MLbase: A Distributed Machine-learning System

Tim Kraska
Brown University
kraskat@cs.brown.edu

Ameet Talwalkar
AMPLab, UC Berkeley
ameet@cs.berkeley.edu

John Duchi
AMPLab, UC Berkeley
jduchi@eecs.berkeley.edu

Rean Griffith
VMware
rean@vmware.com

Michael J. Franklin
AMPLab, UC Berkeley
franklin@cs.berkeley.edu

Michael Jordan
AMPLab, UC Berkeley
jordan@cs.berkeley.edu

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 [7]. 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 forces developers to build ad-hoc solutions to extract and analyze data with third party tools.

With *MLbase* we aim to make machine learning accessible to a broad audience of users and applicable to various data corpora, ranging from small to very large data sets. To achieve this goal, we provide here a design for MLbase along the lines of a database system, with four major foci. First, MLbase encompasses a new Pig Latin-like [22] declarative language to specify machine learning tasks. Although MLbase cannot optimally support every machine learning scenario, it provides reasonable performance for a broad range of use cases. This is similar to a traditional DBMS: highly optimized C++ solutions are stronger, but the DBMS achieves good performance with significantly lower development time and expert knowledge [28]. 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. Third, we aim to provide answers early and improve them in the background, continuously refining the model and re-optimizing the plan. Fourth, we design a distributed run-time optimized for the data-access patterns of machine learning.

In the remainder, we outline our main contributions:

- We describe a set of typical MLbase use cases, and we sketch how to express them in our high-level language
- We show the optimization pipeline and describe the optimizer's model selection, parameter selection, and validation strategies
- We sketch how the optimizer iteratively improves the model in the background
- We describe the MLbase run-time and show how it differs from traditional database pipelines.

6th Biennial Conference on Innovative Data Systems Research (CIDR'13)
January 6-9, 2013, Asilomar, California, USA.

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. Although most patients suffer from a rapidly progressing disease course, some patients (Stephen Hawking, for example) display delayed disease progression. Leveraging the largest database of clinical data for ALS patients ever created, the ALS Prediction Prize [5] 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.MLbase

2.2 Music Recommendation

The Million Song Dataset Challenge [6] is to predict the listening behavior of a set of 110,000 music listeners (i.e., the test-set), deciding which songs of the Million Song Dataset [10] they will listen to based on partial listening history and the listening history of 1M other users (i.e., the training-set). This is an exemplar of collaborative filtering (or noisy matrix completion) problem. Specifically, we receive an incomplete observation of a ratings matrix, with columns corresponding to users and rows corresponding to songs, and we aim to infer the unobserved entries of this ratings matrix. Under this interpretation, 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 [19, 26], 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 ML Research Platform

A key aspect of MLbase is its extensibility to novel ML algorithms. We envision ML experts using MLbase as a platform to experiment with new ML algorithms. MLbase has the advantage that it offers a set of high-level primitives to simplify building distributed machine learning algorithms without knowing the details about data partitioning, message passing, or load balancing. These primitives currently include efficient implementations of gradient and stochastic gradient descent, mini-batch extensions of map-reduce that naturally support divide-and-conquer approaches such as [21], and graph-parallel primitives as in GraphLab [20]. We have already mapped several algorithms, including k-means clustering, LogitBoost [16], various matrix factorization tasks (as described in [21]), and support vector machines (SVM) [13] to these primitives.

Furthermore, we allow the ML expert to inspect the execution plan using a database-like `explain` function and steer the optimizer using hints, making it an ideal platform to easily setup experiments. For example, the hints allow an expert to fix the algorithms, the parameter ranges and/or force a full grid-search in order to generate parameter sensitivity analysis. Even though in this setting, the ML expert does not use the automatic ML algorithm nor the parameter selection, he would still benefit from the run-time optimization.

