CS6140 Assignment 4

Chengbo Gu

1. JWHT Chapter 7, Problem 1

(a) For all $x \le \xi$, $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

Thus, $a_1 = \beta_0, b_1 = \beta_1, c_1 = \beta_2, d_1 = \beta_3$

(b) For all $x > \xi$,

$$\begin{aligned} f(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)^3 \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 3x^2\xi + 3x\xi^2 - \xi^3) \\ &= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\xi^2 \beta_4) x + (\beta_2 - 3\xi\beta_4) x^2 + (\beta_3 + \beta_4) x^3 \end{aligned}$$

Thus, $a_2 = \beta_0 - \beta_4 \xi^3, b_2 = \beta_1 + 3\xi^2\beta_4, c_2 = \beta_2 - 3\xi\beta_4, d_2 = \beta_3 + \beta_4$

(c) $f_1(\xi) = a_1 + b_1\xi + c_1\xi^2 + d_1\xi^3 = \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3$

$$\begin{aligned} f_2(\xi) &= a_2 + b_2\xi + c_2\xi^2 + d_2\xi^3 = (\beta_0 - \beta_4\xi^3) + (\beta_1 + 3\xi^2\beta_4)\xi + (\beta_2 - 3\xi\beta_4)\xi^2 + (\beta_3 + \beta_4)\xi^3 \\ &= \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 + (-\beta_4\xi^3 + 3\xi^3\beta_4 - 3\xi^3\beta_4 + \beta_4\xi^3) = \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 \end{aligned}$$

So we show that $f_1(\xi) = f_2(\xi)$.

(d) $f_1^{'}(\xi) = b_1 + 2c_1\xi + 3d_1\xi^2 = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2$

$$\begin{aligned} f_2^{'}(\xi) &= b_2 + 2c_2\xi + 3d_2\xi^2 = \beta_1 + 3\xi^2\beta_4 + 2(\beta_2 - 3\xi\beta_4)\xi + 3(\beta_3 + \beta_4)\xi^2 \\ &= \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2 + (3\xi^2\beta_4 - 6\xi^2\beta_4 + 3\xi^2\beta_4) = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2 \end{aligned}$$

We can see $f_1^{'}(\xi) = f_2^{'}(\xi)$.

(e) $f_1^{''}(\xi) = 2c_1 + 6d_1\xi = 2\beta_2 + 6\beta_3\xi$

$$f_2^{''}(\xi) = 2c_2 + 6d_2\xi = 2(\beta_2 - 3\xi\beta_4) + 6(\beta_3 + \beta_4)\xi = 2\beta_2 + 6\beta_3 + (-6\xi\beta_4 + 6\xi\beta_4) = 2\beta_2 + 6\beta_3$$

We could safely draw the conclusion that $f_1^{''}(\xi) = f_2^{''}(\xi)$.

2. JWHT Chapter 7, Problem 3

For the given values of β, we have:

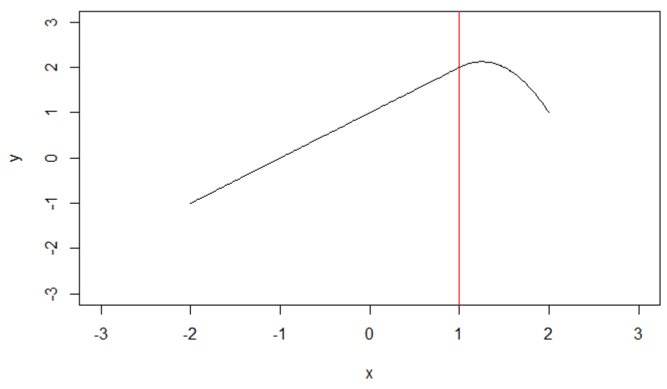
$$Y=\beta_0+\beta_1 X+\beta_2 (X-1)^2 I(X\geq 1)=1+X-2(X-1)^2 I(X\geq 1)$$

Thus,

$$Y=\begin{cases} 1+X & -2\leq X<1 \\ -2X^2+5X-1 & 1\leq X\leq 2 \end{cases}$$

```
curve(-2*x*x+5*x-1,1,2, xlim=c(-3,3), ylim=c(-3,3), ylab="y")
curve(x+1,-2,1, xlim=c(-3,3), ylim=c(-3,3), add=T)
abline(v=1,col="red")
```

The curve would be like:



problem 3. Naive Bayes classifier

Before implementing Na ïve Bayes classifier, I used sample function to partition the dataset into a training set and a validation set of equal size. Notice that there is a categorical feature famhist, I set “Present” to be 1 and “Absent” to be 0 accordingly.
Also, for question 3, we don’t need famhist as a predictor. So basically I made two types of data, one for LDA and Na ïve Bayes, one for other methods.

```
set.seed(123)
Rawdata <- read.table("SouthAfricanHeartDisease.txt", sep=",",
                     stringsAsFactors = FALSE, header = TRUE)
#Rawdata[,11] <- as.factor(Rawdata[,11])
Mydata <- Rawdata[,-6]
Mydata <- Mydata[,-1]

# for naive bayes and LDA
predictors <- Mydata[,1:8]
means <- apply(predictors, 2, mean)
sds <- apply(predictors, 2, sd)

predictors <- t(apply(predictors, 1, function(x) (x-means)/sds))
response <- Mydata[,9]

train <- sample(x=1:nrow(Mydata), size=nrow(Mydata)/2)
trainResponse <- response[train]
testResponse <- response[-train]
```

```
trainPredictors <- predictors[train,]
testPredictors <- predictors[-train,]

# for others
Mydata <- Rawdata[,-1]
Mydata[,5][Mydata[,5]=="Present"] <- 1
Mydata[,5][Mydata[,5]=="Absent"] <- 0
Mydata[,5] <- as.integer(Mydata[,5])
trainSet <- Mydata[train,]
testSet <- Mydata[-train,]
```

(a) Implement Naive Bayes classifier

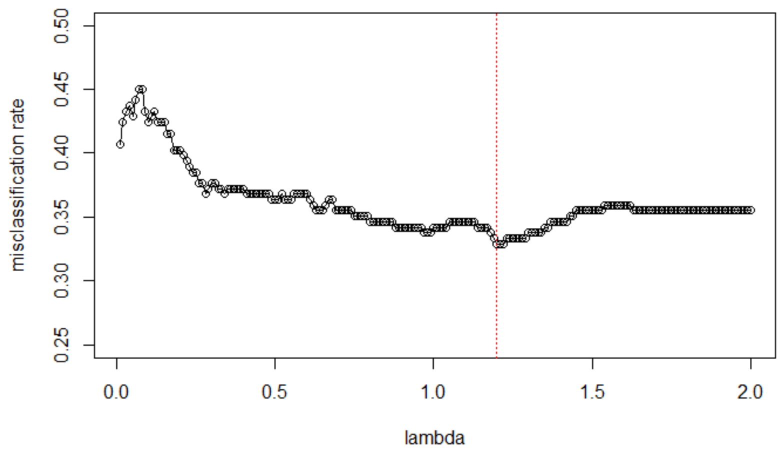
Please see appendix for implementation.

(b) Cross validation of Naive Bayes classifier

```
lambda.seq <- seq(0.01, 2, 0.01)
```

```
naive.bayes.cv(lambda.seq, trainPredictors, trainResponse)
```

```
## Best lambda is : 1.2

naiveBayes.train.scores <- naive.bayes.cl(1.2, trainPredictors, trainPredictors)
naiveBayes.test.scores <- naive.bayes.cl(1.2, testPredictors, trainPredictors)

mean((naiveBayes.train.scores > 0.5) == trainResponse)

## 0.6666667
```



Here I defined my own leave-one-out cross-validation for choosing the smooth parameter for Na ïve Bayes classifier. Again, the implementation could be found in Appendix.

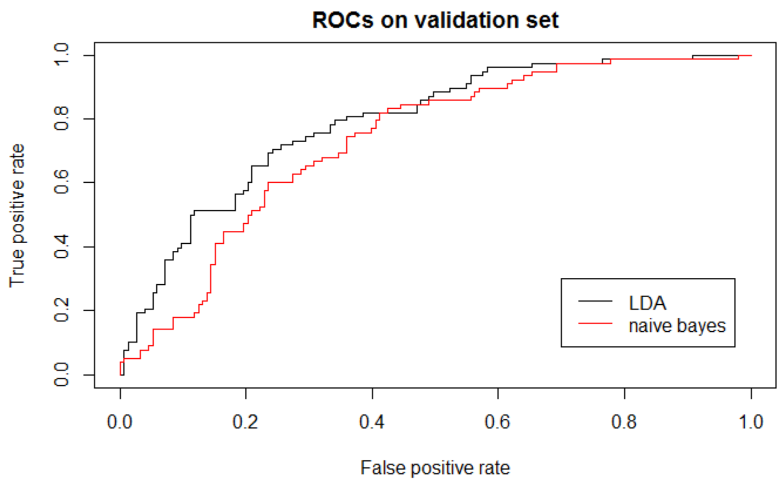
I first produced a grid for lambda which has 199 identical values range from 0 to 2. Then function na ïve.bayes.cv was called to explore the best lambda.

