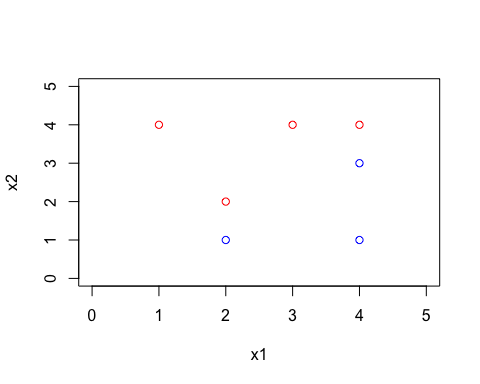
1.

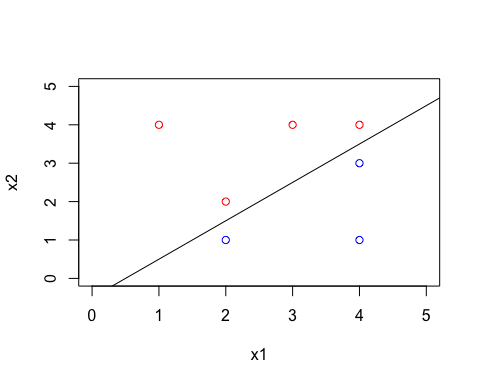
(a)

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)

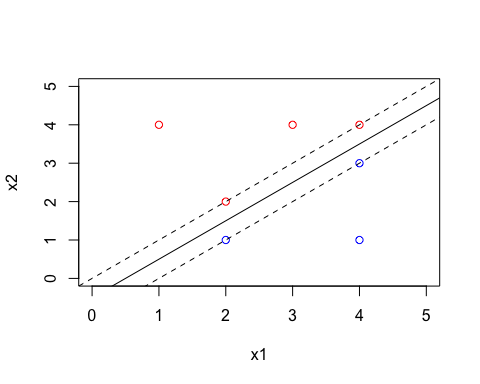
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))  
abline(-0.5, 1)



(c) Classify to Red if X1−X2−0.5 < 0, and classify to Blue otherwise

(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)

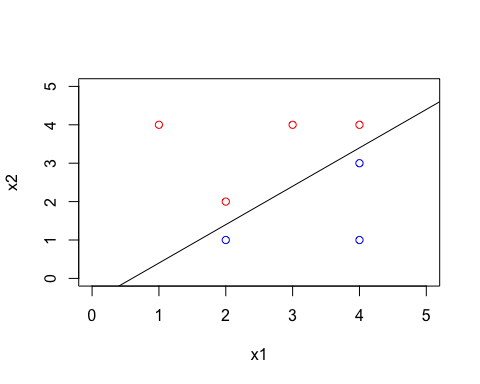


(e) The support vectors are the points (2,1), (2,2), (4,3) and (4,4).

(f) Point (4, 1) is not support vectors, so it’s ok to move it around slightly.

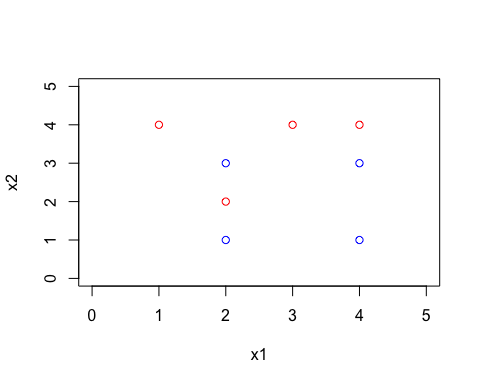
(g)

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))  
abline(-0.6, 1)



(h)

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))  
points(c(2), c(3), col = c("blue"))



2.

xy <- read.table("az-5000.txt", header = TRUE)  
set.seed(1)  
train\_ind <- sample(nrow(xy), 4000)  
train\_data <- xy[train\_ind,]  
test\_data <- xy[-train\_ind,]

(a)

nH <- 10  
n\_weights <- (18 + 1)\*nH + (nH + 1)\*26  
# Based on the rule of thumb, the approximate value of nH should be 19/10 which approximately equal to 2.