3. ARCHITECTURE

Figure 1 shows the general architecture of MLbase, which consists of a master and several worker nodes. A user issues requests using the MLbase declarative task language to the MLbase master. 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

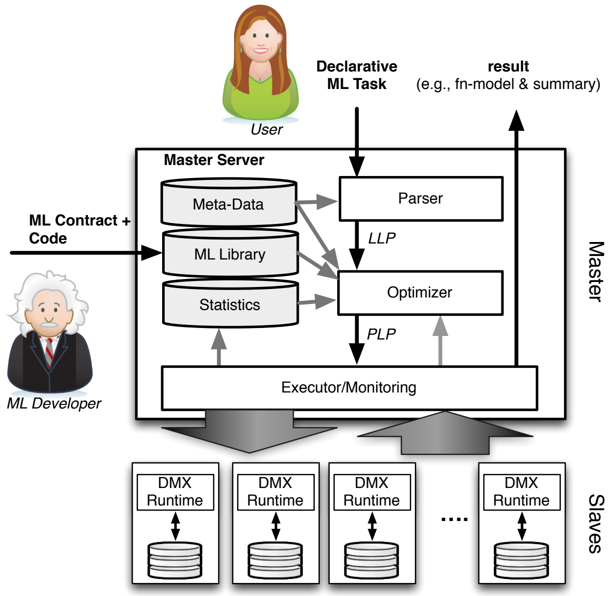


Figure 1: MLbase Architecture
MLbase

the LLP consists of the combinations of ML algorithms, featurization techniques, algorithm parameters, and data subsampling 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 a DBMS, the optimizer also needs 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) to be executed. 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—as in the examples of the previous section—is typically a learned model (`fn-model`) or some other representation (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. In the prototype we have built, we return the learned model as a higher-order function that can be immediately used as a predictive model on new data.[1]

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. The first search therefore stores intermediate steps, including models trained on subsets of data or processed feature values, and maintains statistics on the underlying data and learning algorithms' performance. MLbase may then later re-issue a better optimized plan to the execution module to improve the results the user receives.

This continuous refinement of the model in the background has several advantages. First, the system becomes more interactive, by letting the user experiment with an initial model early on. Second, it makes it very easy to create progress bars, which allow the user to decide on the fly when the quality is sufficient to use the model. Third, it reduces the risk of stopping too early. For example, the user might find, that in the first 10 minutes, the system was not able to create a model with sufficient quality and he is now considering other options. However, instead of letting the system remain idle until the user issues the next request, MLbase continues searching and testing models in the background. If it finds a model with better quality, it informs the user about it. Finally, it is very natural for production systems to continuously improve models with new data. MLbase is designed from the beginning with this use case in mind by making new data one of the dimensions for improving a model in the background.

Another key aspect of MLbase is its extensibility to novel ML algorithms. We envision ML experts constantly adding new ML techniques to the system, with the requirement that developers implement new algorithms in MLbase primitives and describe their properties using a special contract (see the left part of Figure 1). 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 5). The easy extensibility of MLbase will simultaneously make it an attractive platform for ML experts and allow users to benefit from recent developments in statistical machine learning.