From the plot and the message produced by the function, the lambda was selected to be 1.2.

The accuracy on validation set is shown to be 66.67% and AUC is 0.7323 (could be find in part (c)).

(c) Compare the performances of Naïve Bayes classifier and LDA

```
lda.fit <- lda(x=as.matrix(trainPredictors), grouping= as.factor(trainResponse), cv=TRUE)

# lda
# ROC on the validation set
scores <- predict(lda.fit, newdata= testPredictors)$posterior[,2]
pred <- prediction( scores, labels= testResponse )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 1, main="ROCs on validation set")
# print out the area under the curve
lda.test <- unlist(attributes(performance(pred, "auc"))$y.values)

# naive bayes
pred <- prediction(naiveBayes.test.scores, labels=testResponse)
perf <- performance(pred, "tpr", "fpr")

# plot the ROC curve
plot(perf, col= 2, add=T)
# print out the area under the curve
naive.test <- unlist(attributes(performance(pred, "auc"))$y.values)
legend(0.7,0.3, c("LDA", "naive bayes"), col=c(1:2), lty=1)
```



Here we used ROCs to compare the performances. We see clearly that the LDA curve is above on the Na ïve Bayes curve most of the time. With the truth that AUC is 0.7901 for LDA and is 0.7323 for Na ïve Bayes. We could safely draw the conclusion that LDA outperforms Na ïve Bayes on validation set.

LDA is based on assumptions of multivariate normality and equality of covariance matrices of the 2 groups while Na ïve Bayes is based on assumption of independence between features. It is most likely that the assumption of Na ïve Bayes is violated more seriously than LDA's given this certain dataset. As the name of Na ïve Bayes indicates, Na ïve Bayes is a very simple method. It is not surprising to see it is outperformed by others.

problem 4. Tree-based Methods

(a) JWHT Chapter 8, Problem 9, but using the "South African Heart Disease" dataset

(a-b)

```
tree.cl <- tree(as.factor(chd) ~ ., data=trainSet)
summary(tree.cl)

##
## Classification tree:
```

```
## tree(formula = as.factor(chd) ~ ., data = trainSet)
## Number of terminal nodes:  26
## Residual mean deviance:  0.5249 = 107.6 / 205
## Misclassification error rate: 0.1169 = 27 / 231
```

Training error rate is 11.69%. There are 26 terminal nodes.

(c)

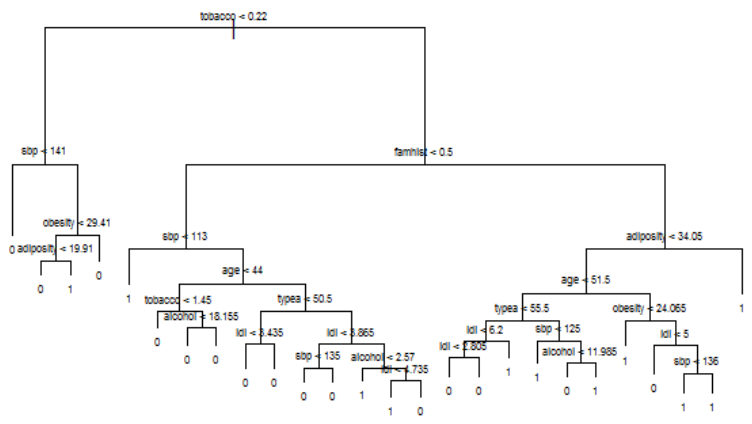
```
tree.cl
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 231 300.500 0 ( 0.64502 0.35498 )
##     2) tobacco < 0.22 67  40.400 0 ( 0.91045 0.08955 )
##       4) sbp < 141 49   0.000 0 ( 1.00000 0.00000 ) *
##       5) sbp > 141 18  22.910 0 ( 0.66667 0.33333 )
##        10) obesity < 29.41 12  16.640 0 ( 0.50000 0.50000 )
##          20) adiposity < 19.91 5   5.004 0 ( 0.80000 0.20000 ) *
##          21) adiposity > 19.91 7   8.376 1 ( 0.28571 0.71429 ) *
##        11) obesity > 29.41 6   0.000 0 ( 1.00000 0.00000 ) *
##   omit several lines here…
##          29) age > 51.5 32  35.990 1 ( 0.25000 0.75000 )
##            58) obesity < 24.065 9   0.000 1 ( 0.00000 1.00000 ) *
##            59) obesity > 24.065 23  29.720 1 ( 0.34783 0.65217 )
##             118) ldl < 5 9  11.460 0 ( 0.66667 0.33333 ) *
##             119) ldl > 5 14  11.480 1 ( 0.14286 0.85714 )
##               238) sbp < 136 5   6.730 1 ( 0.40000 0.60000 ) *
##               239) sbp > 136 9   0.000 1 ( 0.00000 1.00000 ) *
##        15) adiposity > 34.05 11   0.000 1 ( 0.00000 1.00000 ) *
```

Taking the last line of output for example, 15) adiposity > 34.05 11 0.000 1 (0.00000 1.00000). That means, to go to the 15th terminal node, adiposity should be greater than 34.05. There are 11 observation in this branch, the deviance is 0 and the prediction is chd = 1 with probability 1.

(d)

```
plot(tree.cl)
text(tree.cl ,pretty =0, cex=0.5)
```



The figure above shows the initial tree built for training set without pruning. It has many branches and 26 leaves, which is very likely to overfitting.

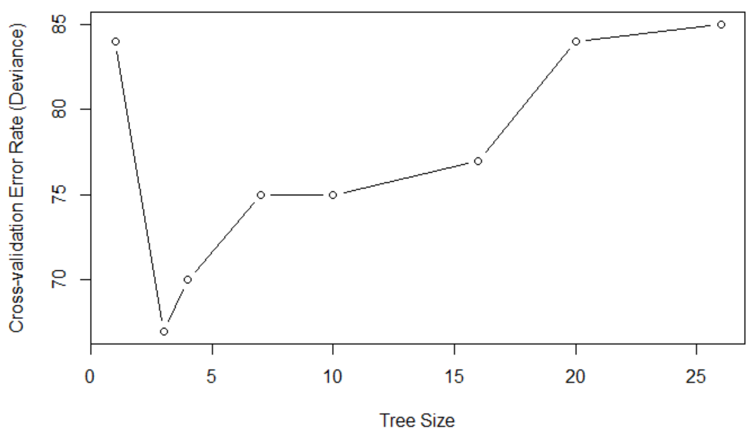
(e)

```
tree.pred <- predict (tree.cl , testSet, type ="class")
table(tree.pred, testSet[,10])

##
## tree.pred   0   1
##         0 117  40
##         1  36  38
```

From the confusion matrix, we find that the misclassification rate for testing set is (40+36)/231=32.90%.

(f-i)

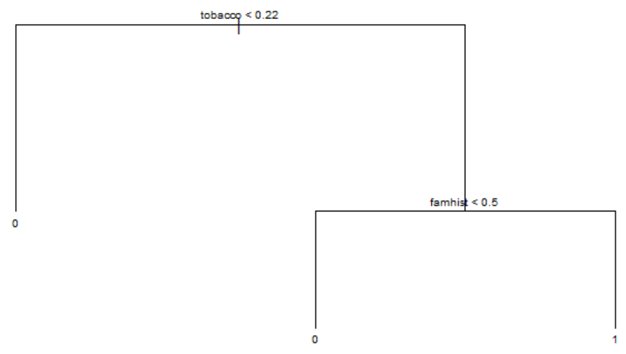
```
set.seed(258)
cv.tree.cl =cv.tree(tree.cl , FUN=prune.misclass)
plot(cv.tree.cl$size ,cv.tree.cl$dev ,type="b", ylab="Cross-validation Error Rate (Devian
ce)", xlab="Tree Size")
```



From the cross-validation plot, we find that the best size for a pruned tree is 3.

Then we can get the best tree by the following R codes.

```
prune.tree =prune.misclass (tree.cl ,best =3)
plot(prune.tree)
text(prune.tree ,pretty =0, cex=0.6)
```



(j)

```
tree.prune.train.pred <- predict(prune.tree, trainSet, type="class")
mean(tree.prune.train.pred != trainSet[,10])

## [1] 0.2554113

prune.train.acc <- 1-mean(tree.prune.train.pred != trainSet[,10])
```

The misclassification rate on training set of pruned tree is 25.54% which is higher than the one (11.69%) of unpruned tree.

