INFO-H515 : Big Data: Distributed Data Management and Scalable Analytics

Scalable analytics - Bike count forecasting in Brussels

Authors:
Mehdi **MOUTON**
Damiano **BERTOLDO**
Loic **QUIVRON**
Pascal **TRIBEL**

Professors:
Dimitris **Sacharidis**
Antonios **Kontaxakis**

Academic year 2022-2023

1 Introduction

Brussels Mobility Bike Count API is an API that provides the number of bikes passing in front of multiple sensors placed in Brussels. It gives multiple information, including the number of bikes and their speed, every 15 minutes.

The data can be used to assert which pair of sensor are the most correlated to each other. This can be very useful in a real world senario as it facilitates the identification of the routes most frequently used by bicycles. To assert the correlation it is used the *Pearson correlation coefficient*:

$$r_{ij}(t) = \frac{\sum_{n=1}^{t}(c_i(n) - \overline{c_i}(t))(c_j(n) - \overline{c_j}(t))}{\sqrt{\sum_{n=1}^{t}(c_i(n) - \overline{c_i}(t))^2}\sqrt{\sum_{n=1}^{t}(c_j(n) - \overline{c_j}(t))^2}} \tag{1.1}$$

In this study we will use different strategies of calculating this coefficient, that can be all the historical data at once, analyzing streaming data, or applying a sliding window approach.

2 Preprocessing

After downloading the data of every sensor, the task was to assure that every sensor didn't have missing values. To assert this problem we have filled the missing values with 0 numebr of bikes passed through the sensor and -1 as the mean speed for every missing date and time gap.

Since the assignment require that all data is consolidated into a single file, the second task involved merging the data from all sensors into a unified file, thereby creating *data.csv*.

2.1 Task preprocessing

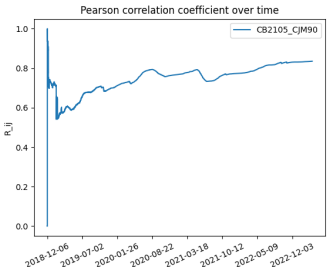
Before computing the coefficient, a series of operation must be done in order to have the right dataset. Given that the original dataset column *Sensor* resulted in a repititon of each *Date* and *Time gap*, we pivoted the dataframe. This reconstruct allowed us to have a unique row for each combination of *Date* and *Time gap*. This adjustment made it easier to compute correlations between two sensors, as we just had to select the corresponding columns of each sensor.

3 Batch processing

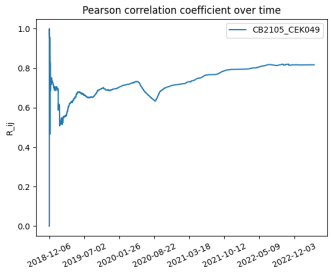
Batch processing is implemented using *Spark*, loading the entire dataset into a *Spark dataframe*. For computational reason only the last *Pearson coefficient* of each sensors pair were used for asserting the most correlated sensors. This is because *Spark* uses the lazy evaluation, therby reducing the computation cost.

Before doing any type of calculation we had to do a task preprocessing, es explained in Section 2.1.

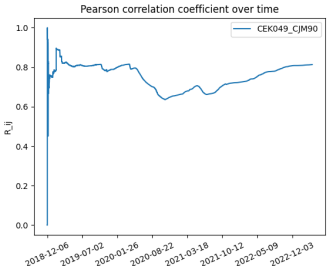
Due to the size of the complete dataset for all sensor pairs, handling the full timeseries calculation for each sensor pair was unfeasible. As a result, we utilized only the most recent values for our computations.



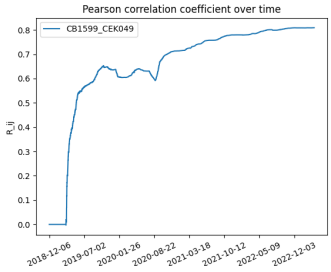
(a) Sensors CB2105 and CJM90



(b) Sensors CB2105 and CEK049



(c) Sensors CEK049 and CJM90



(d) Sensors CB1599 and CEK049

Figure 1: Plot of the top 4 most correlated sensor at 2023-03-31

In Figure 1 we can see trend of the top 4 most correlated sensor using the entire time series of the *Pearson correlation coefficient*.

