Homework4

Shiyu Wang

April 15, 2017

Problem 1

(a)

For x<=ξ, f1(x) has coefficients
a1=β0,b1=β1,c1=β2,d1=β3a1=β0,b1=β1,c1=β2,d1=β3.

(b)

For x>ξ, f(x) has the form of:
β0+β1x+β2x^2+β3^3+β4(x−ξ)^3
=β0+β1x+β2x^2+β3x^3+β4(x^3−3x^2ξ+3xξ^2−ξ^3)
=(β0−β4ξ^3)+(β1+3β4ξ^2)x+(β2−3β4ξ)x^2+(β3+β4)x^3

So
a2=β0−β4ξ^3
b2=β1+3β4ξ^2
c2=β2−3β4ξ
d2=β3+β4

(c)

f1(ξ)=β0+β1ξ+β2ξ^2+β3ξ3
f2(ξ)=(β0−β4ξ^3)+(β1+3β4ξ^2)ξ+(β2−3β4ξ)ξ^2+(β3+β4)ξ^3
=β0−β4ξ3+β1ξ+3β4ξ3+β2ξ2−3β4ξ3+β3ξ3+β4ξ3
=β0+β1ξ+β2ξ^2+3β4ξ^3−3β4ξ^3+β3ξ^3+β4ξ^3−β4ξ^3
=β0+β1ξ+β2ξ^2+β3

(d)

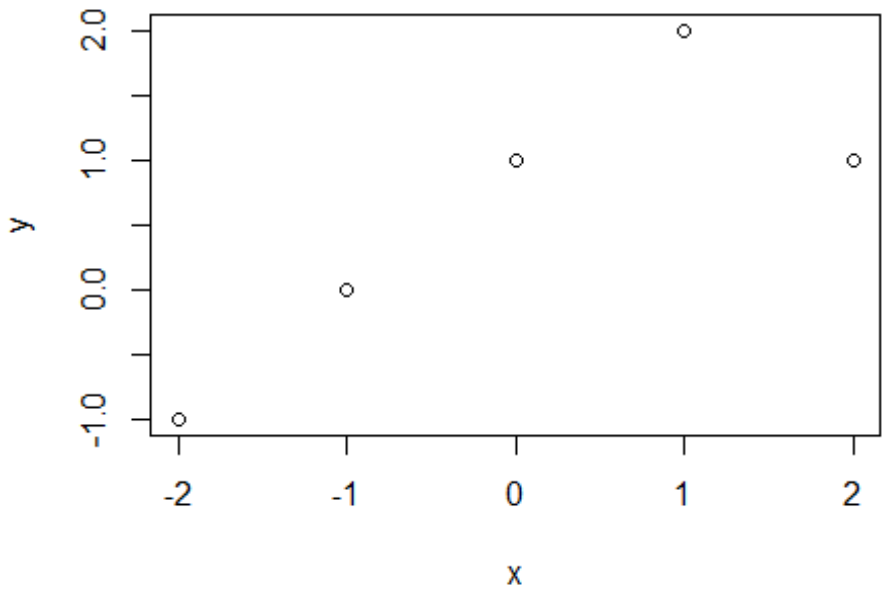
f’(x)=b1+2c1x+3d1x^2
f’1(ξ)=β1+2β2ξ+3β3ξ^2
f’2(ξ)=β1+3β4ξ^2+2(β2−3β4ξ)ξ+3(β3+β4)ξ^2
=β1+3β4ξ^2+2β2ξ−6β4ξ^2+3β3ξ^2+3β4ξ^2
=β1+2β2ξ+3β3ξ^2+3β4ξ^2+3β4ξ^2−6β4ξ^2
=β1+2β2ξ+3β3ξ^2

(e)

f’’(x)=2c1+6d1x
f’’1(ξ)=2β2+6β3ξ
f’’2(ξ)=2(β2−3β4ξ)+6(β3+β4)ξ
=2β2+6β3ξ

Problem 2

```
x = -2:2
y = 1 + x + -2 * (x-1)^2 * I(x>1)
plot(x, y)
```



Problem 3

(a)

```
Code is provided at the end of the report in appendix with python 2.7
```

(b)

```
Accuracy = 68.06
Details of process can be found in appendix code
```

(c)

```
Accuracy  66.20
Details of process can be found in appendix code
LDA and Naive Bayes have similar accuracy because LDA is also based on Naive
Bayes to separate
```

problem 4

(a)

```
# read the data
SAheart <- read.table("http://www-
stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data",
sep=",",head=T,row.names=1)
```

```r
# set the seed
set.seed(1)

library(ISLR)

## Warning: package 'ISLR' was built under R version 3.3.3

attach(SAheart)
Present = ifelse(chd==1, "Yes", "No")
SAheart = data.frame(SAheart, Present)
train <- sample(1:462,231)
valid <- c(1:462)[-train]
SAheart$famhist <- as.character(SAheart$famhist)
SAheart$famhist[which(SAheart$famhist=="Present")] <- "1"
SAheart$famhist[which(SAheart$famhist=="Absent")] <- "0"
SAheart$famhist <- as.numeric(SAheart$famhist)
# create training and validation sets
SAheart_train <- SAheart[train,]
SAheart_train <- SAheart_train[,-10]
SAheart_valid <- SAheart[valid,]
SAheart_valid <- SAheart_valid[,-10]
```

(b)

```r
# fit a tree to the training data
# building tree model with all predictors
# It has 24 terminal nodes.
# Training error is 0.130
library(tree)

## Warning: package 'tree' was built under R version 3.3.3

treeModel = tree(Present~., SAheart_train)
summary(treeModel)

##
## Classification tree:
## tree(formula = Present ~ ., data = SAheart_train)
## Number of terminal nodes:  24
## Residual mean deviance:  0.6281 = 130 / 207
## Misclassification error rate: 0.1299 = 30 / 231
```

c)

