MLbase: A Distributed Machine Learning Wrapper

Ameet Talwalkar[a] **Tim Kraska**[a] **Rean Griffith**[b] **John Duchi**[a] **Joseph Gonzalez**[a]
Denny Britz[a] **Xinghao Pan**[a] **Virginia Smith**[a] **Evan Sparks**[a] **Andre Wibisono**[a]
Michael J. Franklin[a] **Michael I. Jordan**[a]

[a] AMPLab, UC Berkeley [b] VMware

Abstract

Machine learning (ML) and statistical techniques are key to transforming big data into actionable knowledge. In spite of the modern primacy of data, the complexity of existing ML algorithms is often overwhelming—many users do not understand the trade-offs and challenges of parameterizing and choosing between different learning techniques. Furthermore, existing scalable systems that support machine learning are typically not accessible to ML researchers without a strong background in distributed systems and low-level primitives. In this work, we present our vision for MLbase, a novel system harnessing the power of machine learning for both end-users and ML researchers. MLbase provides (1) a simple declarative way to specify ML tasks, (2) a novel optimizer to select and dynamically adapt the choice of learning algorithm, (3) a set of high-level operators to enable ML researchers to scalably implement a wide range of ML methods without deep systems knowledge, and (4) a new run-time optimized for the data-access patterns of these high-level operators.

1 Introduction

Mobile sensors, social media services, genomic sequencing, and astronomy are among a multitude of applications that have generated an explosion of abundant data. Data is no longer confined to just a handful of academic researchers or large internet companies. Extracting value from such Big Data is a growing concern, and machine learning techniques enable users to extract underlying structure and make predictions from large datasets. In spite of this, even within statistical machine learning, an understanding of computational techniques for algorithm selection and application is only beginning to appear [6]. The complexity of existing algorithms is (understandably) overwhelming to layman users, who may not understand the trade-offs, parameterization, and scaling necessary to get good performance from a learning algorithm. Perhaps more importantly, existing systems provide little or no help for applying machine learning on Big Data. Many systems, such as standard databases and Hadoop, are not designed for the access patterns of machine learning, which makes them inaccessible to machine learning researchers without a strong background in distributed systems and forces developers to build ad-hoc solutions to extract and analyze data with third party tools.

With MLbase,[1] we aim to (1) make machine learning accessible to a broad audience of users and applicable to various data corpora, ranging from small to very large data sets, and (2) provide a platform from machine learning researchers to develop scalable implementations of state-of-the-art methods while abstracting away the low-level systems details. To achieve these two goals, we propose here our vision for MLbase along the lines of a relational database system (RDBM), with four major foci. First, MLbase encompasses a new Pig Latin-like [18] declarative language to specify machine learning tasks. Second, MLbase uses a novel optimizer to select machine learning algorithms—rather than relational operators as in a standard DBMS—where we leverage best practices in ML and build a sophisticated cost-based model. Although MLbase cannot optimally support every machine learning scenario, it provides reasonable performance for a broad range of use cases. This is

[1] Parts of this submission overlap with work detailed in [14].

similar to traditional DBMSs: highly optimized C++ solutions are stronger, but the DBMS achieves good performance with significantly lower development time and expert knowledge [22]. Third, we aim to provide answers early and improve them in the background, continuously refining the model and re-optimizing the plan. Fourth, we identify a set of high-level ML operators that will enable ML researchers to scalably implement a wide-range of ML methods without deep systems knowledge and we design a distributed run-time optimized for the data-access patterns of these high-level ML operators.

2 Use Cases

MLbase will ultimately provide functionality to end users for a wide variety of common machine learning tasks: classification, regression, collaborative filtering, and more general exploratory data analysis techniques such as dimensionality reduction, feature selection, and data visualization. Moreover, MLbase provides a natural platform for ML researchers to develop novel methods to these tasks. We now illustrate a few of the many use cases that MLbase will provide and along the way, we describe MLbase's declarative language for tackling these problems.

2.1 ALS Prediction

Amyotrophic Lateral Sclerosis (ALS), commonly known as Lou Gehrig's disease, is a progressive fatal neurodegenerative illness. The ALS Prediction Prize [4] challenges participants to develop a binary classifier to predict whether an ALS patient will display delayed disease progression. MLbase on its own will not allow one to win the ALS prize, but it can help a user get a first impression of standard classifiers performance. Consider the following example "query," which trains a classifier on the ALS dataset:

```
var X = load("als_clinical", 2 to 10)
var y = load("als_clinical", 1)
var (fn-model, summary) = doClassify(X, y)
```

The user defines two variables: `X` for the data (the independent features/variables stored in columns 2 to 10 in the dataset) and `y` for the labels (stored in the first column) to be predicted via `X`. The MLbase `doClassify()` function declares that the user wants a classification model. The result of the expression is a trained model, `fn-model`, as well as a model `summary`, describing key characteristics of the model itself, such as its quality assessment and the model's lineage (see Section 3). The language hides two key issues from the user: (i) which algorithms and parameters the system should use and (ii) how the system should test the model or distribute computation across machines. Indeed, it is the responsibility of MLbase to find, train and test the model, returning a trained classifier function as well as a summary about its performance to the user.

