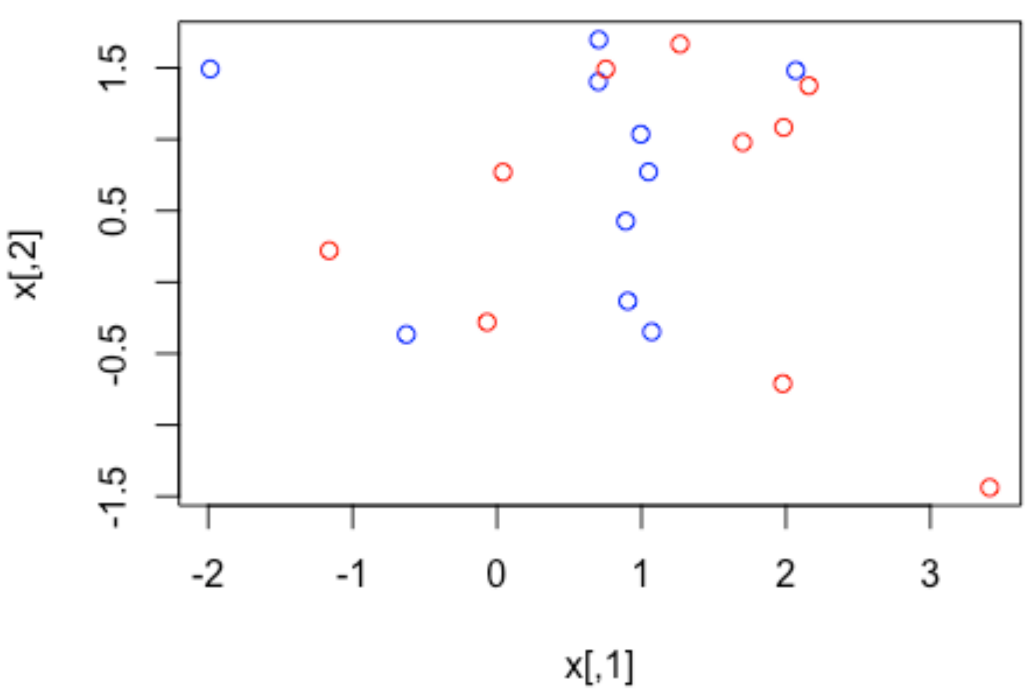
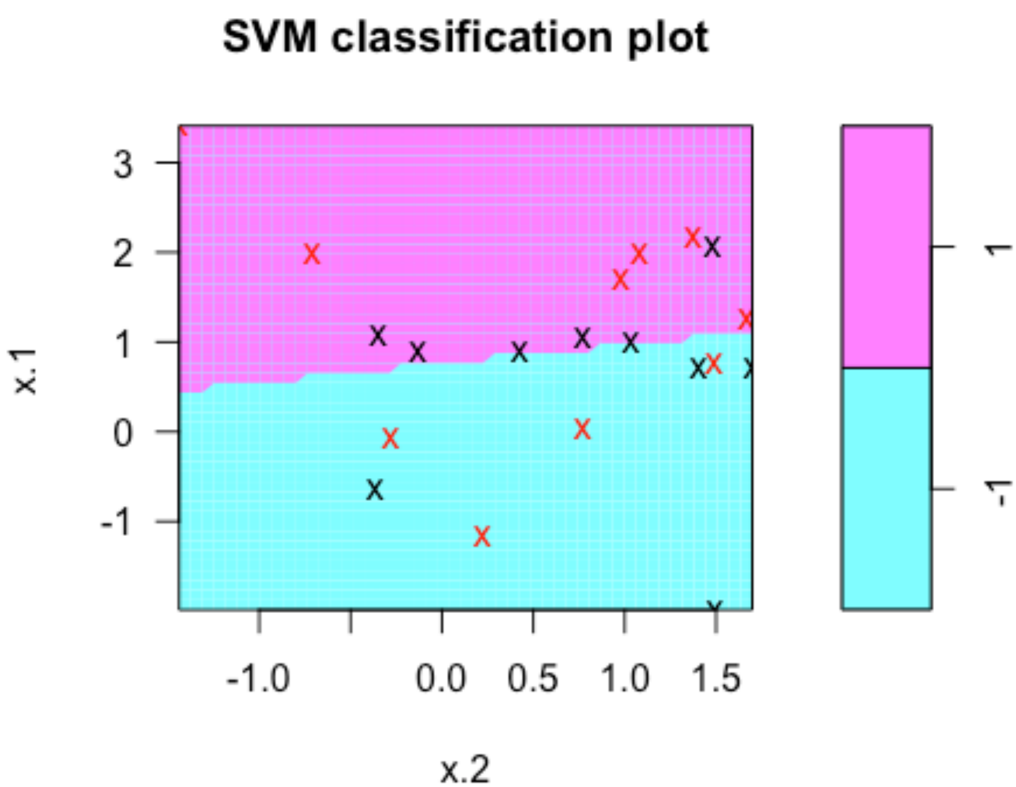
Final Project Analytical Business Intelligence

```r
#Final Project: Analytical Business Intelligence
# Submitted By: Jayendra Bhardwaj
# RUID: 181006372

#Support Vector Classifier

set.seed(6372)
#generating Random Numbers and using command Matrix to generate two sets of
data
x <- matrix(rnorm(20*2), ncol=2)
y <- c(rep(-1,10), rep(1,10))
x[y==1,]=x[y==1,] + 1
plot(x, col=(3-y))
```




```r
#SVM Classification Plot
library(e1071)
dat <- data.frame(x=x, y=as.factor(y))
```

```r
svm.fit<- svm(y ~., data=dat, kernel = 'linear', cost=0.1, scale=FALSE)
plot(svm.fit, dat)
```

