Breast Cancer Classification

Souvik Paul

24/07/2020

Abstract

Various Classification techniques are used in this project to classify the cells into 'Benign' and 'Malign'. But before applying those techniques firstly the data is cleaned up by imputing missing observations (using Random Forest) and detecting outliers (using Mahalanobis Distance). KNN,Logistic Regression, LDA, Naive Bayes, Decision Tree, Random Forest, SVM and ANN are used and in each case effect of outliers is detected. Two models are used for Naive Bayes(Kernel and without Kernel) and ANN (hidden layer : one and two) and multiple models are used for Random Forest and SVM , and finally efficiency of each model are checked. Finally importance of each feature is checked and it is also checked if there is any possibility of excluding any feature (using Random Forest).

Introduction

Classification is a part of Supervised Learning in Machine Learning. And Supervised Learning is perhaps best described by its own name. A Supervised Learning algorithm which is taught by the data it is given. In our modern era Machine Learning is used vastly in medical sciences. To classify presence or absence of any disease like diabetes, heart disease, cancer etc. in human body, Machine Learning techniques are applied and impressive results are found. In this project our intension is to classify presence of breast cancer symptomps in a cell. The importance of classifying cancer patients into high or low risk groups has led many researchers, from the biomedical and bio informatics field, to study the application of Machine Learning methods. The ability of Machine Learning tools to detect key features from complex datasets reveals their importance. A variety of these techniques, including Artificial Neural Network, K Nearest Neighbour, Support vector Machine etc. have been widely applied in cancer research for the development of predictive models as at a fundamental level, it is evident that Machine Learning is helping to improve basic understanding of cancer development average progression.

Data Information

'Wisconsin Breast Cancer (Original)' dataset is used here and it is collected from https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original). This data consists of 698 observations with 11 features. At first stage (Jan 1989) this data consisted of only 367 instances, samples were arrived periodically and after 7 stages sample size becomes 698. These 11 features are –

(i) Sample Code Number - Sample id number

(ii) Clump Thickness- It describes a cell monolayered or multilayered. Multilayered cell implies tendency towards maliganancy.

(iii) Uniformity of Cell Size- It evaluates the consistency in size of the cells in sample. Inconsistency shows tendency towards malignancy.

(iv) Uniformity of Cell shape- It estimates the equality of cell shapes and identify marginal variances. Unequal cell shapes may be regarded as a chance of malignancy.

(v) Marginal Adhesion- It quantifies how much cells on the outside of the epithelial tend to stick together. Lower the quantity higher will be the chance of malignant cells.

(vi) Single Epithelial Cell Size- It relates to cell uniformity, determines if epithelial cells are significantly enlarged. Large cell indicates malignancy.

(vii) Bare Nuclei- It calculate the proportion of the number of cells not surrounded by cytoplasm to those that are. Lower the proportion higher will be the chance of malignant cells.

(viii) Bland Chromatin- It rates the uniform texture of the nucleus in a range from fine to coarse. Coarse rating indicates malignancy.

(ix) Normal Nucleoli- Determines whether the nucleoli are small and rarely visible or larger, more visible and more plentiful. Later case indicates malignancy.

(x) Mitoses- Describes the level of mitotic cell reproduction activity. Higher the level higher will be chance of malignancy.

(xi) Class- '*Benign*' for Normal cell and '*Malignant*' for cancer cell.

All the features **(ii)-(x)** are valued on a scale of 1-10, with 1 being the closest to '*Benign*' and 10 towards '*Malignant*'.

Objective

Objective of this project is to classify the new cells into two groups '*Benign*' and '*Malign*' on the basis other features. And it will checked if we can reduce some features without losing too much information. We will identify the features which are most important in diagnosis.

Data Pre-processing

Here we chose '**Class**' variable as our response variable, where '2' indicates '*Benign*' and '4' indicates '*Malignant*'. Now we import the data and check dimension and first few rows of the data.

```
data=read.delim('data.txt',sep=',',na.strings = '?')
dim(data)
```

```
## [1] 698  11
```

```
head(data)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
## 4  1017023  4  1    1    3  2    1  3    1    1    2
## 5  1017122  8 10   10    8  7   10  9    7    1    4
## 6  1018099  1  1    1    1  2   10  3    1    1    2
```

Now we will exclude the first column which is '**Sample Code Number**' as it is of no use in inference. Then we rename our features appropriately and define our response.

```
data=data[,-1]
colnames(data)=c('Clump_thik','Unif_cell_size','Unif_cell_shape','Mar_adhe','
Epi_cell_size','Bare_nuclei','Bland_chrom','Norm_nucleoli','Mitoses','Class')
y=data[,which(colnames(data)=='Class')]
head(data)
```

```
##   Clump_thik Unif_cell_size Unif_cell_shape Mar_adhe Epi_cell_size
Bare_nuclei
## 1          5              4               4        5             7
10
## 2          3              1               1        1             2
2
## 3          6              8               8        1             3
4
## 4          4              1               1        3             2
1
## 5          8             10              10        8             7
10
## 6          1              1               1        1             2
10
##   Bland_chrom Norm_nucleoli Mitoses Class
## 1           3             2       1     2
## 2           3             1       1     2
## 3           3             7       1     2
## 4           3             1       1     2
## 5           9             7       1     4
## 6           3             1       1     2
```

```
dim(data)
```

```
## [1] 698  10
```

Now we see that there is 698 observations, 9 predictor variables and 1 response variable.

Missing Value Imputation

Firstly we will check if there is any missing value.

```
which(is.na(data)==T)

##  [1] 3513 3530 3629 3635 3648 3654 3725 3739 3765 3782 3784 3787 3805 3811
3901
## [16] 4107

length(which(is.na(data)==T))

## [1] 16

colnames(data)[which(is.na(colSums(data))==T)]

## [1] "Bare_nuclei"
```

This indicates that there are 16 missing values. Only the variable 'Bare_nuclei' contains missing values. Before imputing missing values take a look into the data -

```
str(data)

## 'data.frame':    698 obs. of  10 variables:
##  $ Clump_thik     : int  5 3 6 4 8 1 2 2 4 1 ...
##  $ Unif_cell_size : int  4 1 8 1 10 1 1 1 2 1 ...
##  $ Unif_cell_shape: int  4 1 8 1 10 1 2 1 1 1 ...
##  $ Mar_adhe       : int  5 1 1 3 8 1 1 1 1 1 ...
##  $ Epi_cell_size  : int  7 2 3 2 7 2 2 2 2 1 ...
##  $ Bare_nuclei    : int  10 2 4 1 10 10 1 1 1 1 ...
##  $ Bland_chrom    : int  3 3 3 3 9 3 3 1 2 3 ...
##  $ Norm_nucleoli  : int  2 1 7 1 7 1 1 1 1 1 ...
##  $ Mitoses        : int  1 1 1 1 1 1 1 5 1 1 ...
##  $ Class          : int  2 2 2 2 4 2 2 2 2 2 ...

mis=which(is.na(data$Bare_nuclei)==T)
```

Here all the variables are of integer type but we need the response variable **Class** as categorical. So, we need to modify it.

```
data$Class=as.factor(data$Class)
str(data)

## 'data.frame':    698 obs. of  10 variables:
##  $ Clump_thik     : int  5 3 6 4 8 1 2 2 4 1 ...
##  $ Unif_cell_size : int  4 1 8 1 10 1 1 1 2 1 ...
##  $ Unif_cell_shape: int  4 1 8 1 10 1 2 1 1 1 ...
##  $ Mar_adhe       : int  5 1 1 3 8 1 1 1 1 1 ...
##  $ Epi_cell_size  : int  7 2 3 2 7 2 2 2 2 1 ...
##  $ Bare_nuclei    : int  10 2 4 1 10 10 1 1 1 1 ...
##  $ Bland_chrom    : int  3 3 3 3 9 3 3 1 2 3 ...
##  $ Norm_nucleoli  : int  2 1 7 1 7 1 1 1 1 1 ...
##  $ Mitoses        : int  1 1 1 1 1 1 1 5 1 1 ...
##  $ Class          : Factor w/ 2 levels "2","4": 1 1 1 1 2 1 1 1 1 1 ...
```