4. QUERY OPTIMIZATION

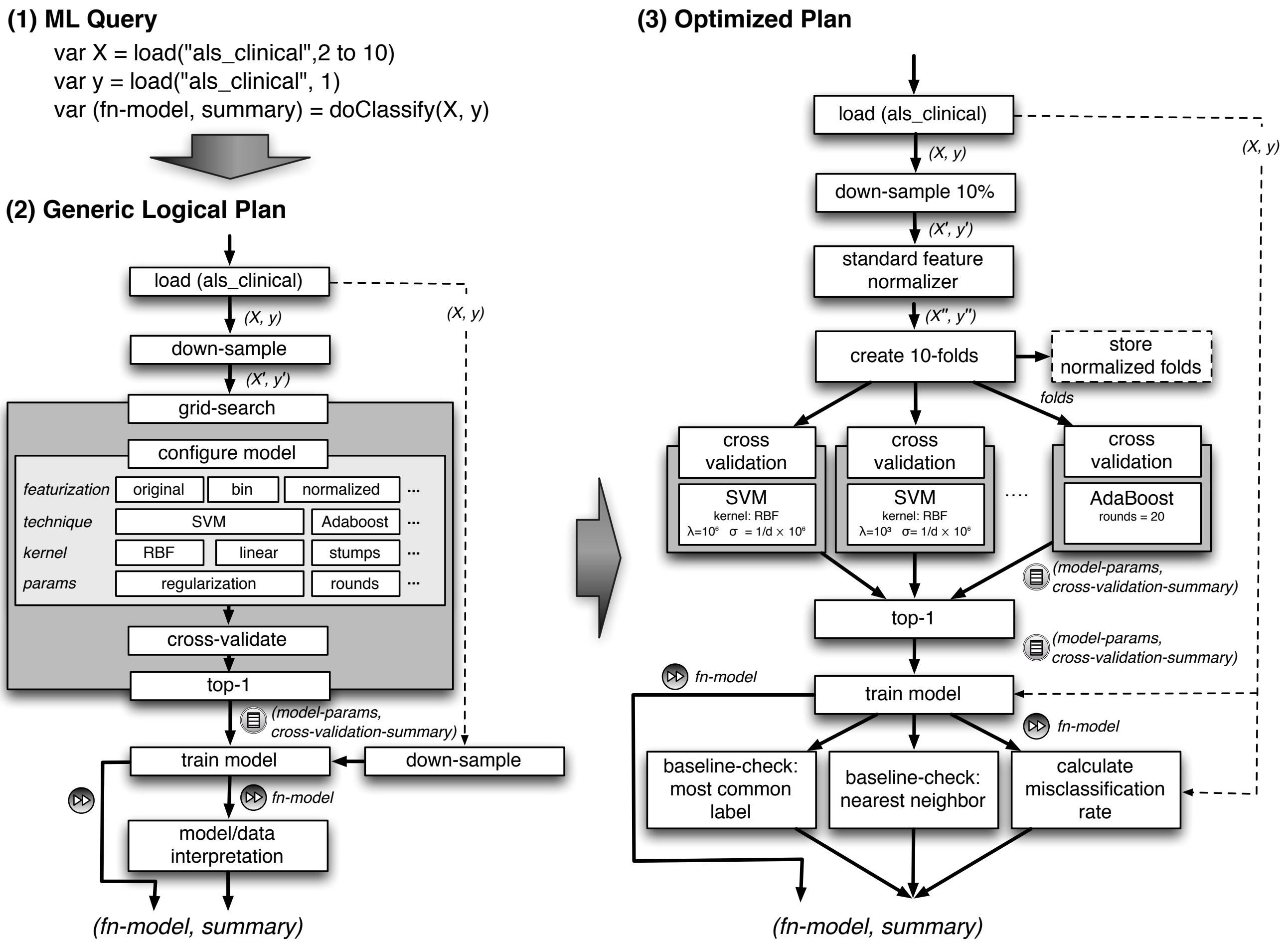
Having described our architecture, we now turn to a deeper description of our query optimization techniques and ideas. Similar to approaches in traditional database systems, we transform the declarative ML task into a logical plan, optimize it, and finally translate it into a physical plan; we describe each of these three below.

4.1 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.

In what follows, we use binary support vector machine (SVM) classification (see, e.g., [24]) as our running example throughout. An SVM classifier is based on a kernel function K, where $\mathsf{K}(x, x')$ is a particular type of similarity measure between data points x, x'. Given a dataset $\{x_1, \ldots, x_n\}$, the

[1] We use the Scala language, which makes it easy to return and serialize functions.



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

Figure 2: Optimization Process

goal is to learn a classifier for negative and positive examples

$$f(x) = \operatorname{sign}\left\{\sum_{i=1}^{n} \alpha_i \mathsf{K}(x, x_i) + b\right\},$$

and finding f requires solving the numerical problem

$$\min_{\alpha, b}.\ \frac{1}{n}\sum_{i=1}^{n} \max\left\{1 - y_i\Big(\sum_{j=1}^{n} \alpha_j \mathsf{K}(x_i, x_j) + b\Big), 0\right\} + \frac{\lambda}{2}\alpha^\top K\alpha,$$

where $K = [\mathsf{K}(x_i, x_j)]_{i,j=1}^n$ is known as the Gram matrix and $\lambda \geq 0$ is a regularization parameter. Examples of kernel functions include linear kernels $\mathsf{K}(x, x') = x^\top x'$ and the RBF kernel $\mathsf{K}(x, x') = \exp(-\|x - x'\|^2/2\sigma^2)$. The parameters MLbase selects may include the size n of the training dataset, the type of kernel to use, kernel parameters (σ 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.

