Machine learning, prediction methods and applications

Pierre Michel

MASTER in Economics - Track EBDS - 2nd Year (2022)

Linear algebra, optimization and linear regression

The purpose of this session is to learn how to handle `numpy` and its features in Python. The first part proposes some exercises to manipulate basic functions from `numpy`. Do not hesitate to visit the following page: https://numpy.org/doc/.

The second part will focus on some monovariable optimization methods. The goal will be to implement three well-known methods (golden-section search, parabolic interpolation and Newton's method). Algorithms in pseudo-code are provided.

Finally we will work on the Boston's housing dataset: the goal is to predict the median value of houses' prices in different suburbs of Boston given different features of the districts (criminality, industrialization, etc…). You will use `numpy` to implement your own version of the linear regression (we call that "coding from scratch"). For the parameter estimation, you will use either the function `minimize` from the package `scipy` or your own implementation of the gradient descent algorithm (the latter is recommended, even if `minimize` is a very interesting function). Check the online help !

Exercice 1: manipulate `numpy`

N-dimensional arrays (`ndarray`)

The arrays in `numpy` have the type `ndarray`, allowing us to create multidimensional arrays (for example a dataset is a 2D-array). Many methods (or functions) can be applied to this type of objects. For example, it is possible to compute the sum (`sum`), the mean (`mean`), the maximum (`max`) of the values contained in an array. These arrays have also some attributes (the most important are `shape` and `size`).

To create a `ndarray` object, different functions can be used: `np.array()`, `np.full()`, `np.zeros()`, `np.ones()`, `np.eye()`, `np.random.randn()`. The goal of this exercice is to manipulate basic functions in `numpy`.

a) Create a matrix $A \in \mathbb{N}^{n \times p}$ (a 2D-array) with $n = 3$ rows and $p = 3$ columns, full of random integer values between 1 and 9, and print its shape. [functions to use: `np.random.randint`, `np.shape`].

b) Create a vector x_1, as the first row of A and a vector x_2, as the third column of A. Print these vectors. Print the value a_{22}. Compute the inner and outer products of x_1 and x_2. [functions to use: `np.inner` and `np.outer`].

c) Create a matrix $B \in \mathbb{N}^{n \times p}$ in the same way as A. Compute AB and BA. What do you observe ? [function to use: `np.dot`].

d) Create a vector $x = (1, 2, 3)$ (a 1D-array), print its shape and size and compute Ax and Bx. [function to use: `np.dot`].

e) Create the identity matrix I_3, the diagonal matrix D_3 (you will choose the values in the diagonal) with the same shape as A and B. Transpose A and B. [functions to use: `np.eye`, `np.diag`, `np.transpose`].

Reminder: monovariable optimization methods

We herein remind three well-know monovariable optimization methods. The pseudo-code of each algorithm is provided. We will denote f a function to minimize defined as $f : [a, b] \to \mathbb{R}$, with $a, b \in \mathbb{R}$.

Golden-section search algorithm

```
Set $\tau = \frac{1+\sqrt{5}}{2}$ (Golden number)
Set $a_0 = a$
Set $b_0 = b$
For $i = 0, ..., N_{max}$ ($N_{max}$ is the number of iterations)
  Set $a' = a_i + \frac{1}{\tau^2}(b_i - a_i)$
  Set $b' = a_i + \frac{1}{\tau}(b_i - a_i)$
  If $f(a') < f(b')$
    Set $a_{i+1} = a_i$
    Set $b_{i+1} = b'$
  Else if $f(a') > f(b')$
    Set $a_{i+1} = a'$
    Set $b_{i+1} = b_i$
  Else if $f(a') = f(b')$
    Set $a_{i+1} = a'$
    Set $b_{i+1} = b'$
```

Parabolic interpolation algorithm

```
Set $x_0, y_0, z_0 \in [a, b]$ with $f(x_0) \geq f(y_0)$ and $f(z_0) \geq f(y_0)$,
For $i = 0, ..., N_{max}$
  Set $f[x_i, y_i] = \frac{f(y_i) - f(x_i)}{y_i - x_i}$
  Set $f[x_i, y_i, z_i] = \frac{f[x_i, z_i] - f[x_i, y_i]}{z_i - x_i}$
  Set $y_{i+1} = \frac{x_i + y_i}{2} - \frac{f[x_i, y_i]}{2f[x_i, y_i, z_i]}$
  If $y_{i+1} \in [x_i, y_i]$
    Set $x_{i+1} = x_i$
    Set $z_{i+1} = y_i$
  Else if $y_{i+1} \in [x_i, z_i]$
    Set $x_{i+1} = y_i$
    Set $z_{i+1} = z_i$
```

