Exercise Lesson07: Neural Networks

Machine Learning using R

Exercise 1: Boston housing data

The Boston housing data set contains information collected by the U.S Census Service concerning housing in the area of Boston Massachusetts.

a) The data set is included in the R package `MASS`. Load the `MASS` package and look at the data by typing `Boston`. Get an overview of the data, what are the dimensions of the data set? You can get more info about the meaning of the variables by calling the help page (**Hint**: `?Boston`).

b) We want to predict the variable `medv` in the data set using a neural network. Since neural networks usually converge much better with standardized predictors, we want to standardize all predictors in the Boston data (**Hint**: `scale()`). Try to find a way to create a data frame in which all predictors are standardized but the target variable `medv` is still in its original form.

c) Fit a single hidden layer neural network to the (scaled) Boston data with only one neuron in the hidden layer and no weight decay. Throughout this exercise, we use `maxit=10000` to make sure the NN can converge (**Hint**: `nnet(..., decay=0, maxit=10000)`). What is the meaning of the `linout` option in the `nnet` function and how do we have to set it in this case?

d) Using only one neuron in the hidden layer might be a too simple structure for the `Boston` data. We now want to let the R package `caret` tune a neural network for us. First, one needs to define a search grid with the possible parameter values. We only want to tune the size of the network and not use any weight decay. Generate a small search grid with the possible sizes 1, 2, 3 and 20. We add the size 20 to see how a much more complex network performs. **Hint**: `expand.grid(..., decay=0)`

e) Use the `train()` function of `caret` to train a single hidden layer neural network along our search grid (This might take approx. 3 minutes). What structure was finally selected? What was the RMSE of this model? How did the most complex NN perform?

f) We want to compare how a Random Forest is performing with this data set. Fit a Random Forest with the `cforest()` function to the Boston data consisting of 500 trees and an `mtry` value of 5. We want to calculate the (OOB) RMSE of the Random Forest. To this end we need to subtract the OOB predictions from the true `medv` values, and take the root of the mean squared differences (**Hint**: `sqrt(mean((d.bos$medv - ...))^2))`. What is the RMSE of the Random Forest?

Exercise 2: Image classification with the MNIST dataset

The MNIST dataset contains a collection of 28x28 pixel images of handwritten digits (grayscale). It is a commonly used data set to train and test machine learning methods for visual processing.

