Data Science Lab
Lab #8

Politecnico di Torino

Intro

The main objective of this laboratory is to put into practice what you have learned on regression techniques. You will test several techniques on data with different shape and complexity. In particular, you will build models to approximate univariate analytical functions and multivariate synthetic datasets. Finally, you will adopt the aforementioned techniques to carry out a forecasting task.

1 Preliminary steps

1.1 Datasets

1.1.1 Synthetic generated data

In the first part of this laboratory you will practise what you learned about regression techniques with some trivial datasets synthetically generated by you. In particular you will generate 4 different datasets: the first 3 can be generated starting from some analytical functions while the fourth one will be generated exploiting the make_regression function available in the scikit-learn library (please refer to the official documentation to learn how this function works).

Learning how to generate synthetic dataset is a very useful skill when you are developing new machine learning algorithms or pipelines to properly test your approaches with known data distributions.
In particular, when you want to solve very complex problems, sometimes you cannot trust the results obtained on real datasets, thus it is suggested to test your algorithms on some custom generated data distributions to ensure that they are correct and that they generalize – and then apply them to the real use cases.

1.1.2 Second World War temperatures

The dataset for the second part of the laboratory is a real dataset made available by the National Oceanic and Atmospheric Administration and hosted on Kaggle.

The dataset contains daily information on weather conditions collected during the World War II, starting from `1940-01-01` until `1945-12-31`. The data has been collected by 159 weather stations located all around the world for a total of 119040 recordings.
Measurements like `Min Temperature`, `Mean Temperature` and `Max Temperature` have been recorded by the sensors other then the `Precipitations` in mm and others.
The dataset page at this link contains all the details for each columns of the dataset.
Two files are available in the dataset:

- *Summaryof Weather.csv*: it contains the daily measurements collected by the sensors. The fields that are useful for this laboratory are:
  - `STA`: id of the weather station
  - `Date`: the date of the recording
  - `MaxTemp`: daily maximum temperature recorded in degree Celsius
  - `MinTemp`: daily minimum temperature recorded in degree Celsius

Figure 1: Sensor *22508* location.

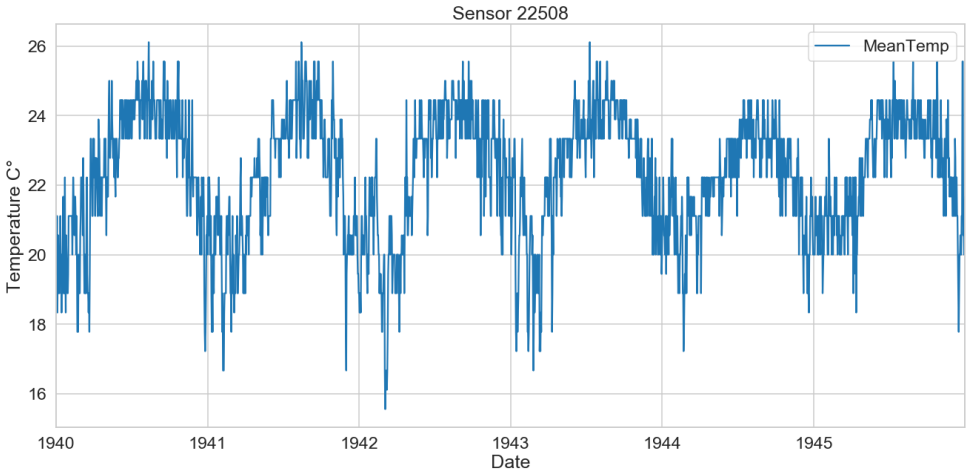

Figure 2: Sensor *22508* daily mean temperature.

  - `MeanTemp`: daily mean temperature in degree Celsius

- *WeatherStationLocations.csv*: it contains some metadata related to the sensors like the `Latitude` and the `Longitude` for each sensor. This file is not relevant for the purposes of this laboratory. For further details refer to the information available at this link.

Notice that not all the weather stations collected data for the whole period, thus, the temporal series are characterised by a lot of missing values in many cases. For simplicity you will work only with the data collected by the sensor identified by the id: *22508*. This sensor is located at `{Lat:21.48333333, Lon:-158.05}`, corresponding to the Honolulu Hawaii island (Figure 1). The mean temperature measured by sensor *22508* will look like the series showed in Figure 2.

