Noname manuscript No.
(will be inserted by the editor)

Dataset Popularity Prediction for Caching of CMS Big Data

Marco Meoni · Raffaele Perego ·
Nicola Tonellotto

Received: date / Accepted: date

Abstract The Compact Muon Solenoid (CMS) experiment at the European Organization for Nuclear Research (CERN) deploys its data collections, simulation and analysis activities on a distributed computing infrastructure involving more than 70 sites worldwide. The historical usage data recorded by this large infrastructure is a rich source of information for system tuning and capacity planning. In this paper we investigate how to leverage machine learning on this huge amount of data in order to discover patterns and correlations useful to enhance the overall efficiency of the distributed infrastructure in terms of CPU utilization and task completion time. In particular we propose a scalable pipeline of components built on top of the Spark engine for large-scale data processing, whose goal is collecting from different sites the dataset access logs, organizing them into weekly snapshots, and training, on these snapshots, predictive models able to forecast which datasets will become popular over time. The high accuracy achieved indicates the ability of the learned model to correctly separate popular datasets from unpopular ones. Dataset popularity predictions are then exploited within a novel data caching policy, called PPC (Popularity Prediction Caching). We evaluate the performance of PPC against popular caching policy baselines like LRU (Least Recently Used). The experiments conducted on large traces of real dataset accesses show that PPC outperforms LRU reducing the number of cache misses up to 20% in some sites.

Keywords Machine Learning · Big Data · Dataset Popularity · Classification · Caching Strategies

INFN & ISTI-CNR, Pisa, Italy
E-mail: marco.meoni@cern.ch

ISTI-CNR, Pisa, Italy
E-mail: raffaele.perego@isti.cnr.it

ISTI-CNR, Pisa, Italy
E-mail: nicola.tonellotto@isti.cnr.it

1 Introduction

The Compact Muon Solenoid (CMS) [13] experiment at the Large Hadron Collider (LHC) particle accelerator of the European Organization for Nuclear Research (CERN) designed and implemented a computing model that gave a crucial contribution to the recent discovery of the Higgs boson [14]. Within this model, distributed monitoring infrastructures have collected for many years many kind of data and metadata about the usage of the distributed computing infrastructure by its large community of users.

The monitoring infrastructure deployed at the Worldwide LHC Computing Grid (WLCG) [33] is undergoing a major revision to better cope with volume and variety of monitored data. This is due to either a higher LHC luminosity (an accelerator parameter directly linked to CMS discovery potential) expected in the next runs, and the deployment of new resources into the infrastructure. In this scenario, the traditional relational database systems previously used to store and serve monitoring events hit scalability limits [4].

To overcome this problem, in 2015 CMS has begun to store into a Hadoop[1] cluster nearly 4 PB of monitoring data produced by its monitoring systems. This data includes information about users, jobs lifecycle, resources utilization, sites efficiency, software releases, datasets access logs and usage statistics. The data is enriched with information coming from conference calendars, internal deadlines and trouble-ticketing services, which can bring valuable insights on the usage patterns.

Utilizing this huge collection of monitoring data, the CMS community is promoting exploratory activities using Big Data analytics approaches to discover patterns and correlations that can be exploited to reduce operational costs and/or to improve throughput and efficiency.

Hadoop offers the ability to process very large data sets using the MapReduce programming model [16]. Over the last decade, it has emerged as the *de-facto* standard for big data processing, largely adopted in both the research and industrial communities [2, 32]. A recent benchmark proved that the CMS dashboard and monitoring systems can largely benefit from MapReduce parallelism and Spark[2] in-memory distributed computing [27]. Specifically, the speedup achieved on the processing of dataset access information ranges between 2x to 50x compared to the previously used RDBMS architecture.

This work leverages the CMS Hadoop data store, and describes a scalable data mining pipeline designed to predict the number of future accesses to new and existing datasets, which can be further probed to identify the ideal number of dataset replicas and their best locations.

Data placement policies based on replicas of popular data across Grid sites have been researched since the early days [31] and have focused on the analysis of dataset popularity distributions from access patterns. We explore state-of-the-art machine learning techniques to learn from past usage information which dataset will become popular in the future and exploit such knowledge to optimize dataset caching at every site of the CMS distributed infrastructure. In order to model our popularity classification problem, we aggregate historical data about datasets ac-

[1] `hadoop.apache.org`
[2] `spark.apache.org`

cesses on a weekly basis. The dataset attributes include physics information parsed from the dataset namespace, as well as static and runtime statistics about the use of each dataset. On the basis of this information we construct different sets of features given in input to streamlining classifiers trained to predict dataset popularity. This prediction task involves many different cases. For example, consider new datasets produced from either LHC collisions or from simulated workflows. For these datasets we have only static information related to their creators, locations, and namespaces. On the other hand, existing datasets, in addition to the above metadata, are associated with historical usage information. The performance of the classifiers trained for the two cases are obviously different, even if the output popularity label predicted has the same meaning.

Our contributions. We list here the main contributions of this work:

1. We propose and implement a scalable data mining pipeline, built on top of the CMS Hadoop data store, to predict the popularity of new and existing datasets accessed by jobs processing any of the 25 event types stored in the distributed CMS infrastructure. We cast the problem of predicting dataset accesses to a binary classification problem where we are interested to forecast if a given dataset will be *popular* or not at time slot t in a given CMS site s, i.e., it will be accessed for more than x times during time slot t at site s. Our experiments show that the proposed predictive models reach very satisfying accuracy, indicating the ability to correctly separate popular datasets from unpopular ones.
2. We propose a novel intelligent data caching policy, PPC (Popularity Prediction Caching). This caching strategy exploits the popularity predictions achieved with our best performing classifier to optimize the eviction policy implemented at each site of the CMS infrastructure. We assess the effectiveness of this caching policy by measuring the hit rates achieved by PPC and caching baselines such as LRU (Least Recently Used) in managing the dataset access requests over a two-year timespan at 6 CMS sites. The results of our simulation show that PPC outperforms LRU up to 20% in some sites.

The paper is structured as follows. Section 2 outlines background knowledge and related work. Section 3 introduces the dataset popularity problem and describes the raw data available, the groups of features extracted from this data and how these features are used for the prediction task. Section 4 presents the experimental settings and discusses the results of the experiments conducted to asses the effectiveness of our solution to the dataset popularity prediction problem. The impact of data cleaning and feature engineering steps is also assessed. Section 5 describes how to harness popularity predictions to implement an effective dataset replacement policy for the CMS distributed infrastructure. Finally, section 6 draws some conclusions and outlines future work.

2 Background and Related Works

Predictive models based on statistical learning techniques are suitable for studying the performance and scalability of complex computing infrastructures. The training process requires to abstract features from a variety of measurements usually collected through historical logging activities, and to devise relevant metrics to estimate the behavior of the system under investigation. Log analysis related to the processing of structured or unstructured information collected through several layers of monitoring architectures is a promising field of research [29, 23].

In our context, dataset access logs represent an attractive yet critical source for data analysis. CMS has recently promoted research activities related to the mining of dataset access patterns leveraging Big Data analytics techniques. The management of the CMS infrastructure would largely benefit from the knowledge of these patterns, which could be exploited for driving dataset caching and replication policies. Job features, site characteristics and dataset access patterns can be analyzed with Machine Learning (ML) techniques able to predict with acceptable accuracy the system behavior, striking a balance between quality of service and storage allocation.

To this regard, two research directions are of particular interest: how to leverage scalable ML for predicting dataset popularity and how to use the obtained predictions to implement an efficient dataset caching policy.