(b)

num\_train\_y = as.numeric(train\_data$char)  
mat\_train\_y = as.matrix(num\_train\_y)  
binary\_train\_y = matrix(0, nrow = nrow(mat\_train\_y), ncol = 26)  
for(i in 1:4000){  
 binary\_train\_y[i, num\_train\_y[i]] = 1  
}  
sum(binary\_train\_y == 1)

# sum is 4000

(c)

library(nnet)  
nns = list()  
for(i in 1:20){  
 nns[[i]]=nnet(char~., data = train\_data, size = i, maxit = 1000)  
}

(d)

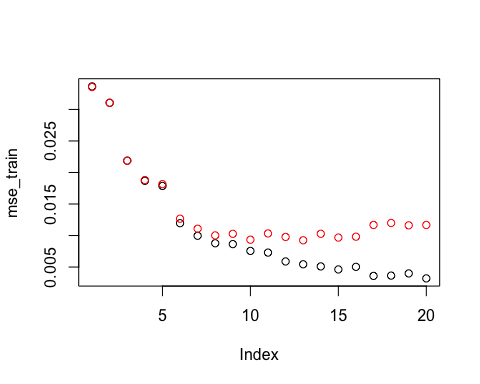
predicted\_train\_y = list()  
for(i in 1:20){  
 predicted\_train\_y[[i]] = nns[[i]]$fitted.values  
}  
library(Metrics)  
mse\_train=0  
for (i in 1:20) {  
 {mse\_train[i] = mse(binary\_train\_y, predicted\_train\_y[[i]])}  
}

(e)

num\_test\_y = as.numeric(test\_data$char)  
mat\_test\_y = as.matrix(num\_test\_y)  
binary\_test\_y = matrix(0, nrow = nrow(mat\_test\_y), ncol = 26)  
for(i in 1:1000){  
 binary\_test\_y[i, num\_test\_y[i]] = 1  
}  
   
predicted\_test\_y = list()  
for(i in 1:20){  
 predicted\_test\_y[[i]] = predict(nns[[i]], test\_data)  
}  
mse\_test = 0  
for (i in 1:20) {  
 {mse\_test[i] = mse(binary\_test\_y, predicted\_test\_y[[i]])}  
}

(f)

plot(mse\_train)  
points(mse\_test, col = "red")



mse\_train[13]

mse\_test[13]

# mse\_train[13] = 0.005439907  
# mse\_test[13] = 0.009245464

3.

set.seed(1)  
library(e1071)  
# Loading the data "spam Data".   
spam <- read.csv("spam.csv" , TRUE)  
  
# "number\_items" is size(85% of data "spam.csv") for training data.  
number\_items <- 0.85 \* nrow(spam)  
  
# Variable "indexes" holds the row index values randomly selected for training data.  
indexes <- sample(1:nrow(spam), number\_items, replace = FALSE)  
  
# "training\_data" is the training data mapped from the index values from spam data  
training\_data <- spam[ indexes,]  
  
# "test\_data" is test data mapped from the negation of the index values from spam data.  
test\_data <- spam[- indexes,]  
  
tuning\_indexes <- sample(1:nrow(training\_data), size= 500,replace=FALSE)  
  
tuning\_data <- spam[tuning\_indexes, ]

(a)

grid\_cost <- seq(10, 100, length = 10)  
grid\_gamma <-seq(0.000001, 0.001, length = 10)  
  
tuning\_x <- model.matrix(type~., tuning\_data)[, -1]  
tuning\_y <- tuning\_data$type  
tune.out <- tune.svm(tuning\_x, tuning\_y, data = tuning\_data, kernel="radial", gamma = grid\_gamma, cost = grid\_cost)  
summary(tune.out)

# The best parameters were gamma = 0.001000 and cost = 80. The best error rate was 8.2%.  
# The best parameters were gamma = 0.000889 and cost = 90. The best error rate was 8.2%.

