ECOLE
POLYTECHNIQUE
DE BRUXELLES

MS-BGDA – INFO-H515

Phase 2 – Forecasting Bike Counts

Scalable Analytics

AMRI Hakim
hakim.amri@ulb.be
000 459 153

CAILLIAU Julian
julian.cailliau@ulb.be
000 459 856

BAGUIA Rania
rania.baguia@ulb.be
000 459 242

JDAOUDI Mehdi
mehdi.jdaoudi@ulb.be
000 457 507

Project submitted under the supervision of
Prof. Dimitrios Sacharidis,
Prof. Gianluca Bontempi
and teaching assistants Théo Verhelst & Cédric Simar

Academic year
2022 - 2023

In the context of the
Specialised Master in Big Data and Data Sciences

Table of Content

List of Figures

List of Tables

List of Equation

Project Video

You can see the presentation video here:
https://universitelibrebruxelles.sharepoint.com/:v:/s/GRP_PROJ-INFO-H515BigDataMan.Ana/ETwTQDf3Z5hNgbmtaXkMJmAB7fV7I2KzD7jqDmNMu1SYAQ?e=UcUxQE

1. Introduction

For several years, the city of Brussels has been encouraging its citizens to use their bikes. To monitor the growth of bike use, 18 sensors recording the number of bikes passing by have been placed all over Brussels. The objective of this work is to predict the future bike counts that each sensor will see. It is important to note that this prediction is taken in a streaming context. Thus, it is necessary to be able to predict the future observation arriving after a certain interval without being able to use the data of all past observations.

The path to finding the best possible prediction is gradual. Obviously, this prediction is a function of the previously collected observations. The first naive prediction model implemented a first persistence model where the future observation is identical to the one just recorded. Then, this logic is pushed a little further as the average of the last V observations (with V = 2, 3, or 4) is used to make the prediction. Furthermore, several linear embedded models will be created. This will be calculated by applying the Recursive Least Squares method. To assess the best, their accuracy will be compared by computing the MSE between the 15-04-2022 and the 31-03-2023. Finally, additional features will be added to the previous model computations to further refine the predictions. To this end, the Spark engine will be used since it allows the parallelization of computations in addition to the analysis of data streams.

2. Producer notebook

First, it is important to mention some points regarding the "data.csv" file that the producer will read. The description of the pre-processing phase is available in phase 1 of the project. What is interesting to remember is that the file contains 6 columns: Date - Time Gap - Count - Average Speed - sensor - timestamp. The timestamp is an additional column to those initially available. It allows to have a continuous time dimension for each sensor. For example, for a sensor, its timestamp for the observation of 06-12-2018 with time group 96, will be 96. Then, its timestamp on 07-12-2018 at time group 1 will be 97.

Regarding the data stream creation, the socket-based approach has been applied in this work. to create the data stream which allows to send batches of data at regular intervals. The sending interval of data was set to 20 seconds and the batch size to 60 days of data. To be able to send the batch, its format was changed using the array2string() function.

3. Project Tasks – Consumer Notebook

3.1. Task 1 – First Persistence Model

3.1.1. Methodology

The first task is to build a model predicting the future bike counts of all sensors as being identical to the last observed bike count. The following formula is used to model the prediction:

$$\hat{y}(t+1, \text{sensor}) = y(t,\text{sensor})$$

Equation 1: First persistence formula

In addition to finding the value of the prediction, it was decided to already implement in the code the calculation of the MSE in anticipation of the following tasks.

Before any prediction can be made, the incoming data must first be prepared. The incoming DStream being a string, it must first be transformed back into a sequence of tuples for each observation. In a second step, the initial RDD is transformed into a paired RDD with as key the sensor and as value the observation. This then allows to group the observations by sensors using a partitionBy() and a groupByKey(). This step is crucial to make the predictions running concurrently in parallel for each sensor.

For each sensor at a timestamp t, a prediction of the sensor's count at t+1 is computed. This prediction needs to be evaluated with the observation at t+1.Therefore, the prediction processing requires to update states with updateStateByKey(). The states are initialized such that for each sensor, at each new batch, the states are updated to keep track of the last forecast, the number of observations, and the error. This will allow to compute the MSE throughout the stream.

