Project 2

Final Report

Big Data and Cloud Computing

Group:
Afonso Pinto - up201503316
Edgar Carneiro - up201503784

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

June 30, 2019

1 Summary

In this project, we made use of Apache Beam and Tensorflow to process the EVENTS dataset. We wrote a number of Python classes that manipulate this dataset and are capable of outputting graphs that can provide human knowledge regarding the dataset, as well as a method of predicting the Length of Stay for patients.

2 Technologies & Workflow

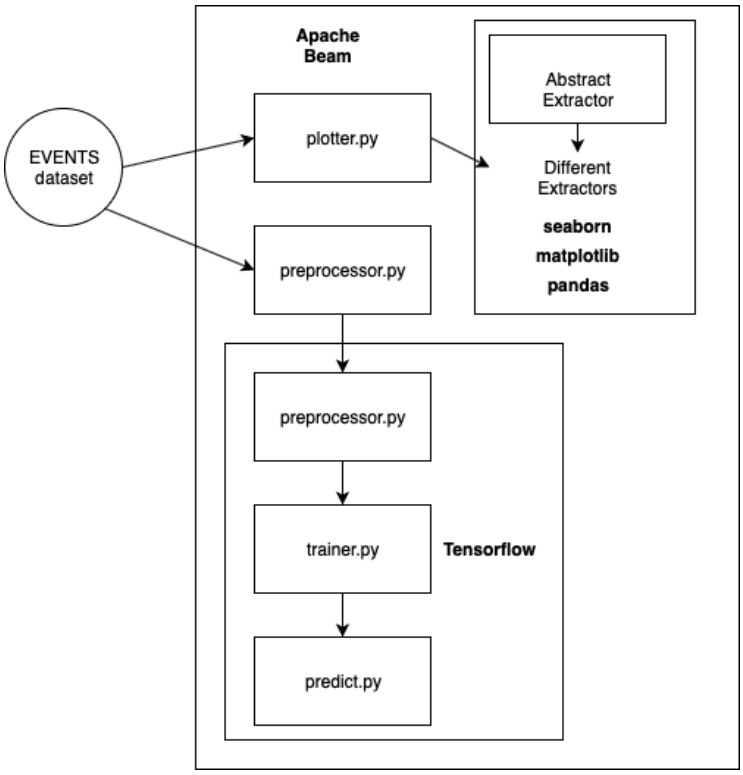

Figure 1: Technologies & Workflow Diagram

3 Development & Results

The project development was focused in two major modules, developed in parallel: the module responsible for performing a statistical analysis on the data and producing **timeline graphs for each patient**, while the second module focused on **predicting the length of stay** - *LoS* - for patients.

3.1 Patient Graphs

In this subsection, we will analyse thoroughly each one of the outputted patient graphs. It is important to bear in mind that the assigned task was to perform a statistical analysis and produce timeline graphs for each patient,

hence, the developed plots focus on providing information that can be of value to patients rather than information that could be interesting to have but was of no use to a specific patient (*e.g.* the type of cases present in the *ICU*).

3.1.1 Values per Time

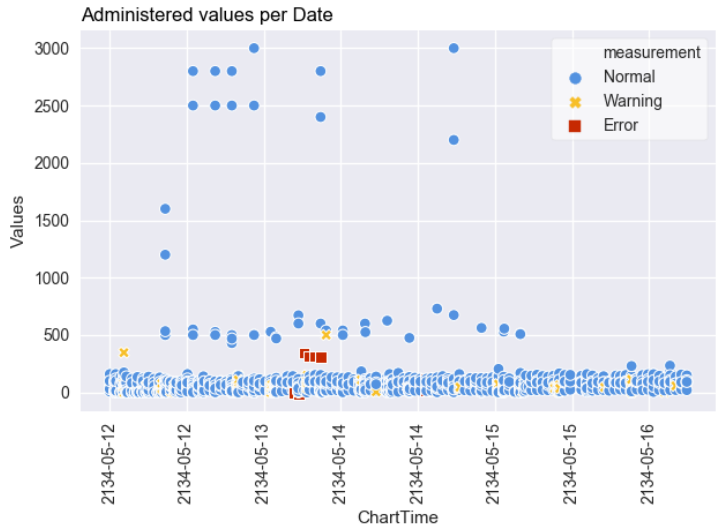

Figure 2: Values per Time Scatterplot

In the **Values per Time** plot, one can visualise all the values that were ever administered to a patient as well as the date they were administered. Notice that all values are represented in the respective Units of Measure. The plot also provides information regarding the measurement status: if it occurred normally or with an error, of if it raised a warning. Figure 2 provides an example of this plot for the patient with *SUBJECT_ID 36*.

