GEORGIA TECH, ISYE DEPARTMENT

ISyE 6740- Take Home Exam 3

Hongzhang Shao

April 15, 2015

CONTENTS

1 Data Manipulation

In the beginning of this problem, we read in data *AngleClosure.csv*, delete the columns corresponding to factor variables *EYE*, *GENDER*, and *ETHNIC*, and then delete rows of the dataset which have any missing values. After that, we store the cleaned data in *0-training-data.r*.

```r
# Clearing memory, reading the data
rm(list=ls())
data.tr <- read.csv("AngleClosure.csv")

# Cleaning the data
data.tr <- data.tr[complete.cases(data.tr),]
data.tr <- data.tr[,-c(1,15,16)]
data.tr[,-21] <- data.matrix(data.tr[,-21])
data.tr[,21] <- factor(data.tr[,21])
row.names(data.tr) <- NULL

# Omiting the rows, storing the data
y <- data.tr[,21]
data.tr <- data.tr[,1:11]
data.tr <- cbind(y, data.tr)
dput(data.tr, "0-training-data.r")
```

6740–t–exam–3–code/1–data–cleaning–1.R

This is about the training data. Then we need to do the same thing to testing data. The code is listed on the next page. Note that here we use the data from right eyes as a pirority. If the right eyes data missed, we use the right eye data instead. The cleaned testing data is stored in *0-testing-data.r*.

In the next few sections, we will first test on five different classification methods to classify these data set, and then use these methods to produce two ensemble methods.

In each method, we will first fit our model base on the training data, and use cross validation to tune all the parameters a model may have. The argument we looking at when tuning parameters would be the average AUC from cross validation. Then we test this model on the tesing data set, and draw the ROC curve.

The code will be pushed to my github: *https://github.com/Steve-Shao* once the exam is handed in. Feel free to download the code and play with it.

```r
# Arranging the testing data - 1
data.name <- names(data.tr)
data.ca <- read.csv("AngleClosure_ValidationCases.csv")
data.co <- read.csv("AngleClosure_ValidationControls.csv")
# Choose right eyes data first. If no, take left eyes data
logic.r.ca <- c(19,21,22,23,24,25,26,27,31,32,36)
logic.r.co <- c(18,20,22,23,24,25,26,29,30,31,35)
logic.l.ca <- c(7,9,11,12,13,14,15,30,31,32,36)
logic.l.co <- c(6,8,10,11,12,13,14,29,30,31,35)

# Combining the data of cases
logic.r <- which(complete.cases(data.ca[,logic.r.ca]))
logic.l <- which(complete.cases(data.ca[,logic.l.ca]))
logic.l <- logic.l[which(!logic.l %in% logic.r)]
data.temp.1 <- data.ca[logic.r,logic.r.ca]
data.temp.2 <- data.ca[logic.l,logic.l.ca]
names(data.temp.1) <- names(data.temp.2) <- data.name[-1]
data.ca <- rbind(data.temp.1, data.temp.2)
y <- rep("YES", nrow(data.ca))
data.ca <- cbind(y, data.ca)

# Combining the data of controls
logic.r <- which(complete.cases(data.co[,logic.r.co]))
logic.l <- which(complete.cases(data.co[,logic.l.co]))
logic.l <- logic.l[which(!logic.l %in% logic.r)]
data.temp.1 <- data.co[logic.r,logic.r.co]
data.temp.2 <- data.co[logic.l,logic.l.co]
names(data.temp.1) <- names(data.temp.2) <- data.name[-1]
data.co <- rbind(data.temp.1, data.temp.2)
y <- rep("NO", nrow(data.co))
data.co <- cbind(y, data.co)

# Comnining the whole data and storing it
data.te <- rbind(data.ca, data.co)
names(data.te) <- data.name
row.names(data.te) <- NULL
dput(data.te, "0-testing-data.r")
```

6740-t-exam-3-code/1-data-cleaning-2.R

2 Random Forest

In random forest, the only parameter that could be tuned is the number of trees. We use the cross validation with 100 iterations to select this parameter. The plot is shown below. From the plot, we could see that the AUC stays stable once we reach 500 trees. As the complexity of the random forest model increases dramatically with the number of trees, here we select 500.