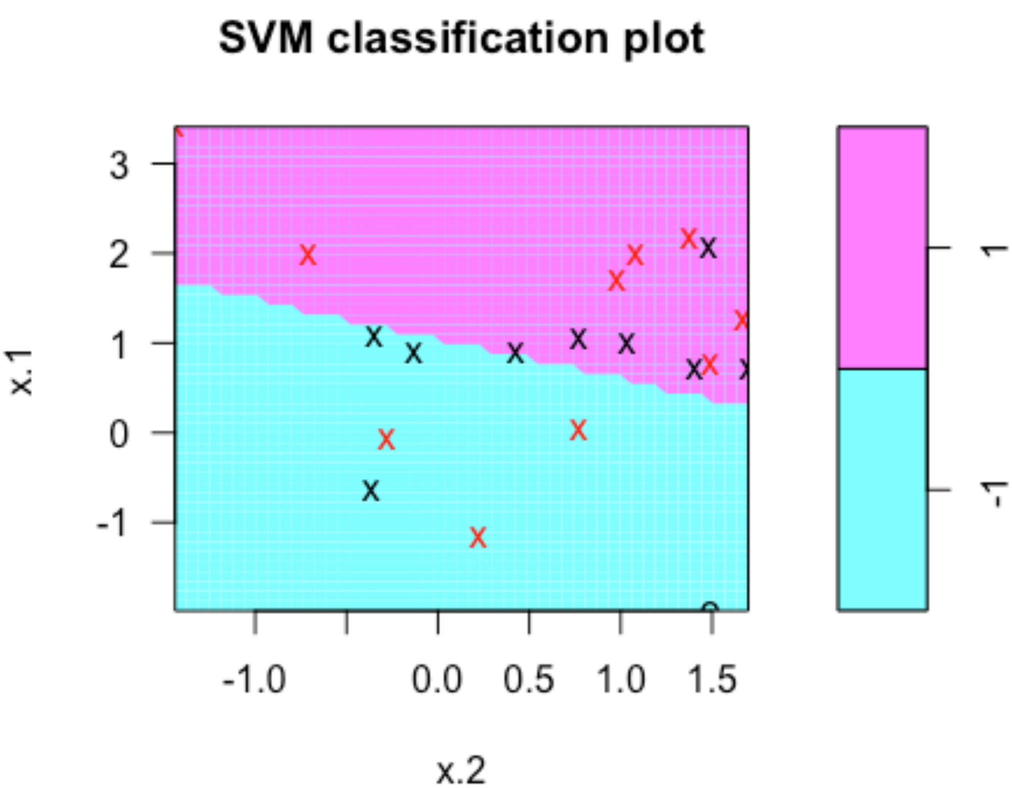



```r
summary(svm.fit) #Identites of Support Vector

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 0.1,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##       gamma:  0.5
##
## Number of Support Vectors:  20
##
##  ( 10 10 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  -1 1

svm.fit$index

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

#The summary lets us know there were 18 support vectors which are {1  2  3  4
5  6  8  9 10 11 12 13 14 15 16 17 18 19},
#9 in the first class and 9 in the second

# Increase number of cost parameter to 10
dat <- data.frame(x=x, y=as.factor(y))
svm.fit1 <- svm(y ~., data=dat, kernel='linear', cost=10, scale=FALSE)
plot(svm.fit1, dat)

summary(svm.fit1)

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  19
##
##  ( 9 10 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1

svm.fit1$index

##  [1]  1  2  3  4  5  6  7  8 10 11 12 13 14 15 16 17 18 19 20

#The summary lets us know there were 18 support vectors which are {1  2  3  4
9 11 16 17 19},
#5 in the first class and 4 in the second

#Comparing SVM with Linear Kernel
set.seed(6372)
```

