ASDM Workshop 9 - Artificial Neural Networks (ANN)

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way of biological nervous systems, such as the brain. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by examples. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. In this workshop we will practice how to implement Artificial Neural Networks using R.

For more details please read article by (Günther and Fritsch, 2010)

URL : https://journal.r-project.org/archive/2010/RJ-2010-006/RJ-2010-006.pdf

Exercise 1:

The dataset, `creditrisk.csv` that we will use in the workshop can be downloaded from Blackboard. We will be using credit risk data to predict whether a **default** will occur within 10 years using 4 input variables, `income`, `age`, `loan` and `LTIR` and 1 class variable `default10yr`.

Data Explanation

clientid	Id of the client
income	Income per year
age	Age of the client
loan	Loan in £.
LTIR	Loan to yearly income ratio
default10yr	Default,1; Not Default 0

Implementation:

We can use different packages available in R to classify our data using artificial neural networks. In this workshop, we will be using "**neuralnet**" package which use a direct adaptive method for faster back propagation learning: The RPROP algorithm (Riedmiller and Braun, 1993) to classify our data based on our input and class variables.

1. Download `creditrisk.csv` dataset from Blackboard and save it to a folder on your F: drive. Open it using excel to get a rough idea about the dataset, e.g, attributes and their values.

2. Start RStudio.

3. Change the working directory

   `File → More → Go To Working Directory…`

   In the `Go To Working Directory` dialogue, navigate and select the folder where you saved your data file eg: `F:\ASDM\Week9`. Click OK.

   `Click Set as Working Directory option`

   or

   using R commands as follow:

```
mypath = "F:\\ASDM\Week9"     # you need to change the string to
                               your directory
setwd(mypath)                 # set working directory
getwd()                       # check if the working directory has
                               changed correctly
```

4. Open a new R script window:

   `File → New File → R script`

5. Read the data file

```
creditrisk_data <- read.csv("creditrisk.csv", header= TRUE)
```

   The data is read into the data frame named `"creditrisk_data"`

6. Inspect the dataset in R

   Once the file has been imported to R, we often want to do few things to explore the dataset:

```
names(creditrisk_data)

head(creditrisk_data)

tail(creditrisk_data)
```

```
> head(creditrisk_data)
  clientid      income         age         loan        LTIR default10yr
1        1 66155.92510 59.01701507 8106.53213100 0.122536751           0
2        2 34415.15397 48.11715310 6564.74501800 0.190751581           0
3        3 57317.17006 63.10804949 8020.95329600 0.139939800           0
4        4 42709.53420 45.75197235 6103.64226000 0.142910532           0
5        5 66952.68885 18.58433593 8770.09923500 0.130989500           1
6        6 24904.06414 57.47160710   15.49859844 0.000622332           0
> tail(creditrisk_data)
     clientid      income         age        loan        LTIR default10yr
1995     1995 24254.70079 37.75162224 2225.284643 0.091746530           0
1996     1996 59221.04487 48.51817941 1926.729397 0.032534539           0
1997     1997 69516.12757 23.16210447 3503.176156 0.050393718           0
1998     1998 44311.44926 28.01716690 5522.786693 0.124635659           1
1999     1999 43756.05660 63.97179584 1622.722598 0.037085668           0
2000     2000 69436.57955 56.15261703 7378.833599 0.106267239           0
```

```
summary(creditrisk_data)
str(creditrisk_data)
```

```
> summary(creditrisk_data)
    clientid         income           age             loan                LTIR              default10yr
 Min.   :   1.0   Min.   :20014   Min.   :18.06   Min.   :    1.378   Min.   :0.0000491   Min.   :0.0000
 1st Qu.: 500.8   1st Qu.:32796   1st Qu.:29.06   1st Qu.: 1939.709   1st Qu.:0.0479035   1st Qu.:0.0000
 Median :1000.5   Median :45789   Median :41.38   Median : 3974.719   Median :0.0994365   Median :0.0000
 Mean   :1000.5   Mean   :45332   Mean   :40.93   Mean   : 4444.370   Mean   :0.0984028   Mean   :0.1415
 3rd Qu.:1500.2   3rd Qu.:57791   3rd Qu.:52.60   3rd Qu.: 6432.411   3rd Qu.:0.1475846   3rd Qu.:0.0000
 Max.   :2000.0   Max.   :69996   Max.   :63.97   Max.   :13766.051   Max.   :0.1999377   Max.   :1.0000
> str(creditrisk_data)
'data.frame':   2000 obs. of  6 variables:
 $ clientid   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ income     : num  66156 34415 57317 42710 66953 ...
 $ age        : num  59 48.1 63.1 45.8 18.6 ...
 $ loan       : num  8107 6565 8021 6104 8770 ...
 $ LTIR       : num  0.123 0.191 0.14 0.143 0.131 ...
 $ default10yr: int  0 0 0 0 1 0 0 1 0 0 ...
```

