor, for each patient.

BRCA = 1 COAD = 2 KIRC = 3 LUAD = 4 PRAD = 5

We added new coloumn called "Type" to defines tumor type within data now, and also converted this coloumn as factor, so classification algorithms can use this as one of the predictive classes.

```
# adding new coloumn as numeric
data.clean$type <- as.numeric(typ)

# changing it to factor
data.clean$type <- as.factor(data.clean$type)

# rearranging type coloumn to be as first coloumn
data.clean <- data.clean %>%
  dplyr::select(type, everything())

# printing head
head(data.clean[1:5])
```

```
##          type gene_11059  gene_5829   gene_2318    gene_30
```

```
## sample_0    5 -0.3721673 -0.4972859 -0.08956186 -0.6768375
## sample_1    4 -0.4665655 -0.4972859 -0.05651513 -0.6768375
## sample_2    5 -0.4211857  0.2410459 -0.79469018 -0.6768375
## sample_3    5 -0.5804781 -0.1092429 -0.24470093 -0.6768375
## sample_4    1 -0.5804781 -0.4972859 -0.79469018 -0.6768375
## sample_5    5 -0.5804781 -0.4972859 -0.04829365  0.8850232
```

Model Construction And Evaluation

Creating Training and testing Data set

Creating training and testing data set using random sampling function with 70% train data and 30% test data.

```
# seed
set.seed(1234)

# random sampling of row number
random_sample <- sample(nrow(data.clean), 0.70*nrow(data.clean))

# creating test and train data
train <- data.clean[random_sample,]
test <- data.clean[-random_sample,]

paste0("Train has : ", nrow(train), " number of rows")
```

```
## [1] "Train has : 560 number of rows"
```

```
paste0("Test has : ", nrow(test), " number of rows")
```

```
## [1] "Test has : 241 number of rows"
```

Model Construction And Evaluation

Construction of Models

KNN Model Creating first classification model as KNN, using class library function KNN with train and test data along with k = 5, as there is chances of 5 groups of data based on different classes present in dataset.

Testing the model on test dataset using predict feature and holdout method for verification. KNN basically using euclidean distance algorithm to find out the class of data point based on nearest given numbers. KNN algorithm works perfectly great for this dataset it gives accuracy of 99.59%

```
# seed
set.seed(12345)

# applying Class library KNN function
knn.model <- knn(train, test, cl=train$type, k = 5)

# storing confusion matrix
```

```r
knn.matrix <- confusionMatrix(knn.model, test$type)

# printing confusion matrix
knn.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 86  0  0  1  0
##          2  0 23  0  0  0
##          3  0  0 50  0  0
##          4  0  0  0 42  0
##          5  0  0  0  0 39
##
## Overall Statistics
##
##                Accuracy : 0.9959
##                  95% CI : (0.9771, 0.9999)
##     No Information Rate : 0.3568
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9946
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000  1.00000   1.0000   0.9767   1.0000
## Specificity            0.9935  1.00000   1.0000   1.0000   1.0000
## Pos Pred Value         0.9885  1.00000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000  1.00000   1.0000   0.9950   1.0000
## Prevalence             0.3568  0.09544   0.2075   0.1784   0.1618
## Detection Rate         0.3568  0.09544   0.2075   0.1743   0.1618
## Detection Prevalence   0.3610  0.09544   0.2075   0.1743   0.1618
## Balanced Accuracy      0.9968  1.00000   1.0000   0.9884   1.0000
```

Model Construction And Evaluation

Construction of Models

Naive Bayes Model For next model we applied Naive Bayes from Klar library, and we used similar holdout method for validation of the model as we used earlier. With NaiveBayes we got accuracy of 98.34%

```r
# seed
set.seed(12345)

# Naive Model from klar class
naive.model <- naiveBayes(type ~ ., data = train, prob = TRUE)

# getting  prediction
```

```r
naive.predict <- predict(naive.model, test)

# printing Crosstable matrix
nb.matrix <- confusionMatrix(naive.predict, test$type)

nb.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 86  0  0  1  1
##          2  0 23  0  0  0
##          3  0  0 48  0  0
##          4  0  0  2 42  0
##          5  0  0  0  0 38
##
## Overall Statistics
##
##                Accuracy : 0.9834
##                  95% CI : (0.9581, 0.9955)
##     No Information Rate : 0.3568
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9782
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000  1.00000   0.9600   0.9767   0.9744
## Specificity            0.9871  1.00000   1.0000   0.9899   1.0000
## Pos Pred Value         0.9773  1.00000   1.0000   0.9545   1.0000
## Neg Pred Value         1.0000  1.00000   0.9896   0.9949   0.9951
## Prevalence             0.3568  0.09544   0.2075   0.1784   0.1618
## Detection Rate         0.3568  0.09544   0.1992   0.1743   0.1577
## Detection Prevalence   0.3651  0.09544   0.1992   0.1826   0.1577
## Balanced Accuracy      0.9935  1.00000   0.9800   0.9833   0.9872
```