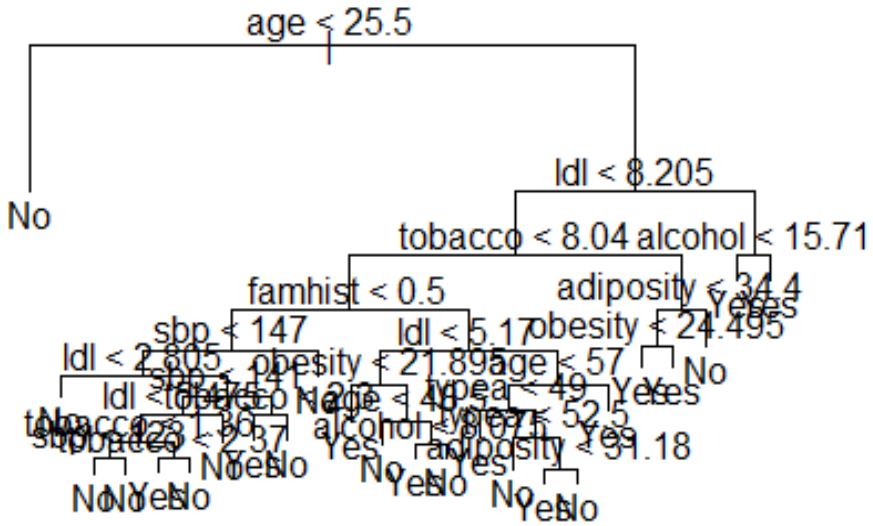
```r
treeModel

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##    1) root 231 308.000 No ( 0.61472 0.38528 )
##      2) age < 25.5 32   0.000 No ( 1.00000 0.00000 ) *
##      3) age > 25.5 199 273.700 No ( 0.55276 0.44724 )
##        6) ldl < 8.205 182 245.900 No ( 0.59341 0.40659 )
```

```
##         12) tobacco < 8.04 150 193.600 No ( 0.65333 0.34667 )
##           24) famhist < 0.5 81  88.250 No ( 0.76543 0.23457 )
##             48) sbp < 147 62  74.700 No ( 0.70968 0.29032 )
##               96) ldl < 2.805 7   0.000 No ( 1.00000 0.00000 ) *
##               97) ldl > 2.805 55  69.550 No ( 0.67273 0.32727 )
##                194) sbp < 141 43  48.900 No ( 0.74419 0.25581 )
##                  388) ldl < 5.475 34  42.810 No ( 0.67647 0.32353 )
##                    776) tobacco < 1.36 14  11.480 No ( 0.85714 0.14286 )
##                     1552) sbp < 123 6   7.638 No ( 0.66667 0.33333 ) *
##                     1553) sbp > 123 8   0.000 No ( 1.00000 0.00000 ) *
##                    777) tobacco > 1.36 20  27.530 No ( 0.55000 0.45000 )
##                     1554) tobacco < 2.37 5   5.004 Yes ( 0.20000 0.80000 )
*
##                     1555) tobacco > 2.37 15  19.100 No ( 0.66667 0.33333 )
*
##                  389) ldl > 5.475 9   0.000 No ( 1.00000 0.00000 ) *
##                195) sbp > 141 12  16.300 Yes ( 0.41667 0.58333 )
##                  390) tobacco < 2.3 7   5.742 Yes ( 0.14286 0.85714 ) *
##                  391) tobacco > 2.3 5   5.004 No ( 0.80000 0.20000 ) *
##             49) sbp > 147 19   7.835 No ( 0.94737 0.05263 ) *
##           25) famhist > 0.5 69  95.520 No ( 0.52174 0.47826 )
##             50) ldl < 5.17 35  43.570 No ( 0.68571 0.31429 )
##              100) obesity < 21.895 6   5.407 Yes ( 0.16667 0.83333 ) *
##              101) obesity > 21.895 29  29.570 No ( 0.79310 0.20690 )
##                202) age < 48.5 17   7.606 No ( 0.94118 0.05882 ) *
##                203) age > 48.5 12  16.300 No ( 0.58333 0.41667 )
##                  406) alcohol < 8.075 6   7.638 Yes ( 0.33333 0.66667 ) *
##                  407) alcohol > 8.075 6   5.407 No ( 0.83333 0.16667 ) *
##             51) ldl > 5.17 34  44.150 Yes ( 0.35294 0.64706 )
##              102) age < 57 28  38.240 Yes ( 0.42857 0.57143 )
##                204) typea < 49 5   0.000 Yes ( 0.00000 1.00000 ) *
##                205) typea > 49 23  31.840 No ( 0.52174 0.47826 )
##                  410) typea < 52.5 5   0.000 No ( 1.00000 0.00000 ) *
##                  411) typea > 52.5 18  24.060 Yes ( 0.38889 0.61111 )
##                    822) adiposity < 31.18 7   5.742 Yes ( 0.14286 0.85714
) *
##                    823) adiposity > 31.18 11  15.160 No ( 0.54545 0.45455
) *
##              103) age > 57 6   0.000 Yes ( 0.00000 1.00000 ) *
##         13) tobacco > 8.04 32  39.750 Yes ( 0.31250 0.68750 )
##           26) adiposity < 34.4 26  25.460 Yes ( 0.19231 0.80769 )
##             52) obesity < 24.495 9   0.000 Yes ( 0.00000 1.00000 ) *
##             53) obesity > 24.495 17  20.600 Yes ( 0.29412 0.70588 ) *
##           27) adiposity > 34.4 6   5.407 No ( 0.83333 0.16667 ) *
##        7) ldl > 8.205 17  12.320 Yes ( 0.11765 0.88235 )
##         14) alcohol < 15.71 12   0.000 Yes ( 0.00000 1.00000 ) *
##         15) alcohol > 15.71 5   6.730 Yes ( 0.40000 0.60000 ) *

# Let's pick 822, it means if adiposity is the predictor, and when its value
is greater than 31.18,
```

```
# there're 7 data points belongs to this region.
# The prediction at this node is chd=1, 14.286% of data points have cha value
to be 0, and the rest of them are 1.
# A * in the line denotes that this is in fact a terminal node
```

d)

```
plot(treeModel)
text(treeModel, pretty = 0)
```



e)

```
treePred = predict(treeModel, SAheart_valid, type="class")
table(SAheart_valid$Present, treePred)
```

```
##      treePred
##        No Yes
##   No  121  39
##   Yes  41  30
```

Error rate is (42+41)/231 = 35.93%

f)

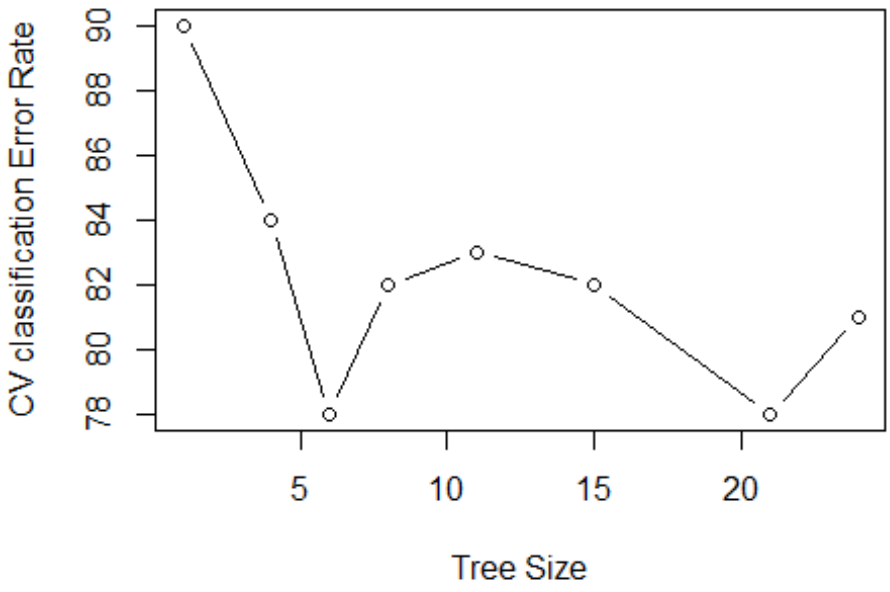
```
treeCV = cv.tree(treeModel, FUN=prune.misclass)
names(treeCV)
```

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

treeCV

## $size
## [1] 24 21 15 11  8  6  4  1
##
## $dev
## [1] 81 78 82 83 82 78 84 90
##
## $k
## [1]     -Inf 0.000000 1.000000 1.250000 1.666667 4.000000 5.000000
8.333333
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The tree with 6 terminal nodes results in the lowest cross-validation error rate, 78

g)

```
plot(treeCV$size ,treeCV$dev ,type="b", xlab="Tree Size", ylab="CV
classification Error Rate")
```



h)

When size = 6, result for cross-validation error rate is lowest.

i)

```
treePrune =prune.misclass(treeModel ,best = 6)
```

j)

```
summary(treePrune)
```

```
##
## Classification tree:
## snip.tree(tree = treeModel, nodes = c(7L, 24L, 51L, 13L, 50L))
## Variables actually used in tree construction:
## [1] "age"     "ldl"     "tobacco" "famhist"
## Number of terminal nodes:  6
## Residual mean deviance:  1.013 = 228 / 225
## Misclassification error rate: 0.2338 = 54 / 231
```

Misclassification error rate is 23.38%, which is higher than before.

k)

```
treePrunePred = predict(treePrune, SAheart_valid, type="class")
table(SAheart_valid$Present, treePrunePred)
```

```
##       treePrunePred
##         No Yes
##   No   133  27
##   Yes   37  34
```

Test error rate is now (27+37)231 = 27.71%, which is lower than before.