3.1.2. Results

Some examples of the values forecasted as well as the MSE associated can be found in Table 1 in the appendix.

Figure 1 below can be used to assess the scalability of the algorithm. This figure represents the computing time in function of the number of cores used and the number of sensors retrieved. It is important to note that the computing time grows more slowly than the number of sensors. This indicates that the algorithm is scalable. It is interesting to mention that in this case, increasing the number of cores does not necessarily lower the processing time. Furthermore, it can be noted that when using 2 cores, the processing time was not influenced by the size of the data received. Erreur ! Source du renvoi introuvable. in the appendix shows how Spark distributes the computation among the cores.

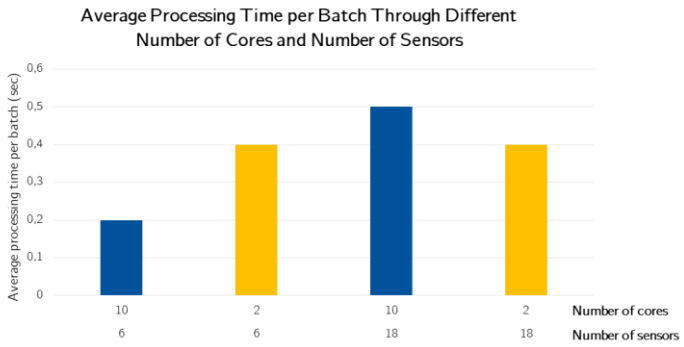

Figure 1 - Experimental results of the processing time in function of the number of sensors and the number of cores for the first persistence model

3.2. Task 2 – Weighted Persistence Model

3.2.1. Methodology

This task requires the implementation of a weighted persistence model whose equation is the following.

$$\hat{y}(t+1, sensor, V) = \frac{\sum_{n=0}^{V-1} y(t-n, sensor)}{V}$$

Equation 2 - Weight persistence model equation

The methodology and the function applied are almost the same as in the previous task. The only difference is that the last V (V = 2, 3, or 4) observations received are used. Therefore, it is necessary to keep track of the last V observed data. Then, the forecast for the following timestamp is computed as the average of the last V observations. In this case, the predictions start when there are enough observations. So, in the case of a prediction with a weighted average equal to 4, no prediction is given before there are four observations available. Also, similarly to task 1, the last observation of the batch that are needed to compute or evaluate the predictions in the following batch are stored in a state, with an updateStateByKey().

3.2.2. Results

Some examples of the values forecasted as well as the MSE associated can be found in Table 2 in the appendix.

The small changes between this task and the previous one has no influence on the parallelization. Indeed, the partitioning of the data is done by sensors which allows to compute the prediction concurrently. Figure 2 shows that, when the data size triples, the processing time only doubles. Moreover, parallelizing on more cores reduces noticeably the processing time. Both of these observations indicate that the model is scalable. On Erreur ! Source du renvoi introuvable. in the appendix is represented the execution by Spark of the function.

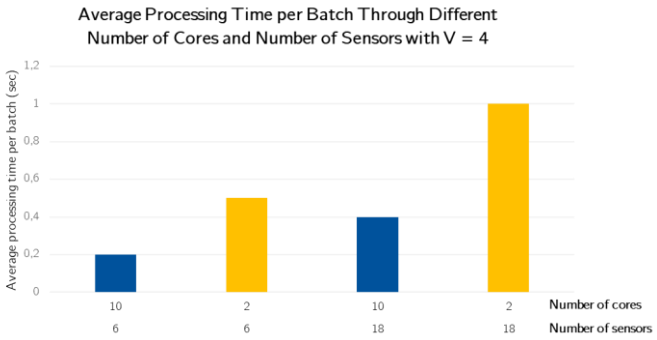

Figure 2 - Experimental results of the processing time in function of the number of sensors and the number of cores for the weighted persistence model

3.3. Task 3 – Embedded RLS

3.3.1. Methodology

In this task, the predictions are done according to the Recursive Least Square (RLS) algorithm. The RLS can learn while observing passing data to predict following observation in a regression setting. The formulas used to create this algorithm can be found in Equation 3 where V is the covariance matrix, $\hat{\beta}$ is the set of parameter estimated by the linear model, and ν is the forgetting factor set to 1. For this task, the last n observations (n = 1, 2, 3, or 4) are used, ie 4 different embedding orders.