Model Construction And Evaluation

Construction of Models

SVM For SVM classification we used e1071 library package, svm(), we provided the training data set and used predict function to use holdout evaluation method. For SVM we got accuracy of 98.76%

```r
# seed
set.seed(12345)

# svm model
svm.model <- svm(type ~ ., data = train, probability =TRUE)
```

```r
# getting prediction
svm.predict <- predict(svm.model, test, probability = TRUE)

# printing Crosstable matrix
svm.matrix <- confusionMatrix(svm.predict, test$type)

svm.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 86  0  0  1  1
##          2  0 23  0  0  0
##          3  0  0 49  0  0
##          4  0  0  1 42  0
##          5  0  0  0  0 38
##
## Overall Statistics
##
##                Accuracy : 0.9876
##                  95% CI : (0.9641, 0.9974)
##     No Information Rate : 0.3568
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9836
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000  1.00000   0.9800   0.9767   0.9744
## Specificity            0.9871  1.00000   1.0000   0.9949   1.0000
## Pos Pred Value         0.9773  1.00000   1.0000   0.9767   1.0000
## Neg Pred Value         1.0000  1.00000   0.9948   0.9949   0.9951
## Prevalence             0.3568  0.09544   0.2075   0.1784   0.1618
## Detection Rate         0.3568  0.09544   0.2033   0.1743   0.1577
## Detection Prevalence   0.3651  0.09544   0.2033   0.1784   0.1577
## Balanced Accuracy      0.9935  1.00000   0.9900   0.9858   0.9872
```

Model Construction And Evaluation

Construction of Models

Decision Tree and Tuning of Tree For decision tree model we used party library, and ctree() function. Forst we created controlled tree, with tree growth control parameter, to make it more fast, as creating decision tree for this large data set takes too much computation power and time. And with controlled growth parameter of mincriterion to divide as 90% interval and minsplit as atleast 300, and we only got accuracy of 71%

To improve the accuracy of the model we used the tuning method, and let the tree grow without any control

parameters. And at this time without any paramters, tree with full length gives us accuracy of 88%, which is significant improvement.

```
# seed
set.seed(123)

# initial model with growth control parameters
tree.model.initial <- ctree(type ~ . , data = train , controls = ctree_control(mincriterion = 0.90, mins

# getting prediction
tree.predict.int <- predict(tree.model.initial, test)

# printing Crosstable matrix
print("Confusion Matrix for Decision Tree with Growth control")
```

```
## [1] "Confusion Matrix for Decision Tree with Growth control"
```

```
confusionMatrix(tree.predict.int, test$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 85  0  0  2  0
##          2  0  0  0  0  0
##          3  0 23 50 41  1
##          4  0  0  0  0  0
##          5  1  0  0  0 38
##
## Overall Statistics
##
##                Accuracy : 0.7178
##                  95% CI : (0.6565, 0.7737)
##     No Information Rate : 0.3568
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6218
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9884  0.00000   1.0000   0.0000   0.9744
## Specificity            0.9871  1.00000   0.6597   1.0000   0.9950
## Pos Pred Value         0.9770      NaN   0.4348      NaN   0.9744
## Neg Pred Value         0.9935  0.90456   1.0000   0.8216   0.9950
## Prevalence             0.3568  0.09544   0.2075   0.1784   0.1618
## Detection Rate         0.3527  0.00000   0.2075   0.0000   0.1577
## Detection Prevalence   0.3610  0.00000   0.4772   0.0000   0.1618
## Balanced Accuracy      0.9877  0.50000   0.8298   0.5000   0.9847
```

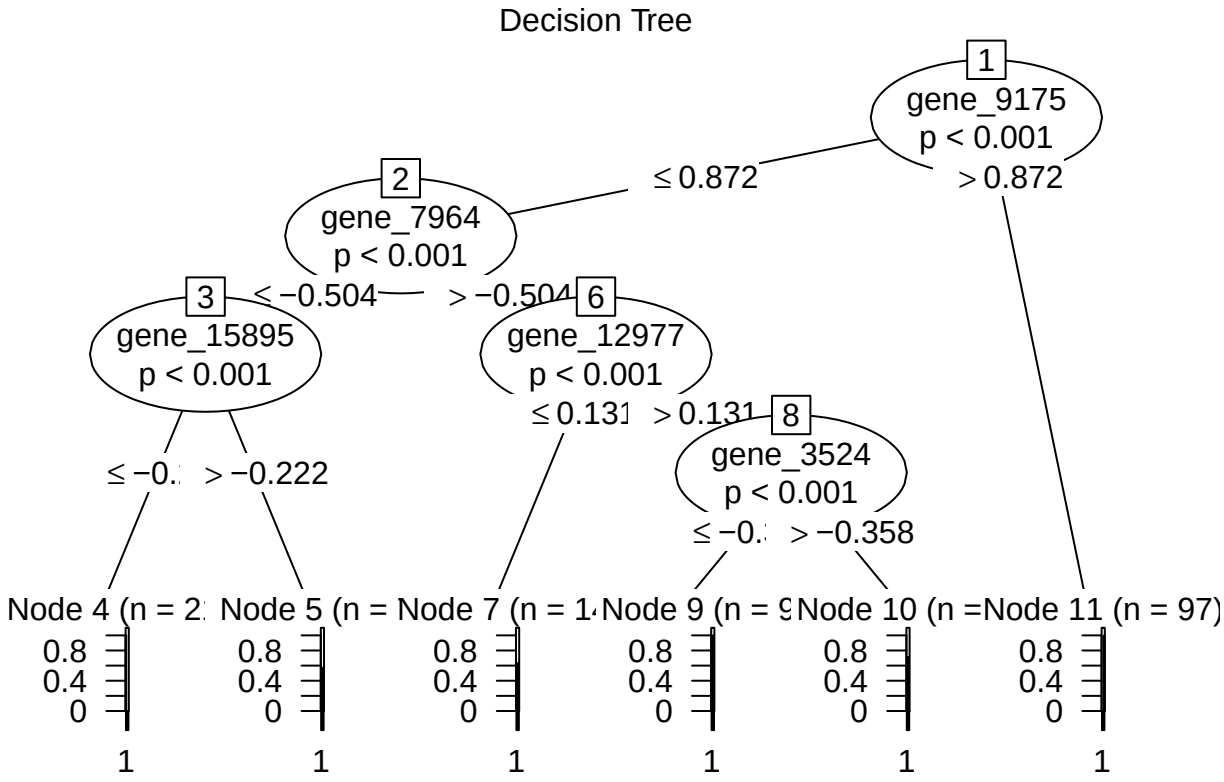
```r
# 2nd model with tuning
tree.model <- ctree(type ~ . , data = train)

# getting prediction
tree.predict <- predict(tree.model, test)

# printing Crosstable matrix
tree.matrix <- confusionMatrix(tree.predict, test$type)

print("Decision Tree without Growth control")
```

```
## [1] "Decision Tree without Growth control"
```

```r
tree.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 83  0  0  0  0
##          2  0  0  0  0  0
##          3  0  0 50  0  0
##          4  2 23  0 43  1
##          5  1  0  0  0 38
##
## Overall Statistics
##
##                Accuracy : 0.888
##                  95% CI : (0.8412, 0.9249)
##     No Information Rate : 0.3568
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.852
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9651  0.00000   1.0000   1.0000   0.9744
## Specificity            1.0000  1.00000   1.0000   0.8687   0.9950
## Pos Pred Value         1.0000      NaN   1.0000   0.6232   0.9744
## Neg Pred Value         0.9810  0.90456   1.0000   1.0000   0.9950
## Prevalence             0.3568  0.09544   0.2075   0.1784   0.1618
## Detection Rate         0.3444  0.00000   0.2075   0.1784   0.1577
## Detection Prevalence   0.3444  0.00000   0.2075   0.2863   0.1618
## Balanced Accuracy      0.9826  0.50000   1.0000   0.9343   0.9847
```

```r
# getting tree plot
plot(tree.model, main = "Decision Tree")
```