(4b i & ii)

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.3.3
```

```
## randomForest 4.6-12
```

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

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.3
```

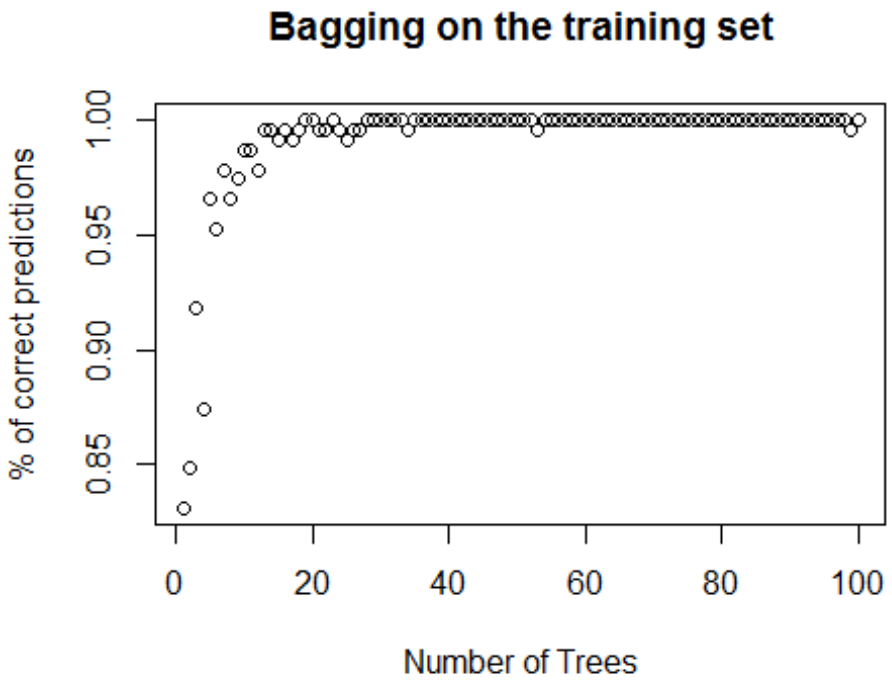
```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

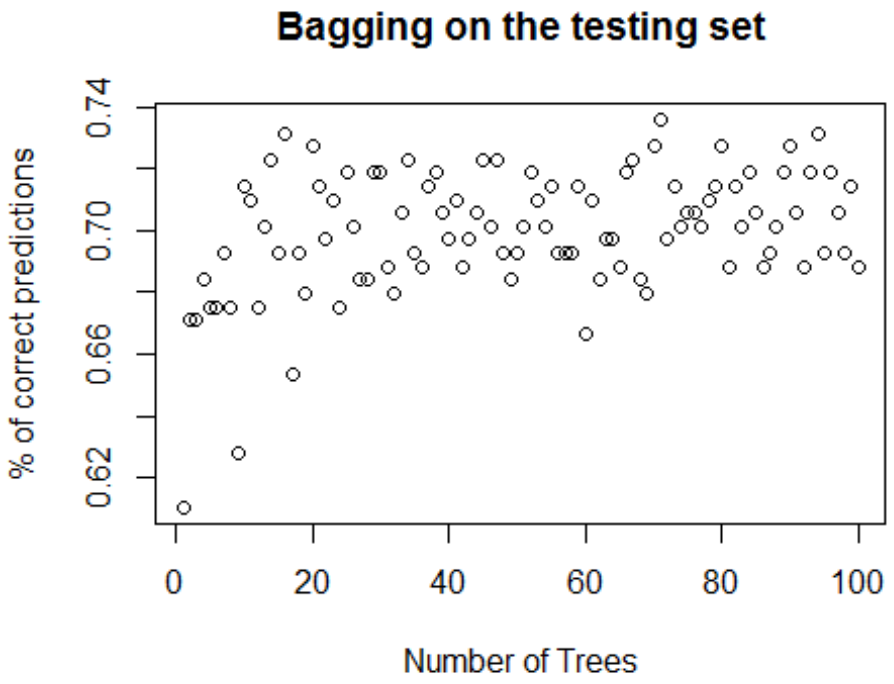
```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

```
##
## Attaching package: 'ggplot2'
```

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

train.errors = rep(0, 100)
test.errors = rep(0, 100)
for (i in 1:100) {
 tree.bag = randomForest(Present~., SAheart_train, mtry=9, ntree=i,
importance=TRUE)
 yhat.bag = predict(tree.bag, SAheart_train)
 yhat.bag.test = predict(tree.bag, SAheart_valid)
 train.errors[i] = confusionMatrix(yhat.bag,
SAheart_train$Present)$overall[1]
 test.errors[i] = confusionMatrix(yhat.bag.test,
SAheart_valid$Present)$overall[1]
}
plot(train.errors, xlab="Number of Trees", ylab="% of correct predictions",
main="Bagging on the training set")
```