(k)

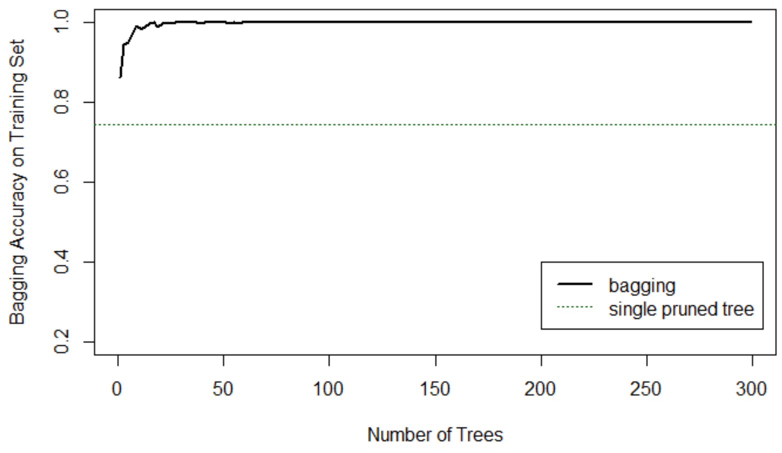
```
tree.prune.test.pred <- predict (prune.tree , testSet, type="class")
mean(tree.prune.test.pred != testSet[,10])

## [1] 0.3376623

prune.test.acc <- 1-mean(tree.prune.test.pred != testSet[,10])
```

The misclassification rate on testing set of pruned tree is 33.77% which is lower than the one (38.53%) of unpruned tree.

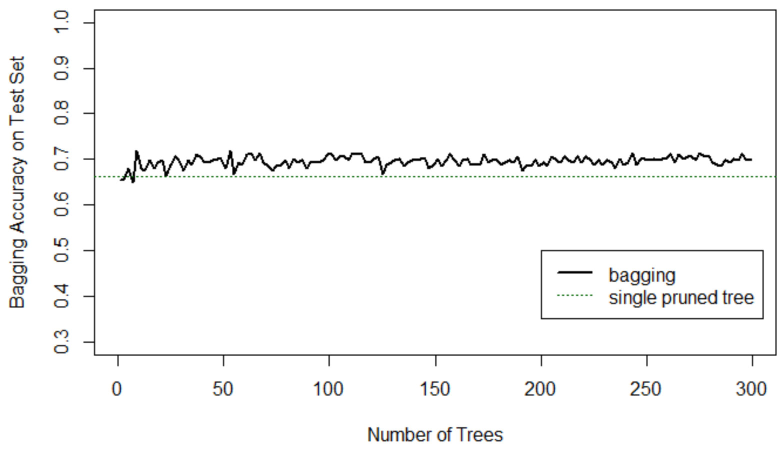
(b) Bagging

In this section, we first made plots with different values of B and the corresponding training/test set % of correct predictions. The B here ranges from 1 to 300 with gap 2.

```
B.seq <- seq(1, 300, 2)
acc.train.seq.bag <- rep(NA, length(B.seq))
acc.test.seq.bag <- rep(NA, length(B.seq))
set.seed (123)
for (i in 1:length(B.seq)) {
  bagging <- randomForest(as.factor(chd)~.,data=trainSet , mtry=9, importance =TRUE, ntre
e = B.seq[i])
  pred.bagging <- predict(bagging, trainSet)
  acc.train.seq.bag[i] <- mean(pred.bagging == trainSet[,10])

  pred.bagging <- predict(bagging ,newdata =testSet)
  acc.test.seq.bag[i] <- mean(pred.bagging == testSet[,10])
}

plot(B.seq, acc.train.seq.bag, ylim=c(0.2,1), type="l", xlab="Number of Trees", ylab="Bag
ging Accuracy on Training Set", lwd=2)
abline(h=prune.train.acc, col="darkgreen", lty=3, lwd=1)
legend(200,0.4, c("bagging", "single pruned tree"),
       lty=c(1,3), lwd=c(2,1),col=c("black", "darkgreen"))
```



```
plot(B.seq, acc.test.seq.bag, ylim=c(0.3,1), type="l", xlab="Number of Trees", ylab="Bagg
ing Accuracy on Test Set", lwd=2)
#lines(B.seq, acc.test.seq, ylim=c(0.6,0.8))
abline(h=prune.test.acc, col="darkgreen", lty=3, lwd=1)
legend(200,0.5, c("bagging", "single pruned tree"),
       lty=c(1,3), lwd=c(2,1),col=c("black", "darkgreen"))
```



From the plots above, we find that as B increases the training set accuracy improves to be 100% very quickly. However, the accuracy for test set improves slightly at the beginning and then fluctuates.

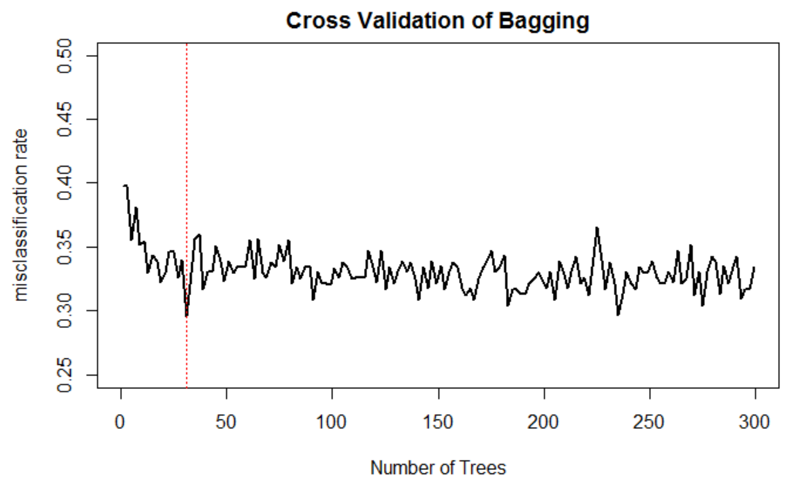
Note that there is a green dotted based line which represents the performances of single pruned tree in both plots. Thus, bagging clearly outperforms single pruned tree with reasonable value of B, which is reasonable.

How does the value of B affect the comparison between bagging and single pruned tree? For training set, value of B seems not to be very important because no matter the value of B, bagging always outperforms. However, for test set, the value of B is significant. For small value of B, single pruned tree would outperform bagging. For B that is larger than a threshold value, bagging would be better.

Next, we use cross-validation to choose the number of trees B for bagging.

```
# cross-validation to choose number of Trees on Bagging
set.seed(321)
data <- trainSet
n <- dim(trainSet)[1]
index <- 1:n
K <- 10
```

```
flds <- createFolds(index, k=K)
miscl.cv <- rep(NA, length(B.seq))

for (j in 1:length(B.seq)){
  miscl.cv.raw <- rep(NA, K)
  for (i in 1:K){
    testID <- flds[[i]]
    data.tr <- data[-testID,]
    data.test <- data[testID,]
    tree.cv <- randomForest(as.factor(chd)~.,data=data.tr , mtry=9, importance =TRUE, ntr
ee = B.seq[j])
    tree.cv.pred <- predict(tree.cv, newdata = data.test, type = "c")
    #fnr.cv.raw[i] <- sum(tree.cv.pred == "FALSE" & data.test$IsB == "TRUE")/sum(data.tes
t$IsB == "TRUE")
    miscl.cv.raw[i] <- mean(tree.cv.pred != data.test[,10])
  }
  miscl.cv[j] <- mean(miscl.cv.raw)
}

plot(B.seq, miscl.cv, type="l", lwd=2.5, ylim=c(0.25,0.5),
     xlab= "Number of Trees", ylab="misclassification rate", main="Cross Validation of Ba
gging")
abline(v=B.seq[which.min(miscl.cv)], lty=3, col="red")
```



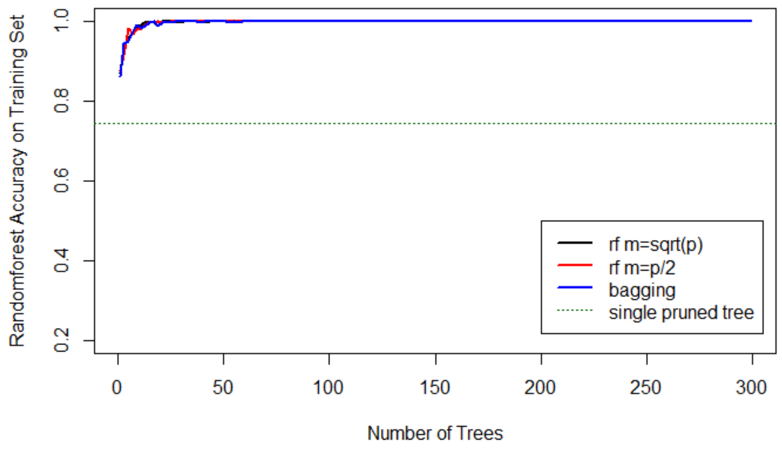
```
cat("Best B for Bagging is :", B.seq[which.min(miscl.cv)], "\n")

## Best B for Bagging is : 31

bagging.tree <- randomForest(as.factor(chd)~.,data=trainSet , mtry=9, importance =TRUE, n
tree = 31)
```

From the plot above, 31 seems to be a good choice of B for bagging. We finally use the line of code above to derive the best bagging model.