2.2 Music Recommendation

The Million Song Dataset Challenge [5] is to predict the listening behavior of a set of 110,000 music listeners based on partial listening histories. We can tackle this collaborative filtering task with MLbase's `doCollabFilter` expression as follows:

```
var X = load("user_song_pairs", 1 to 2)
var y = load("user_ratings", 1)
var (fn-model, summary) = doCollabFilter(X, y)
```

The semantics of `doCollabFilter` are identical to `doClassify`, with the obvious distinction of returning a model `fn-model` to predict song ratings.

2.3 Twitter Analysis

Equipped with snapshots of the Twitter network and associated tweets [15, 20], one may wish to perform a variety of unsupervised exploratory analyses to better understand the data. For instance, advertisers may want to find features that best describe "hubs," people with the most followers or

the most retweeted tweets. MLbase provides facilities for graph-structured data, and finding relevant features can be expressed as follows:

```
var G = loadGraph("twitter_network")
var hubs-nodes = findTopKDegreeNodes(G, k = 1000)
var T = textFeaturize(load("twitter_tweet_data"))
var T-hub = join(hub-nodes, "u-id", T, "u-id")
findTopFeatures(T-hub)
```

In this example, the user first loads the twitter graph and applies the `findTopKDegreeNodes()` function to determine the hubs. Afterwards, the tweets are loaded and featurized with `textFeaturize()`. During this process, every word in a tweet, after stemming, becomes a feature. The result of the featurization as well as the pre-determined hubs are joined together on the user-id, `u-id`, and finally, `findTopFeatures` finds the distinguishing features of the hubs.

2.4 MLbase Extensibility

A key aspect of MLbase is its extensibility to novel ML algorithms. We envision ML experts adding new ML techniques to the system, leveraging MLbase's high-level primitives to aid development. High-level primitives include efficient implementations of gradient and stochastic gradient descent, mini-batch extensions of map-reduce that naturally support divide-and-conquer approaches such as [17], and graph-parallel primitives as in GraphLab [16]. We have already mapped several algorithms, including k-means clustering, LogitBoost [12], various matrix factorization tasks (as described in [17]), and support vector machines (SVM) [10] to these primitives.

Moreover, MLbase requires that developers describe the properties of their new algorithms using a special contract (as shown in the left part of Figure 1 in the Appendix). The contract specifies the type of algorithm (e.g., binary classification), the algorithm's parameters, run-time complexity (e.g., O(n)) and possible run-time optimizations (e.g., synchronous vs. asynchronous learning; see Section 3.3). The easy extensibility of MLbase simultaneously makes MLbase an attractive platform for ML experts and allow users to benefit from recent developments in statistical machine learning.

3 Architecture

A user issues requests using the MLbase declarative task language to the MLbase master. Similar to RDBMs, the system parses the request into a *logical learning plan* (LLP), which describes the most general workflow to perform the ML task. The search space for the LLP consists of the combinations of ML algorithms, featurization techniques, algorithm parameters, and data sub-sampling strategies (among others), and is too huge to be explored entirely. Therefore, an optimizer tries to prune the search-space of the LLP to find a strategy that is testable in a reasonable time-frame. Although the optimization process is significantly harder than in relational database systems, we can leverage many existing techniques. For example, the optimizer can consider the current data layout, materialized intermediate results (pre-processed data) as well as general statistics about the data to estimate the model learning time. However, in contrast to DBMSs, the optimizer needs also to estimate the expected quality for each of the model configurations to focus on the most promising candidates.

After constructing the optimized logical plan, MLbase transforms it into a *physical learning plan* (PLP). A PLP consists of a set of executable ML operations, such as filtering and scaling feature values, as well as synchronous and asynchronous MapReduce-like operations. In contrast to an LLP, a PLP specifies exactly the parameters to be tested as well as the data (sub)sets to be used. The MLbase master distributes these operations onto the worker nodes, which execute them through the MLbase runtime. The result of the execution is typically a learned model (`fn-model`) or some other representation (e.g., relevant features) that the user may use to make predictions or summarize data. MLbase also returns a summary of the quality assessment of the model and the learning process (the model's lineage) to allow the user to make more informed decisions. Moreover, in contrast to traditional database systems, the task here is not necessarily complete upon return of the first result. Instead, we envision that MLbase will further improve the model in the background via additional exploration. Figure 1 of the Appendix shows the general architecture of MLbase.