Model Construction And Evaluation

Construction of Models

Neural Netwowk Model and Tuning of Model For neural net algorithm on this dataset using neuralnet() library and neuralnet() function from that. We used same training and testing dataset with 70%::30% ratio. For neural network we need all numeric variable as neural net dosent work on categorical or factor variables, so we converted it to integer first. Then we applied neuralnet algorithm with stepmax = 1e+08, and rep = 1, to make the prediction fast as data set is too big. With these parameter we got accuracy of around 65% which can be improved.

To make this algorithm more efficiant and increase accuracy as its a difficult task to do because dataset is too big, and it increases big O notation of that and increase time exponentially, for accuracy we are using Softplus smothening of the data with log, exponential function.

To reduce the computational time of the model, we also altered some variables, like hidden threshold = 0.1, stepmax = 1e+08 from 1e+05, learningrate = 0.01 and linear output False, so this algorithm with large dataset can work in Polynomial time instead of Non-Polynomial time.

On based of this data and these parameters, we got the accuracy approximately around 85% which is a big improvement, based on tuning model used.

```
# seed
set.seed(12345)

# creating copy of training and testing data
```

```r
trainNN <- train; testNN <- test

# converting decision class to integer from factor
trainNN$type <- as.integer(trainNN$type)
testNN$type <- as.integer(testNN$type)

# library
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 3.6.3
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```r
# neuralnet model initial model
nn.model <- neuralnet(type ~ ., data=trainNN, rep=1, stepmax = 1e+08, linear.output = F)

# computing with original values
nn.predict <- neuralnet::compute(nn.model, testNN)

# storing result in variable
nn.result <- nn.predict$net.result

# calculating correlation between values
paste0("Correlation Neural Net with default parameters : ", round(cor(nn.result, testNN$type),2))
```

```
## [1] "Correlation Neural Net with default parameters : 0.66"
```

```r
# smothening algorithm
softplus <- function(x) {
  log(1+exp(x))
}

# seed
set.seed(12345)

# tuned model with smothening and different parameters to increase accuracy
nn.model <- neuralnet(type ~ ., data=trainNN, act.fct=softplus, threshold = 0.1, rep=1, stepmax = 1e+08

# computing with original values
nn.predict <- neuralnet::compute(nn.model, testNN)

# storing result in variable
nn.result <- nn.predict$net.result

# calculating correlation between values
#print("Correlation between default values and predicted values by Neural Net")
nn.cor <- cor(nn.result, testNN$type)
```

```r
paste0("Correlation by Neural Net with tuned paramters : ", round(nn.cor, 2))
```

```
## [1] "Correlation by Neural Net with tuned paramters : 0.85"
```

Model Construction And Evaluation

K-fold cross-Validation

For K-fold validation, we are using createFolds() function from caret package. We are using 5 fold cross validation, and for models we are applying this on SVM and NaiveBayes. We created a function with random splitting of train data set in 5 parts, and passing each part as test one by one, and using other parts as train from svm and naiveBayes in each turn. This function take folds as argument. We printed out the results below as accuracy of each model, which only has slight variations.

```r
# creating folds
folds = createFolds(train$type, k = 5)

# function for cross validation, with paramter as folds
cv = lapply(folds, function(x) {

  # creating training and testing data
  training_fold = train[-x, ]
  test_fold = train[x, ]

  # model with train data
  classifier = svm(formula = type ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'radial')

  # prediction with test data using predict function
  y_pred = predict(classifier, newdata = test_fold[,-1])

  # confusion matrix
  cm = confusionMatrix(test_fold$type, y_pred)

  # extracting accuracy for each fold
  accuracy = cm$overall[1]
  return(accuracy)
})

print("K-Fold accuracy for SVM model")
```

```
## [1] "K-Fold accuracy for SVM model"
```

```r
cv
```

```
## $Fold1
##  Accuracy
## 0.9821429
##
```

```
## $Fold2
## Accuracy
##        1
##
## $Fold3
##  Accuracy
## 0.9911504
##
## $Fold4
## Accuracy
##        1
##
## $Fold5
##  Accuracy
## 0.9911504
```

```
# function for cross validation, with paramter as folds
cv = lapply(folds, function(x) {

  # creating training and testing data
  training_fold = train[-x, ]
  test_fold = train[x, ]

  # model with train data
  classifier = naiveBayes(formula = type ~ .,
                  data = training_fold)

  # prediction with test data using predict function
  y_pred = predict(classifier, newdata = test_fold[,-1])

  # confusion matrix
  cm = confusionMatrix(test_fold$type, y_pred)

  # extracting accuracy for each fold
  accuracy = cm$overall[1]
  return(accuracy)
})
print("K-Fold accuracy for NaiveBayes model")
```

```
## [1] "K-Fold accuracy for NaiveBayes model"
```

```
cv
```

```
## $Fold1
##  Accuracy
## 0.9910714
##
## $Fold2
## Accuracy
##        1
##
## $Fold3
## Accuracy
```

```
##        1
##
## $Fold4
##  Accuracy
## 0.9910714
##
## $Fold5
##  Accuracy
## 0.9911504
```