3.1.2 Items Histogram

In the **Items Histogram** plot, one can visualise all the Items that were administered to were ever administered to a given patient, as well as a rough approximation of the administration frequency of that item. This plot provides insight regarding the most used Items by the patient and, consequently, might provide valuable information about which conditions affect the patient. Figure 3 provides an example of this plot for the patient with *SUBJECT_ID 36*.

3.1.3 Item Hours Intake

In the **Item Hours Intake** plot, one can gain insight regarding the most frequent hours at which the patient was administered an item. As we can see in the example given, Figure 4, the patient with *SUBJECT_ID 36* is often administered at *4am, 8am, 12am* and *8pm*. The Patient can also observe which items it have been administered with more frequency and the frequency of the administration.

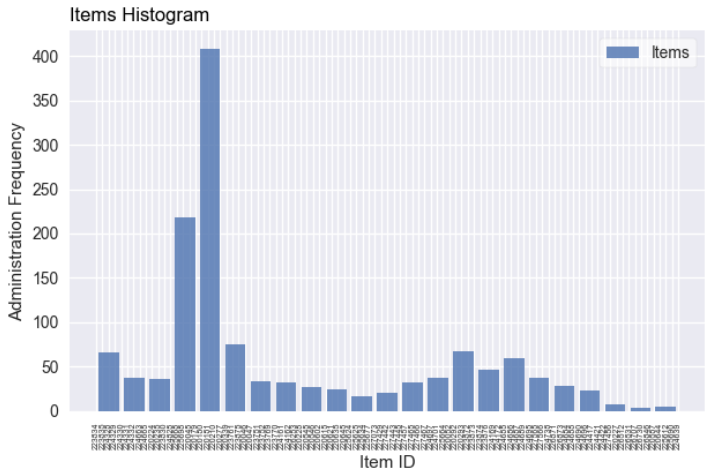

Figure 3: Items Histogram

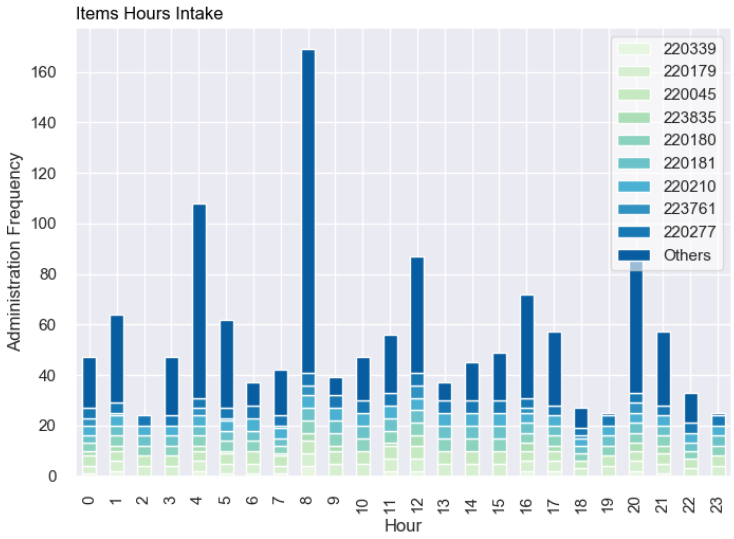

Figure 4: Stacked Daily Items Histogram

3.1.4 Length of Stay

In the **Length of Stay** plot, one can observe an Histogram that provides information regarding the patient length of stay, for every single time the patient has been accepted in the hospital with a different **Hospital Admission ID**. With the presence of rugs in the plot, the patient can also know the exact length of its hospital admission. Figure 5 provides an example of this plot for the patient with *SUBJECT_ID 188*.