The roc curve of the testing data set is shown below. The testing auc is 0.982. The code is listed on the next page.

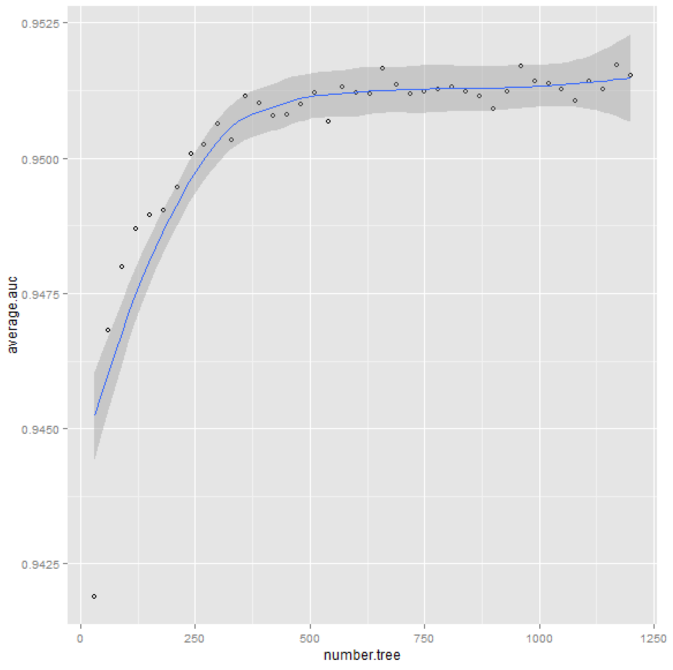


Figure 2.1: The scatter plot of AUC against the number of tree in the random forest. Start from 30 to 1200, with step length 30. From the plot, we could see that the AUC stays stable once we reach 500 trees.

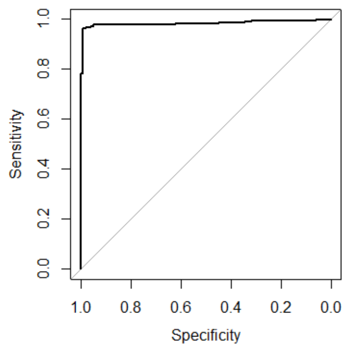


Figure 2.2: The ROC curve produced by pROC package. The auc is 0.982.

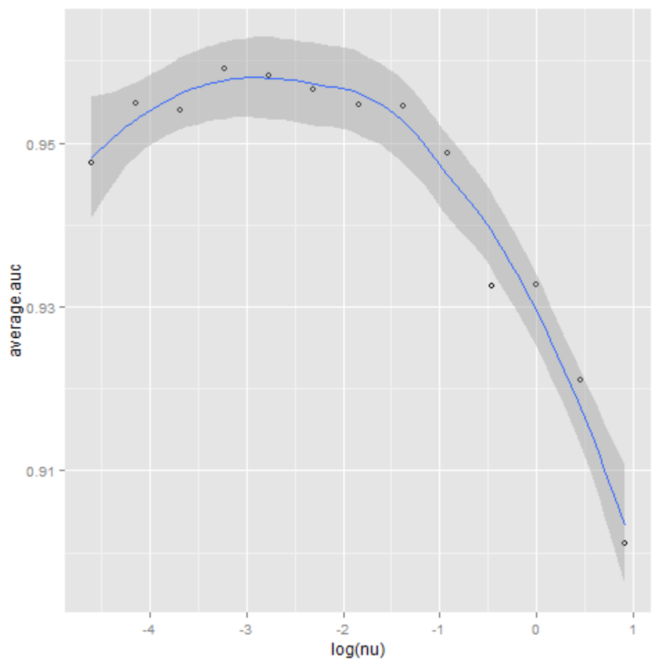
```
# Clearing memory, loading package, reading the data
rm(list=ls())
library(randomForest)
library(pROC)
library(ggplot2)
data.tr <- dget("0-training-data.r")
data.te <- dget("0-testing-data.r")

# ---------------------- Choosing number of trees ---------------------- #

# Initializing the parameter tuning
nIter <- 100
nTree <- seq(30,1200,30)
auc <- matrix(NA, nIter, length(nTree))

for(iter in 1:nIter){
  # Initializing the cross validation
  data.index <- sample(nrow(data.tr))[1:round(nrow(data.tr)/10)]
  for(nt in nTree){
    model  <- randomForest(y~., data=data.tr[-data.index,], ntree=nt)
    yHat <- predict(model, data.tr[data.index,], type="prob")
    roc <- roc(data.tr[data.index,1]~yHat[,1], data.tr)
    auc[iter,which(nTree==nt)] <- auc(roc)
  }
  print(iter)
}
data.plot <- data.frame(nTree, apply(auc,2,mean))
names(data.plot) <- c("number.tree", "average.auc")
ggplot(data.plot, aes(x=number.tree, y=average.auc)) +
  geom_point(shape=1) +  geom_smooth()

# ---------------------- Testing model ---------------------- #

# Fitting the final model, getting the prediction and testing the model
model  <- randomForest(y~., data=data.tr, ntree=500)
dput(model, "0-model-rforest.R")
yHat <- predict(model, data.te, type="prob")
roc <- roc(data.te[,1]~yHat[,1], data.te, plot=T)
> auc(roc)
Area under the curve: 0.982
```