You can download the dataset at:

`https://github.com/dbdmg/data-science-lab/raw/master/datasets/weatherww2.zip`

2 Exercises

In this laboratory you have to build a regression model for several analytical functions as well as for a structured dataset generated with scikit-learn. While many libraries and out-of-the-shelf algorithm implementations are widely available, there is still a lack of sufficient data to test them out. Synthetic data generation is a powerful tool we would recommend to master, as it becomes extremely useful to test and optimize your models.

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

2.1 Univariate regression on analytical functions

In this exercise you will build a regression model to approximate several analytical functions. Then, you will evaluate and discuss the quality of your models with the final goal to optimize their respective hyperparameters. The studied functions will be:

$$f_1(x) = x \cdot \sin(x) + 2x \tag{1}$$

$$f_2(x) = 10\sin(x) + x^2 \tag{2}$$

$$f_3(x) = sign(x)(x^2 + 300) + 20\sin(x) \tag{3}$$

1. Use the following snippet of code to create you initial dataset. Note that `f(x)` should match one of the functions mentioned before.

```
import numpy as np
from sklearn.model_selection import train_test_split

tr = 20
n_samples = 100
X = np.linspace(-tr, tr, n_samples)
y = f(X)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=0.7, random_state=42, shuffle=True
)
y_test = y_test[X_test.argsort()]
X_test.sort()
```

2. Draw and inspect the shape of the function. Which regression model of those you know could achieve better performance?

3. Frame now a regression task to your generated data. Start from the ordinary least squares Linear Regression. Then, choose additional models which you believe could outperform linear regression to approximate the function.

> **Info:** There are many regression models in scikit-learn, other than the ones that you should already be familiar with (e.g. LinearRegression, Ridge, SVR). Part of the models that you adopted for classification have their regression counterparts, such as MLPRegressor and RandomForestRegressor.

   Fit each model to the training data and predict the function value for each test point.

4. Evaluate your regression outcome in terms of a metric of your choice. Refer to `sklearn.metrics` module to find some. Does any model stand out in terms of regression quality in every case or the shape of the function favors (or penalizes) any of them?

5. Back in Point 2, you could visually inspect the function shape[1]. Although this is not always the case, you can exploit this sort of knowledge to enhance your initial representation. With this in mind, choose a set of new features (e.g. polynomials, trigonometric functions) and introduce them in training data. Then, test again your best performing model and discuss whether your scores have improved or worsened.

6. (*) As many real-world tasks request, your model must be robust to noise. To test the model you developed in previous points, you can shift your initial data injecting some random noise. You can add a noise component to each point with the following function:

   ```
   def inject_noise(y):
       """Add a random noise drawn from a normal distribution."""
       return y + np.random.normal(0, 50, size=y.size)
   ```

   Note that you are adding a noise factor drawn from a normal (i.e. Gaussian) distribution with zero mean and standard deviation equal to 50.

   Plot again your data and inspect the new distribution. Then, test your regression models and compare the outcomes with the previous ones. Which model has proved to be more robust to the introduced variation?

[1] Actually, you have one further information, i.e. the *exact* distribution of your data given by the mathematical formula.

2.2 Multivariate regression on synthetic data

In this exercise, you will carry out a multivariate regression analysis. In Exercise 2.1, you addressed a univariate problem where the goal was learning the function mapping between a single input and a single output. Instead, dealing with more complex and high-dimensional data is commonplace in reality. As an example, imagine a car selling service. In that context, a wide, heterogeneous set of data about every car model is gathered (e.g. the engine type, the presence of additional features, the number of sales, etc.). A crucial, yet not straightforward, regression task would be to model the relationship between these attributes and the retail price. Hopefully, experiments would discover that some variables are more (or less) significant than other. This information could then be turned into a business advantage.

In the context of this exercise, you will work on a synthetic, multi-feature dataset generated to match a multivariate regression problem as described for the car selling service.

1. Generate a random synthetic dataset for a regression problem using the scikit-learn's make_regression function. Take your time to understand the construct parameters and their default values. Start with at least 2000 samples and a fixed random state.