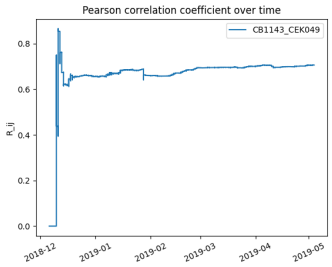
As we can see the correlation is not costant and can have multiple changes through time. This can be due to multiple reason, like street repearing or other obstructions. Also we can notice in Figure 1d that the initial correlation was zero due to the absence of sensor data at the beginning. However, as data accumulated over time, the correlation increased. Then, for what concerns Figure 1a,1b,1c, we can see a spike at the beginning. This was due to the fact that there wasn't enough data to have a clear reliable value.

4 Stream processing

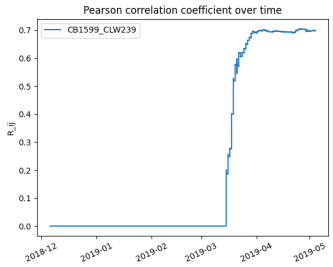
Data streaming is implemented using *Spark*. It flows data from a *Producer* to a *Consumer*. The data used in this paper starts on the 6th of December 2018. There are two personalizable parameters Δ and Π, that correspond to the interval in which the consumer send the data and how many days of data to send.

4.1 Producer

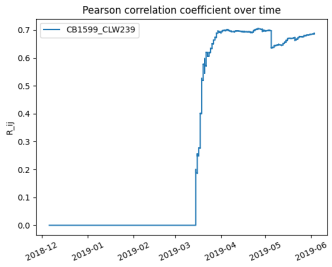
The *producer* collects the data range needed of Π days and send it to the *consumer* every Δ time. Given that the number of days doesn't significantly impact the results, we focused our attention on the value of Δ. We have choose a value of 2 minutes because it is the time that the *consumer* takes to compute all the necessary operations.



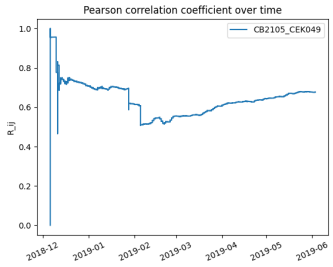
(a) 4th most correlated sensors at iteration 5



(b) 5th most correlated sensors at iteration 5



(c) 4th most correlated sensors at iteration 6



(d) 5th most correlated sensors at iteration 6

Figure 2: Comparison after an iteration on the Stream processing

As said before, changing the value of Π doesn't impact the result as it only transfer more/less data every Δ seconds, in this case we have experimented with a value of 30 days.

4.2 Consumer

Once the data is sent by the *Producer*, the *Consumer* receive the data and compute all the calculation that comprehend a task preprocessing, Section 2.1, and the calculation of the *Pearson coefficient*.

Given that the data is received in Π-day batches, we utilize *Spark* to store this data into a CSV file. Each new batch of data received is appended to the existing data in this file. This allowed us to always have, for each computation, all the historical data, so we could make a reliable calculation of the *Pearson coefficient*.

When we calculated the correlation coefficient batch by batch, we have observed that the top five most correlated sensors can vary as we can see in Figure 2. Here there was a change between the iteration 5 (Figures 2a,2b) and 6 (Figures 2c,2d) of the top 5 most correlated sensor.

As also said in Section 3, a changes in the coefficient could be caused by various problems, like street repairing or other obstructions.

5 Sliding window processing

Taking advantage of Spark Streaming windowing functionality, we can compute the correlation coefficient over a sliding window as illustrated Figure 3

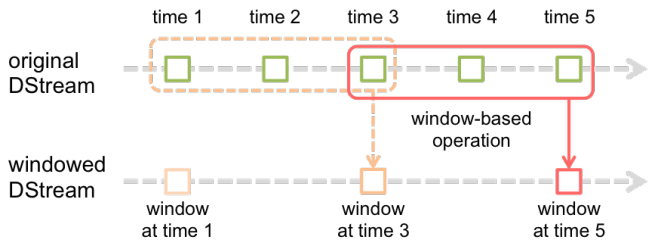


Figure 3: Illustration of the windowed computation process

As the producer side is the same as in Section 4.1, it will not be repeated here.

5.1 Consumer

On the consumer side, the main difference with what is described Section 4.2, is obviously the segmentation of the received data into windows. The result of this process are different to the one of Section 4.2 as the correlation coefficient is varying in time and the sliding window then processes the coefficients at different points in time hence the variation of results.