$$\begin{cases} V_{(t)} & = \frac{1}{\nu}\left(V_{(t-1)} - \frac{V_{(t-1)}x_t^T x_t V_{(t-1)}}{1+x_t V_{(t-1)} x_t^T}\right) \\ \alpha_{(t)} & = V_{(t)} x_t^T \\ e & = y_t - x_t \hat{\beta}_{(t-1)} \\ \hat{\beta}_{(t)} & = \hat{\beta}_{(t-1)} + \alpha_{(t)} e \end{cases}$$

Equation 3 - Recursive Least Squares formulas

The process is very similar to the weighted averaged as the difference is only in the forecasting strategy. The parameter V, $\hat{\beta}$, and ν are initialized for each sensor. Those values are then used to make the forecasts. At each timestamp, the parameters are updated according to the new data and the forecasting error before making the new forecasting. At the last timestamp, the states are updated.

3.3.2. Results

Some examples of the values forecasted as well as the MSE associated with an embedding order of 1 and 2 can be found in Table 5 and Table 6 in the appendix. Increasing the embedding order significantly increases the MSE.

As observable in Figure 3, the scalability lies in the fact that the RLS formula can be applied in parallel with the observations grouped and partitioned according to the sensor. Same as in the previous model, the effects of the number of sensors and the number of nodes on the processing

time shows that the algorithm is scalable. Once again, the parallelization schema from the Spark interface is available in the appendix.

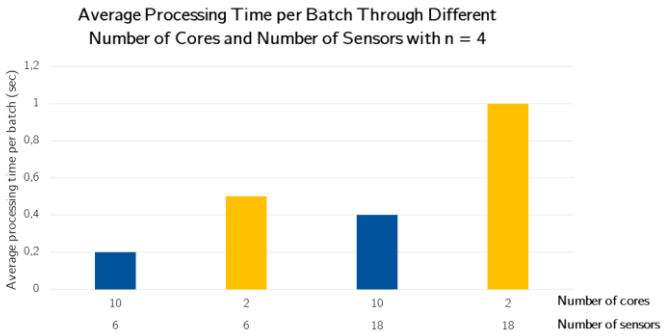

Figure 3 - Experimental results of the processing time in function of the number of sensors and the number of cores for the RLS model

3.4. Task 4 – Comparing Models Performance

3.4.1. MSE Comparison

For each model and embedding order, the MSE were computed over the interval of the 15-04-2022 and the 31-03-2023. All these results are presented in Table 8 in the Appendix.

It can be noticed that, among all the models, the persistence models have a lower MSE than the RLS models. Therefore, it can be concluded the persistence model have a better accuracy than RLS models. Furthermore, for 10 sensors out of 18, the first persistence model is the best model, followed by the weighted average persistence with V = 2 for 5 sensors, and V = 3 for the last 3 sensors.

Among the RLS models, the embedding order 1 model has an MSE in the same order of magnitude as the persistence models. However, a problem has emerged: for RLS models with embedding orders 2, 3 and 4, predictions and MSEs are systematically too high. The origin of the problem lies in the calculation of the $\hat{\beta}$ vector. The calculation of the $\hat{\beta}$ does not seem to be adapted in cases where the embedding order is higher than 1. It is moreover to limit the effects of these embedding orders that the forgetting factor has been fixed at 1.

3.4.2. Introducing time-related numerical features

The last task consists in adding time related features to assess the accuracy gain of the forecasting model. Three features were added: the hour in day, the day of the week and the month in which the observation is taken. These features are extracted in parallel to each incoming observation right after the DStream is read. Each observation is therefore made up of the date, the time group, the count, the speed, the sensor id, the timestamp, the hour, the day of the week and the month.

The same methodology for the implementation of the RLS as in the previous task is used. First, the sizes of the vector β and the covariance matrix will be equal to the sum of the embedding

order and the number of new features. Then, the features vector x_t is now composed of the counts of the last n observations (n being the embedding order), the hour, the day and the month of the last observation. After that, the prediction is done similarly than in the previous section.

