Analysing CMS transfers using Machine Learning techniques

Tommaso Diotalevi[1]*
Supervised by: Daniele Bonacorsi, Nicolò Magini, Valentin Kuznetsov

[1]University of Bologna, Dipartimento di Fisica e Astronomia, Bologna, Italy

*tommaso.diotalevi@studio.unibo.it

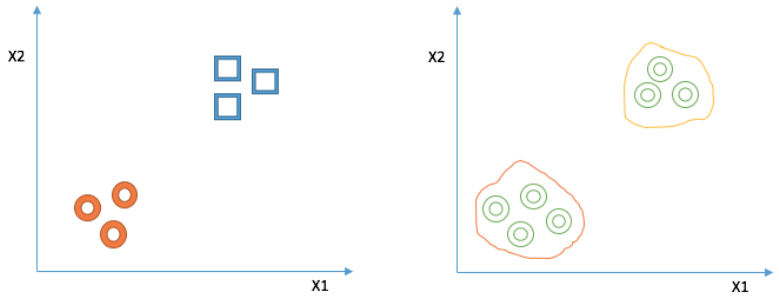
ABSTRACT

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 (**Ph**ysics **E**xperiment **D**ata **Ex**port).
FTS sends metrics about each transfer (e.g. transfer rate, duration, size) to a central HDFS storage at CERN. The work done during these three months, here as a Summer Student, involved the usage of ML techniques, using a CMS framework called DCAFPilot, to process this new data and generate predictions of transfer latencies on all links between Grid sites.
This analysis will provide, as a future service, the necessary information in order to proactively identify and maybe fix latency issued transfer over the WLCG.

Introduction to Machine Learning

Machine Learning (ML) basically is a type of artificial intelligence that provides computers with the ability to learn without being explicitly programmed. It consists in a series of algorithms that can learn from and make predictions on data.
There are two main typologies of ML algorithms:

- **Supervised**, in which a target parameter of our data is given to the algorithm in order to make predictions i.e. the algorithm already knows the possible outcomes for the specific variable (Fig.1a) ;
- **Unsupervised**, in which there is no target parameter and the algorithm has to learn by itself the possible outcomes (Fig.1b).



(a) Schematic representation of a supervised learning algorithm.

(b) Schematic representation of an unsupervised learning algorithm.

Figure 1. Supervised and Unsupervised machine learning techniques.

The work presented on this report focuses on a supervised machine learning technique over a dataset of the GRID Data Transfer Service at CERN called *File Transfer Service* (FTS).

Supervised ML in detail

The supervised learning mainly consists in inferring a particular function starting from *labelled* **training data**. The algorithm analyzes this training data x and produces a prediction function $f(x)$ which can be used for mapping new datasets.

The ML supervised optimization problem consists in minimizing the particular $f(x)$, which can be easily written in a linear form

$$f(x) = \lambda_0 + \lambda_1 * x$$

where training data x are used to optimize $f(x)$. λ_0 and λ_1 are free parameters that the algorithm continuously changes in order to make the prediction more truthful at each step. It is very important to understand that a successful Machine Learning effort is not reaching perfection but making the prediction 'good enough' for the problem in exam.
Since Machine Learning algorithms are mainly based on statistics, *training data* must be statistically significant random samples, otherwise an **overtraining** problem may occur, meaning that the machine learns fake patterns that are mainly caused by correlation between variables.
Another important distinction of the supervised Machine Learning is between *regression* and *classification* systems:

- In a *regression* system the predicted value covers an entire continuous spectrum;
- in a *classification* system the predicted value is discrete, 1 or 0 ('yes' or 'no').

Since my work is focused on a classification problem, the following section will explain this category in more detail.

Classification scorers

How do I verify if an algorithm is 'correct'? Does it predict well my data sample?
In order to answer these questions, I need to define some variables that express the goodness of my learning algorithm. The first step is to check if the prediction is correct compared to the real value of the target (in a supervised algorithm)

- True Positive (**TP**): true predictions (1) which are really true;
- True Negative (**TN**): false predictions (0) which are really false;
- False Positive (**FP**): true predictions that are actually false;
- False Negative (**FN**): false predictions that are actually true.

