DATA ANALYTICS PROJECT

Data Set: Breast Cancer Wisconsin Data

Group Members:

Abhilash R Kashyap	01FB15ECS006
Anirudh S	01FB15ECS038
Anurag C	01FB15ECS046

INDEX

Description of Data

Attribute Information:

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (local variation in radius lengths)

f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Cleaning of Data

a) Converting the 0 values to NA
b) Imputing unknown value with Amelia package

Code:

```
#Converting the 0 values to NA
write.csv(train,'train2.csv')
library(readr)
train2 <- read_csv("C:/abhilash/5th sem/Data Analytics/datasets/breast-cancer-wisconsin-data/train2.csv")

#Deleting the extra column
train2[,2]<-NULL
View(train2)
write.csv(train,'train3.csv')

#Imputed unknown values with Amelia package
library("Amelia", lib.loc="~/R/win-library/3.4")
```

We then divided the dataset into 2 parts:

1) Training Data – consisting of 250 rows
2) Test Data – consisting of 24 rows

Feature Minimisation

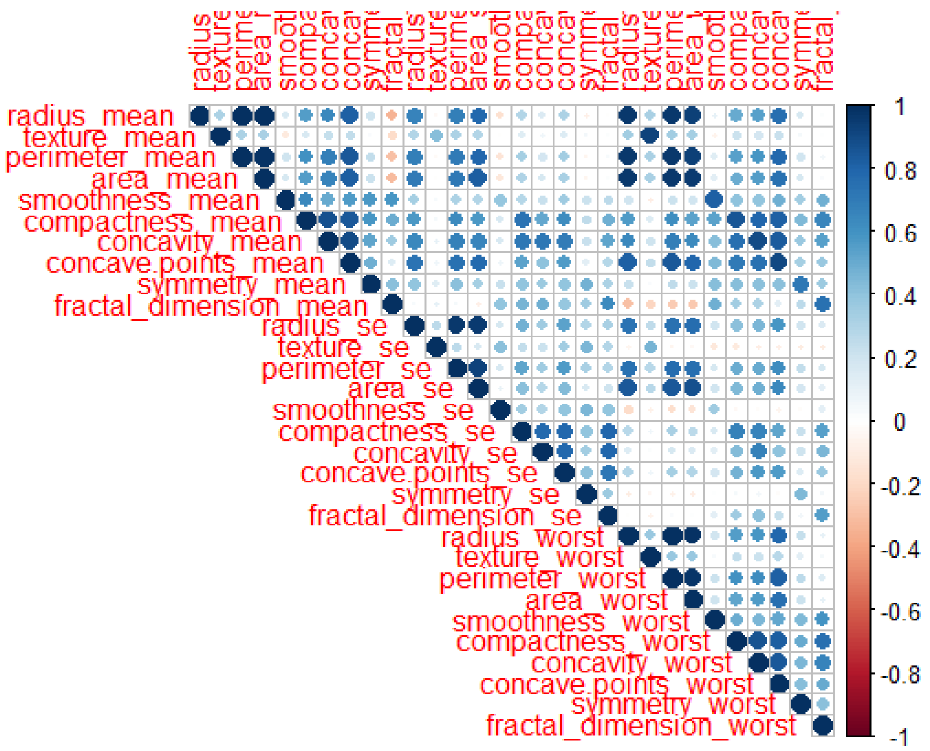
(a) Remove columns with lowest variance after normalization of data , by sorting them in order of variation and separating those with a lower variance (< 0.015). The output of this code becomes the parameters - **Concavity_se , smoothness_se and fractional_dimension_se .**

```
nam <- colnames(Cancer_data)
nam <- nam[4:33]
variances = c()
for(x in nam)
{
  ndf[x] <- (ndf[x] - min(ndf[x]))/(max(ndf[x]) - min(ndf[x]))
  variances[x] <- var(ndf[x])
}
# Sorting the variances
var_ord <- sort(variances)
var_ord

can_drop <- names(var_ord[var_ord < 0.015])
```

(b) Plotting a correlation plot and checking for higher correlation (similarity) between the variables. The following graph is an overview of how the corrplot looks right now.

```
# Using a corrplot to check the correlation oefficients

corrplot(cor(Cancer_data[4:33])  ,  main=" Corrplot" , method = "circle" , type = "upper")
```

Plot : Correlation Plot of all the possible variables in the data set.



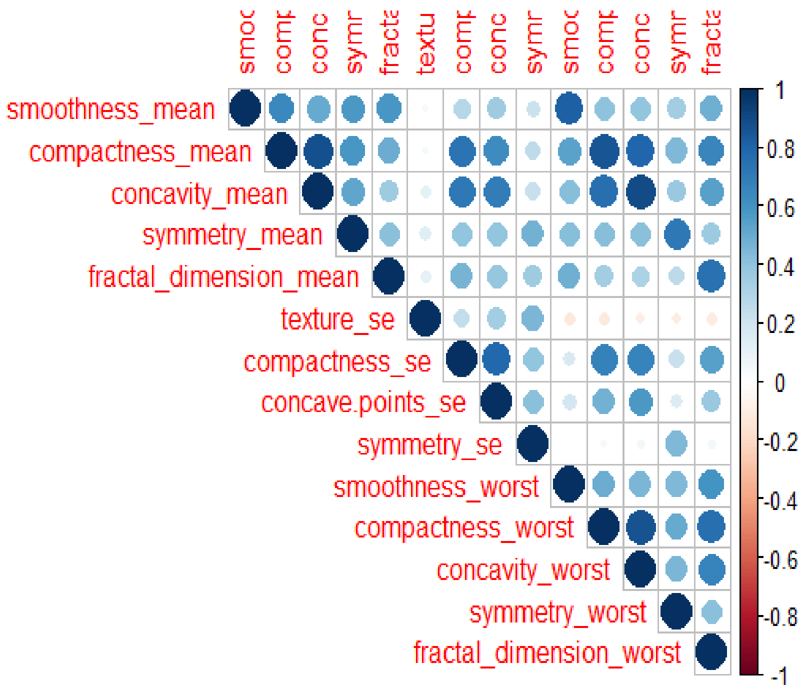
(c) Dropping the attributes which have high correlation with others , and hence updating the parameters which we have to work and classify with. The attributes to be dropped are stored in the drop_att variable , and can_keep specifies the variables to be saved for classification.

```
m <- data.frame(Cancer_data[4:33])

for(i in 1:nrow(m))
{
 for(j in 1:ncol(m))
 {
  if(i!=j)
  {
   r <- m[i,j]
   if(abs(r) > 0.90)
   {
    firstvar[index] <- j
    secondvar[index] <- i
    index <- index + 1
   }
  }
 }
}
drop_att <- unique(c(names(m[,unique(firstvar)]),can_drop))
```

(d) Getting the new , cleaner , and much simpler to understand corrplot. As we can see from the graph , no grid square has a correlation value greater than 0.90 or lesser than -0.90 .

```
corrplot(cor(Cancer_data[,can_keep]))
```

Plot : Correlation Plot of the chosen variables in the data set (after FM).



List of variables to be used in model making is :

can_keep

```
[1] "smoothness_mean"        "compactness_mean"
 [3] "concavity_mean"         "symmetry_mean"
 [5] "fractal_dimension_mean"  "texture_se"
 [7] "compactness_se"         "concave.points_se"
 [9] "symmetry_se"            "smoothness_worst"
[11] "compactness_worst"      "concavity_worst"
[13] "symmetry_worst"         "fractal_dimension_worst" .
```

Problem Statement

Predict whether the cancer is benign or malignant
We are taking dataset and training the models using different classifiers to predict and classify if it is Benign or Malignant.

We are predicting using different classifiers and coming to a conclusion which is the best method.

Naive Bayes classifiers

Literature Survey

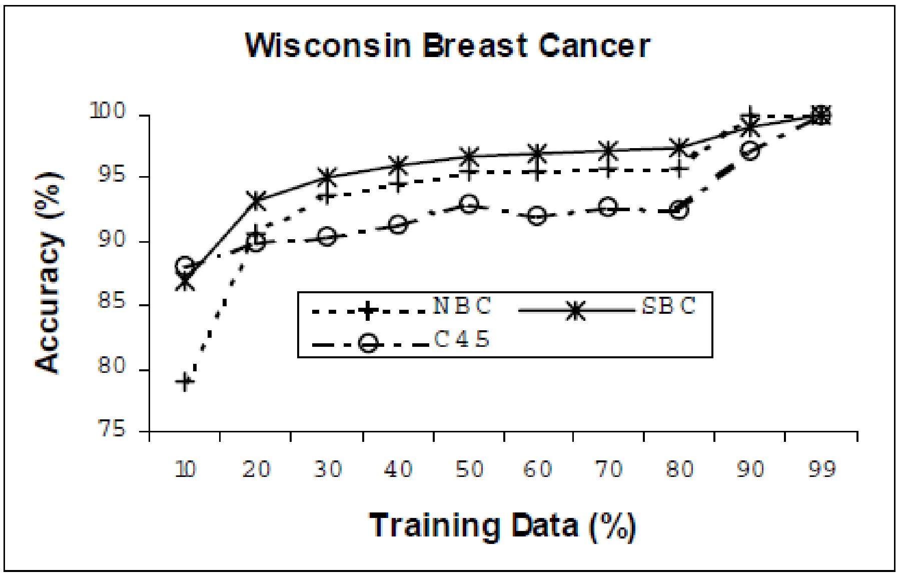
Scaling up the Naive Bayesian Classifier: Using Decision Trees for Feature Selection

