STA 545 Statistical Data Mining I, Fall 2020

Homework 10

Stella Liao

December 9, 2020

Problem 1

a)

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
spam <- read.table("spam.data.txt")
#spam$V58 <- as.factor(spam$V58)

set.seed(2)
train.num <- sample(nrow(spam), 2301)
spam.train <- spam[train.num , ]
spam.test <- spam[-train.num ,]

frac.spam.train <- nrow(subset(spam.train,V58 == 1))/nrow(spam.train)
frac.spam.test <- nrow(subset(spam.test,V58 == 1))/nrow(spam.test)


# the fraction of the training data is spam
frac.spam.train
```

```
## [1] 0.398957
```

```
# the fraction of the testng data is spam
frac.spam.test
```

```
## [1] 0.3891304
```

b) Bagging

```
library(randomForest)
```

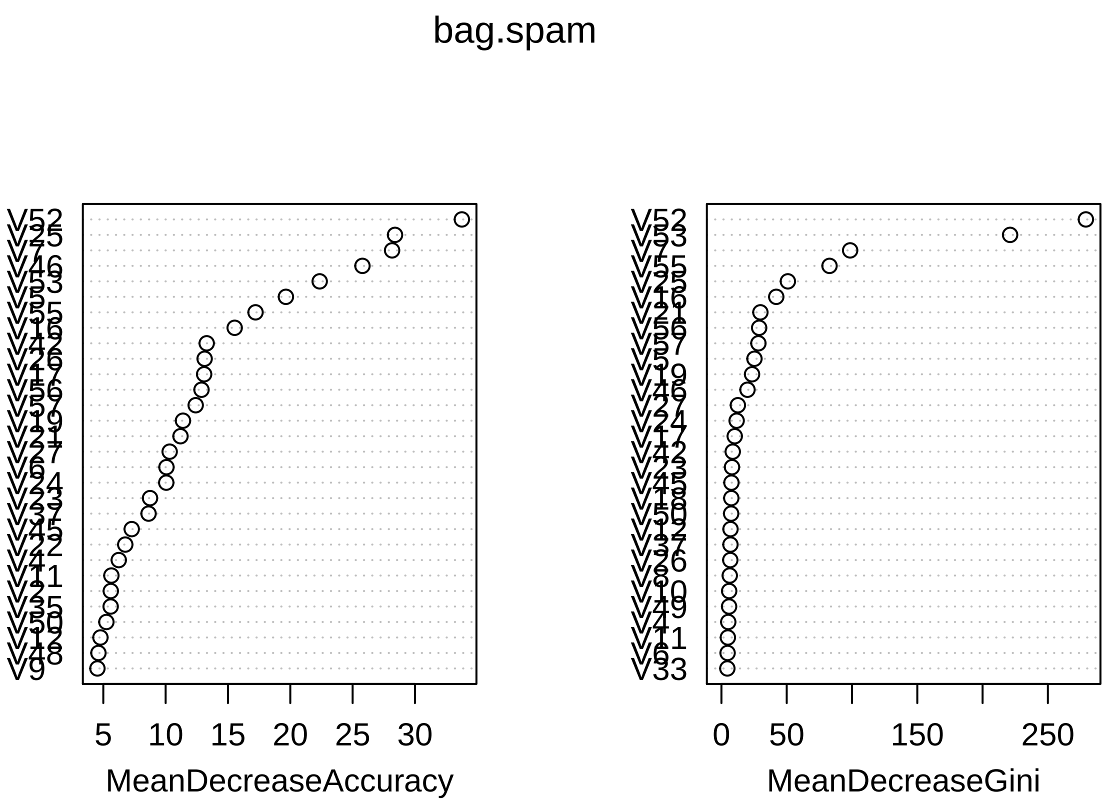
```
## randomForest 4.6-14

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

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
bag.spam = randomForest(as.factor(V58) ~ .,data = spam.train, mtry = 57, importance = TRUE, ntree = 100)

pred.bag.spam <- predict(bag.spam, newdata = spam.test)
bag.error.rate <- mean(pred.bag.spam != spam.test$V58)