Putting these variables, which basically are integer counts, in a 2x2 matrix, we obtain a so called *confusion matrix*, shown in Figure 2. The total number of the four entries $TP+TN+FP+FN = n$ represent the total number of entries in the training sample.
In order to evaluate the best classifier algorithm, there are some scorers which help us choosing the best one. The most important ones are: *Accuracy*, *Precision*, *Recall*, *F1*, computed as

$$\textbf{Accuracy} = \frac{TP+TN}{n} \tag{1}$$

$$\textbf{Precision} = \frac{TP}{TP+FP} \tag{2}$$

$$\textbf{Recall} = \textbf{TPR} = \frac{TP}{TP+FN} \tag{3}$$

$$\textbf{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2TP}{(2TP+FP+FN)} \tag{4}$$

Accuracy, which represents the number of correct predictions over the entire data sample, is not a good metric for a particular scorer: in fact, when a data sample is not distributed equally between different classifications, it may yield to misleading results. *Precision* (or *Positive predictive value*) represents the rate of predicted positives compared to all positive outcomes. *Recall* or *True Positive Rate*(TPR) represents the ratio between well predicted positives and the total amount of positive entries (both correctly predicted and not). *F1* represents the harmonic mean of precision and recall. Other important scorers used for the evaluation of a Machine Learning algorithm are:

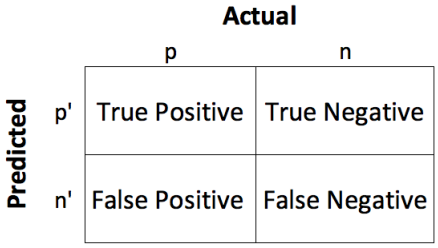


Figure 2. Confusion Matrix.

- The **FPR** (or *False Positive Rate*) defined as

$$\mathbf{FPR} = \frac{FP}{FP+TN} \tag{5}$$

  which is the ratio between incorrect positive predictions and the total amount of negative entries (both correctly predicted and not);

- The **TNR** (or *True Negative Rate*) defined as

$$\mathbf{TNR} = \frac{TN}{TN+FP} \tag{6}$$

  which is the ratio between correct negative predictions and the total amount of negative entries (both correctly predicted and not);

- The **FNR** (or *False Negative Rate*)

$$\mathbf{FNR} = \frac{FN}{FN+TP} \tag{7}$$

  which is the ratio between incorrect negative predictions and the total amount of positive entries (both correctly predicted and not).

Analysing transfers for the CMS experiment using DCAFPilot

In order to perform models using Machine Learning techniques, I used a framework developed for the CMS experiment by Valentin Kuznetsov called DCAFPilot[1] (**D**ata and **C**omputing **A**nalysis **F**ramework **P**ilot). This framework basically contains the metrics and the necessary tools needed for a Machine Learning problem. DCAFPilot was originally designed to predict which dataset will become popular[2] even before being available on the Grid.
As stated above, my project involves studying transfers latencies and not popular datasets, therefore I had to adapt DCAFPilot in order to work properly with a new data format. A schematic representation of the DCAFPilot workflow is shown in Figure 3.
In the next sections I'll talk in more detail about my personal project here at CERN.

Converting .json dataset into .csv

First of all, FTS raw data used for analysis are collected on a HDFS (**Had**oop **F**ile **S**ystem) cluster in a .json format which needs to be converted in a .csv format in order to be correctly read and processed by DCAFPilot.
This operation required the creation of a python script written by Žygimantas Matonis, another Summer Student working on a similar project[3]. This script basically works as follows:

- Takes as input the .json file that has to be converted;
- Since the .json format contains nested records, they need to be flattened into one big record;
- Each record may contain strings that cannot be read by DCAFPilot, therefore text values need to be converted to numerical ones using hashing algorithms;
- Put placeholder if an attribute is missing or blank;
- Fill the .csv output file.

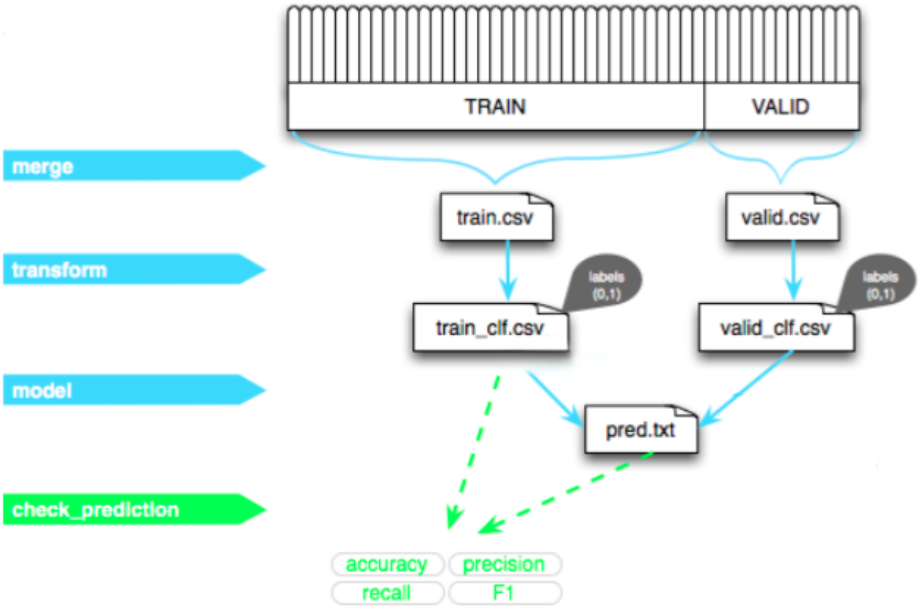


Figure 3. Graphical representation of the DCAFPilot workflow.