3.1 Logical Learning Plan

The first step of optimizing the declarative ML task into our machine-executable language is the translation into a logical learning plan. During this translation many operations are mapped 1-to-1 to LLP operators (e.g., data loading), whereas ML functions are expanded to their best-practice workflows. For instance, in the case of binary SVM classification [10], the parameters MLbase selects may include the size n of the training dataset, the type of kernel to use, kernel parameters (the scale parameter in the case of RBF kernel), regularization values, and whether to process the data vectors x so that their entries lie in particular ranges or are binned into similar groups. The LLP, as visualized in Figure 2 in the Appendix, specifies the combinations of parameters, algorithms, and data subsampling the system must evaluate and cross-validate to test quality. After exploration, the best model is selected, potentially trained using a larger dataset, and sanity-checked using common baseline (for classification, this may be predicting the most common class label).

3.2 Optimization

The optimizer actually transforms the LLP into an optimized plan—with concrete parameters and data subsampling strategies—that can be executed on our run-time. To meet time constraints, the optimizer estimates execution time and algorithm performance (i.e., quality) based on statistical models, also taking advantage of pruning heuristics and newly developed online model selection tools [6]. As a simple example, normalizing features to lie in $[-1, 1]$ often yields performance improvements for SVM classifiers, so applying such normalization before attempting more complicated techniques may be useful for meeting time constraints. As another example, standard AdaBoost algorithms, while excellent for choosing features, may be non-robust to data outliers; a dataset known to contain outliers may render training a classifier using AdaBoost moot. The general accuracy of algorithms is just one of the aspects an optimizer may take into account. Statistics about the dataset itself, different data layouts, algorithm speed and parallel execution strategies (as described in the next section) are just a few additional dimensions the optimizer may exploit to improve the learning process. Figure 2 shows an example optimized plan in step (3).

3.3 Runtime

MLbase's run-time supports a simple set of data-centric primitives for machine learning tasks. The physical learning plan (PLP) composes these primitives together to build potentially complex workflows. The master's responsibility is to distribute these primitives to the workers for their execution, to monitor progress, and take appropriate actions in the case of a node failure. At its core, the runtime primitives can be split into two categories: classical relational operators and special higher-level primitives for machine learning. Whereas the first type make it easy to embed machine-learning algorithm in complex data workflows, the latter allows experts to easily develop distributed versions of algorithms and includes graph-based operators as well as a novel mini-batch map reduce primitive. Each implemented algorithm also has a contract with the runtime environment, which specifies computational guarantees and whether (and which) consistency properties the runtime may relax.

For instance, in the context of gradient descent, we note that the optimizer may take advantage of properties of statistical learning algorithms. Gradient-descent algorithms are robust: they can tolerate noise in gradient estimates, node failures, and even receiving stale (computed out of order) gradient information while providing statistical guarantees [7]. Thus, the runtime contract for a gradient descent update function may specify that asynchrony and (heavy) subsampling are acceptable. This statistical freedom and robustness allows reduced consistency, so the system can forego expensive failure recovery techniques and—in cases such as these—avoid using techniques to deal with straggler nodes.

MLbase's runtime makes it possible to explore these advanced characteristics of ML algorithms in a systematic fashion; moreover, it gives layman users the tools to do so. Of course, not every algorithm can take full advantage of these optimizations, e.g., some algorithms are inherently sequential and require greater consistency while others may not fit the supported MLbase high-level primitives. Nonetheless, in these cases the ML developer has the freedom to use common MapReduce operations and restrict the applicable optimizations in the ML contract. This yields an extensible system that is easily updated with new machine learning techniques while remaining quite usable.

4 Related Work

MLbase is not the first system trying to make machine learning more accessible, but it is novel in that it frees users from algorithm choices, automatically optimizes for distributed execution and provides a platform for ML researchers to implement scalable algorithms without deep systems knowledge. The closest related systems include Weka [3], MADLib [13], Mahout [2] and ScalOps [8]. Weka is a single node system, while MADLib, Mahout and ScalOps are all lower-level systems requiring strong systems knowledge to use and thus do not serve as an accessible platform for most ML researchers. None of these systems addresses the (difficult but necessary) challenge of selecting and configuring learning algorithms.