#the error rate of the ensemble on the testing data
bag.error.rate
```

```
## [1] 0.06347826
```

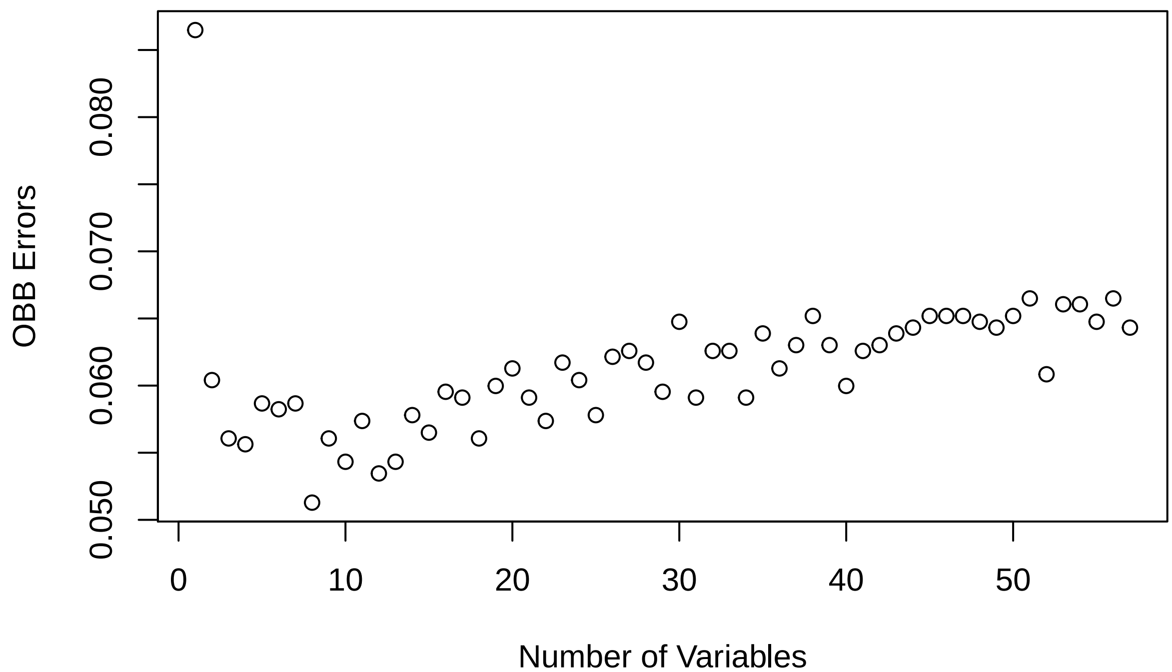
```r
varImpPlot(bag.spam)
```



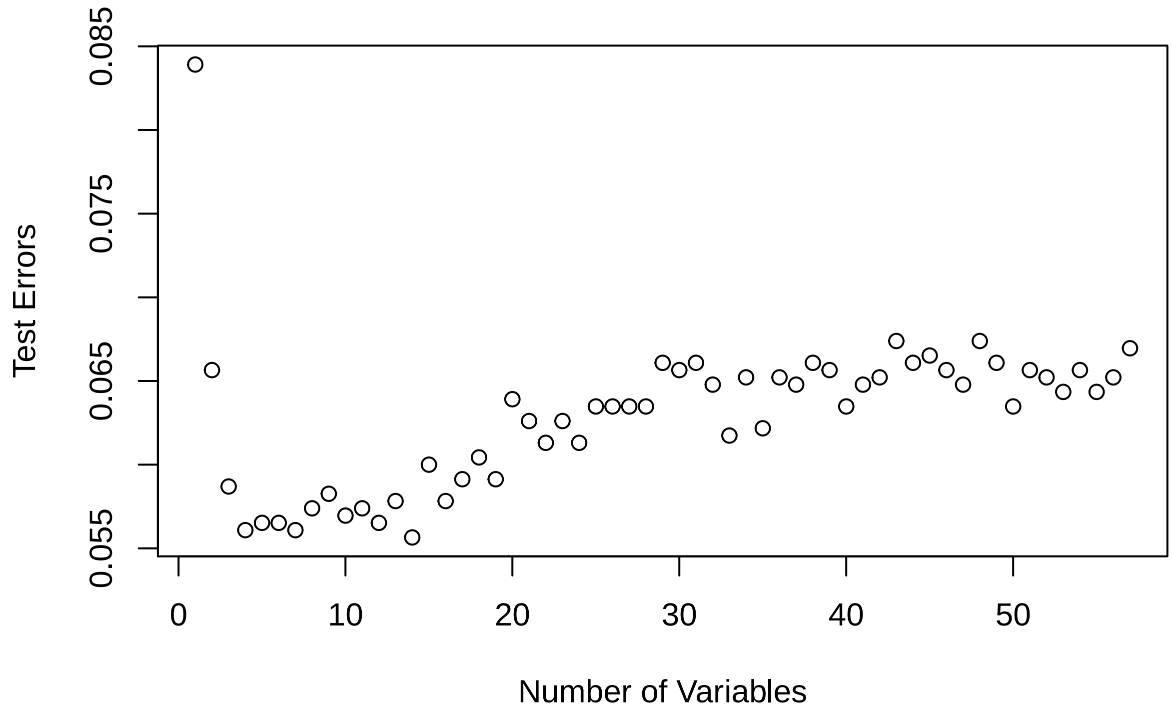
c)Random Forest