Model Construction And Evaluation

Comparison of Model and Result Interpertations

Model comparison using different graphs For model comparision, first thing we did is store all the values of acuuracy of all models, along with kappa values from confusion matrix in data frame. I also created error percentage coloumn for each model, which I calculated by one minus accuracy.

The first step would be to plot the accuracy graph of each model, using ggplot2, bar_chart, with using accuracy based on cross-validation of models, by using test data set as holdout method for validation. On the X-axis there are 5 models, and on the Y-axis, there is accuracy from zero to one. KNN and SVM produced best reults, while Neural_Net and Decision tree are having lowest accuracy.

Next step, we plotted similar graph with Error rate this time, instead of accuracy we plotted for error in each model. Neural Network has the highest error rate, while KNN has the lowest error rate.

So for further comparison, are using NaiveBayes as it is one of the best performing algorithm and decision tree as it has low accuracy as compared to other models. We created two plots using ggplot2, geom_tile function for tile graph. This is a matrix plot for each class as, colour frequency to show the accuracy of each class within model, so we can see here in NaiveBayes only second class has poor acuuracy where as 1st class has nearly perfect accuracy.

In Decision tree model class 2 has poor accuracy and class 1 has good accuracy. To plot this graph we used table feature from confusion matrix() class.

For our final graph we plotted the perfect **Error vs Original graph plot**, using ggplot2, on the Y-axis we plotted classes for each model from 1 to 5, and X-axis we plotted sample number in our test dataset, we have around 200 samples in out test dataset, we extracted only those samples from our 5 models, which were missclassified even in one model, and we got 30 missclassifiactions.

These samples are plotted using geom_point across the graph, with colour coded for each model differently, **Red = NaiveBayes, Green = SVM, Blue = Decision Tree, Yellow = KNN, and Black points as original value of class based on test dataset.** We also used jitter() function to see each point differently in case of overlapping.

I plotted points accross the graph where there is missclassification in each model with different color and black points in that row represents the true class for that sample.

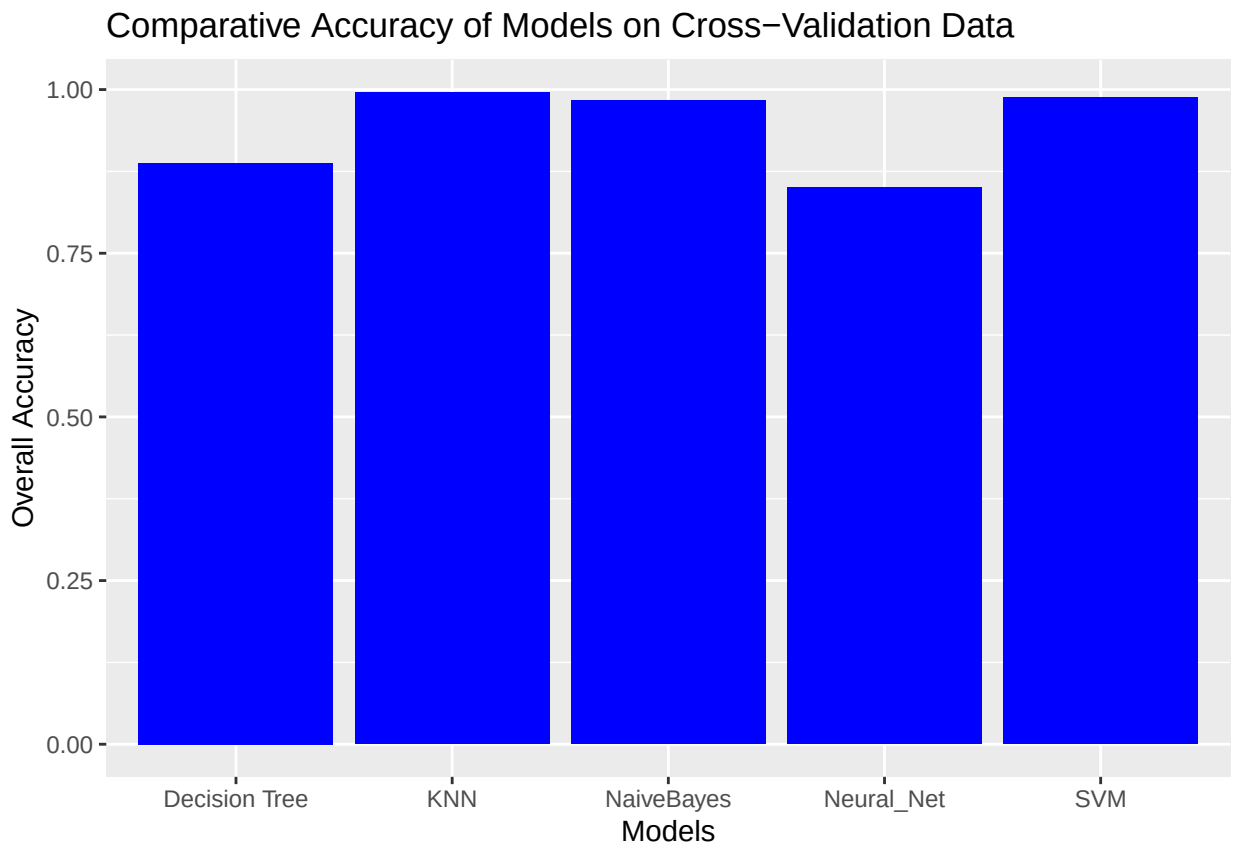
In biological and industrial point of view this graph can be very usefull. As plotting missclassification error for each sample gives us chance to see which patient is missclassified. Because any missclassification for cancer type can cost the other patients life. So by this graph we can see which model missclassified which class, So, that for further use we can pay more attention to similar type of case in our new test data set, which we want to predict.

```
# graph source code for some charts <- http://rstudio-pubs-static.s3.amazonaws.com/25355_641a9bbf060448

# creating data frame with accuracy, kappa and error rate
```