Now we impute the missing values using Random Forest.

```
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

set.seed(99)
data=rfImpute(Class~.,data=data,iter=6)

## ntree      OOB      1      2
##   300:   3.44%  3.06%  4.15%
## ntree      OOB      1      2
##   300:   2.87%  2.84%  2.90%
## ntree      OOB      1      2
##   300:   3.30%  3.06%  3.73%
## ntree      OOB      1      2
##   300:   2.87%  2.84%  2.90%
## ntree      OOB      1      2
##   300:   2.87%  3.06%  2.49%
## ntree      OOB      1      2
##   300:   3.15%  3.06%  3.32%
```

By `iter=6` we repeat the random forest 6 times. After each iteration OOB(out of bag error rate) is printed. This should get smaller if the estimates improve. Since it doesnot, we can conclude our estimates are as good as they are going to get with this random forest method. Now let's see the imputed values.

```
data$Bare_nuclei[mis]

##  [1] 7.743293 8.337288 1.165564 1.166287 1.173149 1.172365 1.169921
1.158993
##  [9] 1.154249 7.724768 1.179019 1.200815 7.831484 1.161845 1.157903
1.168688
```

But here our predictor variables are of integer type. So, we need conversion.

```
data$Bare_nuclei=round(data$Bare_nuclei)
data$Bare_nuclei[mis]

##  [1] 8 8 1 1 1 1 1 1 1 8 1 1 8 1 1 1
```

Spliting the Data into Train-Test

We will now divide the data into 4:1 ratio respectively for Train and Test data. We will chose randomly 80% data for training set and remaining for test set.

```
ind=sample(698,0.8*698)
data_train=data[ind,]
dim(data_train)

## [1] 558  10
```

```
data_test=data[-ind,]
dim(data_test)

## [1] 140  10
```

So, training data consists of 558 observations and test data consists of 140 observations.

Renaming the rows of train set -

```
obs=seq(nrow(data_train))
rownames(data_train)=obs
```

Exploratory Data Analysis

In the train data we have 558 observations and 9 predictor variables and we will see the nature of these 9 variables indivisulaly along with the response.

```
library(skimr)
skim(data_train)
```

Data summary

Name	data_train
Number of rows	558
Number of columns	10
_______________________	
Column type frequency:	
factor	1
numeric	9
________________________	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Class	0	1	FALSE	2	2: 368, 4: 190

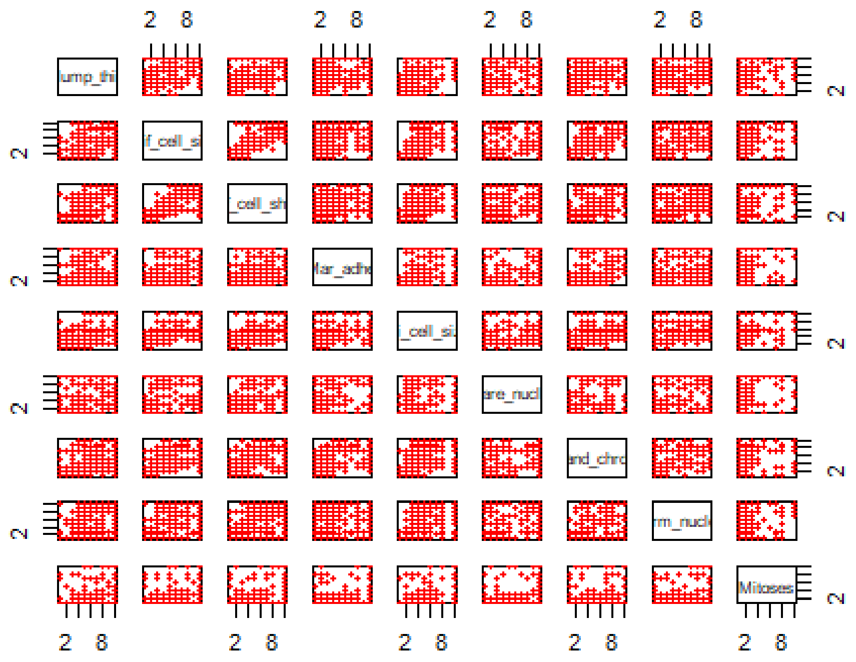
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Clump_thik	0	1	4.39	2.82	1	2	4	6	10	▇▇▇▂▃
Unif_cell_size	0	1	3.13	3.0	1	1	1	5	10	▇▂▁▁▂

				5						
Unif_cell_shape	0	1	3.17	2.96	1	1	1	5	10	▇▂▁▁▁
Mar_adhe	0	1	2.76	2.84	1	1	1	3	10	▇▂▁▁▁
Epi_cell_size	0	1	3.21	2.25	1	2	2	4	10	▇▂▁▁▁
Bare_nuclei	0	1	3.51	3.63	1	1	1	6	10	▇▁▁▁▂
Bland_chrom	0	1	3.46	2.48	1	2	3	5	10	▇▅▁▂▁
Norm_nucleoli	0	1	2.78	2.98	1	1	1	3	10	▇▁▁▁▁
Mitoses	0	1	1.56	1.63	1	1	1	1	10	▇▁▁▁▁

Here 368 observations belongs to benign class and 190observations belong to Malign class. We see that distribution of every predictor is positively skewed with a very thin and large tail. We can observe that ‘Uniformity of cell size’, ‘Uniformity of cell shape’, ‘Marginal adhesion’, ‘Bare nuclei’, ‘Normal nucleoli’ and ‘Mitoses’ have 50% observation as 1 where 50% observations for ‘Epithelial cell size’ and ‘Bland Chromatin’ are less than equal to 2 and 3 respectively.We will check if there is any relationship among the predictors.

```r
plot(data_train[,2:10],pch=20,col='red')
```



Correlation between the variables -

```
cor(data_train[,2:10])

##                 Clump_thik Unif_cell_size Unif_cell_shape  Mar_adhe
## Clump_thik       1.0000000      0.6328403       0.6498360 0.4773114
## Unif_cell_size   0.6328403      1.0000000       0.9078150 0.6981402
## Unif_cell_shape  0.6498360      0.9078150       1.0000000 0.6671898
## Mar_adhe         0.4773114      0.6981402       0.6671898 1.0000000
## Epi_cell_size    0.5293134      0.7693373       0.7308854 0.6031878
## Bare_nuclei      0.5975497      0.6894907       0.7244086 0.6632204
## Bland_chrom      0.5519307      0.7602150       0.7381310 0.6737169
## Norm_nucleoli    0.5337881      0.7090756       0.7132110 0.5923707
## Mitoses          0.3369923      0.4371254       0.4251457 0.4162253
##                 Epi_cell_size Bare_nuclei Bland_chrom Norm_nucleoli
Mitoses
## Clump_thik          0.5293134   0.5975497   0.5519307     0.5337881
0.3369923
## Unif_cell_size      0.7693373   0.6894907   0.7602150     0.7090756
0.4371254
## Unif_cell_shape     0.7308854   0.7244086   0.7381310     0.7132110
0.4251457
## Mar_adhe            0.6031878   0.6632204   0.6737169     0.5923707
0.4162253
## Epi_cell_size       1.0000000   0.6086470   0.6144944     0.6287388
0.4661250
## Bare_nuclei         0.6086470   1.0000000   0.6928189     0.5864210
```

```
0.3445016
## Bland_chrom          0.6144944   0.6928189   1.0000000     0.6568262
0.3438762
## Norm_nucleoli        0.6287388   0.5864210   0.6568262     1.0000000
0.4274455
## Mitoses              0.4661250   0.3445016   0.3438762     0.4274455
1.0000000
```

We see that there is no significant relationship except the pair 'Uniformity of cell size' and 'Uniformity of cell shape'(0.907815).