(c) RandomForest

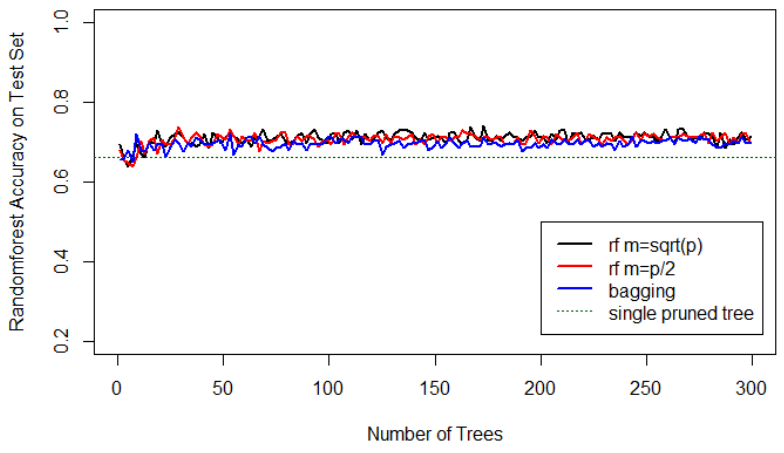
In this section, we would implement RandomForest models. The two different numbers of selected predictors are sqrt(p) and p/2.

```
acc.train.seq.sqrt <- rep(NA, length(B.seq))
acc.test.seq.sqrt <- rep(NA, length(B.seq))
acc.train.seq.by2 <- rep(NA, length(B.seq))
acc.test.seq.by2 <- rep(NA, length(B.seq))
set.seed (123)
for (i in 1:length(B.seq)) {

  rf.tree.sqrt <- randomForest(as.factor(chd)~., data=trainSet , mtry=sqrt(9), importance
 =TRUE, ntree = B.seq[i])
  rf.tree.by2 <- randomForest(as.factor(chd)~., data=trainSet , mtry=9/2, importance =TRU
E, ntree = B.seq[i])
  pred.rf.sqrt <- predict (rf.tree.sqrt, newdata = trainSet)
  acc.train.seq.sqrt[i] <- mean(pred.rf.sqrt ==trainSet[,10])

  pred.rf.sqrt <- predict (rf.tree.sqrt ,newdata = testSet)
  acc.test.seq.sqrt[i] <- mean(pred.rf.sqrt == testSet[,10])

  pred.rf.by2 <- predict(rf.tree.by2, trainSet)
  acc.train.seq.by2[i] <- mean(pred.rf.by2 == trainSet[,10])


  pred.rf.by2 <- predict (rf.tree.by2 ,newdata =testSet)
  acc.test.seq.by2[i] <- mean(pred.rf.by2 == testSet[,10])
}

# black: rf with sqrt(p) features
# red: rf with p/2 features
# blue: bagging
# purple: single pruned tree
plot(B.seq, acc.train.seq.sqrt, ylim=c(0.2,1), type="l", xlab="Number of Trees", ylab="Ra
ndomforest Accuracy on Training Set", lwd=2)
lines(B.seq, acc.train.seq.by2, col="red", lwd=2)
lines(B.seq, acc.train.seq.bag, col="blue", lwd=2)
abline(h=prune.train.acc, col="darkgreen", lty=3, lwd=1)
legend(200,0.5, c("rf m=sqrt(p)", "rf m=p/2", "bagging", "single pruned tree"),
       lty=c(1,1,1,3), lwd=c(2,2,2,1),col=c("black","red", "blue", "darkgreen"))
```



```
plot(B.seq, acc.test.seq.sqrt, ylim=c(0.2,1), type="l", xlab="Number of Trees", ylab="Ran
domforest Accuracy on Test Set", lwd=2)
lines(B.seq, acc.test.seq.by2, col="red", lwd=2)
```

```r
lines(B.seq, acc.test.seq.bag, col="blue", lwd=2)
abline(h=prune.test.acc, col="darkgreen", lty=3, lwd=1)
legend(200,0.5, c("rf m=sqrt(p)", "rf m=p/2", "bagging", "single pruned tree"),
       lty=c(1,1,1,3), lwd=c(2,2,2,1),col=c("black","red", "blue", "darkgreen"))
```



The conclusion here is very similar as that in Bagging section.

From the plots above, we find that as B increases the training set accuracy improves to be 100% very quickly. But the accuracy for test set improves slightly at the beginning and then fluctuates.

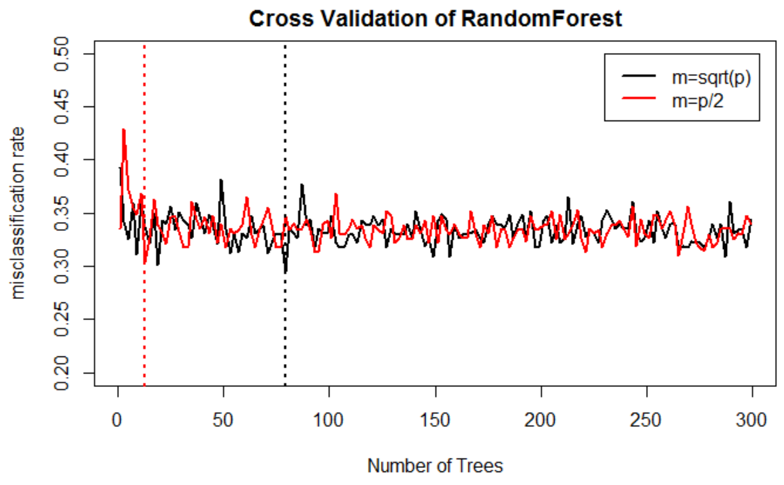
Note that there is a green dotted based line which represents the performances of single pruned tree in both plots. There is also a blue line representing the performance of bagging. Thus, RandomForest models clearly outperform single pruned tree with reasonable value of B.

Plus, the performance of RandomForest and Bagging is very similar regardless of the B values. Remember that Bagging is just a special case of RandomForest and we only have 9 features for this dataset (randomforest models take mtry to be 3 and 4, Bagging takes it to be 9). The equally matched performances make sense here.

The value of B does influence the performance of RandomForest compared with single pruned tree. For training set, value of B seems not to be very important because no matter the value of B, RandomForest always outperforms. However, for test set, the value of B is significant. For small value of B, single pruned tree would outperform RandomForest. For B that is larger than a threshold value, RandomForest would be better.

Next, we use cross-validation to choose the number of trees B for RandomForest.

```r
# 10-fold cross-validation to choose number of Trees on RandomForest
set.seed(321)
data <- trainSet
n <- dim(trainSet)[1]
index <- 1:n
K <- 10
flds <- createFolds(index, k=K)
miscl.sqrt.cv <- rep(NA, length(B.seq))
miscl.by2.cv <- rep(NA, length(B.seq))

for (j in 1:length(B.seq)){
  miscl.sqrt.cv.raw <- rep(NA, K)
  miscl.by2.cv.raw <- rep(NA, K)
  for (i in 1:K){
    testID <- flds[[i]]
    data.tr <- data[-testID,]
    data.test <- data[testID,]
    tree.cv <- randomForest(as.factor(chd)~.,data=data.tr , mtry=sqrt(9), importance =FAL
SE, ntree = B.seq[j])
    tree.cv.pred <- predict(tree.cv, newdata = data.test, type = "c")
    miscl.sqrt.cv.raw[i] <- mean(tree.cv.pred != data.test[,10])

    tree.cv <- randomForest(as.factor(chd)~.,data=data.tr , mtry=9/2, importance =FALSE,
ntree = B.seq[j])
    tree.cv.pred <- predict(tree.cv, newdata = data.test, type = "c")
    miscl.by2.cv.raw[i] <- mean(tree.cv.pred != data.test[,10])
  }
  miscl.sqrt.cv[j] <- mean(miscl.sqrt.cv.raw)
  miscl.by2.cv[j] <- mean(miscl.by2.cv.raw)
}

plot(B.seq, miscl.sqrt.cv, type="l", lwd=2.5, ylim=c(0.2,0.5),
     xlab= "Number of Trees", ylab="misclassification rate", main="Cross Validation of Ra
ndomForest")
lines(B.seq, miscl.by2.cv, lwd=2.5, col="red")
abline(v=B.seq[which.min(miscl.sqrt.cv)], lty=3, col="black", lwd=2)
abline(v=B.seq[which.min(miscl.by2.cv)], lty=3, col="red", lwd=2)
legend(230,0.5, c("m=sqrt(p)", "m=p/2"),
       lty=c(1,1), lwd=c(2,2),col=c("black", "red"))
```