7. Check the dimension and number of points of the "`creditrisk_data`" dataset

```
nrow(creditrisk_data)
ncol(creditrisk_data)
dim(creditrisk_data)
```

```
> dim(creditrisk_data)
[1] 2000    6
```

8. Now we will be specifying our training and test data from the original dataset. Small number of observations need to be used to just test the results.

   Below code will divide the dataset into 60% for training and 40% for test.

```
# set.seed function in R is used to reproduce results i.e.  it produces the same sample again and again.

set.seed(1234)


#sample function can be used to return a random permutation of a vector.

pd <- sample(2, nrow(creditrisk_data), replace=TRUE,  prob=c(0.6,0.4))

trainingdata <- creditrisk_data[pd==1,]

testdata <- creditrisk_data[pd==2,]
```

9. Run below code to check how many observations are in training and test data sets .

```
dim(trainingdata) # Retrieve the dimension of the train data set

dim(testdata) # Retrieve the dimension of the test data set
```

```
> dim(trainingdata) # Retrieve the dimension of the train data set
[1] 1213    6
> dim(testdata) # Retrieve the dimension of the validate data set
[1] 787   6
>
```

10. Since we have training and test datasets, now we can build Neural Networks (NN) with `neuralnet` R package.

neuralnet R package documentation can be downloaded from below URL.
https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

```
#install “neuralnet” package. Ignore this line if “neuralnet”
was already installed.

install.packages(“neuralnet”)


#activate “neuralnet” package

library(neuralnet)
```

11. Train the Neural Network (NN).

Note: *for illustration purpose, in this exercise we only use two input variables,* `LTIR and AGE` *to predict the class variable* `default10yr`.

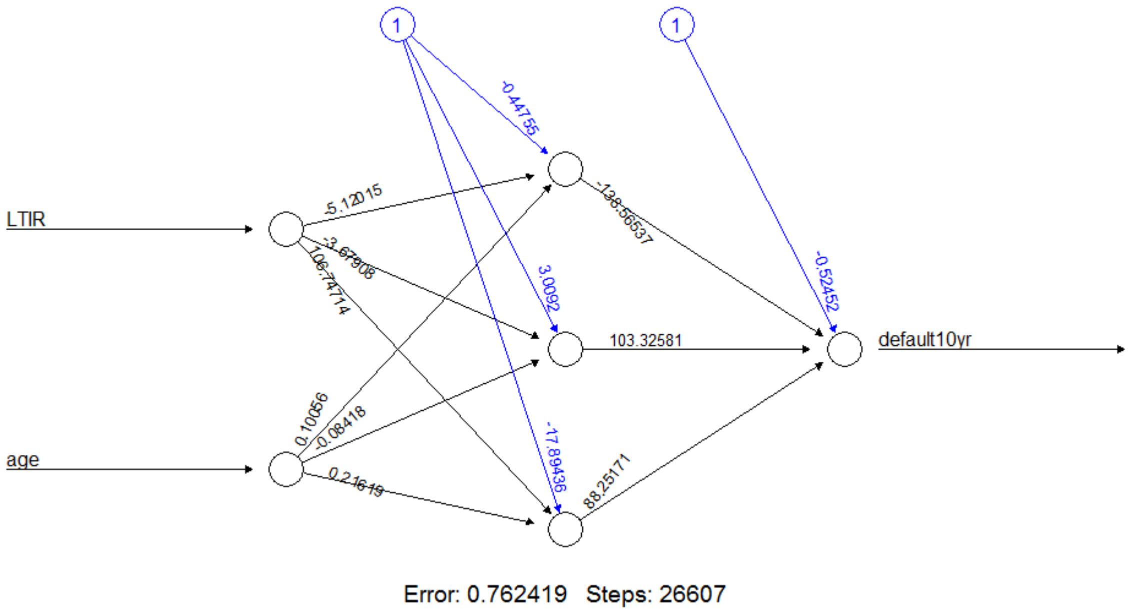
```
#build the neural network (NN) with “neuralnet” package

#neuralnet() function is used to train neural networks

#formula is a symbolic description of the model to be fitted.
#In this case formula is : default10yr ~ LTIR+age

#hidden argument is a vector of integers specifying the number of
hidden neurons (vertices) in each layer.

#lifesign argument is a string specifying how much the function will
print during the calculation of the neural network. 'none',
'minimal' or 'full'.

#linear.output is used to specify whether we want to do regression
linear.output=TRUE or classification linear.output=FALSE

#threshold is a numeric value specifying the threshold for the
partial derivatives of the error function as stopping criteria.
```

```
creditnet <- neuralnet(
               default10yr ~ LTIR+age, trainingdata,
               hidden=3,
               lifesign="minimal",
               linear.output=FALSE,
               threshold=0.1)
```

```
creditnet <- neuralnet(default10yr ~ LTIR+age, trainingdata, hidden=3, lifesign="minimal", linear.output=FALSE, threshold=0.1)
```

```
plot(creditnet, rep = "best")
```