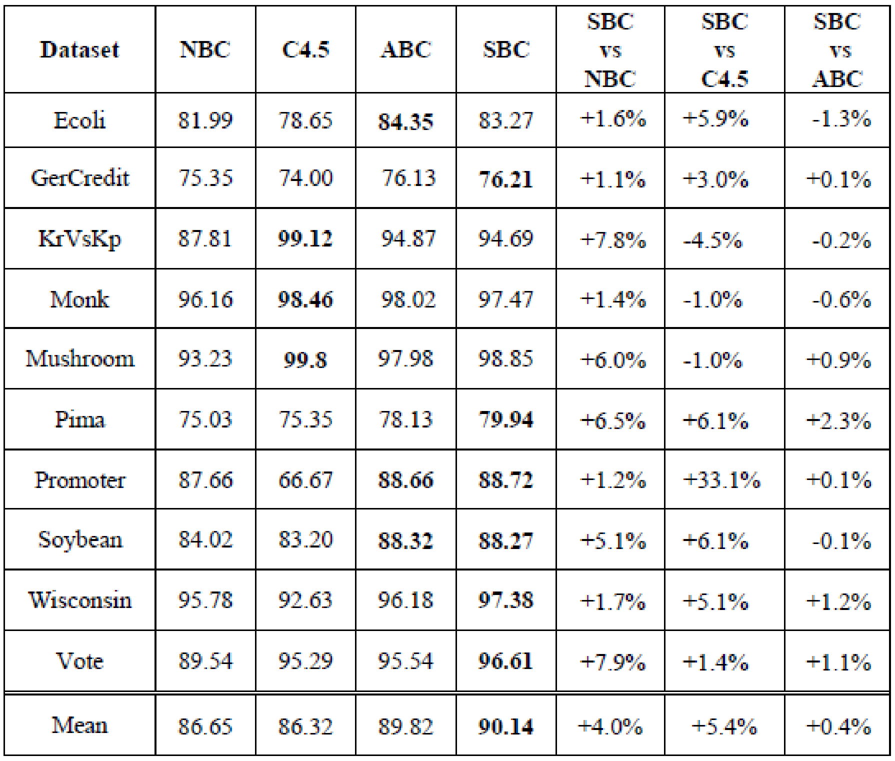
It has been shown that Naïve Bayesian classifier is extremely effective in practice and difficult to improve upon. In this paper, we show that it is possible to reliably improve this classifier by using a feature selection method. Naïve Bayes can suffer from oversensitivity to redundant and/or irrelevant attributes. If two or more attributes are highly correlated, they receive too much weight in the final decision as to which class an example belongs to. This leads to a decline in accuracy of prediction in domains with correlated features. Decision Trees does not suffer from this problem because if two attributes are correlated, it will not be possible to use both of them to split the training set, since this would lead to exactly the same split, which makes no difference to the existing tree. This is one of the main reasons Decision Trees performs better than NB on domains with correlated attributes.

They conjecture that the performance of NB improves if it uses only those features that Decision Trees used in constructing its decision tree. This method of feature selection would also perform well and learn quickly, that is, it would need fewer training examples to reach high classification accuracy.

They present experimental evidence that this method of feature selection leads to improved performance of the Naïve Bayesian Classifier, especially in the domains where Naïve Bayes performs not as well as Decision Trees. They also analyse the behaviour on ten domains from the UCI repository, 5 of which Decision Trees achieves asymptotically higher accuracy than NB (which seems to imply the presence of correlated features.), and 5 on which NB outperforms Decision Trees.

They then compared our algorithm with the Augmented Bayesian classifier (ABC), and the experimental results justify expectation. They also tested SBC on another sufficiently large synthetic dataset and their algorithm appeared to scale nicely. Their Selective Bayesian Classifier always outperforms NB and performs as well as, or better than both Decision Trees and ABC on almost all the domains.





Dataset	NBC	C4.5	ABC	SBC	SBC vs NBC	SBC vs C4.5	SBC vs ABC
Ecoli	81.99	78.65	**84.35**	83.27	+1.6%	+5.9%	-1.3%
GerCredit	75.35	74.00	76.13	**76.21**	+1.1%	+3.0%	+0.1%
KrVsKp	87.81	**99.12**	94.87	94.69	+7.8%	-4.5%	-0.2%
Monk	96.16	**98.46**	98.02	97.47	+1.4%	-1.0%	-0.6%
Mushroom	93.23	**99.8**	97.98	98.85	+6.0%	-1.0%	+0.9%
Pima	75.03	75.35	78.13	**79.94**	+6.5%	+6.1%	+2.3%
Promoter	87.66	66.67	**88.66**	**88.72**	+1.2%	+33.1%	+0.1%
Soybean	84.02	83.20	**88.32**	**88.27**	+5.1%	+6.1%	-0.1%
Wisconsin	95.78	92.63	96.18	**97.38**	+1.7%	+5.1%	+1.2%
Vote	89.54	95.29	95.54	**96.61**	+7.9%	+1.4%	+1.1%
Mean	86.65	86.32	89.82	**90.14**	+4.0%	+5.4%	+0.4%

Our Approach

In machine learning, *naive Bayes classifiers* are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

1) Converting diagnosis column of training dataset into 0's and 1's

0 indicates Benign

1 indicates Malignant

```
#Training file
#Making diagnosis into 0 or 1
#0<- B
#1<- M

train4$diagnosis2<-0
View(train4)
for(x in c(1:nrow(train4))){
  if(train4[x,"diagnosis"]=="M"){
    train4[x,"diagnosis2"]<-1
  }
}
```

2) Converting diagnosis column of testing dataset into 0's and 1's

0 indicates Benign

1 indicates Malignant

```
#Test file
#Making diagnosis into 0 or 1
#0<- B
```

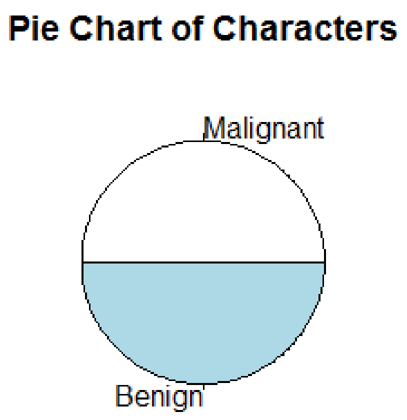
```
#1<- M

test$diagnosis2<-0
for(x in c(1:nrow(test))){
  if(test[x,"diagnosis"]=="M"){
    test[x,"diagnosis2"]<-1
  }
}
```

3) Upsampling

Since the data taken from the original dataset for the training dataset is random, we are not sure how many Benign and Malignant are there. Thus we do upsampling to make sure the proportion is set right to increase the accuracy.

Code

```
#upsampling
library(caret)
up_df <- upSample(x = train4[, -ncol(train4)],y = factor(train4$diagnosis2))
up_df <- as.data.frame(up_df)
b <- 0
e <- 0
for(x in c(1:nrow(up_df))){
  if(up_df[x,"Class"]==1){
    b <- b+1
  }
  if(up_df[x,"Class"]==0){
    e <- e+1
  }
}
k <- c(b, e)
lbls <- c("Malignant", "Benign")
pie(k, labels = lbls, main="Pie Chart of Characters")
```



4) We did Model fitting along with logistic regression. We took the target column as diagnosis. To perform logistic regression we converted them into factors.

```
train4$diagnosis2<-as.factor(train4$diagnosis2)
library(stats)
model <- glm(diagnosis2 ~ .,family=binomial(link='logit'),data=train4)
summary(model)
```

```
Call:
glm(formula = diagnosis2 ~ ., family = binomial(link = "logit"), 
    data = train4)

Deviance Residuals: 
       Min          1Q      Median          3Q         Max  
-3.699e-05  -2.100e-08  -2.100e-08   2.100e-08   3.141e-05  

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -7.498e+02  1.382e+06  -0.001        1
X1                       9.763e-02  3.146e+02   0.000        1
id                       1.947e-08  2.795e-04   0.000        1
radius_mean             -1.001e+02  8.856e+05   0.000        1
texture_mean            -3.268e+00  1.809e+04   0.000        1
perimeter_mean           1.546e+01  1.370e+05   0.000        1
area_mean                2.578e-01  2.071e+03   0.000        1
smoothness_mean          3.859e+02  8.168e+06   0.000        1
compactness_mean        -1.993e+03  5.026e+06   0.000        1
concavity_mean           2.674e+02  3.501e+06   0.000        1
concave.points_mean      2.964e+02  7.330e+06   0.000        1
symmetry_mean            5.970e+01  1.900e+06   0.000        1
fractal_dimension_mean   6.649e+03  1.868e+07   0.000        1
radius_se                3.715e+01  1.562e+06   0.000        1
texture_se              -3.286e+01  1.427e+05   0.000        1
perimeter_se             7.784e+00  1.679e+05   0.000        1
area_se                  1.027e+00  7.729e+03   0.000        1
smoothness_se            3.699e+03  1.566e+07   0.000        1
compactness_se          -1.159e+02  7.792e+06   0.000        1
concavity_se            -5.654e+02  4.531e+06   0.000        1
concave.points_se       -2.900e+03  2.015e+07   0.000        1
symmetry_se             -1.094e+03  1.079e+07   0.000        1
fractal_dimension_se     4.910e+03  2.958e+07   0.000        1
radius_worst             4.410e+01  2.643e+05   0.000        1
texture_worst            7.327e+00  1.743e+04   0.000        1
perimeter_worst         -3.386e+00  2.521e+04   0.000        1
area_worst              -3.481e-01  1.231e+03   0.000        1
smoothness_worst        -1.871e+02  3.832e+06   0.000        1
compactness_worst        1.908e+02  1.466e+06   0.000        1
concavity_worst          1.412e+02  9.256e+05   0.000        1
concave.points_worst     4.350e+02  2.162e+06   0.000        1
symmetry_worst           2.460e+02  7.750e+05   0.000        1
fractal_dimension_worst -1.306e+03  6.210e+06   0.000        1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3.2052e+02  on 249  degrees of freedom
Residual deviance: 1.2286e-08  on 217  degrees of freedom
AIC: 66

Number of Fisher Scoring iterations: 25
```

5) Naïve Bayesian Classification

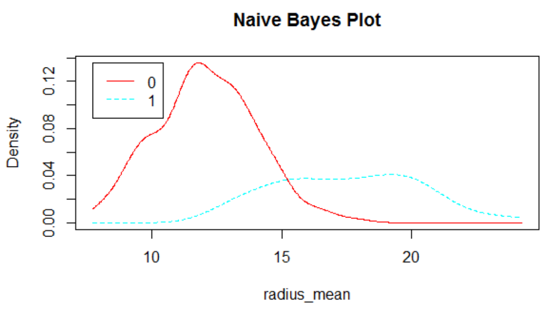
Here we are first using the trained data and coming up with a model. Next, we are predicting the same for the test data. The target column is the diagnosis (Benign or Malignant).

