# Project 52500, 2017/18 - Instructions

April 4, 2018

This is the first part of project. Please read carefully and follow precisely instructions. You can work in groups of up to two students. Email to `or.zuk@mail.huji.ac.il` your solution $R$ code by Sunday 29/4/2018 at 11:59 pm. *Solution will not be accepted at a later date.*

**Instructions:** Implement a classifier in $R$, with two parts implemented in two separate $R$ functions: (i) trains a model from labeled examples (ii) applies the trained model to unlabeled data and generate predicted classes. Choose a team name ($3-20$ characters ) and write *one* $R$ file called 'Classifier_functions_<team-name>.R' with your IDs on first line, and two $R$ functions:

1. **TrainModel** - this function should get as input an two-dimensional array $X$ and a one-dimensional array of labels $y$ and return a learned model $m$.

2. **PredictModel** - this function should get as input a two-dimensional array $X$ and a learned model $m$ and return a one-dimensional array of labels $y$.

**Format:** $X$ is a matrix of size $n \times 784$ where $n$ is the number of input samples (images). $y$ is a vector of length $n$. $m$ can be a variable of your choice (e.g. vector of weights, class, data-frame) but must be of the same type in two functions.

**Data:** Use the MNIST dataset, available in course moodle, containing $60,000$ labeled images of size $28 \times 28$ in the training set and $10,000$ labeled images in the test set. The images are represented in gray-scale, where each pixel value, from 0 to 255, represents its darkness level.

**Requirements:** Your code must run (not crash) with input formats as instructed. The function TrainModel should finish training on the MNIST training set ($n = 60,000$ labeled examples) in 15 minutes on a standard PC laptop. The function 'PredictModel' must finish predicting (on test set of size $10,000$) in 1 minute.

You can (and are encouraged to) email solutions every week. I will evaluate all solutions which arrived at the beginning of each week (Sunday night at 23:59 of 8/4, 15/4, 22/4 and 29/4), and post the accuracy and running time of each team in course moodle on Monday.

**Libraries:** It is recommended to use the keras $R$ library: `https://keras.rstudio.com/`. Read carefully installation and usage instructions. This library lets you easily implement and train

neural-network-based classifiers, and has an interface for the MNIST dataset.

If you want to use other libraries in your solution please consult me for permission.

**Code:** I put three *R* files in moodle. Put these files in a directory of your choice and run the 'MNIST_runR' file. Implement your classifier in a new file 'Classifier_functions_<team-name>.R' and put it in the same directory to run and evaluate your classifier.

1. **MNIST_run.R** - this file loads the MNIST data, trains classifier , predicts on test set and evaluate results. This is the file I will use to evaluate your solutions, except that different training and test sets will be used.

2. **Classifier_functions_average.R** - this file contains and example classifier, which classifies images to digits based on their average darkness. When run on MNIST using MNIST_run.R, this classifier achieves roughly 25% accuracy.

3. **MNIST_general_functions.R** - some general functions which may be helpful for you

**Grading:** If your code crushes (e.g. due to missing variables, wrong variable formats/names etc.) or doesn't finish within specified time your grade for this task will be zero.

If your code runs without errors and finishes within specified time your grade for this task will be: $max\big(0, (acc_{test} - 80) \times 5\big)$, where $acc_{test}$ is your test accuracy in percentiles.

The network presented in chapter 1 of Nielsen's book achieves $\sim$95.4% test accuracy when trained for 30 epochs. Implementing this network correctly in *R* will thus give you a grade of $\sim$77pt (exact grade may be slightly different due to stochasticity and different test set used).

In addition, there will be bonuses for the top solutions:

1. **Accuracy:** Five most accurate solutions obeying running time limits will get bonuses according to Table 1.

2. **Speed:** Five fastest solutions (training time) will get bonuses according to Table 1. To gain speed bonus, your program's accuracy must be $> 90\%$.

3. **Pareto-Optimality:** Your solution is considered pareto optimal if no other solution is *both* faster *and* more accurate. Any pareto-optimal solution not in top three in accuracy or speed will get 3pt bonus.

| rank | 1st | 2nd | 3rd | 4th | 5th |
|---|---|---|---|---|---|
| Bonus | 10 | 5 | 3 | 2 | 1 |

Table 1: Bonus points. 1st teams in both time and accuracy must present their method in class