In Figure 2, we provide a visualization of MLbase plan expansion for the declarative ALS prediction task of Section 2.1. According to best-practice the general plan assumes, that the data is down-sampled to speed-up the training and validation process. As part of the search for a model, MLbase evaluates several learning algorithms applicable to this specific ML task (in classification, these may include SVMs (above) or AdaBoost [15]). Evaluation includes finding appropriate parameters (λ, σ, and featurization of x in the SVM case). The LLP 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 baselines (for classification, this may be predicting the most common class label).

4.2 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 [7]. As an example, it is well-known that normalizing features to lie in $[-1, 1]$ 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.

Figure 2 shows an example optimized plan in step (3). In this example, the optimizer uses standard feature normalization and subsamples the data at a 10% rate. Furthermore, the optimizer uses the best practice of 10-fold cross-validation— equally splitting the data in 10 partitions and setting one partition aside for evaluation in each of ten experiments—while the final model is trained using the full data set (X, y). The runtime evaluates the model using the misclassification rate and against nearest-neighbor and most-common label baselines.

Note, that many of the discussed optimization techniques also apply to unsupervised learning even though it does not allow for automatically evaluating the result with respect to quality (e.g., done through the 10-fold cross validation in Figure 2). For example, for clustering, very basic statistics about the data can help to determine a good initial number of seed clusters. Furthermore, the optimizer can consider building multi-dimensional index structures and/or pre-compute distance matrices to speed up the cluster algorithm.

MLbase allows user-specified hints that can influence the optimizer, which is similar to user influence in database systems. Experts may also modify training algorithms, in essence, the runtime itself, if they desire more control. These hints, which may include recommended algorithms or featurization strategies, makes MLbase a powerful tool even for ML experts.

4.3 Optimizer examples

To demonstrate the advantages of an optimizer for selecting among different ML algorithms, we implemented a prototype using two algorithms: SVM and AdaBoost. For both algorithms, we used publicly available implementations: LIBSVM [12] for SVM and the ML AdaBoost Toolbox [1] for AdaBoost. We evaluated the optimizer for a classification task similar to the one in Figure 2 with 6 datasets from the LIBSVM website: 'a1a', 'australian', 'breast-cancer', 'diabetes', 'fourclass', and 'splice'. To better visualize the impact of finding the best ML model, we performed a full grid search over a fixed set of algorithm parameters, i.e., number of rounds (r) for AdaBoost and regularization (λ) and RBF scale (σ) parameters for SVM. Specifically, we tested $r = \{25, 50, 100, 200\}$, $\lambda = \{10^{-6}, 10^{-3}, 1, 10^3, 10^6\}$, and $\sigma = \frac{1}{d} \times \{10^{-6}, 10^{-3}, 1, 10^3, 10^6\}$, where d is the number of features in the dataset. For each algorithm, set of features and parameter settings, we performed 5-fold cross validation, and report the average results across the held-out fold.

Table 3 shows the best accuracy after tuning the parameters using grid search for the different datasets and algorithms, with and without scaling the features (the best combination is marked in bold). The results show first that there is no clear winning combination for all datasets. Sometimes AdaBoost outperforms SVM, sometimes scaling the features helps, sometimes it does not.

Now we turn to understanding the search problem for parameters itself, depicted in Figures 4 and 5. Figure 4 shows, for fixed regularization λ the impact of the σ parameter in the RBF kernel on the accuracy, whereas Figure 5 visualizes the accuracy for varying the number of rounds r for AdaBoost. As shown in Figure 4, the choice of σ in the SVM problem clearly has a huge impact on quality; automatically selecting σ is important. On the other hand, for the same datasets, it appears that the number of rounds in AdaBoost is not quite as significant once $r \geq 25$ (shown in Figure 5). Hence, an optimizer might decide to use AdaBoost first without scaling and a fixed round parameter to provide the user quickly with a first classifier. Afterwards, the system might explore SVMs with scaled features to improve the model, before extending the search space to the remaining combinations.

	SVM		AdaBoost
	original	scaled	
a1a	**82.93**	**82.93**	82.87
australian	85.22	85.51	**86.23**
breast	70.13	**97.22**	96.48
diabetes	76.44	**77.61**	76.17
fourclass	**100.00**	99.77	91.19
splice	88.00	87.60	**91.20**

Figure 3: Classifier accuracy using SVM with an RBF kernel and using AdaBoost

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.

