CERN Summer Student Work Project Report

Transfer metrics analytics project

Student: Žygimantas Matonis (Vilnius University)
Supervised by: Nicolò Magini, Valentin Kuznetsov, Daniele Bonacorsi

***Abstract* — This report represents work done towards predicting transfer rates/latencies on Worldwide LHC Computing Grid (WLCG) sites using Machine Learning techniques. Topic covered are technologies used for the project, data preparation for ML suitable format and attribute selection as well as a comparison of different ML algorithms.**

I. INTRODUCTION

LHC experiments transfer more than 10 PB/week between all grid sites using the FTS transfer service. In particular, CMS manages almost 5 PB/week of FTS transfers with PhEDEx. FTS sends metrics about each transfer (e.g. transfer rate, duration) to a central HDFS storage at CERN. A proposal was made to use ML techniques to process this raw data and generate predictions of transfer rates/latencies on all links between Grid sites.

The task for me as a student was to prepare data for ML suitable format, converting *JSON* format ASCII files to a flat table format. Secondly, data fields not suitable for predictions should be identified and dropped. Finally, different ML algorithms should be tested and compared.

II. DEVELOPMENT TOOLS

During the project I had to get familiar with the following tools and technologies:

- *Python* as programming language for the project;
- *Scikit-learn* - a ML library for *Python*;
- *XGBoost* - another ML library with binding for python and *Scikit-learn*;
- Additional libraries such as *Pandas*, *NumPy* and etc. for data manipulation;
- *Matplotlib* library for drawing charts;
- *DCAFPilot* - Data and Computing Analytics Framework developed and maintained by V. Kuznetsov with the help of his students.

For a good part of the project, I was using *DCAFPilot* framework to analyze data since it wraps few of mentioned libraries together. However, for the more specific task I wrote my own scripts.

III. WORK DONE

For the first part of the project, the data had to be transformed to a ML suitable format. Metrics about each transfer are stored as JSON objects in ASCII files so input files had to be transformed to CSV file. However, few things had to be taken into account:

- Input files loosely match JSON document structure, with custom EOT character ending each record;
- Each record had nested data and had to be flattened;

- ML algorithms can not process text so text values have to be converted to numerical values using hashing algorithms;
- A placeholder value should be used if attribute or value is missing;
- Input files are large, so the script must be able to process gigabytes of data.

A script was created taking everything into account. It was also shared with other teams having similar issues.

For the second part of the project, I had to identify attributes that can not be used with ML and should be dropped. There can be a few reasons why an attribute can not be used. Firstly, transfer records contained data that in ML terms would be considered output such as if a transfer error occurred, the end time of each transfer phase, etc. Since these attributes are not available at the beginning of the transfer, they can not be used for predictions. Secondly, values can be static, meaning they do not change through all dataset. Such values are worthless and only takes spaces or even mislead algorithms. Finally, values can correlate with each other meaning that if one value changes other value will change proportionally. By doing so initial dataset can be decreased more than half. Figure 1 shows one of the correlation matrices.