(b)

training\_x <- model.matrix(type~., training\_data)[, -1]  
training\_y <- training\_data$type  
# for another case, gamma = 0.001000, cost = 80  
train\_svm <- svm(training\_x, training\_y, kernel = "radial", gamma = 0.000889, cost = 90)  
summary(train\_svm)

# There are 806 support vectors when gamma = 0.001000, cost = 80  
# There are 801 support vectors when gamma = 0.000889, cost = 90

(c)

test\_x <- model.matrix(type~., test\_data)[,-1]  
test\_y <- test\_data$type  
# for another case, gamma = 0.001000, cost = 80  
test\_svm <- svm(test\_x, test\_y, kernel = "radial", gamma = 0.000889, cost= 90)  
test\_pred <- predict(test\_svm, test\_x)  
summary(test\_pred)

confusion\_mat <- table(test\_pred, test\_data$type)  
accuracy <- sum(diag(confusion\_mat))/sum(confusion\_mat)  
confusion\_mat

accuracy

# When gamma = 0.001000, cost = 80:  
# Confusion matrix is  
# test\_pred nonspam spam  
# nonspam 399 22  
# spam 14 256  
# Accuracy is 94.8% on test data.  
# when gamma = 0.00889, cost = 90;  
# Confusion matrix is  
# test\_pred nonspam spam  
# nonspam 399 22  
# spam 14 256  
# Accuracy is 94.8% on test data.

4.

set.seed(1)  
data = read.table("housetype\_data.txt", header = TRUE, sep = ",")  
data$age = as.factor(data$age)  
data$sex = as.factor(data$sex)  
data$ms = as.factor(data$ms)  
data$edu = as.factor(data$edu)  
data$ocu = as.factor(data$ocu)  
data$inc = as.factor(data$inc)  
data$ba = as.factor(data$ba)  
data$di = as.factor(data$di)  
data$hhs = as.factor(data$hhs)  
data$hs = as.factor(data$hs)  
data$hs2 = as.factor(data$hs2)  
data$eth = as.factor(data$eth)  
data$lang= as.factor(data$lang)  
  
num = 0.9\*nrow(data)  
num

# num = 8111.7

training = sample(nrow(data), num)   
trainingdata = data[training,]  
test = -training  
testdata = data[test,]  
sum(is.na(data))

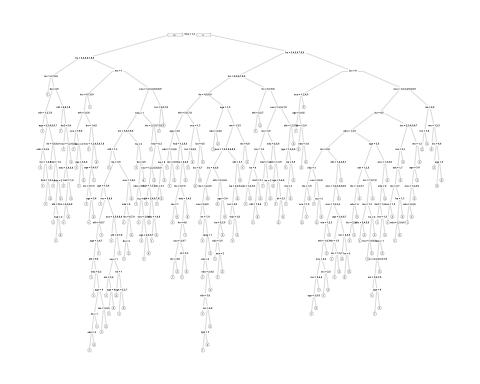
# sum(is.na(data)) = 2671

sum(is.na(trainingdata))

# sum(is.na(trainingdata)) = 2346

(a)

library(rpart)  
library(rpart.plot)  
mytree = rpart(ht~., data = trainingdata, method = "class", cp = 0.0001)  
prp(mytree)

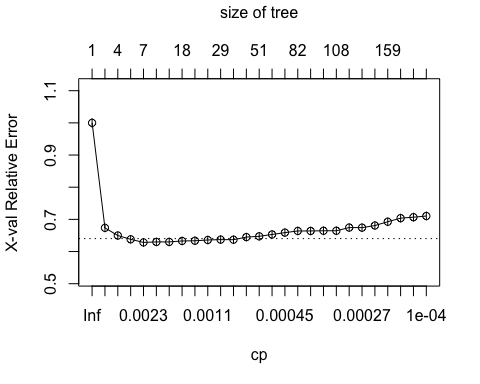


(b)

plotcp(mytree)

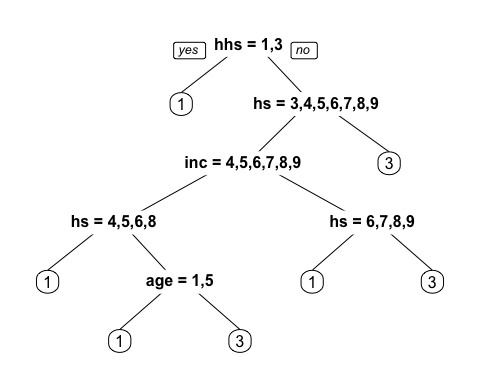
print(mytree$cptable)

which.min(mytree$cptable[,"xerror"])