```r
m.vec <- c(1:57)

cal.error <- function(vec){
  n = length(vec)
  obb.err.vec = rep(0,n)
  test.err.vec = rep(0,n)

  for(i in vec){
    rf <- randomForest(as.factor(V58) ~ .,data = spam.train, mtry = i, ntree = 100)
    pred <- predict(rf, newdata = spam.test)
```

```r
    #calculate obb error
    obb.err.vec[i] <- mean((rf$predicted != spam.train$V58)^2)
    #calculate test error
    test.err.vec[i] <- mean((pred != spam.test$V58)^2)
  }
  return(data.frame(obb = obb.err.vec,test.error = test.err.vec))
}

errors <- cal.error(m.vec)

plot(m.vec, errors$obb, xlab = "Number of Variables", ylab = "OBB Errors")
```




```r
plot(m.vec,errors$test.error, xlab = "Number of Variables", ylab = "Test Errors")
```




```r
opt.m.test <- m.vec[which.min(errors$test.error)]
opt.m.obb <- m.vec[which.min(errors$obb)]

#optimal parameter m making test error smallest
opt.m.test
```

```
## [1] 14
```

```r
#optimal parameter m making OBB error smallest
opt.m.obb
```

```
## [1] 8
```

```r
# The error rate of the random forest on the testing data
rf.error.rate <- min(errors$test.error)
rf.error.rate
```

```
## [1] 0.05565217
```

d)AdaBoost

```r
library(ada)
```

```
## Loading required package: rpart
```

```r
ada.spam <- ada(spam.train[,1:57],spam.train[,58],loss="exponential",
                type="discrete",iter=100, bag.frac=1)

pred.ada.spam <- predict(ada.spam,newdata=spam.test[,1:57],type="vector")
ada.error.rate <- mean(spam.test[,58]!=pred.ada.spam)

# The error rate of the Adaboost classifier on the testing data
ada.error.rate
```

```
## [1] 0.05478261
```

e)Logistic Regression and Evaluation

```r
glm.spam = glm(V58 ~ ., data = spam.train, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
pred.glm.spam <- ifelse(predict(glm.spam, newdata = spam.test , "response") > 0.5, 1, 0)
glm.error.rate <- mean(spam.test[,58]!=pred.glm.spam)

# The error rate of the  Logistic Regression classifier on the testing data
glm.error.rate
```

```
## [1] 0.07608696
```

To sum up, the error rate on the testing data of Bagging, Random Forest, AdaBoost and Logistic Regression are around 0.06, 0.06, 0.05, 0.08; based on the test error, AdaBoost has the best performance in this case.

Problem 2

```r
# separate data for k-fold cross validation;
cv.group<-function(k,datasize,seed){
    cvlist<-list()
    set.seed(seed)
    n<-rep(1:k,ceiling(datasize/k))[1:datasize]
    temp<-sample(n,datasize)
    x<-1:k
    dataseq<-1:datasize
  cvlist<-lapply(x,function(x) dataseq[temp==x])
  return(cvlist)
}
whole.cv.list = cv.group(5,nrow(spam.train),2)