2.1 Dataset Popularity

The field of dataset popularity prediction for the CMS experiment was pioneered by the initiative at Cornell University [24]. The paper shows a proof-of-concept based on a use case belonging to a larger CMS Analytics project having the ultimate goal to build adaptive models of CMS Computing [7]. The authors tested the feasibility of training a dataset popularity classifier from the information retrieved from the previous CMS monitoring infrastructure.

Another major experiment at CERN, LHCb, relies on a data mining engine for studying the interactions of heavy particles containing the bottom quark. Yandex has provided the LHCb core simulation software with fast access to the details of specific event types. Geared specifically to the LHCb needs, the search system indexes and identifies events satisfying specific selection criteria using the MatrixNet algorithm [22].

Similarly, the ATLAS experiment has developed a very simple popularity prediction tool [5] for its distributed data management system. Historical access information about files, datasets, users and sites are used to make a prediction about the future popularity of data and its possible trends.

Dataset popularity is also addressed outside the particle physics domain. For instance, Netflix handles video replicas and tries to implements services that give users what they want, ideally before they know it. This is achieved through multiple neural network models for different distributed datasets [15]. The approach is based on the intuitive idea that the preferences of a user can be inferred by exploiting past ratings of that user, as well as users with related behavior patterns. Since only a few factors contribute to individual taste, hyper-parameter optimization is performed upon each combination of users' preference parameters.

Each of these systems is tailored to the specific needs and data formats of the corresponding research domain. They need to take into account billions of records of historical data, which makes it impractical or impossible standard processing with conventional DBMS. Big Data infrastructures can aid the development of simulation systems capable to aggregate massive amount of data with speed and ease, as well as trigger the discovery of common patterns.

Our work moves forward from the experience in [24] and explores Big Data ML techniques aligned with CMS recent developments [27]. Our data analytics and prediction platform is implemented by a pipeline of scalable Spark components fully integrated in the CMS Hadoop data store. Previous work focused on only five physics datatiers, while we propose an online solution for building classification models targeting the entire range of 25 CMS datatiers (thus covering the whole CMS experiment) and keeping them fresh through a viable model update strategy. Furthermore, we discuss in details the features used to train our model, introduce new features involving an improvement of about 10% in the overall classification accuracy, and distinguish the prediction task for new and already existing datasets. Unfortunately, the results of the previous experience are not comparable with ours due to the radical changes in the monitoring infrastructure [27, 28].

2.2 Dataset caching

Our work exploits the outcomes of the proposed dataset popularity prediction pipeline to feed a novel intelligent caching strategy. A similar approach is discussed in [8] where the LRU policy is extended to make it sensible to Web access patterns extracted using data mining techniques. The authors build a caching strategy called S2 implementing multiclass classification based on decision trees. S2 is applied to a label representing the distance between a given URL and the next access. S2 results in having an accuracy ranging between 70% and 87% on the two workloads tested.
The work in [30] proposes a linear ranking function for replacing the objects in a Web document cache and improves the results in [8]. Rather than extending the LRU policy by using frequent patterns from Web log data, [30] uses a model that extends the Greedy Dual Size Frequency caching (GDSF) policy [12]. A survey of the approaches to data caching based on web logs was conducted by [18]. Although the work is not very recent, it is interesting to notice that only three works used classifiers as we do. The author in [34] leverages previous results from [30] and extends the ranking function with predictions of the future occurrence frequency of an object based on a rule table, and improves the hit rates of the basic GDSF policy. A similar approach based on rule table is described in [21].

We share the same experimental approach of the above related works by running simulations on increasing cache size with actual trace data. Nonetheless, while the better performance in previous approaches against LRU-based policies is due to their capacity to adapt to data access patterns, we argue that in our setting LRU itself has its own ability to adjust to data locality and we rather enhance this attitude by supplementing the eviction mechanism with the knowledge of future accesses. Moreover, our work addresses a completely different problem – the CMS dataset accesses – characterized by different data access distributions and a much more complex feature space with respect to Web browsing.

3 A scalable pipeline for dataset popularity prediction

Our scalable dataset popularity prediction pipeline is depicted in Fig. 1. It highlights the process chain, from the raw data ingestion and preparation steps up to the ML component producing (and keeping updated) the machine learned model which is in turn exploited by the PPC strategy driving dataset caching in the various CMS sites.

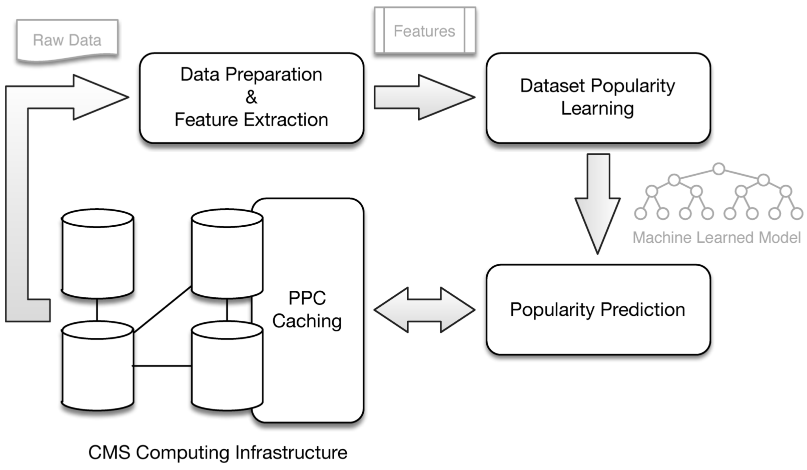
Raw Data
Features
Data Preparation & Feature Extraction
Dataset Popularity Learning
Machine Learned Model
PPC Caching
Popularity Prediction
CMS Computing Infrastructure

Fig. 1 The pipeline of components implementing dataset popularity prediction and the use of the ML model trained by the PPC strategy.

In the following we discuss in details the main components implementing the pipeline for data preparation and training of the dataset popularity classifier.

3.1 Data preparation and feature extraction

Raw data at the LHC is organized hierarchically by time-windows called *Run*, a unit of data acquisition or simulation process. A similar structure is simulated for Montecarlo events. During each Run, either it is related to LHC collisions or Montecarlo simulation, data is organized into sets called *blocks*, the smallest units of transferable information among sites. Data consists of mostly ROOT-format [10] *Logical File Names* (LFN), a site-independent name for a file. In turn, blocks are grouped in datasets, which constitute the entry point for data analysis and data transfers. On average, a dataset has a size of about 2 TB and includes few or thousands of LFN.

A typical CMS dataset namespace is defined by three main parts concatenated in a slash-separated name `/primaryDataset/processedDataset/dataTier`, as it is shown in Fig. 2. However the syntax is not fully enforced and this constitutes a problem for automated parsing tools. `primaryDataset` is a string that describes either the selection process on real data or the physics simulated event types.

`processedDataset` describes the processing chain that is applied and the data taking era. `dataTier` (or, simply, *tier*) describes the kind of event information stored from each step in the processing simulation and reconstruction chain. Examples of tiers include RAW and RECO, and for Montecarlo event, GEN, SIM or DIGI or a combination of them. For example the GEN-SIM-DIGI-RECO dataTier includes the generation (Montecarlo), simulation (Geant), digitalization and reconstruction steps.

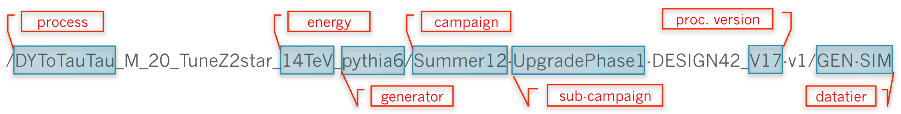
process
energy
campaign
proc. version
/DYToTauTau_M_20_TuneZ2star_14TeV_pythia6/Summer12-UpgradePhase1-DESIGN42_V17-v1/GEN-SIM
generator
sub-campaign
datatier

Fig. 2 Main information that can be extracted from the dataset name.