   ```
   X, y = make_regression(n_samples=2000, random_state=42)
   ```

   

   Info: a fixed random state helps you to reproduce the outcome. You have to use it whenever you want your results, obtained from any random initialization, to be the same for different runs. This functionality is available for every scikit-learn object that makes use of a random state.

2. Test the regression pipeline that you developed in Exercise 2.1. Pay enough attention to the differences with the previous exercise:

   - How does your model handle the presence of multiple features?
   - Is there any correlation among features? How does this impact the model performance?

3. Using the function constructor, make the problem harder for your regression model. Try to regenerate the dataset adding some noise using the `noise` parameter and to increase or reduce the gap between `n_features` and `n_informative` features.

   - How does the model behave in this case?
   - Train a Linear Regressor and inspect the coefficients learned for the non-informative features. What do these values mean?

2.3 Temperature series forecasting

Forecasting is the process of making predictions of the future based on past and present data (see forecasting on Wikipedia). The forecasting problem can be solved with machine learning methods (e.g. Linear Regression, Random Forest Regression, Neural Networks) and with complex statistical methods (e.g. ARIMA, Holt Winter). In this exercise you will work with the *Second World War Weather* dataset trying to predict the next day temperature given the historical trend of temperatures.

1. Load the *Second World War Weather* dataset. In particular you have to load the data contained in *Summaryof Weather.csv*.
2. Inspect the content of the dataset identifying if there are missing values for the sensors in the dataset, checking for the top 10 most complete sensors (in terms of collected data) the distribution of the recorded mean temperatures (`MeanTemp` column).
   - Can you identify if these sensors are located in part of the world with similar weather conditions?.
   - Is it necessary to normalize the data in this case?
   - Which pre-processing step can be useful to solve the forecasting task?
3. For simplicity, we will exploit the data collected by a specific sensor. Filter the data by `STA` (Station) and extract the mean temperature measurements corresponding to sensor with id `22508`. See 1.1 for more details on this sensor.

   Info: Do not forget to keep track also of the date on which each measurements has been taken and be sure that each date is properly converted to a Datetime data type like the datetime64 type provided by numpy.

4. Now you should have an array containing all the measurements collected by sensor `22508` representing the time series of the mean temperatures. Plot the temperatures time series exploiting matplotlib and show on the x axis the corresponding datetimes.
5. To predict the next day temperature, a rolling window pre-processing should be performed to convert the time series into a structured dataset. The rolling windows on the time series can be applied by extracting, from the original time series, a window of length W that moves over the series, with step 1, like the example showed in Figure 3. The dependent variable to be predicted will be the value positioned at $t + W + 1$, where t is the position of the rolling window. The final structure of the dataset extracted with the rolling window will be similar to the one showed in Table 1.

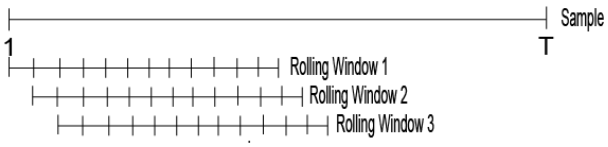

Figure 3: Rolling window

Table 1

Window$_t$	t	$t+1$	...	$t+W$	$t+W+1$
Window$_0$	...	...	...	...	...
Window$_1$	...	...	...	...	...
...	...	...	...	...	...
Window$_{T-W-1}$	...	...	...	...	...

6. Split the dataset into train and test sets and keep in mind that you are evaluating the forecasting of a time series. To train your model you can consider to use the data from 1940 to 1944 and test the trained models on 1945 data. A different way of performing the cross validation on time series is the TimeSeriesSplit strategy available in Scikit-learn.

7. Now, using the regression techniques that you have learned, try to identify which is the most accurate regression model evaluating the r2_score and the mean_squared_error.

8. To visualize the forecasted time series with respect to the real test time series, plot the test values and the forecasted values on the same chart.

    - Are the two series similar?
    - Have the seasonality and the trend of the series been correctly predicted?
    - Are you able to predict a horizon longer than 1 day with this kind of techniques? Why?

9. Try to change the pre-processing step and the hyper-parameters of the pipeline and repeat the analysis to see how your model performance will change.