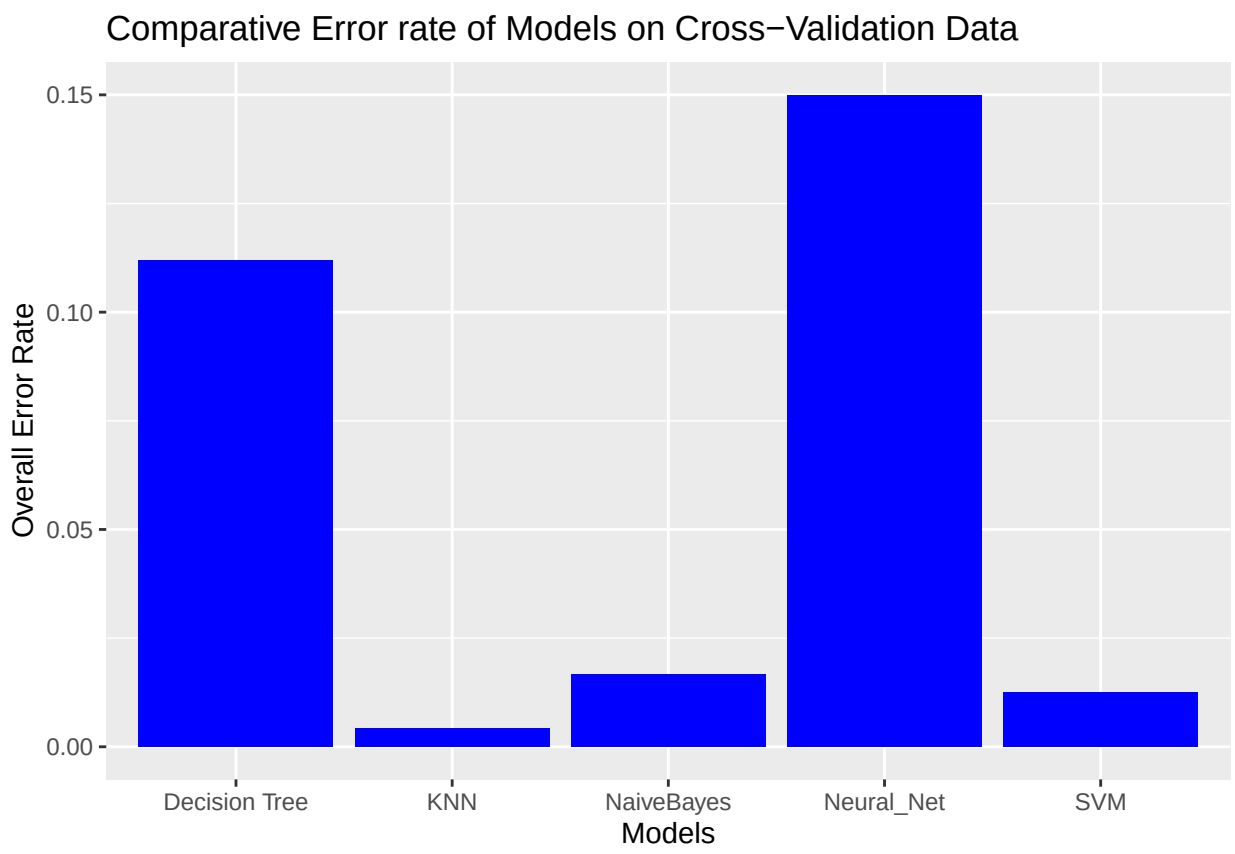
```r
results <- data.frame("Accuracy" = c(knn.matrix$overall[1], nb.matrix$overall[1], svm.matrix$overall[1]

# storing error rate
results$Error <- 1 - results$Accuracy

# result interpertation
head(results,5)
```

```
##    Accuracy     KAPPA        Models       Error
## 1 0.9958506 0.9945528           KNN 0.004149378
## 2 0.9834025 0.9781950    NaiveBayes 0.016597510
## 3 0.9875519 0.9836436           SVM 0.012448133
## 4 0.8879668 0.8519622 Decision Tree 0.112033195
## 5 0.8499923 0.8499923    Neural_Net 0.150007699
```