Choosing training and validation set

The HDFS cluster contains months of FTS logs and Terabytes of data. Having at my disposal only a limited storage (a few GB on my Virtual Machine here at CERN), I had to take only a few days of transfer in order to create my training and validation set. After a careful selection, I decided to pick two days of July 2016 (1^{st} and 2^{nd}) as training set and the 3^{rd} day as validation set, with a split in terms of total entries of 68/32%.

Merging datasets

After the selection of training and validation sets, I had to merge multiple days into one file for each of them and modify the structure of the .csv files in order to prepare them for analysis.
This operation (labeled with 'merge' in Figure 3) is done by DCAFPilot, using a tool called *merge_csv.py* executable by command line:

Code 1. merge_csv usage

```
$ merge_csv --fin <input_files> --fout <merged_output>
```

As already stated above the code of DCAFPilot is optimized for popularity problems and not transfer analysis: in this case I made some minor changes to the script in order to take as input also FTS logs[4].
The result of this step consist in two big files called *train_merge.csv* and *valid_merge.csv*.

Transformation for Machine Learning analysis

The next step of DCAFPilot workflow (labeled with 'transform' in Figure 3) is called transform_csv. This operation is done by a tool called *transform_csv.py* also executable by command line:

Code 2. transform_csv usage

```
$ transform_csv --fin <input_file> --fout <transformed_output>
--target <target_variable> --target-thr= <threshold_number>
--drop <list of variables to drop> --log-all --log-bias <bias_value>
```

Starting from the merged file (obtained from the previous step) as –fin input, the script adds a column (target) containing only 0 and 1 values, according to the target value chosen in –target: if it's below the desired threshold (inserted in –target-thr) it's a 0 otherwise it's 1.

The .csv file then passes through a logarithmic transformation (enabled using the –log-all option) in order to shrink all integer values into smaller values. The –log-bias option prevent values from becoming negative by adding a constant to each value. Another important transformation applied by this tool to .csv data regards the –drop option. As previously stated, Machine Learning algorithms can overtrain: this happens when some correlations exist between variables that may mislead the final prediction. Therefore, these variables have to be removed from the data file (by inserting them into the –drop option).
In Figure 4, the correlation matrix (using R programming language[5]) of the entire dataset is shown. According to their correlation factor r I removed, among all variables, the ones with $r > |0.6|$.

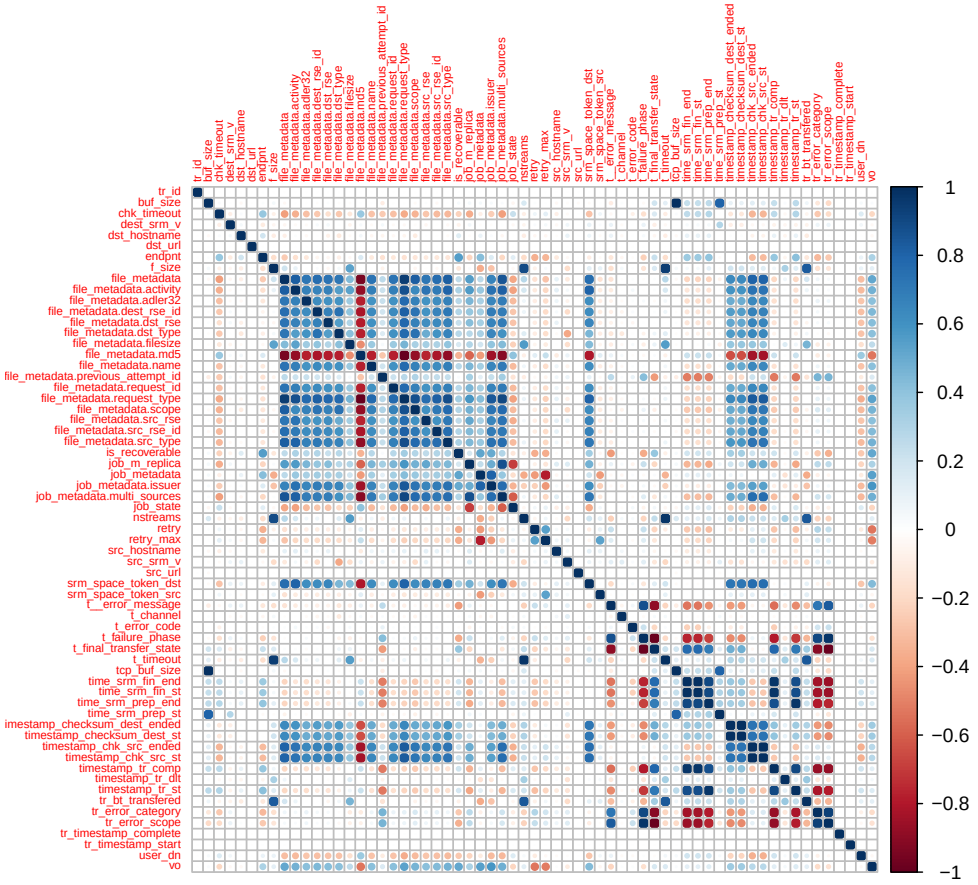


Figure 4. Correlation matrix of every FTS log variable.

