LAB 3 – CLASSIFIERS USING R

Part I: Neural Networks

We looked in class how to train a neural net. In this lab, we will use R packages to train neural net classifiers.

1. Install the **neuralnet** package:

```
install.packages("neuralnet", dependencies = T)
```

2. Load the package:

```
library(neuralnet)
```

3. Get the data:

We will work with the Boston Housing Dataset again. This time, we will use a different strategy to load the dataset:

```
install.packages("MASS", dependencies = TRUE)
library(MASS)
b <- Boston
```

4. Pre-processing the data:

Neural nets are bit tricky to train. We have to ensure that there are no missing or NA values in the data. Let's check if there are any NA values in any of the columns:

```
apply(b, MARGIN = 2, FUN = function(x) sum(is.na(x)))
```

apply is used to apply a function to each element of the data frame. If you want to apply it to the row-wise, use MARGIN = 1, and if you want to apply it to the column-wise, use MARGIN = 2. FUN defines the function to use for each element in the group. Here we want to find sum of elements in a group if they are 0.

Secondly, we want to normalize the data. Let's make all the value between 0 and 1. First, let's find the max and min value column-wise.

```
maxs = apply(b, MARGIN = 2, max)
mins = apply(b, MARGIN = 2, min)
```

scaled = as.data.frame(scale(b, center = mins, scale = maxs - mins))

Advanced R users:

What is the above line of code doing?
If you had to code the above, you would do:

```
maxs = apply(b, MARGIN = 2, max)
mins = apply(b, MARGIN = 2, min)
b1 <- b
for(i in 1:ncol(b)) {
 # grab the i^th column
 x <- b[,i]
 # normalize values between 0 and 1
 for(j in 1:length(x)) {
  x1[j] <- (x[j]-mins[i])/(maxs[i]-mins[i])
 }
 b1[,i] <- x1
}
```

5. Creating training and test datasets:

```
trainIndex <- sample(1:nrow(scaled), 0.8 * nrow(scaled))
train <- scaled[trainIndex, ]
test <- scaled[-trainIndex, ]
```

7. Training a neuralnet model:

Remember to create a model in R, you need a formula of type y ~ x, where y is the predicted value and x is the set of attributes. In many cases, you can use y ~ . , where "." represents all the attributes i.e. all columns except the predicted column y.

In neuralnet, you cannot use the "." symbol. You have to construct the formula yourself like: y ~ x1 + x2 ….

Is there an easier way? Yes. Grab all the column names, except the predicted value's, and join them together using + sign.

```
n <- names(train)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
```

OK, now let's train the neural net. First look up the following:

?neuralnet

How many hidden layers and neurons are there by default?

Run the following code:

nn <- neuralnet(f,data=train)

Plot the neural net obtained using:

plot(nn)

See the summary of the neural net. What was the training error?

nn$result.matrix

8. Using the created model on the training data:

Let's try to predict the output for the test data using the NN model created:

pred <- compute(nn,test[,1:13])

We are using the first 13 columns => because they represent attributes.

To get a better feel for the error metric, we need to go back to original scale. Thus, we need to "de-scale" the data.

pred.scaled <- pred$net.result *(max(b$medv)-min(b$medv))+min(b$medv)

Let's also de-scale the actual values:

real.values <- (test$medv)*(max(b$medv)-min(b$medv))+min(b$medv)

9. Let's fine Mean Squared Error (MSE):

MSE.nn <- sum((real.values - pred.scaled)^2)/nrow(test)

10. Let's plot predicted vs actual values:

```
plot(real.values, pred.scaled, col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')
```

11. Adding more layers:

In step 7, when you created the model, you used the default values for the following parameters:

number of hidden layers = 1
number of nodes in the hidden layer = 1

Now, let's add more power to the model by changing the model construction step as follows:

```
nn <- neuralnet(f,data=train_,hidden=c(5,3))
```

which means use 2 layers with 5 nodes in the first and 3 nodes in the second.

Make this change and re-run the steps 7 – 10. Do you notice a reduction in MSE?

Part II: Decision Trees

We are going to build decision trees using the rpart package in R, which implements recursive partitioning.

1. Install the **rpart** package:

```
install.packages("rpart", dependencies = T)
```

2. Load the package:

```
library(rpart)
```

3. Get the data:

We will use the prostate cancer progression dataset. It is part of the rpart package and can be loaded as:

```
data("stagec")
stagec
```

4. Look at the help of the method used to create the tree:

```
?rpart
```

5. Let's create the tree:

fit <- rpart(pgstat ~ age + eet + g2 + grade + gleason + ploidy, data = stagec, method = 'class', parms = list(split = "information"))

In the above equation, we construct a model for pgstat as a function of the other attributes.
The method = "class" directive indicates that this is a classification problem.
The split = "information" directive indicates that we use the information gain to split the nodes.

6. Let's see what the tree model looks like:

print(fit)

It's a very interesting plot. Each node is printed and its children are printed below it. There is a convention that the children of node numbered x are numbered 2x and 2x+1. If a node has * next to it, it indicates a leaf (terminal) node. The values in parenthesis indicate the probability of various classes at the node.

7. You can find more interesting details by typing the following command:

summary(fit)

You will view details such as variable importance, details of each of the nodes, etc.
"Improve" refers to the improvement in information by the split.

8. Plot the tree as follows:

plot(fit)
text(fit, all = TRUE)

9. Pruning the tree:

This package uses a complexity parameter to prune leaves that have complexity below a certain threshold. You can see more details here:
https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

10. Using a model for prediction.

Like other packages, rpart has a predict method. Details are found here:
https://stat.ethz.ch/R-manual/R-devel/library/rpart/html/predict.rpart.html