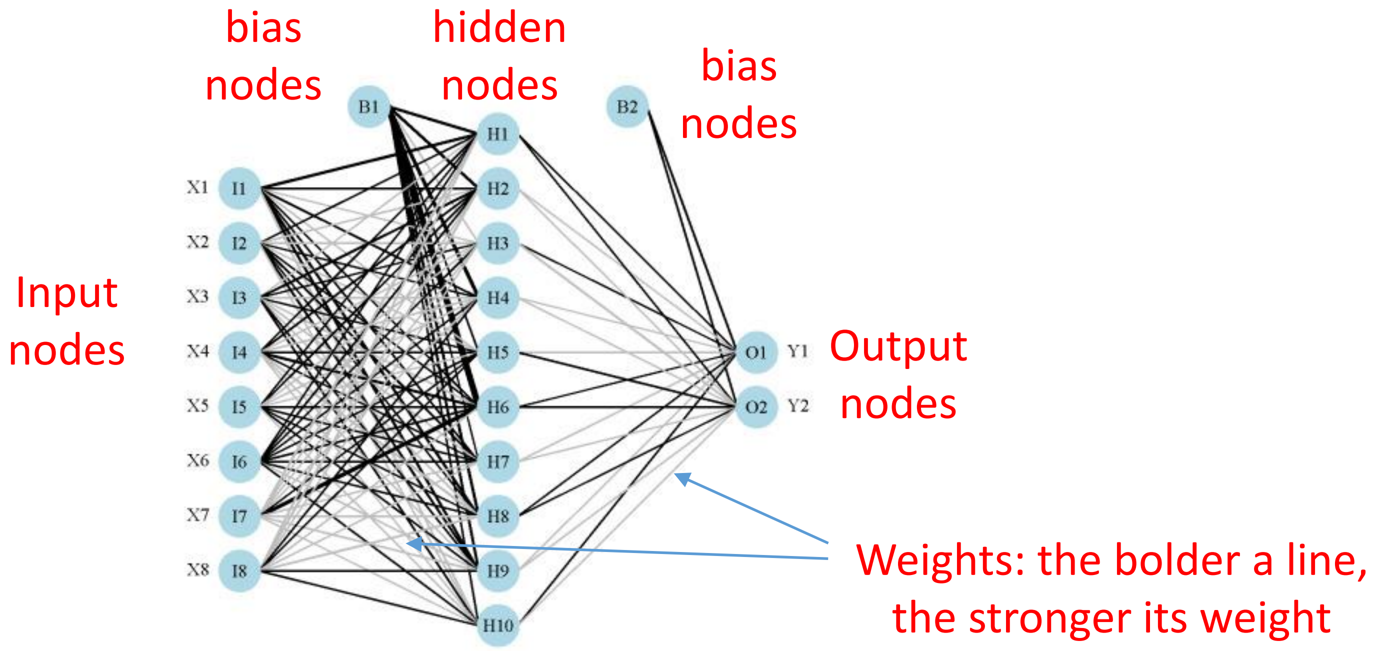
RDataMining

Week 6 Lab in R

Dr. Liang (Leon) Chen

Instructions

- R script for Neural Network classifier is provided in the next slide, please follow it and complete the lab in R.
- You do not need to type notes (starting at #), but it’s a good manner to have them in you code.
- In order to see codes and notes clearly, I show the script in RStudio.



Neural Network Classifier

Here, we just develop a neural network model without making any prediction; of course, you can use the function predict() with a class output to put predictions in a new data frame, just as you did in previous weeks.

```r
#define and choose the dataset
Lab6Data<-read.csv(file.choose(),header=T)
#view the dimensions of the dataset
dim(Lab6Data)
#show the top few rows to see view the data
head(Lab6Data)
#remove the first column id
Lab6Data[1]<-NULL
#install library dplyr
install.packages('dplyr')
library(dplyr)
library(reshape2)
#generate correlation coeffcients matrix
Lab6cor<-as.matrix(cor(Lab6Data))
#melt correlation coefficient matrix to an arrange and sort by its absolute value
#based on this arrange, you can find the largest correlation coeffients
Lab6cormelt <- arrange(melt(Lab6cor), -abs(value))
Lab6cormelt
#assign the self-correlation and upper part of the correlation matrix as zero
Lab6cor[!lower.tri(Lab6cor)] <- 0
#remove highly-correlated variables, with correlation coefficients greater than 0.8
Lab6Data2<-Lab6Data[,!apply(Lab6cor,2,function(x) any(abs(x) > 0.8))]
#check the attribute names of the new and clean dataset
names(Lab6Data2)
#install the package nnet for neural network
install.packages("nnet")
library(nnet)
#set the seed to make sure you can get the same result as mine; of course, you can change the seed number later.
set.seed(1000)
#build a neural network model using Phone_sale as target attribute and other attribtues as predictor attributes.
#the size parameter indicates the number of nodes we wish to use the hidden layer
#the maxit parameter indicates the maximum iterations.
Lab6Net<-nnet(Phone_sale ~., data=Lab6Data2, size=8,maxit=10000)
#install the package NeuralNetTools
install.packages("NeuralNetTools")
library(NeuralNetTools)
#plot the nueral network model
plotnet(Lab6Net)
```