#create formula
names=names(spam.train)
f =as.formula(paste("V58 ~", paste(names[!names %in% "V58"], collapse = " + ")))

#implement neural network model and get the test error one time
library(neuralnet)
```

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

## The following object is masked from 'package:dplyr':
##
##     compute
```

```r
cv.test.nn<-function(i,j,cv.list){

  #i refers to i-th dataset, the testing data
  #j refers to the number of neurons
  data.train <- spam.train[-cv.list[[i]],]
    data.test<- spam.train[cv.list[[i]],]

    nn = neuralnet(f, data = data.train, hidden = j, threshold=0.01,
                   algorithm = "rprop+", err.fct="ce",
                   act.fct="logistic",linear.output=FALSE)
```

```
    pred.nn = compute(nn,data.test)
    nn.pred<- lapply(pred.nn$net.result, function(x) ifelse(x>0.5,1,0))
  accu <- sum(nn.pred == data.test$V58)/nrow(data.test)

  return(list(accu,j))
}

#get the optimal number of neurons in the single layer for neural network model
i <- c(1:5)
j <- c(10,20,30,40,50) # the number of neurons that we need to test

i.s<-rep(i,times=length(j)) # [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
j.s<-rep(j,each=5) #[1] 10 10 10 10 10 20 20 20 20 20 30 30 30 30 30 40 40 40 40 40 50 50 50 50 50
ijs <- cbind(i.s,j.s)

i = 0
j = 0

tmp.accus <-NULL
accus <- rep(0,5) # contain accuracy of all numbers of neurons that we need to test
neus <- rep(0,5)
index = 1
old_neu = 10

for(p in 1:nrow(ijs)){
  if(ijs[p,2] == old_neu){
    i = ijs[p,1]
    j = ijs[p,2]
    accu_neuron = cv.test.nn(i,j,whole.cv.list)
    accu = accu_neuron[[1]]
    neu = accu_neuron[[2]]
    tmp.accus <- c(tmp.accus,accu)
    }
  accus[index] = mean(tmp.accus)
  neus[index] = j
  index = as.integer(j/10)
  old_neu = as.integer(j+10)
}

opt.h = neus[which.max(accus)]

#the optimal number of neurons in the single hidden layer
opt.h
```

```
## [1] 30
```

```
#fit the model to the spam data using opt.h
nn.spam = neuralnet(f, data = spam.train, hidden = opt.h, threshold=0.01,
                    algorithm = "rprop+", err.fct="ce",
                    act.fct="logistic",linear.output=FALSE)

pred.nn.spam = compute(nn.spam,spam.test)
net.prediction.spam<- lapply(pred.nn.spam$net.result, function(x) ifelse(x>0.5,1,0))
nn.error <- mean((net.prediction.spam != spam.test$V58)^2)
```

```
nn.error
```

```
## [1] 0.09391304
```

In this case, we separately used 10, 20, 30, 40 and 50 as the number of neurons in our 5-fold cross validation to choose the optimal number of neurons, and based on the accuracy of classification results, we finally chose 10 as the the optimal number of neurons in the single hidden layer.

Problem 3

a)data preparing

```
set.seed(50321222)
X1 <- rnorm(100,0,1)
X2 <- rnorm(100,0,1)
Z <- rnorm(100,0,1)

X.train <- as.matrix(rbind(X1,X2))
a1 <- as.matrix(rbind(3,3))
a2 <- as.matrix(rbind(3,-3))

v = t(a1) %*% X.train
Y.train <- t(1/(1 + exp(-v))+(t(a2)%*%X.train)^2+0.3*Z)
X.train <- t(X.train)

X1 <- rnorm(1000,0,1)
X2 <- rnorm(1000,0,1)
Z <- rnorm(1000,0,1)
X.test <- as.matrix(rbind(X1,X2))
a1 <- as.matrix(rbind(3,3))
a2 <- as.matrix(rbind(3,-3))

v = t(a1) %*% X.test
Y.test <- t(1/(1 + exp(-v))+(t(a2)%*%X.test)^2+0.3*Z)
X.test <- t(X.test)
```

