PROCEEDINGS OF SCIENCE

Progress in Machine Learning studies for the CMS computing infrastructure

D. Bonacorsi*
University of Bologna, Italy
E-mail: daniele.bonacorsi@unibo.it

V. Kuznetsov
Cornell University, USA
E-mail: vkuznet@gmail.com

N. Magini

Fermilab, USA

T. Diotalevi

University of Bologna, Italy

A. Repečka

Vilnius University, Lithuania

Z. Matonis

Vilnius University, Lithuania

K. Kančys

Vilnius University, Lithuania

International Symposium on Grids and Clouds 2017 -ISGC 2017-
5-10 March 2017
Academia Sinica, Taipei, Taiwan

*Speaker.

PoS(ISGC2017)023

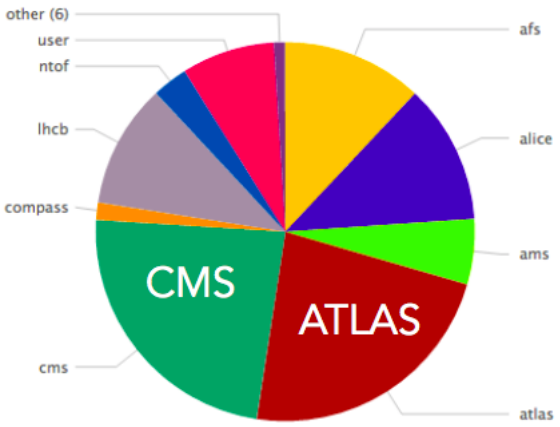

Figure 1: Data volumes collected by CERN experiments in 2016 (credits: I. Bird).

1. Data volumes and transfers in LHC Run-2

Computing systems for LHC experiments developed together with Grids worldwide. While a complete description of the original Grid-based infrastructure and services for LHC experiments and its recent evolutions can be found elsewhere [1, 2], it is worth to mention here the scale of the computing resources needed to fulfil the needs of LHC experiments in Run-1 and Run-2 so far. As of September 2016, the Worldwide LHC Computing Grid (WLCG) [3, 4] coordinated a hierarchical multi-tiered computing infrastructure, which comprises 167 computing centres in 43 countries. The total CPU power corresponds to about 3.8 millions of HepSpec06, i.e. the power of roughly 350k today's fastest cores [1]. The total disk space is around 310 PB. The overall tape space at the Tier-0 and Tier-1 sites is about 390 PB.

The Run-2 data taking period in 2016 broke many records. Massive volumes of data were written to storage during the year: only in the June-August period, the rate of data being written to storage was greater than 500 TB/day, with ATLAS and CMS contributing the most (see Fig.1). A total of 10.7 PB of data was recorded on disks only in July 2016. The entire 2016 year ended up to total volume of more than 35 PB of data for all LHC experiments, and the CERN tape archive reached the 160 PB level. The resulting load on the data distribution infrastructure, deployed across computing centres over high- performance networks, caused an increase in the global data transfer traffic, with overall rates as high as 40 GB/s (twice more than Run-1), regular transfers of the order of 80 PB/month, corresponding to many billions of files being dealt with in the experiments' data management system.

2. Towards Machine Learning approaches in CMS

The data volume in Grid transfers is only one of the aspects of the data challenges posed by Run-2 activities to the current CMS computing system. In several areas of computing operations, it is becoming clear since Run-1 that margins of optimisation may come from a deeper a-posteriori

[1] Actually, WLCG has deployed many more, as up 5 years old cores are still used in production.

understanding of the way CMS has been utilising both the Grid components and each layer of the experiment-specific software stack. This is a path that goes through the exploitation of existing tools and techniques to mine and process the information contained in archived resources (e.g. in databases and in operations logs) to extract useful information to tune the CMS operations. Among these techniques, Machine Learning (ML) approaches play a primary role. Through a careful information extraction and data preparation phase on LHC Run-1/2 archives and databases, such "computing metadata" can be made ready to be processed by ML algorithms - exploiting libraries and techniques quite widely adopted by the data science community - and thus offer actionable insight and predictive capabilities for future LHC runs. This data covers information on data transfers as well as job submissions, site performances, releases details, infrastructure and services behaviours, analysis accesses, etc. Here is where the Big Data zoology and Analytics techniques may come into play: a "data scientist" approach on the CMS computing metadata - complementary to the way we already handle the metadata for monitoring purposes - has a great potential and may ultimately allow CMS to build a complete and adaptive modelling of all the experiments workflows.

