INFO523: R Exercise

Week 10 Chapter 8 Classification R Exercise: ANN

Goal: Practice basic R commands/methods for classification. If you are already familiar with some of the commands/methods, you can just practice the ones that are new to you.

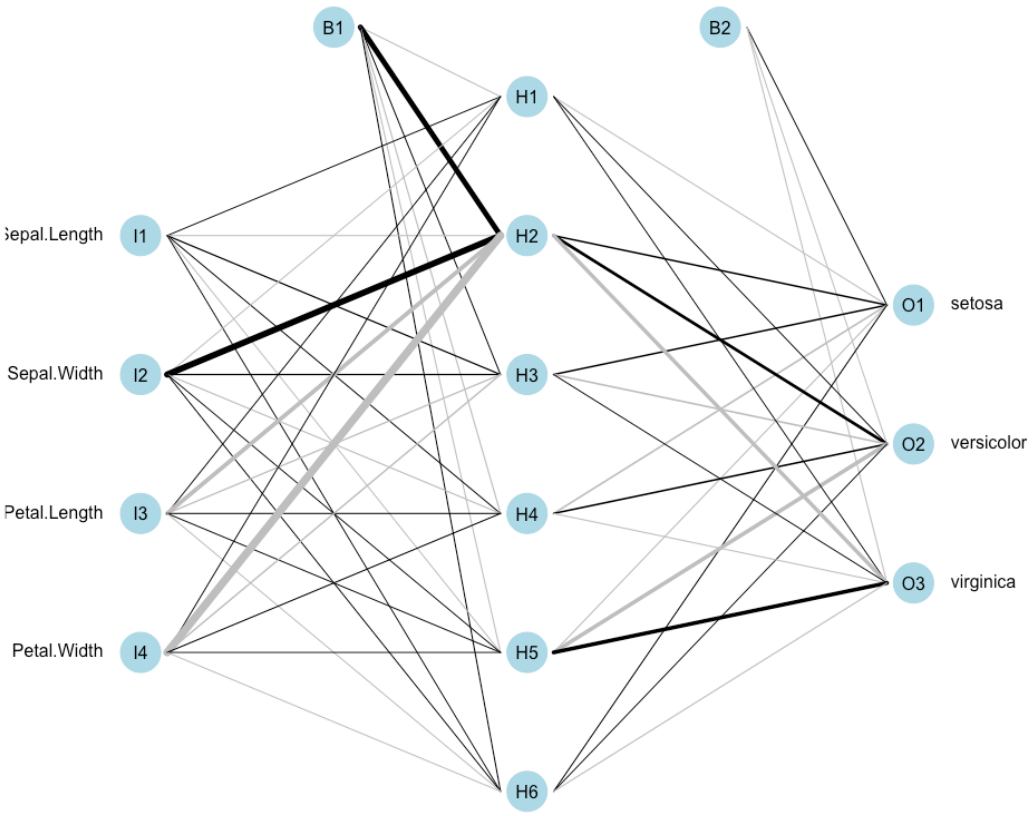
*> # Blue text are explanations.
*> Blue text are R commands/methods your will type into an R console.
*Black and Red text are output from R

```
> #libraries used in this exercise
>
> install.packages("neuralnet")
> install.packages("NeuralNetTools")
> install.packages("h2o")
>
> library(nnet) #base R package, support feed-forward NN with one hidden layer.
> library(h2o) # We use it to demonstrate "deep" learning (feed-forward multilayer ANN),
but it contains a lot more. It is the R interface for 'H2O', the scalable open source
machine learning
> #platform that offers parallelized implementations of many supervised and
> #unsupervised machine learning algorithms such as Generalized Linear
> #Models, Gradient Boosting Machines (including XGBoost), Random Forests,
> #Deep Neural Networks (Deep Learning), Stacked Ensembles, Naive Bayes, Cox
> #Proportional Hazards, K-Means, PCA, Word2Vec, as well as a fully automatic
> #machine learning algorithm (AutoML).
>
> library(NeuralNetTools) # for exploration of NN models
> library(ggplot2) # for visualization of NN
>
> # Other ANN packages: RSNNS, FCNN4R, neuralnet,
> # and more recent deep net packages include mxnet, darch, deepnet.
>
> # NNs with a few hidden layers are traditional NNs.
> # Deep Nets often have more layers (2 and above). The more layers, the more weights to
learn, and often the more training examples are required.
>
> # ANNs can be used for classification and for regression
>
> # classify iris data into species, using sepal and petal sizes
> data(iris)
>
> # min-max scaling on each of the numeric attributes
> # Students: try with and without scaling and observe performance difference
```

```
> (iris$Sepal.Length <- as.vector(scale(iris$Sepal.Length, center=min(iris$Sepal.Length),
scale=max(iris$Sepal.Length)-min(iris$Sepal.Length))))
 [1] 0.22222222 0.16666667 0.11111111 0.08333333 0.19444444 0.30555556 0.08333333 0.19444444
0.02777778 0.16666667 0.30555556 0.13888889 0.13888889 0.00000000 0.41666667……
> (iris$Sepal.Width <- as.vector(scale(iris$Sepal.Width, center=min(iris$Sepal.Width),
scale=max(iris$Sepal.Width)-min(iris$Sepal.Width))))
……
> (iris$Petal.Length <- as.vector(scale(iris$Petal.Length, center=min(iris$Petal.Length),
scale=max(iris$Petal.Length)-min(iris$Petal.Length))))
…
> (iris$Petal.Width <- as.vector(scale(iris$Petal.Width, center=min(iris$Petal.Width),
scale=max(iris$Petal.Width)-min(iris$Petal.Width))))
>
>
>
> set.seed(1234)
> summary(iris)
 Sepal.Length     Sepal.Width      Petal.Length     Petal.Width           Species
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   setosa    :50
 1st Qu.:0.2222   1st Qu.:0.3333   1st Qu.:0.1017   1st Qu.:0.08333   versicolor:50
 Median :0.4167   Median :0.4167   Median :0.5678   Median :0.50000   virginica :50
 Mean   :0.4287   Mean   :0.4406   Mean   :0.4675   Mean   :0.45806