The building blocks of the dataset name constitute the first set of categorical features that can be harnessed to run ML algorithms. Additional features are derived by combining and aggregating on a weekly basis the dataset properties like size or number of files, and the usage information such as number of accesses, number of bytes read, number of users using it, CPU time spent, etc. We name *dataframe* the result of this process of feature construction. This brings to the identification of a first set of relevant attributes. They are summarized in Table 1 and represent the basic set of training features exploited by ML algorithms. A portion of the raw data and the whole set of weekly dataset popularity samples are publicly available[3] in SVMlight format in order to make results reproducible and foster knowledge and improvement in the field. Datasets and user names are converted into numeric values for anonymization.

3.2 Dataset popularity learning

Modeling dataset popularity represents the first contribution of this work. We must take into account that the definition of popularity is all but unambiguous. Informally speaking, we can say that dataset popularity quantifies the user activity. A dataset is popular when it is used "often" in user jobs. Understanding whether a dataset is popular or not helps to answer to questions like what new datasets will be accessed the most or how many replicas of them is convenient to store and where these replicas should be located in the CMS distributed infrastructure. This would improve the efficiency of any distributed storage system because dataset access can be optimized creating several replicas at the sites where they are most likely to be used.

The relationship between the features listed in Table 1 and popularity class is primarily related to the dataset usage metrics like *naccesses*, *cputime*, *nusers*, *readbytes*. In fact, different cutoffs applied to the values of these usage features, or to their combinations, can lead to different definitions of popularity, with each dataset being designated as popular or unpopular depending on whether the resulting value is higher or lower than the cutoff.

[3] `github.com/mmeoni/CMS-popularity`

Table 1 Categorical and numerical training features extracted from the set of possible attributes.

Type	Name	Description
Static features	week	week of aggregation
	size	size of a dataset, expressed in GB
	nfiles	number of LFN in the dataset
	nblocks	number of blocks in the dataset
	nevents	number of collisions described in the dataset
Physics domain	process	group of events with related topology
	energy	energy at which collisions take place at LHC
	generator	software for events generation
	campaign	data reprocessing and simulated events production phase
	subcampaign	a subphase of a given campaign
	version	version of the processed data
	datatier	type of event information stored in the dataset
	software	reconstruction software
	era	acquisition Era
	luminosity	number of collisions produced in a time and space unit
Infrastructure	serverdomain	domain satisfying the access request
	serversite	server name
	servercountry	country the server belongs to
	clientdomain	domain of the client requesting the access to the dataset
	clientsite	physical site of the client
	clientcountry	country the client belongs to
	username	username accessing the dataset
	protocol	application protocol used to access the dataset
Usage features	naccesses	number of weekly accesses to a dataset
	nusers	number of weekly users accessing a dataset
	cputime	CPU time weekly utilized to access a dataset
	readbytes	weekly number of bytes read, expressed in GB
	nreplicas	weekly number of replicas for a dataset
	nconferences	number of conferences where the DS is mentioned
	μ(naccesses)	average number of weekly accesses
	σ(naccesses)	variance in number of weekly accesses

Since the real ratio measured from the replica catalogue at CERN is roughly 25%, it makes sense to start our studies from a realistic threshold that already satisfies the current storage deployment. That ratio results in the cutoffs listed in Table 2, which can be in turn used to convert the regression problem of predicting the number of future accesses to a dataset into the related binary classification problem of predicting if a dataset will be popular in the future.

Table 2 Popularity cutoffs. Different thresholds applied to the usage features can lead to different definitions of popularity.

Usage feature	Cutoff
naccesses	50
cputime	1 h
nreplicas	1
readbytes	400 GB

Training the predictive models that will suggest what datasets become popular over time leverages Spark and its scalable machine learning library. Spark has proved to be very successful as an effective platform for running CMS computing analytics

[28]. Hence, it can be used as a tool to streamline and compare different predictive prototypes capable of gathering dataframes that organize features extracted from several CMS data services. Scalability and fast distributed data processing are two critical factors for current CMS data analyses; Spark offers both, together with a simple programming abstraction through the Scala language, which in turns provides powerful caching and persistence capabilities. This allow us to compute dataframes in nearly real time as the raw data is being produced, with the addition to pass them on directly to the MLlib [26] algorithms available in Spark.

For the purpose of this work, we test a set of standard ML baseline algorithms (Decision Tree, SVM, Logistic Regression) and two state-of-the-art algorithms based on decision trees: Random Forest (RF) [9] and Gradient Boosted Trees (GBT) [20]. The hyper-parameters used to train the classifiers are chosen according to the recommendations in the MLlib developer guide[4]. In modern applied machine learning, models based on tree ensembles like the ones learned using RF and GBT algorithms almost always outperform singular decision trees or simpler models [17]. Moreover, they results to be much more robust to overfitting. RF is an ensemble learning boosting meta-algorithm that trains a set of decision trees by exploiting sampling with replacement on both the features and sample space. GBT instead uses any arbitrary differentiable loss function to drive the iterative learning of new decision trees minimizing the error due to incorrectly classified examples. Although there is substantial variability in the performance measured across problems and metrics from different experiments and domains, GBT performs generally better than RF, particularly when dimensionality is low [11]. This is also evident by the fact that GBT are the most common choice in solutions for ML competitions, such as Kaggle[5].

In binary classification, accuracy may not be the best estimator to use [35]. Guessing the more common class could in fact yield very high accuracy in presence of highly unbalanced classes. For this reason, it is usually preferable to use different metrics that are less sensitive to imbalance when evaluating the predictive performance of classifiers. Our goal is to identify the members of the positive (rare) class successfully, the popular dataset in our study.

Thus, we consider two additional performance measures: *precision*, which measures the classifier exactness, *recall*, which measures the classifier completeness. Because they both provide valuable information about the quality of a classifier, they are further combined into the single general-purpose F1-score, which is defined as their harmonic mean. The F1 score favors classifiers that are strong in both precision and recall, rather than classifiers that emphasize one at the cost of the other.

We finally combine true and false positive rates in the *Receiver Operating Characteristic* (ROC) curve. The area under this curve represents the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative one: the larger the area under the ROC (auROC), the better the discrimination power of the predictive model.

[4] `spark.apache.org/docs/1.5.2/mllib-classification-regression.html`

[5] `www.kaggle.com/competitions`

4 Experiments

CMS dataset access logs are stored in a Hadoop File System (HDFS) and aggregated on a weekly basis as described in the previous section. Section 4.1 discusses why this aggregation period was chosen as prediction time granularity. Raw data aggregation, training of the model and evaluation on the test set are implemented via a scalable pipeline of Apache Spark components developed in Scala, a Java binding of Spark. Additionally, Zeppelin is used as Web-based notebook for quick interactive data analytics. It allows for straightforward prototyping of aggregation and learning algorithms.

Popularity raw data on HDFS is stored in CSV, JSON, Parquet or AVRO formats, and is available starting from March 2015. This data represents the output of several streamers performing continuous data ingestion to the Hadoop ecosystem. Table 3 lists the main sources of structured and unstructured data feeding feature construction.

Table 3 Information sources used to compute the features of dataset popularity samples.