Creating ML Model

This is the most important step because it creates the model starting from the transformed .csv file, giving as output a prediction file computed using a given algorithm, called *classifier* in a classification problem. This step is labeled with 'model' in Figure 3. The DCAFPilot tool used for this operation is called *model.py* and, consistently with the other tools, can be executed from the terminal:

Code 3. model usage

```
$ model --learner <classifier_name> --idcol <id>
--target <target> --trainfile= <training_input_file> --scaler <scaler_name>
--newdata <validation or new data input file>
--predict <prediction output file name>
```

After choosing a classifier (passed as –learner argument), this tool takes as input two different files: the training .csv and the validation .csv files. The –target argument is taken from the previous step (*transform_csv*) and the –idcol refers to an

identification variable created during the data preparation.
As previously explained, the training set is used to build our model while the validation set is used to tune our model parameters. The option –newdata in particular can accept two types of data: validation set or entirely new data. The validation set undergoes the same process stated above for the training set, creating therefore target variables with labels that our model has to correctly predict; while newdata does not contain labels and therefore can be used for testing the algorithm later on.

Check Prediction

The last step (labeled with 'check_prediction' in Figure 3) is called check_prediction. This operation basically compares the prediction file produced during the previous step with the labels of the validation set. The results of this comparison are the scorers defined above for the quality of the Learning algorithm e.g. TP, TN, FP, FN, accuracy, precision, recall and F1. The DCAFPilot tool is called *check_prediction.py*, executable by command line:

Code 4. check_prediction usage

```
$ check_prediction --fin <validation_file> --fpred <prediction_file> --scorer
<scorer_name>
```

The prediction file is the result of the previous step while the final output is a series of numeric values (chosen using the –scorer option) displayed on screen.

Results

My project here at CERN consisted in training different models, using as input FTS logs (from an HDFS cluster here at CERN) and considering different classifiers in order to compare performances. For more information and all the code written during this stay, there is a GitHub repository[4] especially created for this purpose.

The first choice to make is the target variable: the FTS logs contain a lot of parameters regarding transfers e.g. file size, time of completion, time of start, number of replicas etc...
The most intuitive variable to analyze (which is the one I chose for this project) regarding transfer latencies is the delta transfer time defined as the difference between the transfer completion and the transfer start; this includes also the transfer overhead that is the excess time before and after the transfer due to initial preparations and checksums.
Figure 5 shows the histogram of this variable on the entire dataset (876372 valid entries for two days of transfer).
Dropping only variables with correlation factor $r > |0.6|$, Figures 6,7,8,9 show, for each different scorer, the trend of each classifier (shown in the x-axis) varying the threshold of delta transfer time, from 30 seconds to 8 minutes. In particular Figure 9 shows the FNR (False Negative Rate defined in Eq 7). This scorer is very important for transfer latencies: it measures in fact the rate of False Negatives, which are transfers affected by latency issues that the algorithm has misidentified as normal, normalized by the number of positive entries contained in the dataset.
From Figure 10, in which I plotted Accuracy, Precision, Recall and F1 (as defined from Eqn 1,2,3,4) with a threshold of 2 minutes, some classifiers have very low values compared to others and therefore had to be discarded e.g. *GaussianNB*, *BernoulliNB*, *LinearSVC* and *RidgeClassifier*. The remaining classifiers, from Figure 9, show that a classification of transfers slower than a given threshold is possible and prediction are plausible. It is also worth remembering that numeric values are not important since they are related to the initial dataset which can be expanded; these models however are designed to work independently on the number of entries.

Conclusions

My Summer Student project involved the analysis of FTS logs using Machine Learning techniques. In order to perform this type of analysis, I deepened my knowledge with several tools and environments such as:

- *Python* - creating scripts for data preparation and modification;
- *DCAFPilot* - creating different models with Machine Learning algorithms;
- *R* - creating histograms and correlation matrices as well as fast operations on big datasets;
- *Hadoop framework*[6] - During this stay here at CERN, I also attended a series of lectures concerning Apache Hadoop[7] and its growth in Computing for High Energy Physics. In particular the Spark platform, which contains interesting Machine Learning tools that could improve this project for the near future.