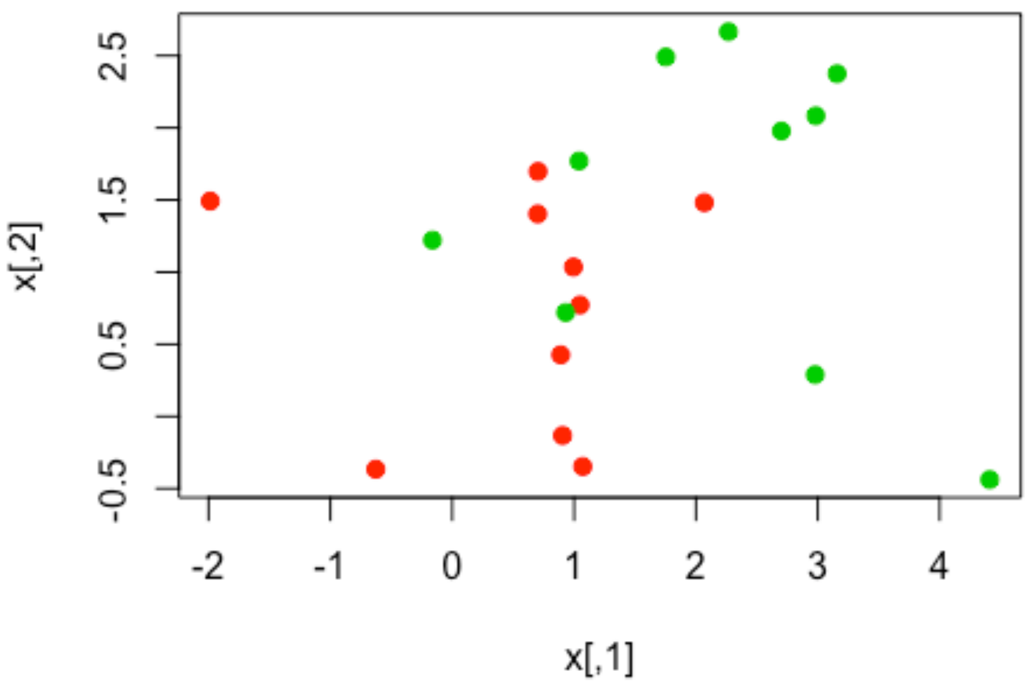
```r
tune.out <- tune(svm, y ~., data=dat, kernel='linear',
                 ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.7
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.70  0.4216370
## 2 1e-02  0.70  0.4216370
## 3 1e-01  0.75  0.3535534
## 4 1e+00  0.85  0.2415229
## 5 5e+00  0.85  0.2415229
## 6 1e+01  0.85  0.2415229
## 7 1e+02  0.85  0.2415229

#The best cost is 1 for the output.
# best performance: 0.25

# Getting the best Model

bestmod = tune.out$best.model
summary(bestmod)

##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost =
c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.001
##       gamma:  0.5
##
## Number of Support Vectors:  20
##
##  ( 10 10 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1

#Here we see that cost= 1 results in the lowest cross-validation error rate.

#Generating the Test data

xtest=matrix(rnorm(20*2), ncol=2)
ytest=sample(c(-1,1), 20, rep=TRUE)
xtest [ ytest ==1 ,]= xtest [ ytest ==1 ,] + 1
testdat=data.frame(x=xtest, y=as.factor(ytest))
yhat <- predict(tune.out$best.model, testdat)
#install.packages("caret")
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone
'zone/tz/2018c.
## 1.0/zoneinfo/America/New_York'
```




```
confusionMatrix(yhat, testdat$y)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1 1
##         -1  5 3
##         1   7 5
##
##                Accuracy : 0.5
##                  95% CI : (0.272, 0.728)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.8725
##
##                   Kappa : 0.0385
##  Mcnemar's Test P-Value : 0.3428
##
##             Sensitivity : 0.4167
##             Specificity : 0.6250
##          Pos Pred Value : 0.6250
##          Neg Pred Value : 0.4167
##              Prevalence : 0.6000
##          Detection Rate : 0.2500
```

