DATA11002 Introduction to Machine Learning
Programming project for separate examinations (Fall 2017)

General instructions:

- These instructions apply to projects associated with separate examinations after the course given in Fall 2017 until further notice
- 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. For example, R, python, and Matlab/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, explain in detail 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.

Task 1

In this task 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 Lecture 7).

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 task 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 ± 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 5 of the Fall 2017 course.

Task 2

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

(a) Load the data. Let the first (by document ID) 90% of documents from each group 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 `(group,word)`, estimate the probability that a document from the given group contains the word `word`. Apply Laplace smoothing. Check that the most probably occurring words are typical "stopwords" like *the*, *to*, *and*, etc. Also estimate the prior probabilities $P(\texttt{group})$ for the 20 groups using Laplace smoothing.

(c) Use the Naive Bayes classifier, based on the estimates from part (b) above, to classify the documents in the training set. Do the same for the test set. Give the resulting *confusion matrices* and the corresponding error rates. Which groups get mixed up most with each other? (Use the newsgroup labels instead of just their indices, and try to explain the results.)

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` $= \log P(\texttt{group})$ and add to it the term $\log P(w_i \mid \texttt{group})$ for all words w_i occurring in the document and the term $\log(1 - P(w_{i'} \mid \texttt{group}))$ for all words $w_{i'}$ that do *not* occur in the document. The maximum probability class will have the highest `logp` value.

Task 3

In this task you will explore linear and non-linear dimension reduction techniques, apply them to a dataset containing 28 × 28 pixel grayscale images – which, you may be surprised to hear, is *not* the usual MNIST.

(a) Load the image dataset but keep only the items in classes 5 ("sandal"), 7 ("sneaker"), and 9 ("angle boot"). You can use either all of the data from these classes or a smaller subset such as those in the 10000 item test set if you have trouble with efficiency.

Hint: The data can be loaded using the exact same scripts that work for the standard MNIST data.

(b) Implement PCA. You are allowed to use ready-made functions for the eigenvalue decomposition (or singular value decomposition) but you shouldn't use functions such as `prcomp` in R.

Hint: Remember to center and normalize the data.

(c) Plot the data in terms of the first two principal components. Try to use colors, or even better use the images instead of the dots, to show the different classes so that you can tell whether the classes are separated or not.

(d) Now find an implementation of t-distributed stochastic neighbor embedding (t-SNE). Unlike above with PCA, you should use a ready-made implementation.

Apply t-SNE to obtain a two-dimensional representation of the data and produce similar plots as you did with PCA. Compare the results.

Hint: In R, try the package `tsne`. And python, try `sklearn.manifold.TSNE`.