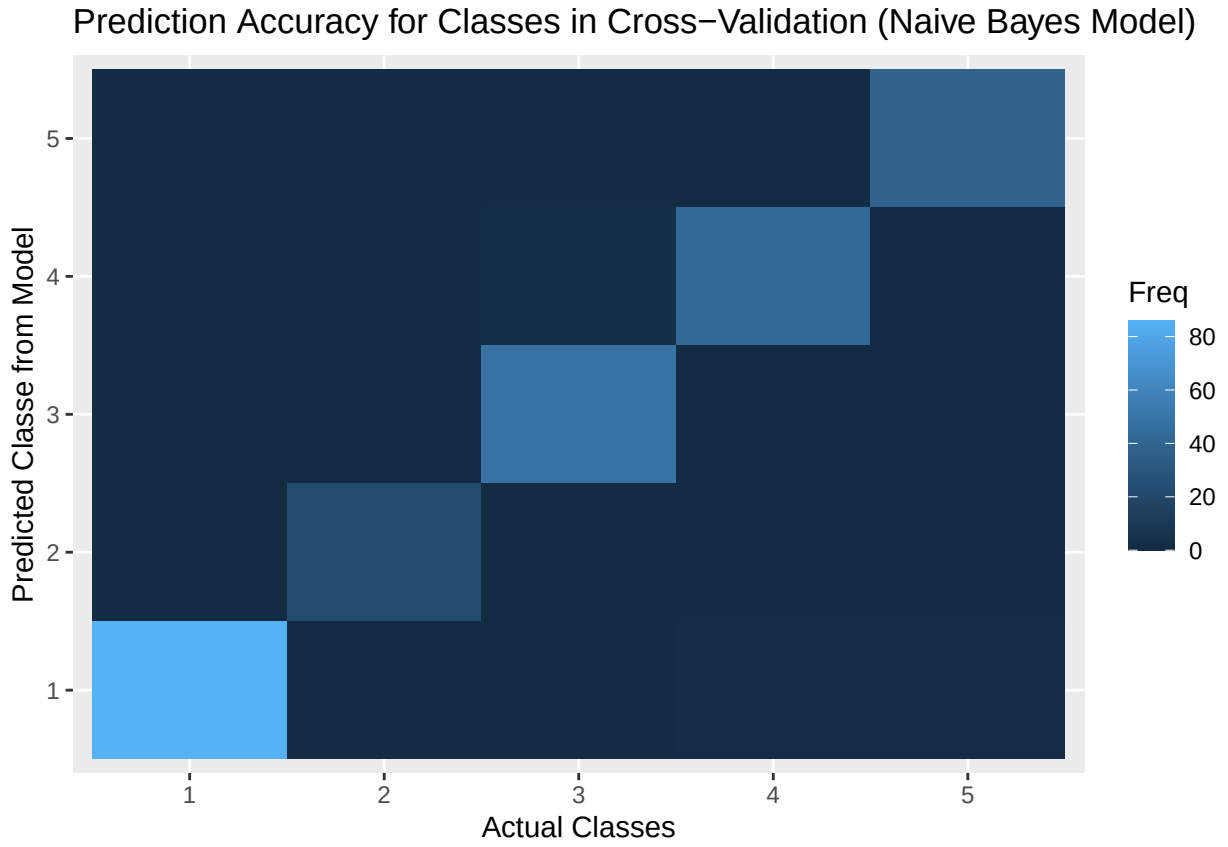
```r
# setting row names as each model
rownames(results) <- c("KNN", "NaiveBayes", "SVM", "Decision Tree", "Neural_Net")

# plotting accuracy bar chart
ggplot(aes(x=Models, y=Accuracy), data=results) +
    geom_bar(stat='identity', fill = 'blue') +
    ggtitle('Comparative Accuracy of Models on Cross-Validation Data') +
    xlab('Models') +
    ylab('Overall Accuracy')
```



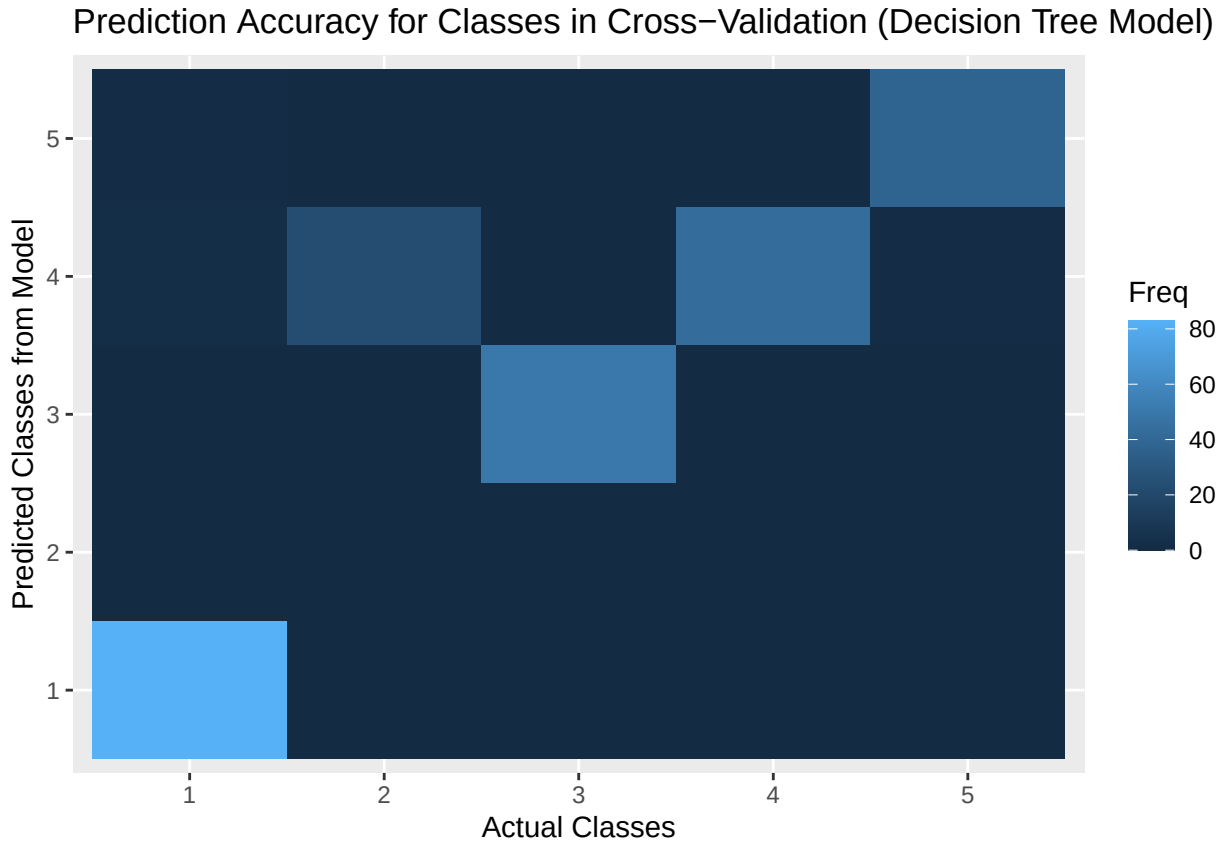
```
# plotting error rate bar chart
ggplot(aes(x=Models, y=Error), data=results) +
    geom_bar(stat='identity', fill = 'blue') +
    ggtitle('Comparative Error rate of Models on Cross-Validation Data') +
    xlab('Models') +
    ylab('Overall Error Rate')
```




```
# plotting accuracy tile matrix for each class, for Naivebayes
ggplot(data=data.frame(nb.matrix$table)) +
    geom_tile(aes(x=Reference, y=Prediction, fill=Freq)) +
    ggtitle('Prediction Accuracy for Classes in Cross-Validation (Naive Bayes Model)') +
    xlab('Actual Classes') +
    ylab('Predicted Classe from Model')
```