Result



Note: *The mean squad error can be different every time you run NN.*

12. To test resulting output accuracy, use the following code.

```
#subset() function return subsets of vectors, matrices or data
frames which meet conditions.

#Select only LTIR and age inputs

temp_test <- subset(testdata, select = c("LTIR", "age"))
```

```
head(temp_test)

#compute() function computes the outputs of all neurons for
specific arbitrary covariate vectors given a trained neural
network.

creditnet_results <- compute(creditnet, temp_test)

names(creditnet_results)
str(creditnet_results)
```

```
> creditnet_results <- compute(creditnet, temp_test) # computes the results of each case
> names(creditnet_results)
[1] "neurons"    "net.result"
> str(creditnet_results)
List of 2
 $ neurons   :List of 2
  ..$ : num [1:787, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:787] "2" "3" "4" "5" ...
  .. .. ..$ : chr [1:3] "1" "LTIR" "age"
  ..$ : num [1:787, 1:4] 1 1 1 1 1 1 1 1 1 1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:787] "2" "3" "4" "5" ...
  .. .. ..$ : NULL
 $ net.result: num [1:787, 1] 0.000000000000033183544904141916869790007726948033450753 4 0
363667161941869210295 4 0.00000000000804026372508114775365661852646326224203221 5 0.9999999
4736328125 0.000000000000000000000000000000000000000000000000018 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:787] "2" "3" "4" "5" ...
  .. ..$ : NULL
>
```

13. Use following code to check the accuracy of each case:

```
results <-data.frame(actual = testdata$default10yr,
                     prediction = creditnet_results$net.result)

head(results)
tail(results)

results[90:105, ]      # show part of the result
```

Result:

```
> results[90:105, ]                # show part of the result
    actual                                                  prediction
245      1 0.9998916493854772546967524249339476227760314941406250000
247      0 0.00000000000000000000002113979235963096632100813310195520
249      1 0.9999999999999997335464740899624302983283996582031250000
250      0 0.000000000000000000037069122956658492136407939687714474530
261      0 0.0000000000000000000071020018240519263139944039497919448
267      0 0.0000000000000000000000000000000000000000000002362727383
270      0 0.000000000000000000730944452016047714268398571846319100587
272      1 1.0000000000000000000000000000000000000000000000000000000
274      1 0.99999999998332689266078432410722598433494567871093750000
277      0 0.00000000000000000000000000000000000000000001103047398443
283      0 0.000000000000000000419219748390422729516363586910188132606
284      0 0.00000000000000000007968566399559549563117594184191716522
285      0 0.00000000000000000000000049593800080324227578902618329693
289      0 0.0000000000000000000000000000000000000000000009908804307363
290      0 0.00000000002499649010171587380346042150947027948859613389
291      0 0.000000000000000000000000000000000000000000343787386631897
```