Source	Items	Type	Description
EOS	786,934,116	structured	Disk storage system at CERN
AAA	2,370,570,956	structured	CMS XrootD federation for Grid data
CRAB	1,177,951	structured	Grid infrastructure for job submission
DBS3	5,193,522	structured	Global dataset/fileblocks catalogue
Block-Replicas	805,614,541	structured	Global replica catalogue
PhEDEx	58,227,786	structured	Fileblock locator and export service
CADI	1,791	semi-struct	CMS conference database

The set of dataset popularity samples built from these specific sources contains instances of both unpopular and popular dataset in the week observed. A dataset is labeled as popular according to the cutoffs in Table 2. Specifically, each dataset weekly sample is initially modeled by the 31 features listed in Table 1 and one additional popularity label 0/1 that assesses the popularity of the dataset in the week successive to the one to which the sample refer to.

We follow the process discussed in section 3.2 and compute the performance metrics. In the tests conducted we observed that the cutoff on the number of weekly dataset accesses ($naccesses > 50$) results in a better auROC than the cutoff on other metrics such as *cputime* (plotted, as example, in the same figure). For this reason, *naccesses* is the selected metrics for all the next experiments.

4.1 Choice of the prediction timespan

We analyzed the input dataframes to determine what time interval represents the best timespan for dataset popularity prediction. The dataset popularity labels used during the training are computed by exploiting the complete knowledge available in the actual historical data. Our classification task asks in fact to predict if a given dataset will be popular in the next future. Since the historical data gives us a complete knowledge of what happened in the past, we use this knowledge as an oracle for building a large set of training samples. A training sample has

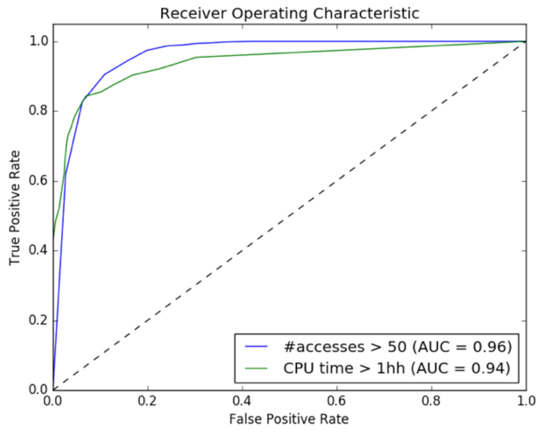

Fig. 3 ROC of the popularity classifier with different cutoffs (50 accesses or 1 hour CPU time).

the form $< id, t, f_1, f_2, \ldots, f_n, Y >$, where *id* is the identifier of the dataset, t is the discrete time at which the features $f_1, \ldots, f_n$, refer to, and Y is the binary popularity label indicating if dataset *id* will be popular or not at time $t+1$. Note that we produce several samples for each dataset, one for each pair of time intervals $(t, t+1)$ covered by our historical data. The value of label Y for dataset *id* at time t will be 0 (*unpopular*) when the number of accesses to *id* at time $t+1$ is lower than the threshold, 1 (*popular*) otherwise.

Thereinafter we consider a single week as the timespan for our popularity prediction problem. We choose a weekly timespan justified by the empirical observation that when the temporal window in the training set is increased for example to 3 weeks (i.e., dataset popularity in $week_{i+2}$ predicted using the features computed from data collected in $week_i$ and $week_{i+1}$), the number of positive samples decreases remarkably and prediction accuracy decreases.

The two upper plots in Fig. 4 display the number of datasets that have been accessed in 2 and 3 consecutive weeks, while the two distributions in the bottom of the figure show the *total* number of weeks each dataset is accessed and the number of *consecutive* weeks each dataset is accessed. From these plots we see that most datasets have a relatively short access pattern and increasing the timespan does not allow to capture their dynamics.

Table 4 Characteristics of the dataset used for training and testing.

Number of samples	307,025
Positive samples (popular datasets)	68,661
Negative samples (unpopular datasets)	238,364
Number of different datasets	47,775
Number of different CMS sites	63
Number of different CMS tiers	25
Timespan (in weeks)	105

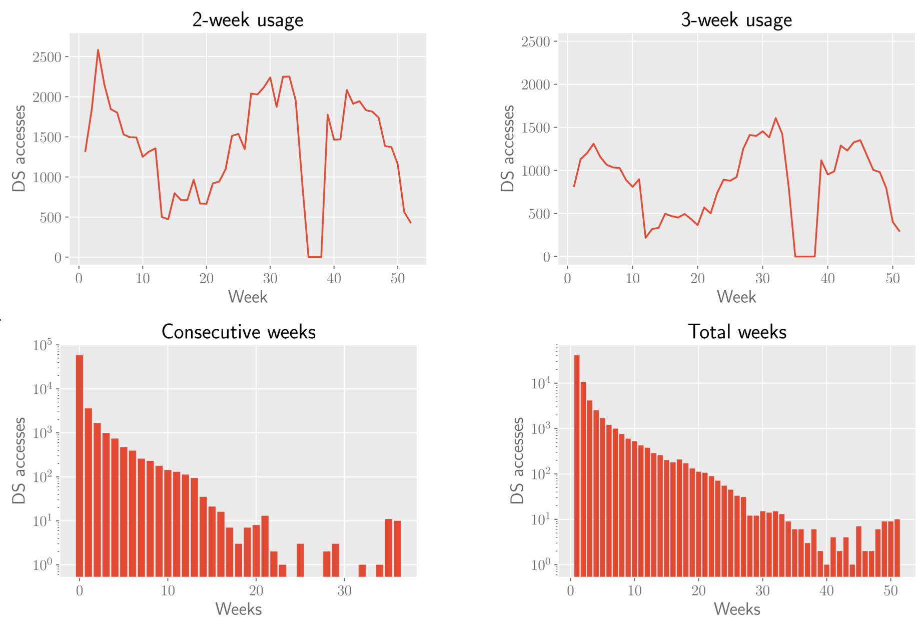

Fig. 4 Number of dataset accesses vs. number of total and consecutive weeks and number of accesses in 2 and 3 consecutive weeks.

Table 4 details the characteristics of the dataset used for learning and assessing the classifiers for dataset popularity. In this dataset each sample is initially represented by the 31 features of Table 1. In the following we will show how additional features are introduced to enhance prediction accuracy.

4.2 Prediction performance

Table 5 summarizes the performance of models learned with different ML algorithms when the cutoff *naccesses* > 50 is used to establish the popularity label for the week successive to the one when the features in each sample are computed. The feature set from Table 1 is applied to all models. Models are trained using Spark MLlib machine learning algorithms on more than 300k samples derived from dataset accesses in 2015 and 2016. The measures of the tests' accuracy are used as baseline for a series of enhancements that will be discussed in Sect. 4.3.

Table 5 Performance of various classifiers for naccesses > 50

Classifier	**auROC**	**Accuracy**	**Precision**	**Recall**	**F1**
Decision Tree	0.647	0.743	0.737	0.742	0.740
SVM	0.660	0.740	0.719	0.743	0.716
Logistic Regression	0.645	0.720	0.698	0.717	0.650
Random Forest	0.744	0.758	0.782	0.757	0.769
GBT	**0.773**	**0.769**	**0.792**	**0.770**	**0.781**

The ensemble methods in our setup confirm to be more precise than base models. Consistently with the literature, *Random Forest* and *GBT* models result to significantly outperform *Decision Tree, SVM,* and *Logistic Regression* models learned on the same data.

4.3 Enhancement of the prediction model

In the following we discuss how the prediction performance reported in Table 5 can be improved by further cleaning the data and introducing additional features. The strategies tested for improving the performance of our best classifier (GBT) are described below:

– **Removal of Unpopular Tiers (RUT)**. As already described, LHC event information from each step in the simulation and reconstruction chain is logically grouped into what is called a tier. Fig. 5 shows the popularity distribution of the 25 tiers defined. Among these 25 total types, only the 12 of them that are more used and more important from the physics point of view are retained in the dataset. The tiers filtered out are the ones associated to testing jobs and the ones that are utilized in a short timespan. This filtering produces a cleaner training set and improves the precision of the model and the F1 score by 4-5% as shown in the row labeled GBT_{RUT} of Table 6.