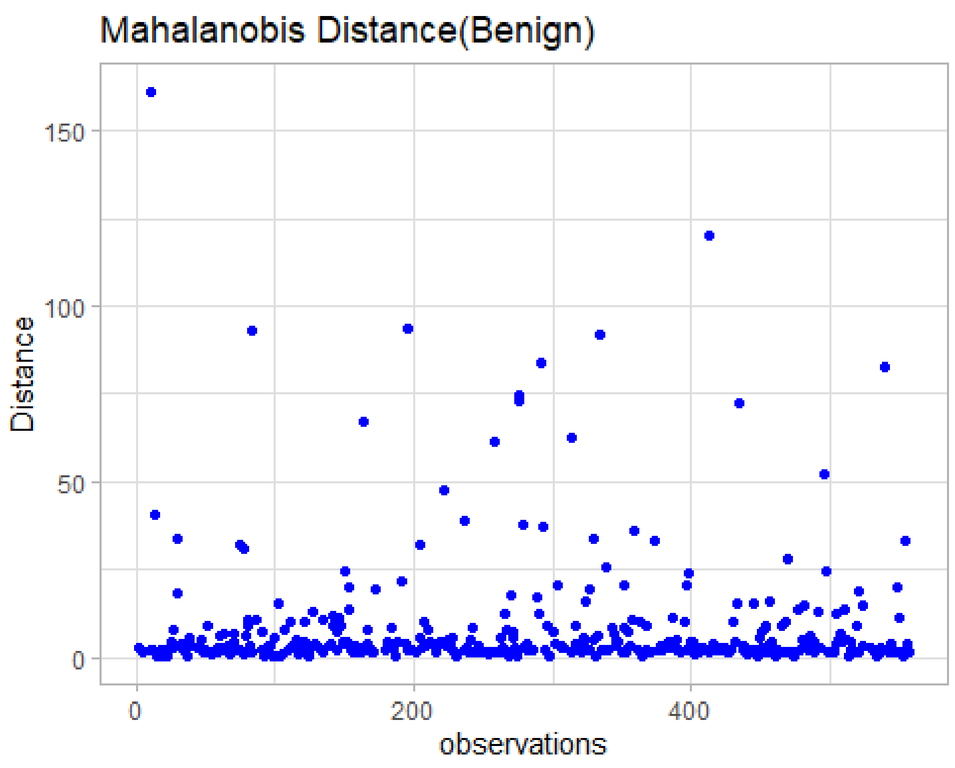
Outliers

We will find outliers separately for class 'Benign' and 'Malign'. We will use **Mahalanobis Distance** to detect otliers.

```
x=data_train[,2:10]
ind1=which(data_train$Class=='2')
x_ben=x[ind1,]
x_mal=x[-ind1,]
```

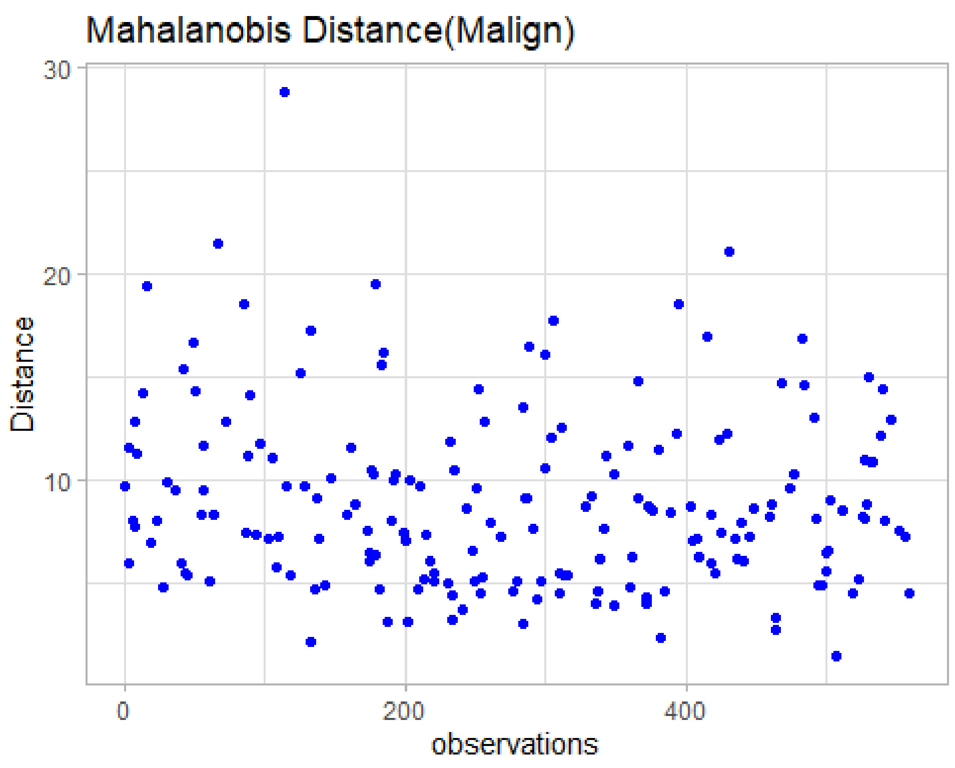
x is the matrix of 9 predictor variables and `x_ben` and `x_mal` correspond to 'Benign' and 'Malign' class respectively. Now we plot 'Mahalanobis Distance' for each point and check if there is any unusual distance (which implies outliers).

```
md1=mahalanobis(x_ben,colMeans(x_ben),cov(x_ben))
library(ggplot2)

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin

obs1=obs[ind1]
df1=as.data.frame(obs1,md1)
ggplot(df1,aes(x=obs1,y=md1))+geom_point(col='blue')+ggtitle('Mahalanobis
Distance(Benign)')+labs(x='observations',y='Distance')+theme_light()
```



It seems that the point whose distance is greater than 150 is unusual.So, we keep track of it.

```
out=which(md1>150)
md2=mahalanobis(x_mal,colMeans(x_mal),cov(x_mal))
obs2=obs[-ind1]
df2=as.data.frame(obs2,md2)

## Warning in as.data.frame.integer(obs2, md2): 'row.names' is not a
character
## vector of length 190 -- omitting it. Will be an error!

ggplot(df2,aes(x=obs2,y=md2))+geom_point(col='blue')+ggtitle('Mahalanobis
Distance(Malign)')+labs(x='observations',y='Distance')+theme_light()
```



Here the point greater than 25 looks like unusual , so we will puu it on out.

```
out=append(out,which(md2>25))
out

##  11 114
##   4  38

out=as.numeric(names(out)) #store the rowname such that it can be easily
detected from train data
```

So, we regard 11th and 114th point of train data as outliers.

Classification

We will build classification models based on various methods, each with the outliers and without the outliers, and will check difference in the accuracy. In this cancer classification process **False Negative** (cells predicted as benign but actually malign) is considered as most severe error. Better model will be judged on the basis of percentage of **False Negative** and **Accuracy**.

We will create a separate dataset without outliers.

```
data_train_out=data_train[-out,]
dim(data_train_out)
```

```
## [1] 556  10
```

```
train_class=data_train[,1] #train data response
train_class_out=data_train_out[,1]
test_class=data_test[,1]  #test data response
```

K-Nearest Neighbour

KNN is a simple algorithm that stores all availabe cases and classifies new cases based on distance function. A case is classified by majority vote of its neighbour with the case being assigned to the class most common among its K nearest neighbours measured by a distance function. If K=1 then the new case simply assigned to the class of nearest neighbour. Default distance (between two vector **x** and **y**) function is -

$$Euclidean = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Other distance functions are **Manhattan**, **Minkowski** etc.