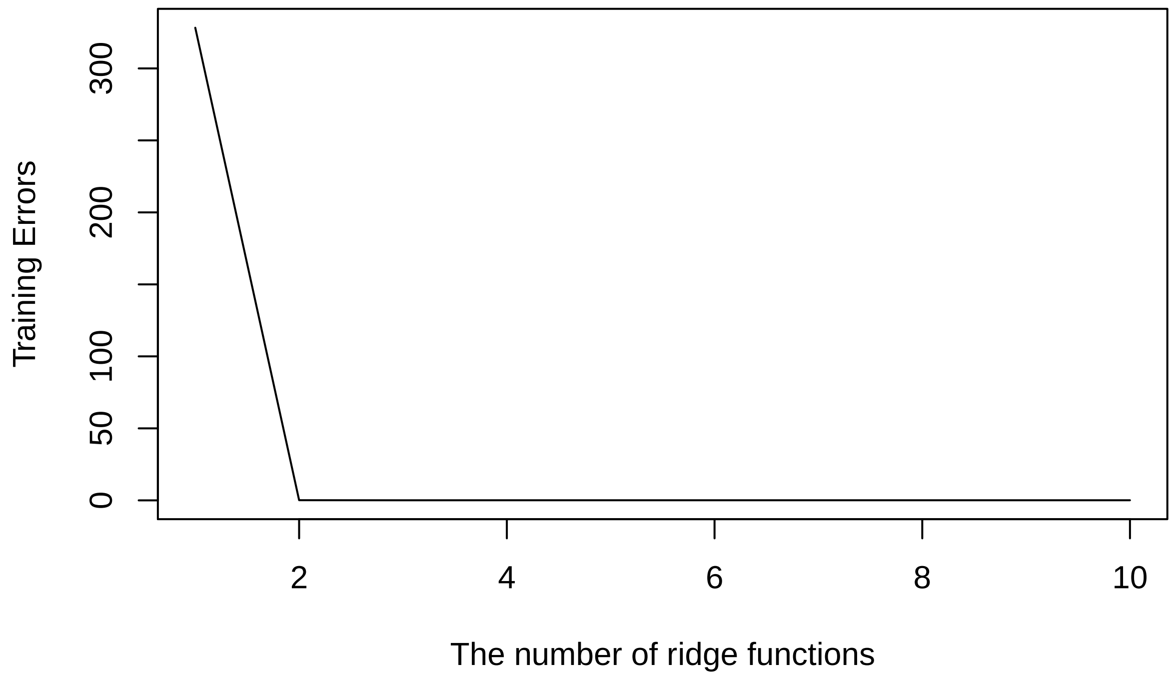
b)

```
library(MASS)
```

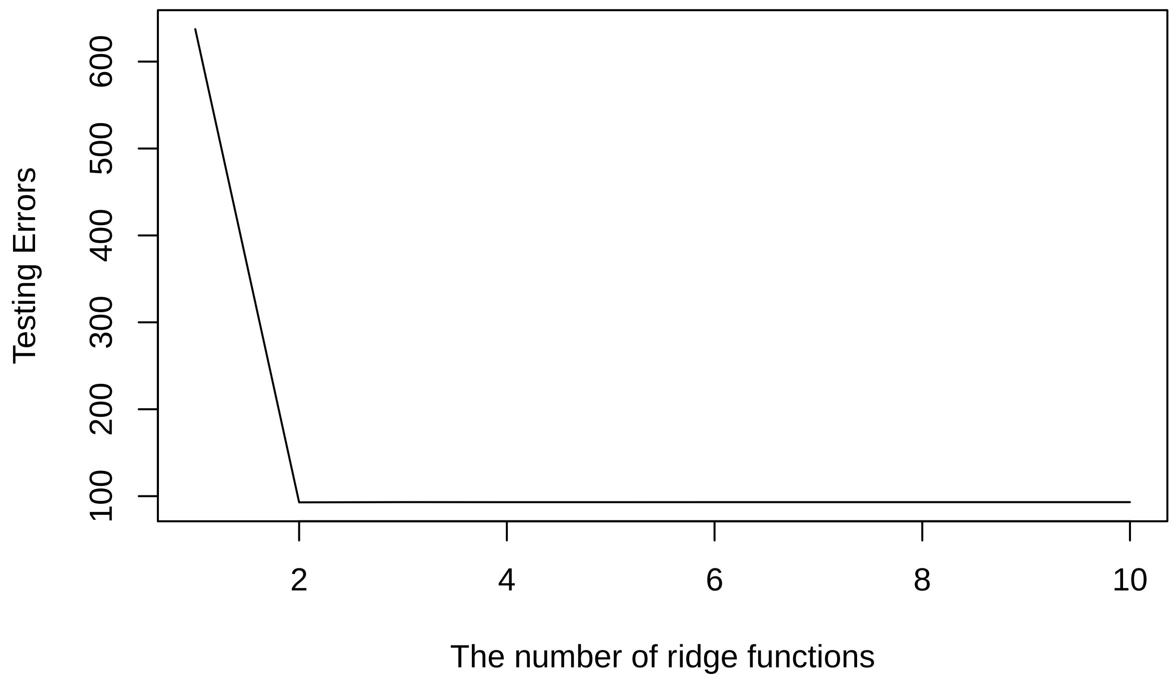
```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
train.err.all=rep(NA,10)
test.err.all=rep(NA,10)

for(i in 1:10){
  ppr.model=ppr(X.train,Y.train,nterms=i,
                sm.method="gcvspline")
  predicted.values=predict(ppr.model,newdata=X.test)
  train.err.all[i]= mean((ppr.model$fitted.values-Y.train)^2)
  test.err.all[i]=mean((predicted.values-Y.test)^2)
```

```
}
plot(1:10,train.err.all,xlab="The number of ridge functions",ylab="Training Errors",type="l")
```