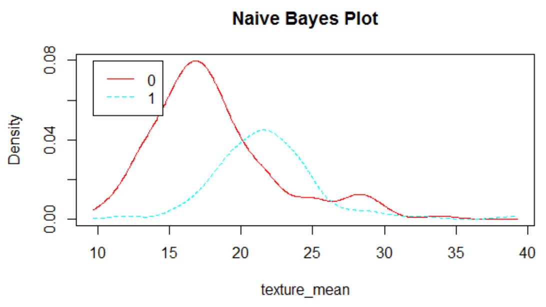
Once we predicting is done, we are plotting Naïve Bayes plots to all columns.

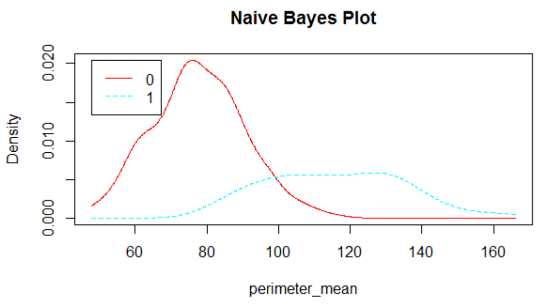
```
#Bayesian Classification

library(e1071)
model1 <- naiveBayes(as.factor(diagnosis2) ~ ., data = train4)
plot(model1)
pred<-predict(model1,test)
class(model1)
summary(model1)
print(model1)
```

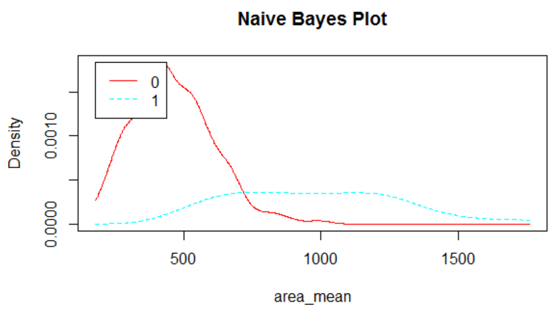
6) Naïve Bayes Graphs

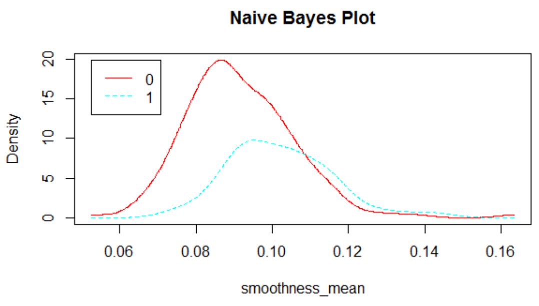
We plot Naïve Bayes Graphs to all the columns. These graphs basically tells us for each attribute what is the density of people whose cancer was Benign or Malignant. We can see many interesting observations here.