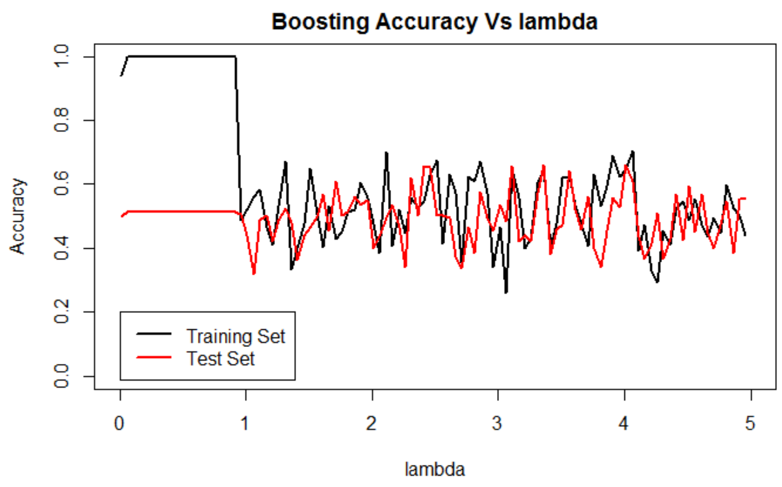
```
cat("Best B for RandomForest with m=sqrt(p) is :", B.seq[which.min(miscl.sqrt.cv)], "\n")

## Best B for RandomForest with m=sqrt(p) is : 79

cat("Best B for RandomForest with m=p/2 is :", B.seq[which.min(miscl.by2.cv)], "\n")

## Best B for RandomForest with m=p/2 is : 13

rf.sqrt.tree <- randomForest(as.factor(chd)~.,data=trainSet , mtry=sqrt(9), importance =T
RUE, ntree = 79)
rf.by2.tree <- randomForest(as.factor(chd)~.,data=trainSet , mtry=9/2, importance =TRUE,
ntree = 13)
```

From the plot above 79 and 13 are the proper values of B for RandomForest with m=sqrt(p) and RandomForest with m=p/2. We finally get the best RandomForest models.

(d) Boosting

In this section, we would build boosting model.

```
set.seed(123)
grid <- seq(0.01, 5, 0.05)
acc.train.seq.boost <- rep(NA, length(grid))
acc.test.seq.boost <- rep(NA, length(grid))
for (i in 1:length(grid)) {
  boost.tree <- gbm(chd~., data=trainSet, distribution="bernoulli", n.trees =1000, intera
ction.depth=4, shrinkage=grid[i])
  boost.tree.train.pred <- predict(boost.tree, data=trainSet, n.trees=1000, type="respons
e") > 0.5
  acc.train.seq.boost[i] <- mean(boost.tree.train.pred == trainSet$chd)

  boost.tree.test.pred <- predict(boost.tree, data=testSet, n.trees=1000, type="response")
 > 0.5
  acc.test.seq.boost[i] <- mean(boost.tree.test.pred == testSet$chd)
}

# black: train
# red: test
plot(grid, acc.train.seq.boost, type="l", xlim=c(0,5), ylim=c(0,1), xlab="lambda", ylab="
Accuracy", lwd=2, main="Boosting Accuracy Vs lambda")
lines(grid, acc.test.seq.boost, type="l", col="red", lwd=2)

legend(0,0.2, c("Training Set", "Test Set"),
       lty=c(1,1), lwd=c(2,2),col=c("black", "red"))
```



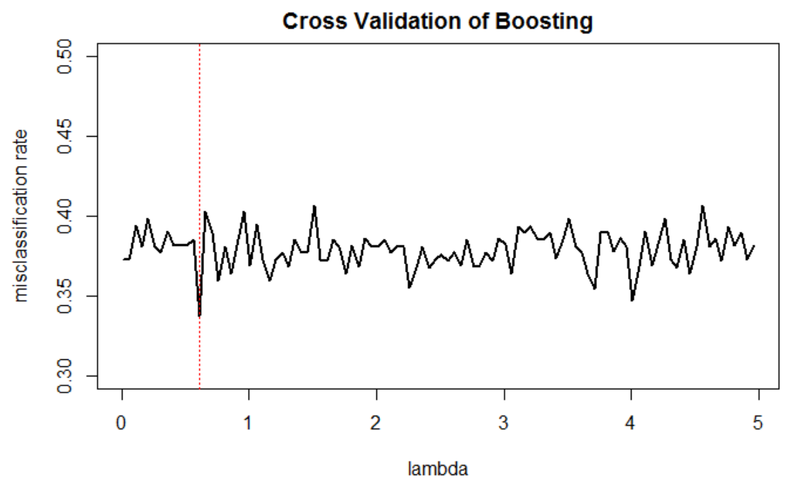
Here I plot d(i) and d(ii) together.

We see that the accuracy on training set starts from nearly 1.0 and starts fluctuating near lambda = 1. The accuracy on test set follows similar pattern except that it starts around 0.5. It can be conclude that with small values of lambda, boosting models are badly overfitting.

Next, we use cross-validation to choose proper lambda for Boosting.

```
# 10-fold cross-validation to choose lambda on Boosting
set.seed(321)
data <- trainSet
n <- dim(trainSet)[1]
index <- 1:n
```

```
K <- 10
flds <- createFolds(index, k=K)
miscl.cv <- rep(NA, length(grid))

for (j in 1:length(grid)){
  miscl.cv.raw <- rep(NA, K)
  for (i in 1:K){
    testID <- flds[[i]]
    data.tr <- data[-testID,]
    data.test <- data[testID,]
    tree.cv <- gbm(chd~., data=data.tr, distribution="bernoulli", n.trees =1000, interact
ion.depth=4, shrinkage=grid[i])
    tree.cv.pred <- predict(tree.cv, newdata = data.test, type = "response", n.trees=1000)
 > 0.5
    #fnr.cv.raw[i] <- sum(tree.cv.pred == "FALSE" & data.test$IsB == "TRUE")/sum(data.tes
t$IsB == "TRUE")
    miscl.cv.raw[i] <- mean(tree.cv.pred != data.test[,10])
  }
  miscl.cv[j] <- mean(miscl.cv.raw)
}

plot(grid, miscl.cv, type="l", lwd=2, ylim=c(0.3,0.5), xlab="lambda", ylab="misclassifica
tion rate",
     main="Cross Validation of Boosting")
abline(v=grid[which.min(miscl.cv)], col="red", lty=3)
```



```
cat("Best lambda for Boosting with B=1000 is :", grid[which.min(miscl.cv)], "\n")

## Best lambda for Boosting with B=1000 is: 0.61
```

From the plot above, we see that the best lambda there is 0.61. Finally we have the best boosting model.

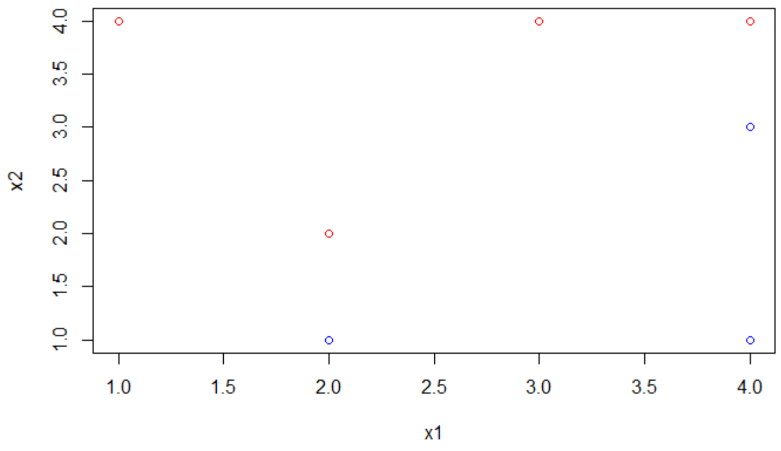
```
boosting.tree <- gbm(chd~., data=trainSet, distribution="bernoulli", n.trees=1000, intera
ction.depth=4, shrinkage=0.61)
summary(boosting.tree)

##                     var   rel.inf
## tobacco       tobacco 17.864434
## age               age 16.165435
## ldl               ldl 12.700520
## alcohol       alcohol 11.161555
## adiposity adiposity 11.152392
## typea         typea 10.107254
## sbp             sbp  9.931618
## obesity     obesity  7.762373
## famhist     famhist  3.154419
```

Tobacco and age are the most important 2 predictors in the boosting model. They are not the same with the ones in single pruned tree. For single pruned tree, the most important 2 are tobacco and famhist. Tobacco is still important in boosting model, however, famhist is the least important here.