3. CMS framework for Machine Learning studies

This paper describes the current status of the set-up and execution of Machine Learning techniques to study CMS data transfer latencies, starting for the experience gained in studying the access patterns to CMS data by the physics community.

Only a relatively small fraction of real-world Machine Learning systems is actually composed of the Machine Learning code itself: there is a "hidden technical debt" [5] payed by the surrounding infrastructure, which may be vast and complex once it is called to enforce and support useful complex prediction systems. In exploring the adoption of a Machine Learning approach in studying any specific problem, it is hence valuable to first design a light toolkit that equips the researchers with an agile way to explore various paths and understand what the infrastructure of the final ML-based prediction system might be. Such a system allows e.g. to perform all the needed data treatment and handling actions that are preliminary to the application of Machine Learning algorithms themselves - in such a way that different technique (e.g. standard machine learning, online learning, custom based solutions) could be applied. CMS developed a framework called DCAF [7], that stands for "Data and Computing Analytics Framework". The aim of this project is to collect information from CMS data-services and represent it in a form suitable for analysis, e.g. via machine learning tools. DCAF is basically a wrapper around standard Machine Learning libraries such as scikit-learn and Spark MLib: it provides uniform APIs to invoke different classifiers without needing to know any detail of the underlying technology.

The DCAF framework consists of several layers. The *storage* layer is used to keep the information extracted from various CMS data-services; currently DCAF uses MongoDB and the associated py-mongo driver as a storage solution, which could be replaced with any other key-value store or RDBMS database. A *mapping* layer is used to provide mapping between sensitive information and its numerical representation (e.g. DNs into set of unique ids). An *aggregation layer* collects and merges information from various CMS data-services. A *representation* layer takes care of data representation in a form that is suitable for analysis frameworks (e.g. representing dataframes into CSV data format). An *analysis layer* will either use a custom analysis algorithm or provide bridges

to other learning system libraries; the easiest solution would be to use python scikit-learn library [6] which already provides a broad variety of learning algorithms. More details on DCAF can be found in [7].

4. The dataset popularity study: moving to Apache Spark

One application of DCAF has been a study of the data access pattern in CMS, in terms of the so-called "popularity" of the data for distributed analysis users. This may be defined by different possible metrics, e.g. number of accesses to a dataset, number of users accessing it, CPU-hrs spent in accessing it, etc. A proper definition of the popularity concept in the context of preparing for the application of ML techniques has been subject of previous work [8, 9]. The ability to predict the popularity of a CMS dataset is crucial as it allows to optimise the storage utilisation. Storage is by far the hot spot in any spending review of LHC computing, and in particular the disk storage is the most expensive computing resource: the ability to trigger the replication of heavily accessed datasets to more Tiers, as well as to trigger the disk deletion of not necessary replicas, or even entire single-copy not-accessed datasets, becomes a fundamental asset.

CMS used DCAF to attack the popularity prediction challenge as a supervised classification ML problem. Widely known classifiers like XGBClassifier and RandomForestClassifier performed in the model at the level of a TPR (True Positive Rate) of 90% or more, and a TNR True Negative Rate) of around 95% [9]. The study also showed at least a factor or 2.5 of difference in favour of RandomForestClassifier in the classifier running time [10].

Most useful for CMS operations is to constantly update the Machine Learning based predictions in "sliding" time windows. Using the DCAF framework, it may take a couple of days to collect up to 12 months of data. Dataframes from 6, 9 and 12 months in the past were used in a test to train the classifiers, and a dedicated study demonstrated that 6 months is a sufficiently large time window in terms of quality of the model built with this data: extending to a large window would have increased the risk of unwanted overfitting effects [9, 10]. As a consequence, the model was trained on 6 months worth of data to predict the popularity of CMS datasets in the up-coming week; then, this week was added to the training set and the first week in the past was dropped (thus yielding the time windows to indeed "slide"); then, the classifiers were re-trained and the predictions were extracted for the following week; and so on. Once the system was demonstrated to work, it was decided to improve the overall machinery by moving the modelling part to Apache Spark [11] and to exploit one of the HDFS clusters available at CERN. Three classifiers available in Spark+MLlib libraries - i.e. Decision Trees [12], Random Forest [13], Gradient-Boosted Trees (GBTs) [14] - have been used.