3.4.3. Results

Some examples of the values forecasted as well as the MSE associated with an embedding order of 1 and 2 can be found in Table 9 and Table 10 in the appendix. Here increasing the embedding order increases the MSE as well.

Adding time-related numerical features has improved the accuracy of the predictions. In fact, for 14 sensors out of 18, the RLS with n = 1 has become the best model as it can be seen in Figure 4.

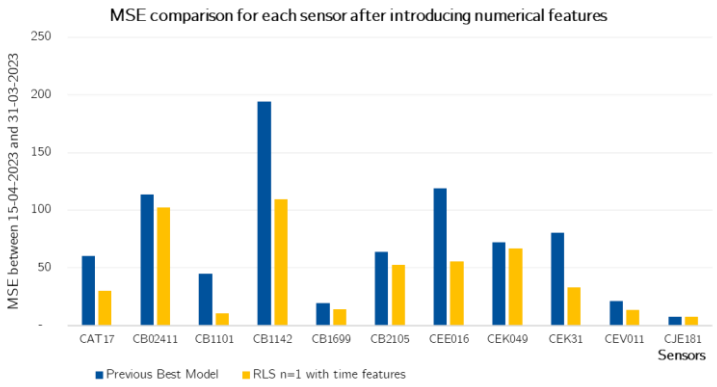

Figure 4 - MSE comparison for each sensor after introducing numerical features.

Regarding the scalability of the model, the conclusion is similar as in the previous section sine the model did not change except that more features are taken into account in the prediction process.

4. Appendix

4.1. First persistence model

```
('CB02411', {'SSE': 3836453, 'Forecast': 6, 'N': 151392, 'N_MSE': 33696, 'MSE': 113.85822822377207})
('CB1142', {'SSE': 6537940, 'Forecast': 6, 'N': 151392, 'N_MSE': 33696, 'MSE': 194.03294257308207})
('CB1143', {'SSE': 3263964, 'Forecast': 4, 'N': 151392, 'N_MSE': 33696, 'MSE': 96.86790324974032})
('CB1699', {'SSE': 739062, 'Forecast': 2, 'N': 151392, 'N_MSE': 33696, 'MSE': 21.93387742988574})
('CEK049', {'SSE': 2435587, 'Forecast': 4, 'N': 151392, 'N_MSE': 33696, 'MSE': 72.28333580649948})
('CJM90', {'SSE': 6689738, 'Forecast': 10, 'N': 151392, 'N_MSE': 33696, 'MSE': 198.53800267101943})
('COM205', {'SSE': 1486833, 'Forecast': 2, 'N': 151392, 'N_MSE': 33696, 'MSE': 44.126220507493 69})
```

Table 1 - First persistence model results

Number of cores	number of sensors	Batch i (s)	Batch j (s)	Batch k (s)	Average time (s)
10	6	0,2	0,2	0,2	0,2
10	18	0,5	0,5	0,5	0,5
2	6	0,4	0,4	0,4	0,4
2	18	0,4	0,4	0,4	0,4

Table 2 - Experimental results, for the first persistence model of the processing time when varying the numbers of cores and the number of sensors.

