Project II

Section C: Trees

I. Classification Trees:

1) Loading the libraries:
library(rpart); library(caret)
rpart() package is used to create the tree. It allows us to grow the whole tree using all the attributes present in the data.

Loading data, setting the names of columns, and set columns type:
mydata <- read.table('heart.dat')
mydata_names <- c('age', 'sex', 'chestPain', 'bloodPressure', 'cholestoral', 'bloodSugar', 'elecResults', 'maxHeartRate', 'inducedAngina','oldpeak', 'slopeST','numVessels','thal', 'heartDisease')
names(mydata) <- mydata_names
mydata$sex <- as.factor(mydata$sex)
And in same way to deal with other attributes.

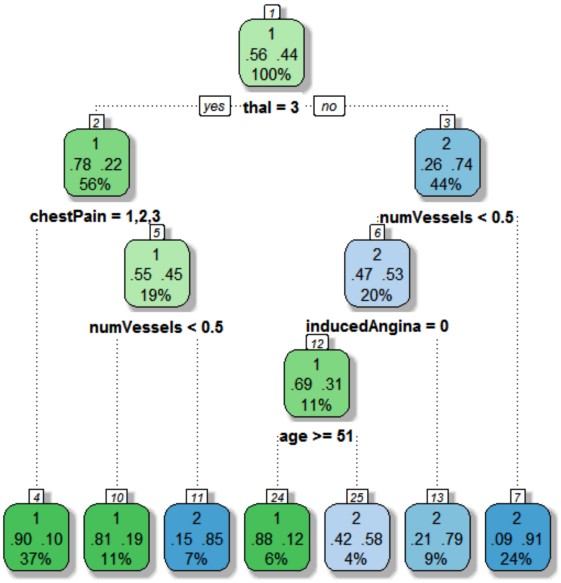
First using rpart to build a tree model:
frmla <- heartDisease~.
Using 4-fold cross-validation:
tc <- trainControl("cv", 4)
(train.rpart <- train(frmla, data = mydata, method = "rpart", trControl=tc))

k-fold	Accuracy	Size	Runtime
4	0.8036655	13	0.64
5	0.8148148	13	0.90
6	0.7814815	13	0.89
7	0.7553885	13	0.75
8	0.7776292	13	0.77

my.tree <- rpart(frmla, data = mydata, method = “class”)
rpart syntax takes ‘dependent attribute’ and the rest of the attributes are independent in the analysis. rpart() returns a Decison tree created for the data.
To enhance it, let us take some help from rattle :
library(rattle); library(rpart.plot); library(RColorBrewer)

Rattle() is one unique feature of R which is specifically built for data mining in R. rpart.plot() and RcolorBrewer() functions help us to create a beautiful plot. ‘rpart.plot()’ plots rpart models. It extends plot.rpart and text.rpart in the rpart package. RcolorBrewer() provides us with beautiful color palettes and graphics for the plots.

fancyRpartPlot(my.tree)



2) To validate the model we use the printcp and plotcp functions. 'CP' stands for Complexity Parameter of the tree.

Syntax : printcp (x) where x is the rpart object.

We prune the tree to avoid any overfitting of the data. The convention is to have a small tree and the one with least cross validated error given by printcp() function i.e. 'xerror'. To find out how the tree performs, is calculated by the printcp() function, based on which we can go ahead and prune the tree.

printcp(my.tree)

Classification tree:

rpart(formula = frmla, data = mydata, method = "class")

Variables actually used in tree construction:

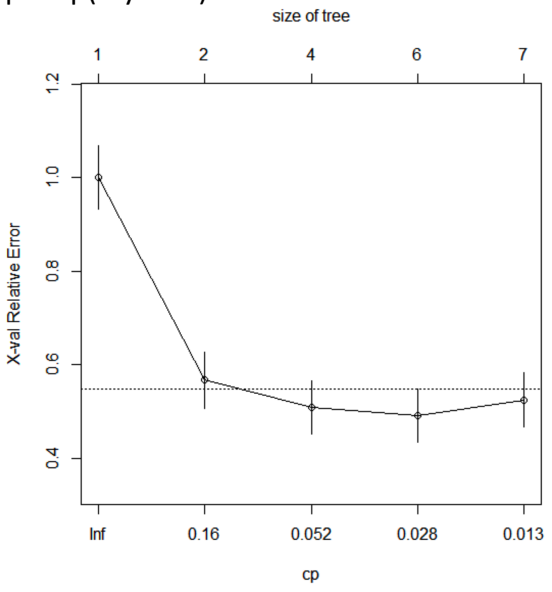
[1] age chestPain inducedAngina numVessels thal

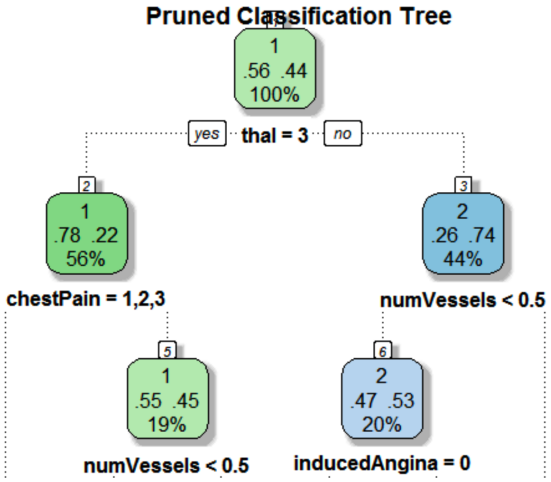
Root node error: 120/270 = 0.44444

From above we can see that the root node error is 0.44.

The value of cp should be least, so that the cross-validated error rate is minimum.

plotcp(my.tree)





Plotcp() provides a graphical representation to the cross validated error summary. Prune the tree to create an optimal decision tree :
ptree <- prune(my.tree, cp= my.tree$cptable[which.min(my.tree$cptable[,"xerror"]),"CP"])
fancyRpartPlot(ptree, uniform=TRUE,main="Pruned Classification Tree")
We can see that after prune, the age node disappeared. The most important attributes are age, chestPain, inducedAngina, numVessels, and thal.

3) randomForest implements Breiman's random forest algorithm for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points. Random forests generates its sequence of models by training them on subsets of the data. The subsets are drawn at random from the full training set.
The random forests algorithm (for both classification and regression) is as follows:

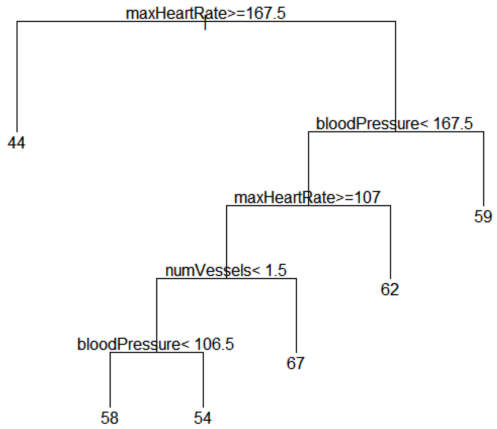
1. Draw ntree bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an un- pruned classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predic- tors, randomly sample mtry of the predictors and choose the best split from among those variables. (Bagging can be thought of as the special case of random forests obtained when mtry = p, the number of predictors.)
3. Predict new data by aggregating the predic- tions of the ntree trees (i.e., majority votes for classification, average for regression).