The results of the classifiers validation on the popularity use-case before and after moving to Spark are shown in Figure 2 and Figure 3 respectively. In Figure 2 the metrics used are TPR and TNR, while in Figure 3 the metrics used are the areas under PR and ROC curves. For reference: PR curves plot precision versus recall, where precision = TP/(TP+FP) and recall = TP/(TP+FN); ROC curves plot FPR vs TPR, with FPR = FP/(FP+TN), and TPR = TP/(TP+FN). The main result of this part of the work is that the Spark Machine Learning code has been developed and tested with the dataset popularity data sets and shows results that are comparable to those obtained previously using scikit-learn [9, 10]. It was demonstrated that a Spark platform can be used with no

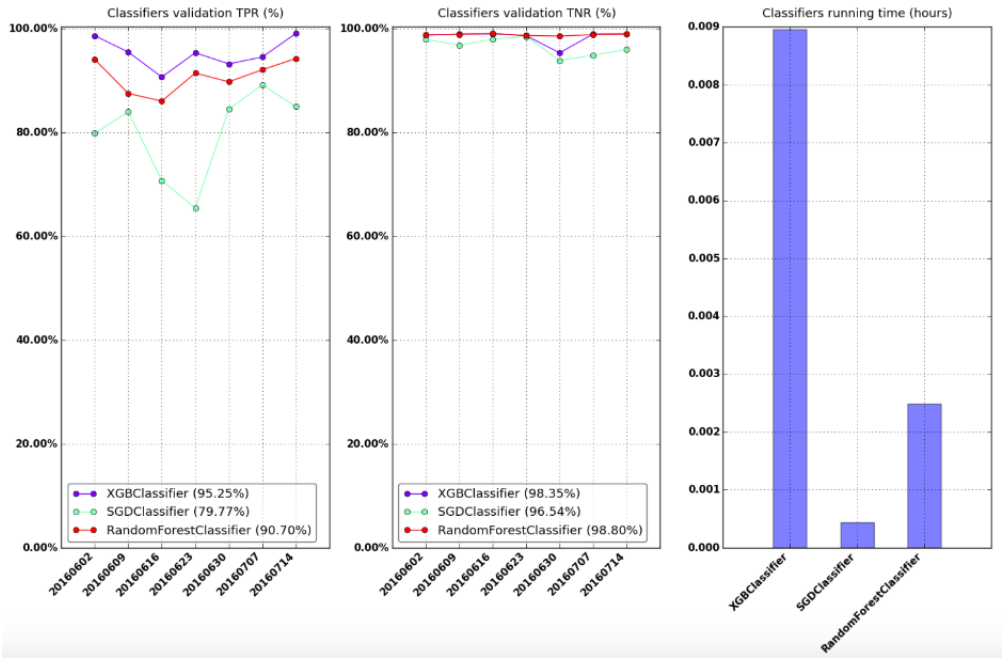

Figure 2: Classifiers validation using scikit-learn. Metrics are TPR and TNR (see test for details) [10]

