Ch2 Ex1: Comparing two databases

Pablo Martínez Agulló
Introduction to Machine Learning for Scientific Computing

April 9, 2019

The aim of this exercise is to compare OECD and IMF databases to check if there is a correlation. In the first case, the *Better life index* is presented for each country. In the second we see the *Gross domestic product per capita* depending on the country.

1 Procedure

My python script for this exercise is at:
`https://github.com/MartinezAgullo/PhD-Course-ML/blob/master/ML_Ch2_Ex_1.py`

1.1 Download the databases

The Organisation for Economic Cooperation and Development (OECD) statistics page includes data and metadata for OECD countries and selected non-member economies. Going to `https://stats.oecd.org/` we can access to this information and the *Better life index* can be found in the Social Protection and Well-being tab. This database is download by clicking the Export button and choosing the type of file (CSV, Excel, PC-Axis, etc). We choose the `.xls` format.

The International Monetary Fund (IMF) main page has a tab for data where a range of time series data on IMF lending, exchange rates and other economic and financial indicators are published. The data we are going to use is stored in the World Economic Outlook (WEO) database (`https://www.imf.org/external/pubs/ft/weo/2016/01/weodata/index.aspx`). We click in `Download WEO Data: April 2016 Edition` and `By Countries (country-level data)`, then we select the Gross domestic product per capita in dollars and prepare the report. In the bottom of the page, there is a button to download the database in Excel format by taping on `Your WEO Report`.

1.2 Prepare the data

We load the datasets on the Excel or LibreOfficeCalc. First of all, we make sure that we are loading it in a way that the info is distributed in columns as we had it in the web page (when the numbers have dots and commas, they are often separated in different columns). Then we prune the columns which contain the info that is not relevant and the rows with

the heads. Then we remove the dots every three digits and save the document as `.csv` file. In my case it was not possible to do this therefore I created a `.txt` and copied the columns into it. Using the find and replace tool I, firstly, changed the spaces that defined the columns by the dot and comma (;) and, secondly, the comma of the decimals by a dot.

1.3 Read the data

Firstly we install the pandas library of Python. We read the `.csv` using the command `pandas.read_csv()`. Our code looks like:

```
import pandas as pd
gdp_per_capita = pd.read_csv('IMF.txt', delimiter = ';')
better_life_index = pd.read_csv('OECD.txt', delimiter = ';')
```

Now we need to find the elements that are coincident, i.e., the inner join of the two lists. For this purpose, the `merge()` command is used. This command creates a dataframe with the information of the the other two. We can specify if we want the inner, outer, left or right join using the following syntax: `df1.merge(df2, how = 'outer')`, being `df1` and `df2` the two merged dataframes. The default value is inner.

At this point we have 180 countries in the IMF list and 30 in OECD, all of them in the IMF list too. With this we can play easily show some stats like, for instance, which country has the higher GDP. For use the `max()` and `loc[]` commands:

```
print("Country:"+ gdp_per_capita.loc[gdp_per_capita['GDP'].idxmax()]['Country'])
print("GDP:"+gdp_per_capita.GDP.max())
```

1.4 Data manipulation

First of all, it is necessary to split the dataset in three parts:

- Training (50%): The sample of data used to fit the model
- Validation (25%): The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters
- Testing (25%): The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. It is only used once a model is completely trained

As we do not want to take biased subjoins of the main dataset, we have to make the sub-datasets following a random distribution. For this, Numpy has the function `np.random.choice`. The following line creates an array of `int(df.Country.count()/2)` random integers from 0 to `df.Country.count()` without repetition.

```
import numpy as np
np.random.choice(df.Country.count(), int(df.Country.count()/2), False)
```

Using this array of random numbers (named `index1`) in a loop over the elements of our merged dataset will allow us to separate the datasets

```
Training_df = pd.DataFrame(columns = ['Country', 'BLI', 'GDP'])
Aux_df = pd.DataFrame(columns = ['Country', 'BLI', 'GDP'])
i = 0
```

```
while i < int(merged.Country.count()):
      if i in index1:
        Training_df = Training_df.append(merged.loc[i], ignore_index=True)
      else:
        Aux_df = Aux_df.append(merged.loc[i], ignore_index=True)
      i = i+1
```

Following the same procedure, the validation and test dataframes are obtained from the `Aux_df` dividing it by two.