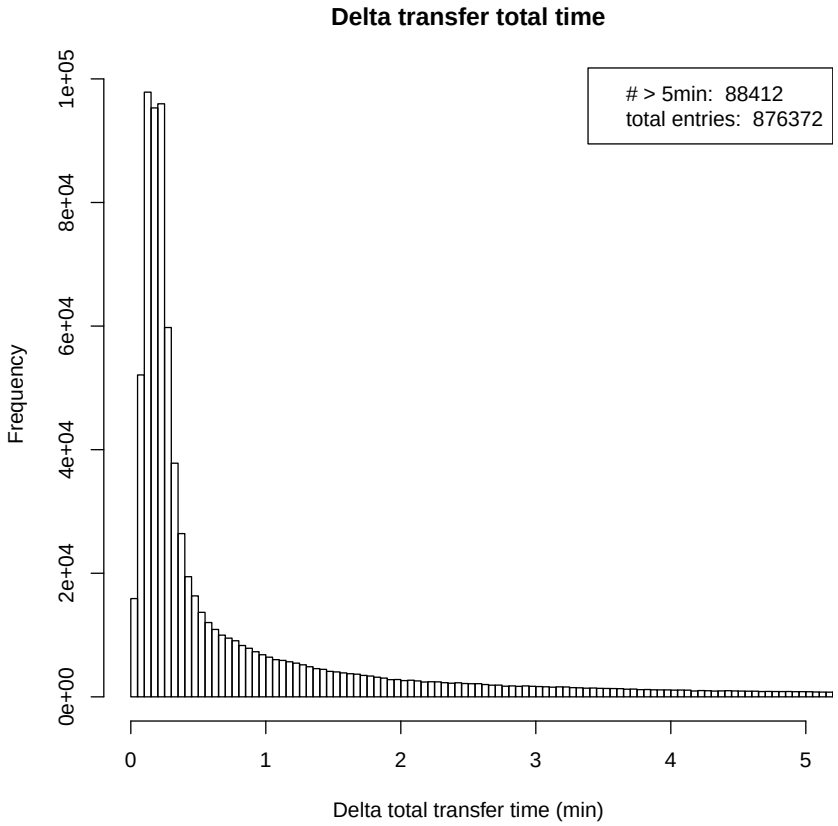


Figure 5. Histogram showing the delta transfer time of the training dataset.

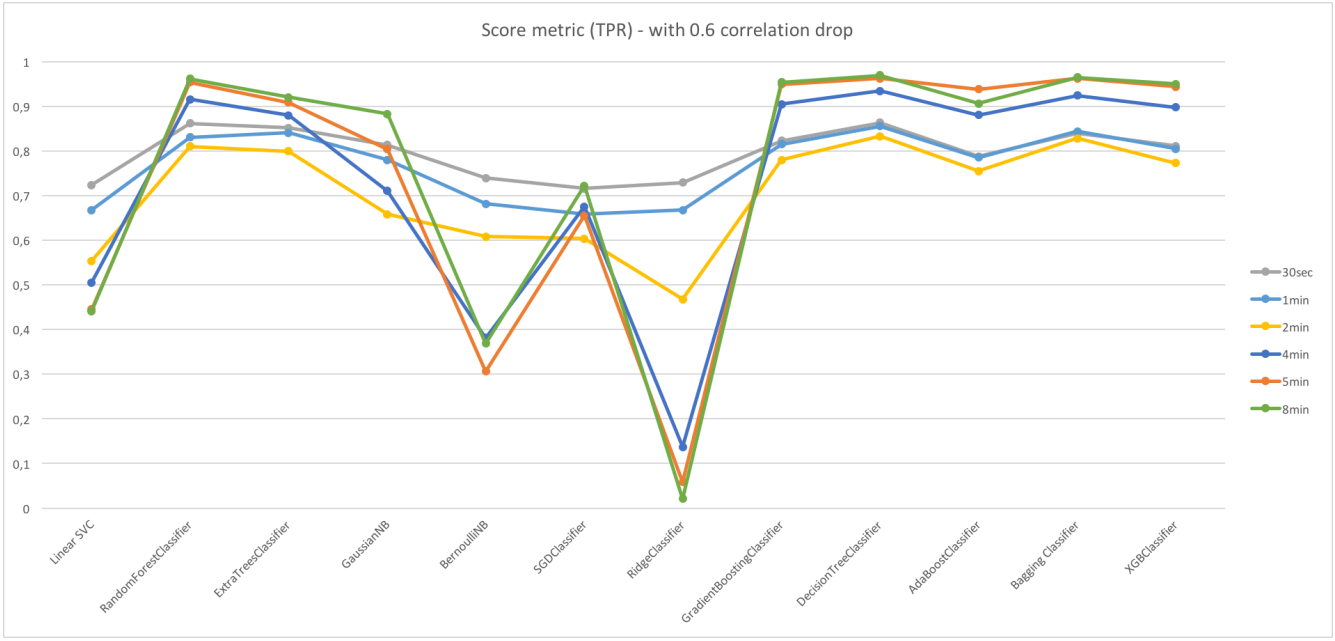


Figure 6. Comparison of different classifiers with different thresholds. In this plot the True Positive Rate (TPR) is shown.