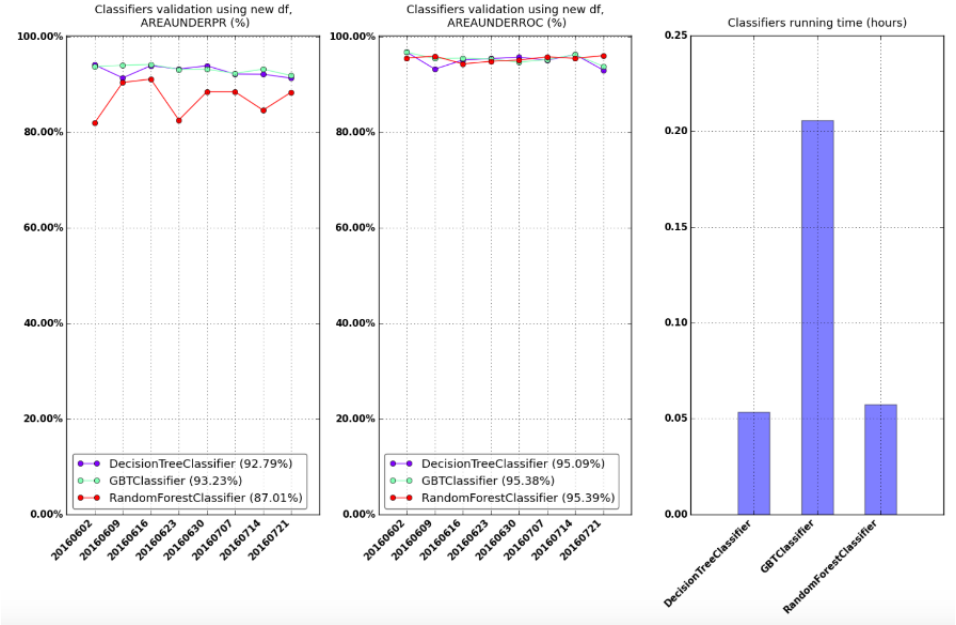

Figure 3: Classifiers validation using Spark. Metrics are the area under PR and ROC (see test for details) [10]

loss whatsoever in the quality of the model and its prediction, and in addition it allows to deal more efficiently with much larger dataframe (i.e. the data collection that is fed to Machine Learning). The difference in time that has been observed is not matter of concern: it comes from the implementation and on how work is scheduled on worker nodes in Spark; this overhead will diminish once we will analyse larger dataframes.

5. First studies of data transfer latencies with Apache Spark

As outline in the introduction, CMS transfer operations are well established to fulfil the management of the massive data volumes at LHC, and to achieve the desired level of throughput. Nevertheless, after Run-I and now in Run-II, the transfer operators still lack a reliable method to identify early those transfers that will eventually need manual intervention to reach completion. The so-called file transfer "latencies" can be related to a large set of underlying problems in the transfer infrastructure - from site issues of various kinds to network problems. The capability to spot them at a relative early stage would allow to take manual actions but - in a even more ambitious approach - the ability to predict transfer requests that may eventually suffer from some issues after the transfer has started could benefit from preventive actions.

It was understood since quite some years that this problem is very complex, and needs all possible data to be archived and kept for some time to make sure any future work could make use of this data and the most adequate technology to attack this problem could be exploited. For this reason, since 2012, PhEDEx - the reliable and scalable CMS dataset replication system - was equipped with a monitoring component to measure the transfer latencies at the level of individual files. Since then, statistics have been aggregated for blocks of files: a precious and large historical log of all transfer latencies started to be collected and it is being archived since then [15].

As a first approach, R-based studies were performed on this data to explore and classify various latency types and their signatures. A set of "skew variables" were introduced to properly describe the latency types, and a main categorisation into "late stuck" and "early stuck" was achieved [16, 19]. An example of the the late-stuck case (e.g. low performance tails observed in completing the transfer) is shown in figure 4. More examples and a thorough discussion can be found in[16]. This part of the work was crucial as a preliminary exploration to build the next steps.

The input data needs a transformation into a Machine Learning suitable form. The gLite File Transfer System (FTS) [17] raw data were collected and injected into a CERN HDFS cluster. A conversion step from JSON objects in ASCII files to a flat table format (CSV) was needed. Data preparation is then the primary work needed on this data set, and various actions were taken. Data needed to be cured as the JSON document structure was loosely matched, custom EOF, etc. Each record had nested records, and theses needed to be flattened. Hashing algorithms were used to convert text into numerical values. Placeholders were manually set for all missing attributes. All the data preparation actions described above were scripted into a unique piece of code which was designed so to be able to process relatively large (or the order of dozens of GBs) input files [18]. As it happens for all CMS Analytics efforts, the code is public for other CMS users to reuse it for similar projects.