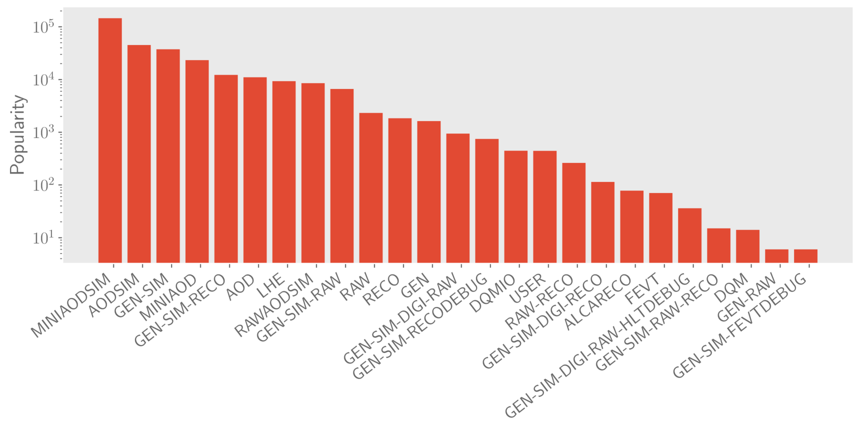

Fig. 5 Tier popularity distribution.

– **Rank-based features (RBF)**. The analysis of the distribution of numerical features such as *cputime*, *naccesses* or *readbytes*, shows how their range is subjected to significant fluctuations. For examples, the number of weekly accesses to a dataset can vary from few units to tens of thousands. Even larger is the range involving the total number of read bytes: while a Montecarlo simulation job is mostly CPU bound, an analysis job can perform a large amount of reads. No matter what the measure is, usually there exist a small amount of values that can be seen as ”outliers”, while the values above the popularity threshold

are affected by scattering. In order to reduce variance we sort the numerical features in all the samples by increasing values and substitute in each sample the real feature value with its rank in the global order [25]. While low values do not incur in remarkable transformation, the scattered instances occurring for high and rare values become compacted. The introduction of rank-based features has the advantage of further improving the classification accuracy as shown in the row labeled $GBT_{RUT+RBF}$ of Table 6.
– **Daily-trend features (DTF)**. The feature set is extended with the number of accesses on each weekday and, for each day, a (-1,+1) column is added to indicate whether the number of accesses decreases or increases with respect to the previous day. Furthermore, also average and variance of the number of accesses within each week is computed. Consequently, the number of numeric features in the dataset is more than doubled. The introduction of daily trends among the features results in a significant improvement of the predictive power of the classifier which reaches a precision above 0.88 as shown in the row labeled $GBT_{RUT+RBF+DTF}$ of Table 6.

Table 6 Incremental improvements of classifier performance achieved exploiting RUT, RBF, and DTF strategies.

Classifier	**Precision**	**Recall**	**F1**
GBT	0.792	0.770	0.781
GBT_{RUT}	0.820	0.842	0.831
$GBT_{RUT+RBF}$	0.836	0.848	0.842
$GBT_{RUT+RBF+DTF}$	**0.881**	**0.870**	**0.875**

4.4 Site-level models

Since the CMS infrastructure is distributed we assess here the opportunity of training a specific classifier per each site. In fact, some locality could exist in dataset accesses that make some items most frequently accessed from some sites rather than others. Similarly, there may exist certain combinations of Primary Dataset or Era description in the dataset full name (see Fig. 2) that are more frequent for specific sites. Table 7 shows the performance achieved by single-site classifiers learned from the samples referred to 6 specific sites of the CMS infrastructure. These 6 sites were chosen by first grouping the CMS sites in three groups on the basis of the number of weekly samples recorded in the training dataset: more than 30k samples, between 30k and 10k samples, and less than 10k samples. Then, two sites from each one of these groups were randomly chosen. This strategy allows us to have a small set of sites that can be considered representative of the large diversity among the CMS sites.

By looking at the figures reported in Table 7 we can note that the variance in the number of training samples results in different precision and recall values of the resulting local models. Small sites having a low number of samples lead to poor classifier performance. These results do not show a clear advantage of adopting a local model instead of a global one learned on the samples from all the sites.

Table 7 Performance of local, site-level models and the global one.

Site	Samples	Precision	Recall	F1
All Sites	296,360	0.881	0.870	0.875
`hep.wisc.edu`	30,487	0.845	0.843	0.844
`datagrid.cea.fr`	4,955	0.857	0.714	0.779
`unl.edu`	31,021	0.870	0.862	0.866
`in2p3.fr`	27,123	0.968	0.945	0.956
`cr.cnaf.infn.it`	16,394	0.900	0.833	0.865
`pi.infn.it`	4,945	0.901	0.801	0.847

The figures in the table point out that only the local classifier at the `in2p3.fr` site outperforms the global model, with an impressive recall of 0.945 when the local model is scored on the local test set. This is however not due to a better performance of the local model on this site, but rather to its particular (and easier to forecast) distribution of accesses. In fact, on a separate test we also scored the global model on the same local test set and obtained a recall of 0.943, far higher than the average recall of the global model (i.e., 0.870).

4.5 Seen and Unseen datasets

In this section we assess the performance of our prediction model to classify the popularity of new in contrast with previously seen datasets. The traces of dataset accesses available allow us to extract a vast range of usage features convenient to extend the learning power of the predictors for existing datasets. However, datasets never accessed in the past obviously miss these usage features. We are thus interested to understand if our classifier can learn how to use static and physics-domain features only, or, conversely, at what extent the lack of usage features jeopardizes its ability to accurately predict popularity.

The 320k samples represented in our training set refer to about 50k different datasets. In order to answer the above research question we included an additional set of 50k samples, one for each dataset. Each new sample refers to the week before the first access to the specific dataset and is labeled on the basis of the popularity of the dataset in the subsequent week. All usage attributes are set to zero in these samples since no access to the corresponding dataset was actually performed. Another approach in literature addresses the same issue by enriching the set of samples with a number of rows randomly obtained from a global catalogue service [7]. Since this approach may introduce samples also for datasets that will never be accessed, and might hinder the exploitation by the classifier of dataset seasonality, we preferred to adopt a different solution that better model the reality.

Table 8 reports precision, recall and F1 score measured in the classification of new and old datasets. `New` refers to the newly introduced datasets, only described by static and physics-domain features. Conversely, `Old` refers instead to the datasets for which usage features are available, i.e., the dataset samples considered in all the previous experiments (see Table 6). The experimental results confirm the relevance of the usage features in predicting dataset popularity, since all classification measures are higher for the `Old` datasets. Nevertheless, the performance loss in the classification of new datasets is relatively small, hence the static features related to the physics domain encapsulate enough knowledge to be successfully exploited by the classifier.

Table 8 Classification performance on new and old datasets.

Datasets	Precision	Recall	F1
`New`	0.836	0.848	0.842
`Old`	0.881	0.870	0.875

4.6 Model aging and refreshing

The models learned to predict dataset popularity discussed so far are static. They are in fact trained on a static dataset extracted from the CMS access logs recorded in one year and tested on a test set obtained from the accesses recorded in the first weeks of the following years. The drawback of this approach is that access patterns and dataset characteristics could change over time. This change could give rise to an overall aging of the prediction model learned. Model aging is commonly observed in different application scenarios (e.g., [3]). To face up model aging it is necessary to re-train the model as new access patterns acquire popularity. Different strategies to keep the model updated can be adopted. The first decision to make is the minimum number of observations required to train the model. This may be thought of as the initial window. Starting at the beginning of the time series, March 2015, the static model discussed is trained using 297,086 samples throughout end of 2016. The model obtained is then used to predict dataset popularity for the next time step, e.g., the first week of 2017. After this time we have at disposal new information for updating the model, e.g., all the dataset accesses logged during the first week of 2017. This data can be used to refresh the model and obtain more accurate prediction for the following week, e.g., the second week of 2017. Second, we need to decide whether the new model has to be trained on all data available or only on the most recent observations. This determines whether an expanding window, reinforced to include the new weeks, or sliding window, for moving along the time series, is used.