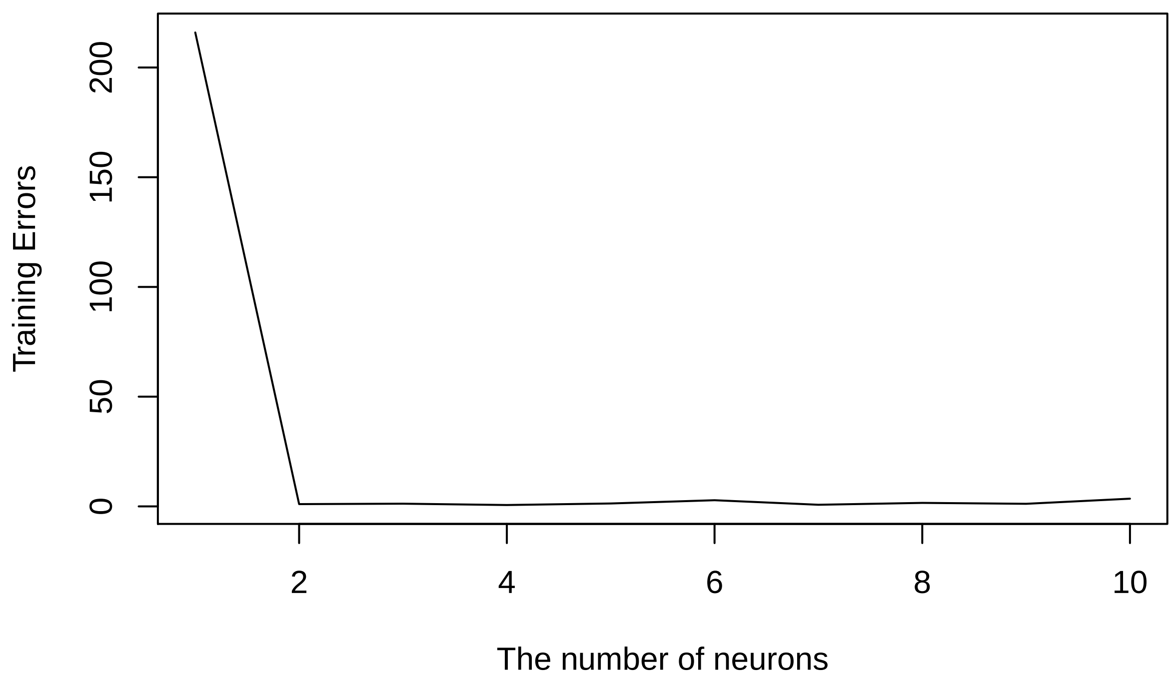

```
plot(1:10,test.err.all,xlab="The number of ridge functions",ylab="Testing Errors",type="l")
```