Data preparation is not over. All data not suitable for a Machine Learning approach should be identified and dropped. For example: *i)* attributes regarding the end of a transfer (e.g. occurrence

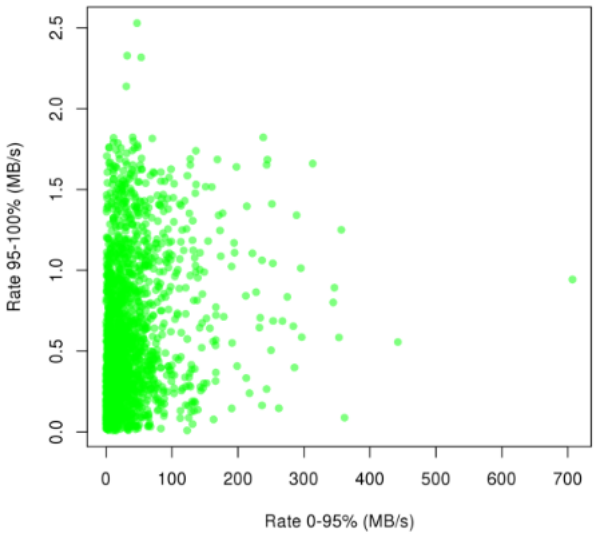

Figure 4: Example of transfer rate of the last 5% of a CMS data block vs rate of the first 95% for "late stuck" samples. More details in the text and in 4.

of a failure, or duration of a transfer phase, etc) are unknown at the start, and cannot be used for predictions; *ii)* some values are static through all datasets, i.e. uninformative and worthless, waste space and even may mislead algorithms; *iii)* obviously correlated attributes (e.g. number of files versus file size) also need to be dropped. All this actions together improve the purity of the data sample used for the study, but reduced its size to about 50% of the original CMS metadata set. The correlation matrix of all FTS logs variables is shown in figure 5.

Once these data preparation was completed, the transfer latency problem, exploiting this dataset, was attacked as a supervised Machine Learning classification problem, similarly to what was done for data popularity. We started at a manageable scale, first. The HDFS cluster contains months of FTS logs, corresponding to TBs of data: the first prototype, aimed at functionality demonstration, was done with just July 2016 as reference. Machine Learning algorithms were applied via the CMS DCAF machinery: a 70%/30% ratio among data used for training/validation was chosen; the model was trained on the training set, several Machine Learning algorithm were applies, and it was evaluated with standard scorers (from MAE to accuracy, precision, recall, F1). Some preliminary results are discussed in [18] and shown in Fig 6, 7, 8. At the current status of the work, we are focusing on RandomForest Regressor and GradientBoost Regressor from Scikit-learn library and XGBRegressor from XGBoost library. Fig. 6 and 7 show different classifiers in terms of TPR and TNR respectively. Time and memory consumption are also being investigates, despite not shown in these plots. In Fig. 8 a comparison of different classifiers in terms of Accuracy, Precision, Recall and F1 scorers is shown. Despite early to draw conclusions, it seems that RandomForestRegressor shows a good prediction rate overall, at a considerable cost in time and memory consumption, and may be outperformed by XGBRegressor after proper parameter tuning.