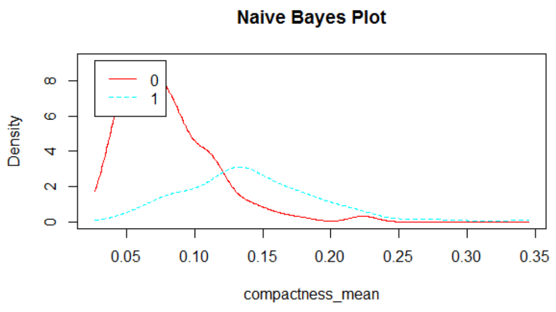


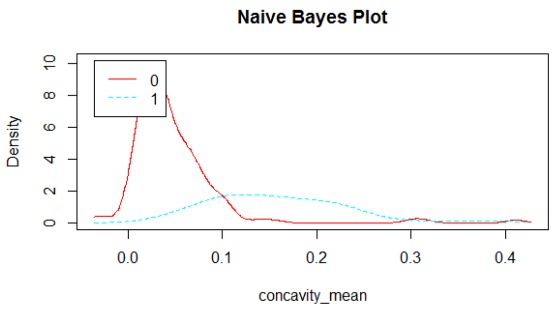


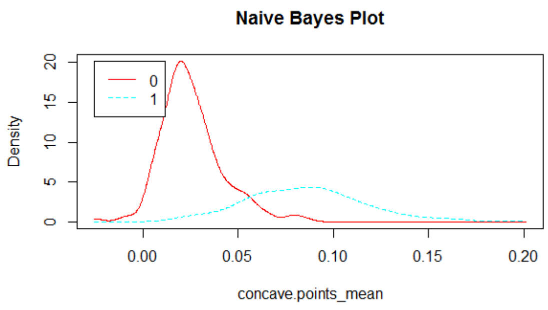


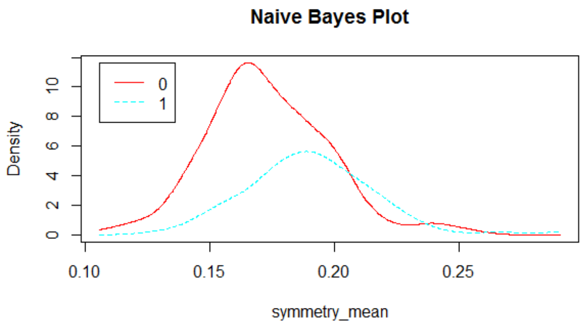


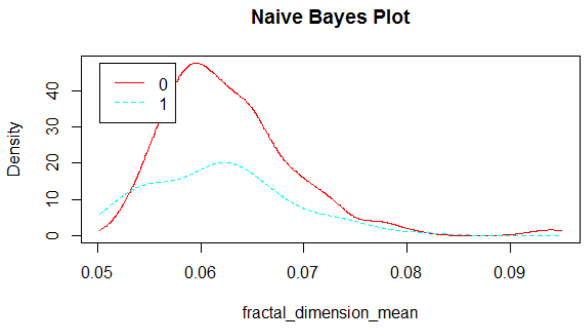


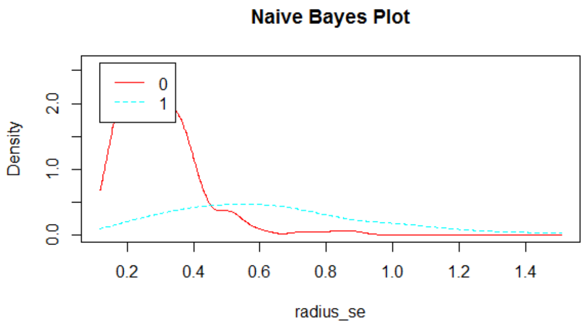


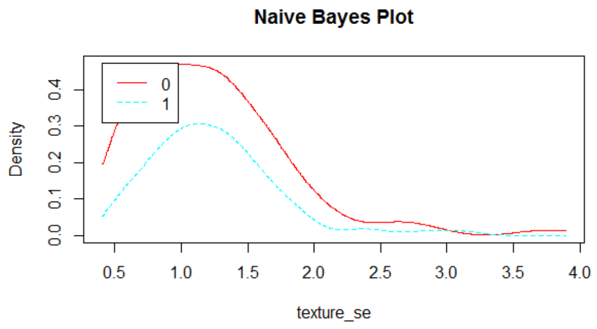

Naive Bayes Plot
Density
0
1
concavity_mean

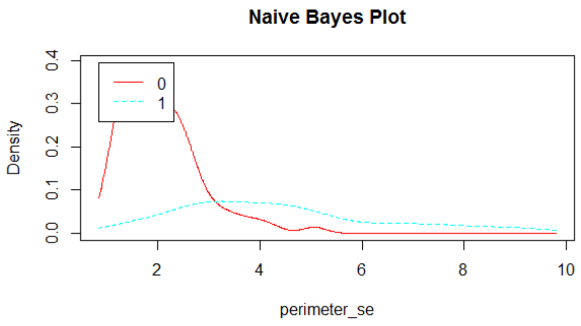

Naive Bayes Plot
Density
0
1
concave.points_mean

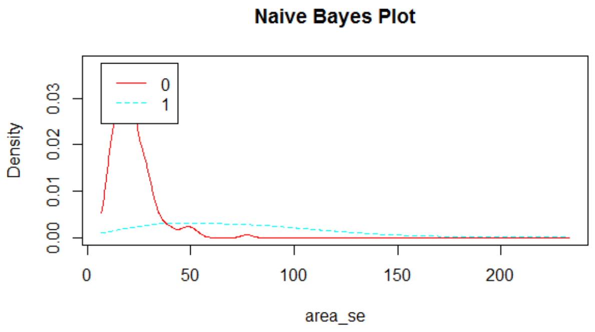

Naive Bayes Plot
Density
0
1
symmetry_mean

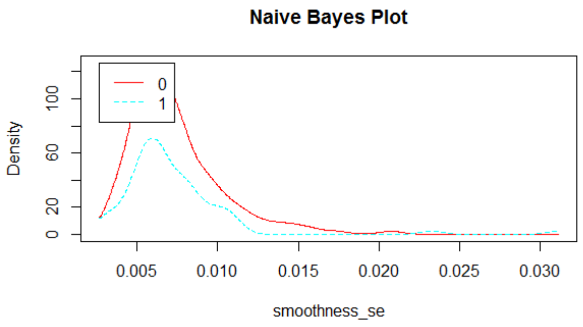

Naive Bayes Plot
Density
0
1
fractal_dimension_mean

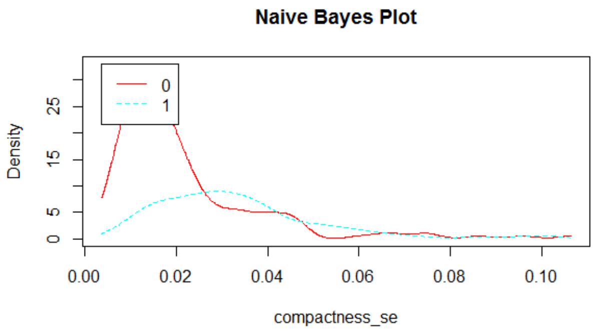

Naive Bayes Plot
Density
0
1
radius_se

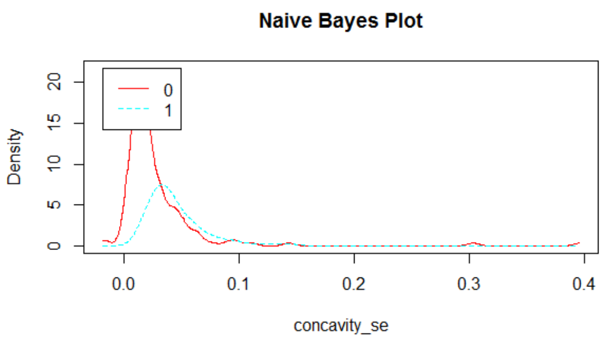

Naive Bayes Plot
Density
0
1
texture_se

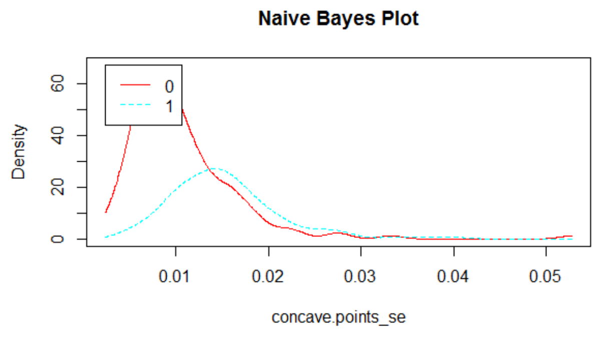

Naive Bayes Plot
Density
0
1
perimeter_se

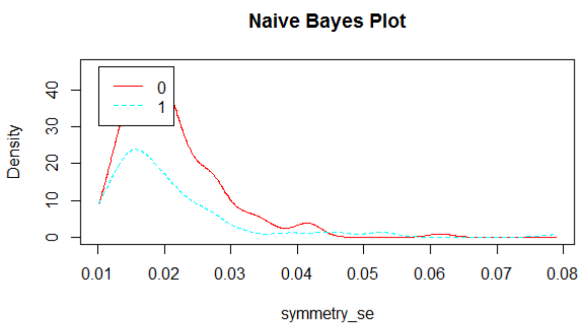

Naive Bayes Plot
Density
0
1
area_se

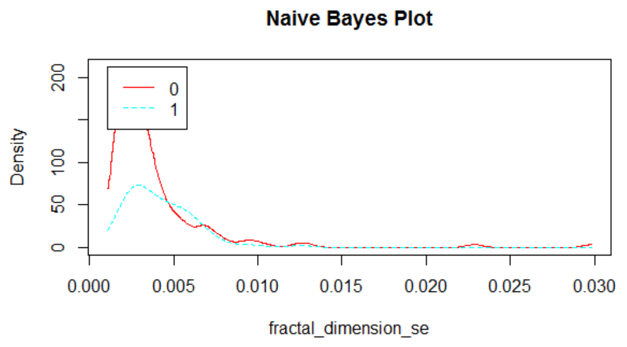

Naive Bayes Plot
Density
0
1
smoothness_se

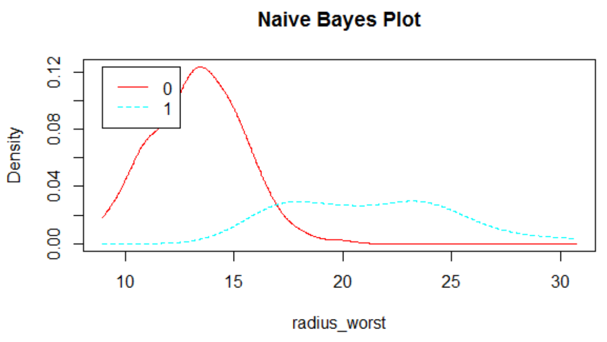

Naive Bayes Plot
Density
0
1
compactness_se

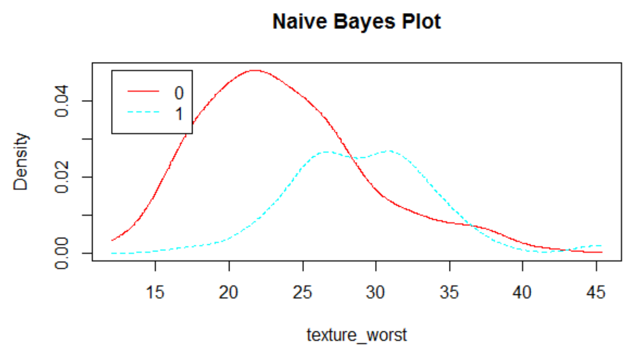

Naive Bayes Plot
0
1
Density
0
5
10
15
20
0.0
0.1
0.2
0.3
0.4
concavity_se

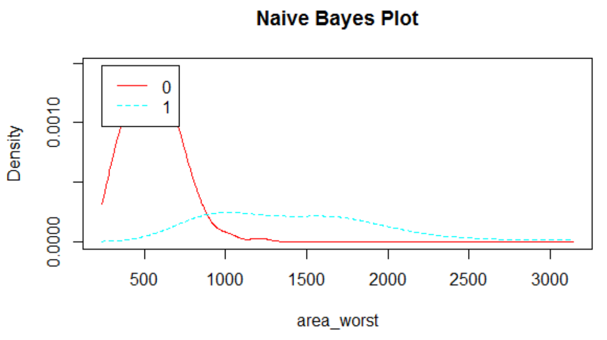

Naive Bayes Plot
0
1
Density
0
20
40
60
0.01
0.02
0.03
0.04
0.05
concave.points_se

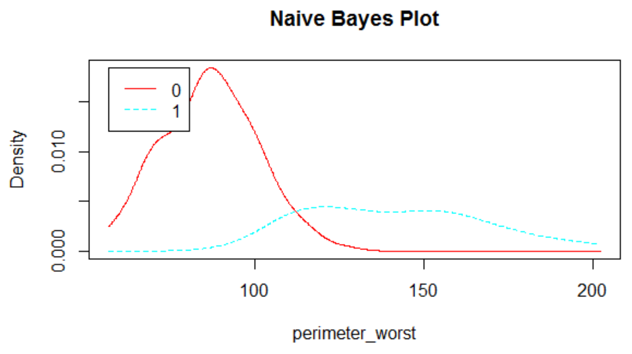

Naive Bayes Plot
0
1
Density
0
10
20
30
40
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
symmetry_se

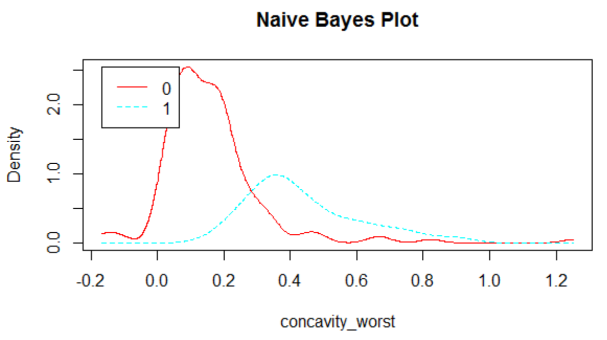

Naive Bayes Plot
0
1
Density
0
50
100
200
0.000
0.005
0.010
0.015
0.020
0.025
0.030
fractal_dimension_se

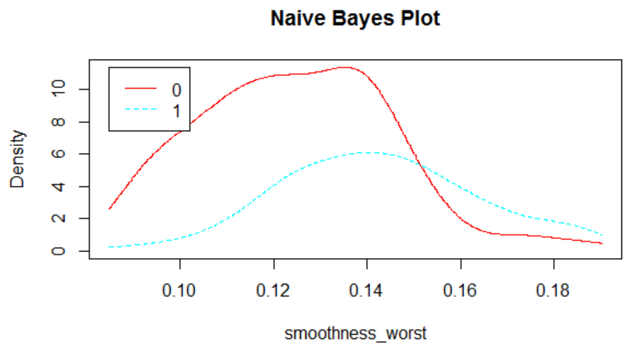

Naive Bayes Plot
0
1
Density
0.00
0.04
0.08
0.12
10
15
20
25
30
radius_worst

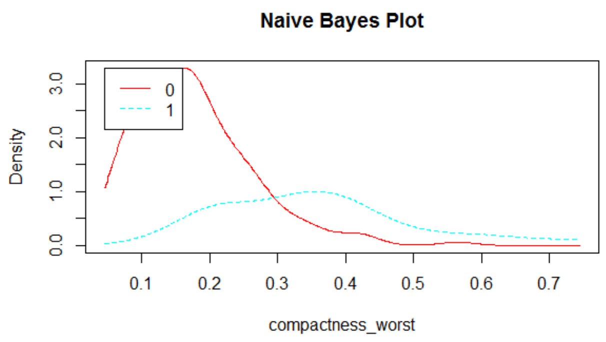

Naive Bayes Plot
0
1
Density
0.00
0.02
0.04
15
20
25
30
35
40
45
texture_worst

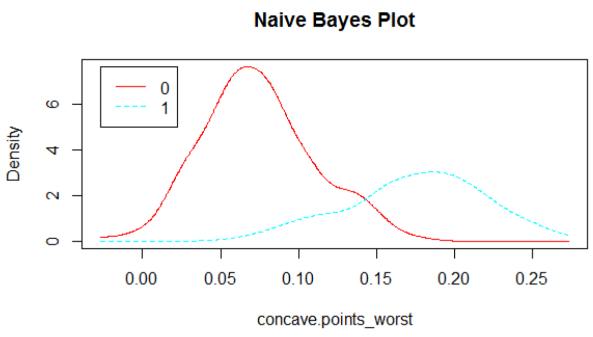

Naive Bayes Plot
0
1
Density
0.0000
0.0010
500
1000
1500
2000
2500
3000
area_worst