4) library(randomForest)
mydata$heartDisease <- as.factor(mydata$heartDisease)
fit <- randomForest(frmla, data=mydata, importance=TRUE, ntree=2000)

Ntree	No. of variables tried at each split	OOB estimate of error rate	runtime
200	3	18.89%	0.08
500	3	16.67%	0.24
1000	3	17.78%	0.44
1500	3	18.15%	0.91
2000	3	17.415	0.88

II. Regression Trees:
1) frmla <- age~.
tc <- trainControl("cv", 4)
ptm <- proc.time()

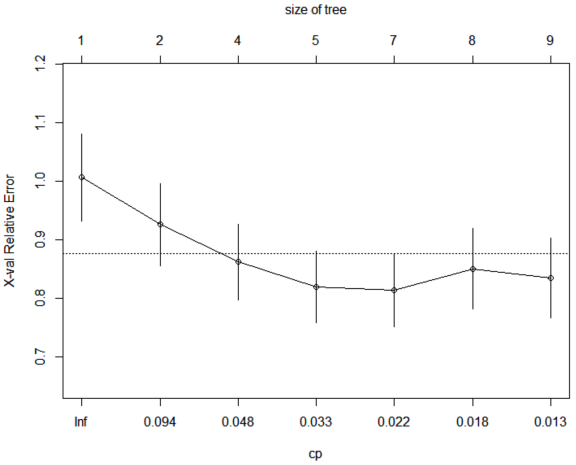
(train.rpart <- train(frmla, data = mydata, method = "rpart", trControl=tc)); proc.time() - ptm

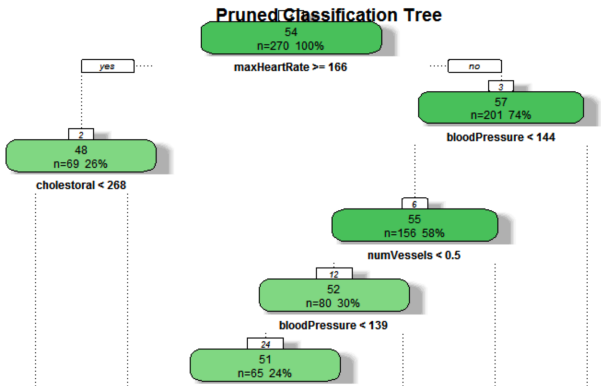


K-fold	Cp	RMSE	Rsquared	RMSE SD	Size	Runtime
4	0.04215786	8.043341	0.24057399	0.5667254	11	0.67
5	0.04215786	8.368788	0.19663620	1.3654864	11	0.67
6	0.04215786	7.962834	0.25007372	0.6828587	11	0.66
7	0.04215786	8.495650	0.16278646	0.5582985	11	0.67
8	0.04215786	8.171950	0.21596851	0.8549097	11	0.71

4) Using same method as the classification tree

> printcp(my.tree)
> plotcp(my.tree)
> ptree <- prune(my.tree, cp= my.tree$cptable[which.min(my.tree$cptable[,"xerror"]),"CP"])
> fancyRpartPlot(ptree, uniform=TRUE,main="Pruned Classification Tree")





Regression tree:
rpart(formula = frmla, data = mydata)

Variables actually used in tree construction:
[1] bloodPressure cholestoral heartDisease
[4] maxHeartRate numVessels oldpeak

Root node error: 22320/270 = 82.668

We can see that after prune tree, the variables actually used in tree construction including: bloodPressure, cholestoral, heartDisease, maxHeartRate, numVessel, and oldpeak. The number of node decreased from 13 to 6.

Section D: Artificial Neural Networks and Deep Learning

I. Classification using Artificial Neural Networks:

1) Firstly, I want to use neuralnet library in R. However, when I built the model and got the result, I found that the predictions were all near 1. Then I thought there have some problems and I tried to adjust the parameters but the results didn't change, no matter how many hidden layers I used. Then I tried the other library called nnet. This time I can get the normal result but the nnet only provides one hidden layer.

2) Firstly, I tried the neuralnet:
Neuralnet is used to train neural networks using backpropagation, resilient backpropagation with or without weight backtracking or the modified globally convergent version .