 3rd Qu.:0.5833   3rd Qu.:0.5417   3rd Qu.:0.6949   3rd Qu.:0.70833
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000

>
> #sample 100 rows as training examples and use the rest as test
> #we are not doing k-fold validation for this example, but will do that for later
examples
> #we are doing holdout for this example
> sample <- sample(1:nrow(iris), 100)
> train <-iris[sample, ]
> test <- iris[-sample,]
>
>
> #training an ann using nnet
> #arguments for nnet()
> #"Species ~ .": the formula that takes the form class ~ var1 + var2 + ....
> #We saw formula of this form before when doing log-linear modeling
> #In the formula, if 'class' is a factor, then ann will perform classification. If
'class' is a numerical variable, then the ann will perform regression
> #size: number of nodes in the hidden layer
> #trace: whether to output the process of the learning
> #maxit: max iterations to complete before stoping the training
> #the () that inclose everything will print the result (n) on the counsel
> set.seed(123)
> (n <- nnet(Species ~ ., train, size=6, trace=FALSE, maxit=1000))
a 4-6-3 network with 51 weights
```

```
inputs: Sepal.Length Sepal.Width Petal.Length Petal.Width
output(s): Species
options were - softmax modelling
>
> #4-6-3 network: 4= input layer nodes, number of input variables, 6= hidden layer nodes,
3=output layer nodes, three species/classes
> #use the model n to classify test examples
> #the below will output the class probabilities of each observation
> (predict <- predict(n, test))

          setosa    versicolor    virginica
1    9.999997e-01  2.921283e-07 0.000000e+00
7    9.999997e-01  2.862858e-07 0.000000e+00
11   9.999997e-01  2.848554e-07 0.000000e+00
12   9.999997e-01  2.819721e-07 0.000000e+00
15   9.999997e-01  2.955659e-07 0.000000e+00
16   9.999997e-01  2.524128e-07 0.000000e+00
18   9.999997e-01  2.878112e-07 0.000000e+00
…
>
> #contrast with:
> #use 'type = "class"' to get the predicted class labels
> (predict <- predict(n, test, type="class"))
[1] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
 [9] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
[17] "setosa"     "setosa"     "versicolor" "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[25] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor" "virginica"  "versicolor" "versicolor"
[33] "versicolor" "versicolor" "versicolor" "versicolor" "virginica"  "virginica"  "virginica"  "versicolor"
[41] "versicolor" "virginica"  "versicolor" "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
[49] "versicolor" "virginica"

>
> #compare predict with the truth in test
> #predict and ts$Species are two vectors
> #Calling table() on them creates a contingency table of the counts at each combination
of factor levels.
> (contingency <- table(predict, test$Species))

predict      setosa versicolor virginica
  setosa         18          0         0
  versicolor      0         17         4
  virginica       0          1        10

>
> #Compute accuracy = correct / total
> (accuracy <- sum(diag(contingency))/sum(contingency))
[1] 0.9
>
> #plotnet() plots the network architecture with the learned weights and bias for each
layer.
```