The aging of the static model is demonstrated in the plot reported in Fig. 6, where weekly test sets obtained from the accesses in 2016 and 2017 (61 total weeks) are used to score the classifier trained on all 2015's samples. As we can see from the plot the accuracy tends to decrease as the time gap between the training dataset and the test sets increases.

In order to address such model aging issue, we deployed and assessed two different techniques to update and keep fresh the prediction model:

– A **reinforcement** approach, where the training set used to build the prediction model is weekly extended with new samples from the previous week, and a fresh model trained from the reinforced dataset;
– a **sliding-window** approach, where the training set used to build the prediction model is weekly updated by substituting the samples from the oldest week with the samples of the previous week. Again a new model is trained weekly from this updated dataset that maintains its time-coverage constant over time.

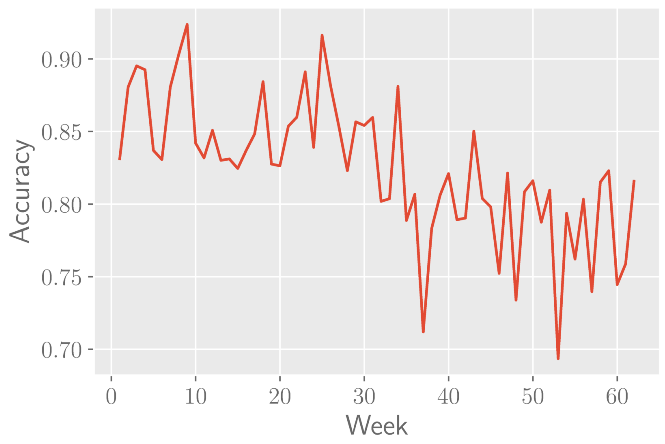

Fig. 6 Demonstration of aging of the static model.

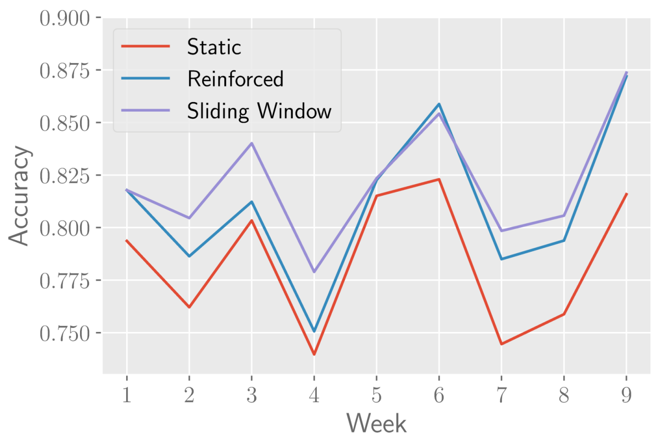

Fig. 7 Comparison between different model update techniques.

The plot in Fig. 7 shows the effectiveness of the two model refreshing techniques. The accuracy of the static and the two model update techniques are compared on the first 9 weeks of year 2016. The static model results to perform worst because of the aging effect. On the other hand, the sliding-window model performs constantly better than the reinforced model. This is likely to be due to the noise introduced in the model by the older samples that, according to the reinforced approach, are always maintained in the dataset. We note however that the aging effect is quite slow to manifest. From Fig. 7 we note a maximal loss in accuracy of about 5% after 9 weeks. The relatively small degradation in such a quite long time period does not justify the adoption of complex streaming approaches to address concept drift in the classification task [1]. Furthermore, similar approaches rely on the continuous update of features that is not possible in our system where the values of dynamic usage features are the result of aggregation jobs performed daily or weekly.

5 Dataset caching

The CMS computing model is based on the Worldwide LHC Computing Grid (WLCG) hierarchical tier structure compounding over hundreds of computing sites. The CERN IT centre represents the WLCG Tier-0 site. 14 worldwide large computing centres with thousands of CPUs, PB of disk storage, tape storage systems and 24/7 Grid support service are referred as Tier-1 centres. Tier-1 centres make data available to hundreds of Tier-2 sites, regional centres for specific analysis tasks.

The CMS sites store datasets locally in order to minimize the network traffic when they are accessed by HEP data-intensive analysis jobs. The current site replacement policy follows the LRU) policy to select which element is evicted when a site is full and a new dataset is needed. LRU exploits the temporal locality of dataset, which we have demonstrated to be characterized by a relatively short access pattern (see Section 4.1).

We aim at enhancing LRU policy with additional knowledge of the dataset access patterns. Therefore we test the Static Dynamic Cache (SDC) [19], which harnesses access frequency, and we also define a novel approach to data caching called Popularity Prediction Cache (PPC), which instead relies on the knowledge of DS popularity in the next week. We demonstrate that PPC offers a big edge over both LRU and SDC.

SDC has proven very effective for caching search engine query results and outperforms LRU. It stores the result of most frequently submitted queries in a static portion of the cache, while the remaining portion of the cache manages query results with the LRU eviction policy. The experiments always start from an initial warm cache with the static portion fixed in size and refreshed periodically. Likewise, we run SDC pre-initializing the static portion of each cache with the datasets most frequently accessed from the starting of the simulation up to the current week. The size of the static portion is 25% of the entire cache. This portion is refreshed every 1,000 misses to compensate the degradation of the hit rate. The refresh operation is considered zero-cost with respect to the misses count.

PPC leverages the outcomes from the popularity classifier. The dataset eviction policy is driven by the popularity predictions. Specifically, when choosing which dataset to evict PPC behaves like a standard LRU acting on unpopular datasets only. In fact, it harnesses the knowledge of dataset popularity in the next week and if a given dataset according to LRU would be selected for eviction but is classified as popular by the prediction model, it is maintained in the cache. Hence, only unpopular least recently used datasets are the candidates to be evicted from the cache. False positive misclassifications may lead to extra disk space overhead, while false negatives may involve longer latencies of user jobs. In terms of numerical examples, 1% in false positive rate on 500 datasets of 2 TB average size would cause 10 TB of extra transfer.

PPC, SDC and LRU performance are assessed by comparing the hit rate for increasing sizes of the caches. The experiment reproduces all dataset access requests throughout 2015 at the 6 most accessed CMS sites. We always perform a cold start of the cache at each site, thus generating a number of compulsory cache misses corresponding to the first reference to each distinct dataset. The analysis of the distributions of popular datasets over time (see Fig. 4) outlines different kinds of locality in the dataset access requests. Namely, some datasets are popular only within relatively short time intervals, or they may become suddenly popular due to proximity to some paper submission deadlines.

Site statistics for the 6 most used CMS Grid sites are detailed in Table 9. The table shows the number of datasets (N), the popular ones (and their percentage), the number of total accesses (in millions), the number of compulsory misses (M_c) and the maximal hit rate ($H_{\max}$), computed as

$$H_{\max} = 1 - \frac{M_c}{N}.$$

Note that $H_{\max}$ represents that best hit rate value that any caching strategy can reach.

Table 9 Statistics of the 6 most accessed sites.