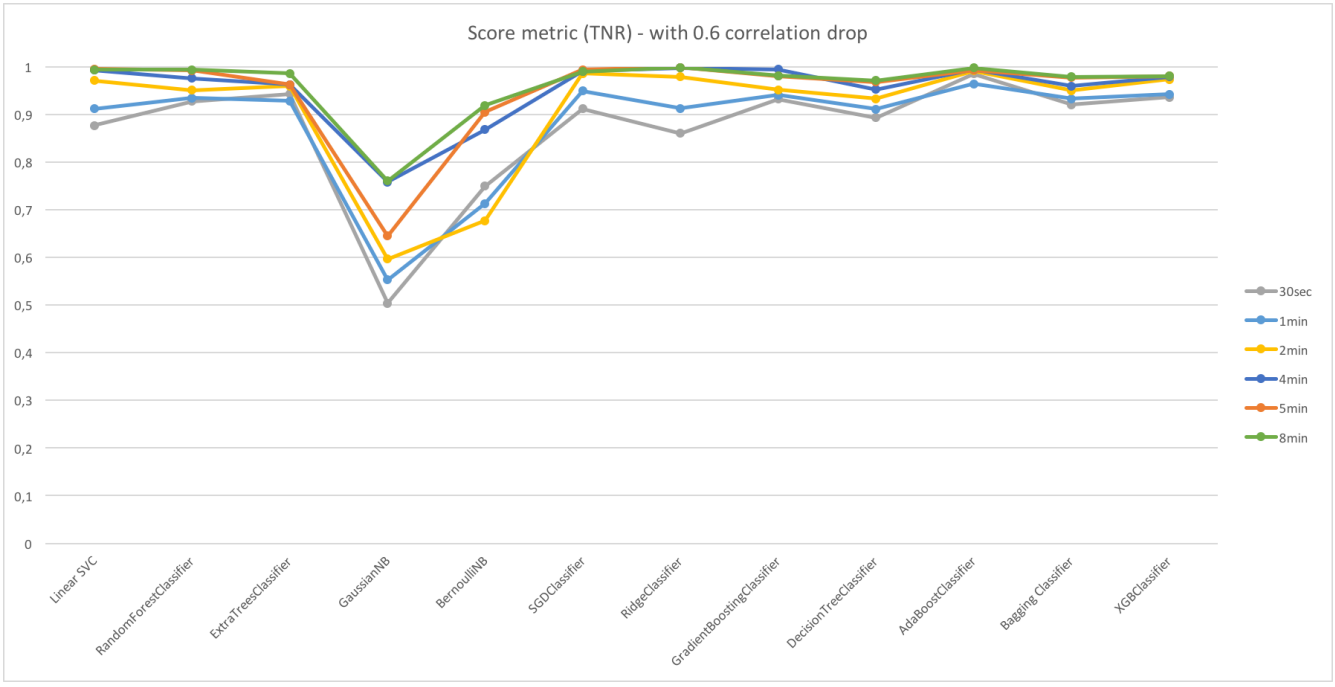


Figure 7. Comparison of different classifiers with different thresholds. In this plot the True Positive Rate (TNR) is shown.