Now we use it to classify our data.

```
library(class)
test_class_pred_k1=knn(train=data_train,test=data_test,cl=train_class,k=10)
#we use 10 nearest neighbour
tab_k1=table(test_class_pred_k1,test_class)
```

Now we make function to check percentage of accuracy , mis-classification and false negative error.

```
#error function
err=function(tab){
  print('The Confusion Matrix is :')
  print(tab)
  a=(sum(diag(tab)))/(sum(tab))
  m=1-a
  f=tab[1,2]/(sum(tab))
  t=c(a,m,f)*100
  names(t)=c('Accuracy %','Mis-classification %','False Negative %')
  print(t)
}
err(tab_k1)
```

```
## [1] "The Confusion Matrix is :"
##                    test_class
## test_class_pred_k1  2  4
##                   2 88  3
##                   4  1 48
##           Accuracy % Mis-classification %     False Negative %
##            97.142857             2.857143             2.142857
```

Now we check without outliers.

```
test_class_pred_k2=knn(train = data_train_out, test = data_test,
cl=train_class_out,k=10 )
tab_k2=table(test_class_pred_k2,test_class)
err(tab_k2)
```

```
## [1] "The Confusion Matrix is :"
##                     test_class
## test_class_pred_k2   2  4
##                    2 88  3
##                    4  1 48
##            Accuracy % Mis-classification %     False Negative %
##             97.142857             2.857143             2.142857
```

We see that accuracy is good and there is no effect of outliers.

Logistic Regression

Logistic Regression is used to predict binary response variable. Though it may be used as classifier (binary) by setting a threshold value. Let the response variable Y takes values 0 and 1. Then take

$$P(Y=1)=E(Y)=\frac{1}{1+e^{-(\beta_0+\Sigma_{j=1}^{p}\beta_j Xj)}}$$

where β_j's are regression coefficient,X_j is j_{th} predictor. Logistic regression finds β_j by IRLS method. y_i is predicted as

$$\widehat{y_i}=\frac{1}{1+e^{-(\widehat{\beta_0}+\Sigma_{j=1}^{p}\widehat{\beta_J}X_{ji})}}$$

Suppose we chose a threshold 0<p<1. If $\widehat{y_i}$ > p then it is assigned to '1', otherwise assigned to '0'.

Now we use it to classify our data.

```
model=glm(Class~.,data=data_train,family = binomial) #'binary' is used for
two class problem
summary(model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial, data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2872  -0.1254  -0.0614   0.0273   2.5774
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -10.19518    1.28495  -7.934 2.12e-15 ***
## Clump_thik        0.61022    0.14703   4.150 3.32e-05 ***
## Unif_cell_size   -0.07711    0.19107  -0.404  0.68651
## Unif_cell_shape   0.30715    0.21000   1.463  0.14358
## Mar_adhe          0.13296    0.12743   1.043  0.29677
## Epi_cell_size     0.17297    0.16293   1.062  0.28840
## Bare_nuclei       0.36688    0.09420   3.895 9.83e-05 ***
## Bland_chrom       0.49567    0.16323   3.037  0.00239 **
## Norm_nucleoli     0.11882    0.10368   1.146  0.25179
## Mitoses           0.72320    0.30172   2.397  0.01654 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 715.766  on 557  degrees of freedom
## Residual deviance:  98.987  on 548  degrees of freedom
## AIC: 118.99
##
## Number of Fisher Scoring iterations: 8
```

```
prob=predict(model,data_test,type='response')
```

prob will give the probability that the response is ‘1’. But for our case it should be ‘4’. Now we check the correspondence between ‘1’ and ‘4’.

```
contrasts(data_train$Class)
```

```
##   4
## 2 0
## 4 1
```

We see that ‘1’ corresponds to ‘4’ here.

```
pred=rep(2,nrow(data_test))
#As cancer case is sensative, we chose a low threshold 0.25
pred[prob>0.25]=4
tab_l1=table(pred,test_class)
err(tab_l1)
```

```
## [1] "The Confusion Matrix is :"
##     test_class
## pred  2  4
##    2 87  3
##    4  2 48
##            Accuracy % Mis-classification %     False Negative %
##             96.428571             3.571429             2.142857
```

Now we check without outliers.

```
model1=glm(Class~.,data=data_train_out,family = binomial)
prob1=predict(model1,data_test,type='response')
```

```
pred1=rep(2,nrow(data_test))
pred1[prob>0.25]=4
tab_l2=table(pred1,test_class)
err(tab_l2)

## [1] "The Confusion Matrix is :"
##      test_class
## pred1  2  4
##     2 87  3
##     4  2 48
##          Accuracy % Mis-classification %     False Negative %
##           96.428571             3.571429             2.142857
```

We see that accuracy is good and low error rates in both cases.

Linear Discriminant Analysis

Cosider a training set consists of feature vectors $\underset{\sim}{x}$ with known class y. The classification problem is then to find a good predictor for the class y of any sample of the same distribution (not necessary from the trainig set) given only feature vector $\underset{-}{x}$.

LDA approaches the problem by assumig **p($\underset{\sim}{x}$|y=0)** and **p($\underset{\sim}{x}$|y=1)** are both normally distributed with **N($\underset{\sim}{\mu_0}$, $\sum$)** and **N($\underset{\sim}{\mu_1}$, $\sum$)** respectively. Under this assumption the Bayes optimal solution is to predict the point as being from the class ‘1’ if

$$(\underset{\sim}{\mu_1} - \underset{\sim}{\mu_0})^T \sum^{-1} \underset{\sim}{x} > \frac{1}{2}(\underset{\sim}{\mu_1} - \underset{\sim}{\mu_0})^T \sum^{-1} (\underset{\sim}{\mu_1} + \underset{\sim}{\mu_0})$$

$\underset{\sim}{\mu_0}, \underset{\sim}{\mu_1}$ are estimated from class ‘0’ and class ‘1’ (for our case ‘2’ and ‘4’ respectively) and $\sum$ is estimated from all observations.

Now we use it to classify our data.

```
library(MASS)
model2=lda(Class~.,data=data_train)
model2

## Call:
## lda(Class ~ ., data = data_train)
##
## Prior probabilities of groups:
##         2         4
## 0.6594982 0.3405018
##
## Group means:
##   Clump_thik Unif_cell_size Unif_cell_shape Mar_adhe Epi_cell_size
Bare_nuclei
## 2   2.945652       1.355978        1.434783 1.377717      2.095109
```

```
1.372283
## 4   7.184211       6.552632        6.521053 5.442105      5.378947
7.663158
##   Bland_chrom Norm_nucleoli  Mitoses
## 2    2.103261      1.277174 1.073370
## 4    6.100000      5.684211 2.505263
##
## Coefficients of linear discriminants:
##                          LD1
## Clump_thik      0.18145400
## Unif_cell_size  0.12088820
## Unif_cell_shape 0.08239885
## Mar_adhe        0.01445409
## Epi_cell_size   0.04822144
## Bare_nuclei     0.26313009
## Bland_chrom     0.14010989
## Norm_nucleoli   0.09042823
## Mitoses         0.02529140

pred2=predict(model2,data_test)
names(pred2)

## [1] "class"     "posterior" "x"
```

Here class gives the prediction, posterior gives the posterior probability of each class for all observations, x is linear discriminant.

```
tab_ld1=table(pred2$class,test_class)
err(tab_ld1)

## [1] "The Confusion Matrix is :"
##    test_class
##      2  4
##   2 88  5
##   4  1 46
##            Accuracy % Mis-classification %     False Negative %
##             95.714286             4.285714             3.571429
```

Now we check without outliers.

```
model3=lda(Class~.,data=data_train_out)
pred3=predict(model3,data_test)
tab_ld2=table(pred3$class,test_class)
err(tab_ld2)

## [1] "The Confusion Matrix is :"
##    test_class
##      2  4
##   2 88  6
##   4  1 45
##           Accuracy % Mis-classification %     False Negative %
##            95.000000             5.000000             4.285714
```

We see that the model built with outliers gives a better result and it has very good accuracy , low errors.