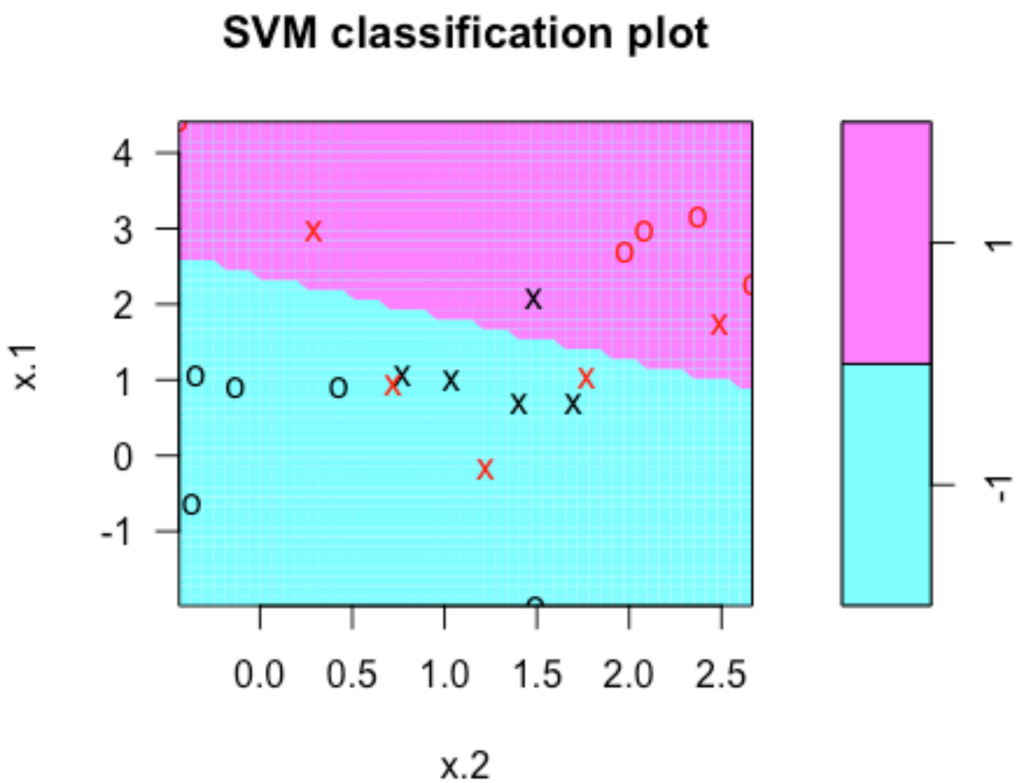
```
##     Detection Prevalence : 0.4000
##        Balanced Accuracy : 0.5208
##
##         'Positive' Class : -1
##

#consider a situation in which the two classes are linearly separable
x[y==1 ,]= x[y==1 ,]+01
plot(x, col =(y+5) /2, pch =19)
```