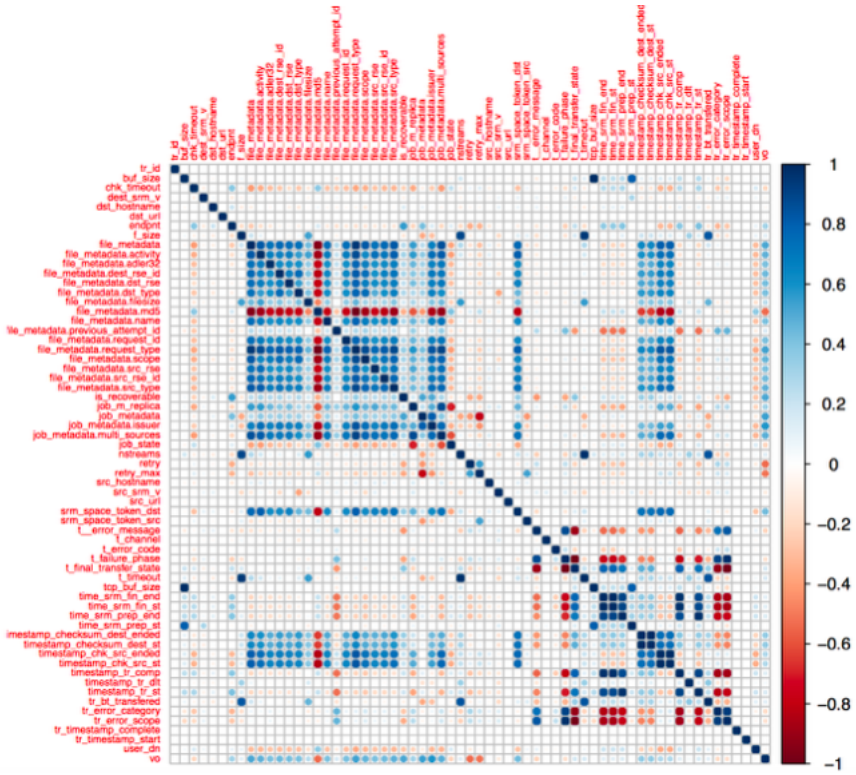

Figure 5: Correlation matrix of all FTS logs. See text for details.

6. Summary and next steps

In this paper we presented progress in adopting Machine Learning (ML) techniques on selected sectors of the CMS Computing infrastructure. In each case, we performed the data preparation phase, the necessary data cleaning and the evaluation of a variety of ML algorithms. We moved previous exploratory studies towards more performant solutions through the use of Apache Spark on Hadoop clusters maintained by CERN-IT. The obtained results demonstrate that we can apply ML in CMS Computing to the study of the CMS dataset access pattern and to the early identification of latencies in CMS data transfers on Grids. Currently we are exploring the possibility to apply ML predictions in a production environment.

In the example case of the popularity, these will be the steps to follow next: *i)* take a full pool of popular datasets for a selected time window; *ii)* take predictions of datasets which will become popular; *iii)* take an union of the aforementioned two groups and look at their existing number of replicas; *iv)* for those with no replicas, trigger one replica in PhEDEx (at high priority); for those with a single replica, trigger an additional replica (at medium priority); for those with >1 replicas take no actions; *v)* the replicas would need to be placed in "best" sites based on site metrics (e.g. site storage capacity, site stability in terms of site readiness and/or number of successful CRAB jobs running on-site), also taking into account the depth of PhEDEx queues and CRAB queues. These steps should be tried out on a few "dedicated" sites to measure the effect of ML-driven data placement, e.g. measure the latency of CRAB job for datasets you placed versus the normal latency in CRAB queues.

In the example case of the transfer latencies, these will be the steps to follow next: *i)* based on model predictions, identify transfer requests which may reveal latency symptoms which are characteristic of the "early stuck" of "late stuck" classes; *ii)* in case of "early stuck" candidates, trigger a check on the actual non-corruption state of all files belonging to the dataset, and invalidate files as needed; *iii)* in case of "late stuck" candidates, automatically point them to the attention of the transfer team in Computing Operations to closely monitor them and take actions as needed; These steps should allow to perform a clean measurement of the actual completion time of transfers acted upon based on ML-provided information versus the normal completion time of transfers left in the system with no ad-hoc actions.

The outlined steps will allow to draw lessons on how to move these ML-based approaches fully in production for the CMS experiment.

References

[1] M. Aderholz et al., "*Models of Networked Analysis at Regional Centres for LHC Experiments (MONARC), Phase 2 Report*", CERN/LCB 2000-001 (2000)

[2] I. Bird et al, "*Update of the Computing Models of the WLCG and the LHC Experiments*", CERN-LHCC-2014-014, LCG-TDR-002

[3] J. D. Shiers, "*The Worldwide LHC Computing Grid (worldwide LCG)*", Computer Physics Communications 177 (2007) 219–223, CERN, Switzerland

[4] http://lcg.web.cern.ch/lcg/

[5] D. Sculley et al, "*Hidden Technical Debt in Machine Learning Systems*", Software Engineering for Machine Learning, NIPS 2015, pp. 2503-2511

[6] http://scikit-learn.org/

[7] https://github.com/dmwm/DMWMAnalytics/tree/master/Popularity/DCAFPilot

[8] D. Bonacorsi et al, "*Exploiting CMS data popularity to model the evolution of data management for Run-2 and beyond*", J. Phys. Conf. Ser. 664 (2015) 032003