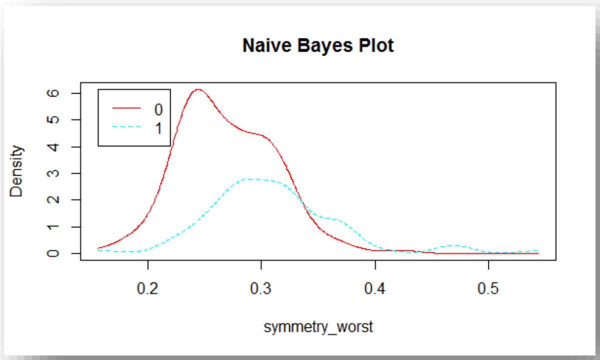

Naive Bayes Plot
0
1
Density
0.000
0.010
100
150
200
perimeter_worst


Naive Bayes Plot
0
1
Density
0.0
1.0
2.0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
concavity_worst


Naive Bayes Plot
0
1
Density
0
2
4
6
8
10
0.10
0.12
0.14
0.16
0.18
smoothness_worst







From the above graphs we cannot come into definite conclusions. But it plays a critical role when analysing certain situation. Many very interesting observations can be seen. But one thing to remember here is "Correlation is not Causation".

7) Confusion Matrix

We can calculated the confusion matrix to know the accuracy of our model.

Code

```
#Confusion Matrix

library(caret)
confusionMatrix(pred,as.factor(test$diagnosis2))

Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0  3  1
         1  0 20

               Accuracy : 0.9583
                 95% CI : (0.7888, 0.9989)
    No Information Rate : 0.875
    P-Value [Acc > NIR] : 0.1797

                  Kappa : 0.8333
 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 1.0000
            Specificity : 0.9524
         Pos Pred Value : 0.7500
         Neg Pred Value : 1.0000
```

```
          Prevalence : 0.1250
       Detection Rate : 0.1250
  Detection Prevalence : 0.1667
     Balanced Accuracy : 0.9762

       'Positive' Class : 0
```

Here we an accuracy of **95.83**

8) Sampling using Kfolds technique

We divide our dataset into 10 folds and each taken as a testing and training dataset and we are coming up with the model and calculating the average accuracy. This is repeated 3 times. This is one of the best methods of sampling.

Code

```
#10 folds 3 repeatation
# load the library
library(caret)
# define training control
train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
# train the model
View(train4)
model3 <- train(diagnosis2~., data=train4, trControl=train_control, method="nb")
# summarize results
print(model3)
```

```
Naive Bayes

250 samples
 32 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 225, 224, 225, 226, 225, 224, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9206282  0.8251482
   TRUE      0.9363718  0.8568205

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust'
 was held constant at a value of 1
Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.
```

Therefore we can conclude that K-folds is giving lesser accuracy than the training set we have chosen earlier. Thus we keep the old Training dataset.

SUPPORT VECTOR MACHINES (SVM)

Literature survey

Paper read: DATA CLASSIFICATION USING SUPPORT VECTOR MACHINE by DURGESH K. SRIVASTAVA and LEKHA BHAMBHU

Following is the brief summary of the paper

Introduction to SVM

The Support Vector Machine (SVM) was first proposed by Vapnik and has since attracted a high degree of interest in the machine learning research community. Several recent studies have reported that the SVM (support vector machines) generally are capable of delivering higher performance in terms of classification accuracy than the other data classification algorithms.

We have experimented with a number of parameters associated with the use of the SVM algorithm that can impact the results. These parameters include choice of kernel functions, the standard deviation of the Gaussian kernel, relative weights associated with slack variables to account for the non-uniform distribution of labeled data, and the number of training examples.

Kernel functions for SVM

Training vectors xi are mapped into a higher (may be infinite) dimensional space by the function Φ. Then SVM finds a linear separating hyperplane with the maximal margin in this higher dimension space .C > 0 is the penalty parameter of the error term. Furthermore, K(xi , xj) ≡ Φ(xi) T Φ(xj) is called the kernel function. There are many kernel functions in SVM, so how to select a good kernel function is also a research issue. However, for general purposes, there are some popular kernel functions.

For general purposes, there are some popular kernel functions.

• Linear kernel: K (xi , xj) = xi T xj.

• Polynomial kernel: K (xi , xj) = (γ xi T xj + r)d , γ > 0

• RBF kernel : K (xi , xj) = exp(-γ ‖xi - xj‖2) , γ > 0

• Sigmoid kernel: K (xi , xj) = tanh(γ xi T xj + r)

Here, γ, r and d are kernel parameters. In these popular kernel functions, RBF is the main kernel function because of following reasons.

1. The RBF kernel nonlinearly maps samples into a higher dimensional space unlike to linear kernel.
2. The RBF kernel has less hyper-parameters than the polynomial kernel.
3. The RBF kernel has less numerical difficulties.

Model Selection

Model selection is also an important issue in SVM. Recently, SVM have shown good performance in data classification. Its success depends on the tuning of several parameters which affect the generalization error. We often call this parameter tuning procedure as the model selection. If you use the linear SVM, you only need to tune the cost parameter C. Unfortunately, linear SVM are often applied to linearly separable problems.

Many problems are non-linearly separable. For example, Satellite data and Shuttle data are not linearly separable. Therefore, we often apply nonlinear kernel to solve classification problems, so we need to select the cost parameter (C) and kernel parameters (γ, d).

Conclusion

This paper speaks on the importance of choosing the necessary kernel function and values of other parameters. Therefore, it can be seen that the choice of kernel function and best value of parameters for a particular kernel is critical for a given amount of data.

SVM CLASSIFICATION TO CLASSIFY TYPES OF BREAST CANCER TO BE MALIGNANT(cancerous) OR BENIGN(non-cancerous)

Data set used- Wisconsin Breast Cancer data set.

There were a large number of samples present in the original data set of Breast Cancer with 31 attributes. A few data points were missing, so we had to drop those observation from the table.

Phase 1

Training the SVM.

Data is cleaned and 250 samples are picked up for training the Support Vector Machine Model.

Phase 2

Testing the SVM.

25 samples are picked for the testing phase. Necessary plots are drawn and the efficiency is analyzed based on the actual value obtained and the expected value.

Types of SVM

SVM classification is broadly divided into 2 categories – C type and nu type.

SVM use hyperplanes to perform classification. While performing classifications using SVM there are 2 types of SVM

- C SVM
- Nu SVM

C and nu are regularization parameters which help implement a penalty on the misclassifications that are performed while separating the classes. Thus helps in improving the accuracy of the output.

C ranges from 0 to infinity and can be a bit hard to estimate and use. A modification to this was the introduction of nu which operates between 0-1 and represents the lower and upper bound on the number of examples that are support vectors and that lie on the wrong side of the hyperplane.

Both have a comparative similar classification power, but the nu- SVM has been harder to optimize.

Approach taken

We do both types of classification initially and calculate which of the SVM's have a better accuracy. Then we explore by using various kernals, and judging the model in terms of how efficient it is, comparing the predicted values with the actual values obtained. For each kernel, we find the optimum value for the cost parameter by using the tuned function.

The necessary excel files are imported with the working directories appropriately set.

SVM C-type classification linear

We have to determine the appropriate value for the cost parameter. For this we use the “tuned” function. The problem with this step is that, the tuned function wants the dependent variable to have numeric binary values. So we add another column, having 0’s for benign and 1’s for malignant. We delete it after this step as we do not need it. After this step we go back to the default way of representing Malignant and Benign cancer as “M” and “B” respectively.

```
train1$diagnosis1 <- ifelse(train1$diagnosis=="M",1,0)
train1$diagnosis <- NULL


tuned <- tune(svm, diagnosis1~., data=train1, kernel="linear",
          ranges=list(cost=c(0.001, 0.01, 0.1, 1,10,100)))
summary(tuned)

train1$diagnosis <- ifelse(train1$diagnosis1==1,"M","B")
train1$diagnosis1 <- NULL
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.05980001

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.07738974 0.01724585
2 1e-02 0.06359907 0.01649902
3 1e-01 0.05980001 0.01661449
4 1e+00 0.06014053 0.01658099
5 1e+01 0.06124749 0.01823747
6 1e+02 0.06324680 0.01865530
```

Now we train the SVM based on the observations in the training data set by setting the cost parameter as 0.1 as obtained in the earlier step.

```
svmfit<-svm(diagnosis~., data=train1, kernel="linear", cost=.1, scale=FALSE,
         type="C-classification")
print(svmfit)
```

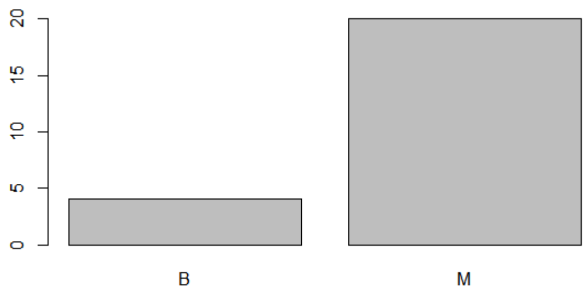
The following output is obtained.

```
Call:

svm(formula = diagnosis ~ ., data = train1, kernel = "linear", cost = 0.1,
    type = "C-classification", scale = FALSE)


Parameters:
SVM-Type:  C-classification
```

```
SVM-Kernel:  linear
cost:  0.1
gamma:  0.03333333

Number of Support Vectors:  19
```

Now we have the required SVM model ready. We need to test it on some samples to compare the obtained output with the expected output to calculate the accuracy.

```
p<- predict(svmfit, test1, type="class")
plot(p)
```



Now we calculate the accuracy using a confusion matrix.

