Data Science Lab
Lab #1

Politecnico di Torino
October 9-10, 2019

Intro

The purpose of this laboratory is to get you acquainted with Python. More specifically, you will learn how to read different types of datasets (CSV and JSON). You will read the dataset contents from the filesystem into the program's memory, to then perform several operations on the data.

1 Preliminary steps

1.1 Python 3 availability

In order to make sure that Python is correctly installed on the device, open a terminal and run the command `python3 --version` (or, `python --version`). The results should be similar to the one displayed below.

```
$ python --version
Python 3.7.4
$ python3 --version
Python 3.7.4
```

Make sure Python 3 is installed: the version should be in the form `3.x.x`.

1.2 Interactive environment availability

Python scripts (identified by the `.py` extension) can either be run from the Python interpreter (`python script.py`), or in an interactive environment. Two popular options are Jupyter Notebook and IPython. You can install either by following the installation steps available. You can launch Jupyter Notebook with the command `jupyter notebook` and IPython with `ipython`.

1.3 Datasets download

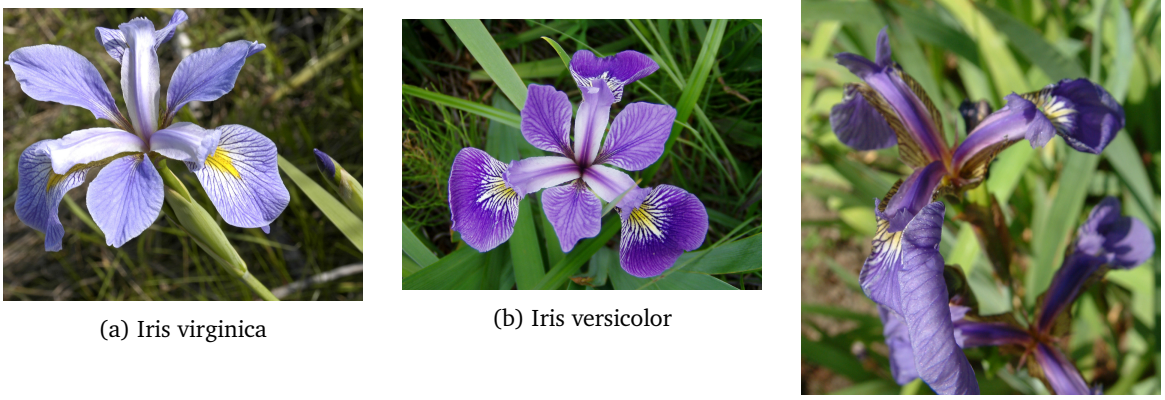
For this lab, three different datasets will be used. Here, you will learn more about them and how to retrieve them.

Iris

Iris is a particularly famous *toy dataset* (i.e. a dataset with a small number of rows and columns, mostly used for initial small-scale tests and proofs of concept). This specific dataset contains information about the Iris, a genus that includes 260-300 species of plants (you can read more about Iris on Wikipedia). The Iris dataset contains measurements for 150 Iris flowers, each belonging to one of three species: Virginica, Versicolor and Setosa. (50 flowers for each of the three species). These three species all present similar flowers, as you can see in Figure 1.

Each of the 150 flowers contained in the Iris dataset is represented by 5 values:

- sepal length, in cm



(a) Iris virginica

(b) Iris versicolor

(c) Iris setosa

Figure 1: Iris virginica, versicolor and setosa (images from Wikipedia)

- sepal width, in cm
- petal length, in cm
- petal width, in cm
- Iris species, one of: Iris-setosa, Iris-versicolor, Iris-virginica

Each row of the dataset represents a distinct flower (as such, the dataset will have 150 rows). Each row then contains 5 values (4 measurements and a species label).

The dataset is described in more detail on the UCI Machine Learning Repository website. The dataset can either be downloaded directly from there (`iris.data` file), or from a terminal, using the `wget` tool. The following command downloads the dataset from the original URL and stores it in a file named `iris.csv`.

```
$ wget "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data" -O iris.csv
```

Info: If you are not already, you should get familiar with the terminal. Downloading files with `wget` (or `curl`) is just one of the many, many things that can be done with it.

The dataset is available as a Comma-Separated Values (CSV) file. These files are typically used to represent tabular data. Each row is represented on one of the lines. Each of the rows contains a fixed number of columns. Each of the columns (in each row) is separated by a comma (`,`), hence the name. You can read more about CSV files on Wikipedia.

The following are 3 lines taken from the Iris dataset. You should check the contents of the CSV file yourself to get a sense of what CSV files look like.

```
5.0,3.6,1.4,0.2,Iris-setosa
6.3,2.3,4.4,1.3,Iris-versicolor
7.2,3.0,5.8,1.6,Iris-virginica
```

To read CSV files, Python offers a module called `csv`. This module allows using `csv.reader()`, which reads a file row by row. For each row, it returns a list of columns that can be processed as needed. The following is an example of how a CSV file can be read in Python.

```
import csv

with open("file.csv") as f:
    for cols in csv.reader(f):
        print(int(cols[0]) + int(cols[1]), cols[2])
```

Assuming that the following are the contents of `file.csv`

```
1,2,Row 1
3,4,Row 2
5,6,Row 3
```

then the previous code snippet will print:

```
3 Row 1
7 Row 2
11 Row 3
```

Note that `cols[0]` and `cols[1]` are being converted to integers (with `int`) before adding them. By default, `csv.reader` converts all fields read into strings (`str`). If the values are instead to be treated as another type (e.g. `int`, or `float`), the values need to be explicitly converted. As a side exercise, what would happen if `cols[0]` and `cols[1]` were not casted to `int`?