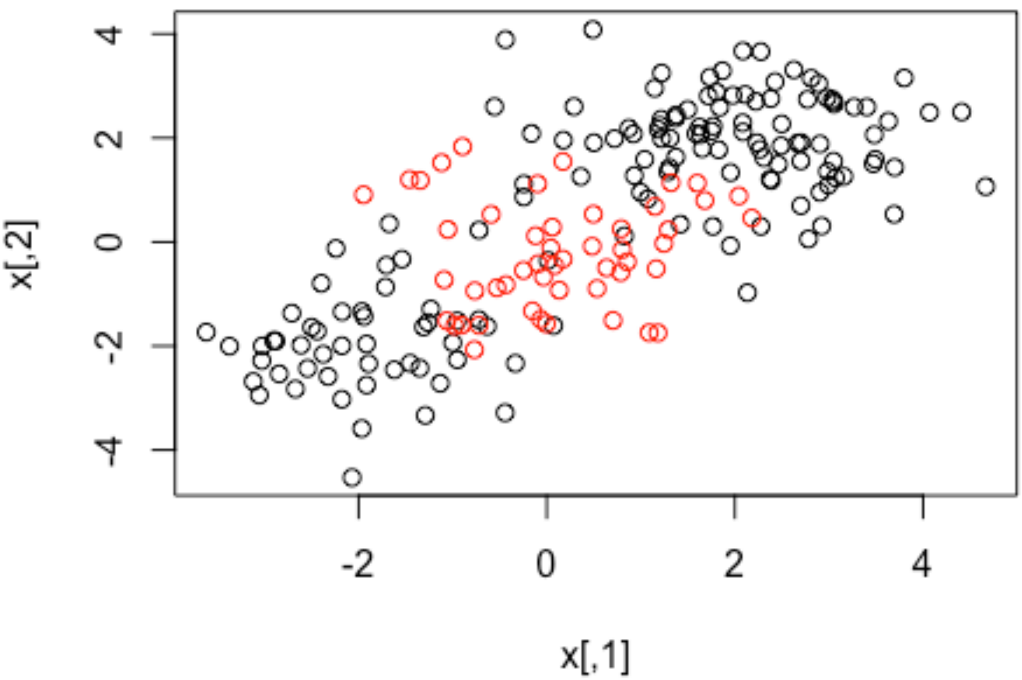

```
# Fiting the support vector classifier and plotting the hyperplane
dat=data.frame(x=x,y=as.factor (y))
svmfit =svm(y~ ., data=dat , kernel ="linear", cost =1e5)
summary (svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
```

```
##       gamma:  0.5
##
## Number of Support Vectors:  10
##
##  ( 5 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

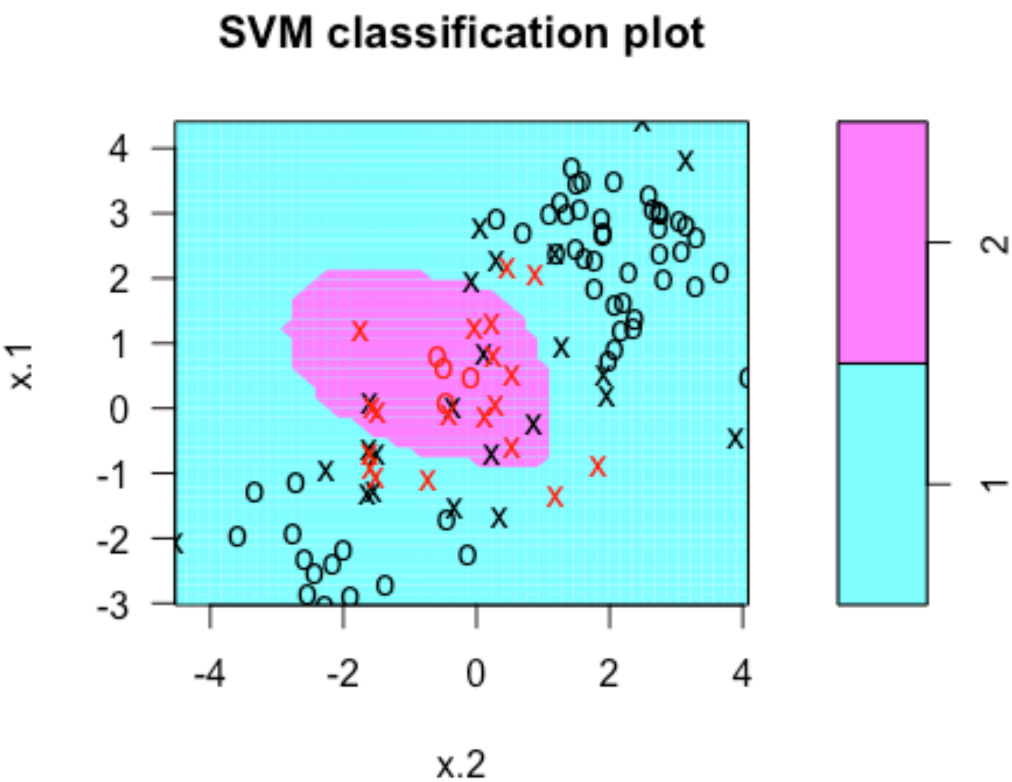
```
plot(svmfit , dat)
```