Naive Bayes Classifier

Naive Bayes Classifier deals with independent set of features. Given a new observation represented by vector $\underset{\sim}{x} = (x_1, x_2, \cdots, x_n)$ of n features, it assigns probabilities $\mathbf{p}(C_k|x_1, x_2, \cdots, x_n)$ for each of k possible classes C_k (in our data k=2) to this new observation . Using Bayes' Theorem we have $p(C_k|x_1, x_2, \cdots, x_n) = \frac{p(C_k)p(\underset{\sim}{x}|C_k)}{p(\underset{\sim}{x})}$

The Bayes classifier is the function that assigns a class label $\hat{y} = C_k$ for some k if $p(C_k)p(\underset{\sim}{x}|C_k)$ is maximum over k=1(1)k.

Now we use it to classify our data.

```
library(naivebayes)

## naivebayes 0.9.7 loaded

model4=naive_bayes(Class~.,data=data_train,usekernel = T)
pred4=predict(model4,data_test)
tab_n1=table(pred4,test_class)
err(tab_n1)

## [1] "The Confusion Matrix is :"
##      test_class
## pred4  2  4
##     2 88  2
##     4  1 49
##           Accuracy % Mis-classification %     False Negative %
##            97.857143             2.142857             1.428571
```

Now we check without outliers.

```
model5=naive_bayes(Class~.,data=data_train_out,usekernel = T)
pred5=predict(model5,data_test)

## Warning: predict.naive_bayes(): more features in the newdata are provided
as
## there are probability tables in the object. Calculation is performed based
on
## features to be found in the tables.

tab_n2=table(pred5,test_class)
err(tab_n2)
```

```
## [1] "The Confusion Matrix is :"
##      test_class
## pred5  2  4
##     2 88  2
##     4  1 49
##           Accuracy % Mis-classification %     False Negative %
##            97.857143             2.142857             1.428571
```

We see that accuracy is good and low error rates in both cases. Here we use Kernel by `usekernel=T`. Now we check if use of kernel is better. `usekernel=F` is default setting.

```
model6=naive_bayes(Class~.,data=data_train)
pred6=predict(model6,data_test)
```

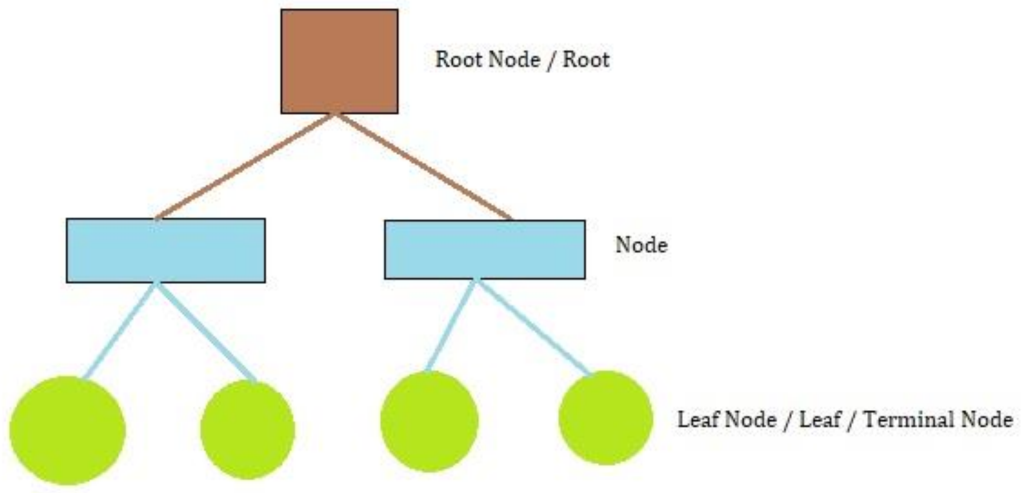
```
## Warning: predict.naive_bayes(): more features in the newdata are provided
as
## there are probability tables in the object. Calculation is performed based
on
## features to be found in the tables.
```

```
tab_n3=table(pred6,test_class)
err(tab_n3)
```

```
## [1] "The Confusion Matrix is :"
##      test_class
## pred6  2  4
##     2 86  1
##     4  3 50
##           Accuracy % Mis-classification %     False Negative %
##           97.1428571            2.8571429            0.7142857
```

Accuracy is differed by 0.7% (we can infer both accuracy is fine) but kernel-less model reduces 50% false negative error , so we consider this model as better.

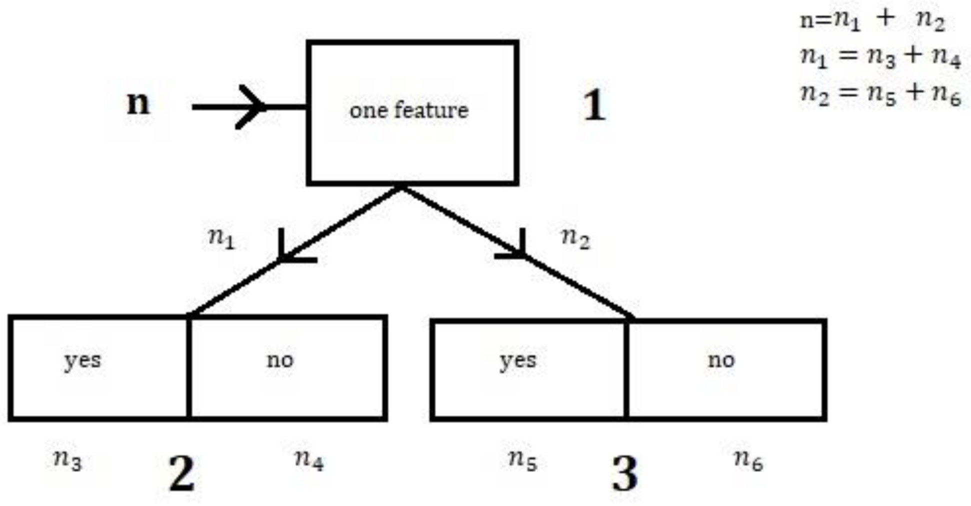
Decision Tree



Nodes in a Tree

The above picture is a example of a decision tree. We put features and their cut_off in the nodes to divide the observations and leaf node gives the output class. To chose a feature in each node Gini impurity is used and that feature is selected whose Gini impurity is minimum. To set a cut-off for a numerical feature, arrange the observations in increasing order w.r.t. that particular feature, then find average values of that feature of consecutive observations, check Gini impurity by chosing each average as cut-off and chose that average as cut-off for which the Gini-impurity is minimum.

Let there are **n** observations with response 'yes' and 'no'-



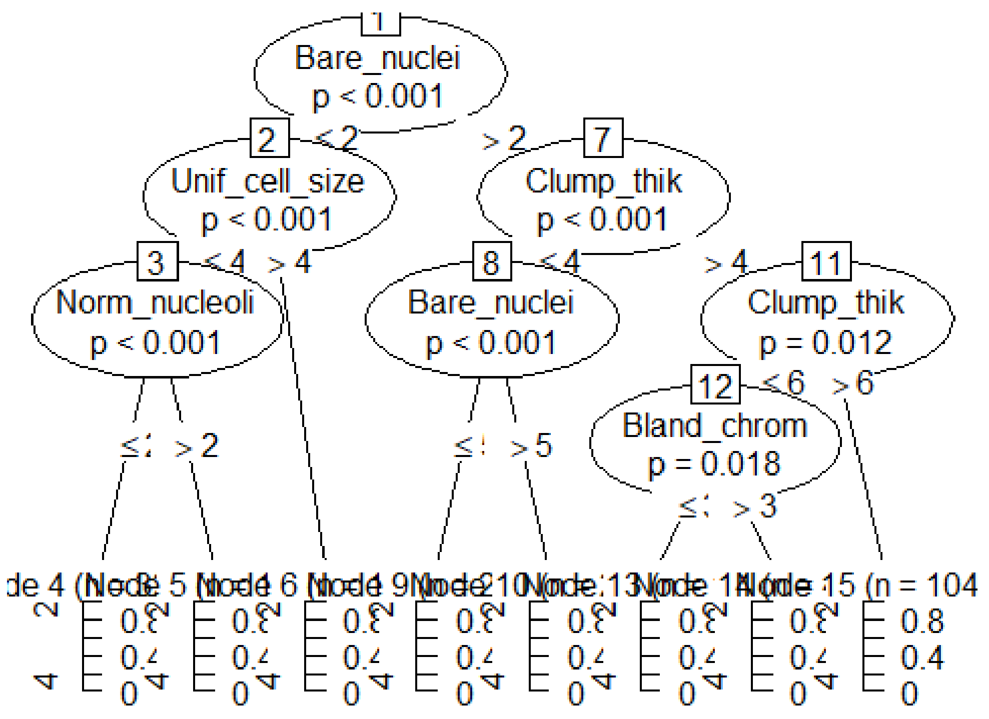
For the above picture , Gini impurity for node 2, $g_1 = 1 - (\frac{n_3}{n_1})^2 - (\frac{n_4}{n_1})^2$

