# 582631 Introduction to Machine Learning
# Programming project for separate examinations (Fall 2016)

General instructions:

- These instructions apply to projects associated with separate examinations after the course given in Fall 2016
- Return a written report (as PDF) and one directory (as a zip or tar.gz file) including all your code.
- Do not include any of the data files in your solution file.
- Mention your name and student ID in the report.
- The report is the main basis for grading: All your results should be in the report. We also look at the code, but we won't however go fishing for results in your code.
- The code needs to be submitted as a file or set of files that can be run in the usual Linux environment of the department. At least R, Matlab and Octave are supported. Commands to run your programs must be given in the report.
- In your report, the results will often be either in the form of a figure or program output. In both cases, add some sentences which explain what you are showing and why the results are the answer to the question.
- If we ask you to test whether an algorithm works, always give a few examples showing that the algorithm indeed works
- If there is something unclear about the problem setting, please ask by e-mail to the lecturer, `teemu.roos@cs.helsinki.fi`. However, I will not answer any "how-to" questions about implementing the assignments. It's part of your job to figure out all implementation issues, using information available on the Internet etc. if needed.

## Exercise 1

In this exercise you will implement the perceptron algorithm[1] for training a linear classifier, and apply it to the MNIST dataset of handwritten digits.

(a) First, create your own implementation of the perceptron algorithm (see *Linear models* and *Perceptron algorithm* in the slides for Lectures 9–10).

To verify that it is working correctly, create simple small ($N = 4$ or similar) two-class datasets in two dimensions for which it is easy to visually see whether the classes are linearly separable or not, and apply the algorithm (with a bias term) to those datasets, drawing the learned decision boundary into the scatterplot of the points. In this way, give

[1] Although the perceptron is currently *not* included in the learning objectives of the course, this exercise will be relatively straightforward to implement based what you have learned. Feel free to use the internet to find more information.

several (at least 5) examples showing that your implementation of the algorithm finds a separating hyperplane when such a hyperplane exists, and does not converge when such a hyperplane does not exist.

*Hint:* Recall that the bias term can be included by using the *useful trick* mentioned in the slides of Lectures 3–4.

(b) Load the first $N = 5000$ MNIST digits. Take the first $N/2$ digits to be training set, and the remaining $N/2$ to be the test set. Select only those digits in the training set which are either zeros or ones, and apply your implementation of the perceptron algorithm to this data. (Remember that the corresponding class vector has to be converted to $\pm 1$.) Does the algorithm converge? After how many iterations? Applying the learned linear classifier to the test set (using only those instances which are zeros or ones), what is the error rate? Plot the pixel weights as an image (leaving out the bias weight), does it resemble a zero or a one? Why or why not?

*Hint:* For instructions on how to read the data in `R`, see Exercise set 2 of the Fall 2016 course.

## Exercise 2

In this exercise you will implement and apply the Naive Bayes classifier to the 20 newsgroups dataset.

(a) Load the 20 newsgroups data. Let the first (by document ID) 90% of documents from each newsgroup make up the training set, while the remaining documents constitute the test set. Thus, the training set consists of docIDs [1:432, 481:1003, ...] while the test set is given by [433:480, 1004:1061, ...] We will only consider the presence/absence of words in documents, not their frequency, so we will not use the third column of the main data matrix.

(b) For each combination ⟨`newsgroup,word`⟩, estimate the probability that a document from that newsgroup contains that word. Apply Laplace smoothing. For each of the 20 newsgroups, list the 200 words that have the highest probabilities of occurrence (in decreasing order of probability). This is a sanity check that the estimated probabilities make sense. Also estimate the prior probabilities $P(g)$ for the 20 different newsgroups $g$ using Laplace smoothing.

(c) Use the Naive Bayes classifier, based on the estimates from part (b) above, to classify the training set. Then classify the test set. Give the resulting *confusion matrix* and the corresponding error rates. Which newsgroups get mixed up most with each other? How well is the classifier working?

*Hint:* The product of the feature probabilities tends to get very small which may cause numerical problems. To avoid this, you can use a logarithmic scale to store the probabilities, i.e., initialize a probability `logp =` $\log P(g)$ and add to it the terms $\log P(w_1 \mid g), \ldots, \log P(w_n \mid g)$. The maximum probability class will have the highest `logp` value.

## Exercise 3

In this exercise you will implement agglomerative hierarchical clustering and apply it to a small subset of movies from the Movielens dataset.

(a) Create your own implementation of agglomerative hierarchical clustering, supporting both the single-link and complete-link approaches. Your algorithm should take as input the similarity matrix among the objects and output a list showing which clusters are joined at each step of the algorithm, and at what similarity value the joins occur (i.e. the 'height' of the dendrogram at which each pair of combined clusters are joined).

(b) Create a simple dataset of 5 randomly selected points in 2 dimensions and plot these points. Let the similarity between any two given points be equal to the negative euclidean distance between the points. Compute the 5-by-5 similarity matrix and apply your hierarchical clustering algorithm (using both single-link and complete-link) to this dataset. Show the results. (This is a sanity check that your hierarchical clustering algorithm is working properly.)

(c) Load the Movielens data. Let the similarity between any two movies be equal to the Jaccard coefficient, i.e. the number of users who rated both movies divided by the number of users who rated at least one of the movies. (The web page of the course has a function that, given two different movie IDs, outputs this Jaccard coefficient.) Verify in your code that the similarity between the movies *Toy Story* and *GoldenEye* is 0.217.

(d) Finally, select 20 movies that you are familiar with, such that some movies are clearly similar to each other (in terms of the genre, director, actors, etc—sequels may be good in this respect) while others are presumed to be more different. Compute the 20-by-20 similarity matrix among the movies you selected, using the Jaccard coefficient as the similarity measure. Then cluster the 20 movies using your implementation of agglomerative hierarchical clustering, both using the single-link and complete-link approaches. Show and explain the results. Did they conform to what you expected, or is the clustering very different from your intuitive understanding?