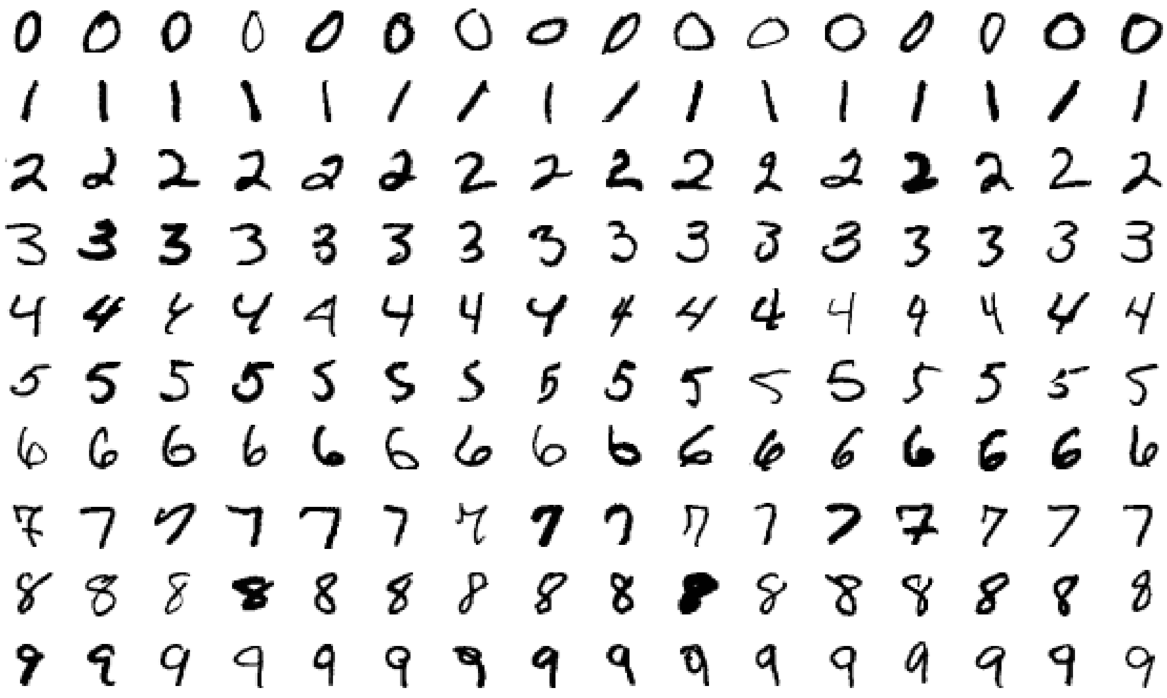

Figure 1: Examples from MNIST data

a) As described in the lecture, grayscale pictures are often represented by tables with three axes. In R, tables of higher dimensions are represented by "multi-way arrays". We can create a multi-way array with the `array()` command, which requires a vector of numbers to fill the array (`data=`) and a second vector indicating the dimensions of the array (`dim=`). Create a three-dimensional array including the numbers 1 to 24 with the dimensions `c(2,4,3)`. Look at the created table in R.

b) We can access individual elements in multi-way arrays like for normal tables using the square brackets (`[ ]`). Display only the element of the second row, of the third column of the first slice. What happens if you select one entire slice, e.g. the second one?

c) Read in the the MNIST data containing the 28x28 pixel images. The data is stored in the "**Mnist_training_and_test_data.RData**" file. This is not a .csv file as usual but an .RData file. RData files store saved R-objects. Once downloaded, read in the the data with `load("Mnist_training_and_test_data.RData")`. Now there should be four new objects in your environment: `test_x.0`, `test_y.0`, `train_x.0` and `train_y.0`. These objects store the pictures and the corresponding labels of the training and test set. Look at the dimensions of the arrays. How many pictures are in the training and test set? Also look at the target variable of the data (`test_y.0` and `train_y.0`).

d) In the second image of the training data, what is the grayscale value of the pixel in the top right corner of the image?

e) Each slice in the `train_x.0` array corresponds to one picture (3rd dimension). We can plot one picture by selecting only one slice in the array and feed it into the `image()` function. Select the 18th picture of train_x.0 and visualize it with `image()`.

f) To directly feed the MNIST pictures to a standard neural network, we need to "flatten" the images, so that each image is one row and the columns represent all the pixels of a picture. Run the code below to flatten the images of the training and test set. The code additionally scales the grayscale values to a range of 0-1 by dividing the values by 255 (better for NN). The target variable is turned into a factor since predicting the written number on a picture is a classification task. Finally, the pixel values are combined with the target variable into data frames. Look at the final data frames.

```r
### Turn into 2d matrices (each row one picture):
train_x <- array(as.numeric(aperm(train_x.0, perm = c(3, 1, 2))), dim = c(60000, 784))
test_x <- array(as.numeric(aperm(test_x.0, perm = c(3, 1, 2))), dim = c(10000, 784))
### Scale to 0-1 range:
train_x <- train_x/255
test_x <- test_x/255
### Turn number labels to factor:
train_y <- as.factor(train_y.0)
test_y <- as.factor(test_y.0)
### Combine to dataframe:
mnist.dtra <- data.frame("y"=train_y, train_x)
mnist.dte <- data.frame("y"=test_y, test_x)
### Look at dimensions:
dim(mnist.dtra)
dim(mnist.dte)
```

g) Now the data is ready to be fed to a classifier, e.g. a neural network. We want to train a single hidden layer NN on the training data and check its performance on the test data. Since training a neural network with `nnet()` of size=20 on the MNIST data takes approximately 20 hours, you can download an already fitted NN stored in the "NN_mnist.RData" file. The file contains a fitted NN (`nne.mnist` object). It was created with the command:

```r
nne.mnist <- nnet(y~., mnist.dtra, size=20, decay=0, maxit=10000, MaxNWts = 16000)
```

Predict the written numbers in the test set with the neural network and create a confusion matrix showing how well it performed (see slides). Calculate the accuracy of the predictions (accuracy = 1 - errorrate).

i) **Extra**: Compare how a Random Forest performs in the MNIST classification task. For a change we will use the classic `randomForest()` function from the R package with the same name, because it runs faster than `cforest()`. Fitting a Random Forest to the MNIST training data will be much faster than fitting a NN, in this case it will take approx. 10 minutes. Fit a Random Forest to the MNIST training data consisting (only) of 50 trees and check its performance using the test set. **Hint**: `library(randomForest); randomForest(..., ntree=50)`. (The `mtry` argument can be left away, it will automatically be set to a rule-of-thumb value).