Gini impurity for node 3, $g_2 = 1 - (\frac{n_5}{n_2})^2 - (\frac{n_6}{n_2})^2$

Gini impurity for node 1, $g = \frac{n_1}{n} g_1 + \frac{n_2}{n} g_2$

Now we use it to classify our data.

```
library(party)

## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

tree=ctree(Class~.,data=data_train,controls = ctree_control(mincriterion =
0.9,minsplit = 30))
plot(tree)
```



```
pred7=predict(tree,data_test)
tab_d1=table(pred7,test_class)
err(tab_d1)

## [1] "The Confusion Matrix is :"
##      test_class
## pred7  2  4
##     2 88  5
##     4  1 46
##          Accuracy % Mis-classification %     False Negative %
##           95.714286             4.285714             3.571429
```

Here `mincriterion=0.9` means a variable is used if there is 90% confidence that the variable is significant. `minsplit=30` means a node will further be splited if there is atleast 30 observations.

Now we check without outliers.

```
tree1=ctree(Class~.,data=data_train_out,controls = ctree_control(mincriterion
= 0.9,minsplit = 30))
pred8=predict(tree1,data_test)
tab_d2=table(pred8,test_class)
err(tab_d2)

## [1] "The Confusion Matrix is :"
##      test_class
## pred8  2  4
##     2 88  5
##     4  1 46
##           Accuracy % Mis-classification %     False Negative %
##            95.714286             4.285714             3.571429
```

We see that accuracy is good and low error rates in both cases.

Random Forest Classification

It uses multiple decisison trees on the basis of samples designed by Bootstrap method. Then run a new observation through all the trees and then it is assigned to a class whose frequency is highest.

Bootstrap method uses 'With Replacement' ploicy to collect **n** sample from population of size **n**. So, clearly many observations are not taken into account but their class is known. Run this 'out of bag' observation through the trees and check how much proportion of the trees incorrectly classify the observation. This proportion is termed as 'out of bag' error.

Now we use it to classify our data.

```
library(randomForest)
model9=randomForest(Class~.,data=data_train,ntree=200,mtry=3) #mtry is no. of
feature used randomly to form a node of a tree.mtry is generally taken as
sqroot of no. of features.
model9

##
## Call:
##  randomForest(formula = Class ~ ., data = data_train, ntree = 200,
mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 3.76%
## Confusion matrix:
##     2   4 class.error
## 2 354  14  0.03804348
## 4   7 183  0.03684211

pred9=predict(model9,data_test)
tab_r1=table(pred9,test_class)
err(tab_r1)

## [1] "The Confusion Matrix is :"
##      test_class
## pred9  2  4
##     2 88  2
##     4  1 49
##           Accuracy % Mis-classification %     False Negative %
##            97.857143             2.142857             1.428571
```

Now we check without outliers.

```
model10=randomForest(Class~.,data=data_train_out,ntree=200,mtry=3)
pred10=predict(model10,data_test)
tab_r2=table(pred10,test_class)
err(tab_r2)

## [1] "The Confusion Matrix is :"
##       test_class
## pred10  2  4
##      2 88  2
##      4  1 49
##           Accuracy % Mis-classification %     False Negative %
##            97.857143             2.142857             1.428571
```

Now we vary `ntree` and `mtry` and check the best values of these using `tune()`.

```
library(e1071)
tune_out=tune(randomForest,Class~.,data=data_train,ranges =
list(ntree=c(20,50,75,100,150,200,250,300),mtry=c(1:9)))
tune_out$best.parameters #best parameters among the given parameters

##   ntree mtry
## 6   200    1

bmodel=tune_out$best.model #best model using best parameter
pred11=predict(bmodel,data_test) #prediction using best model
tab_r3=table(pred11,test_class)
err(tab_r3)

## [1] "The Confusion Matrix is :"
##       test_class
## pred11  2  4
##      2 88  1
##      4  1 50
##           Accuracy % Mis-classification %     False Negative %
##           98.5714286            1.4285714            0.7142857
```

Here the new model (best) produces the best result

Support Vector Machine

In SVM algorithm, we plot each data item as a point in n_dimensional space (where n is number of feature) with the value of each feature being the value of a particular co-ordinate. Then we perform the Classification by finding the hyperplane that differentiates the two classes very well. The points which are closest to hyperplane are termed as support vectors. The area between the separator (hyperplane) and the support vectors are called **margin**.

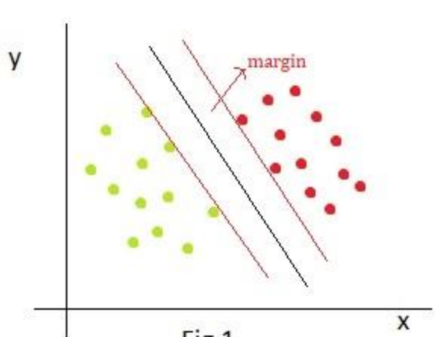


Fig 1

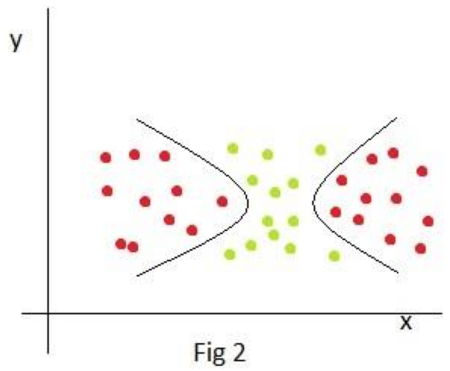


Fig 2

svm

There may be situation where we can't divide the points by a hyperplane. For this, SVM algorithm has **kernel** trick. A kernel is a function that quantifies the similarity of two observations. It converts non-separable problem to separable problem by using some higher dimentional curve i.e. non-linear separation (using this trick we can classify more than two classes). Example of kernel are *radial*,*polynomial*. Radial kernel is

$$K(x_i, x_{i'}) = e^{(-\gamma \sum_{j=1}^{p}(x_{ij}-x_{i'j})^2)}$$

Now we use it to classify our data.

```
library(e1071)
model12=svm(Class~.,data=data_train,kernel='linear',cost=10,scale = F)
model12
```

```
##
## Call:
## svm(formula = Class ~ ., data = data_train, kernel = "linear", cost = 10,
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  46
```

```
model12$index #index of support vectors
```

```
##  [1]  56  64 103 108 115 175 178 179 185 243 285 288 341 388 403 407 420
431 441
## [20] 448 482 502 522  75  78  83  99 135 164 195 222 237 258 275 279 292
293 314
## [39] 329 334 374 398 435 497 520 539
```

```
pred12=predict(model12,data_test)
tab_s1=table(pred12,test_class)
err(tab_s1)
```

```
## [1] "The Confusion Matrix is :"
##        test_class
## pred12  2  4
##      2 88  3
##      4  1 48
##            Accuracy % Mis-classification %     False Negative %
##             97.142857             2.857143             2.142857
```