Other related systems also exist. Google Predict [1] is Google's proprietary web-service for prediction problems, but restricts the maximum training data-size to 250MB. There have been other efforts to build distributed run-times for more advanced analytical tasks. Hyracks [9] and Spark [21] both have special iterative in-memory operations to better support ML algorithms. In contrast to MLbase, however, they do not have learning-specific optimizers, nor do they take full advantage of the characteristics of ML algorithms (e.g., specification of contracts allowing relaxed consistency). Finally, in [11] the authors show how many ML algorithms can be expressed as a relational-friendly convex-optimization problem, whereas the authors of [19] present techniques to optimize inference algorithms in a probabilistic DBMS. We leverage these techniques in our run-time, but our system aims beyond a single machine and extends the presented optimization techniques.

5 Conclusion

We described MLbase, a system aiming to make ML more accessible to non-experts and to provide an accessible platform for ML researches to scalably implement a wide range of learning methods without a deep systems knowledge. We have presented our long-term vision for MLbase, and we are currently in the process of building the entire system.

References

[1] Google Prediction API. https://developers.google.com/prediction/.

[2] Mahout. http://mahout.apache.org/.

[3] Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[4] ALS Prediction Prize. http://www.prize4life.org/page/prizes/predictionprize, 2012.

[5] Million Song Dataset Challenge. http://www.kaggle.com/c/msdchallenge, 2012.

[6] A. Agarwal, P. Bartlett, and J. Duchi. Oracle inequalities for computationally adaptive model selection. In *Conference on Learning Theory*, 2011.

[7] A. Agarwal and J. Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems 25*, 2011.

[8] V. Borkar et al. Declarative systems for large-scale machine learning. *Bulletin of the Technical Committee on Data Engineering*, 35(2):24, June 2012.

[9] V. R. Borkar et al. Hyracks: A flexible and extensible foundation for data-intensive computing. In *ICDE*, 2011.

[10] C. Cortes and V. N. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[11] X. Feng et al. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*, 2012.

[12] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998.

[13] J. M. Hellerstein et al. The madlib analytics library or mad skills, the sql. In *PVLDB*.

[14] T. Kraska et al. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[15] H. Kwak et al. What is twitter, a social network or a news media? In *WWW*, 2010.

[16] Y. Low et al. Graphlab: A new framework for parallel machine learning. In *UAI*, 2010.

[17] L. Mackey, A. Talwalkar, and M. I. Jordan. Divide-and-conquer matrix factorization. In *NIPS*, 2011.

[18] C. Olston et al. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, 2008.

[19] D. Z. Wang et al. Hybrid in-database inference for declarative information extraction. In *SIGMOD*, 2011.

[20] J. Yang and J. Leskovec. Temporal variation in online media. In *WSDM*, 2011.

[21] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[22] M. Zukowski et al. Monetdb/x100 - a dbms in the cpu cache. *IEEE Data Eng. Bull.*, 28(2), 2005.