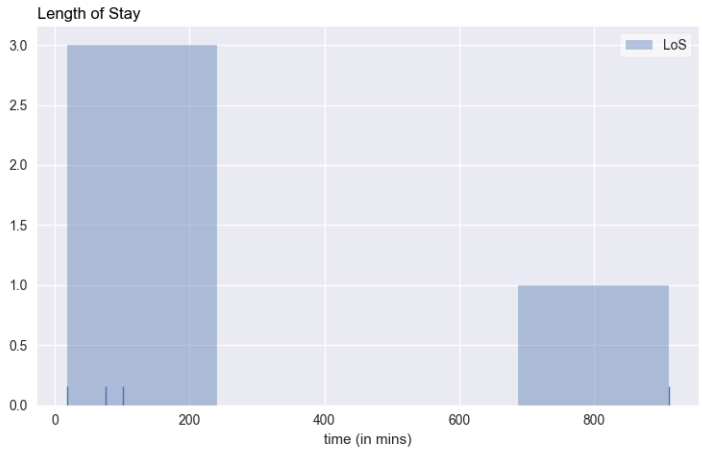

Figure 5: Length of Stay Histogram

3.2 Care Giver - Item - Quantity Relation

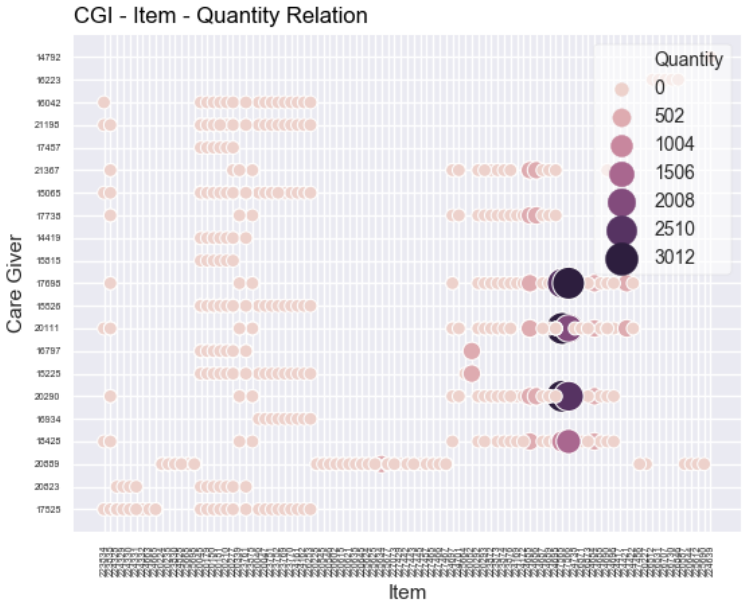

Figure 6: Care Giver - Item - Quantity Bubbleplot

In the **Care Giver - Item - Quantity Relation** plot, one can establish a relation between the care giver, the item it administers and the quantity it administers. This plot is useful for the patient since it allows him to know which Care Givers administer specific items, which Care Givers administer high quantities, which may be useful for the patient to infer and understand its medical situation. Figure 5 provides an example of this plot for the patient

with *SUBJECT_ID 36*.

3.3 Predict Length of Stay

In this subsection we will analyse how the team has approached the problem of predicting the **length of stay (LoS)** for patients, using a machine learning method. We both agreed that the length of stay should be computed as the maximum date an item had been administered minus the minimum date an item had been administered, given a certain Hospital Admission ID.

We opted to approach the problem has a **regression** since the LoS is a possible number from a continuous scale.

3.3.1 Data Extractor

This step of the beam Pipeline is used to compute metrics that will be necessary in the *preprocessing step*, such as the total number of distinct Items in the dataset.

Notice that we can not join this step with the preprocessing step because of the way beam works: you can only input and output from files to files, and not to vairables.

3.3.2 Preprocessing

This is an Apache Beam pipeline that will do all the preprocessing necessary to train a Machine Learning model. It uses tf.Transform, which is part of TensorFlow Extended, to do any processing that requires a full pass over the dataset.

We're doing the following manipulations in order to extract valuable features from the dataset.

1. Use Apache Beam to parse the CSV file. the data read to contain only the HADM_ID entries in a given time window (24hours by default).
2. Calculate the mean of each distinct item in the filtered dataset.
3. Calculate the Length of Stay as explained in 3.3.
4. Reduce all the information aforementioned into the desired schema: (HADM_ID, Item0, Item1, ..., CGID (Boolean value representing the existence or not of a caregiver, Length of Stay)

In the next step we will train a Neural Network to do the predictions. Neural Networks are more stable when dealing with small values, so we decided to normalize the inputs to a small range (from 0 to 1). Since there's no maximum number that an item value can have, we have to go through the entire dataset to find the minimum and maximum counts, to do so we take advantage of tf.Transform integrated with our Apache Beam pipeline. In order to fix the possibility of missing values, we opted in for a statistical approach where we replace them with the mean we previously calculated. This is an approximation which can add variance to the dataset, but yields better results compared to removal of rows and columns.

After preprocessing our dataset, we also want to split it into a training and evaluation dataset. The training dataset will be used to train the model. The evaluation dataset contains elements that the training has never seen, and since we also know the "answers" (the molecular energy), we'll use these to validate that the training accuracy roughly matches the accuracy on unseen elements.

3.3.3 Training & Testing

We'll train a Deep Neural Network Regressor in TensorFlow. This will use the preprocessed data stored within the working directory in the previous step. The TensorFlow model takes the unnormalized inputs, applies the tf.Transform's graph of operations to normalize the data, and then feeds that into our DNN regressor.

To visualize the training job, we can use TensorBoard.

```
tensorboard --logdir \$WORK\_DIR/model
```

Here are the measures of the metrics we are tracking: **Root Mean Squared Error:** 158.8, **Mean Absolute Error:** 87.12, **Mean Cosine Distance:** -9.1479 e+5.