Now we check without outliers.

```
model13=svm(Class~.,data=data_train_out,kernel='linear',cost=10,scale = F)
model13$index #index of support vectors
```

```
##  [1]  55  63 102 107 113 173 176 177 183 241 283 286 339 386 401 405 418
429 439
## [20] 446 480 500 520  74  77  82  98 133 162 193 220 235 256 273 277 290
291 312
## [39] 327 332 372 396 433 495 518 537
```

```
pred13=predict(model13,data_test)
tab_s2=table(pred13,test_class)
err(tab_s1)
```

```
## [1] "The Confusion Matrix is :"
##        test_class
## pred12  2  4
##      2 88  3
##      4  1 48
##            Accuracy % Mis-classification %     False Negative %
##             97.142857             2.857143             2.142857
```

We can also include gamma in the model.Now we will vary `kernel`,`cost` and gamma and look for best parameters.

```
tune.out=tune(svm,Class~.,data=data_train,
  ranges =
list(kernel=c('linear','radial'),cost=c(0.001,0.01,0.1,1,5,10,100),gamma=c(0.
5,1,2,3,4)))
tune.out$best.parameters
```

```
##   kernel cost gamma
## 3 linear 0.01   0.5
```

```
model14=tune.out$best.model
pred14=predict(model14,data_test)
tab_s3=table(pred14,test_class)
err(tab_s3)
```

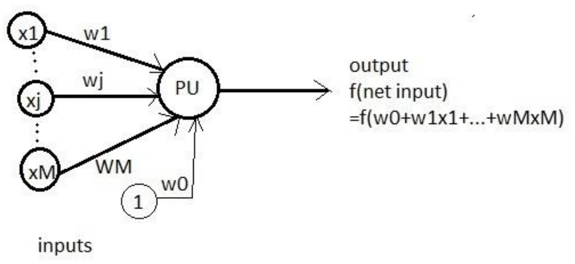
```
## [1] "The Confusion Matrix is :"
##        test_class
## pred14  2  4
##      2 88  2
##      4  1 49
##           Accuracy % Mis-classification %     False Negative %
##            97.857143             2.142857            1.428571
```

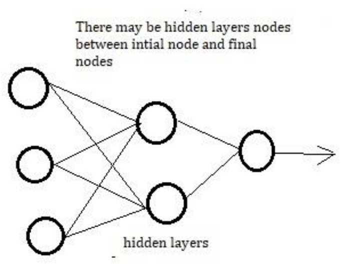
So,as expected, tuned model gives the best result.

Artificial Neural Network

ANN is crude networks of neurons based on the neural structure of the brain. They processes record one at a time and learn by comparing their classification of the record with the known actual classification of the record. The errors in the initial classification of the first record is fed back into the network, used to modify the networks algorithm for further iterations.

A neuron in an ann is




There may be hidden layers nodes between intial node and final nodes
hidden layers

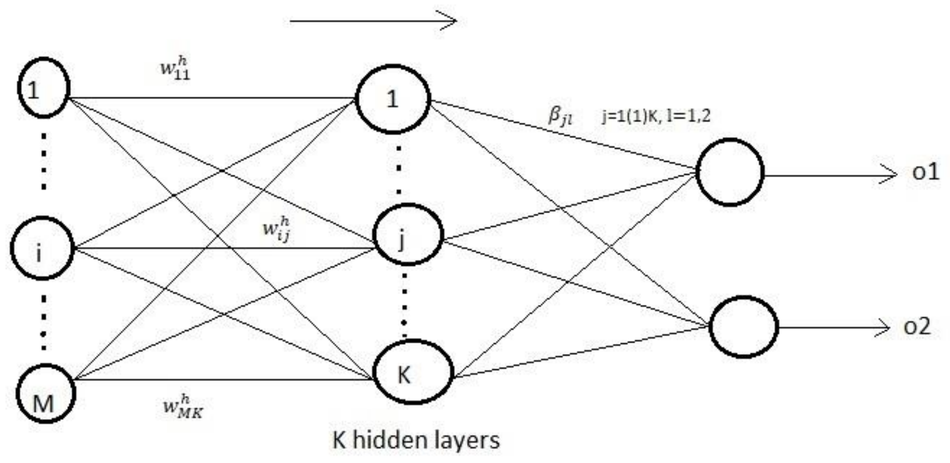
(i) A set of input values(x) and associated weights(w) and a bias input 1(or 0) with weight w_0

(ii) output of the net input $w_0 + \sum w_i x_i$ through a function **f** i.e. f($w_0 + \sum w_i x_i$)

(iii) **PU** collects the inputs from all input node and converts to net input $w_0 + \sum w_i x_i$, **PU** process the input value and generates the output

f($w_0 + \sum w_i x_i$). Examples of **f** are **f**(x)=$\frac{1}{1+e^{-x}}$ (sigmoid transfer function), **f**(x)=tanh(X) (tanh transfer function)

Note that there may be hidden node shown in teh right hand side of the above picture.



Let (x_p, y_p) be the training observation, p=1(1)n.

For j_{th} hidden node, input $net_{pj}^h = \sum_{i=1}^M w_{ij}\, x_{pi}$ (ignoring bias)

output $f(net_{pj}^h) = i_{pj}$ (say)

For l_{th} final node, input $T_l = \sum_{j=1}^K \beta_{jl}\, i_{pj}$

output $o_l = \frac{e^{T_l}}{\sum_{l=1}^L e^{T_l}}$ (for our case L=2)

New observation is classified to class 'l' if o_l is maximum over $l = 1(1)L$

Note that different **f** may be used in different node.

Weights w_{ij}, β_{jl} are estimed by minimizing the sum of square error $\sum_{p=1}^n \sum_{l=1}^L (y_{pl} - o_l)^2$

Now we use it to classify our data. Here firstly we standardize our features.

```
train_norm=data_train
test_norm=data_test
for(i in 2:10){
  train_norm[,i]=(train_norm[,i]-min(train_norm[,i]))/(max(train_norm[,i])-
min(train_norm[,i]))
  test_norm[,i]=(test_norm[,i]-min(test_norm[,i]))/(max(test_norm[,i])-
min(test_norm[,i]))
}
train_norm_out=train_norm[-out,]
library(neuralnet)
model15=neuralnet(Class~.,data=train_norm,hidden = 3,err.fct =
'sse',linear.output = F)
plot(model15)
output1=compute(model15,test_norm[,-1])
p1=output1$net.result #gives the probability of being classified as first and
second class for each observation
pred15=ifelse(p1[,2]>0.25,4,2) #defining if prob(being '4')>o.25 then ,
classified as '4'.As cancer is case sensitive we use a lower threshold 0.25
tab_a1=table(pred15,test_class)
err(tab_a1)
```

```
## [1] "The Confusion Matrix is :"
##       test_class
## pred15  2  4
##      2 88  2
##      4  1 49
##          Accuracy % Mis-classification %     False Negative %
##           97.857143             2.142857             1.428571
```

Now we check without outliers.

```
model16=neuralnet(Class~.,data=train_norm_out,hidden = 3,err.fct =
'sse',linear.output = F)
output2=compute(model16,test_norm[,-1])
p2=output2$net.result
pred16=ifelse(p2[,2]>0.25,4,2)
tab_a2=table(pred16,test_class)
err(tab_a2)
```

```
## [1] "The Confusion Matrix is :"
##       test_class
## pred16  2  4
##      2 86  0
##      4  3 51
##          Accuracy % Mis-classification %     False Negative %
##           97.857143             2.142857             0.000000
```

We see that later case provides better result with zero false negative error. So, here excluding outliers is effective.