Accuracy using caret package and confusion matrix

```
library(caret)
table_pred <- table(p, test1$diagnosis)
confusionMatrix(table_pred)

Confusion Matrix and Statistics


p   B  M
 B  3  1
 M  0 20

   Accuracy : 0.9583
   95% CI : (0.7888, 0.9989)
   No Information Rate : 0.875
   P-Value [Acc > NIR] : 0.1797

   Kappa : 0.8333
   Mcnemar's Test P-Value : 1.0000

   Sensitivity : 1.0000
       Specificity : 0.9524
```

```
         Pos Pred Value : 0.7500
         Neg Pred Value : 1.0000
         Prevalence : 0.1250
         Detection Rate : 0.1250
        Detection Prevalence : 0.1667
        Balanced Accuracy : 0.9762

       'Positive' Class : B
```

We can see that the accuracy is 95.83%

SVM C type classification with kernel = polynomial

```
train1$diagnosis1 <- ifelse(train1$diagnosis=="M",1,0)
train1$diagnosis <- NULL

tuned <- tune(svm, diagnosis1~., data=train1, kernel="polynomial",
           ranges=list(cost=c(0.001, 0.01, 0.1, 1,10,100)))
summary(tuned)

train1$diagnosis <- ifelse(train1$diagnosis1==1,"M","B")
train1$diagnosis1 <- NULL


Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.2152279

- Detailed performance results:
   cost     error dispersion
1 1e-03 0.2754988 0.07017171
2 1e-02 0.2152279 0.05123325
3 1e-01 0.2251165 0.12789878
4 1e+00 0.3623168 0.44306833
5 1e+01 2.8293963 6.24619175
6 1e+02 5.7214136 8.13964436
```

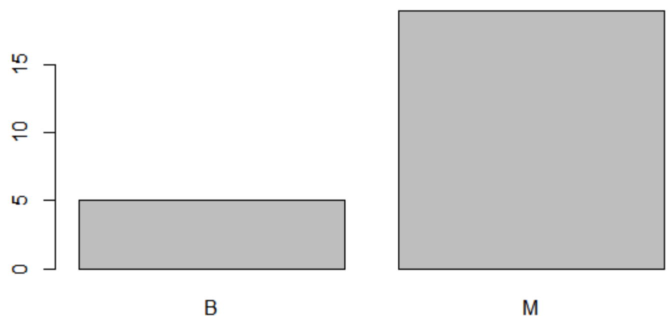
So the best cost parameter is this is 0.01.

```
svmfit<-svm(diagnosis~., data=train1, kernel="polynomial", cost=.01, scale=FALSE,
          type="C-classification")
print(svmfit)


Call:
svm(formula = diagnosis ~ ., data = train1, kernel = "polynomial", cost = 0.01,
   type = "C-classification", scale = FALSE)
```

```
Parameters:
 SVM-Type:  C-classification
 SVM-Kernel:  polynomial
      cost:  0.01
    degree:  3
     gamma:  0.03333333
    coef.0:  0

Number of Support Vectors:  17
```

The required SVM model is ready and then is used for testing.

```
p<- predict(svmfit, test1, type="class")
plot(p)
```



```
Confusion Matrix and Statistics


p     B  M
  B   3  2
  M   0 19

               Accuracy : 0.9167
                 95% CI : (0.73, 0.9897)
    No Information Rate : 0.875
    P-Value [Acc > NIR] : 0.4082

                  Kappa : 0.7037
 Mcnemar's Test P-Value : 0.4795

            Sensitivity : 1.0000
            Specificity : 0.9048
         Pos Pred Value : 0.6000
         Neg Pred Value : 1.0000
             Prevalence : 0.1250
         Detection Rate : 0.1250
   Detection Prevalence : 0.2083
      Balanced Accuracy : 0.9524

       'Positive' Class : B
```

So accuracy obtained is 91.67%

SVM C type classification kernel=sigmoid

```
train1$diagnosis1 <- ifelse(train1$diagnosis=="M",1,0)

train1$diagnosis <- NULL

tuned <- tune(svm, diagnosis1~., data=train1, kernel="sigmoid",
        ranges=list(cost=c(0.001, 0.01, 0.1, 1,10,100)))
summary(tuned)

train1$diagnosis <- ifelse(train1$diagnosis1==1,"M","B")
train1$diagnosis1 <- NULL
```

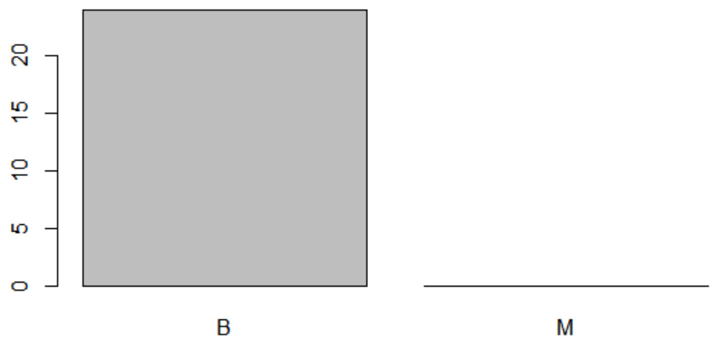
```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.08139677

- Detailed performance results:
   cost        error   dispersion
1 1e-03 2.898305e-01 8.448385e-02
2 1e-02 1.481360e-01 4.826611e-02
3 1e-01 8.139677e-02 2.652643e-02
4 1e+00 1.924732e+00 1.160940e+00
5 1e+01 1.509745e+02 7.656473e+01
6 1e+02 1.831228e+04 1.096614e+04
```

So the optimum cost parameter is 0.1

```
svmfit<-svm(diagnosis~., data=train1, kernel="sigmoid", cost=.1, scale=FALSE,
       type="C-classification")
print(svmfit)
```

```
Call:
svm(formula = diagnosis ~ ., data = train1, kernel = "sigmoid", cost = 0.1, type = "C-
classification",
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  sigmoid
       cost:  0.1
      gamma:  0.03333333
     coef.0:  0

Number of Support Vectors:  170
```

```
p<- predict(svmfit, test1, type="class")
plot(p)
table_pred <- table(p, test1$diagnosis)


confusionMatrix(table_pred)
```



```
Confusion Matrix and Statistics


p    B  M
  B  3 21
  M  0  0

               Accuracy : 0.125
                 95% CI : (0.0266, 0.3236)
    No Information Rate : 0.875
    P-Value [Acc > NIR] : 1

                  Kappa : 0
 Mcnemar's Test P-Value : 1.275e-05

            Sensitivity : 1.000
            Specificity : 0.000
         Pos Pred Value : 0.125
         Neg Pred Value :   NaN
             Prevalence : 0.125
         Detection Rate : 0.125
   Detection Prevalence : 1.000
      Balanced Accuracy : 0.500

       'Positive' Class : B
```

So accuracy obtained is 12.5%

<u>SVM Nu-type classification kernel=linear</u>

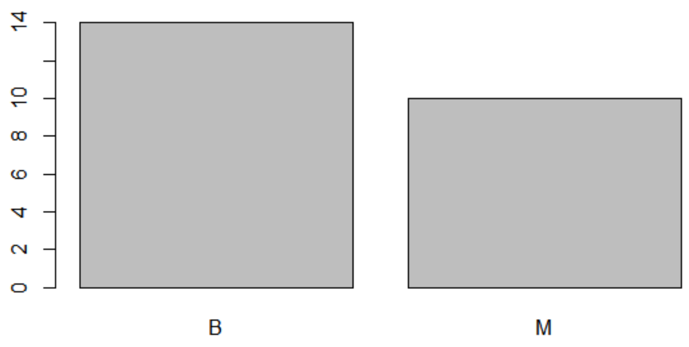
For linear kernel, as we saw earlier, optimum cost parameter= 0.1

```
svmfit<-svm(diagnosis~., data=train1, kernel="linear", cost=.1, scale=FALSE,
      type="nu-classification")
print(svmfit)
```

```
Call:
svm(formula = diagnosis ~ ., data = train1, kernel = "linear", cost = 0.1, type = "nu-
classification",
    scale = FALSE)


Parameters:
   SVM-Type:  nu-classification
```

```
 SVM-Kernel:  linear 
      gamma:  0.03333333 
         nu:  0.5 

Number of Support Vectors:  126
```

p<- predict(svmfit, test1, type="class")
plot(p)



library(caret)
table_pred <- table(p, test1$diagnosis)
confusionMatrix(table_pred)

```
Confusion Matrix and Statistics


p    B  M
  B  3 11
  M  0 10
                                          
               Accuracy : 0.5417          
                 95% CI : (0.3282, 0.7445)
    No Information Rate : 0.875           
    P-Value [Acc > NIR] : 0.999991        
                                          
                  Kappa : 0.1852          
 Mcnemar's Test P-Value : 0.002569        
                                          
            Sensitivity : 1.0000          
            Specificity : 0.4762          
         Pos Pred Value : 0.2143          
         Neg Pred Value : 1.0000          
             Prevalence : 0.1250          
         Detection Rate : 0.1250          
   Detection Prevalence : 0.5833          
      Balanced Accuracy : 0.7381          
                                          
       'Positive' Class : B
```