```
#Generating the data with a non-linear class boundary
set.seed (6372)
x=matrix (rnorm (200*2) , ncol =2)
x[1:100 ,]=x[1:100 ,]+2
x[101:150 ,]= x[101:150 ,] -2
y=c(rep (1 ,150) ,rep (2 ,50) )
dat=data.frame(x=x,y=as.factor (y))
plot(x, col=y)
```




```
#Fiting the training data
train=sample (200 ,100)
svmfit =svm(y~., data=dat [train, ], kernel ="radial", gamma =1,cost =1)
plot(svmfit , dat[train ,])
```



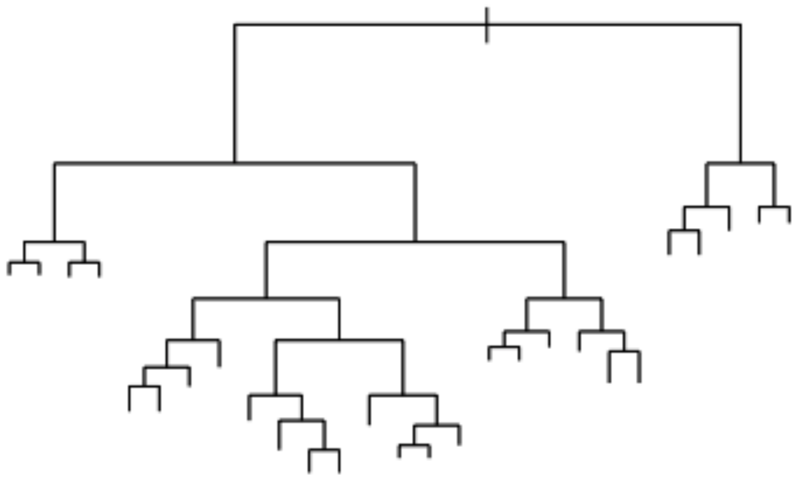

```
set.seed (6372)
tune.out=tune(svm , y~., data=dat[train ,], kernel ="radial", ranges
=list(cost=c(0.1 ,1 ,10 ,100 ,1000), gamma=c(0.5,1,2,3,4) ))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.14
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.23 0.13374935
## 2  1e+00   0.5  0.14 0.08432740
## 3  1e+01   0.5  0.16 0.06992059
## 4  1e+02   0.5  0.16 0.08432740
## 5  1e+03   0.5  0.15 0.09718253
```

```
## 6  1e-01   1.0  0.23 0.13374935
## 7  1e+00   1.0  0.16 0.08432740
## 8  1e+01   1.0  0.16 0.08432740
## 9  1e+02   1.0  0.15 0.08498366
## 10 1e+03   1.0  0.14 0.09660918
## 11 1e-01   2.0  0.23 0.13374935
## 12 1e+00   2.0  0.17 0.09486833
## 13 1e+01   2.0  0.16 0.09660918
## 14 1e+02   2.0  0.14 0.09660918
## 15 1e+03   2.0  0.19 0.09944289
## 16 1e-01   3.0  0.23 0.13374935
## 17 1e+00   3.0  0.19 0.09944289
## 18 1e+01   3.0  0.16 0.10749677
## 19 1e+02   3.0  0.17 0.12516656
## 20 1e+03   3.0  0.20 0.11547005
## 21 1e-01   4.0  0.23 0.13374935
## 22 1e+00   4.0  0.19 0.09944289
## 23 1e+01   4.0  0.16 0.10749677
## 24 1e+02   4.0  0.18 0.12292726
## 25 1e+03   4.0  0.20 0.11547005
```

```r
#Therefore, the best choice of parameters involves cost=1 and gamma=0.5

#Percentage of Misclassified objects
yhat <- predict(tune.out$best.model, dat[-train,])
confusionMatrix(yhat, dat[-train, 'y'])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 68 11
##          2  5 16
##
##                Accuracy : 0.84
##                  95% CI : (0.7532, 0.9057)
##     No Information Rate : 0.73
##     P-Value [Acc > NIR] : 0.006802
##
##                   Kappa : 0.5636
##  Mcnemar's Test P-Value : 0.211300
##
##             Sensitivity : 0.9315
##             Specificity : 0.5926
##          Pos Pred Value : 0.8608
##          Neg Pred Value : 0.7619
##              Prevalence : 0.7300
##          Detection Rate : 0.6800
##    Detection Prevalence : 0.7900
##       Balanced Accuracy : 0.7620
```

```
##
##        'Positive' Class : 1
##

#The optimal values of cost is 1 and gamma=0.5. Percentage of misclassified
objects is 10 percent.


