Exercise:
Neural Networks

Machine Learning and Prediction Modelling

Exercise 1: Corona stress data

We work again with the corona stress data set (`Covid_stress_data_subs.rda`). Check the previous exercise for a description of the individual variables in the data.

a) Load the data into R using the `load` function. This time, we want to predict the value of the `conspiracy` variable which expresses the tendency of the participants towards conspiracy thinking. Take a look at the distribution of this variable by creating a histogram (**Hint**: `hist()`).

b) We want to predict the variable `conspiracy` using a neural network. Since neural networks usually converge much better with standardized predictors, all (numeric) predictors in our data should be standardized (**Hint**: `scale()`). Therefore, we want to create a data frame in which all numeric predictors are standardized but the target variable `conspiracy` and all categorical variables are still in their original form. There are many different ways to perform such a transformation of data in R. You can either try to find a solution yourself or use the code presented in the solution sheet (solving the problem with a for-loop).

c) Fit a single hidden layer neural network to the (scaled) corona 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?

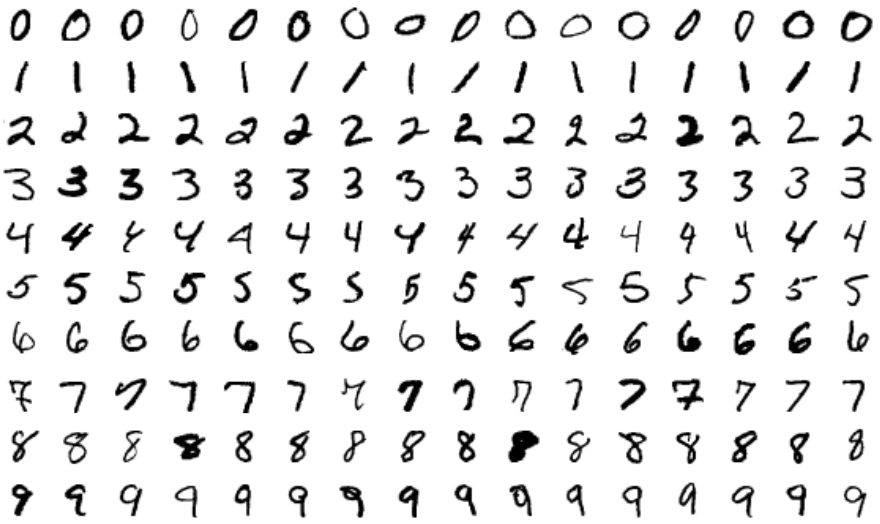
d) How many input neurons does our neural network have? Can you think of a reason why this number is not equal to the number of predictor variables?

e) Using only one neuron in the hidden layer might be a too simple structure for the corona 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, 4 and 5. (**Hint**: `expand.grid(..., decay=0)`)

f) Use the `train` function of `caret` to train a single hidden layer neural network along our search grid (this will take approx. 3 minutes). Note: Since `caret` will do the standardization of the variables for us, we can use the original data set (`dat_cvid`). What structure was finally selected? What was the RMSE of this model? Fix the random seed prior to running the code (`set.seed()`). Note: Even when using the same seed as in the solution sheet (`set.seed(2332)`), it can happen that your results differ from ours. There are various possible reasons for this, such as for example having a different R-version installed.

Exercise 2: MNIST data (handwritten digits)

The MNIST data set 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.



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` function, which requires a vector of numbers to fill the array (`data` argument) and a second vector indicating the dimensions of the array (`dim` argument). Create a three-dimensional array including the numbers 1 to 24 with the dimensions c(2,4,3). Look at the created array 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 MNIST data containing the 28x28 pixel images. The data is stored in the `Mnist_training_and_test_data.rda` file. Once loaded, 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 (3rd dimension) in the `train_x.0` array corresponds to one picture. 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.

```
### 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)
```

```
## [1] 60000   785
```

```
dim(mnist.dte)
```

```
## [1] 10000   785
```

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 use the already fitted NN stored in the `NN_mnist.rda` file. The file contains a fitted NN (`nne.mnist` object). It was created with the command:

```
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 (**Hint**: `predict(..., type='class')`). Calculate the accuracy of the predictions (accuracy = 1 - missclassification rate).

h) Compare how a Random Forest performs in the MNIST classification task. Fitting a Random Forest with `cforest` to the MNIST training data and generating predictions for the test data also takes a long time (approx. 3 hours). Therefore, we have already fitted a Random Forest (consisting of 500 trees) to the training data and used it to create predictions for the test data. The predictions have been created with the following commands:

```
library("partykit")
set.seed(111)
### Create random forest:
rf_mnist <- cforest(y~., mnist.dtra, ntree=500)
### Generate predictions for test data:
prds_rf <- predict(rf_minst, newdata = mnist.dte)
```

The created predictions are stored in the `RF500_mnistPreds.rda` file. Load the predictions in R and investigate the predictive accuracy of the Random Forest.