Accuracy is 54.17%

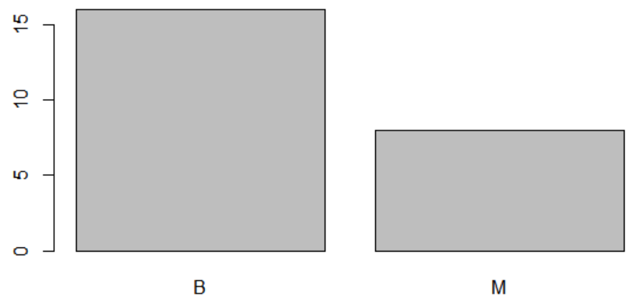
SVM Nu-type classification kernel=”polynomial”

For polynomial type, we saw earlier that the optimum cost parameter is 0.01.

```
svmfit<-svm(diagnosis~., data=train1, kernel="polynomial", cost=.01, scale=FALSE,
        type="nu-classification")
print(svmfit)
```

```
Call:
svm(formula = diagnosis ~ ., data = train1, kernel = "polynomial", cost = 0.01,
    type = "nu-classification", scale = FALSE)


Parameters:
   SVM-Type:  nu-classification
 SVM-Kernel:  polynomial
     degree:  3
      gamma:  0.03333333
     coef.0:  0
         nu:  0.5

Number of Support Vectors:  126
```

```
p<- predict(svmfit, test1, type="class")
plot(p)
```



```
library(caret)
table_pred <- table(p, test1$diagnosis)

confusionMatrix(table_pred)
```

```
Confusion Matrix and Statistics


p    B  M
  B  3 13
  M  0  8

               Accuracy : 0.4583
                 95% CI : (0.2555, 0.6718)
    No Information Rate : 0.875
    P-Value [Acc > NIR] : 0.9999999

                  Kappa : 0.1333
```

```
 Mcnemar's Test P-Value : 0.0008741

            Sensitivity : 1.0000
            Specificity : 0.3810
         Pos Pred Value : 0.1875
         Neg Pred Value : 1.0000
             Prevalence : 0.1250
         Detection Rate : 0.1250
   Detection Prevalence : 0.6667
      Balanced Accuracy : 0.6905

       'Positive' Class : B
```

Accuracy obtained is 45.83%

SVM nu-type classification kernel=”sigmoid”

As seen before, the optimum value of the cost parameter is 0.1.

It seems that for the given dataset some values for *nu* type classification are infeasible and instead of skipping them it decides to throw away the whole work and crash instead.

RESULTS:

SVM CLASSIFICATION

C TYPE CLASSIFICATION

KERNEL	ACCURACY	COST
LINEAR	**95.83%**	0.1
POLYNOMIAL	**91.67%**	0.01
SIGMOID	**12.5%**	0.1

NU TYPE CLASSIFICATION

KERNEL	ACCURACY	COST
LINEAR	**54.17%**	0.1
POLYNOMIAL	**45.83%**	0.01
SIGMOID	**(N.A)**	0.1

Decision Trees

Literature survey

Paper read: Erin J. Bredensteiner and Kristin P. Bennett. Feature Minimization within Decision Trees. National Science Foundation. 1996.

Summary

Decision trees for classification can be constructed using mathematical programming. Within decision tree algorithms, the feature minimization problem is to construct accurate decisions using as few features or attributes within each decision as possible. Feature minimization is an important aspect of data mining since it helps identify what attributes are important and helps produce accurate and interpretable decision trees.
In feature minimization with bounded accuracy, we minimize the number of features using a given misclassification error tolerance.
The resulting minimization algorithm produces more compact, accurate, and interpretable trees.
In a decision tree, several linear discriminates are applied recursively to form a nonlinear separation of the space Rn into disjoint regions, each corresponding to set A or set B. The goal is to obtain a decision tree, with one or more decisions, which generalizes well, i.e., correctly classifies future points.
Feature minimization is an important aspect of multivariate decision tree construction. The goal of feature minimization is to construct good decisions using as few features as possible. By minimizing the number of features used at each decision, understandability of the resulting tree is increased and the number of data evaluations is decreased. Feature minimization is not necessary in univariate decision tree algorithms in which each decision in the tree is based on a single feature or attribute.
A tree with multivariate decisions can represent more complex relationships using
fewer decisions than univariate trees. However multivariate decisions with too many attributes can be difficult to interpret.
Our goal is to make both a small number of decisions and to utilize only necessary attributes in each decision.
There is a trade off between the complexity of each decision and the number of decisions required in the tree. Multivariate decision trees typically have many fewer decisions than univariate decision trees constructed using one attribute per decision. Univariate decision trees have the advantages that single attribute decisions help avoid over-parameterization and the resulting trees are more readily interpretable provided the number of decisions is not excessive.

Implementing Feature Minimization in our Project

Since the Wisconsin Data set has 31 parameters which may/may not affect the final classification , we have to implement feature minimization if we are going to use Decision Trees , as the fewer parameters we give into the model creation function , the lesser shall be levels in tree , and hence we might get better results.

Therefore we implement feature minimization by using variance of the parameters and correlation values of parameters in between themselves. Those parameters which have variance less than 0.015 are not considered in model estimation , as their variance is too less to affect the final model .

Also , variables which have a correlation of more than 0.9 with other variables , implies that they have a strong similar behavior with each other. Hence we can drop the other variables and take only the first variable. This is our method of feature minimization , and this results in only 14 critical , dissimilar parameters being chosen .

DECISION TREE CLASSIFICATION TO CLASSIFY TYPES OF BREAST CANCER, AS MALIGNANT (cancerous) OR BENIGN (non cancerous)

- The cleaned data set is taken , and the training data is created randomly from the original data set . It has 250 observations , with which we are going to work with. Firstly , we perform feature minimization to the data , so as to get the parameters which affect the training data the most . We shall be using only these variables for our model construction and prediction.

- **Step 1 -** We have the reduced or minimized number of features , we can start to train our decision tree on these column variables , and see whether the accuracy of the decision tree prediction is good enough for it to be considered as a suitable classifier in our data set.

  (a) We first start off by specifying the packages that we may need for making a decision tree model , and also for plotting it in a suitable way . Hence the list of models required are as follows :

  Rpart : used for for "Recursive Partitioning and Regression Trees" and uses the CART decision tree algorithm.

  RColorBrewer : The R Analytic Tool To Learn Easily (Rattle) provides a Gnome (RGtk2) based interface to R functionality for data science.

  Rattle : The R Analytic Tool To Learn Easily (Rattle) provides a Gnome (RGtk2) based interface to R functionality for data science

  Caret : The caret package (short for _C_lassification _A_nd _RE_gression _T_raining) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for data splitting, pre-processing, feature selection , model tuning using resampling , variable importance estimation
  as well as other functionality.

  Rpart.plot : Plot an rpart model, automatically tailoring the plot for the model's response type.

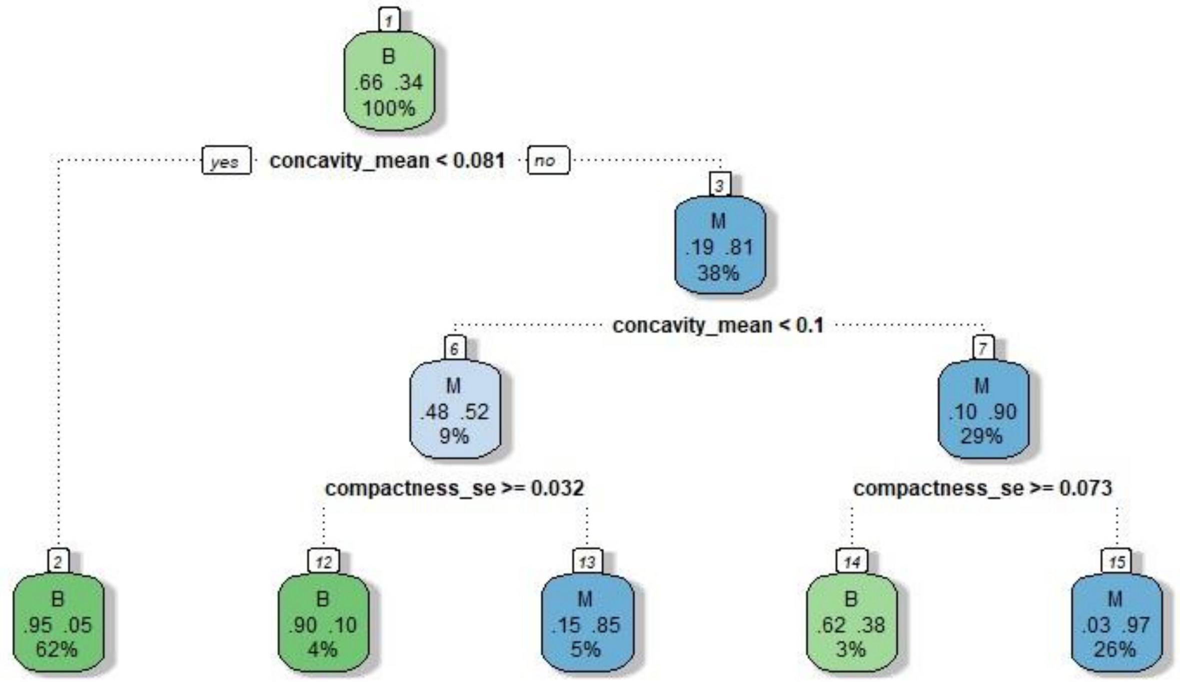
The parameters taken here have a correlation limit of 0.80 with each other and their indiviual variation should be greater than 0.015 .