```
plot(test.errors, xlab="Number of Trees", ylab="% of correct predictions",
main="Bagging on the testing set")
```

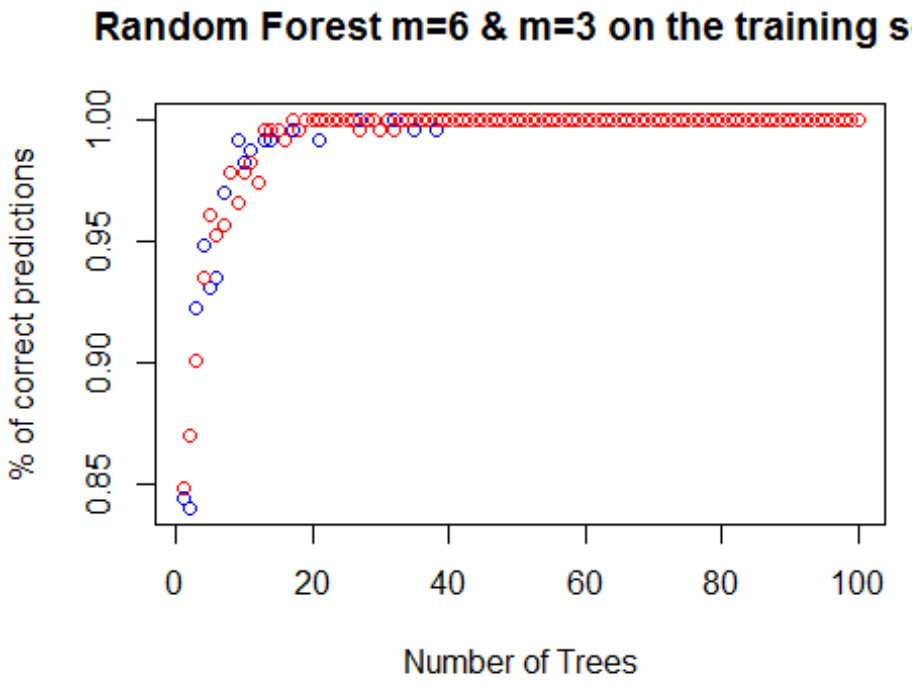


(4b iii)

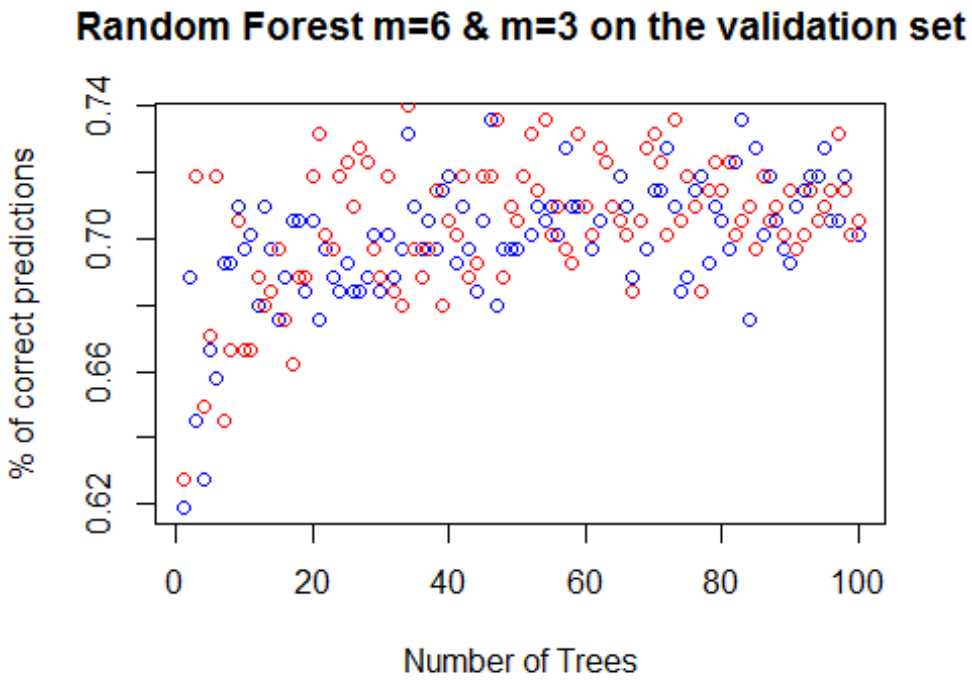
When B increase the accuracy on the training set is keep increasing, but it is not true for valid set. From the plot, we can see it may cause overfit if B is too large. ##### (4c i & ii)

```
train.errors1 = rep(0, 100)
train.errors2 = rep(0, 100)
test.errors1 = rep(0, 100)
test.errors2 = rep(0, 100)
for (i in 1:100) {
  rf1 = randomForest(Present~., SAheart_train, mtry=6, ntree=i, importance =
TRUE)
  rf2 = randomForest(Present~., SAheart_train, mtry=3, ntree=i, importance =
TRUE)
 yhat.rf1 = predict(rf1, SAheart_train)
 yhat.rf2 = predict(rf2, SAheart_train)
 yhat.rf1.test = predict(rf1, SAheart_valid)
 yhat.rf2.test = predict(rf2, SAheart_valid)
 train.errors1[i] = confusionMatrix(yhat.rf1,
SAheart_train$Present)$overall[1]
 test.errors1[i] = confusionMatrix(yhat.rf1.test,
SAheart_valid$Present)$overall[1]
 train.errors2[i] = confusionMatrix(yhat.rf2,
SAheart_train$Present)$overall[1]
 test.errors2[i] = confusionMatrix(yhat.rf2.test,
SAheart_valid$Present)$overall[1]
}
```

```
x=seq(1, 100)
plot(x, train.errors1, xlab="Number of Trees", ylab="% of correct
predictions", type="p", main="Random Forest m=6 & m=3 on the training set",
col="blue")
points(x, train.errors2, col="red")
```




```
plot(x, test.errors1, xlab="Number of Trees", ylab="% of correct
predictions", type="p", main="Random Forest m=6 & m=3 on the validation set",
col="blue")
points(x, test.errors2, col="red")
```



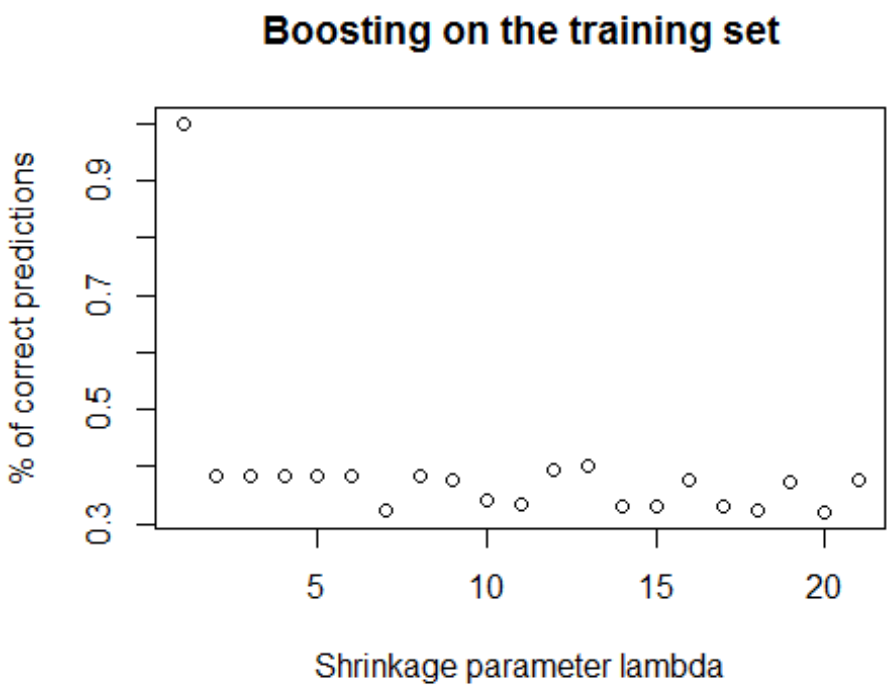
(4c iii)

This is similar as in Bagging. As we can see the training set accuracy is quickly increasing towards to 100%. However, it also cause overfit which doesn't improve accuracy of valid set prediction much after B > 20.

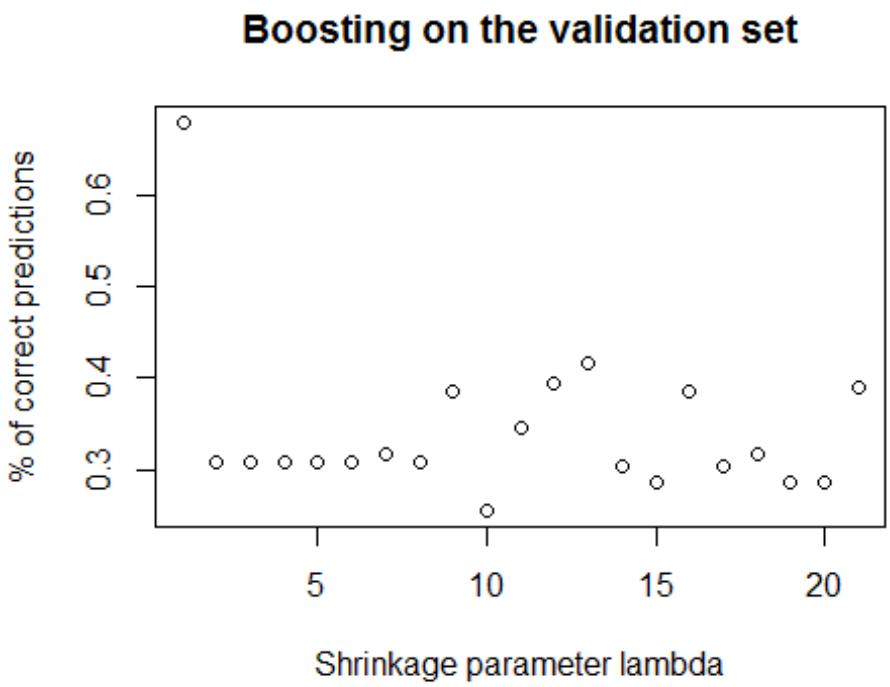
(4d)