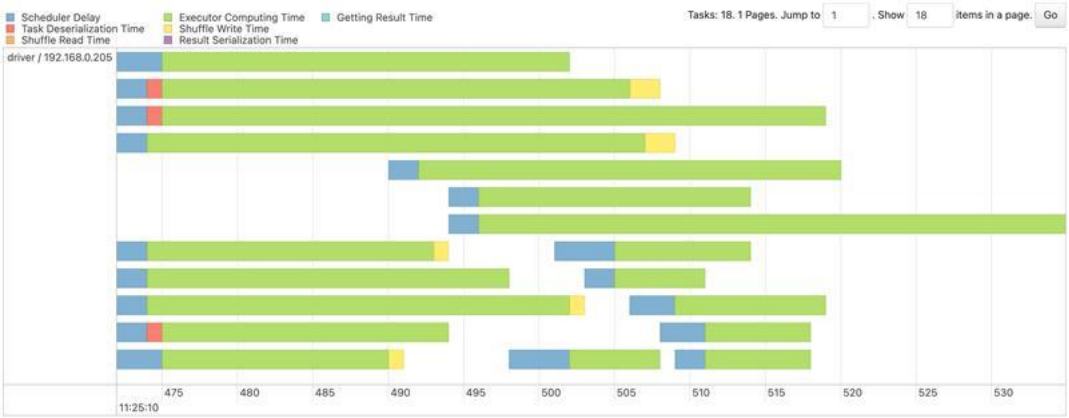

Figure 5 - Screenshot Spark User First persistence model

4.2. Weighted persistence model

$$\hat{y}(t+1, sensor, V) = \frac{\sum_{n=0}^{V-1} y(t-n, sensor)}{V}$$

Equation 4 - Weight persistence model equation

('CB1101', {'SSE': 1960886.5, 'Forecast': 10.5, 'N': 151391, 'N_MSE': 33696, 'MSE': 58.19517732601276, 'LastObservations': [('2023-03-31', 96, 11, 14, 'CB1101', 151392)]})
('CB2105', {'SSE': 2154775.5, 'Forecast': 9.0, 'N': 151391, 'N_MSE': 33696, 'MSE': 63.94941385962309, 'LastObservations': [('2023-03-31', 96, 5, 22, 'CB2105', 151392)]})
('CEE016', {'SSE': 4020281.5, 'Forecast': 2.5, 'N': 151391, 'N_MSE': 33696, 'MSE': 119.31388930108325, 'LastObservations': [('2023-03-31', 96, 5, 13, 'CEE016', 151392)]})

Table 3 - Weighted persistence model results

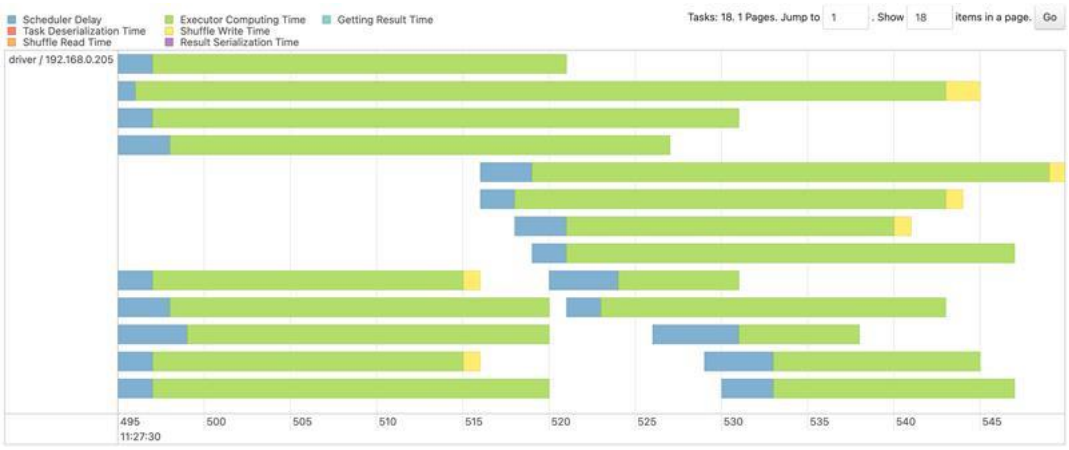

Figure 6 - Screenshot Spark User Weighted persistence model

Number of cores	Number of sensors	Average number of last observations	Batch i (s)	Batch j (s)	Batch k (s)	Average time (s)
10	6	4	0,2	0,2	0,2	0,2
10	18	4	0,5	0,5	0,5	0,5
2	6	4	0,4	0,4	0,4	0,4
2	18	4	1	1	1	1

Table 4 - Experimental results, for the weighted persistence model, of the processing time when varying the numbers of cores and the number of sensors.