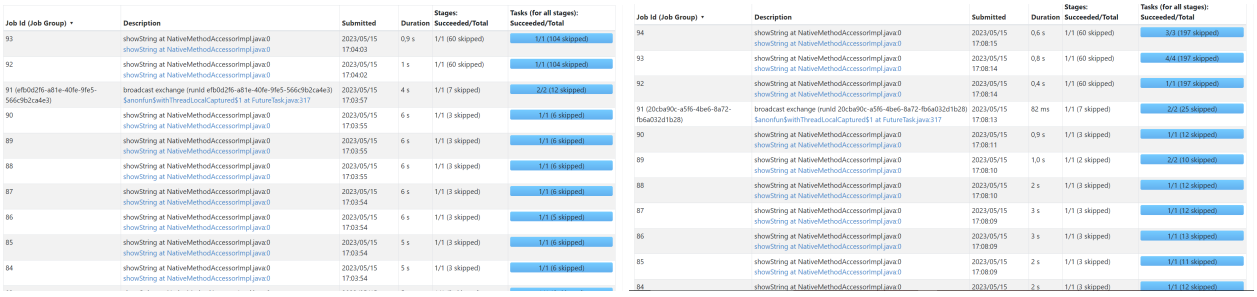
6 Scalability

To evaluate the scalability, we computed a DataFrame containing the correlation coefficients between the sensor *CAT17* and all the others. As Spark uses lazy computation, we used the command *show()* on the resulting dataframe so that all computations are done and we can evaluate the scalability. This command has been done in two different environments, the first one where Spark has been limited to 2 cores and the other one where it has been setup to use all available cores, in our case 8 cores. On Figure 5, one can observe a decrease in the computation time, proving the scalability of the process.

Another possible way to scale the computation is to give a specific subset of sensor pair to a specific node. Since doing the preprocessing needed (Section 2.1) is not computationally expensive.

```
Top5 most correlated sensors:
                      0
CB02411_CEK049  0.765724
CEK049_CJM90    0.750014
CB02411_CJM90   0.734156
CB2105_CJM90    0.707106
CB2105_CEK049   0.694289
51840it [00:00, 59838.03it/s]                                   (0 + 1) / 1]
100%|████████████████████████████████████| 153/153 [01:22<00:00,  1.85it/s]
Top5 most correlated sensors:
                      0
CB02411_CEK049  0.765724
CEK049_CJM90    0.750014
CB02411_CJM90   0.734156
CB2105_CJM90    0.707106
CB2105_CEK049   0.694289
51840it [00:00, 63186.17it/s]                                   (0 + 1) / 1]
100%|████████████████████████████████████| 153/153 [01:25<00:00,  1.79it/s]
Top5 most correlated sensors:
                      0
CEK049_CJM90    0.889583
CB02411_CEK049  0.880615
CB02411_CJM90   0.879970
CB1143_CB2105   0.705698
CB1143_CJM90    0.693394
51840it [00:00, 64189.38it/s]                                   (0 + 1) / 1]
100%|████████████████████████████████████| 153/153 [01:23<00:00,  1.83it/s]
Top5 most correlated sensors:
                      0
CEK049_CJM90    0.889583
CB02411_CEK049  0.880615
CB02411_CJM90   0.879970
CB1143_CB2105   0.705698
CB1143_CJM90    0.693394
51840it [00:00, 61495.73it/s]                                   (0 + 1) / 1]
100%|████████████████████████████████████| 153/153 [01:36<00:00,  1.59it/s]
```

Figure 4: Top 5 most correlated sensors over 4 iterations



(a) Spark taskboard with 2 cores

(b) Spark taskboard with 8 cores

Figure 5: Spark taskboard over two different environments

7 Conclusion

During this project, we leveraged Spark's capabilities in order to perform computations over a large dataset in three different settings. Batch processing, streaming and windowed streaming. We found out that each of these methods returns a different set of most-correlated sensors. This is expected as the correlation coefficient is itself a time series and thus varies over time, this variation in time explains the difference between the batch processing results and the streaming ones. *Link to the video presentation* : `https://drive.google.com/file/d/1ezfqG6mHonxhoiEJhYpeFXBCksI_bnf7/view?usp=sharing`