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

```
f <- V65 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
   V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20 + V21 +
   V22 + V23 + V24 + V25 + V26 + V27 + V28 + V29 + V30 + V31 +
   V32 + V33 + V34 + V35 + V36 + V37 + V38 + V39 + V40 + V41 +
   V42 + V43 + V44 + V45 + V46 + V47 + V48 + V49 + V50 + V51 +
   V52 + V53 + V54 + V55 + V56 + V57 + V58 + V59 + V60 + V61 +
   V62 + V63 + V64
nn <- neuralnet(f, data = traindata, algorithm = "rprop+",hidden = c(10,5), linear.output = F,
act.fct = "logistic")
```

However, no matter what parameters I adjusted the results were all near 1. I used the same parameters on the other datasets I can get the normal results, then I suspected the dataset not applied to this library.
So next I decided to change to the other neural net library: nnet

Nnet model fits single-hidden-layer neural network, possibly with skip-layer connections. Multinom function is a function in nnet library fits multinomial log-linear models via neural networks.

Usage:

```
multinom(formula, data, weights, subset, na.action,
         contrasts = NULL, Hess = FALSE, summ = 0, censored = FALSE,
         model = FALSE, ...)
```

3 & 4) Run the experiments:

```
optdigits.tra$V65 <- as.factor(optdigits.tra$V65)
optdigits.tes$V65 <- as.factor(optdigits.tes$V65)
```

mymodel <- multinom(f, data = optdigits.tra, maxit = 500, trace=T)
Firstly, I want to see which variables are more important in this model:
topmodels <- varImp(mymodel)
topmodels$Variables <- row.names(topmodels)
topmodels <- topmodels[order(-topmodels$Overall),]
> head(topmodels)

	Overall	Variables
V41	3728.337668	V41
V64	3198.621576	V64
V48	3013.202163	V48
V56	2934.146871	V56
V49	2550.168265	V49
V25	1843.437888	V25

From above we can see that variables V41, V64, V48, V56,V49, and V25 are more important variables in this neural net model.
Next I wanted to use this model to do prediction. In this part, I have done two parts of prediction, one for probability and one for classification:
preds1 <- predict(mymodel, type = "probs", newdata = optdigits.tes)
preds2 <- predict(mymodel, type = "class", newdata = optdigits.tes)

```
> head(preds1)
  0             1 2 3 4             5             6             7             8             9
1 1  0.000000000e+00 0 0 0  0.000000000e+00  0.000000000e+00  0.000000000e+00  0.000000000e+00  0.000000000e+00
2 0  1.000000000e+00 0 0 0  0.000000000e+00  0.000000000e+00  0.000000000e+00 5.166444459e-320  0.000000000e+00
3 0 6.083391250e-202 1 0 0  0.000000000e+00 5.849227967e-217  0.000000000e+00 1.738458819e-239  0.000000000e+00
4 0  0.000000000e+00 0 1 0  0.000000000e+00  0.000000000e+00 8.834397924e-303  0.000000000e+00 7.159769926e-137
5 0  0.000000000e+00 0 0 1  0.000000000e+00  0.000000000e+00  0.000000000e+00  0.000000000e+00  0.000000000e+00
6 0 4.922174418e-291 0 1 0 9.266468521e-199  0.000000000e+00  0.000000000e+00 2.313583326e-264  8.112354249e-13
```

We can see that the probabilities of the first six instances are all 1 when doing the prediction.
Next I show the first six instances classification results:
> head(preds2)
[1] 0 1 2 3 4 3
library(Metrics)

> postResample(optdigits.tes$V65, preds2)

Accuracy	Kappa
0.8870339455	0.8744698042

The total classification prediction accuracy on test data is 88.7%, its pretty good.

```
> table(optdigits.tes$V65, preds2)
   preds2
      0   1   2   3   4   5   6   7   8   9
  0 170   1   0   0   1   6   0   0   0   0
  1   1 170   0   0   4   1   3   1   1   1
  2   4   7 156   1   0   0   6   1   1   1
  3   0   0  10 155   0   2   2   8   3   3
  4   0   8   0   0 153   1   9   3   1   6
  5   0   0   1   5   1 173   0   1   0   1
  6   4   2   0   0   4   3 168   0   0   0
  7   0   0   3   0   2  17   2 150   0   5
  8   2   5   0   7   3   5   6   4 141   1
  9   1   6   0   0   2   5   0   5   3 158
```

From the confusion matrix above we can see that the most error classification part coming from class five.