4.3. RLS Model

$$\begin{cases} V_{(t)} & = \frac{1}{\nu}\left(V_{(t-1)} - \frac{V_{(t-1)}x_t^T x_t V_{(t-1)}}{1+x_t V_{(t-1)} x_t^T}\right) \\ \alpha_{(t)} & = V_{(t)} x_t^T \\ e & = y_t - x_t \hat{\beta}_{(t-1)} \\ \hat{\beta}_{(t)} & = \hat{\beta}_{(t-1)} + \alpha_{(t)} e \end{cases}$$

Equation 5 - Recursive Least Squares formula

```
15,('CJE181', {'BetaVector': array([[-0.61776854],
       [ 1.37205992]]), 'Variance': array([[ 9.03148176e-06, -1.30397991e-06],
       [-1.30397991e-06,  7.85436777e-07]]), 'ForgettingFactor': 1, 'SSE':
array([[509927.57792001]]), 'Forecast': array([[0.75429137]]), 'N': 145632, 'MSE':
array([[15.13317836]]), 'N_MSE': 33696, 'LastObservations': []})
```

Table 5 - RLS model results with embedding order 1

```
('CJE181', {'BetaVector': array([[  74.31535322],
       [-389.94985329],
       [ 344.42368583]]), 'Variance': array([[ 8.89848920e-06, -7.17904798e-07, -7.17928015e-07],
       [-7.17904798e-07,  1.64381665e-06, -1.19428841e-06],
       [-7.17928015e-07, -1.19428841e-06,  1.64381945e-06]]), 'ForgettingFactor': 1, 'SSE':
array([[3.52501471e+10]]), 'Forecast': array([[418.73903905]]), 'N': 151391, 'MSE':
array([[1046122.5981381]]), 'N_MSE': 33696, 'LastObservations': [('2023-03-31', 96, 1, 4, 'CJE181',
151392)]})
```

Table 6 - RLS model results with embedding order 2

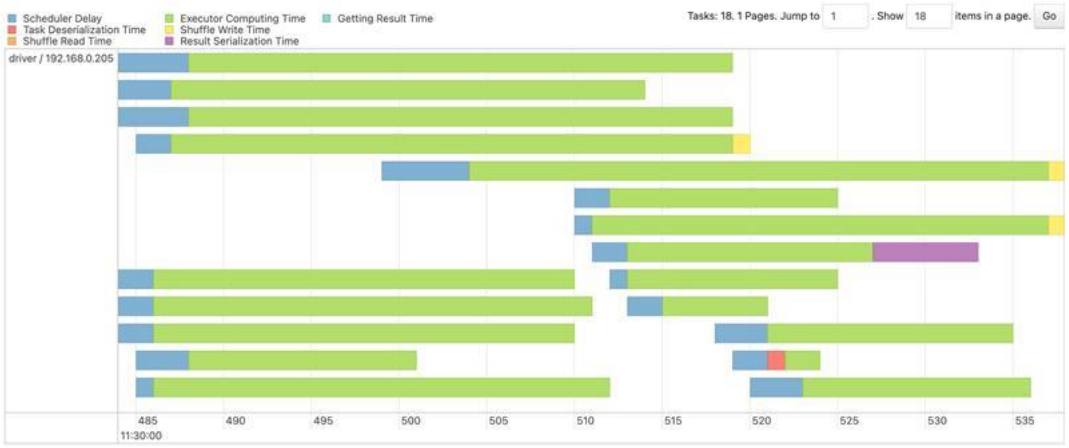

Figure 7 - Screenshot Spark RLS model

Number of cores	Number of sensors	Embedding order	Batch i (s)	Batch j (s)	Batch k (s)	Average time (s)
10	6	1	0,2	0,2	0,2	0,2
10	18	1	0,5	0,5	0,5	0,5
2	6	1	0,4	0,4	0,4	0,4
2	18	1	1	1	1	1

Table 7 - Experimental results, for the RLS model, of the processing time when varying the numbers of cores and the number of sensors.