Here are some of the graphs you can find there evaluating:

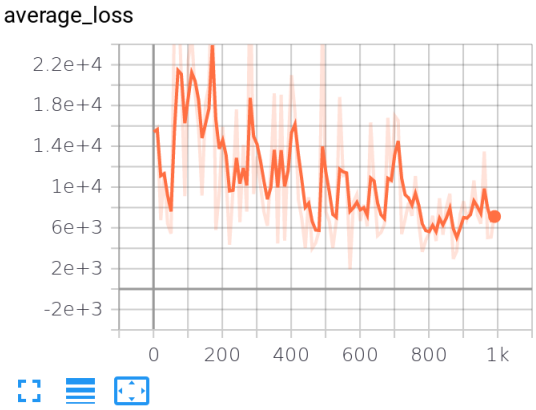

Figure 7: Average loss graph

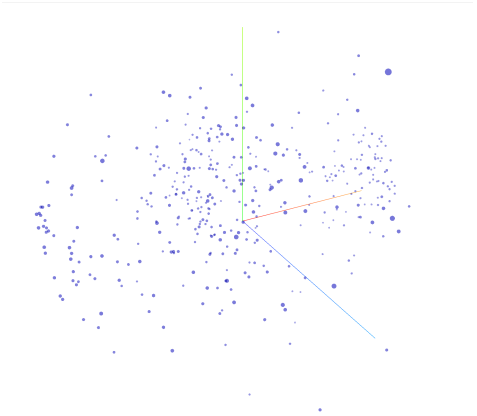

Figure 8: Eval Projector

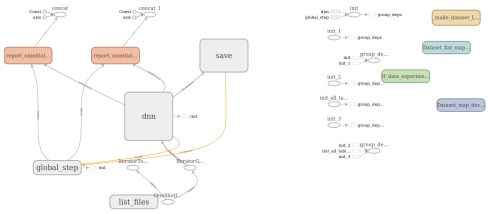

Figure 9: Execution Graphs

3.3.4 Predictions

We implemented batch predictions since these work best if there's a large amount of predictions to make.

You can access them in $WORK_DIR/predictions.

```
==> ./tmp/predictions/part-00000-of-00001 <==
{"predictions": [70.76386260986328], "value": 20.366666666666667}
{"predictions": [26.397756576538086], "value": 17.85}
{"predictions": [81.29413604736328], "value": 110.91666666666667}
{"predictions": [45.158870697021484], "value": 18.583333333333332}
{"predictions": [24.575925827026367], "value": 16.4}
{"predictions": [39.61642837524414], "value": 24.933333333333334}
{"predictions": [148.3115997314453], "value": 80.15}
{"predictions": [28.36361312866211], "value": 103.2}
{"predictions": [41.44767379760742], "value": 17.35}
{"predictions": [27.87032127380371], "value": 71.68333333333334}
```

Figure 10: Example of predictions

3.4 Other relevant notes

- Several warnings are raised when running the program, because of the usage of python2.
- Check how to run all the scripts using: *python2 src/plotter.py -h*, *python2 src/data_extractor.py -h*, *python2 src/preprocess.py -h*, *python2 src/train.py -h* and *python2 src/test.py -h*.
- Example of scripts usage:

4 Difficulties

- Besides the documentation, there is very few support for apache beam in python.
- Some operations are very difficult to execute in apache beam when compared to, for example, the map-reduce paradigm in pyspark. Apache Beam does not have natural support to store the result of PTransforms over PCollections in dynamic variables, forcing us to store the data in files and dividing the execution pipeline in several steps, where the next step reads data from files created in the previous step.

```
# Plottings for patient 36
python2 src/plotter.py -i EVENTS_1000000.csv -p 36

# LoS Prediction
python2 src/data_extractor.py -i EVENTS_100.csv
python2 src/preprocess.py -i EVENTS_100.csv -w tmp
python2 src/trainer.py -w tmp
python2 src/predict.py -w tmp -m
    tmp/model/export/final/1561600285 batch
     -i EVENTS_100.csv -o tmp/predictions

# LoS Prediciton in a shell script
./run-local.sh --input-file EVENTS_1000000.csv
```

- Apache beam can not infer the schema.

5 Conclusions

The realization of this project provided the group with extensive learning. We were able to acquire a greater knowledge in the area of Big Data and Machine Learning, more specifically using the Apache Beam tool for handling data, Tensorflow for Machine Learning and Seaborn and Matplotlib for data visualization.

Regarding cross validation, we started implementation, even though doing it on apache beam proved to be much harder than expected. We successfully implemented the partitioning of the dataset into five folds, and proceeded to training those folds. However, we were not able to programmatically gather the output metrics of the resultant models to posteriorly present the average values for the evaluated metrics.