Citybik.es

Citybik.es is a website that offers an Application Programming Interface (or API, for short) for the usage of bike-sharing services throughout the world. Among the others, data for one of Turin's bike sharing system ([TO]Bike) is available. For [TO]Bike, the information available is at a "station" granularity. This means that all the data available regards the bike stations: some of the useful information available is the station name, its position (in terms of latitude and longitude), the number of available bikes and the number of free docks. The data is offered in near real-time (i.e. it is updated every 15-30 minutes).

The API endpoint to request the data about for the [TO]Bike service is the following:

```
http://api.citybik.es/v2/networks/to-bike
```

You can either download the data from your browser, or using `wget` (as shown for the Iris dataset).

This dataset is in the JSON (JavaScript Object Notation) format. This format allows storing complex data structure (i.e. not only tabular data). You can either store basic data types (strings, numbers, boolean) or lists of them (e.g. `[0, true, "test"]`) and dictionaries (e.g. `{ "key1": "value1", "key2": false, "key3": [3, 2, 1]}`).

Lists and dictionaries are the same ones found in Python (`list` and `dict`). As such, any JSON file can be loaded as a Python data structure (which can possibly be nested). To do that, a Python module called `json` is available. It can be used as follows:

```
import json

with open("file.json") as f:
    obj = json.load(f)
    print(obj)
```

In this example, the file `file.json`, which contains a JSON object, is opened (with `open()`) and read (with `json.load()`). The contents of the object are then stored into the `obj` variable.

You can read more about JSON on Wikipedia.

MNIST

The MNIST dataset is another particularly famous dataset. It contains several thousands of hand-written digits (0 to 9). Each hand-written digit is contained in a 28×28 8-bit grayscale image. This means that each digit has 784 (28^2) pixels, and each pixel has a value that ranges from 0 (black) to 255 (white). Figure 2 shows one such digit.

The dataset can be downloaded from the following URL:

```
https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/mnist_test.csv
```

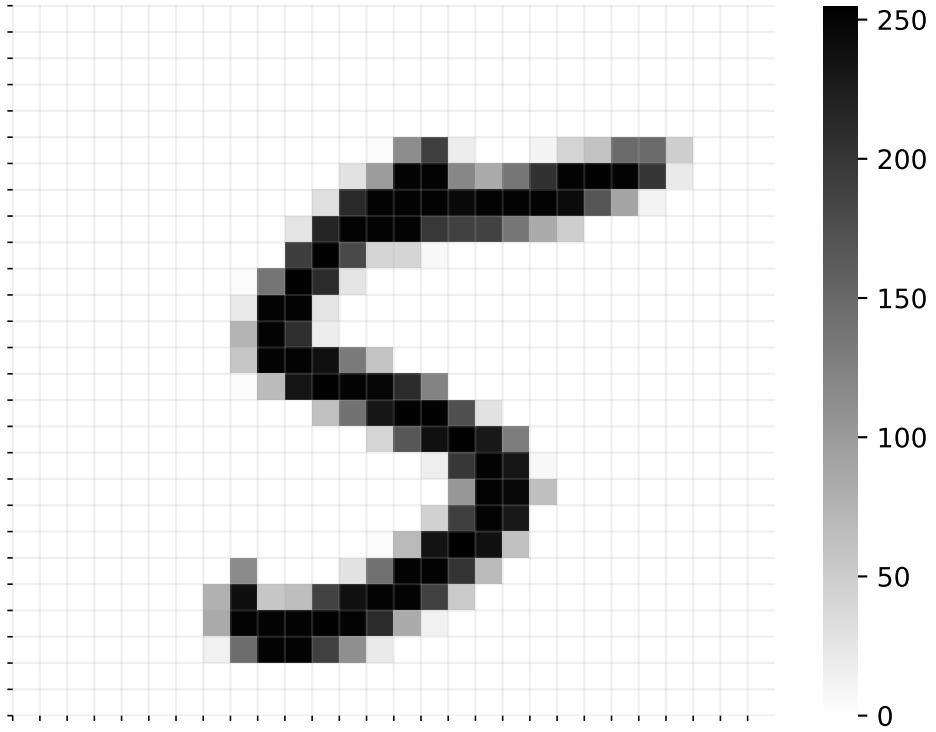


Figure 2: One of the digits from MNIST

In this case, MNIST is represented as a CSV file. Similarly to the Iris dataset, each row of the MNIST datasets represents a digit. For the sake of simplicity, this dataset contains only a small fraction ($10,000$ digits out of $70,000$) of the real MNIST dataset, which is known as the MNIST test set (you will learn more on training and test sets). For each digit, 785 values are available: the first one is the numerical value depicted in the image (e.g. for Figure 2 it would be 5). The following 784 columns represent the grayscale image in row-major order (for more information about row- and column-major order of matrices, see Wikipedia).