(c)

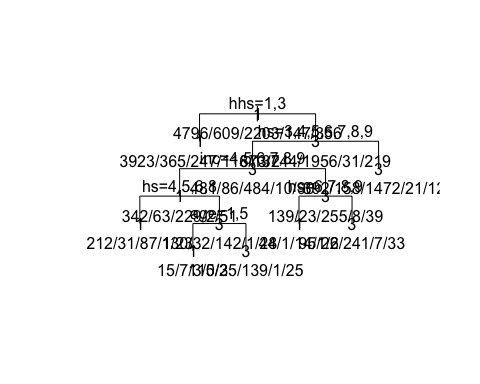
pruned\_tree = prune(mytree, cp = 0.0014077426)   
prp(pruned\_tree)



(d)

plot(pruned\_tree, compress = TRUE, uniform = TRUE, margin = 0.2)  
text(pruned\_tree, all = TRUE, pretty = 0, splits = TRUE, use.n = TRUE)

# We have pruned the tree and now it looks much smaller than the original tree. The house type strongly connects with hs (persons in your household) and connects with hhs (householder status), inc (annual income of household), and age. All of the rest predictors don't affect the house type.



(e)

summary(pruned\_tree)

## Call:  
## rpart(formula = ht ~ ., data = trainingdata, method = "class",   
## cp = 1e-04)  
## n= 8111   
##   
## CP nsplit rel error xerror xstd  
## 1 0.326696833 0 1.0000000 1.0000000 0.01335551  
## 2 0.017043741 1 0.6733032 0.6736048 0.01213496  
## 3 0.009049774 3 0.6392157 0.6497738 0.01199820  
## 4 0.003619910 4 0.6301659 0.6380090 0.01192794  
## 5 0.001407743 6 0.6229261 0.6283560 0.01186890  
##   
## Variable importance  
## hhs hs age inc ms ocu hs2 di   
## 55 13 11 9 5 3 3 1   
##   
## Node number 1: 8111 observations, complexity param=0.3266968  
## predicted class=1 expected loss=0.4087042 P(node) =1  
## class counts: 4796 609 2203 147 356  
## probabilities: 0.591 0.075 0.272 0.018 0.044   
## left son=2 (4788 obs) right son=3 (3323 obs)  
## Primary splits:  
## hhs splits as LRL, improve=1197.7280, (165 missing)  
## hs splits as RRLLLLLLL, improve= 292.7188, (287 missing)  
## inc splits as RRRRRRLLL, improve= 183.9096, (335 missing)  
## age splits as LRRLLLL, improve= 168.4505, (0 missing)  
## ba splits as RRRLL, improve= 165.4075, (825 missing)  
## Surrogate splits:  
## age splits as LLRLLLL, agree=0.664, adj=0.186, (165 split)  
## inc splits as LRRRRLLLL, agree=0.649, adj=0.148, (0 split)  
## hs splits as RLLLLLLLL, agree=0.642, adj=0.131, (0 split)  
## ms splits as LRRLL, agree=0.623, adj=0.085, (0 split)  
## ocu splits as LLRRLLRLL, agree=0.610, adj=0.053, (0 split)

# Yes, there are surrogate splits used in the construction of the optimal tree as shown.  
# A surrogate split is used to indicate rows that have n/a in the primary splitter.  
# For example, in Node number 1, hhs is the primary split, and age is the surrogate. So if hhs is missing in the observation, then age will be used to determine which tree nodes branch to go to.

(f)

pretree = predict(pruned\_tree, testdata, type = "class")  
httest = testdata$ht  
confusionM = table(pretree, httest)  
sum(diag(confusionM))

## [1] 659

sum(confusionM)

## [1] 902

accuracy = sum(diag(confusionM))/sum(confusionM)  
accuracy

## [1] 0.7305987

# Accuracy is 73.06%