Exercise 5 - Support Vector Machines and Neural Networks

The goal of this exercise is to explore ANN's (Artificial Neural Networks), and SVM's (Support Vector Machines) with different kernels. This is to better understand the methods, and to get an introduction to black box methods.

- Literature

- Chapter 7 (Black Box Methods - Neural Networks and Support Vector Machines) of "Machine Learning with R" (First Edition) by Brett Lantz, Packt Publishing Ltd., second edition, 2015.
- Chapter 9 (Support Vector Machines) of "An Introduction to Statistical Learning with Applications in R" from G. James, D. Witten, D., T. Hastie, R. Tibshirani. Springer 2013.

Exercise 5.1: SVM

5.1.1 Train a SVM classifier on the data, and evaluate the performance of the classifier. For example, you can do a disjunct study: half data as the training data and the other half as the test data.

You can use the R implementation of SVM: library("kernlab").
The function is ***ksvm***
classifier_rbf <- ksvm(V1~ ., data = id, kernel = "rbfdot", kpar=list(sigma=0.05), C = 1)
In the experiments, please try different parameters to compare the results. There are 3 main parameters:

- *kernel: "kernel ="rbfdot""*
- *kernel parameters: "kpar = "automatic""*
- *the cost parameter: "C = 1"*

For the kernel type, you can try linear (vanilladot), polynomial (polydot) and radial basis (rbfdot) kernel.

Each of these kernels has different parameters to tune. Please take a look at the help document and try some reasonable values (https://www.rdocumentation.org/packages/kernlab/versions/0.9-29/topics/ksvm).

5.1.2 In the experiment, you can also try different value of 'C' to evaluate the performance of the classifier (C is the cost of constraints violation).

Exercise 5.2: Neural Networks:

5.2.1 Create a matrix (as below in grey background) for the training classes. It has N rows (the same as the number of training data) and 10 columns (binary). The column with '1' marks the corresponding class as the example shown below (eg. cyper '0' is represented by '1000000000').

0 ->	1	0	0	0	0	0	0	0	0	0
1 ->	0	1	0	0	0	0	0	0	0	0
…	…									
9 ->	0	0	0	0	0	0	0	0	0	1

Example code:

```
lev <- levels(id$X1) # Number of classes

# Create a list probabilities, for all labels
nnTrainingClass <- matrix(nrow = length(id$X1), ncol = 10, data = 0)

for(i in 1:length(id$X1)) { # Set probabilities to one for matching class
    matchList <- match(lev,toString(id$X1[i]))
    matchList[is.na(matchList)] <- 0
    nnTrainingClass[i,] <- matchList
}
trainingClass <- as.data.frame(nnTrainingClass)
```

5.2.2 Train a neural network with N inputs and 10 outputs, based on the modified training data.

You can use the "RSNNS" library. The R implementation of neural networks is ***mlp***

```
network <- mlp(id[,-1], trainingClass, size = c(2,2,2), maxit = 2, hiddenActFunc = "Act_TanH",
    learnFunc="Std_Backpropagation", learnFuncParams = c(0.01,0))
```

There are many parameters you can tune in a multi-layer perceptron NN, please try different parameters to find the optimal combination:

- "size = c(20,20,20)": the amount of hidden layers and hidden nodes.
- maxit = 2: Max iteration to learn
- hiddenActFunc = "Act_TanH": activation function
- learnFunc="Std_Backpropagation" : for standard back-propagation "learning rate" and "maximum output difference"
- learnFuncParams = c(0.01,0): learning parameters (depends on what function is used)

Then you can visualize the training by:

```
plotIterativeError(networkModel)
```

5.2.3 Evaluate the neural network with the test data.

You can use the ”predict” function

```
predictions <- predict(network, id2[,-1])
```

You can use the following code to convert the mlp output into class labels (0 - 9)

```
responselist <- matrix(nrow = length(predictions[,1]), ncol = 1, data = "Na")

for(i in 1:nrow(predictions)) {
    responselist[i,] <- toString( which(predictions[i,]==max(predictions[i,])) - 1 )
}
responselist <- data.frame(responselist)
responselist[,1] <- as.factor(responselist[,1])

# Calculating the accuracy
cf <- confusionMatrix(responselist[,1], id2[,1])
print( sum(diag(cf))/sum(cf) )
```

5.2.4 You can also try pre-processing of the data before feeding them to the NN. You can compare the results using raw data for the training (above) and results using the first some PCA components for the training.