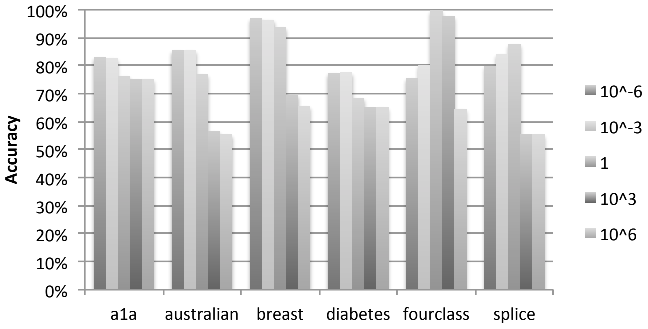


Figure 4: Impact of different $\sigma = \frac{1}{d} \times \{10^{-6}, 10^{-3}, 1, 10^3, 10^6\}$ on the SVM accuracy with an RBF kernel and $\lambda = 10^{-6}$ on LIBSVM data-sets

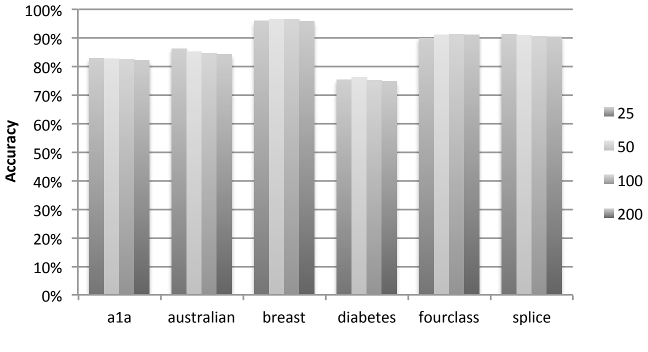


Figure 5: Impact of $r = \{25, 50, 100, 200\}$ on AdaBoost on LIBSVM data-sets

5. 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. Due to lack of space, we only outline the basic techniques of our run-time.

At its core, the run-time supports the main relational operators, predicate filters, projects, joins and simple transformations by applying a higher-order function (similar to a *map* in the map-reduce paradigm). However, machine learning algorithms often use special templates, that power-users (ML experts) may implement for specific algorithms. 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.

As a working example, consider gradient descent. Gradient descent algorithms broadly require two methods: a gradient computation and an update function. The gradient computation G simply takes a datum x and current parameters θ of a model, computing a gradient $G(x, \theta)$ of the objective. The update function U maps current parameters to new parameters using a computed gradient, yielding the following pattern:

```
while Not($condition\ for\ completeness$) do
    $\theta = U\left(\theta, \frac{1}{|X|}\sum_{x \in X} G(x, \theta)\right)$
end while
```

During execution, the gradient function G is invoked for every datum x of the dataset X at the parameters θ. The update function U modifies the current parameters θ based on the average of all the computed gradients to form a new set of parameters, and the process repeats until a MLbase-defined termination condition holds.

Patterns such as these leave the optimizer significant freedom in its specification of run-time behavior and constructs. As a simple example, the system may use different termination conditions or data distribution strategies. Focusing more specifically on our gradient-based learning example, 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 [8]. 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.

Perhaps surprisingly, relaxing consistency can in some cases improve the convergence rate and result in significantly fewer iterations [23]. To demonstrate this, we show in Figure 6 the root-mean-square error (RMSE) for a collaborative filtering algorithm, alternating-least-square (ALS), with precise synchronous and approximate asynchronous gradient aggregation, over the number of iterations (i.e., rounds) of the algorithm for a synthetic data set. The figure shows that prediction error decreases to the same point, but that the asynchronous method offers substantial performance benefits. On larger real datasets, the effects are more pronounced. This example is one of a family of examples that show that understanding the statistical properties of the data and algorithms—such as that gradient descent is robust to sub-sampling and asynchrony—can yield substantial improvements in run-time performance, which we leverage.