```
# i & ii
library(gbm)

## Warning: package 'gbm' was built under R version 3.3.3

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3

pows = seq(-3, -1, by=0.1)
lambdas = 10^pows
length.lambdas = length(lambdas)
```

```
train.errors3 = rep(0, length.lambdas)
test.errors3 = rep(0, length.lambdas)
SAheart_train <- SAheart[train,]
SAheart_train <- SAheart_train[,-11]
SAheart_validation <- SAheart[valid,]
SAheart_validation <- SAheart_validation[,-11]
for (i in 1:length.lambdas) {
   tree.boost = gbm(chd~., data = SAheart_train, distribution = "bernoulli",
n.trees = 1000, shrinkage = i, verbose = F)
   yhat.boost = predict(tree.boost, SAheart_train, n.trees = 1000,
type="response")
   yhat.boost.test = predict(tree.boost, SAheart_validation, n.trees = 1000,
type="response")
   yhat.boost1 = ifelse(yhat.boost>=0.5, 1, 0)
   yhat.boost1.test = ifelse(yhat.boost.test>=0.5, 1, 0)
   train.errors3[i] = confusionMatrix(yhat.boost1,
SAheart_train$chd)$overall[1]
   test.errors3[i] = confusionMatrix(yhat.boost1.test,
SAheart_validation$chd)$overall[1]
}

plot(train.errors3, xlab="Shrinkage parameter lambda", ylab="% of correct
predictions", main="Boosting on the training set")
```



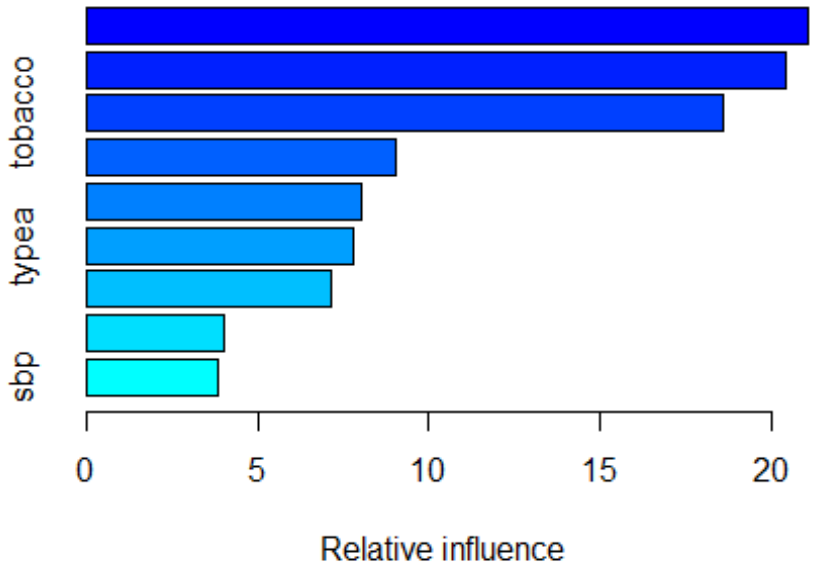

```
plot(test.errors3,xlab="Shrinkage parameter lambda", ylab="% of correct
predictions", main="Boosting on the validation set")
```




```
#While the lambda increases from 0.001 to 0.1, the accuracy is decresing,
this is because the slower the tree grows, the more accuracy the prediction
would be.
```

(4c)

```
tree.boost = gbm(chd~., data = SAheart_train, distribution = "bernoulli",
n.trees = 1000, shrinkage = 0.01, verbose = F)
summary(tree.boost)
```



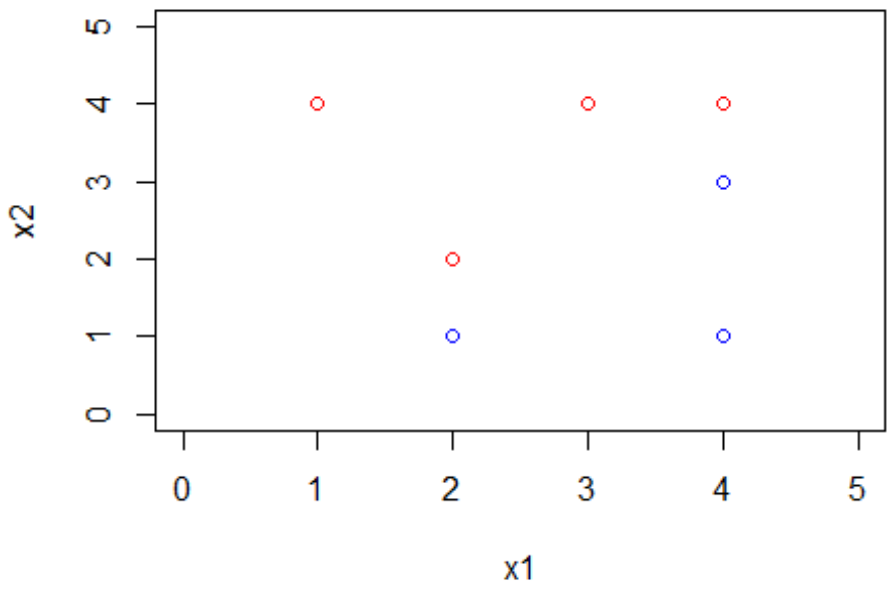