4.4. Models' performances and RLS Model with added numerical features

	Persistance Models				RLS Models			
Sensor	**FP**	**WP2**	**WP3**	**WP4**	**RLS1**	**RLS2**	**RLS3**	**RLS4**
CAT17	60	61	72	87	68	424.363	400.306	1.329.773
CB02411	114	128	164	208	132	304.317	22.738.030	7.197.933
CB1101	45	58	80	102	48	1.986	60.080	70.368
CB1142	194	289	409	532	220	14.813	71.061.723	85.617.100
CB1143	97	111	144	183	548	220.660	11.061.764	110.633.385
CB1599	22	21	24	27	29	243.271	656.644	3.358.747
CB1699	22	19	20	21	27	210.551	68.376	3.574.750
CB2105	73	64	66	71	92	2.446.383	41.176	258.712.228
CEE016	127	119	119	122	156	2.203.115	698.285	2.255.801
CEK049	72	74	91	110	84	92.000	12.764.034	10.352.336
CEK18	71	100	141	183	83	18.507	11.609.139	488.310
CEK31	81	104	143	186	88	209.421	460.962	728.050
CEV011	21	22	26	31	27	1.359.944	228.471	28.496.671
CJE181	9	8	7	7	15	1.046.123	6.266	10.944.714
CJM90	199	243	319	404	221	31.985	28.657.938	10.012.325
CLW239	20	18	19	21	29	8.085	22.610	101.694
COM205	44	39	41	45	50	1.696.133	934.895	118.965.105
CVT387	21	19	20	22	31	73.348	964.353	694.625

Table 8 - Models' performances comparison with MSE


```
('CJE181', {'BetaVector': array([[-0.32743534],
       [ 1.37423541],
       [-0.02128854],
       [ 0.03520284],
       [-0.01979323]]), 'Variance': array([[ 6.26672402e-05, -9.59370256e-07, -1.58689601e-06,
        -5.05629953e-06, -3.25905476e-06],
       [-9.59370256e-07,  7.86112596e-07, -2.17403773e-08,
         7.44329588e-08, -4.15372244e-08],
       [-1.58689601e-06, -2.17403773e-08,  1.37914841e-07,
        -2.05928566e-09,  1.14820084e-09],
       [-5.05629953e-06,  7.44329588e-08, -2.05928566e-09,
         1.66022299e-06, -3.16189208e-09],
       [-3.25905476e-06, -4.15372244e-08,  1.14820084e-09,
        -3.16189208e-09,  5.22633339e-07]]), 'ForgettingFactor': 1, 'SSE':
array([[1101003.95175069]]), 'Forecast': array([[0.61730683]]), 'N': 151392, 'MSE':
array([[7.27258524]]), 'N_MSE': 151392, 'LastObservations': []})
```


Table 9 - RLS model with added features results with embedding order 1


```
{'CJE181', {'BetaVector': array([[  20.94317107],
       [-390.15888592],
       [ 342.63699974],
       [   4.79639354],
       [  -4.59072212],
       [   2.66977331]]), 'Variance': array([[ 6.66616597e-05, -4.29562273e-07, -6.95345845e-07,
        -1.64152992e-06, -5.26721226e-06, -3.58428754e-06],
       [-4.29562273e-07,  1.65416064e-06, -1.18822540e-06,
        -2.67592528e-08,  4.46684900e-08, -2.56275116e-08],
       [-6.95345845e-07, -1.18822540e-06,  1.64931307e-06,
        -3.65452682e-09,  4.48748356e-08, -2.53591068e-08],
       [-1.64152992e-06, -2.67592528e-08, -3.65452682e-09,
         1.43837906e-07, -2.92723164e-09,  1.67636511e-09],
       [-5.26721226e-06,  4.46684900e-08,  4.48748356e-08,
        -2.92723164e-09,  1.72675297e-06, -3.39569405e-09],
       [-3.58428754e-06, -2.56275116e-08, -2.53591068e-08,
         1.67636511e-09, -3.39569405e-09,  5.73897361e-07]]), 'ForgettingFactor': 1, 'SSE':
array([[5.19784566e+10]]), 'Forecast': array([[468.34004711]]), 'N': 145631, 'MSE':
array([[356921.35247863]]), 'N_MSE': 145631, 'LastObservations': [('2023-03-31', 96, 1, 4, 'CJE181',
151392, 24, 4, 3)]})
```