#Decision Trees for Classification
library (tree)
library (ISLR)

## Warning: package 'ISLR' was built under R version 3.4.2

attach (Carseats)
View(Carseats)

## Warning: running command ''/usr/bin/otool' -L '/Library/Frameworks/
## R.framework/Resources/modules/R_de.so'' had status 1

High=ifelse (Sales <=8," No"," Yes ")
Carseats =data.frame(Carseats ,High)
tree.carseats =tree(High~.-Sales ,Carseats )
summary (tree.carseats )

##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400

#Plotting the tree carseats
plot(tree.carseats )
```



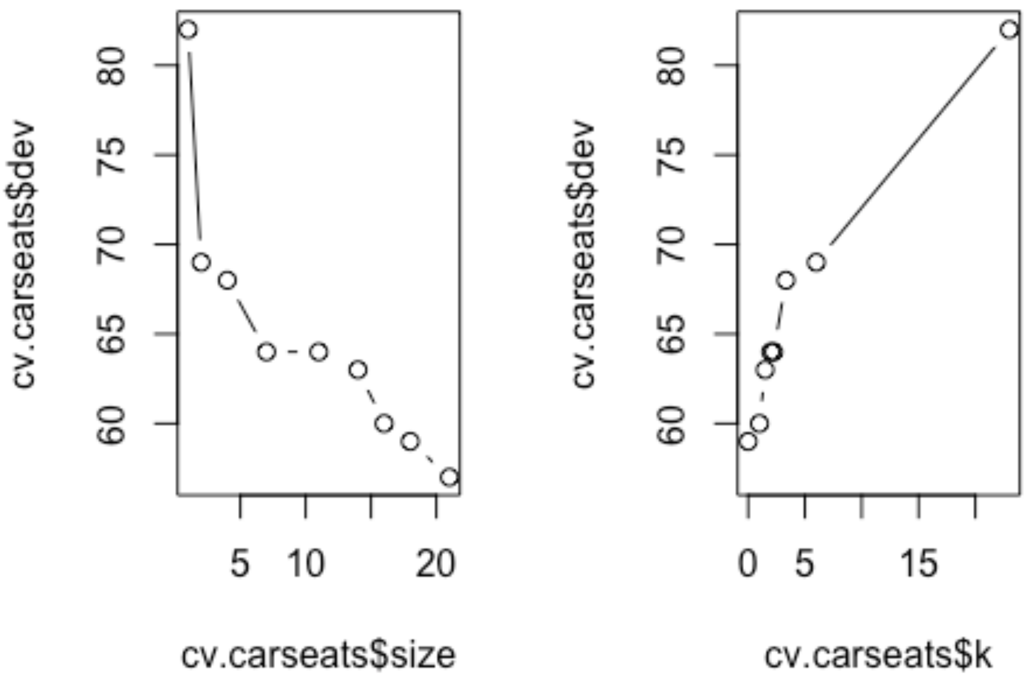
```
tree.carseats

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500  No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600  No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530  Yes  ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220  No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000  No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730  Yes  ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470  Yes  ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170  Yes  ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941  Yes  ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800  No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200  No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890  No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150  No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300  No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000  No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407  Yes  ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477  No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000  No ( 1.00000 0.00000 ) *
```