This procedure to split the datasets using loops can be extrapolated to any other distribution. For instance, if we want to split our data in two subsets with 70% (train) and 30% (test) we could do just one loop with another definition of `index1`:

```
np.random.choice(int(df.Country.count()), int(df.Country.count()*0.7), False)
```

1.5 Model

In this exercise we are using a linear model to relate the BLI to the GDP per capita. Once we have the datasets split we perform the linear regression:

```
slope, intercept, r_value, p_value, std_err = stats.linregress(df['GDP'],df['BLI'])
```

In this case we have user `stats` from `scipy` library

2 Results

Our model predicts:

$$BLI = intercept + slope \times GDP$$

$$BLI = 4.962 + 4.429 \cdot 10^{-5} \times GDP$$

Due to the fact that everytime that we run the scrip the training set is different, the output of the linear regression is going to be different. The results disiplayed above correspond to once of these runs. The correlation index in this case – Figure 1– was $R = 0.816$.

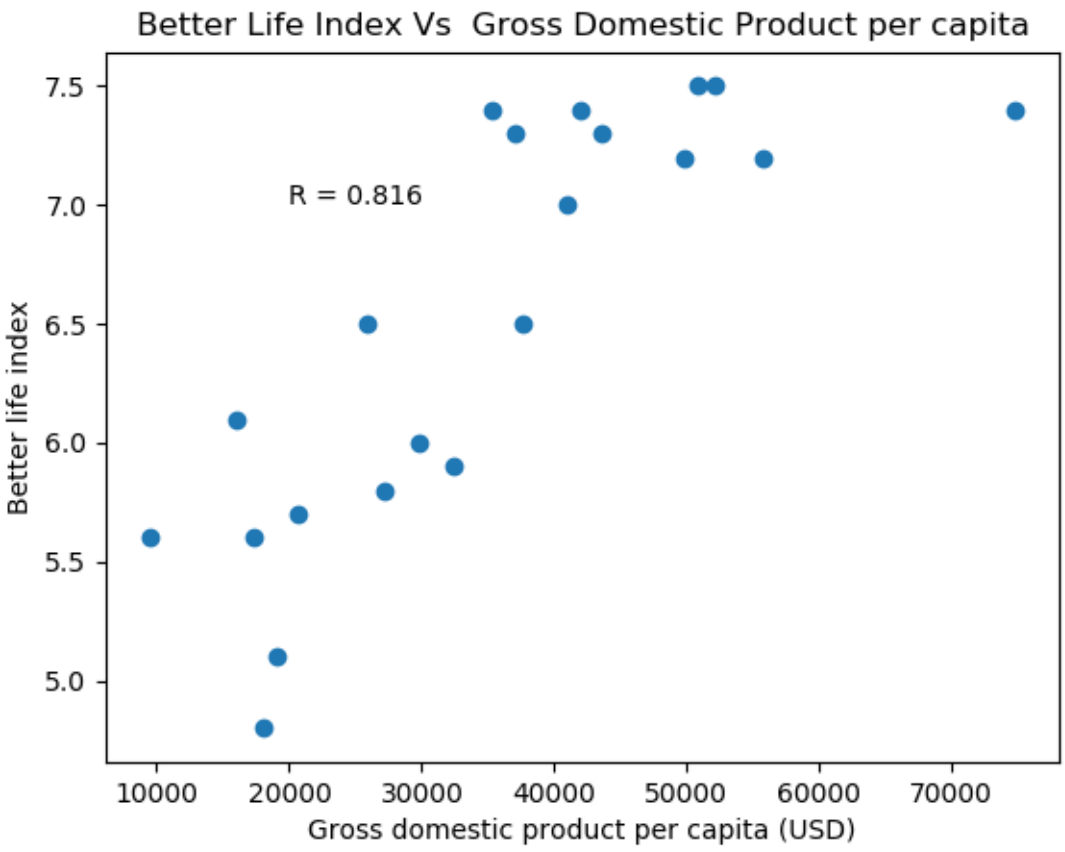

Figure 1: Distribution of the training set