[9] V. Kuznetsov et al, "*Predicting dataset popularity for the CMS experiment*", J. Phys. Conf. Ser. 762 (2016) 012048

[10] K. Kancys, CERN Summer Student 2016 report (on github: http://cern.ch/go/8lGQ)

[11] http://spark.apache.org/

[12] https://spark.apache.org/docs/latest/mllib-decision-tree.html

[13] https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.tree.RandomForest

[14] https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.tree.GradientBoostedTrees

[15] T. Chwalek et al, "*No file left behind - monitoring transfer latencies in PhEDEx*", J. Phys. Conf. Ser. 396 (2012) 032089

[16] D. Bonacorsi et al, "*Monitoring data transfer latency in CMS computing operations*", J. Phys. Conf. Ser. 664 (2015) 032033

[17] https://egee-jra1-dm.web.cern.ch/egee-jra1-dm/fts/

[18] Z. Matonis, CERN Summer Student 2016 report (on cds.cern.ch: http://cern.ch/go/7RXN)

[19] T. Diotalevi, CERN Summer Student 2016 report (on cds.cern.ch: http://cern.ch/go/9hmp)

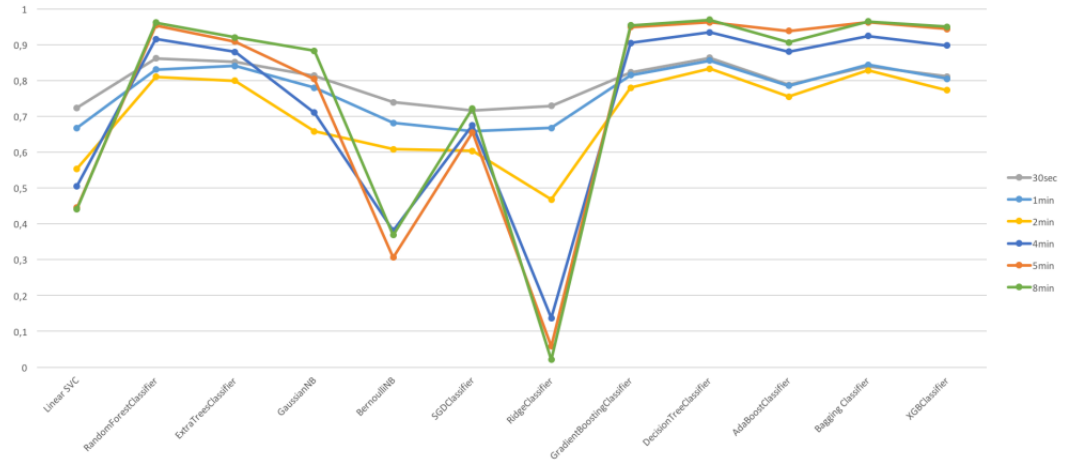

Figure 6: Comparison of different classifiers (x-axis) at different transfer completion time thresholds (different colours). In this plot the True Positive Rate (TPR) is shown [19].