```
##          21) CompPrice > 124.5 128 150.200  No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680  Yes  ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702  No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930  Yes  ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481  Yes  ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600  No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050  Yes  ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702  No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540  No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400  No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010  No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300  Yes  ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742  Yes  ( 0.14286 0.85714 )
*
##                 349) CompPrice > 152.5 5   5.004  No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000  No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830  Yes  ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020  Yes  ( 0.20000 0.80000 )
##            44) CompPrice < 130.5 14  18.250  Yes  ( 0.35714 0.64286 )
##              88) Income < 100 9  12.370  No ( 0.55556 0.44444 ) *
##              89) Income > 100 5   0.000  Yes  ( 0.00000 1.00000 ) *
##            45) CompPrice > 130.5 11   0.000  Yes  ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490  No ( 0.75000 0.25000 )
##            46) CompPrice < 122.5 10   0.000  No ( 1.00000 0.00000 ) *
##            47) CompPrice > 122.5 10  13.860  No ( 0.50000 0.50000 )
##              94) Price < 125 5   0.000  Yes  ( 0.00000 1.00000 ) *
##              95) Price > 125 5   0.000  No ( 1.00000 0.00000 ) *
##     3) ShelveLoc: Good 85  90.330  Yes  ( 0.22353 0.77647 )
##       6) Price < 135 68  49.260  Yes  ( 0.11765 0.88235 )
##        12) US: No 17  22.070  Yes  ( 0.35294 0.64706 )
##          24) Price < 109 8   0.000  Yes  ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460  No ( 0.66667 0.33333 ) *
##        13) US: Yes 51  16.880  Yes  ( 0.03922 0.96078 ) *
##       7) Price > 135 17  22.070  No ( 0.64706 0.35294 )
##        14) Income < 46 6   0.000  No ( 1.00000 0.00000 ) *
##        15) Income > 46 11  15.160  Yes  ( 0.45455 0.54545 ) *
```

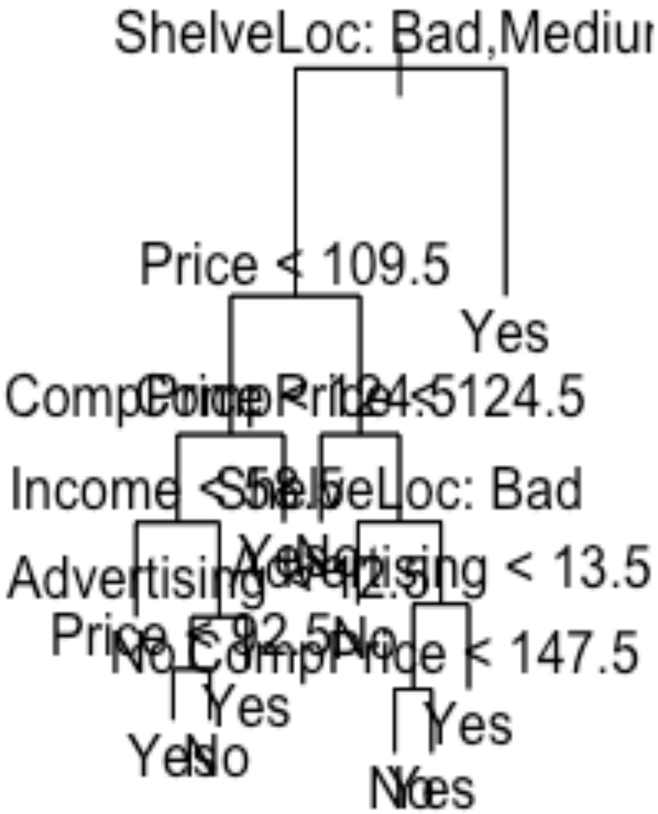
```r
#Evaluating the performance of Classification
set.seed (6372)
train=sample (1: nrow(Carseats), 200)
Carseats.test=Carseats [-train ,]
High.test=High[-train ]
tree.carseats =tree(High~.-Sales ,Carseats ,subset =train )
tree.pred=predict (tree.carseats ,Carseats.test ,type ="class")
table(tree.pred ,High.test)
```

```
##          High.test
## tree.pred  No  Yes
##       No    95   20
##       Yes   23   62
```

```r
#Optimal number of leaves
set.seed (6388)
cv.carseats =cv.tree(tree.carseats ,FUN=prune.misclass )
names(cv.carseats)

## [1] "size"   "dev"    "k"      "method"

cv.carseats

## $size
## [1] 21 18 16 14 11  7  4  2  1
##
## $dev
## [1] 57 59 60 63 64 64 68 69 82
##
## $k
## [1]      -Inf  0.000000  1.000000  1.500000  2.000000  2.250000  3.333333
## [8]  6.000000 23.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"

#Plotting the error rate
par(mfrow =c(1,2))
plot(cv.carseats$size ,cv.carseats$dev ,type="b")
plot(cv.carseats$k ,cv.carseats$dev ,type="b")
```




```
#Applyting the prune.misclass

prune.carseats =prune.misclass(tree.carseats,best =9)
plot(prune.carseats)
text(prune.carseats,pretty=0)
```


ShelveLoc: Bad,Mediur
Price < 109.5
Yes
ShelveLoc: Bad
Advertising < 13.5
Yes
Yes