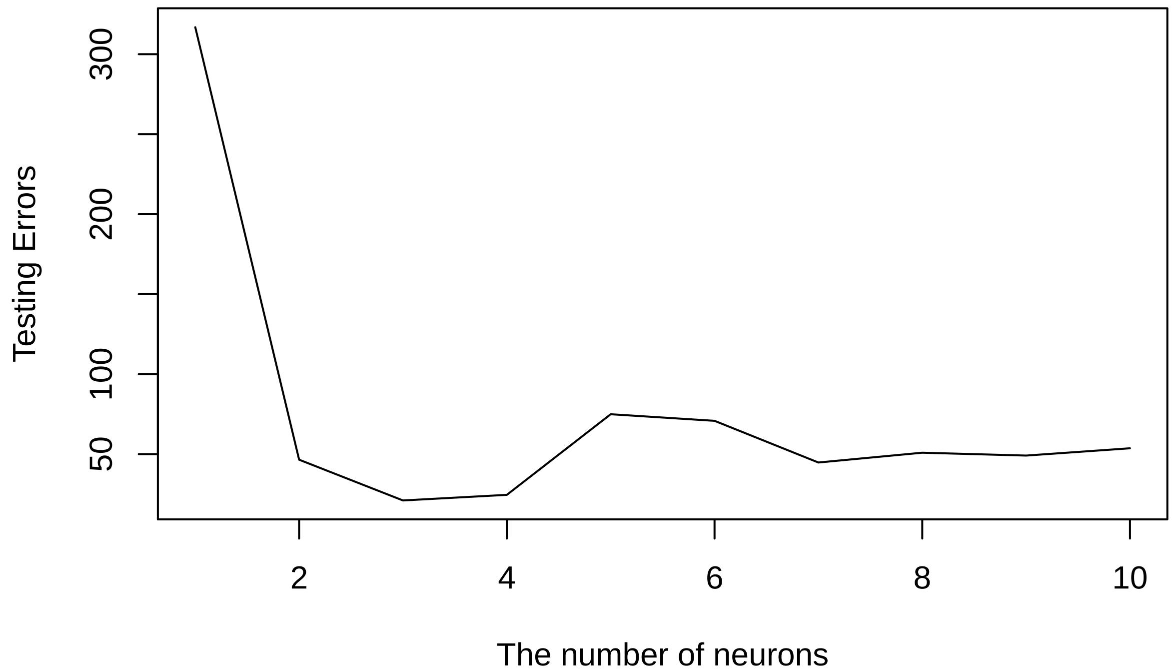
c)

```
library(MASS)
train.original <- data.frame(X.train, Y.train)
test.original <- data.frame(X.test, Y.test)
maxs=apply(train.original, 2, max)
```

```
mins=apply(train.original, 2, min)


train.scaled=as.data.frame(scale(train.original, center = mins, scale = maxs - mins))
test.scaled=as.data.frame(scale(test.original, center = mins, scale = maxs - mins))
train.err.all=rep(NA,10)
test.err.all=rep(NA,10)

for(i in 1:10){
  nn=neuralnet(Y.train~X1+X2,data=train.scaled, hidden=i, threshold=0.01,
              err.fct="sse", act.fct="logistic",linear.output=T)

  nn.scaled.prediction=compute(nn,test.scaled[,1:2])
  nn.scaled.fitted=compute(nn,train.scaled[,1:2])

  nn.prediction=nn.scaled.prediction$net.result*(max(train.original$Y.train)
                                          -min(train.original$Y.train))+min(train.original$Y.train)

  nn.fitted=nn.scaled.fitted$net.result*(max(train.original$Y.train)
                                          -min(train.original$Y.train))+min(train.original$Y.train)
  train.err.all[i]= mean((nn.fitted-train.original[,3])^2)
  test.err.all[i] = mean((nn.prediction-test.original[,3])^2)
}

plot(1:10,train.err.all,xlab="The number of neurons",ylab="Training Errors",type="l")
```




```
plot(1:10,test.err.all,xlab="The number of neurons",ylab="Testing Errors",type="l")
```


Testing Errors
300
200
100
50
2
4
6
8
10
The number of neurons