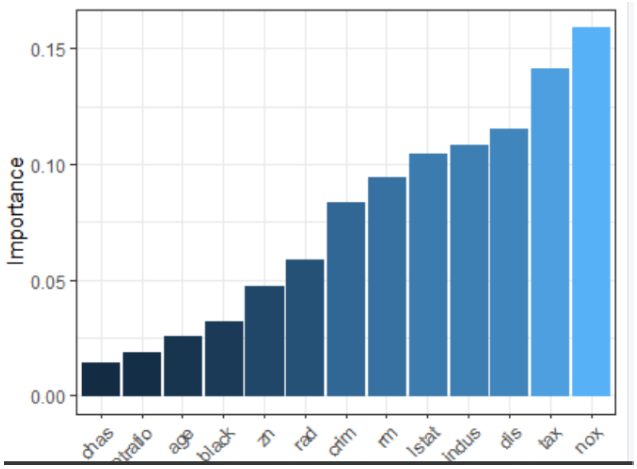
```
> #negative weights in grey, positives in black. The thickness of the line indicates the
relative mangnitude of the weights
> plotnet(n)
```



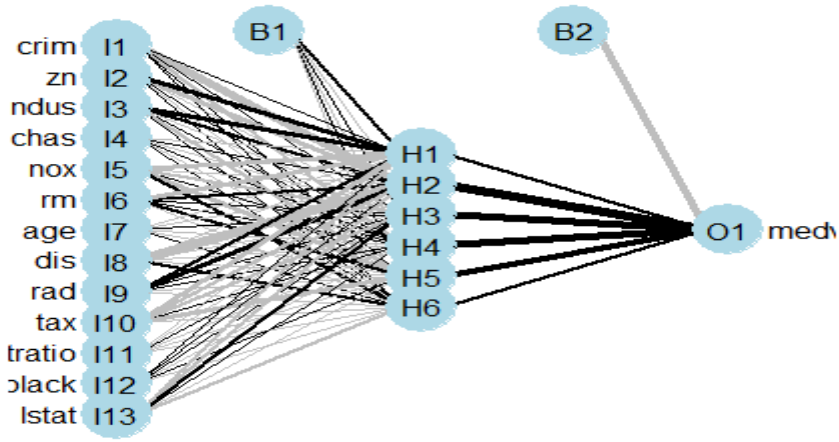

```
> #use h2o
> #h2o instances can be deploy on the cloud but we do it on local machine
> h2oinstance <- h2o.init(ip="localhost", startH2O = TRUE)
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         4 minutes 18 seconds
    H2O cluster timezone:       America/Phoenix
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.38.0.1
    H2O cluster version age:    1 month and 6 days
    H2O cluster name:           H2O_started_from_R_yanhan_hlm108
    H2O cluster total nodes:    1
    H2O cluster total memory:   1.76 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 4.2.1 (2022-06-23)
```

```
> #we need to convert our training and test examples to H2OFrame
> set.seed(1234)
> sample <- sample(1:nrow(iris), 100)
> #if run into column type error, check and make sure the class of the variables in iris
are either numeric or factor
> train.h2o <- as.h2o(as.data.frame(iris[sample, ]), destination_frame = "train.h2o")
  |=======================================================================| 100%
> test.h2o <- as.h2o(iris[-sample, ])
  |=======================================================================| 100%
> #many parameters can be set by the user. Here we do a 2-layer model, each with 100
nodes.
> #h2o doesn't take a formula, you have to use x and y to indicate your features (x) and
your class label attribute (y)
> model.h2o <- h2o.deeplearning(x=1:4, y=5, training_frame = train.h2o, hidden=c(100,100))
  |=======================================================================| 100%
> predict.h2o <- h2o.predict(model.h2o, test.h2o)[, "predict"]
  |=======================================================================| 100%
> (cm.h2o <-table(as.vector(predict.h2o), as.vector(test.h2o$Species)))

             setosa versicolor virginica
  setosa         18          0         0
  versicolor      0         17         1
  virginica       0          1        13

> #Let's do a regression using nnet ############################
> #for regression, ANN's output layer will have only one output node, which output the
numeric value predicted by the regression
>
> #use Boston housing data to predict property's med values
> data(Boston, package="MASS")
> #examine the variables in Boston dataset
> names(Boston)
 [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"     "dis"
 [9] "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
> lapply(Boston, class)

>
> #all variables besides medv can be min-max transformed and provide meaningful info to
predict medv
> #compute max and min by column ("2" means by column)
> #Students: try with and without scaling and observe performance difference
> maxs <- apply(Boston[,-14], 2, max)
> mins <- apply(Boston[,-14], 2, min)
> (Boston[,-14] <- as.data.frame(scale(Boston[,-14], center=mins, scale=maxs-mins)))
        crim    zn      indus chas       nox       rm       age       dis       rad       tax   ptratio
1  0.000000e+00 0.180 0.06781525    0 0.31481481 0.5775053 0.64160659 0.2692031 0.00000000 0.20801527 0.2872340
2  2.359225e-04 0.000 0.24230205    0 0.17283951 0.5479977 0.78269825 0.3489620 0.04347826 0.10496183 0.5531915
3  2.356977e-04 0.000 0.24230205    0 0.17283951 0.6943859 0.59938208 0.3489620 0.04347826 0.10496183 0.5531915
```

```