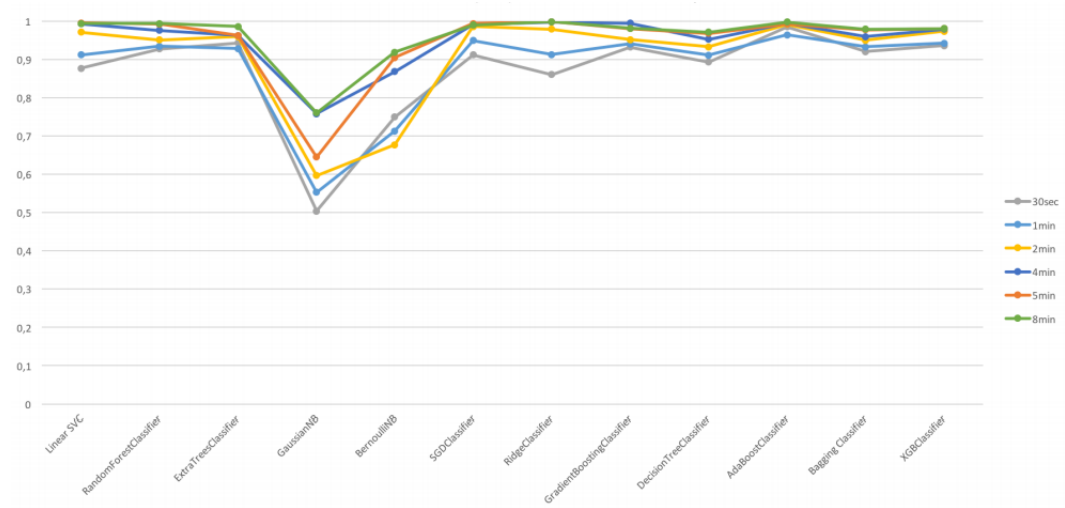

Figure 7: Comparison of different classifiers (x-axis) at different transfer completion time thresholds (different colours). In this plot the True Negative Rate (TNR) is shown [19].

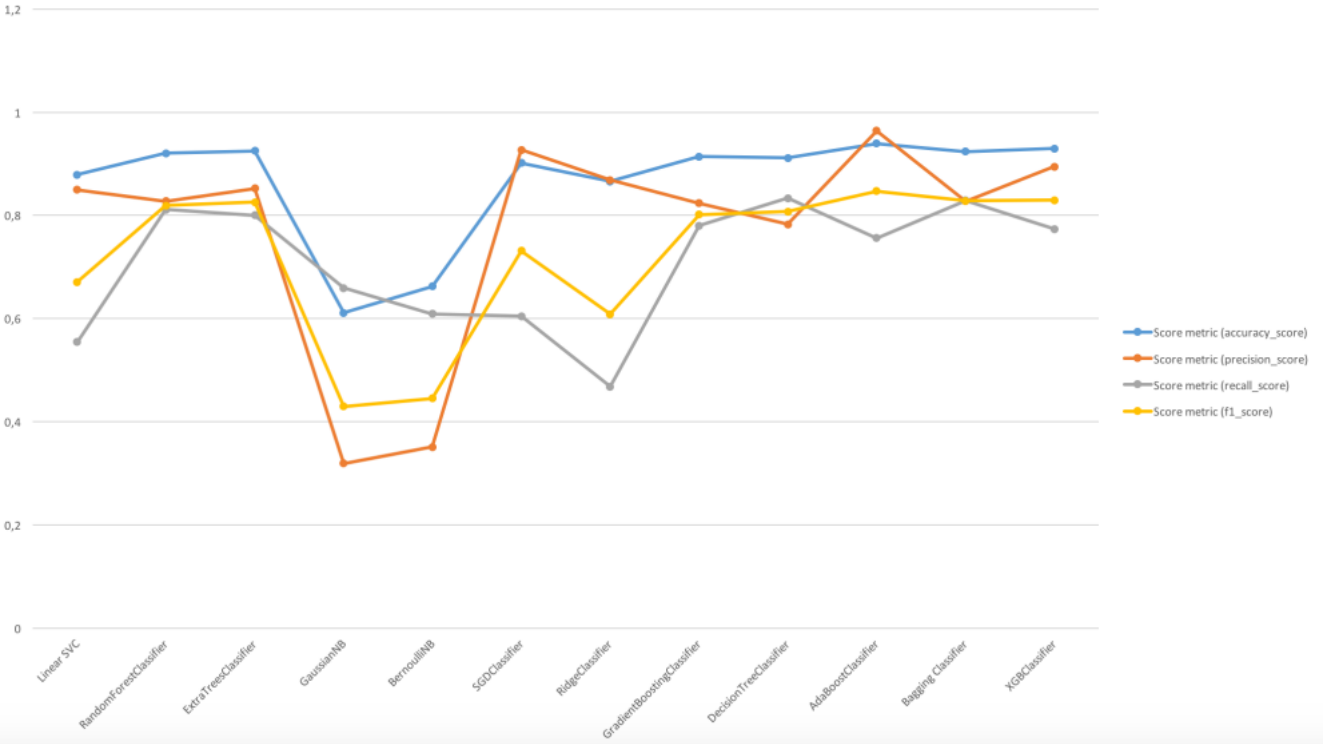

Figure 8: Comparison of different classifiers (x-axis) in terms of Accuracy, Precision, Recall and F1 scorers (in different colours) [19].