14. Round the results to understand prediction accuracy

```
#sapply() function takes list or vector as input and
#applies function to each element of the list or vector
#always tries to return a vector or matrix; but if not
possible, will return a list

results$prediction <- sapply(creditnet_results$net.result,
                             round, digits=0)


results[90:105, ]
```

```
> results[90:105, ]
    actual prediction
245      1          1
247      0          0
249      1          1
250      0          0
261      0          0
267      0          0
270      0          0
272      1          1
274      1          1
277      0          0
283      0          0
284      0          0
285      0          0
289      0          0
290      0          0
291      0          0
```

15. Build confusion matrix for prediction accuracy

```
#What is a Confusion Matrix?

#A confusion matrix is a summary of prediction results on a
classification problem.

#The number of correct and incorrect predictions are summarized with
count values and broken down by each class.


confusionmatrix <- table(testdata$default10yr,
                         results$prediction)
```

use `print(confusionmatrix)` command to see the output.

```
> confusionmatrix <- table(testdata$default10yr, results$prediction)
> print(confusionmatrix)

      0   1
  0 666   0
  1   2 119
```

Calculate classification accuracy

```
sum(diag(confusionmatrix))/sum(confusionmatrix)
```

```
> sum(diag(confusionmatrix))/sum(confusionmatrix)
[1] 0.9974587039
```

This result shows that our classification is 99.75% accurate.

Or

equivalently classification error is 0.25%.

```
1-sum(diag(confusionmatrix))/sum(confusionmatrix)
```

```
> 1-sum(diag(confusionmatrix))/sum(confusionmatrix)
[1] 0.002541296061
```

16. You can also use other available R packages to build models in ANN.
    For example: nnet, RSNNS and caret.

Exercise 2:

Use same **creditrisk.csv** data set and build confusion matrix using Artificial Neural Networks (ANN) algorithm to predict classification accuracy using all 4 input variables eg: income, age, loan and LTIR. In this case also divide the dataset into 60% for training and 40% for test.

Exercise 3:

Repeat the exercise 2 with the normalised data and compare the accuracy percentages with exercise 2 classification accuracy.

Exercise 4:

Repeat the exercise 2 and 3, in this case divide the dataset into 80% for training set and 15% for test set 1 and 5% for test set 2. Compare the accuracy percentages with test set 1 and test set 2.

Part 2: Exercises

Exercise 1 :

This exercise aims to detect a Fetal State from the readings of BPM, APC, FMPS, UCPS, DLPS, SDPS and PDPS from the CTG monitoring records, so automatic Fetal State detection system can be developed. The data set is called **Cardiotocographic.csv** and it can be downloaded from Blackboard Week 4 folder or below mentioned URL.

CTG monitoring is widely used to assess fetal wellbeing. Cardiotocography (CTG) is a technical means of recording the fetal heartbeat and the uterine contractions during pregnancy. The machine used to perform the monitoring is called a cardiotocograph, more commonly known as an electronic fetal monitor (EFM).

Data Set Information:

2126 fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them (Eg: fetal state N, S, P).

Link : https://archive.ics.uci.edu/ml/datasets/cardiotocography

BPM	Beat per minutes
APC	Accelerations per second
FMPS	Fetal movement per second
UCPS	Uterine contractions per second
DLPS	Light declaration per second
SDPS	Severe declaration per second
PDPS	Prolonged declaration per second
NSP	Fetal State Class code N=normal (1); S=Suspect (2); P=Pathologic (3)

1. Calculate classification accuracy percentage using K-Nearest Neighbour algorithm.

2. Calculate classification accuracy using Decision Tree algorithm.

3. Calculate classification accuracy using Artificial Neural Network (ANN) algorithm.

4. Compare the three set of classification accuracy percentages and find the best algorithm to develop an automatic Fetal State detection system.

Exercise 2:

Repeat the exercise 1 with the normalised data and compare the accuracy percentages.