MLbase: A System for Distributed Machine Learning

Ameet Talwalkar
Problem: Scalable implementations difficult for ML Developers...

CHALLENGE: Can we simplify distributed ML development?
Problem: ML is difficult for End Users...

Too many algorithms
Too many ways to preprocess...
Too many knobs...
Difficult to debug

CHALLENGE: Can we automate ML pipeline construction?
MLbase: Cluster computing system designed for iterative computation (most active project in Apache Software Foundation)

MLlib: Spark’s core ML library

MLI: API to simplify ML development

MLOpt: Declarative layer to automate hyperparameter tuning

MLbase aims to simplify development and deployment of scalable ML pipelines
History of MLlib

Initial Release

- Developed by MLbase team in AMPLab
- Scala, Java
- Shipped with Spark v0.8 (Sep 2013)

15 months later...

- 80+ contributors from various organization
- Scala, Java, Python
- Latest release part of Spark v1.1 (Sep 2014)
What’s in MLlib?

- Alternating Least Squares
- Lasso
- Ridge Regression
- Logistic Regression
- Decision Trees
- Naïve Bayes
- Support Vector Machines
- K-Means
- Gradient descent
- L-BFGS
- Random data generation
- Linear algebra
- Feature transformations
- Statistics: testing, correlation
- Evaluation metrics

Collaborative Filtering for Recommendation
Prediction
Clustering
Optimization Primitives
Many Utilities
Benefits of MLlib

• Part of Spark
  • Integrated data analysis workflow
  • Free performance gains
Benefits of MLlib

- Part of Spark
  - Integrated data analysis workflow
  - Free performance gains
- Scalable, with rapid improvements in speed
- Python, Scala, Java APIs
- Broad coverage of applications & algorithms
Performance

Spark: 10-100X faster than Hadoop & Mahout

On a dataset with 660M users, 2.4M items, and 3.5B ratings
MLlib runs in 40 minutes with 50 nodes
Performance

Steady performance gains

~3X speedups on average

- Decision Trees
- ALS
- K-Means
- Logistic Regression
- Ridge Regression

Speedup
(Spark 1.0 vs. 1.1)
ML Developer API (MLI)

- **Shield ML Developers from low-details**
  - Provide familiar mathematical operators in distributed setting
  - Standard **APIs** defining ML algorithms and feature extractors

- **Tables**
  - Flexibility when loading data
  - Common interface for feature extraction / algorithms

- **Matrices**
  - Linear algebra (on local partitions at first)
  - Sparse and Dense matrix support

- **Optimization Primitives**
  - Distributed implementations of common patterns
MLI, MLlib and Roadmap

• MLlib incorporate ideas from MLI
  • Matrices and optimization primitives already in MLlib
  • Tables and ML API will be in next release

• Longer term for MLlib
  • Scalable implementations of standard ML methods and underlying optimization primitives
  • Further support for ML pipeline development (including hyper parameter tuning using ideas from MLOpt)

Feedback and Contributions Encouraged!
Vision
MLlib / MLI
MLOpt
User declaratively specifies task

PAQ = Predictive Analytic Query

Search through MLlib to find the best model/pipeline

**SELECT** e.sender, e.subject, e.message
**FROM** Emails e
**WHERE** e.user = 'Bob'
**AND PREDICT**(e.spam, e.message) = false **GIVEN** LabeledData

**ABSTRACT**

The proliferation of massive datasets combined with the development of sophisticated analytical techniques have enabled a wide variety of novel applications such as improved product recommendations, automatic image tagging, and improved speech driven interfaces. These and many other applications can be supported by Predictive Analytic Queries (PAQs). The major obstacle to supporting these queries is the challenging and expensive process of PAQ planning, which involves identifying and training an appropriate predictive model. Recent efforts aiming to automate this process have focused on single node implementations and have assumed that model training itself is a black box, thus limiting the effectiveness of such approaches on large-scale problems. In this work, we build upon these recent efforts and propose an integrated PAQ planning architecture that combines advanced model search techniques, bandit resource allocation via runtime algorithm introspection, and physical optimization via batching. The resulting system, **TuPAQ**, solves the PAQ planning problem with comparable accuracy to exhaustive strategies but an order of magnitude faster, and can scale to models trained on terabytes of data across hundreds of machines.

**1. INTRODUCTION**

Over the past four decades, a great deal of database systems research has focused on finding efficient execution strategies for a number of different workloads – single node and distributed transactional, analytical, and stream processing workloads have all been active areas of study. Rapidly growing data volumes coupled with the maturity of sophisticated statistical techniques have led to demand for support of a new type of workload: predictive analytics over large scale, distributed datasets. Indeed, the support of predictive analytic queries in database systems is an increasingly well studied area. Systems like MLbase [37], MADLib [33], COLUMBUS [38], MauveDB [27], BayesStore [52] and DimmWitted [56] are all efforts to integrate statistical query processing with a data management system.

Concretely, users would like to issue queries that involve reasoning about predicted attributes, where predicted attributes are derived from a set of observed input attributes along with a labeled training dataset. We refer to such queries as Predictive Analytic Queries, or PAQs. The predictions returned by the system should be highly accurate both on training data and new data as it comes into the system. PAQs consist of traditional database queries along with new predictive clauses, and these predictive clauses are the focus of this work. Examples of PAQs are illustrated in Figure 1, with the predictive clauses highlighted in green.