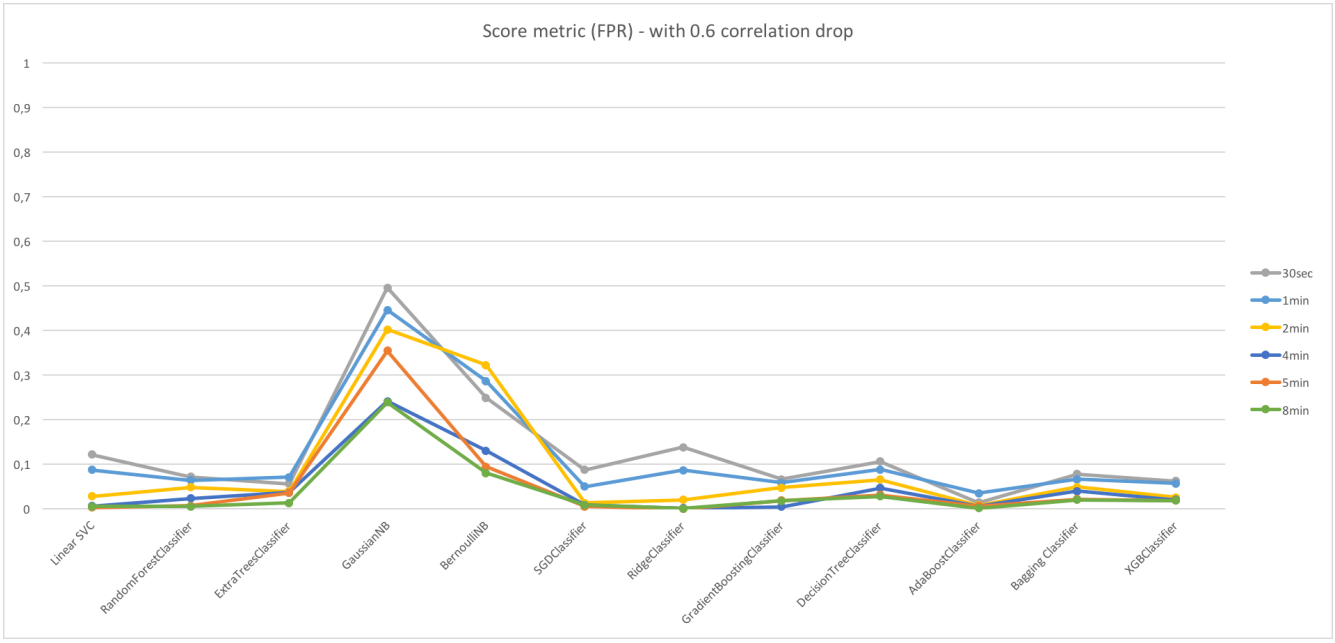


Figure 8. Comparison of different classifiers with different thresholds. In this plot the True Positive Rate (FPR) is shown.

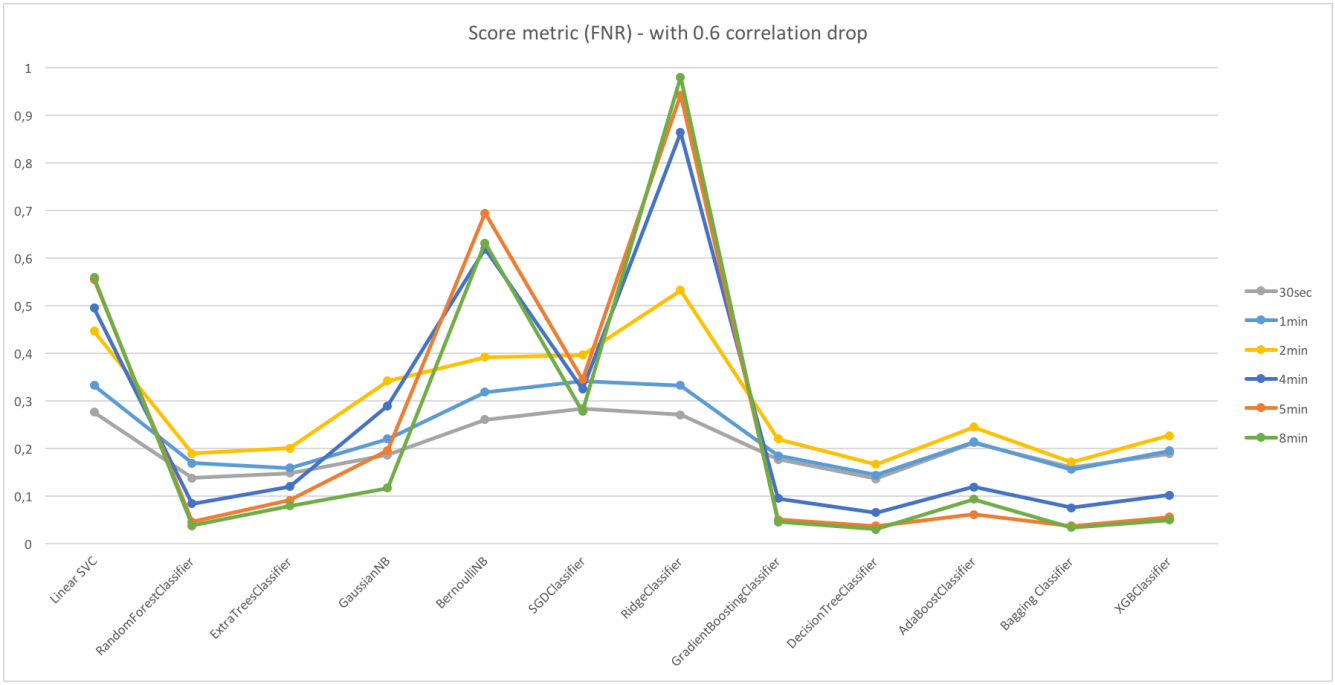


Figure 9. Comparison of different classifiers with different thresholds. In this plot the True Positive Rate (FNR) is shown.