5. JWHT Chapter 9, Problem 3.

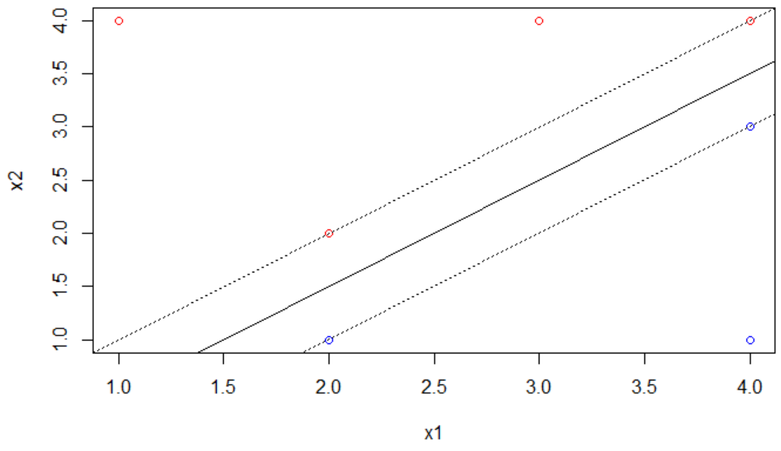
(a)

```
x1 <- c(3,2,4,1,2,4,4)
x2 <- c(4,2,4,4,1,3,1)
y <- c("red", "red","red","red","blue","blue","blue")
plot(x1,x2, col=y)
```



(b)

```
plot(x1,x2, col=y)
abline(a=-0.5, b=1)
abline(a=0, b=1, lty=3)
abline(a=-1, b=1, lty=3)
```



The hyperplane here is :

$$0.5 - x_1 + x_2 = 0$$

(c)

The classification rule is:

Red if $0.5 - x_1 + x_2 > 0$

Blue if $0.5 - x_1 + x_2 < 0$

(d)

The margin here is the perpendicular distance from one dotted line to the solid line.

And the value of the margin is $\frac{\sqrt{2}}{4}$.

(e)

The support vectors are:

(2,1),(2,2),(4,3),(4,4)

(f)

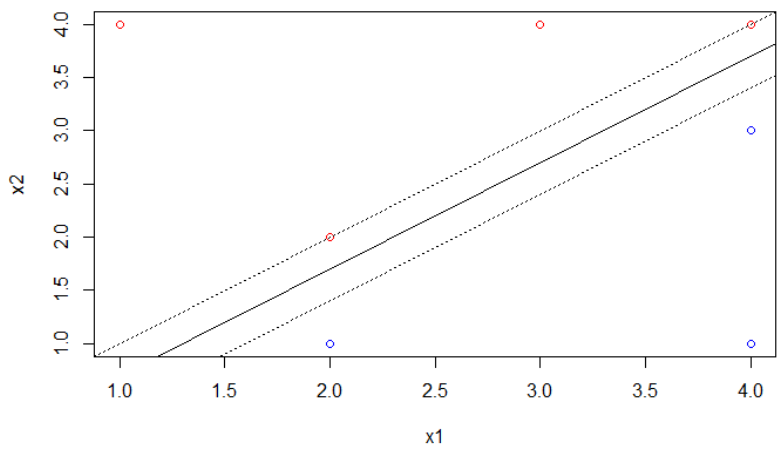
For SVM, the points that would affect the hyperplane are the support vectors.

Clearly the seventh point (4,1) is not a support vector and far from the hyperplane.

Thus, a slight movement of the seventh observation would not affect the maximal margin hyperplane.

(g)

```r
plot(x1,x2, col=y)
abline(a=-0.3, b=1)
abline(a=0, b=1, lty=3)
abline(a=-0.6, b=1, lty=3)
```

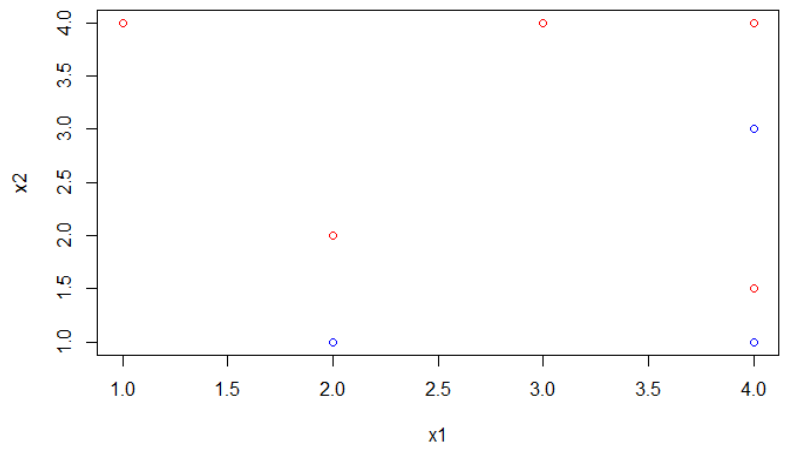


There are infinite hyperplanes that are not the optimal ones. Here the one is $0.3 - x_1 + x_2 = 0$.

(h)

```r
x1 <- c(3,2,4,1,2,4,4,4)
x2 <- c(4,2,4,4,1,3,1,1.5)
```

```
y <- c("red", "red","red","red","blue","blue","blue", "red")
plot(x1,x2, col=y)
```



The two classes won't be linearly separable with an additional red point (4, 1.5).

6. SVM and Neural Networks

(a) Train support vector machine the "South African Heart Disease" dataset, and evaluate its performance on the validation set.

In this section, we train a SVM model with radial kernel.

To not be computational expensive, only a small scale of values for tuning parameters would be tested by cross-validation. For C, these values are 0.01, 0.1, 1, 10, 100, 1000. For gamma, these values are 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2, 3, 4.

```
svmdata <- Mydata
svmdata$chd[svmdata$chd == 0] <- -1
svm.trainSet <- svmdata[train,]
svm.testSet <- svmdata[-train,]

set.seed(123)
tune.out <- tune(svm , as.factor(chd)~., data=svm.trainSet, kernel ="radial",
                 ranges =list(cost=c(0.01, 0.1, 1, 10 ,100 ,1000), gamma=c(0.005, 0.01, 0.
05, 0.1, 0.5, 1,2,3,4)))
svm.best <- tune.out$best.model
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10  0.01
##
## - best performance: 0.2731884
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-02 0.005 0.3554348 0.08544121
## 2  1e-01 0.005 0.3554348 0.08544121
## 3  1e+00 0.005 0.3382246 0.09257879
## 4  1e+01 0.005 0.2903986 0.09032997
## 5  1e+02 0.005 0.2731884 0.09710325
## 6  1e+03 0.005 0.3594203 0.07723428
## 7  1e-02 0.010 0.3554348 0.08544121
## 8  1e-01 0.010 0.3554348 0.08544121
## 9  1e+00 0.010 0.3295290 0.08877613
## 10 1e+01 0.010 0.2731884 0.09924274
## omit several lines here…
## 50 1e-01 4.000 0.3554348 0.08544121
## 51 1e+00 4.000 0.3554348 0.08544121
## 52 1e+01 4.000 0.3554348 0.08544121
## 53 1e+02 4.000 0.3554348 0.08544121
## 54 1e+03 4.000 0.3554348 0.08544121
```

```
mean(predict(svm.best, svm.testSet) == svm.testSet$chd)
```

```
## [1] 0.7316017
```

The best SVM model fitted with C=10 and gamma=0.01.

The accuracy of this SVM model on test set is 73.16% while the AUC is 0.7795 (codes for AUC are in appendix, not shown here).

(b) Train neural networks on the training set of the "South African Heart Disease" dataset, and evaluate its performance on the validation set.

In this section, we train an ANN model.

