Labs
Machine Learning Course
Fall 2019

EPFL
School of Computer and Communication Sciences
Martin Jaggi & Rüdiger Urbanke
www.epfl.ch/labs/mlo/machine-learning-cs-433

Problem Set 1, Sept 19, 2019
(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, clone (using command line or a git desktop client) 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`

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

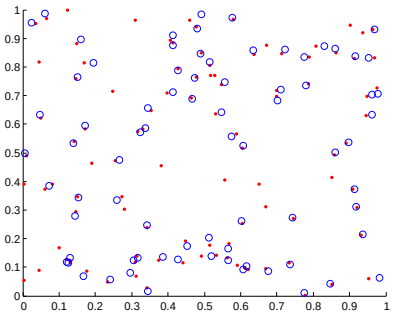

Figure 1: Two sets of points in the plane. The circles are a subset of the dots and have been perturbed randomly.

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

In this exercise, you are **not** required to understand the statistics and machine learning concepts described here yet. The goal here is just to practically implement the assignment of data to two given distributions, in Python.

A subtask of many machine learning algorithms is to compute the likelihood $p(\boldsymbol{x}_n|\boldsymbol{\theta})$ of a sample $\boldsymbol{x}_n$ for a given density model with parameters $\boldsymbol{\theta}$. Given k models, we now want to assign $\boldsymbol{x}_n$ to the model for which the likelihood is maximal: $a_n = \arg\max_m p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}_m)$, where $m = 1, \ldots, k$. Here $\boldsymbol{\theta}_m = (\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ are the parameters of the m-th density model ($\boldsymbol{\mu}_m \in \mathbb{R}^d$ is the mean, and $\boldsymbol{\Sigma}_m$ is the so called covariance matrix).

We ask you to implement the assignment step for the two model case, i.e. $k = 2$. As input, your function receives a set of data examples $\boldsymbol{x}_n \in \mathbb{R}^d$ (indexed by $1 \leq n \leq N$) as well as the two sets of parameters $\boldsymbol{\theta}_1 = (\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\boldsymbol{\theta}_2 = (\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ of two given multivariate Gaussian distributions:

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

$|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}^{-1}$ its inverse. Your function must return the 'most likely' assignment $a_n \in \{1, 2\}$ for each input point n, where $a_n = 1$ means that $\boldsymbol{x}_n$ has been assigned to model 1. In other words in the case that $a_n = 1$, it holds that $p(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) > p(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$.

Theory Questions

In addition to the practical exercises you do in the labs, as for example above, we will in future labs also provide you some theory oriented questions, to prepare you for the final exam. As the rest of the exercises, it is *not* mandatory to solve them, but we would recommend that you at least look at - and try - some of them during the semester. From last year's experience, many students where surprised by the heavy theoretical focus of the final exam after having worked on the two very practical projects. Do not fall for this trap! Passing the course require acquiring both a practical and a theoretical understanding of the material, and those exercises should help you with the latter.

However, please note that the difficulty of these exercises might not be of the same level as the exam and are not enough by themselves; you should read the additional material given at the end of the lectures and do the exercises in the recommended books - see The course info sheet.

Note that we will try to, but might not, provide solutions.

This week, as we just started the course, there are no exercises. You should refresh your mind of the prerequisites, especially on the following topics.

- 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 Linear Algebra and Learning from Data or Introduction to Linear Algebra. Some chapters are available online, and the books (along with many other textbooks on linear algebra) should be available at the EPFL Library.
- 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

  If you need a refresh, check Chapter 2 in Pattern Recognition and Machine Learning by Christopher Bishop, available at the EPFL Library.