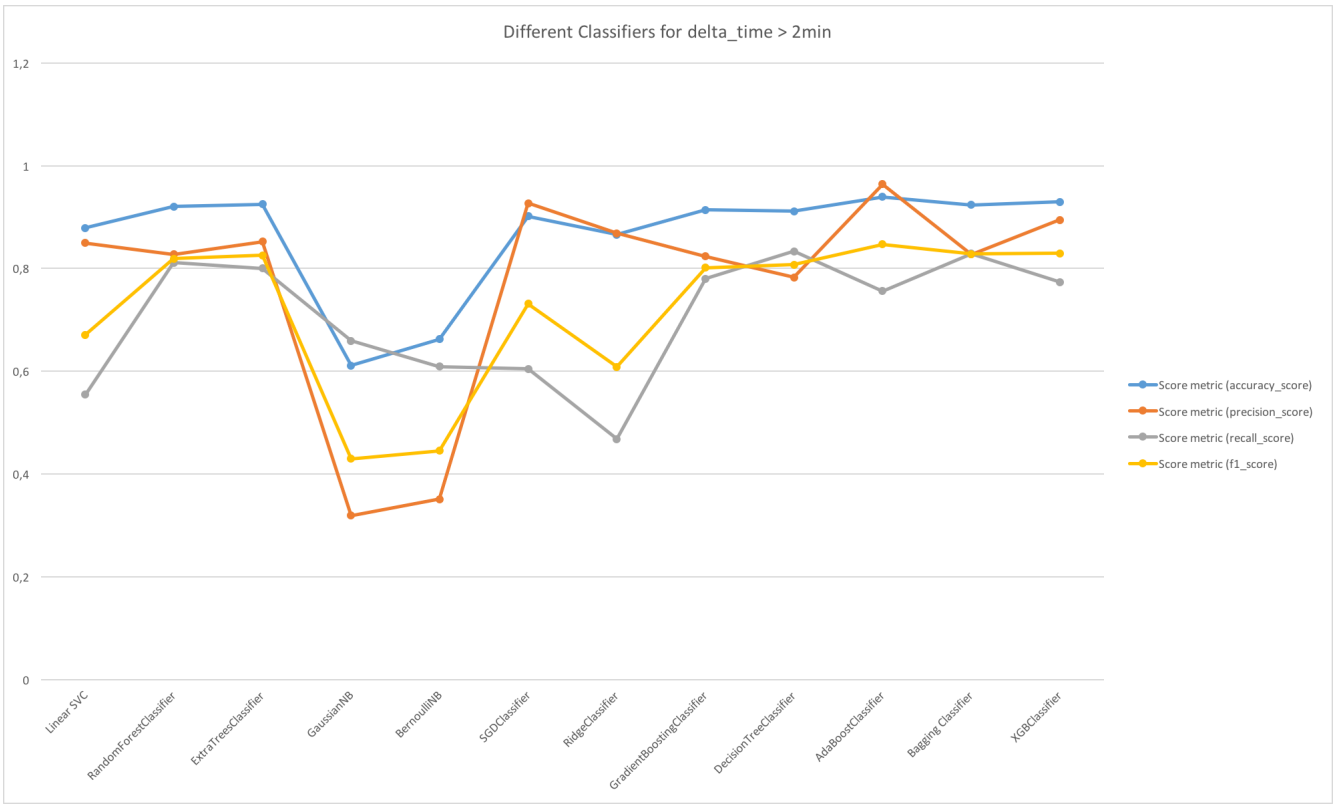


Figure 10. Comparison of different classifiers with 2 minutes threshold. In this plot Accuracy, Precision, Recall and F1 are shown.

This project is still a work in progress, many improvements can be done in order to have better results; plans for the short time include a better study of the FTS logs, a more accurate target variable choice and better thresholds as well as integrating the entire work done in PhEDEx[8] for a more specific analysis regarding CMS data replica performances. Long term plans instead involve the implementation of a real service for CMS monitoring systems for a better and faster control on WLCG transfers.

References

1. Dcafpilot repository. URL https://github.com/dmwm/DMWMAnalytics/tree/master/Popularity/DCAFPilot.
2. Giommi, L. Predicting CMS datasets popularity with Machine Learning. URL http://www.infn.it/thesis/PDF/getfile.php?filename=10091-Giommi-triennale.pdf.
3. Matonis, Z. Transfer metrics analytics project (2016). URL https://cds.cern.ch/record/2209068.
4. Diotalevi, T. Project repository. URL https://github.com/Tommaso93/FTS-and-ML.git.
5. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria (2016). URL https://www.R-project.org/.
6. Internet page of apache hadoop. URL http://hadoop.apache.org.
7. Hadoop lectures indico page. URL https://indico.cern.ch/event/546000.
8. Bonacorsi, D. *et al.* Monitoring data transfer latency in cms computing operations URL http://stacks.iop.org/1742-6596/664/i=3/a=032033.