Newton(-Raphson)'s algorithm

Newton's method is not really an optimization method. It aims to find "zeros" of a function $f : \mathbb{R} \to \mathbb{R}$, in other words it tries to solve $f(x) = 0$. The idea is to apply this method in order to solve the equation $\nabla f(x) = 0$, 1st order necessary condition for the detection of the extrema of a function. the pseudo-code of the algorithm is as follows:

```
Set $k = 0$
Select $x^{(0)}$ in the neighborhood of $x^*$ (the "zero")
Select $\epsilon > 0$
While $|x^{(k+1)} - x^{(k)}| \geq \epsilon$ and $k \leq k_{max}$
    Set $x^{(k+1)} = x^k - \frac{f(x^k)}{f'(x^k)}$
    Set $k = k + 1$
```

Exercice 2: Monovariable optimization

We are looking for the minima of the function f defined as follows:

$$\begin{aligned} f : \mathbb{R} - \{i^2; i \in \mathbb{Z}\} &\to \mathbb{R} \\ &\mapsto \sum_{i=1}^{20} \left(\frac{2i - 5}{x - i^2} \right)^2 \end{aligned}$$

a) Plot this function in Python using the function `plot` from the package `matplotlib.pyplot`. Comment about its minima and its behavior.

b) Implement the Golden-section search and try it on f.

c) Implement the parabolic interpolation method and try it on f.

d) Implement the Newton's method and try it on f.

e) Compare the different algorithms (in terms of estimated values ans convergence's speed).

Exercice 3: Linear regression from scratch

In this exercice you will implement the **cost function** and the **gradient** for linear regression in Python (see Session's slides), using simple loops in a first time.

You will use a dataset from the UCI machine learning repository: Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. The data was created by Harrison, D. and Rubinfeld, D.L. in the article "Hedonic prices and the demand for clean air" published in the Journal of Environment Economics & Management, vol.5, 81-102, 1978.

a) Load the dataset `housing.data`. A supplementary column of ones will be added to the dataset (it will correspond to the intercept parameter θ_0). Note: information about the dataset and its features are available in the file `housing.names`.

b) Split the dataset in two subsamples: a training (`train.X`) and a test set (`test.X`). For this, the dataset should be shuffled (i.e rows are randomly permuted). The two thirds of the first rows will be selected for the training set, the remaining for the test set. Only predictor variables will be in those datasets. The prices will be stored in separate datasets `train.y` and `train.x`, respectively the training and test examples. The training set will be used to find the best choice of θ for predicting the house prices and then the test set will be used to check its performance.

c) Implement the batch gradient descent and the stochastic gradient descent algorithms. Test these two algorithms to find the best θ (the one minimizing the cost function). Note: you can test your functions using the pre-loaded optimization function `minimize` from the package `scipy`.

d) Compare the results and the convergence speed of your two approaches. You will perform a "runtime analysis" by measuring the time taken for estimating the θ in both situations and test mean square error. Add a plot of predicted versus observed prices for the examples in the test set.

Recap: The main functions you have to create in this exercice are the following:

- `predict`: for $X \in \mathbb{R}^{n \times p}$ and $\theta \in \mathbb{R}^p$, returns $X\theta$.
- `error`: for $X \in \mathbb{R}^{n \times p}$, $\theta \in \mathbb{R}^p$ and $y \in \mathbb{R}^n$, returns $X\theta - y$.
- `cost_function`: for $X \in \mathbb{R}^{n \times p}$, $\theta \in \mathbb{R}^p$ and $y \in \mathbb{R}^n$, returns $J(\theta) = \frac{1}{2}\sum_i(\theta^T x^{(i)} - y^{(i)})^2$
- `gradient`: for $X \in \mathbb{R}^{n \times p}$, $\theta \in \mathbb{R}^p$ and $y \in \mathbb{R}^n$, returns $\forall j \in \{1, ..., p\}$, $\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)}(h_\theta(x^{(i)}) - y^{(i)})$

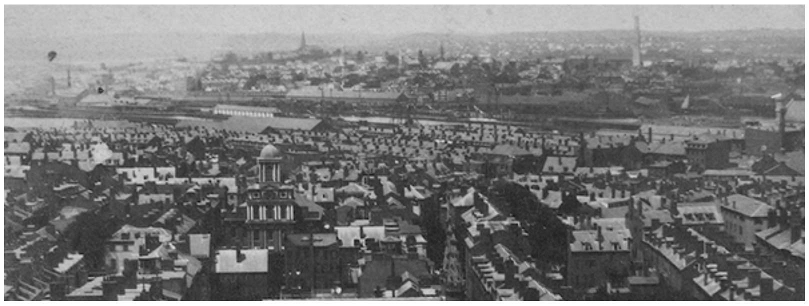

Figure 1: West Boston