For details, please click dplyr and reshape2; If you do not have the reshape2 library, please install it using the install.packages()

Data cleaning: remove the first column (ID) and then remove highly correlated attribute(s)
If your arrange() does not work, you can skip it because it does not impact your result.

Absolute value of correlation coefficients

For details about the package nnet, please click here

For details about the package NeuralNetworkTools, please click here

Model Evaluation in R: Using Logistic Regression Model as an example

```
#loading required libraries
library("e1071")
library("caret")
#set random seed to make the sampling reproduciable
set.seed(123)
smp_size <- floor(0.7 * nrow(Lab6Data2))
train_ind <- sample(seq_len(nrow(Lab6Data2)), size = smp_size)
train <- Lab6Data2[train_ind, ]
test <- Lab6Data2[-train_ind, ]
#check the ratio of train set
nrow(train)/nrow(Lab6Data2)
#build a logistic regression model using the train set
LRmodel<-glm(Phone_sale ~., family = "binomial", train)
#apply the model to the test set; probabilities are generated
LRp<-predict(LRmodel, test, type = "response")
#check the summary of those probabilities
summary(LRp)
#because the probabilities are quite small, we reduce the threshold to 0.2
LRpredict<-ifelse(LRp > 0.20, 1, 0)
#generate a simple confusion matrix using the table function
table(LRpredict, test[["Phone_sale"]])

#alternatively we can use confusionMatrix function to get more details.
#check the type of both LRpredict and Phone_sale
typeof(LRpredict)
typeof(test[["Phone_sale"]])
#The confusionMatrix function requires factors with the same level
LRp_class<-as.factor(LRpredict)
confusionMatrix(LRp_class, as.factor(test[["Phone_sale"]]))
```

Data Partition: There are various methods for this purpose. Here is an example

The confusionMatrix function requires library caret; if you encounter a problem with that, you can use table() to generate a simple confusion matrix

Some outputs

Simple confusion matrix using the table function

```
LRpredict    0    1
        0 1148  158
        1  151   39
```

As mentioned in the appendix in our RM instruction, performance measures except accuracy will be different when choosing different positive cases. In this confusion matrix, the default positive case is 0. In order to generate a confusion matrix with 1 as the positive case, we have specify this argument positive = “1”.

Accuracy = (1148+39)/1496=79.34%

Specificity =

Sensitivity =

Complete confusion matrix using the confusionMatrix function

```
confusionMatrix(LRp_class, as.factor(test[["Phone_sale"]]))
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1148  158
         1  151   39

               Accuracy : 0.7934
                 95% CI : (0.772, 0.8137)
    No Information Rate : 0.8683
    P-Value [Acc > NIR] : 1.0000

                  Kappa : 0.083

 Mcnemar's Test P-Value : 0.7329

            Sensitivity : 0.8838
            Specificity : 0.1980
         Pos Pred Value : 0.8790
         Neg Pred Value : 0.2053
             Prevalence : 0.8683
         Detection Rate : 0.7674
   Detection Prevalence : 0.8730
      Balanced Accuracy : 0.5409

       'Positive' Class : 0
```

```
> confusionMatrix(LRp_class, as.factor(test[["Phone_sale"]]), positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1148  158
         1  151   39

               Accuracy : 0.7934
                 95% CI : (0.772, 0.8137)
    No Information Rate : 0.8683
    P-Value [Acc > NIR] : 1.0000

                  Kappa : 0.083

 Mcnemar's Test P-Value : 0.7329

            Sensitivity : 0.19797
            Specificity : 0.88376
         Pos Pred Value : 0.20526
         Neg Pred Value : 0.87902
             Prevalence : 0.13168
         Detection Rate : 0.02607
   Detection Prevalence : 0.12701
      Balanced Accuracy : 0.54086

       'Positive' Class : 1
```

Accuracy = (1148+39)/1496=79.34%

Recall/Sensitivity = 39/(39+258)=19.80%

Specificity= 1148/(1148+151)=88.38%

Precision = 39/(39+151)=20.53%