6740-t-exam-3-code/2-rforest.R

3 Adaptive Boosting

Adaptive Boosting method has quite a stable performance, therefore 10 iterations are good enough for tuning the parameter. The parameter that need to be tuned there is *nu*. From the plot, we could see that the best nu stays stays around 0.0631. Although the number of iterations in adaptive boosting method could also be seen as a parameter, it doesn't influence much once it exceeds 40.

The roc curve of the testing data set is shown below. The testing auc is 0.9793. The code is listed on the next page.



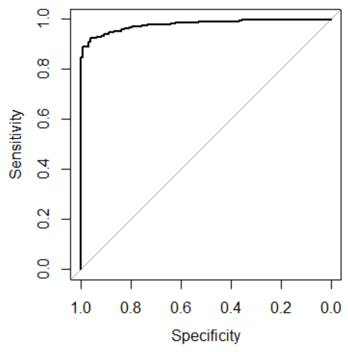


Figure 3.1: The scatter plot of AUC against the nu in the adaboost. Start from -2 to 0.4, with step length 0.2 in a log scale. From the plot, we could see that the best nu stays stays around 0.0631.

Figure 3.2: The ROC curve produced by pROC package. The auc is 0.9793.

```r
# Clearing memory, loading package, reading the data
rm(list=ls())
library(ada)
library(pROC)
library(ggplot2)
data.tr <- dget("0-training-data.r")
data.te <- dget("0-testing-data.r")

# -------------------- Choosing value of nu -------------------- #

# Initializing the parameter tuning
nIter <- 10
nu <- 10^seq(-2,0.4,0.2)
auc <- matrix(NA, nIter, length(nu))

for(iter in 1:nIter){
  # Initializing the cross validation
  data.index <- sample(nrow(data.tr))[1:round(nrow(data.tr)/10)]
  for(u in nu){
    model  <- ada(y~., data=data.tr[-data.index,], loss="logistic", type="discrete",
    iter=50, nu=u)
    yHat <- predict(model, data.tr[data.index,], type="prob")
    roc <- roc(data.tr[data.index,1]~yHat[,1], data.tr)
    auc[iter,which(nu==u)] <- auc(roc)
  }
  print(iter)
}
data.plot <- data.frame(nu, apply(auc,2,mean))
names(data.plot) <- c("nu", "average.auc")
ggplot(data.plot, aes(x=log(nu), y=average.auc)) +
  geom_point(shape=1) +  geom_smooth()

# -------------------- Testing model -------------------- #

# Fitting the final model, getting the prediction and testing the model
model  <- ada(y~., data=data.tr, loss="logistic", type="discrete", iter=50, nu=0.0631)
# dput(model, "0-model-adaboost.R")
yHat <- predict(model, data.te, type="prob")
roc <- roc(data.te[,1]~yHat[,1], data.te, plot=T)
auc(roc)
```

6740-t-exam-3-code/2-adaboost.R

4 LASSO and Ridge Regression on Logistic Regression

This approach is to frist use LASSO to select variables, and then use Ridge regression to fit the logistic regression model. The package we use here is *glmnet*. To achieve this, we first conduct the entire LASSO approach, to get totally 72 lambdas (This means LASSO terminates after 72 iterations). Then we take lambda as the parameter that need to be tuned, and pass it one by one to ridge regression, and perform a cross validation to estimate the AUC.

