Labs
Machine Learning Course
Fall 2016

EPFL
School of Computer and Communication Sciences
Martin Jaggi & Rüdiger Urbanke
mlo.epfl.ch/page-136795.html
epfmlcourse@gmail.com

Problem Set 1, Sept 22, 2016 (Efficient Python/NumPy Programming)

Introduction

For computational efficiency of typical operations in machine learning applications, it is very beneficial to use NumPy arrays together with vectorized commands, instead of explicit `for` loops. The vectorized commands are better optimized, 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 three small problems that are typical for the field of machine learning.

Getting Started

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

github.com/epfml/ML_course/tree/master/labs/ex01/python_setup_tutorial.md

After you are set up, we recommend downloading the folder of ex01, and start by filling in the template notebooks 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.

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`

For users who were more familiar with Matlab, a nice comparison of the analogous functions can be found here: `https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html`

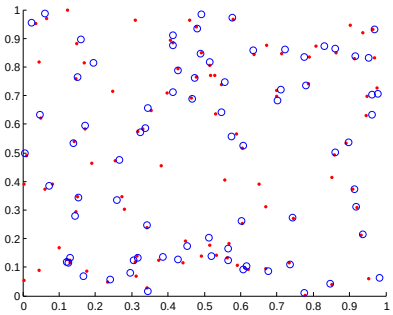

Figure 1: Two sets of points in the plane. The circles are a subset of the dots and have been perturbed randomly.

Task A: Matrix Standardization

The different dimensions or features of a data sample often show different variances. For some subsequent operations, it is a beneficial preprocessing step to standardize 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. Note that this is not equivalent to data whitening, which additionally de-correlates the dimensions (by means of a coordinate rotation).

Write a function that accepts 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 to 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.

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 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 $\mathbf{Q}$.

Task C: Likelihood of a Data Sample

A subtask of many machine learning algorithms is to compute the likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ of a sample $\boldsymbol{x}$ for a given density model with parameters $\boldsymbol{\theta}$. Given k models, we now want to assign $\boldsymbol{x}_i$ to the model for which the likelihood is maximal: $a_i = \arg\max_m p(\boldsymbol{x}_i|\boldsymbol{\theta}_m)$, where $m = 1, \ldots, k$. $\boldsymbol{\theta}_m$ are the parameters of the m-th density model.

We ask you to implement the assignment step for the two model case, i.e. $k = 2$. As input, your function receives a data matrix $\boldsymbol{x} \in \mathbb{R}^{d \times n}$ (where $\boldsymbol{x}_i$ is the i-th column of $\boldsymbol{x}$) and the parameters $\boldsymbol{\theta}_1 = (\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\boldsymbol{\theta}_2 = (\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ of two multivariate Gaussian distributions:

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right).$$

$|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}^{-1}$ its inverse. Your function returns the assignment vector $\boldsymbol{a} \in \{1, 2\}^n$, where $a_i = 1$ means that $\boldsymbol{x}_i$ has been assigned to model 1. Therefore, it holds that $p(\boldsymbol{x}_i|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) > p(\boldsymbol{x}_i|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$.