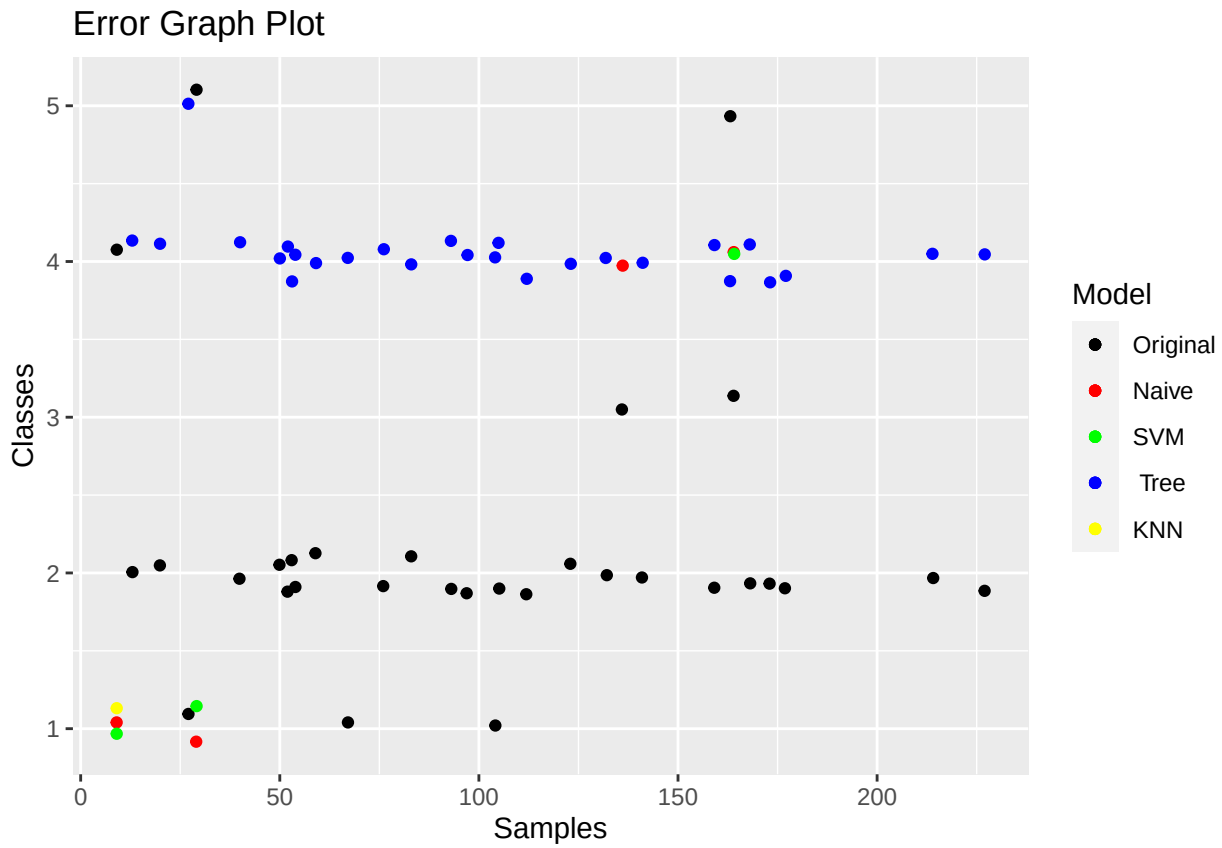



```
# plotting accuracy tile matrix for each class, for decision tree
ggplot(data=data.frame(tree.matrix$table)) +
    geom_tile(aes(x=Reference, y=Prediction, fill=Freq)) +
    ggtitle('Prediction Accuracy for Classes in Cross-Validation (Decision Tree Model)') +
    xlab('Actual Classes') +
    ylab('Predicted Classes from Model')
```




```
# creating new data frame for storing each prediction
df1 <- data.frame(sample = (rownames(test)), nb = naive.predict, svm = svm.predict, tree = tree.predict

# creating empty coloumns to store missclassifications
df1$new <- 0
df1$new2 <- 0
df1$new3 <- 0
df1$new4 <- 0

# converting patient names to numeric
df1$sample <- as.numeric(df1$sample)

# extracting missclassification for each sample for each model
for (i in 1:nrow(df1)){
  if(df1[i,2] != df1$original[i]){
    df1$new[i] <- df1[i,2]
  }
  if(df1[i,3] != df1$original[i]){
    df1$new2[i] <- df1[i,3]
  }
  if(df1[i,4] != df1$original[i]){
    df1$new3[i] <- df1[i,4]
  }
  if(df1[i,5] != df1$original[i]){
    df1$new4[i] <- df1[i,5]
  }
```

```
}

# removing the patients with no missclassifications
df1 <- df1[-(which(df1$new==0 & df1$new2 == 0 & df1$new3 == 0 & df1$new4 == 0)),]

# setting other parameters as NA, so they dont get plotted in graph
df1$new <- ifelse(df1$new == 0 , NA, df1$new)
df1$new2 <- ifelse(df1$new2 == 0 , NA, df1$new2)
df1$new3 <- ifelse(df1$new3 == 0 , NA, df1$new3)
df1$new4 <- ifelse(df1$new4 == 0 , NA, df1$new4)


# ggplot2 code for each model, each missclassification, diferent colour coded
ggplot(aes(x = df1$sample, y = as.numeric(df1$original)), data = df1) + geom_point(aes(x = df1$sample, y
  geom_point(aes(x = df1$sample, y = as.numeric(df1$new), color = "red"), position=position_jitter(h=0.
  geom_point(aes(x = df1$sample, y = as.numeric(df1$new2), color = "green"), position=position_jitter(h
  geom_point(aes(x = df1$sample, y = as.numeric(df1$new3), color = "blue"), position=position_jitter(h=
  geom_point(aes(x = df1$sample, y = as.numeric(df1$new4), color = "yellow"), position=position_jitter(
                          breaks = c("black", "red", "green", "blue", "yellow"),
                          labels = c("Original", "Naive", "SVM", " Tree", "KNN"),
                          guide = "legend")
```