```
##                  var   rel.inf
## ldl              ldl 21.074627
## age              age 20.436518
## tobacco     tobacco 18.572606
## adiposity adiposity  9.043009
## obesity     obesity  8.035933
## typea         typea  7.792631
## famhist     famhist  7.136053
## alcohol     alcohol  4.034260
## sbp             sbp  3.874363
```

ldl, age and tobacco are highly correlated to chd.

Problem 5

(a)

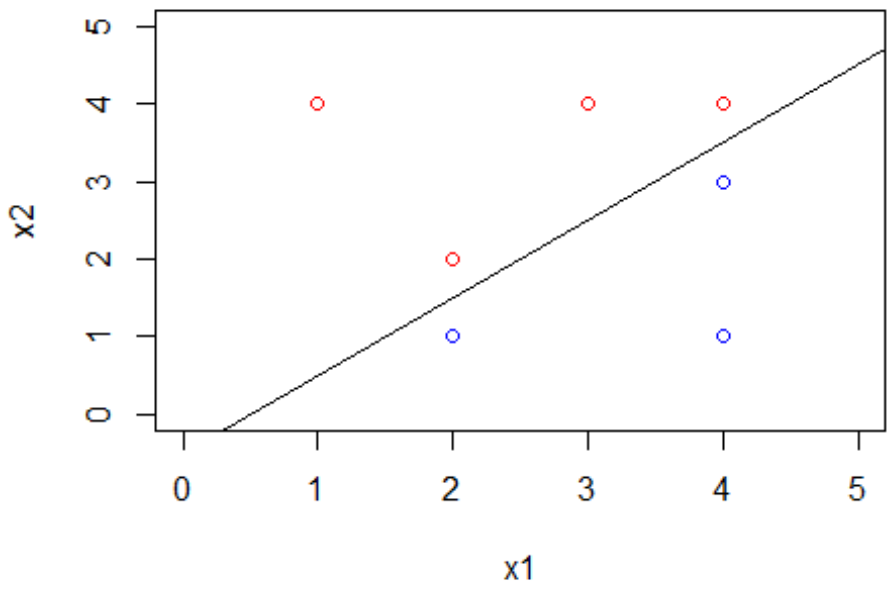
```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
```



(b)

(2,2), (4,4) (2,1), (4,3) => (2,1.5), (4,3.5) b = (3.5 - 1.5) / (4 - 2) = 1 a = X2 - X1 = 1.5 - 2 = -0.5

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
```

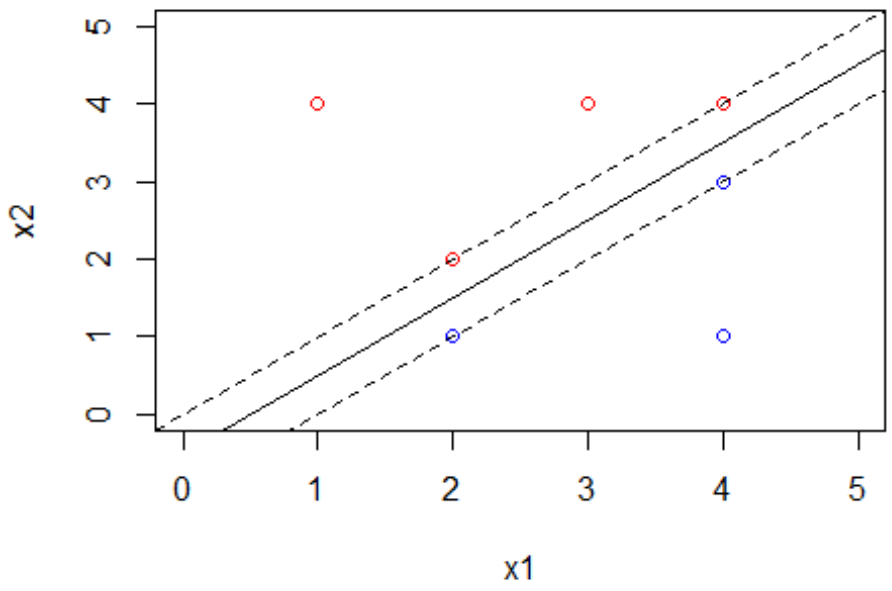


(c)

0.5 - X1 + X2 > 0

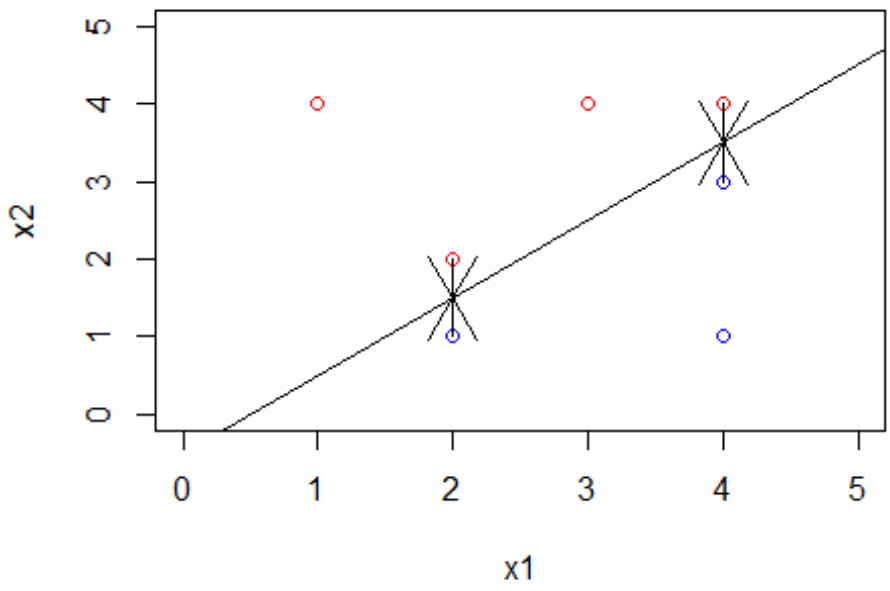
(d)

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



(e)

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
arrows(2, 1, 2, 1.5)
arrows(2, 2, 2, 1.5)
arrows(4, 4, 4, 3.5)
arrows(4, 3, 4, 3.5)
```

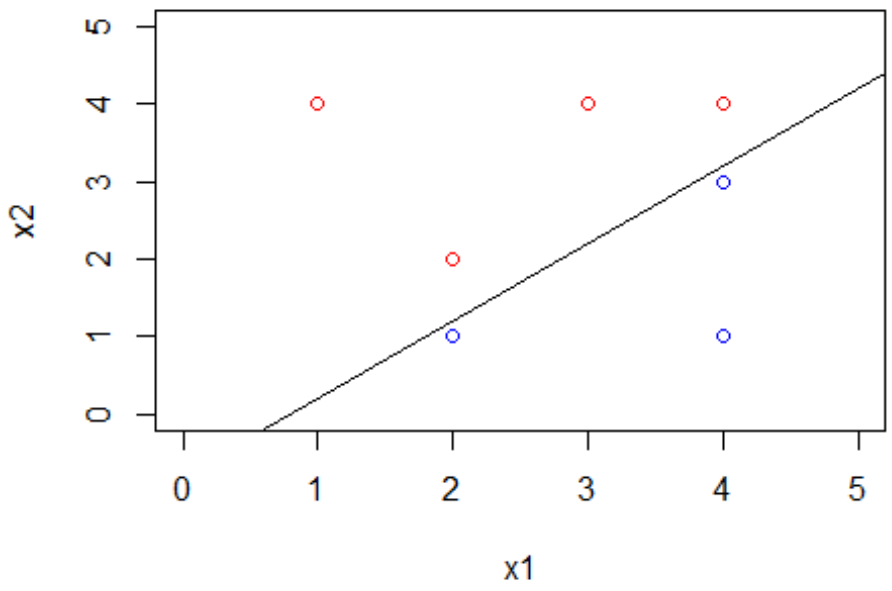


(f)

A slight movement of observation #7 (4,1) blue would not have an effect on the maximal margin hyperplane because it is out of margin and far from it.

(g)

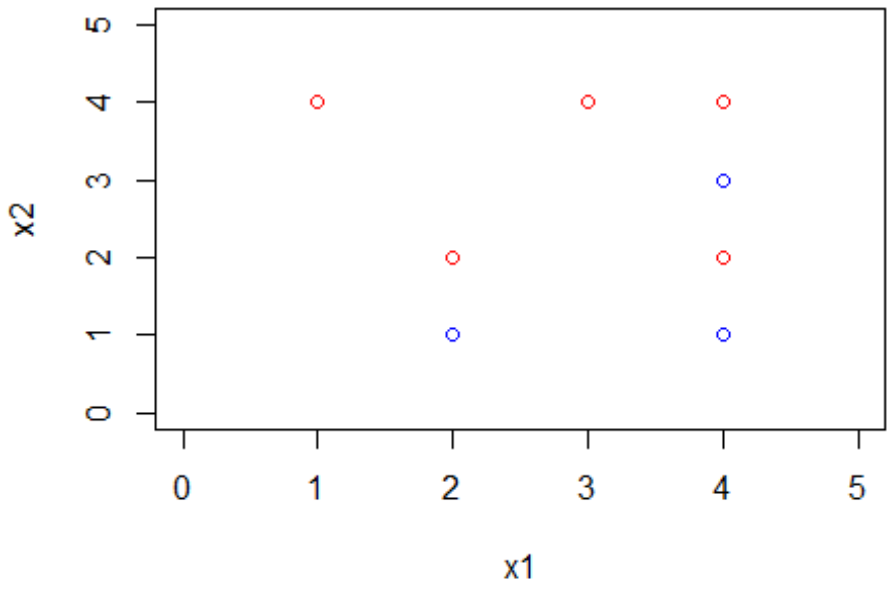
```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.8, 1)
```



$-0.8 - X1 + X2 > 0$

(h)

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
points(c(4), c(2), col = c("red"))
```