From the plot, we could see that the here is a tend of decresing, and therefore we should select the largest lambda, which is 0.2483. After giving this parameter to Ridge regression and use the full training data set to fit the model, we could use it to predict the response for the testing data. The code is listing on the following pages.

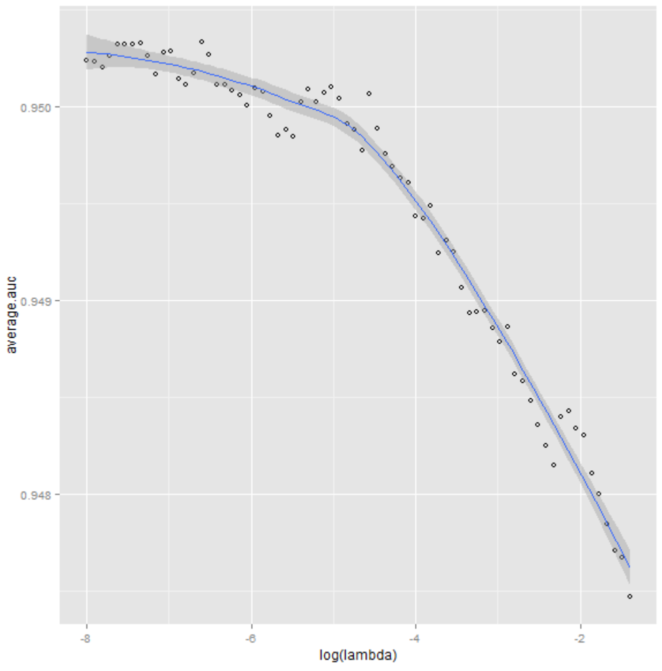


Figure 4.1: The scatter plot of AUC against the log of lambdas in logistic regression. There are totally 72 lambdas produced. From the plot, we could see that the here is a tend of decresing, and therefore we should select the largest lambda, which is 0.2483.

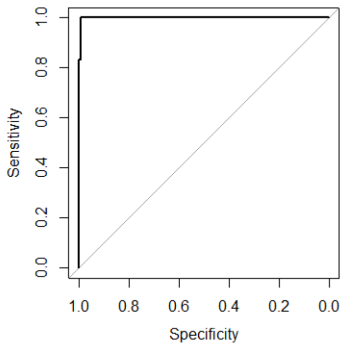


Figure 4.2: The ROC curve produced by pROC package. The auc is 0.9983.

```r
# Clearing memory, loading package, reading the data
rm(list=ls())
library(glmnet)
library(pROC)
library(ggplot2)
data.tr <- dget("0-training-data.r")
data.te <- dget("0-testing-data.r")

# -------------------- Choosing value of lambda -------------------- #

# Initializing the parameter tuning
nIter <- 10
# Doing variable selection using lasso, get lambdas
y.tr <- data.tr[,1]
x.tr <- data.matrix(data.tr[,-1])
model <- glmnet(x=x.tr, y=y.tr, family = "binomial", alpha=1)
lambda <- model$lambda
auc <- matrix(NA, nIter, length(lambda))

# Using ridge regression to tune the parameter
for(iter in 1:nIter){
  # Initializing the cross validation
  data.index <- sample(nrow(data.tr))[1:round(nrow(data.tr)/10)]
  y.tr <- data.tr[-data.index,1]
  x.tr <- data.matrix(data.tr[-data.index,-1])
  y.te <- data.tr[data.index,1]
  x.te <- data.matrix(data.tr[data.index,-1])
  for(l in lambda){
    model <- glmnet(x=x.tr, y=y.tr, family = "binomial", alpha=0, lambda=l)
    yHat <- predict(model, newx=x.te, type="response")
    roc <- roc(data.tr[data.index,1]~yHat, data.tr[-data.index,])
    roc$auc
    auc[iter,which(lambda==l)] <- roc$auc
  }
  print(iter)
}
data.plot <- data.frame(lambda, apply(auc,2,mean))
names(data.plot) <- c("lambda", "average.auc")
ggplot(data.plot, aes(x=log(lambda), y=average.auc)) +
  geom_point(shape=1) + geom_smooth()
```