Site	N	Popular (%)	Accesses	$\mathbf{M_c}$	$\mathbf{H}_{\max}$
`datagrid.cea.fr`	3,338	648 (19.4)	1.66 M	1,504	0.55
`desy.de`	3,639	603 (16.6)	0.86 M	1,069	0.71
`fnal.gov`	4,528	1,581 (34.9)	4.72 M	735	0.84
`hep.wisc.edu`	16,131	4,278 (26.5)	77.98 M	5,817	0.64
`jinr-t1.ru`	4,089	798 (19.5)	1.64 M	1,341	0.67
`lnl.infn.it`	5,601	1,034 (18.5)	1.44 M	1,552	0.72

Table 10 Hit rate comparison of dataset caching among LRU, SDC and PPC.

Site	Policy	100	200	400	800	$\mathbf{H}_{\max}$
	LRU	0.26	0.45	0.50	0.52	
`datagrid.cea.fr`	PPC	**0.31**	**0.46**	0.50	0.52	0.55
	SDC	0.30	0.45	**0.52**	**0.54**	
	LRU	0.03	0.29	0.64	**0.70**	
`desy.de`	PPC	**0.12**	**0.36**	**0.65**	**0.70**	0.71
	SDC	0.10	0.33	0.64	**0.70**	
	LRU	0.23	0.51	0.72	**0.84**	
`fnal.gov`	PPC	**0.39**	0.**56**	**0.73**	**0.84**	0.84
	SDC	0.25	0.52	**0.73**	**0.84**	
	LRU	0.00	0.05	0.29	0.45	
`hep.wisc.edu`	PPC	**0.19**	**0.20**	**0.35**	**0.46**	0.64
	SDC	0.04	0.10	0.24	0.45	
	LRU	0.20	0.55	0.64	**0.66**	
`jinr-t1.ru`	PPC	**0.28**	**0.56**	0.64	**0.66**	0.67
	SDC	0.25	0.54	**0.65**	**0.66**	
	LRU	0.11	**0.55**	**0.67**	0.69	
`lnl.infn.it`	PPC	**0.21**	**0.55**	**0.67**	0.69	0.72
	SDC	0.16	0.53	**0.67**	**0.71**	

10 reports the hit rates for different cache sizes and different caching strategies at the 6 most used CMS sites. From the table it is clear that PPC outperforms both LRU and SDC for small cache sizes. For example, for the site with most dataset, e.g., `hep.wisc.edu`, PPC obtains a hit rate of 19% using a very small cache fitting only 100 datasets, versus a 4% hit rate for SDC and a practically 0% hit rate for LRU. Similar benefits, slightly reduced, are obtained with cache sizes of 200 datasets. Hence, PPC is the best policy when the cache size is limited, which makes it very effective in production sites with limited storage or bandwidth. Increasing the cache size makes the PPC strategy less competitive w.r.t. SDC and LRU, mostly because all the strategies perform quite similarly when approaching the maximal hit rate $H_{\max}$.

The performance of the LRU, SDC and PPC caching strategies can be further compared simulating the most efficient caching algorithm, which always evicts the dataset that will not be needed for the longest time in the future. This optimal result is referred to as Bélády's algorithm (OPT) [6]. Since it is generally impossible to predict how far in the future a dataset will be needed, this is not implementable in practice. The OPT hit rate can be calculated only using historical data to compare the effectiveness of the actually chosen cache algorithms with respect to the theoretical optimum. We further compare the performance of the PPC caching strategy with the trained classifier w.r.t. the PPC strategy using the optimal popularity predictors, i.e., the popularity oracle with 100% accuracy (PPC*).

The comparison of results for the `hep.wisc.edu` site is shown in Fig. 8. It shows the hit rate performance of PPC w.r.t. LRU and SDC for small cache sizes reported in Table 10. Moreover the differences between the hit rates of the trained PPC strategy and the oracle PPC* strategy are very small (less than 0.02 for cache size 100 and 0.05 for cache sizes $200-300$), confirming the benefits of the high accuracy predictors in popularity caching.

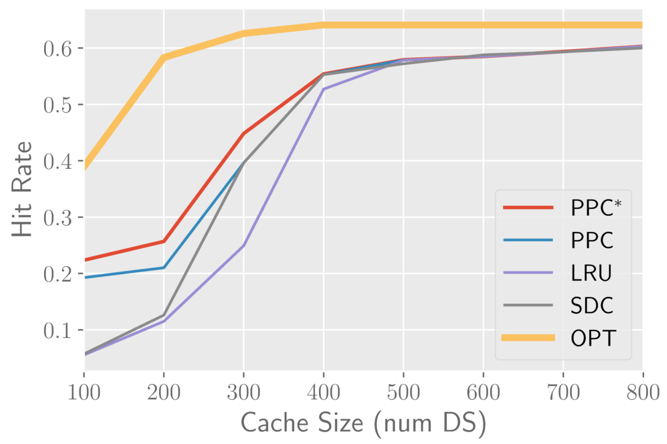

Fig. 8 Simulation on historical data of the caching algorithms, including the theoretical optimal (OPT) and the 100% accurate classifier (PPC*).

After assessing the performance of PPC with respect to LRU and SDC on a large historical collection of dataset accesses, we investigate the performance of an actual deployment of PPC. Hence, we use the previous data to warm up the cache and we evaluate the performance of PPC on a new week of access log data, as shown in Fig. 9. Also with warm caches and actual data, the performance of PPC are close to the oracle PPC* strategy, with even smaller differences for cache size $100 - 400$. We do not report graphs on LRU/SDC performance achieved on other CMS sites since all the experiments conducted confirm the best performance of PPC.

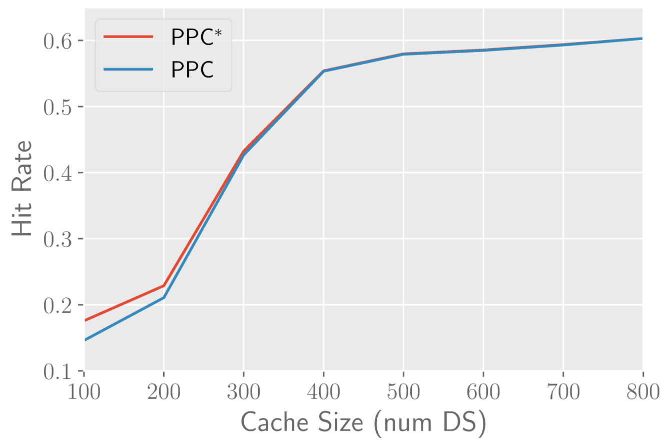

Fig. 9 Hit rate difference between optimal and real classifier is negligible when using cache warm start.

6 Conclusions

This work addresses dataset popularity prediction and dataset caching at the CMS experiment at CERN. It harnesses the historical usage data recorded by the large computing infrastructure in CMS as a rich source of information for system tuning and capacity planning.

We investigate how to leverage scalable machine learning on this huge amount of data. We analyzed the accesses to the CMS datasets recorded by the CMS monitoring system and discover patterns and correlations useful to enhance the overall efficiency of the distributed infrastructure.

In particular, we implement a scalable pipeline of Spark components whose goal is collecting from different sites the dataset access logs, organizing them into weekly snapshots, and training, on these snapshots, predictive models able to accurately forecast which datasets will become popular over time. The F1 measure of the best performing model is 0.875, which indicates an accurate ability to correctly classify popular datasets from unpopular ones.