Problem 6

(a)

```
data= read.table("http://www-
stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data",
sep=",",head=T,row.names=1)
data$chd <- factor(data$chd)
set.seed(1)
sub <- sample(nrow(data), floor(nrow(data)/2))
train<-data[sub,]
valid<-data[-sub,]
head(train)
```

```
##     sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
## 123 120    0.00 5.01     26.13  Absent    64   26.21   12.24  33   0
## 172 118    0.75 2.58     20.25  Absent    59   24.46    0.00  32   0
## 265 136    5.00 4.19     23.99 Present    68   27.80   25.86  35   0
## 418 158   16.00 5.56     29.35  Absent    36   25.92   58.32  60   0
## 93  143    0.46 2.40     22.87  Absent    62   29.17   15.43  29   0
## 412 178   20.00 9.78     33.55  Absent    37   27.29    2.88  62   1
```

```
library(caret)
library(ggplot2)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.3
```

```
library(MASS)
control <- trainControl(method="cv", number=12)
metric <- "Accuracy"
```

```
# Linear Kernel
# Train
model.svm <- train(chd~., data=train, method="svmLinear", metric=metric,
trControl=control)
```

```
## Loading required package: kernlab
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
prediction.svm <- predict(model.svm, train)
confusionMatrix(prediction.svm, train$chd)$overall[1]
```

```
##  Accuracy
## 0.7272727
```

```r
# Valid
model.svm <- train(chd~., data=train, method="svmLinear", metric=metric,
trControl=control)
prediction.svm <- predict(model.svm, valid)
confusionMatrix(prediction.svm, valid$chd)$overall[1]
```

```
##  Accuracy
## 0.7099567
```

```r
# Radial Kernel
# Train
model.svm <- train(chd~., data=train, method="svmRadial", metric=metric,
trControl=control)
prediction.svm <- predict(model.svm, train)
confusionMatrix(prediction.svm, train$chd)$overall[1]
```

```
##  Accuracy
## 0.8225108
```

```r
# Valid
model.svm <- train(chd~., data=train, method="svmRadial", metric=metric,
trControl=control)
prediction.svm <- predict(model.svm, valid)
confusionMatrix(prediction.svm, valid$chd)$overall[1]
```

```
##  Accuracy
## 0.7402597
```

```r
# ROC
#Tainning
library(ROCR)
```

```
## Loading required package: gplots
```

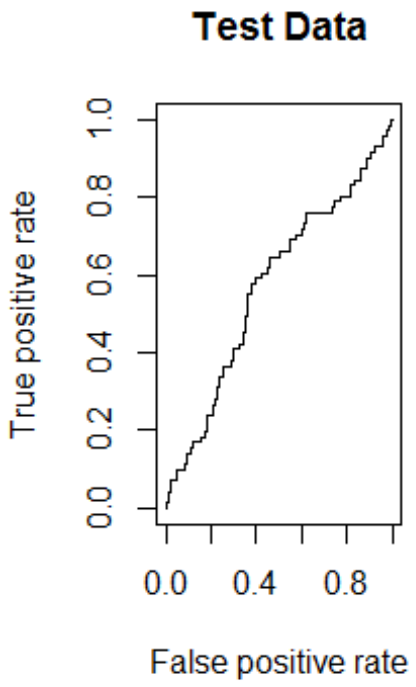
```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```r
rocplot=function(pred, truth, ...){
  predob = prediction(pred, truth)
  pref = performance(predob, "tpr", "fpr")
  plot(pref,...)
}

svmfit.opt=svm(chd~., data=train, kernel="radial", gamma=0.01, cost=1,
decision.values=T)
fitted=attributes(predict(svmfit.opt, train,
decision.values=TRUE))$decision.values
par(mfrow=c(1, 2))
```

```
rocplot(fitted, valid$chd, main="Test Data")
pred = prediction(fitted, valid$chd)
unlist(attributes(performance(pred, "auc"))$y.values)
```

```
## [1] 0.5710387
```




```
data= read.table("http://www-
stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data",
sep=",",head=T,row.names=1)
data$famhist = as.numeric(data$famhist)
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)

scaled.data <- as.data.frame(scale(data, center = mins, scale = maxs - mins))
set.seed(1)
sub <- sample(nrow(scaled.data), floor(nrow(scaled.data)/2))
train<-scaled.data[sub,]
valid<-scaled.data[-sub,]
head(train)
```

```
##           sbp    tobacco       ldl adiposity famhist     typea   obesity
## 123 0.1623932 0.00000000 0.2808362 0.5423776       0 0.7846154 0.3610414
## 172 0.1452991 0.02403846 0.1114983 0.3779021       0 0.7076923 0.3061481
## 265 0.2991453 0.16025641 0.2236934 0.4825175       1 0.8461538 0.4109159
## 418 0.4871795 0.51282051 0.3191638 0.6324476       0 0.3538462 0.3519448
## 93  0.3589744 0.01474359 0.0989547 0.4511888       0 0.7538462 0.4538896
```

```
## 412 0.6581197 0.64102564 0.6132404 0.7499301       0 0.3692308 0.3949184
##        alcohol       age chd
## 123 0.08315782 0.3673469   0
## 172 0.00000000 0.3469388   0
## 265 0.17569128 0.4081633   0
## 418 0.39622257 0.9183673   0
## 93  0.10483049 0.2857143   0
## 412 0.01956655 0.9591837   1
```

```
feats <- names(scaled.data[,c(1:9)])

# Concatenate strings
f <- paste(feats,collapse=' + ')
f <- paste('chd ~',f)

# Convert to formula
f <- as.formula(f)

f
```

```
## chd ~ sbp + tobacco + ldl + adiposity + famhist + typea + obesity +
##     alcohol + age
```

```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 3.3.3
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:ROCR':
##
##     prediction
```

```
nn <- neuralnet(f,data=train,hidden=c(10, 10, 10),linear.output=FALSE)
# Compute Predictions off Test Set
predicted.nn.values <- compute(nn,valid[,c(1:9)])

# Check out net.result
print(head(predicted.nn.values$net.result))
```

```
##                     [,1]
## 1 0.9999999937234249625
## 2 0.9869098296607109466
## 3 0.0405459177880382535
## 4 0.9999903811227067729
## 7 0.0000000001823047063
## 8 0.0000000372967220780
```

```
plot(nn)
predicted.nn.values$net.result <-
```

```
sapply(predicted.nn.values$net.result,round,digits=0)
table(valid$chd,predicted.nn.values$net.result)

##
##       0   1
##   0 119  41
##   1  30  41

# Accuracy = (125+38)/231 = 70.56%
```

Problem 7

Because our training and valid sets is randomly picked, we can't directly compare the accuracy of this homework to the one in homework3. roughly Speaking, they are performed similar because this problem is a simple 2 class classification. I believe one of the reasons to cause them slightly different is the random training and validation dataset.