6740–t–exam–3–code/2–logistic–1.R

The roc curve of the testing data set is shown on the previous page. The testing auc is 0.9983. The code is listed on the next page.

```
# -------------------- Testing model -------------------- #

y.tr <- data.tr[,1]
x.tr <- data.matrix(data.tr[,-1])
y.te <- data.te[,1]
x.te <- data.matrix(data.te[,-1])

# Fitting the final model, getting the prediction and testing the model
model <- glmnet(x=x.tr, y=y.tr, family = "binomial", alpha=0, lambda=0.2482957578)
dput(model, "0-model-logistic.R")

> model$beta
11 x 1 sparse Matrix of class "dgCMatrix"
s0
AOD750     -0.3185054961
TISA750    -0.6526404782
IT750       0.1709290159
IT2000      0.0987614727
ITCM        0.1401309094
IAREA       0.0077074521
ICURV       0.3179336602
ACW_mm     -0.0698256028
ACA        -0.0181303345
ACV        -0.0020997805
LENSVAULT   0.0002093944

yHat <- predict(model, newx=x.te, type="response")
roc <- roc(data.te[,1]~yHat, data.te, plot=T)

> roc$auc
Area under the curve: 0.9983
```

6740-t-exam-3-code/2-logistic-2.R

5 Support Vector Machine

There are two parameters that need to be tuned in svm: *gamma* and *cost*. Therefore we use a grid search to find the best combination of the parameters, by estimating the AUC by doing cross validation. We show different AUC's in different colors and plot them in a matrix to pick out the best. The plot is shown as below.

From the plot, we could see that we just need to avoid the write cross line to get a good combination of parameters. Here we take the 26th gamma and the last cost (There is a tiny darker spot, which might not be seen clearly on the plot). That is, gamma=0.0001, cost=1. Another reasonable choice for gamma and cost is gamma=1.00e-09, cost=1.00e-09.

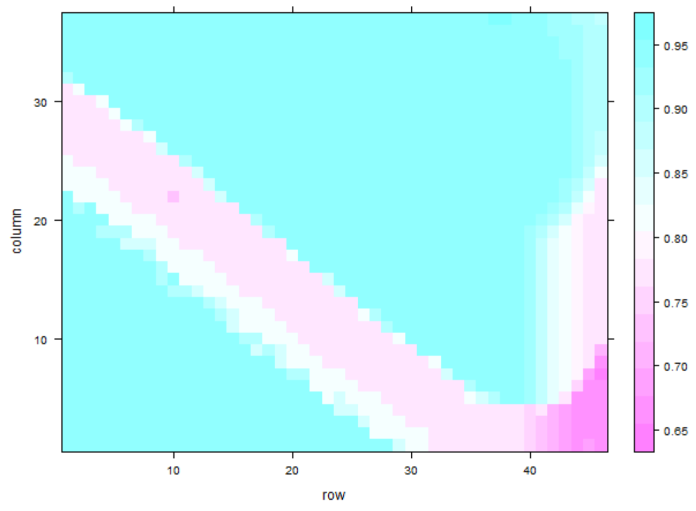


Figure 5.1: The grid plot of AUC against the gamma and the cost. From the plot, we could see that we just need to avoid the write cross line to get a good combination of parameters. Here we take the 26th gamma and the last cost (There is a tiny darker spot, which might not be seen clearly on the plot). That is, gamma=0.0001, cost=1.

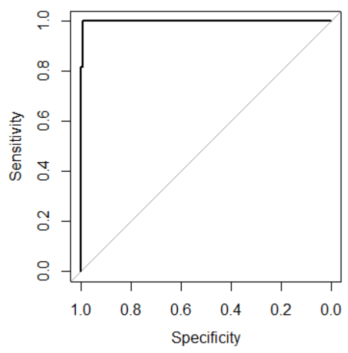


Figure 5.2: The ROC curve produced by pROC package. The auc is 0.9983.