The multinom function does not have too many parameters could be adjusted, then I decided to run a 10-fold cross-validation experiment for it.

```
totalError <- c()
cv <- 10
cvDivider <- floor(nrow(optdigits.tra)/(cv+1))

# 10-fold cross-validation test:
ptm <- proc.time()
for (cv in seq(1:cv)){
  # assign chunk to data test
  dataTestIndex <- c((cv * cvDivider):(cv*cvDivider + cvDivider))
  dataTest <- optdigits.tra[dataTestIndex,]
  # everything else to train
  dataTrain <- optdigits.tra[-dataTestIndex,]
  myModel <- multinom(f, data = optdigits.tra, maxit = 500, trace = T)
  pred <- predict(myModel, dataTest)

  # classification error
  err <- ce(dataTest$V65, pred)
  totalError <- c(totalError, err)
}
proc.time() – ptm

> proc.time() - ptm
   user  system elapsed
 87.735   0.331  88.818

> totalError
 [1] 0 0 0 0 0 0 0 0 0 0
```

The total value on this training data set is 0, that means the model perfect match the training data set.

Section E: Support Vector Machines

I Classification using Support Vector Machines (SVMs)

```
library(e1071)
1)  Replace the original target value “8” to “1”, else to “-1”
optdigits.tra <- read.table("optdigits.tra",sep=",")
optdigits.tes <- read.table("optdigits.tes", sep = ",")
traindata <- optdigits.tra
traindata$V65[traindata$V65 != 8] <- -1
traindata$V65[traindata$V65 == 8] <- 1
traindata$V65 <- as.factor(traindata$V65)
# scale the data and reset col1 and col40 to 0
traindata[,1:64] <- apply(traindata[,1:64], 2, scale)
traindata[,1] <- 0
traindata[,40] <- 0

testdata <- optdigits.tes
testdata$V65[testdata$V65 != 8] <- -1
testdata$V65[testdata$V65 == 8] <- 1
testdata$V65 <- as.factor(testdata$V65)
# scale the data and reset col1,col33 and col40 to 0
testdata[,1:64] <- apply(testdata[,1:64],2,scale)
testdata[,1] <- 0
testdata[,33] <- 0
testdata[,40] <- 0

2)
> svmfit <- svm(f, data = traindata, kernel = "linear", type="C", cost = 1, scale = FALSE)
> summary(svmfit)
Call:
svm(formula = f, data = traindata, kernel = "linear", type = "C", cost = 1, scale = FALSE)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.015625
Number of Support Vectors:  244

> pred <- predict(svmfit, testdata)
> (result <- postResample(testdata$V65, pred))
    Accuracy       Kappa
0.9621591541 0.7602787292
```