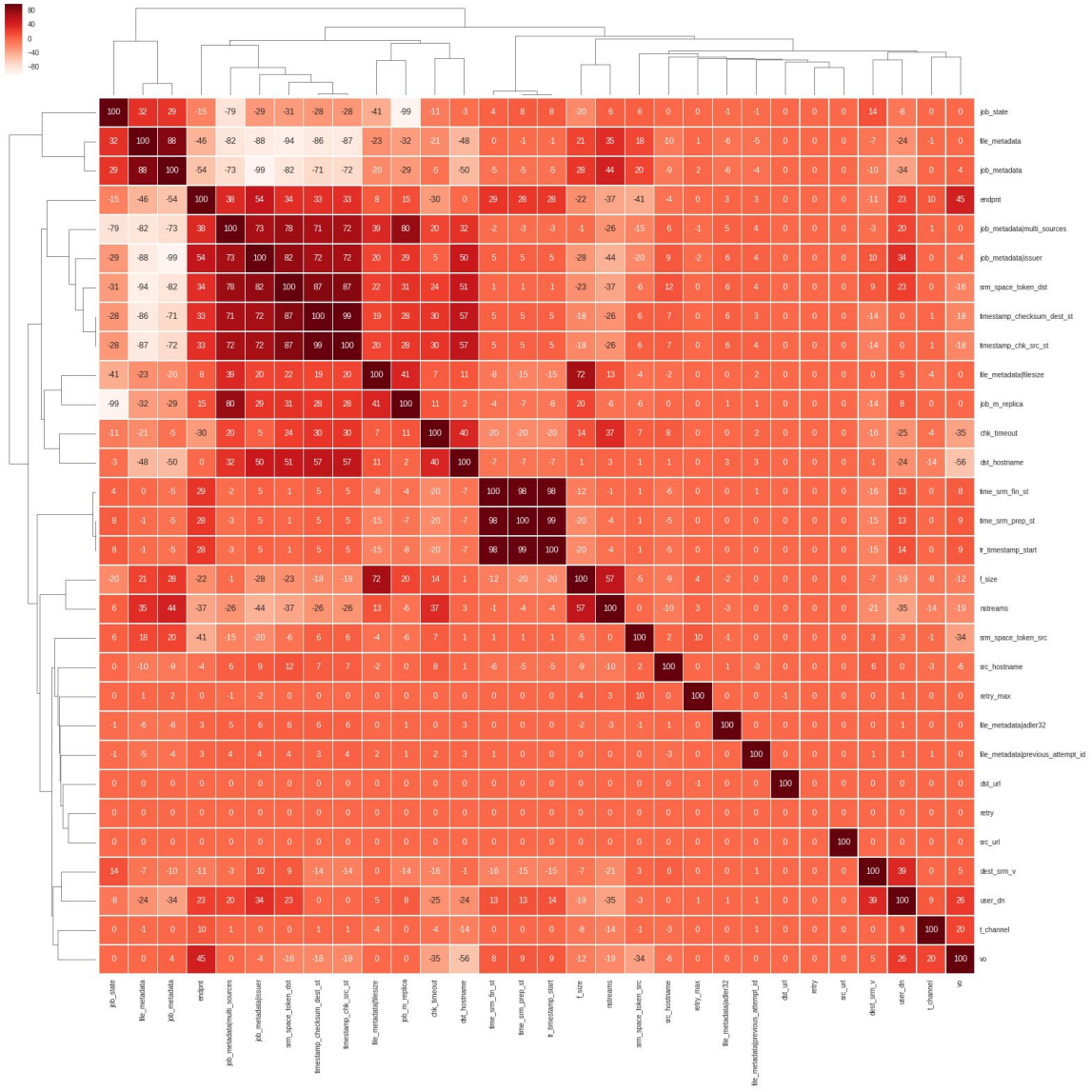

Figure 1. Correlation matrix.

For the last part of the project, a comparison between different ML algorithms and parameters was needed. Algorithms were compared with each other depending how much time and memory it takes to train

models and by also comparing error rate on test data using MAE(Mean Absolute Error) metric. Quite a few algorithms were tested, however, for future benchmarking these were selected:

- *RandomForestRegressor* from *Scikit-learn* library;
- *GBRegressor* from *Scikit-learn* library*;*
- *XGBRegressor* from *XGBoost* library*;*

All of the tests were performed on the same dataset. Figure 2 shows how *XGBRegressor* algorithm's performance depends on different parameters. As you can see increasing "learning_rate" and "subsample" parameters increases the time needed to train model but almost has no impact on memory. However, it does improve algorithm's accuracy ("4.MAE" column, the lower value the better).

1	1.parameters	2.time_to_train_in_sec	3.RAM_usages_in_MiB	4.MAE
2	{'n_estimators': 200, 'subsample': 0.9, 'learning_rate': 0.1, 'objective': 'reg:logistic'}	12.2035648823	207.32421875	32749.4339853081
3	{'n_estimators': 200, 'subsample': 0.85, 'learning_rate': 0.1, 'objective': 'reg:logistic'}	12.324919939	207.27734375	31568.7834644602
4	{'n_estimators': 200, 'subsample': 1, 'learning_rate': 0.15, 'objective': 'reg:logistic'}	19.8856809139	211.41796875	26778.7836500032
5	{'n_estimators': 200, 'subsample': 0.95, 'learning_rate': 0.15, 'objective': 'reg:logistic'}	13.6029081345	207.40234375	26200.4541423126
6	{'n_estimators': 200, 'subsample': 0.85, 'learning_rate': 0.15, 'objective': 'reg:logistic'}	12.2356331348	207.31640625	25925.3380543625
7	{'n_estimators': 200, 'subsample': 0.9, 'learning_rate': 0.15, 'objective': 'reg:logistic'}	15.5717151165	207.3359375	25378.5118438082
8	{'n_estimators': 200, 'subsample': 0.95, 'learning_rate': 0.2, 'objective': 'reg:logistic'}	13.7010810375	211.41015625	24506.4315143314
9	{'n_estimators': 200, 'subsample': 1, 'learning_rate': 0.2, 'objective': 'reg:logistic'}	20.7377018929	211.421875	23899.9587562132

Figure 2. *XGBRegressor* algorithm result with different parameters.

Figure 3 and 4 shows how *RandomForestRegressor* and *GBRegressor* algorithm performs with same parameters on the same dataset. As you can see from the figures, *RandomForestRegressor* has a better prediction rate overall, however, increasing "n_estimators" parameter's value drastically increases time and memory consumption while giving modest improvement in prediction rate. On the other hand, though *GBRegressor* algorithm has worse prediction rate, it is faster and does benefit from increased "n_estimators" parameter's value.

1	1.parameters	2.time_to_train_in_sec	3.RAM_usages_in_MiB	4.MAE
2	{'n_estimators': 50}	55.0980620384	399.4140625	24444.334563692
3	{'n_estimators': 100}	111.4150829315	653.2265625	24428.480657335
4	{'n_estimators': 200}	224.486289978	1146.84765625	24259.972300372

Figure 3. *RandomForestRegressor* algorithm result with different parameters.

1	1.parameters	2.time_to_train_in_sec	3.RAM_usages_in_MiB	4.MAE
2	{'n_estimators': 50}	15.6836359501	178.3828125	45884.7167851247
3	{'n_estimators': 100}	31.2205569744	178.41796875	41668.2656253256
4	{'n_estimators': 200}	63.3703970909	178.54296875	39059.5073047259

Figure 4. *GBRegressor* algorithm result with different parameters.

It is not a trivial task to select optimal parameters for the ML algorithm or the right ML algorithm for the task. After comparing results in figure 2 and 3 it can be seen that even though RandomForestRegressor has better MAE score then XGBRegressor with similar parameters in the beginning, after carefully adjusting parameters for XGBRegressor it is eventually outperformed. However, optimal parameters can change

depending on the dataset. For that reason, I created a script to automatically run ML algorithms with different parameters so that results could be compared and the right parameters with the right algorithm would be chosen.

IV. CONCLUSIONS

Even though the project is still not over, I feel that I contributed as much as I could having the time available for me. In return, I tried or deepened my knowledge in different technologies such as *Machine Learning* and *Python*. I also had unforgettable experience being a Summer Student which I believe will be beneficial in my future career.