The roc curve of the testing data set is shown below. The testing auc is 0.9983. The code is listed on the next page.

```r
# Clearing memory, loading package, reading the data
rm(list=ls())
library(e1071)
library(pROC)
library(lattice)
data.tr <- dget("0-training-data.r")
data.te <- dget("0-testing-data.r")

# Initializing values
nIter <- 10
gamma <- 10^seq(-9,0,0.2)
cost <- 10^seq(-9,0,0.25)
my.auc <- array(0, dim=c(nIter,length(gamma),length(cost)))

# --------------------- Choosing gamma and cost --------------------- #

for(iter in 1:nIter){
  # Initializing the cross validation
  data.index <- sample(nrow(data.tr))[1:round(nrow(data.tr)/10)]
  for(g in gamma) { for(c in cost) {
    model  <- svm(y~., data=data.tr[-data.index,], gamma=g, cost=c, probability=TRUE)
    yHat <- predict(model, data.tr[data.index,], probability=TRUE)
    yHat <- attr(yHat, "probabilities")
    roc <- roc(data.tr[data.index,1]~yHat[,1], data.tr[data.index,])
    my.auc[iter,which(gamma==g),which(cost==c)] <- roc$auc
  }}
  print(iter)
}
test <- apply(my.auc,c(2,3),mean)
levelplot(test)

# --------------------- Testing model --------------------- #

# Fitting the final model
model  <- svm(y~., data=data.tr, gamma=0.0001, cost=1, probability=TRUE)
dput(model, "0-model-svm.R")
# Getting the prediction
yHat <- predict(model, data.te, probability=TRUE)
yHat <- attr(yHat, "probabilities")
# Computing the ROC and AUC
roc <- roc(data.te[,1]~yHat[,1], data.te, plot=T)

> roc$auc
Area under the curve: 0.9983
```

6740–t–exam–3–code/2–svm.R

6 Neural Network

Again we use a grid search to tune the two parameters for neural network: the size and the decay. From the plot, we could see that we just need to select the one on the left bottom conner. That is, size=6 and decay=0.5.

The neural network method performs well just as the other methods when doing cross validation. Yet it fails to do a good seperate for the tesing data. The roc curve of the testing data set is shown below. The testing auc is 0.9287. The code is listed on the next page.

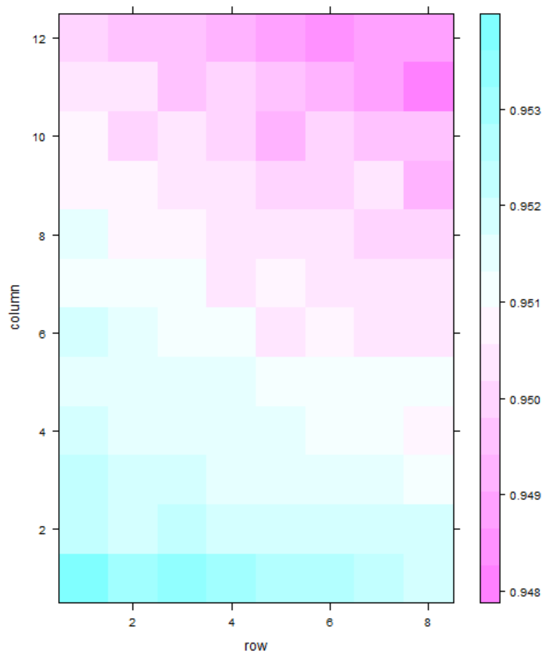


Figure 6.1: The grid plot of AUC against the size and the decay. From the plot, we could see that we just need to select the one on the left bottom conner. That is, size=6 and decay=0.5.