•Logistic regression and Nearal Network doesn't perform well because we didn't check the correlationship between predictors.Andfrom the pair(train), we can see there are correlation between parameters.

•One drawback of diagonal LDA, Naive Bayes and SVM is that it depends on all of the features. This is the reason it can'tperform better in this problem.

•Logistic with Lasso shrunk the dataset to filter out the less related predictorswith zero coef. If we test different number of predictors, the result possibly become even better.

• Nearest shrunken centroids only depends on a subset of the features, for reasons of accuracy andinterpret ability. With an accurate threshold with less err rate, it performs well in this problem

Appendix

```
import csv
import random
import math
import operator
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA


random.seed(123)


# loadDataset and create training set and test set
def load_data(filename, split, trainingSet, testSet):
    with open(filename, 'rb') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset) - 1):
            for y in range(9):
```

```
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])


# separate class of chd for '0' & '1'
def separate(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated


# find euclidean distance between two instances (rows)
def get_distance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)


# find prior probability for each class
def priorProb(dict, dataset):
    total = len(dataset)
    prior = {}
    for k in dict:
        if (k == '1'):
            class1 = len(dict[k])
            prior1 = float(class1) / float(total)
            prior[k] = prior1
            print(prior1)
        else:
            class0 = len(dict[k])
            prior0 = float(class0) / float(total)
            prior[k] = prior0
            print(prior0)
    return prior


# get the most similar neighbors for a test instance
def get_neighbours(train, test, width):
    distances = []
    length = len(test) - 1
    for x in range(len(train)):
```

```
        dist = get_distance(train[x], test, length)
        distances.append((train[x], dist))

    distances.sort(key=operator.itemgetter(1))
    # print(distances)
    neighbours = []

    for x in range(len(distances)):
        if (distances[x][1] < width):
            neighbours.append(distances[x][0])

    return neighbours


# Gaussian product kernel
def kernel(neighborsDic, testInstance, width):
    difference1 = []
    difference0 = []
    length = len(testInstance) - 1
    n1 = 0
    n0 = 0
    for k in neighborsDic:
        if (k == "1"):
            for neighbor in neighborsDic[k]:
                diff1 = get_distance(neighbor, testInstance, length)
                difference1.append((neighbor, diff1))
                n1 += 1
            print("difference is 1" + "\n")
            print(difference1)
            print(n1)
        else:
            for neighbor in neighborsDic[k]:
                diff0 = get_distance(neighbor, testInstance, length)
                difference0.append((neighbor, diff0))
                n0 += 1
            print("difference is 0" + "\n")
            print(difference0)
            print(n0)

    sum1 = 0
    sum0 = 0
    for d in difference1:
        sum1 += math.exp(-(math.pow(d[1] / width, 2)) / 2)
    print("sum1 is: " + "\n")
    print(sum1)

    for d in difference0:
        sum0 += math.exp(-(math.pow(d[1] / width, 2)) / 2)
    print("sum0 is: " + "\n")
```

```
    print(sum0)

    width2 = math.pow(width, 2)
    print("width2 is: ", width2)
    nValue1 = n1 * math.pow(2 * width2 * math.pi, length / 2)
    print("nValue1 is: ", nValue1)
    if (nValue1 == 0):
        product1 = 0
    else:
        product1 = 1 / nValue1 * sum1

    nValue0 = n0 * math.pow(2 * width2 * math.pi, length / 2)
    print("nValue0 is: ", nValue0)
    if (nValue0 == 0):
        product0 = 0
    else:
        product0 = 1 / nValue0 * sum0

    print(product1)
    print(product0)

    prob = {}
    prob["1"] = product1
    prob["0"] = product0

    return prob


# make prediction for the test set
def get_response(prior, prob, testInstance):
    r = []
    prob1 = prior["1"] * prob["1"]
    prob0 = prior["0"] * prob["0"]

    totalProb = prob1 + prob0
    if (totalProb == 0):
        freq1 = prob1
        freq0 = prob0
    else:
        freq1 = prob1 / totalProb
        freq0 = prob0 / totalProb

    print(freq1)
    print(freq0)
    if (freq1 <= freq0):
        response = "0"
        r.append((testInstance, response))

    else:
```

```
        response = "1"
        r.append((testInstance, response))

    print(r)
    return r


if __name__ == "__main__":
    trainingSet = []
    testSet = []
    split = 0.5
    # data.csv is the dataset "South African Heart Disease";
    # however, it doesn't contain a header line and non-continous value such
as famhist
    load_data('data.csv', split, trainingSet, testSet)
    print 'Train set: ' + repr(len(trainingSet))
    print 'Test set: ' + repr(len(testSet))
    # print trainingSet

    separatedClass = separate(trainingSet)
    # print('separatedClass instances: {0}').format(separatedClass)
    prior = priorProb(separatedClass, trainingSet)

    width = 50

    # make prediction for each test set
    responses = []
    correct = 0
    print(testSet)
    for x in range(len(testSet)):
        neighbors = get_neighbours(trainingSet, testSet[x], width)
        print("neighbors" + "\n")
        print(neighbors)

        neighborsClass = separate(neighbors)
        print("neighborsClass" + "\n")
        print(neighborsClass)

        prob = kernel(neighborsClass, testSet[x], width)

        response = get_response(prior, prob, testSet[x])
        print("testSet[x] is:")
        print(testSet[x])

        if (testSet[x][-1] == response[0][1]):
            correct += 1
        responses.append(response)
        print("responses is " + "\n")
        print(responses)
```

```
accuracy = (correct / float(len(testSet))) * 100.0

# LDA analysis
trainingSetX = []
trainingSetY = []
testSetX = []
testSetY = []
for trainx in trainingSet:
    trainy = trainx.pop()
    trainingSetX.append(trainx)
    trainingSetY.append(trainy)
for testx in testSet:
    testy = testx.pop()
    testSetX.append(testx)
    testSetY.append(testy)
X = np.array(trainingSetX)
Y = np.array(trainingSetY)
lda = LDA()
lda.fit(X, Y)
predict = lda.predict(testSetX)
print(predict)
testY = np.array(testSetY)
print(testY)
accuracyLDA = np.sum(predict == testY) / float(len(testSet)) * 100.0

print "Naive Bayes Accuracy: " + str(accuracy)
print "LDA Accuracy: " + str(accuracyLDA)
```