The accurate predictions computed and kept fresh by the proposed pipeline of scalable Spark components running on the CMS Hadoop cluster are then exploited at each CMS site by a novel data caching policy, called Popularity Prediction Caching (PPC). We evaluate the performance of PPC against popular caching policy baselines like LRU and its variations. The experiments conducted on large traces of real dataset accesses show that PPC outperforms LRU and it increases the number of cache hits up to 20% at some sites. Notably, PPC results in being the best caching policy with limited cache size, which makes it very effective in production sites with limited storage or bandwidth. This result is particularly important for the efficiency of the CMS Grid infrastructure since it allows to deploy an effective data placement policy creating replicas of datasets at the sites where they are most likely to be used. Enhancing the CMS data placement policy involves a significant improvement of resource usage and a consequent reduction of the large cost of this important infrastructure.

In general, caching strategies can be evaluated using different metrics, and hit rate is the most common. However, very few works take into account the cost of misses when evaluating the performance of caching strategies and propose cost-aware caching strategies. As a future work, we will develop network transfer and financial cost models, taking into account the dataset transfer cost as well as the electricity prices when computing the cost, and we will evaluate cost-aware caching eviction policies, which takes into account both transfer time and costs (e.g., energy expenditure to transfer dataset).

7 Acknowledgments

The first author thanks Tommaso Boccali for his help with the references to Physics subjects and the scientific affiliation with INFN and CERN, as well as Luca Menichetti for his valuable assistance with the Hadoop cluster at CERN and the related software frameworks. The authors thank the CMS experiment for the access to the computing resources and the monitoring logs, and the members of the CMS publications committee.

References

1. Abdulsalam, H., Skillicorn, D.B., Martin, P.: Classification Using Streaming Random Forests. IEEE Transactions on Knowledge and Data Engineering **23**(1), 22–36 (2011)
2. Bajaber, F., El Shawi, R., Batarfi, O., Altalhi, A., Barnawi, A., Sakr, S.: Big data 2.0 processing systems: Taxonomy and open challenges. Journal of Grid Computing **14** (2016)
3. Baraglia, R., Castillo, C., Donato, D., Nardini, F.M., Perego, R., Silvestri, F.: Aging effects on query flow graphs for query suggestion. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009, pp. 1947–1950 (2009)
4. Baranowski, Z., Grzybek, M., Canali, L., Garcia, D.L., Surdy, K.: Scale out databases for CERN use cases. Journal of Physics: Conference Series **664**(4), 042,002 (2015)
5. Beermann, T., Maettig, P., Stewart, G., Lassnig, M., Garonne, V., Barisits, M., Vigne, R., Serfon, C., Goossens, L., Nairz, A., Molfetas, A., the Atlas collaboration: Popularity Prediction Tool for ATLAS Distributed Data Management. Journal of Physics: Conference Series **513**(4), 042,004 (2014)
6. Belady, L.A.: A Study of Replacement Algorithms for Virtual-Storage Computer. IBM Systems Journal **5**(2), 78–101 (1966)
7. Bonacorsi, D., Kuznetsov, V., Wildish, T., Giommi, L.: Exploring Patterns and Correlations in CMS Computing Operations Data with Big Data Analytics Techniques. In: Proceedings, International Symposium on Grids and Clouds 2015 (ISGC2015): Taipei, Taiwan, March 15-20, 2015, vol. ISGC2015, p. 008 (2015)

8. Bonchi, F., Giannotti, F., Gozzi, C., Manco, G., Nanni, M., Pedreschi, D., Renso, C., Ruggieri, S.: Web log data warehousing and mining for intelligent web caching. Data Knowl. Eng. **39**(2), 165–189 (2001)
9. Breiman, L.: Random Forests. Machine Learning **45**(1), 5–32 (2001)
10. Brun, R., Rademakers, F.: ROOT: An object oriented data analysis framework. Nucl. Instrum. Meth. **A389**, 81–86 (1997)
11. Caruana, R., Karampatziakis, N., Yessenalina, A.: An empirical evaluation of supervised learning in high dimensions. In: Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008, pp. 96–103 (2008)
12. Čerkasova, L., Laboratories, H.P.: Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy. HP Laboratories technical report. Hewlett-Packard Laboratories (1998)
13. Chatrchyan, S., et al.: The CMS Experiment at the CERN LHC. JINST **3**, S08,004 (2008)
14. Chatrchyan, S., et al.: Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. Phys. Lett. **B716**, 30–61 (2012)
15. Daruru, S., Marin, N.M., Walker, M., Ghosh, J.: Pervasive parallelism in data mining: dataflow solution to co-clustering large and sparse Netflix data. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009, pp. 1115–1124 (2009)
16. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM **51**(1), 107–113 (2008)
17. Dietterich, T.G.: Ensemble methods in machine learning. In: Multiple Classifier Systems, First International Workshop,Cagliari, Italy, June 21-23, 2000, Proceedings, pp. 1–15 (2000)
18. Facca, F.M., Lanzi, P.L.: Mining interesting knowledge from weblogs: a survey. Data Knowl. Eng. **53**(3), 225–241 (2005)
19. Fagni, T., Perego, R., Silvestri, F., Orlando, S.: Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data. ACM Trans. Inf. Syst. **24**(1), 51–78 (2006)
20. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Ann. Statist. **29**(5), 1189–1232 (2001)
21. Huang, Y., Hsu, J.: Mining web logs to improve hit ratios of prefetching and caching. Knowl.-Based Syst. **21**(1), 62–69 (2008)
22. Hushchyn, M., Charpentier, P., Ustyuzhanin, A.: Disk storage management for LHCb based on Data Popularity estimator. CoRR **abs/1510.00132** (2015)
23. Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P.: An Analysis of Traces from a Production MapReduce Cluster. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20 May 2010, Melbourne, Victoria, Australia, pp. 94–103 (2010)
24. Kuznetsov, V., Li, T., Giommi, L., Bonacorsi, D., Wildish, T.: Predicting dataset popularity for the CMS experiment. Journal of Physics: Conference Series **762**(1), 012,048 (2016)
25. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N.: Speeding up Document Ranking with Rank-based Features. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015, pp. 895–898 (2015)
26. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M.J., Zadeh, R., Zaharia, M., Talwalkar, A.: MLlib: Machine Learning in Apache Spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
27. Meoni, M., Boccali, T., Magini, N., Menichetti, L., Giordano, D.: Xrootd popularity on hadoop clusters. Journal of Physics: Conference Series **898**(7), 072,027 (2017)
28. Meoni, M., Kuznetsov, V., Boccali, T., Menichetti, L.M., Rumševičius, J.: Exploiting Apache Spark platform for CMS computing analytics. In: To appear (2017)
29. Oliner, A.J., Ganapathi, A., Xu, W.: Advances and challenges in log analysis. Commun. ACM **55**(2), 55–61 (2012)
30. Qiang, Y., Henry, Z.H.: Web-Log Mining for Predictive Web Caching, Knowledge and Data Engineering. Knowledge and Data Engineering, IEEE Transactions on Knowledge and Data Engineering **39**, 1050–1053 (2003)
31. Ranganathan, K., Foster, I.: Simulation studies of computation and data scheduling algorithms for data grids. Journal of Grid Computing **1**, 53–62 (2003)

32. Shamsi, J., Khojaye, M.A., Qasmi, M.A.: Data-intensive cloud computing: Requirements, expectations, challenges, and solutions. Journal of Grid Computing **11**(2), 281–310 (2013). DOI 10.1007/s10723-013-9255-6
33. Shiers, J.: The Worldwide LHC Computing Grid (worldwide LCG). Computer Physics Communications **177**(1-2), 219–223 (2007)
34. Songwattana, A.: Mining Web Logs for Prediction in Prefetching and Caching. 2008 Third International Conference on Convergence and Hybrid Information Technology (ICCIT) **02**, 1006–1011 (2008)
35. Zhu, X., Davidson, I.: Knowledge Discovery and Data Mining: Challenges and Realities. IGI Global, Hershey, PA, USA (2007)