We selected nnet to do the task. There are lots of parameters that could be tuned like size, maxit and decay. However, to be simple, we only considered a small scale of values for “size” to do cross-validation. 300 and 5e-4 are the standard values for the parameters maxit and decay.

```
set.seed(666)
data <- trainSet
n <- dim(trainSet)[1]
index <- 1:n
K <- 10
flds <- createFolds(index, k=K)
sizes <- seq(1, 20, 1)
miscl.cv <- rep(NA, length(sizes))

for (j in 1:length(sizes)){
  miscl.cv.raw <- rep(NA, K)
  for (i in 1:K){
    testID <- flds[[i]]
    data.tr <- data[-testID,]
    ideal <- class.ind(data.tr$chd)
    data.test <- data[testID,]
    ann.cv <- nnet(data.tr[,-10], ideal, size=sizes[j], softmax=TRUE, maxit = 300, decay
= 5e-4)
    ann.cv.pred <- predict(ann.cv, data.test[,-10], type = "class")
    miscl.cv.raw[i] <- mean(ann.cv.pred != data.test[,10])
  }
  miscl.cv[j] <- mean(miscl.cv.raw)
}

plot(sizes, miscl.cv, type="l", xlab="size", ylab="misclassification rate", ylim=c(0.3, 0.
5)
     , main="Cross-validation of ANN", lwd=2)
abline(v=sizes[which.min(miscl.cv)], col="red", lty=3)

 cat("The best size for ANN is :", sizes[which.min(miscl.cv)],"\n")

## The best size for ANN is : 2
```

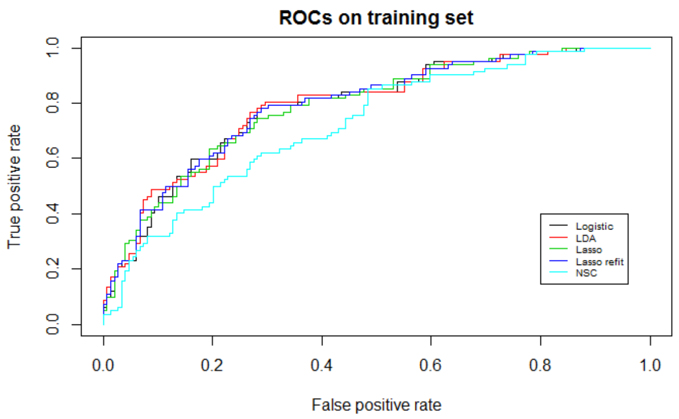
From the cross-validation plot above, the best size for ANN is 2.

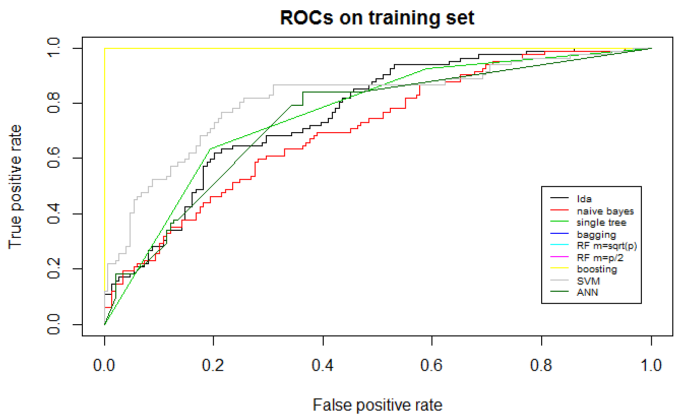
We last get the best ANN model and evaluated the performance on test set.

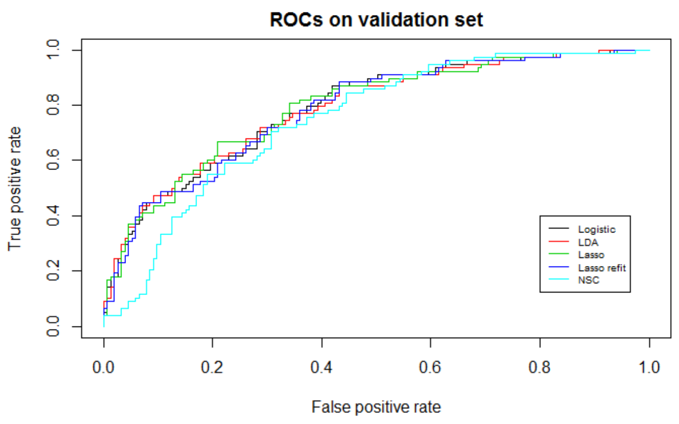
```
set.seed(345)
ideal <- class.ind(trainSet$chd)
ANN <- nnet(trainSet[,-10], ideal, size=2, softmax=TRUE, maxit = 300, decay = 5e-4)

mean(predict(ANN, testSet[, -10], type="class") == testSet$chd)

## [1] 0.6363636
```

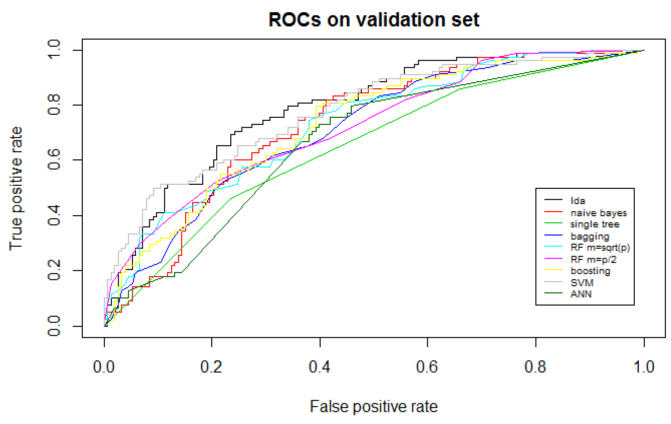
The accuracy of this ANN model on test set is 63.64% while the AUC is 0.6680 (codes for AUC are in appendix, not shown here).

7. Summarize the classification results obtained for the "South African Heart Disease" in homework 3 and 4. Which method performed better, which performed worse? Discuss the possible reasons.









AUC	LDA	Na ïve Bayes	Single Tree	Bagging	Rf.sqrt	Rf.by2	Boosting	SVM	ANN	Logistic	Lasso	Lasso refit	NSC
Training	0.7629	0.7071	0.7652	1	1	1	1	0.8144	0.7450	0.7910	0.7878	0.7923	0.7234
Validation	0.7901	0.7323	0.6507	0.7109	0.7330	0.7247	0.7316	0.7795	0.6680	0.7894	0.7921	0.7847	0.7476

Here we make the plots and table for the performance of models we made in homework 3 (left) and homework 4 (right).

The lasso model is the best regarding the performance on validation set followed by LDA, Logistic and Lasso refit. SVM comes next and stands alone since it is not a probabilistic method. NSC, Na ïve Bayes and these tree-based methods (except for Single Tree) could be clustered into next category. Finally, ANN model and the worst model Single Tree come.

We tried several probabilistic models to capture the nonlinearity among the features like ANN, tree-based methods. However, these models turn out to be less accurate. We observe that the difference between the performance of training and validation set for ANN is not significant. So overfitting is less likely to be the issue here. Moreover, tree-based methods suffer a lot from overfitting. Most of them have 100% accuracy on training set while holding much lower accuracy on validation set. It may come from the fact that the numbers of trees in these models are too big even though they are chosen by cross-validation. Bagging, Rf.sqrt and Rf.by2 have similar performance because the number of features here is 9, which is relatively small. A single classification tree is the worst model here, which is not surprising. This method has great uncertainty which means we would likely to have a totally different model if running the script again (without set.seed()).

Our self-made “idiot Bayes” is comparable with those tree-based methods, but it has the advantage that it is not overfitting for this task. Of course for this problem, the assumption of Na ïve Bayes is violated. From my point, the predictor age could be correlated with many other predictors. A slightly better model is NSC. NSC is not that bad even though the number of features (9) is relatively small compared with the number of observations (462), which means it is not the best situation to apply NSC.

SVM is good towards this dataset and it is a top 5 classifier. It is slightly overfitting, which is acceptable. Besides, its performance towards training set is the best except for these tree-based methods. SVM would work better if the dataset is balanced or with little noise. This dataset seems belong to this situation.

LDA performs surprisingly well here. The reason behind this is that the multivariate normality and equality of covariance matrices assumptions are not seriously violated. At least the assumptions here are more suitable compared to that in Na ïve Bayes.

Finally, we can discuss the best models. They are models based on Logistic Regression. That is to say, linear models are sufficed here. It is not necessary for us to transform the features to higher dimensional space. According to the AUC values, the one with bias (Lasso) outperforms. So introducing bias here seems to be a good choice and thus our model would be more stable.

To draw a conclusion, there is no free lunch. One would always do model competing given different datasets.