Now we check what will happen if we take two layers, 3 nodes at first layer and a single node in second hidden layer. We will use the data without outlier.

```
model17=neuralnet(Class~.,data=train_norm_out,hidden = c(3,1),err.fct =
'sse',linear.output = F)
output3=compute(model17,test_norm[,-1])
p3=output3$net.result
pred17=ifelse(p3[,2]>0.25,4,2)
tab_a3=table(pred17,test_class)
err(tab_a3)
```

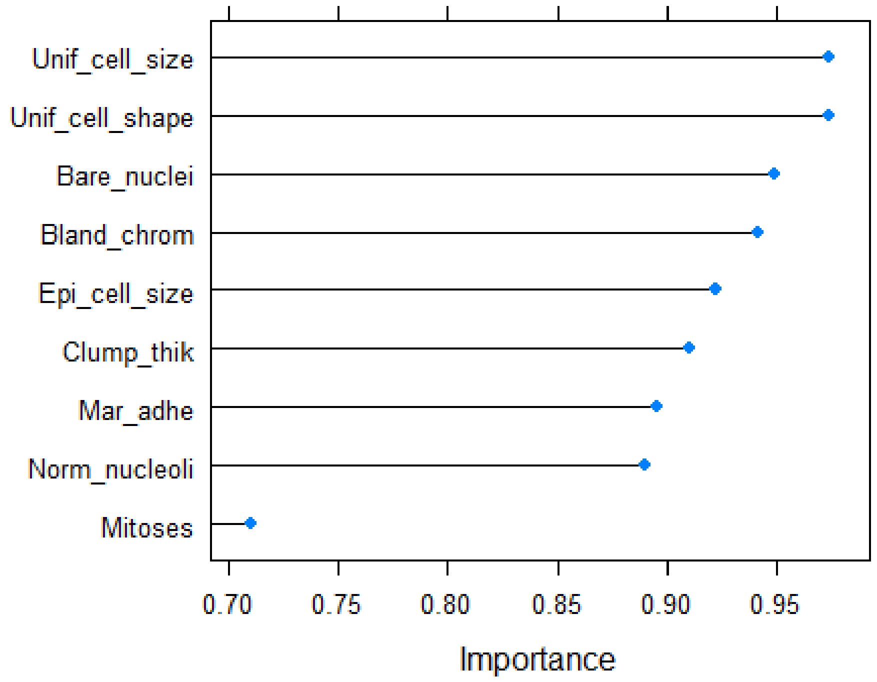
```
## [1] "The Confusion Matrix is :"
##       test_class
## pred17  2  4
##      2 87  1
##      4  2 50
##          Accuracy % Mis-classification %     False Negative %
##          97.8571429            2.1428571            0.7142857
```

It gives good results but not better than before as error in false negative increases here.

Important Features

Now we will check importance of each features based on train set. We use 10-fold Cross Validation (CV) which is repeated 5 times on the basis of the method learning vector quantization (lvq).

```
library(caret)

## Loading required package: lattice

control=trainControl(method='repeatedcv',number = 10,repeats = 5)
m=train(Class~.,data=data,method='lvq',trControl=control)
importance=varImp(m,scale=F)
print(importance)

## ROC curve variable importance
##
##                    Importance
## Unif_cell_size      0.9740
## Unif_cell_shape     0.9735
## Bare_nuclei         0.9484
## Bland_chrom         0.9411
## Epi_cell_size       0.9218
## Clump_thik          0.9101
## Mar_adhe            0.8956
## Norm_nucleoli       0.8897
## Mitoses             0.7101

plot(importance)
```



We see that the how much a feture is important (checking their ranks).

Feature Selection

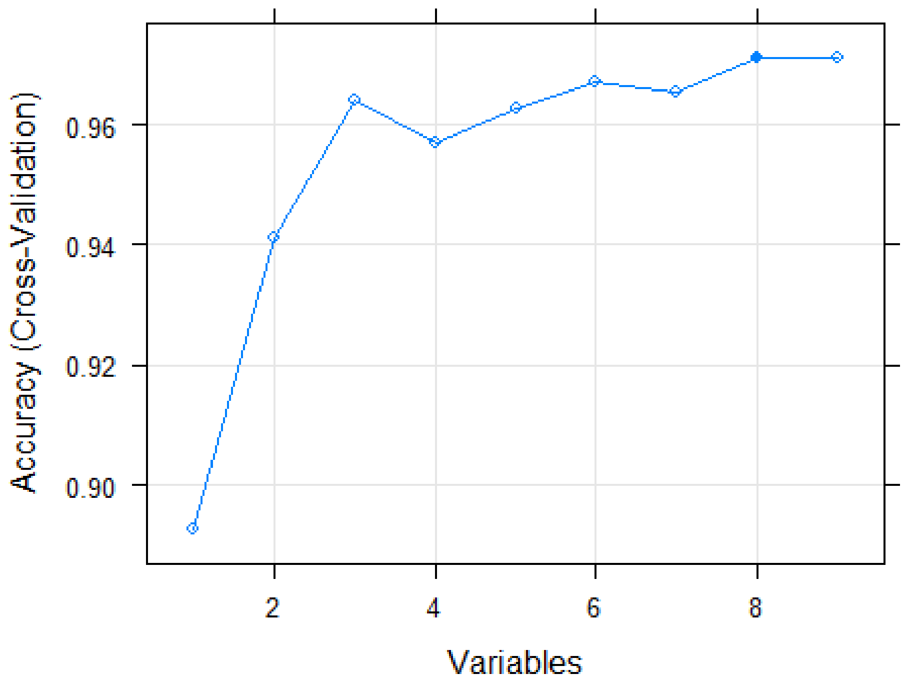
For feature selection we will use Random Forest and check by 10-fold Cross Validation once.

```
control1=rfeControl(functions = rfFuncs, method = 'cv',number=10)
results=rfe(data[,2:10],data[,1],sizes = 1:9, rfeControl = control1)
print(results)

##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          1   0.8924 0.7595    0.04192 0.09486
##          2   0.9412 0.8697    0.03000 0.06524
##          3   0.9641 0.9218    0.02174 0.04600
##          4   0.9570 0.9061    0.02037 0.04217
##          5   0.9627 0.9183    0.02168 0.04675
##          6   0.9670 0.9277    0.02257 0.04889
##          7   0.9656 0.9241    0.01943 0.04296
##          8   0.9713 0.9372    0.02030 0.04467         *
##          9   0.9713 0.9365    0.02248 0.05045
##
## The top 5 variables (out of 8):
##    Bare_nuclei, Clump_thik, Unif_cell_size, Bland_chrom, Unif_cell_shape
```

```
predictors(results)
```

```
## [1] "Bare_nuclei"     "Clump_thik"      "Unif_cell_size"  "Bland_chrom"
## [5] "Unif_cell_shape" "Norm_nucleoli"   "Mar_adhe"        "Epi_cell_size"
```

```
plot(results,type=c('g','o'))
```



From `predictors(results)` and the above graph it is clear that we can remove the feature 'Mitosis' without any loss in accuracy. The top five predictors are Bare_nuclei, Clump_thik, Unif_cell_size, Bland_chrom, Unif_cell_shape and from the graph it is clear that we can use only Bare_nuclei, Clump_thik and Unif_cell_size to get accuracy more than 96% if there is a problem of time and cost .

Conclusion

Every classification model shows accuracy more than 95%, mis_classification error less than 3% and false negative error less than 2.2% (except Decision Tree, there false negative error was 3.57%, accuracy of this model lowest among all models). We can conclude that we are able to build strong models with respect to our data. We see that the most feature is

‘Uniformity of cell size’. ‘Mitosis’ can be removed for classisfication with no loss in accuracy. Overall we can say that our models are pretty good.

Acknowledgement

I would like to thank various websites like www.machinelearningmastery.com, Analytics Vidhya, UCI machine Learning Respiratory, sites.google.com, www.saedsayad.com, www.markdownguide.org etc , Dr. Bharatendra Rai youtube channel, StatQuest youtube channel , WolBerg 1990’s paper on Multisurface method of pattern separation and special thanks to the book ‘Introduction to Statistical Learning’.