Given recent advances in statistical methodology, supervised machine learning (ML) techniques are a natural way to support the predictive clauses in PAQs. In the supervised learning setting, a statistical model leverages training data to relate the input attributes to the desired output attribute. Furthermore, ML methods learn better models as the size of the training data increases, and recent advances in distributed ML algorithm development, e.g., [13, 44, 40] enable large-scale model training in the distributed setting. Unfortunately, the application of supervised learning techniques to a new input dataset is computationally demanding and technically challenging. For a non-expert, the process of carefully preprocessing the input attributes, selecting the appropriate ML model, and tuning the model is expensive and difficult.

**SQL**

**Result**

**PAQ**

**Model**

**ML**

- User declaratively specifies task
- PAQ = Predictive Analytic Query
- Search through MLlib to find the best model/pipeline
A Standard ML Pipeline

- In practice, model building is an iterative process of continuous refinement
- Our grand vision is to automate the construction of these pipelines
Training A Model

- Iteratively read through data
  - compute gradient
  - update model
  - repeat until converged
- Requires *multiple passes*
- Common access pattern
  - ALS, Random Forests, etc.
- Minutes to train an SVM on 200GB of data on a 16-node cluster

$$w := w - \alpha \nabla Q(w) = w - \alpha \sum_{i=1}^{n} \nabla Q_i(w),$$
The Tricky Part

- **Model**
  - Logistic Regression, SVM, Tree-based, etc.
- **Model hyper-parameters**
  - Learning Rate, Regularization, etc.

- **Featurization**
  - Text: n-grams, TF-IDF
  - Images: Gabor filters, random convolutions
  - Random projection? Scaling?
In practice, model building is an iterative process of continuous refinement.

Our grand vision is to automate the construction of these pipelines.

Start with one aspect of the pipeline - model selection.
One Approach

✧ **Sequential Grid Search**
  ✧ Search over all hyperparameters, algorithms, features, etc.

✧ **Drawbacks**
  ✧ Expensive to compute models
  ✧ Hyperparameter space is large

✧ **Common in practice!**
A Better Approach

✦ Better resource utilization
   ✦ through batching

✦ Early Stopping

✦ Improved Search
A Tale Of 3 Optimizations

Better Resource Utilization

Early Stopping

Improved Search
Better Resource Utilization

✧ Typical model update requires 2-4 flops/double

✧ But modern memory much slower than processors
  ✧ We can do 25 flops / double read!
  ✧ This equates to 6-8 model updates per double we read, assuming models fit in cache

✧ Train multiple models simultaneously
What Do We See In Spark?

- 2x and 5x increase in models trained/sec with batching
What Do We See In Spark?

- These numbers are with vector-matrix multiplies

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>1.50</td>
<td>1.51</td>
<td>1.38</td>
</tr>
<tr>
<td>5</td>
<td>4.05</td>
<td>2.53</td>
<td>1.93</td>
<td>1.36</td>
</tr>
<tr>
<td>10</td>
<td>5.31</td>
<td>3.40</td>
<td>2.37</td>
<td>1.14</td>
</tr>
</tbody>
</table>
What Do We See In Spark?

- These numbers are with vector-matrix multiplies

- Can do better when rewriting in terms of matrix-matrix multiplies
A Tale Of 3 Optimizations

Better Resource Utilization

Early Stopping

Improved Search
Early Stopping

- Each point is a trained model
- Some models look bad early
- So we give up early!
- So far a heuristic…
  - …but can be framed as a multi-armed bandit problem
Early Stopping

✦ Each point is a trained model

✦ Some models look bad early
✦ So we give up early!

✦ So far a heuristic…
✦ …but can be framed as a multi-armed bandit problem
A Tale Of 3 Optimizations

Better Resource Utilization

Algorithmic Speedups

Improved Search
What Search Method?

- Various derivative-free optimization techniques
  - Simple ones (Grid, Random)
  - Classic Derivative-Free (Nelder-Mead, Powell’s method)
  - Bayesian (e.g., SMAC, TPE)

- What should we do?
Comparison of Search Methods Across Learning Problems

What Search Method?
Putting It All Together

- First version of MLbase optimizer
- 30GB dense images (240K x 16K)
- 2 model families, 5 hyperparams
- Baseline: grid search
- Our method: combination of
  - Batching
  - Early stopping
  - Random or TPE

20x speedup compared to grid search
15 minutes vs 5 hours!
Does It Scale?

- 1.5TB dataset (1.2M x 160K)
- 128 nodes, thousands of passes over data
- Tried 32 models in 15 hours
- Good answer after 11 hours
Future Work

Automated ML Pipeline Construction

Data → Feature Extraction → Model Training → Final Model
Other Future Work

- Ensembling
- Leverage sampling
- Better parallelism for smaller datasets
- Multiple hypothesis testing issues
**MLOpt**: Declarative layer to automate hyperparameter tuning

**MLI**: API to simplify ML development

**MLlib**: Spark’s core ML library

**Spark**: Cluster computing system designed for iterative computation

**MLbase website**
www.mlbase.org

**MLlib Programming Guide**
spark.apache.org/docs/latest/mllib-guide.html

**Spark user lists**
spark.apache.org/community.html

**Scalable Machine Learning**
www.edx.org/course/scalable-machine-learning-uc-berkeleyx-cs190-1x