Table 10 - RLS model with added features results with embedding order 2

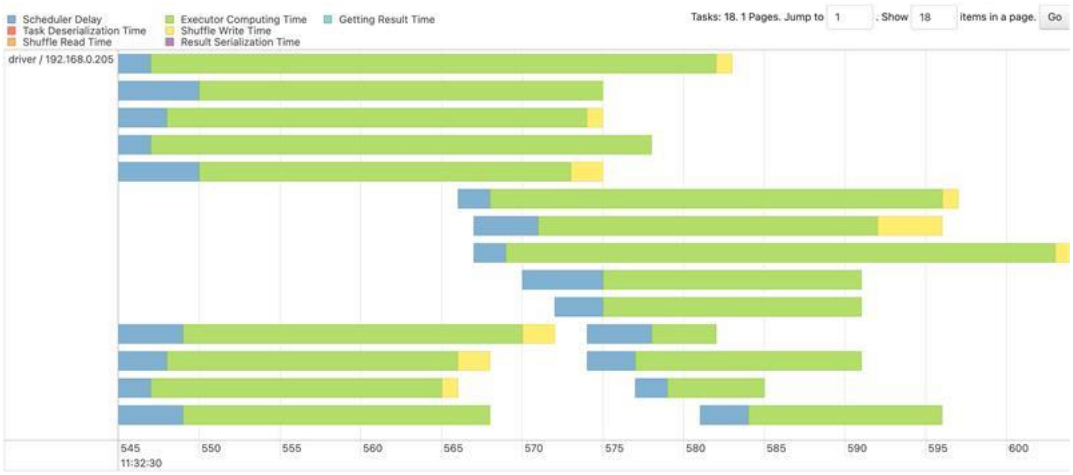

Figure 8 - Screenshot Spark RLS model with numerical features

cores	nb sensors	embedding order	Batch i	Batch j	Batch k
10	6	1	0,2	0,2	0,2
10	18	1	0,5	0,5	0,5
2	6	1	0,4	0,4	0,4
2	18	1	1	1	1

Table 11 - Experimental results, for the RLS model with additional features, of the processing time when varying the number of cores and the number of sensors.

Sensor	FP	WP2	WP3	WP4	RLS1	RLS2	RLS3	RLS4	RLSnum1	RLSnum2	RLSnum3	RLSnum4
CAT17	60	61	72	87	68	424.363	400.306	1.329.773	30	150.274	152.325	450.206
CB02411	114	128	164	208	132	304.317	22.738.030	7.197.933	102	354.966	11.619.600	30.730.958
CB1101	45	58	80	102	48	1.986	60.080	70.368	11	459	13.921	15.653
CB1142	194	289	409	532	220	14.813	71.061.723	85.617.100	109	6.515	24.924.459	1.061.470
CB1143	97	111	144	183	548	220.660	11.061.764	110633385,00	1.063	1.267.239	34.796.881	23.222.060
CB1599	22	21	24	27	29	243.271	656.644	3.358.747	23	123.640	44.864	1.795.912
CB1699	22	19	20	21	27	210.551	68.376	3.574.750	14	78.558	31.762	1.265.832
CB2105	73	64	66	71	92	2.446.383	41.176	258.712.228	53	138.643	55.971	31.082.030
CEE016	127	119	119	122	156	2.203.115	698.285	2.255.801	55	645.454	266.643	678.635
CEK049	72	74	91	110	84	92.000	12.764.034	10.352.336	67	133.649	7.268.666	1.393.247
CEK18	71	100	141	183	83	18.507	11.609.139	488.310	291	4.149	4.629.799	3.652.789
CEK31	81	104	143	186	88	209.421	460.962	728.050	33	74.738	145.329	219.239
CEV011	21	22	26	31	27	1.359.944	228.471	28.496.671	13	517.843	76.185	9.557.822
CJE181	9	8	7	7	15	1.046.123	6.266	10.944.714	7	356.921	3.259	3.615.884
CJM90	199	243	319	404	221	31.985	28.657.938	10.012.325	147	390.191	19.755.790	1.213.265
CLW239	20	18	19	21	29	8.085	22.610	101.694	23	3.985	10.209	62.340

COM205	44	39	41	45	50	1.696.133	934.895	118.965.105	31	3.947.945	503.237	11.820.308
CVT387	21	19	20	22	31	73.348	964.353	694.625	13	22.603	34.970	198.661

Table 12 - MSE computed by each model for each sensor.