4  2.927957e-04 0.000 0.06304985    0 0.15020576 0.6585553 0.44181256 0.4485446 0.08695652 0.06679389 0.6489362
5  7.050701e-04 0.000 0.06304985    0 0.15020576 0.6871048 0.52832132 0.4485446 0.08695652 0.06679389 0.6489362
6  2.644715e-04 0.000 0.06304985    0 0.15020576 0.5497222 0.57466529 0.4485446 0.08695652 0.06679389 0.6489362
7  9.213230e-04 0.125 0.27162757    0 0.28600823 0.4696302 0.65602472 0.4029226 0.17391304 0.23664122 0.2765957
8  1.553672e-03 0.125 0.27162757    0 0.28600823 0.5002874 0.95983522 0.4383872 0.17391304 0.23664122 0.2765957
9  2.303251e-03 0.125 0.27162757    0 0.28600823 0.3966277 1.00000000 0.4503542 0.17391304 0.23664122 0.2765957


…
>
> set.seed(1234)
> #use 2/3 data for training, rest for testing
> sample <- sample(1:nrow(Boston), nrow(Boston)*0.67)
> train <- Boston[sample,]
> test <- Boston[-sample,]
> #linout: tells nnet that we are doing a regression and want a linear regression output
value
> #decay: the learning rate, the smaller the value, the longer the training time needed,
the net effect of learning rate on classification performance is hard to tell, you will
have to try out different learning rate.
> set.seed(123)
> nr <- nnet(medv ~ ., train, linout=TRUE, size=6, decay=0.01, maxit=2000, trace=FALSE)
> predict <- predict(nr, test)
> (MAD <- mean(abs(predict-test$medv)))
[1] 6.532228

> #visualization
>
> #garson only plots NNs with one output node, so we couldn't use it for classification
problems
> #garson plots the importance of different attributes to the predicated variable.
> #in this example, we see pollution (nox), tax, and dis are among the top three factors
predicting property values
>
> garson(nr) + theme(axis.text.x=element_text(angle=45, hjust=1))
```



```
>
> #BUT, do not just take this as the 'truth'. Try scaling/without scaling, different split
of training and test examples, different learning rate, or hidden layer nodes etc.
> #you will find the important factors contributing to medv change.
> #because the small number of training examples, ANN can be quite sensitive to the
parameter changes.
> #In general, training/finding a robust ANN model is not very easy. ANN models have many
different configurations/architectures that have been shown to be effective for different
types of problems.
>
> #plotnet() plots the network architecture with the learned weights and bias for each
layer.
> #negative weights in grey, positives in black. The thickness of the line indicates the
relative magnitude of the weights
> plotnet(nr)
```




```
> #use h2o deep learning for regression
> train.h2o <- as.h2o(train, "train.h2o")
  |=================================================================================| 100%
> test.h2o <- as.h2o(test, "test.h2o")
  |=================================================================================| 100%
> #epochs is the max number of learning iterations
> set.seed(123)
> model.h2o <- h2o.deeplearning(x=1:13, y=14, training_frame = train.h2o, hidden=c(100,
100, 100, 100), epochs=500)
  |=================================================================================| 100%
> #for regression, the h2o output is just a vector of values
> predict.h2o <-as.vector(h2o.predict(model.h2o, train.h2o))
  |=================================================================================| 100%
> (MAD <- mean(abs(predict.h2o-train$medv)))
[1] 0.3019941
```

[ADVANCED]

1. Construct and evaluate an ANN on your dataset using appropriate metric.
2. Use Performance Estimates learned last week to find a set of best parameters for your network.