In this experiment, I used the linear kernel and cost is 1 to do the svm model. The number of support vectors in this model is 244. The accuracy of this model on test data is 0.962.

```
3)  library(e1071)
svmfit1 <- svm(f, data = traindata, kernel = "linear", type="C", cost = 0.1, scale = FALSE)
```

```
svmfit2 <- svm(f, data = traindata, kernel = "linear", type="C", cost = 1, scale = FALSE)
svmfit3 <- svm(f, data = traindata, kernel = "linear", type="C", cost = 10, scale = FALSE)
pred1 <- predict(svmfit1, testdata)
# table(predict = pred1, truth = testdata$V65)
(result1 <- postResample(testdata$V65, pred1))
pred2 <- predict(svmfit2, testdata)
(result2 <- postResample(testdata$V65, pred2))
pred3 <- predict(svmfit3, testdata)
(result3 <- postResample(testdata$V65, pred3))
```

```
svmfit1 <- svm(f, data = traindata, kernel = "polynomial", type="C", degree = 3, scale = FALSE)
svmfit2 <- svm(f, data = traindata, kernel = "polynomial", type="C", degree = 6, scale = FALSE)
svmfit3 <- svm(f, data = traindata, kernel = "polynomial", type="C", degree = 10, scale = FALSE)
pred1 <- predict(svmfit1, testdata)
 (result1 <- postResample(testdata$V65, pred1))
pred2 <- predict(svmfit2, testdata)
(result2 <- postResample(testdata$V65, pred2))
pred3 <- predict(svmfit3, testdata)
(result3 <- postResample(testdata$V65, pred3))
```

```
svmfit1 <- svm(f, data = traindata, kernel = "radial", type="C", gamma = 0.01, scale = FALSE)
svmfit2 <- svm(f, data = traindata, kernel = "radial", type="C", gamma = 0.1, scale = FALSE)
svmfit3 <- svm(f, data = traindata, kernel = "radial", type="C", gamma = 0.5, scale = FALSE)
pred1 <- predict(svmfit1, testdata)
 (result1 <- postResample(testdata$V65, pred1))
pred2 <- predict(svmfit2, testdata)
(result2 <- postResample(testdata$V65, pred2))
pred3 <- predict(svmfit3, testdata)
(result3 <- postResample(testdata$V65, pred3))
```