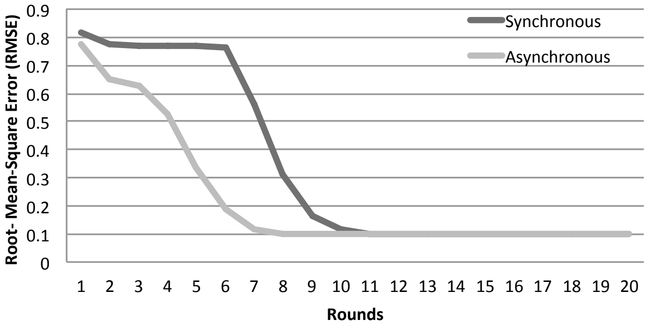


Figure 6: Synchronous vs. asynchronous execution of alternating least squares

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; some are inherently sequential and require greater consistency, some may not fit the supported MLbase patterns. 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.

6. RELATED WORK

MLbase is not the first system trying to make machine learning more accessible, but it is the first to free users from algorithm choices and to automatically optimize for distributed execution. Probably most related to MLbase are Weka [4], MADLib [18], and Mahout [3]. Weka is a collection of ML tools for data mining that simplifies their usage by providing a simple UI. Weka, however, requires expert knowledge to choose and configure the ML algorithm and is a single node system. On the database and distributed side, Mahout's goal is to build a scalable ML library on top of Hadoop, while MADLib provides an ML library for relational database systems. Neither system addresses the (difficult but necessary) challenge of optimizing the learning algorithms.

Google Predict [2] is Google's proprietary web-service for prediction problems, but restricts the maximum training data-size to 250MB. In [9] the authors make the case that databases should natively support predictive models and present a first prototype called Longview. We extend this vision by supporting all kinds of ML algorithms not just predictive models. Furthermore, our focus is on the optimization for ML instead of the language integration within the relational model.

Recently, there have been efforts to build distributed runtimes for more advanced analytical tasks. For example, Hyracks [11] and AMPLab's Spark [27] 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). SystemML [17] proposes an R-like language and shows how it can be optimized and compiled down to MapReduce. However, SystemML tries to support ML experts to develop efficient distributed algorithms and does not aim at simplifying the use of ML, for example, by automatically tuning the training step. Still, the the ideas of SystemML are compelling and we might leverage them as part of our physical plan optimization.

Finally, in [14] the authors show how many ML algorithms can be expressed as a relational-friendly convex-optimization problem, whereas the authors of [25] 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.

7. CONCLUSION

We described MLbase, a system aiming to make ML more accessible to non-experts. The core of MLbase is its optimizer, which transforms a declarative ML task into a sophisticated learning plan. During this process, the optimizer tries to find a plan that quickly returns a first quality answer to the user, allowing MLbase to improve the result iteratively in the background. Furthermore, MLbase is designed to be fully distributed, and it offers a run-time able to exploit the characteristics of machine learning algorithms. We are currently in the process of building the entire system. In this paper, we reported first results showing the potential of the optimizer as well as the performance advantages of algorithm-specific execution strategies.

8. ACKNOWLEDGMENTS

This research is supported in part by NSF CISE Expeditions award CCF-1139158, gifts from Amazon Web Services, Google, SAP, Blue Goji, Cisco, Cloudera, Ericsson, General Electric, Hewlett Packard, Huawei, Intel, Microsoft, NetApp, Oracle, Quanta, Splunk, VMware and by DARPA (contract #FA8650-11-C-7136).

9. REFERENCES

[1] GML AdaBoost Toolbox. `http://www.inf.ethz.ch/personal/vezhneva/Code/AdaBoostToolbox_v0.4.zip`.

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

[3] Mahout. `http://mahout.apache.org/`.

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

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

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

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

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

[9] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. The case for predictive database systems: Opportunities and challenges. In *CIDR*, pages 167–174, 2011.

[10] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *ISMIR*, 2011.

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

[12] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2, 2011.

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

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

[15] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

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

[17] A. Ghoting et al. Systemml: Declarative machine learning on mapreduce. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 231–242, Washington, DC, USA, 2011. IEEE Computer Society.

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

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

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

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

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

[23] B. Recht et al. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[24] J. Shawe-Taylor and N. Christianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

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

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

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

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