------------------------Code------------------------------

```
fit1 <- rpart(
            diagnosis ~   compactness_mean + concavity_mean +
                          symmetry_mean + fractal_dimension_mean  +
                          texture_se + compactness_se + concave.points_se +
                          symmetry_se + smoothness_worst+ compactness_worst +
                          concavity_worst + symmetry_worst + fractal_dimension_worst,
                data=Cancer_data,
                method="class"
)
fancyRpartPlot(fit1)
```

(b) Explanation: The model for the decision tree is created using the rpart() function , which takes in the 'formula' for the model , which is nothing but the way the variables depend on each other . this , along with the data set name , and method being class , is passed to the function . The function generates the best Decision tree possible for the given data and shows it to us .

Plot : <u>Decision Tree Model built by the rpart package as shown.</u>



(c) The way to read this decision tree is simple . If the value concavity_mean for any observation is less than 0.081 , we can directly classify them as benign . Hence knowing this , we can classify the data into levels B or M in a faster and more efficient way. Hence it can be seen that only two variables decide the outcome of the classification. To show whether or not my observation is correct , we check the accuracy of the model's predictions against some test data.

```
prediction <- predict(fit1, test[,can_keep], type = "class")

table_pred <- table(Predict = prediction,Actual = test$diagnosis)

confusionMatrix(table_pred)
```

(d) The predict function takes the observations in the test data set , runs it through the model , and keeps the predictions in the variable named the same way. We then check the accuracy of the prediction using a ConfusionMatrix function. Output of the code is as shown below :

```
Confusion Matrix and Statistics

           Actual
Predicted   B   M
        B   3   5
        M   0  16

               Accuracy : 0.7917
                 95% CI : (0.5785, 0.9287)
    No Information Rate : 0.875
    P-Value [Acc > NIR] : 0.92971
```

(e) The accuracy of the model is ~80% , which is considered to be borderline in statistical terms . Ideally we would like to achieve an accuracy of > 90% . Let's see whether our next step helps us achieve a better accuracy or not.

Step 2 – When we use this method of having a fixed training and testing data , we are taking a lot of assumptions and performing the analysis , i.e. we assume that the training data is ideal and represents how most of the data will be in the population . Also the 'accuracy' of the model is tested on the test data , and if the test data has many outliers or just has variables of a different trend than the training data , then as the model couldn't capture this , it shall mostly give us wrong results for this particular test data . Hence to remove this ambiguity with respect to training and test data , we use K Fold Validation , where the entire training data is split into k folds , and each of the folds is used once as the test data set , so we won't be leaving out any underlying trends or missing out on any important inferences .

(a) Using K-Fold validation : It works in the same way as described above . , one fold is used as a test data and the rest k-1 folds are used for training.
Hence the same model is repeatedly tested . Keep in mind , K Fold validation is not for creation of models , just for checking it's validity with the entire dataset as a training set . The entire procedure is given below.

-------------------Using K-Fold validation-----------------------

```
# "k- fold cross validation". steps:
#
# Randomly split your entire dataset into k"folds".
# For each k folds in your dataset, build your model on k – 1 folds of
# the data set. Then, test the model to check the effectiveness for kth fold.
# Record the error you see on each of the predictions.
# Repeat this until each of the k folds has served as the test set.
```

# The average of your k recorded errors is called the cross-validation error

# and will serve as your performance metric for the model.

#Using caret to perform K-fold on the rpart tree

(b) Here we are setting seeds for whatever result we get , since the model is trained with random data sets , and if we run the same model again , we may lose our earlier results . Hence if we use seeds , the datasets are still selected at random but the same data sets are created (just to get the same results the next time we execute the code).

(c) The trainControl() function is responsible for determining the number of folds we are going to do for cross validation method (cv) , and some other settings related with the output.

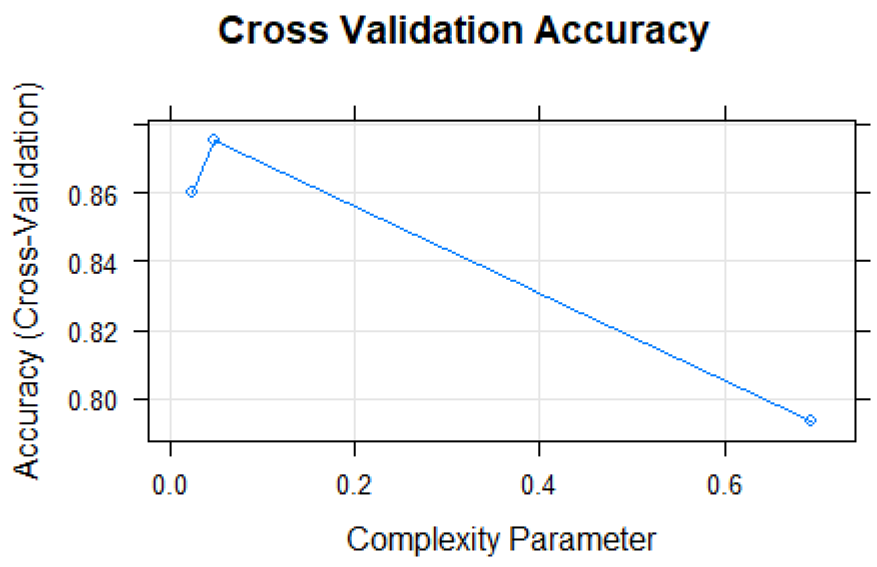
(d) The train() is to train the model that is created when use rpart() function with the dataset(same as decision tree). We also have to set the way in which we way we are going to control the training , by setting the trControl parameter in train function.

```
set.seed(11)


train_control<- trainControl(method="cv", number=10,verboseIter = TRUE ,savePredictions =
TRUE)

model1 <- train(
                diagnosis  ~  compactness_mean + concavity_mean +
                              symmetry_mean + fractal_dimension_mean  +
                              texture_se + compactness_se + concave.points_se +
                              symmetry_se + smoothness_worst+ compactness_worst +
                              concavity_worst + symmetry_worst + fractal_dimension_worst,

                data=Cancer_data,

                trControl=train_control,

                method="rpart"

               )


model1$results
```

No.	cp	Accuracy	Kappa	AccuracySD	Kappa SD
1	0.023	0.8598	0.6833	0.0342	0.0758
2	0.047	0.8753	0.7267	0.0454	0.0975
3	0.694	0.7933	0.4697	0.0922	0.3272

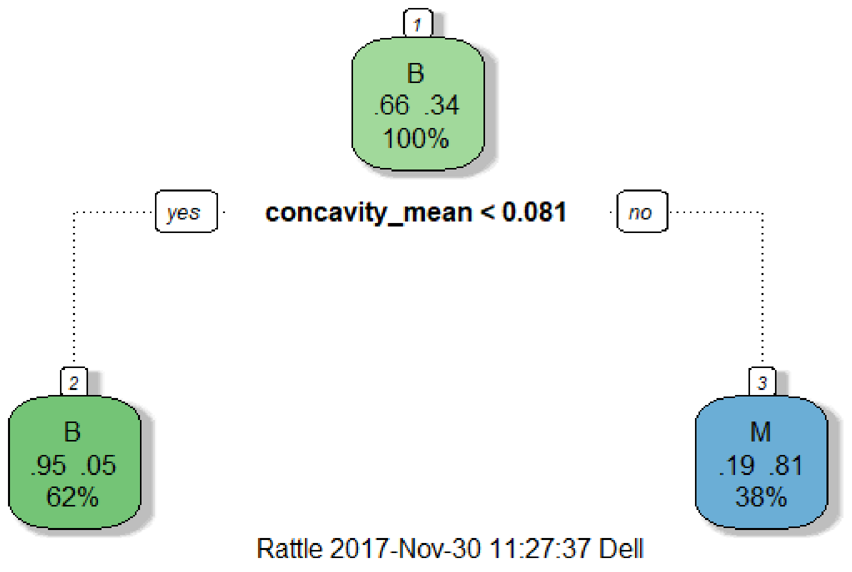
(e) The train() function gives us the output of the accuracy of the model with random samples , along with the Standard dev of the accuracy . Here we get an accuracy of ~85 % , which is better than the previous accuracy that we got (79%). However , this is a better trained model of our decision tree , and if we want to compare the two decision trees , we have to compare using the same test data that we used for the earlier model .
Therefore we now use the predict function , and compare the two models with the same test data set.

```
print(model1)

# CART

# 250 samples

#

# 13 predictor

# 2 classes: 'B', 'M'

#

# No pre-processing

# Resampling: Cross-Validated (10 fold)

# Summary of sample sizes: 226, 226, 225, 224, 226, 225, ...

 # Accuracy was used to select the optimal model using  the largest value.

# The final value used for the model was cp = 0.04705882.


#using best model of CV to predict


plot(model1 , main = "Cross Validation Accuracy")
```



The final model that we get from the from the K folds validation is shown below :

fancyRpartPlot(model1$finalModel)



predictions <- predict(model1 , test)

confusionMatrix(table(Predict = predictions , Actual = test$diagnosis))

The output of the Confusion matrix is :

```
Confusion Matrix and Statistics

            Actual
     Predict    B   M
           B   3    2
           M   0    19

          Accuracy : 0.9167
            95% CI : (0.73, 0.9897)
 No Information Rate : 0.875
 P-Value [Acc > NIR] : 0.4082
```

Therefore we see that the model that has been trained K – Fold Validation is giving a higher accuracy (92%) , than the model which has been trained with our original training data (79%) . The increase in accuracy is almost 17% , hence we shall use the K fold model for future cl assification.