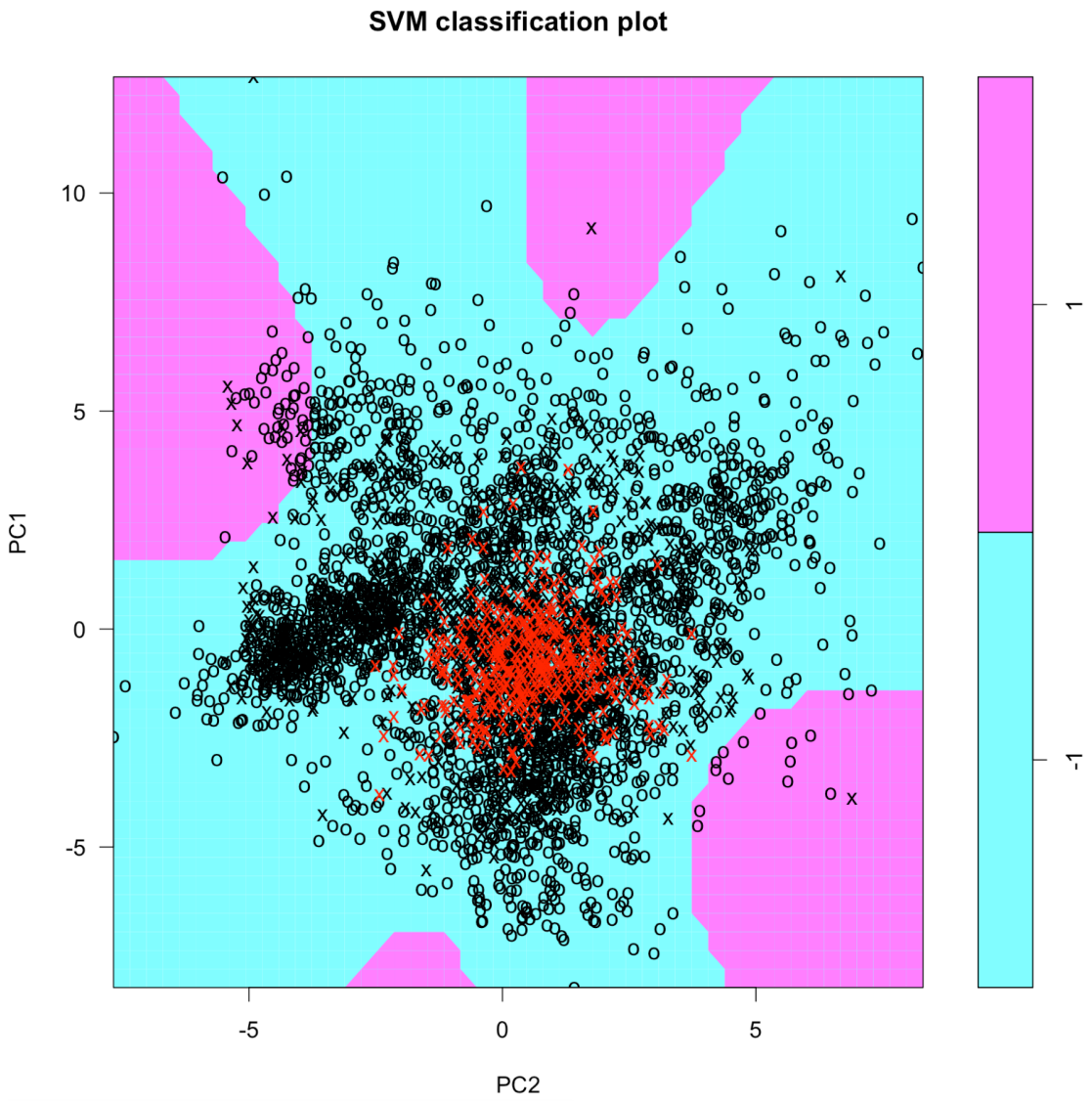
```
svmfit1 <- svm(f, data = traindata, kernel = "sigmoid", type="C", coef0 = 0, scale = FALSE)
svmfit2 <- svm(f, data = traindata, kernel = "sigmoid", type="C", coef0 = 1, scale = FALSE)
svmfit3 <- svm(f, data = traindata, kernel = "sigmoid", type="C", coef0 = 2, scale = FALSE)
pred1 <- predict(svmfit1, testdata)
 (result1 <- postResample(testdata$V65, pred1))
pred2 <- predict(svmfit2, testdata)
(result2 <- postResample(testdata$V65, pred2))
pred3 <- predict(svmfit3, testdata)
(result3 <- postResample(testdata$V65, pred3))
```

Kernel	Parameters	Accuracy
Linear	Cost = 0.1	0.966
	Cost = 1	0.962
	Cost = 10	0.962
polynomial	Degree = 3	0.987

	Degree = 6	0.988
	Degree = 10	0.983
radial	gamma = 0.01	0.903
	gamma = 0.1	0.903
	gamma = 0.5	0.903
sigmoid	coef0 = 0	0.903
	coef0 = 1	0.903
	coef0 = 2	0.903

4) In this part, in order to visualize the SVM model in 2-Dimension, then I will use the PCA first, then build the SVM model second, and last plot the SVM model in 2-Dimension.

```
optdigits.tra <- read.table("optdigits.tra",sep=",")
# remove all 0 value columns and V65
traindata <- optdigits.tra
mytest <- traindata[, -c(1,40)]
pca <- prcomp(mytest[,-63], retx=TRUE, center=TRUE, scale=TRUE)
summary(pca)
newdata <- pca$x[,1:2]
newdata <- data.frame(newdata)
# set the target variable
newdata[,3] <- traindata[,65]
newdata$V3[newdata$V3 != 8] <- -1
newdata$V3[newdata$V3 == 8] <- 1
newdata$V3 <- as.factor(newdata$V3)
library(e1071)
svmfit <- svm(V3 ~ PC1+PC2, data = newdata, kernel = "polynomial", type="C", degree = 6, scale = FALSE)
plot(svmfit, newdata)
```


SVM classification plot
10
5
0
-5
PC1
-5
0
5
PC2
1
-1