Queen Mary
University of London

MTH786P **Machine Learning with Python, Winter 2019**
Coursework 1 **M. Benning**

Introduction

For computational efficiency of typical operations in machine learning applications, it is very beneficial to use NumPy arrays together with vectorised commands, instead of explicit `for` loops. The vectorised commands are better optimised, and bring the performance of Python code (and similarly e.g. for Matlab) closer to lower level languages like C. In this exercise, you are asked to write efficient implementations for two small problems that are typical for the field of machine learning.

Getting Started

Follow the Python setup tutorial provided on the module `github` repository here:

MLwithPython/tree/master/labs/ex01/python_setup_tutorial.md

After you are set up, clone or download the repository, and start by filling in the template notebooks in the folder `/labs/ex01`, for each of the 3 tasks below.

To get more familiar with vector and matrix operations using NumPy arrays, it is also recommended to go through the `npprimer.ipynb` notebook in the same folder.

Note: The following three exercises could be solved by `for`-loops. While that's ok to get started, the goal of this exercise sheet is to use the more efficient vectorized commands instead:

Useful Commands

We give a short overview over some commands that prove useful for writing vectorized code. You can read the full documentation and examples by issuing `help(func)`.
At the beginning: `import numpy as np`

- `a * b, a / b`: element-wise multiplication and division of matrices (arrays) $\boldsymbol{a}$ and $\boldsymbol{b}$
- `a.dot(b)`: matrix-multiplication of two matrices $\boldsymbol{a}$ and $\boldsymbol{b}$
- `a.max(0)`: find the maximum element for each column of matrix $\boldsymbol{a}$ (note that NumPy uses zero-based indices, while Matlab uses one-based)
- `a.max(1)`: find the maximum element for each row of matrix $\boldsymbol{a}$
- `np.mean(a), np.std(a)`: compute the mean and standard deviation of all entries of $\boldsymbol{a}$
- `a.shape`: return the array dimensions of $\boldsymbol{a}$
- `a.shape[k]`: return the size of array $\boldsymbol{a}$ along dimension k
- `np.sum(a, axis=k)`: sum the elements of matrix $\boldsymbol{a}$ along dimension k
- `linalg.inv(a)`: returns the inverse of a square matrix $\boldsymbol{a}$

A broader tutorial can be found here: `http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf`

Task A: Matrix Standardisation

The different dimensions or features of a data sample often show different variances. For some subsequent operations, it is a beneficial preprocessing step to standardise the data, i.e. subtract the mean and divide by the standard deviation for each dimension. After this processing, each dimension has zero mean and unit variance.

Write a function that accepts a data matrix $\boldsymbol{x} \in \mathbb{R}^{n \times d}$ as input and outputs the same data after normalization. n is the number of samples, and d the number of dimensions, i.e. rows contain samples and columns features.

Task B: Pairwise Distances in the Plane

One application of machine learning in computer vision is interest point tracking. The location of corners in an image is tracked along subsequent frames of a video signal (see Figure 1 for a synthetic example). In this context, one is often interested in the pairwise distance of all points in the first frame to all points in the second frame. Matching points according to minimal distance is a simple heuristic that works well if many interest points are found in both frames and perturbations are small.

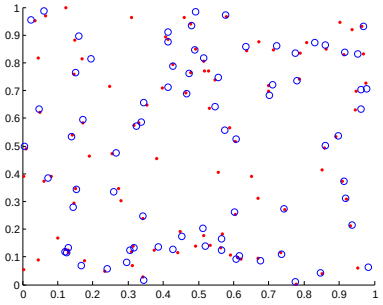

Figure 1: Two sets of points in the plane. The circles are a subset of the dots and have been perturbed randomly.

Write a function that accepts two matrices $\mathbf{P} \in \mathbb{R}^{p\times 2}, \mathbf{Q} \in \mathbb{R}^{q\times 2}$ as input, where each row contains the (x, y) coordinates of an interest point. Note that the number of points (p and q) do not have to be equal. As output, compute the pairwise distances of all points in $\mathbf{P}$ to all points in $\mathbf{Q}$ and collect them in a matrix $\mathbf{D}$. Element $D_{i,j}$ is the Euclidean distance of the i-th point in $\mathbf{P}$ to the j-th point in $\mathbf{Q}$.

Theory Questions

In addition to the practical exercises you do in the tutorials, we will in future also provide theory questions, in order to prepare you for the final exam. As for the rest of the exercises, it is *not* mandatory to solve them, but we would recommend that you try them during the term. Many students could otherwise be surprised by the theoretical nature of the final exam after having worked on very practical coursework and projects. Do not fall for this trap! Passing the course requires acquiring both a practical and a theoretical understanding of the material, and those exercises should help you with the latter.

This week, as we just started the course, there are no theoretical exercises. Instead, it could be useful to get engaged in some of the prerequisites:

- Make sure your linear algebra is fresh in memory, especially
  - Matrix manipulation (Multiplication, Transpose, Inverse)
  - Ranks, Linear independence
  - Eigenvalues and Eigenvectors

  You can use the following resources to help you get up to speed if needed.
  - The Linear Algebra handout
  - Gilbert Strang's Introduction to Linear Algebra.

- If it has been long since your last calculus class, make sure you know how to handle Gradients. You can find a quick summary and useful identities in The Matrix Cookbook.
- For probability and statistics, you should at least know about
  - Conditional and joint probability distributions
  - Bayes theorem
  - Random variables, independence, variance, expectation
  - The Gaussian distribution

.