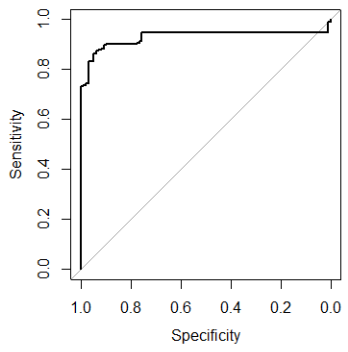


Figure 6.2: The ROC curve produced by pROC package. The auc is 0.9287.

```r
# Clearing memory, loading package, reading the data
rm(list=ls())
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw
           /466c1474d0a505ff044412703516c34f1a4684a5/nnet_plot_update.r')
source_url('https://gist.github.com/fawda123/6206737/raw
           /2e1bc9cbc48d1a56d2a79dd1d33f414213f5f1b1/gar_fun.r')
library(nnet)
library(devtools)
library(pROC)
library(lattice)
data.tr <- dget("0-training-data.r")
data.te <- dget("0-testing-data.r")

# -------------------- Choosing size and decay -------------------- #

# Initializing values
nIter <- 50
size <- seq(6,27,3)
decay <- seq(0.5,6.0,0.5)
#   seq(0.2,2.0,0.2)
my.auc <- array(0, dim=c(nIter,length(size),length(decay)))

for(iter in 1:nIter){
  # Initializing the cross validation
  data.index <- sample(nrow(data.tr))[1:round(nrow(data.tr)/10)]
  for(s in size) { for(d in decay) {
    model <- nnet(y~., data=data.tr[-data.index,], size=s, decay=d)
    yHat <- predict(model, data.tr[data.index,], type="raw")
    roc <- roc(data.tr[data.index,1]~yHat, data.tr[data.index,])
    my.auc[iter,which(size==s),which(decay==d)] <- roc$auc
  }}
  print(iter)
}
test <- apply(my.auc,c(2,3),mean)
levelplot(test)

# Fitting the final model
model  <- nnet(y~., data=data.tr, size=6, decay=0.5)
# Getting the prediction
yHat <- predict(model, data.te, type="raw")
# Computing the ROC and AUC
roc <- roc(data.te[,1]~yHat, data.te, plot=T)

> roc$auc
Area under the curve: 0.9287
```

6740-t-exam-3-code/2-neural-net.R

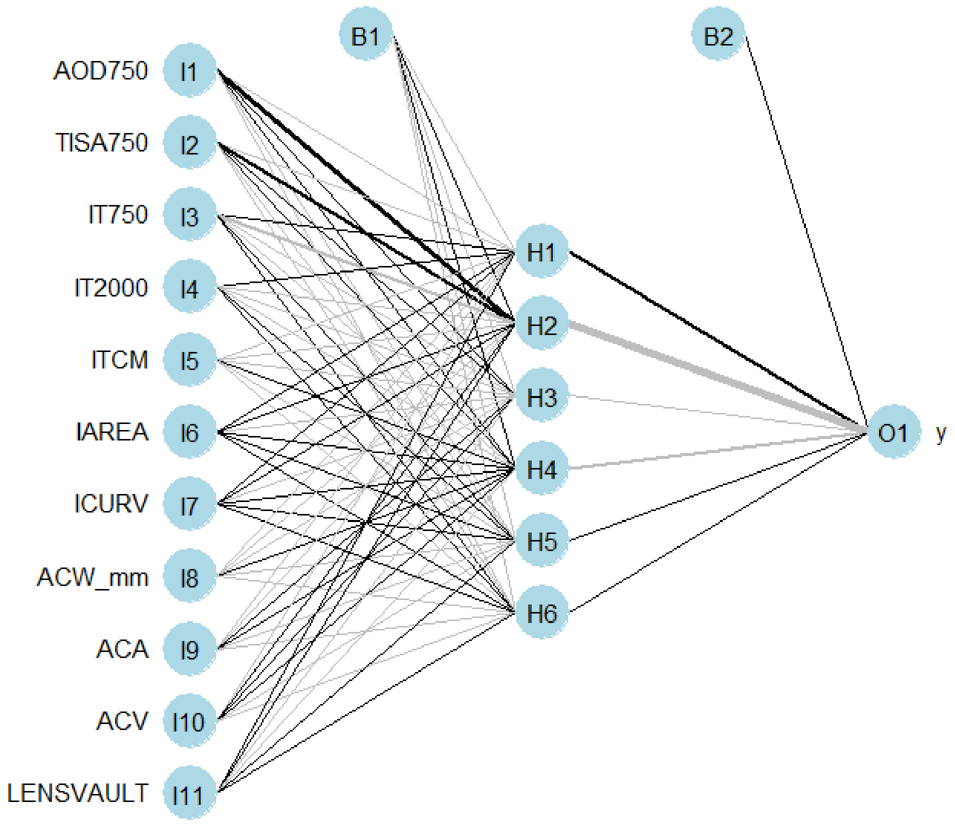


Figure 6.3: The visualization of the neural network