The MNIST dataset in CSV format can be read with the same approach used for Iris, keeping in mind that, in this case, the digit label (i.e. the first column) is an integer from 0 to 9, while the following 784 values are integers between 0 and 255.

2 Exercises

Note that exercises marked with a (*) are optional, you should focus on completing the other ones first.

2.1 Iris dataset

1. Load the previously downloaded Iris dataset as a list of lists (each of the 150 lists should have 5 elements). You can make use of the `csv` module presented. You can read more about the `csv` module on the official documentation.

2. Compute and print the mean and the standard deviation for each of the 4 measurement columns (i.e. sepal length and width, petal length and width). Remember that, for a given list of n values $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, the mean μ and the standard deviation σ are defined respectively as:

$$\mu = \frac{1}{n}\sum_{i}^{n} x_i$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i}^{n}(x_i - \mu)^2}$$

3. Compute and print the mean and the standard deviation for each of the 4 measurement columns, separately for each of the three Iris species (versicolor, virginica and setosa).

4. (*) Based on the results of exercises 2 and 3, which of the 4 measurements would you considering as being the most characterizing one for the three species? (In other words, which measurement would you consider "best", if you were to guess the Iris species based only on those four values?)

5. (*) Based on the considerations of Exercise 3, assign the flowers with the following measurements to what you consider would be the most likely species.

```
5.2, 3.1, 4.0, 1.2
4.9, 2.5, 5.6, 2.0
5.4, 3.2, 1.9, 0.4
```

2.2 Citybik.es dataset

1. Load the previously downloaded Citybik.es dataset as a Python dictionary. You can make use of the `json` module presented. You can find the full documentation for the `json` module here. After the dictionary is loaded, explore its contents from an interactive shell.

   Info: you can receive a dictionary's keys and values with its `.keys()` and `.values()` methods.

2. Count and print the number of active stations (a station is active if its `extra.status` field is `"online"`).

3. Count and print the total number of bikes available (field `free_bikes`) and the number of free docks (field `empty_slots`) throughout all stations.

4. (*) Given the coordinates (latitude, longitude) of a point (e.g. $45.074512, 7.694419$), identify the closest bike station to it that has available bikes. For computing the distance among two points (given their coordinates), you can use the function `distance_coords()` defined in the code snippet below (which is an implementation of the great-circle distance):

```
from math import cos, acos, sin

def distance_coords(lat1, lng1, lat2, lng2):
    """Compute the distance among two points."""
    deg2rad = lambda x: x * 3.141592 / 180
    lat1, lng1, lat2, lng2 = map(deg2rad, [ lat1, lng1, lat2, lng2 ])
    R = 6378100 # Radius of the Earth, in meters
    return R * acos(sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(lng1 - lng2))
```

2.3 MNIST dataset

1. Load the previously downloaded MNIST dataset. You can make use of the `csv` module already presented.

2. Create a function that, given a position $1 \leq k \leq 10,000$, prints the k^{th} digit of the dataset (i.e. the k^{th} row of the csv file) as a grid of 28×28 characters. More specifically, you should map each range of pixel values to the following characters:

   - $[0, 64) \rightarrow$ `" "`
   - $[64, 128) \rightarrow$ `"."`
   - $[128, 192) \rightarrow$ `"*"`
   - $[192, 256) \rightarrow$ `"#"`

   So, for example, you should map the sequence `0, 72, 192, 138, 250` to the string `" .#*#"`. For Figure 2 (which is the 130^{th} image in the dataset), the resulting output would be:

```
          .#      **
         .##..*#####
        ##########.
       ########***.
      ###*
     *###
     ##
    .##
     ###*
     .#####.
        *####
          *####*
            ###
             .##
             ###
           .###
  .      *###.
 .# .*####*
.######.
 *###.
```

3. Compute the Euclidean distance between each pair of the 784-dimensional vectors of the digits at the following positions: 26^{th}, 30^{th}, 32^{nd}, 35^{th}.

   Info: Remember that Python arrays are indexed from 0, so the k^{th} value will be at position $k-1$

4. (*) Based on the distances computed in the previous step and knowing that the digits listed in Exercise 3 are (not necessarily in this order) $7, 0, 1, 1$, can you assign the correct label to each of the digits of Exercise 3?

5. (*) There are 1,135 1's and 980 0's in the dataset. For all 0's and 1's separately, count the number of times each of the 784 pixels is black (use 128 as the threshold value). You can do this by building a list `Z` and a list `O`, each containing 784 elements, containing respectively the counts for the 0's and the 1's. `Z[i]` and `O[i]` contain the number of times the i^{th} pixel was black for either class. For each

value i, compute `abs(Z[i] - O[i])`. The i with the highest value represents the pixel that best separates the digits "0" and "1" (i.e. the pixel that is most often black for one class and white for the other). Where is this pixel located within the grid? Why is it?