Appendix

Naïve.bayes classifier

```r
# naive bayes classifier
# this function receives a certain value of lambda, predictors and
# the train predictors that are used to estimate density
# it returns the raw scores of NBC
naive.bayes.cl <- function(lambda, predictors, trainPredictors) {
  n <- dim(predictors)[1]
  naiveBayes.raw <- rep(NA, n)
  for (i in 1:n) {
    # data processing
    data.test.predictors <- predictors[i,]
    ones <- which(trainResponse %in% c(1))
    response.ones <- which(trainResponse %in% c(1))
    len.ones <- length(response.ones)
    len.zeros <- length(trainResponse) - len.ones

    # KDE
    density <- dnorm(abs(data.test.predictors - trainPredictors)/lambda)
    density.ones <- density[ones,]
    density.zeros <- density[-ones,]
    density.one <- prod(apply(density.ones, 2, sum)/len.ones)
    density.zero <- prod(apply(density.zeros, 2, sum)/len.zeros)

    # prior estimated probability
    prior.ones <- len.ones/length(trainResponse)
    prior.zeros <- len.zeros/length(trainResponse)

    # posterior probability
    prob.one <- prior.ones * density.one / (prior.ones * density.one + prior.zeros *
density.zero)
    naiveBayes.raw[i] <- prob.one
  }
  return(naiveBayes.raw)
}
```

Leave-one-out Cross-validation for NBC

```r
# leave-one-out cross validation
# This function receives a lambda sequence, predictors and response
# returns a lambda selected by cross-validation
# also plots accuracy vs lambda
naive.bayes.cv <- function(lambda.seq, predictors, response){
  miscl <- rep(NA, length(lambda.seq))
  for (k in 1:length(lambda.seq)) {
    lambda <- lambda.seq[k]
    n <- length(response)
    naiveBayes.raw <- rep(NA, n)

    # iteratively choose one as test data every time
    for (i in 1:n) {
      data.tr.predictors <- predictors[-i,]
      data.tr.response <- response[-i]
      data.test.predictors <- predictors[i,]
      data.test.response <- response[i]

      # select ones and zeros
      ones <- which(data.tr.response %in% c(1))
      response.ones <- which(response %in% c(1))
      len.ones <- length(response.ones)
      len.zeros <- n - len.ones

      # KDE
      density <- dnorm(abs(data.test.predictors - data.tr.predictors)/lambda)
      density.ones <- density[ones,]
      density.zeros <- density[-ones,]

      # prior estimated probability
      prior.ones <- len.ones/n
      prior.zeros <- len.zeros/n

      # KDE for different classes
      density.one <- prod(apply(density.ones, 2, sum)/len.ones)
      density.zero <- prod(apply(density.zeros, 2, sum)/len.zeros)

      # posterior for class one
      prob.one <- prior.ones * density.one / (prior.ones * density.one + prior.zeros * de
nsity.zero)
      naiveBayes.raw[i] <- prob.one
    }
    # assign the class based on raw scores
    cl <- sapply(naiveBayes.raw, function(x) x>0.5)
    miscl[k] <- 1-mean(cl == response)
  }
  plot(lambda.seq, miscl, ylim=c(0.25, 0.5), ylab="misclassification rate", xlab="lambda",
 type="l")
  abline(v=lambda.seq[which.min(miscl)], col="red", lty=3)
  cat("Best lambda is :", lambda.seq[which.min(miscl)],"\n")
}
```

ROCs on training set

```r
# ROCs on the training set
# lda
scores <- predict(lda.fit, newdata= trainPredictors)$posterior[,2]
pred <- prediction( scores, labels= trainResponse )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 1, main="ROCs on training set")
# print out the area under the curve
lda.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# naive bayes
pred <- prediction(naiveBayes.train.scores, labels=trainResponse)
perf <- performance(pred, "tpr", "fpr")

# plot the ROC curve
plot(perf, col= 2, add=T)
# print out the area under the curve
naive.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# single pruned tree
scores <- predict(prune.tree, newdata= trainSet[,-10], type="vector")[,2]
pred <- prediction(scores, labels= trainSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 3, add=T)
# AUC
single.tree.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# bagging
pred.bagging <- predict(bagging.tree ,newdata =trainSet, type="prob")[,2]
pred <- prediction(pred.bagging, labels= trainSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 4, add=T)

# AUC
bagging.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# random forest
pred.rf.sqrt <- predict(rf.sqrt.tree ,newdata =trainSet, type="prob")[,2]
pred <- prediction(pred.rf.sqrt, labels= trainSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 5, add=T)
# AUC for rf.sqrt
rf.sqrt.train <- unlist(attributes(performance(pred, "auc"))$y.values)

pred.rf.by2 <- predict(rf.by2.tree ,newdata =trainSet, type="prob")[,2]
pred <- prediction(pred.rf.by2, labels= trainSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 6, add=T)
# AUC for rf.by2
rf.by2.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# boosting
boosting.pred <- predict(boosting.tree, trainSet, type='response', n.trees=1000)
pred <- prediction(boosting.pred, labels= trainSet$chd )
```

```
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 7, add=T)

# AUC for boosting
boosting.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# SVM
scores <- attributes(predict(svm.best, svm.trainSet, decision.values =TRUE))$decision.val
ues
pred <- prediction(scores, svm.trainSet$chd)
perf <- performance(pred, "tpr", "fpr")
plot(perf, col = 8, add=T)
# AUC for svm
svm.train <- unlist(attributes(performance(pred, "auc"))$y.values)

# ANN
scores <- predict(ANN, trainSet[,-10], type="raw")[,2]
pred <- prediction(scores, svm.trainSet$chd)
perf <- performance(pred, "tpr", "fpr")
plot(perf, col = "darkgreen", add=T)
# AUC for ANN


ANN.train <- unlist(attributes(performance(pred, "auc"))$y.values)
legend(0.8,0.5, c("lda", "naive bayes", "single tree", "bagging", "RF m=sqrt(p)",
                  "RF m=p/2", "boosting", "SVM", "ANN") ,col=c(1:8, "darkgreen"), lty=1,
cex=0.6)
```

ROCs on validation set

```r
# ROCs on the validation set
# lda
#scores <- predict(lda.fit, newdata= trainPredictors)$posterior[,2]
scores <- predict(lda.fit, newdata = testPredictors)$posterior[,2]
pred <- prediction( scores, labels= testResponse )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 1, main="ROCs on validation set")


# naive bayes
pred <- prediction(naiveBayes.test.scores, labels=testResponse)
perf <- performance(pred, "tpr", "fpr")

# plot the ROC curve
plot(perf, col= 2, add=T)

# single pruned tree
scores <- predict(prune.tree, newdata= testSet[,-10], type="vector")[,2]
pred <- prediction(scores, labels= testSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 3, add=T)
# AUC
single.tree.test <- unlist(attributes(performance(pred, "auc"))$y.values)

# bagging
pred.bagging <- predict(bagging.tree ,newdata =testSet, type="prob")[,2]
pred <- prediction(pred.bagging, labels= testSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 4, add=T)
# AUC
bagging.test <- unlist(attributes(performance(pred, "auc"))$y.values)


# random forest
pred.rf.sqrt <- predict(rf.sqrt.tree ,newdata =testSet, type="prob")[,2]
pred <- prediction(pred.rf.sqrt, labels= testSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 5, add=T)
# AUC for rf.sqrt
rf.sqrt.test <- unlist(attributes(performance(pred, "auc"))$y.values)

#rf.by2.tree <- randomForest(as.factor(chd)~.,data=trainSet , mtry=9/2, importance =TRUE,
 ntree = 13)
pred.rf.by2 <- predict(rf.by2.tree ,newdata =testSet, type="prob")[,2]
pred <- prediction(pred.rf.by2, labels= testSet$chd )
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 6, add=T)
# AUC for rf.by2
rf.by2.test <- unlist(attributes(performance(pred, "auc"))$y.values)

# boosting
boosting.pred <- predict(boosting.tree, testSet, type='response', n.trees=1000)
pred <- prediction(boosting.pred, labels= testSet$chd )
```

```r
perf <- performance(pred, "tpr", "fpr")
plot(perf, col= 7, add=T)

# AUC for boosting
boosting.test <- unlist(attributes(performance(pred, "auc"))$y.values)

# SVM
scores <- attributes(predict(svm.best, svm.testSet, decision.values =TRUE))$decision.valu
es
pred <- prediction(scores, svm.testSet$chd)
perf <- performance(pred, "tpr", "fpr")
plot(perf, col = 8, add=T)
# AUC for svm
svm.test <- unlist(attributes(performance(pred, "auc"))$y.values)

# ANN
scores <- predict(ANN, testSet[,-10], type="raw")[,2]
pred <- prediction(scores, svm.testSet$chd)
perf <- performance(pred, "tpr", "fpr")
plot(perf, col = "darkgreen", add=T)
# AUC for ANN

ANN.test <- unlist(attributes(performance(pred, "auc"))$y.values)
legend(0.8,0.5, c("lda", "naive bayes", "single tree", "bagging", "RF m=sqrt(p)",
                  "RF m=p/2", "boosting", "SVM", "ANN") ,col=c(1:8, "darkgreen"), lty=1,
cex=0.6)
```