A Supplemental Figures

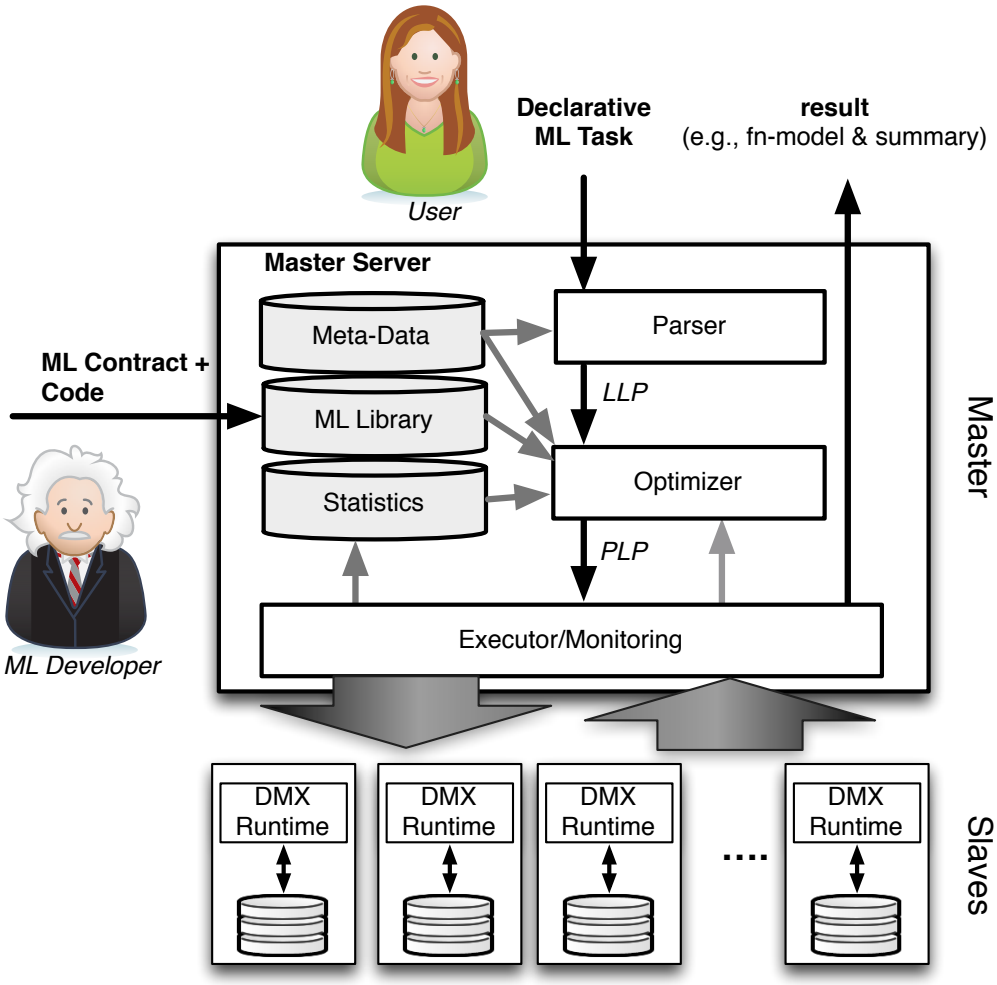

Figure 1: MLbase Architecture

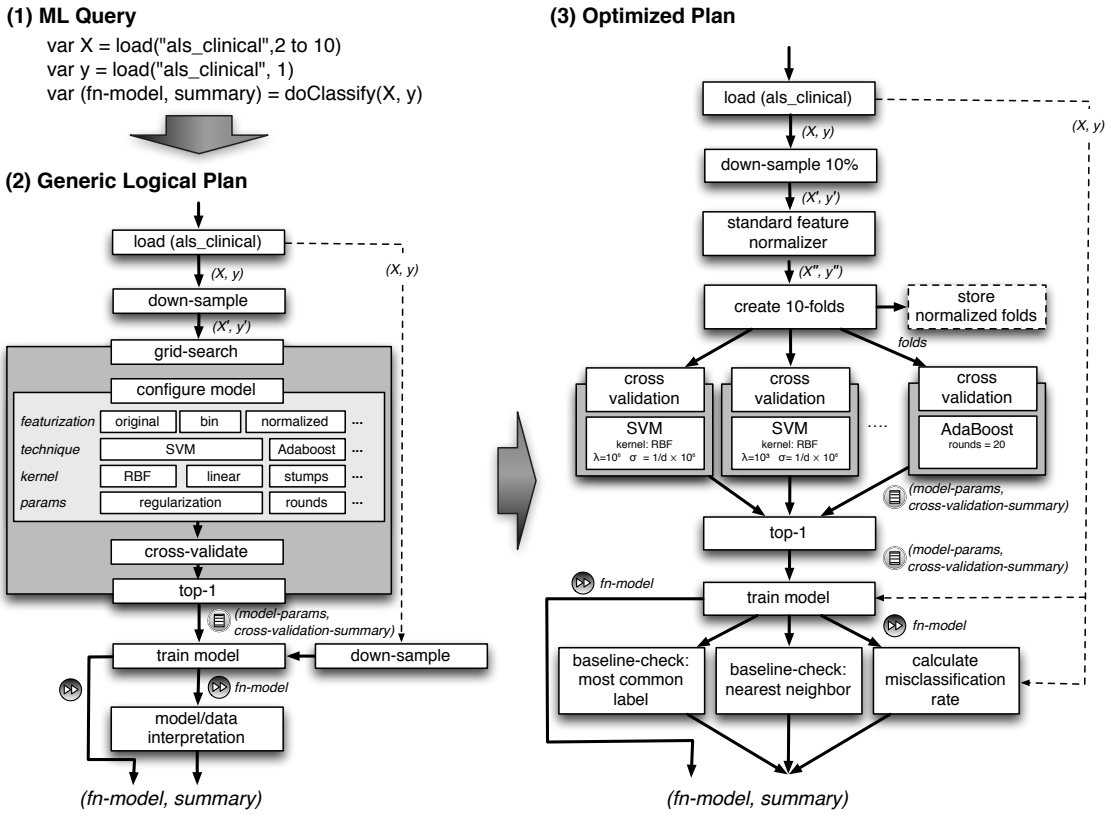

Figure 2: Optimization Process