Model Construction And Evaluation

Ensemble Model

Creating an ensemble model using all the predictions we have done, using all the previous models, and trying to build boosting and bagging algorithm using mode function to predict the final outcome as, we are predicting classes, so mode works best in analysing which one is best, and can be better than using mean and median fucntion.

For ensembling we created a data frame with all the predictions by models in previous stages for test data set. Then, we created mode function to calculate highest occuring class predicted by each models for particular sample. To calculate mode we used apply() function, along each row one by one, and stored the final prediction in new variable in data frame. In the end we created confusion matrix to evaluate the accuracy and other factors.

And surprisingly, even though Neural Net prediction and Decision Tree predictions are not high like more than 90%, but in ensemble model, all the models boosted the results and gave high accuracy of **99.2%**. This ensemble model, not only have high accuracy but this also increase the performance of each individual model also. This model displays, nearly perect ensemble model, which can definetley be used for industrial usage in future.

```
# creating data frame with all predictions
ensemble.model <- data.frame(knn.model, naive.predict, svm.predict, tree.predict, round(nn.result))

# mode function
calculate_mode <- function(x) {
  uniqx <- unique(x)
  uniqx[which.max(tabulate(match(x, uniqx)))]
}

# getting mode of each row to predict final outcome
ensemble.model$Final <- apply(ensemble.model, 1, calculate_mode)

head(ensemble.model)
```

```
##          knn.model naive.predict svm.predict tree.predict round.nn.result.
## sample_2         5             1           1            5                 1
## sample_4         1             1           1            1                 1
## sample_5         5             5           5            5                 5
## sample_6         3             3           3            3                 2
## sample_7         5             5           5            5                 5
## sample_8         1             1           1            1                 1
##          Final
## sample_2     1
## sample_4     1
## sample_5     5
## sample_6     3
## sample_7     5
## sample_8     1
```

```
# Printing confusion matrix
print("Ensemble model Confusion Matrix")
```

```
## [1] "Ensemble model Confusion Matrix"
```

```
confusionMatrix(as.factor(ensemble.model$Final), test$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 86  0  0  1  1
##          2  0 23  0  0  0
##          3  0  0 50  0  0
##          4  0  0  0 42  0
##          5  0  0  0  0 38
##
## Overall Statistics
##
##                Accuracy : 0.9917
##                  95% CI : (0.9703, 0.999)
##     No Information Rate : 0.3568
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9891
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000  1.00000   1.0000   0.9767   0.9744
## Specificity            0.9871  1.00000   1.0000   1.0000   1.0000
## Pos Pred Value         0.9773  1.00000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000  1.00000   1.0000   0.9950   0.9951
## Prevalence             0.3568  0.09544   0.2075   0.1784   0.1618
## Detection Rate         0.3568  0.09544   0.2075   0.1743   0.1577
## Detection Prevalence   0.3651  0.09544   0.2075   0.1743   0.1577
## Balanced Accuracy      0.9935  1.00000   1.0000   0.9884   0.9872
```

```
paste0("Ensemble Model Accuracy : ", round(confusionMatrix(as.factor(ensemble.model$Final), test$type)$
```

```
## [1] "Ensemble Model Accuracy : 99.2%"
```

Summary

This project with high accuracy on different machine learning models, shows great potential to be used for further industrial use to predict the type of tumor present in cancer patient by gene expression values, in pre-stages of disease, which can be very helpful for starting early treatment for cancer.