Evan Sparks and Ameet Talwalkar
UC Berkeley

Collaborators: Tim Kraska\textsuperscript{2}, Virginia Smith\textsuperscript{1}, Xinghao Pan\textsuperscript{1}, Shivaram Venkataraman\textsuperscript{1}, Matei Zaharia\textsuperscript{1}, Rean Griffith\textsuperscript{3}, John Duchi\textsuperscript{1}, Joseph Gonzalez\textsuperscript{1}, Michael Franklin\textsuperscript{1}, Michael I. Jordan\textsuperscript{1}

\textsuperscript{1}UC Berkeley \quad \textsuperscript{2}Brown \quad \textsuperscript{3}VMware
Problem: Scalable implementations difficult for ML Developers...
Problem: ML is difficult for End Users...

- Too many algorithms...
- Too many knobs...
- Difficult to debug...
- Doesn’t scale...
- Doesn’t scale...
1. Easy scalable ML development (ML Developers)
2. User-friendly ML at scale (End Users)

Along the way, we gain insight into data intensive computing
Vision
MLI Details
Current Status
ML Workflow
Matlab Stack

- Lapack: low-level Fortran linear algebra library
- Matlab Interface
  - Higher-level abstractions for data access / processing
  - More extensive functionality than Lapack
  - Leverages Lapack whenever possible
- Similar stories for R and Python
**Spark**: cluster computing system designed for iterative computation

**MLlib**: low-level ML library in Spark

**MLI**: API / platform for feature extraction and algorithm development
  - Platform independent

**ML Optimizer**: automates model selection
  - Solves a search problem over feature extractors and algorithms in MLI
**Example: MLlib**

- **Goal:** Classification of text file
- **Featurize data manually**
- **Calls MLlib’s LR function**

```scala
def main(args: Array[String]) {
    val mc = new MLContext("local", "MLILR")
    // Read in file from HDFS
    val rawTextTable = mc.csvFile(args(0), Seq("class", "text"))
    // Run feature extraction
    val classes = rawTextTable(??, "class")
    val ngrams = tfIdf(nGrams(rawTextTable(??, "text"), n=2, top=3))
    val featureizedTable = classes.zip(ngrams)
    // Classify the data using Logistic Regression.
    val lrModel = LogisticRegression(featureizedTable, stepSize=0.1, numIter=12)
}
```
Example: MLI

- Use built-in feature extraction functionality
- MLI Logistic Regression leverages MLlib
- Extensions:
  - Embed in cross-validation routine
  - Use different feature extractors / algorithms
  - Write new ones

```scala
def main(args: Array[String]) {
  val mc = new MLContext("local", "MLILR")

  // Read in file from HDFS
  val rawTextTable = mc.csvFile(args(0), Seq("class","text"))

  // Run feature extraction
  val classes = rawTextTable(??, "class")
  val ngrams = tfIdf(nGrams(rawTextTable(??, "text"), n=2, top=30000))
  val featureizedTable = classes.zip(ngrams)
```

Fig. 15: Matrix Factorization via ALS code in MATLAB (top) and MLI (bottom).
Example: ML Optimizer

- User declaratively specifies task
- ML Optimizer searches through MLI

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

**MLI Details**

Current Status

ML Workflow
Lay of the Land

Ease of use

Performance, Scalability

Matlab, R

Mahout

GraphLab, VW

MATLAB

R

GraphLab

Mahout

VOWPAL WABBIT

+ Easy (Resembles math, limited set up)
+ Sufficient for prototyping / writing papers
  - Ad-hoc, non-scalable scripts
  - Loss of translation upon re-implementation

+ Scalable and (sometimes) fast
+ Existing open-source libraries
  - Difficult to set up, extend
Examples

‘Distributed’ Divide-Factor-Combine (DFC)
- Initial studies in MATLAB (Not distributed)
- Distributed prototype involving compiled MATLAB

Mahout ALS with Early Stopping
- Theory: simple if-statement (3 lines of code)
- Practice: sift through 7 files, nearly 1K lines of code
Lay of the Land

Ease of use vs. Performance, Scalability

- **Matlab, R**
  - + Easy (Resembles math, limited set up)
  - + Sufficient for prototyping / writing papers
  - — Ad-hoc, non-scalable scripts
  - — Loss of translation upon re-implementation

- **MLI**
  - —

- **MLlib**
  - + Scalable and (sometimes) fast
  - + Existing open-source libraries
  - — Difficult to set up, extend

- **Mahout**
  - —

- **GraphLab, VW**
  - —

- **Vowpal Wabbit**
  - —
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
val x: RDD[spark.util.Vector]
val x: RDD[breeze.linalg.Vector]
val x: RDD[BIDMat.SMat]

NEW
val x: MLTable

✦ Abstract interface for arbitrary backend
✦ Common interface to support an optimizer
ML Developer API (MLI)

✧ Shield ML Developers from low-details
  ✧ provide familiar mathematical operators in distributed setting

✧ Table Computation (MLTable)
  ✧ Flexibility when loading data (heterogenous, missing)
  ✧ Common interface for feature extraction / algorithms
  ✧ Supports MapReduce and relational operators

✧ Linear Algebra (MLSubMatrix)
  ✧ Linear algebra on *local* partitions
  ✧ Sparse and Dense matrix support

✧ Optimization Primitives (MLSolve)
  ✧ Distributed implementations of common patterns

✧ DFC: ~50 lines of code
✧ ALS: early stopping in 3 lines; < 40 lines total
# MLI Ease of Use

## Logistic Regression

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>11</td>
</tr>
<tr>
<td>Vowpal Wabbit</td>
<td>721</td>
</tr>
<tr>
<td>MLI</td>
<td>55</td>
</tr>
</tbody>
</table>

## Alternating Least Squares

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>20</td>
</tr>
<tr>
<td>Mahout</td>
<td>865</td>
</tr>
<tr>
<td>GraphLab</td>
<td>383</td>
</tr>
<tr>
<td>MLI</td>
<td>32</td>
</tr>
</tbody>
</table>
MLI/Spark Performance

- **Walltime**: elapsed time to execute task

- **Weak scaling**
  - fix problem size *per processor*
  - ideally: constant walltime as we grow cluster

- **Strong scaling**
  - fix total problem size
  - ideally: linear speed up as we grow cluster

- **EC2 Experiments**
  - m2.4xlarge instances, up to 32 machine clusters
Logistic Regression - Weak Scaling

- Full dataset: 200K images, 160K dense features
- Similar weak scaling
- MLI/Spark within a factor of 2 of VW’s walltime
Logistic Regression - Strong Scaling

- Fixed Dataset: 50K images, 160K dense features
- MLI/Spark exhibits better scaling properties
- MLI/Spark faster than VW with 16 and 32 machines
ALS - Walltime

<table>
<thead>
<tr>
<th>System</th>
<th>Walltime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>15443</td>
</tr>
<tr>
<td>Mahout</td>
<td>4206</td>
</tr>
<tr>
<td>GraphLab</td>
<td>291</td>
</tr>
<tr>
<td>MLI/Spark</td>
<td>481</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
- MLI/Spark an order of magnitude faster than Mahout
- MLI/Spark within factor of 2 of GraphLab
Vision
MLI Details
**Current Status**
ML Workflow
MLI Functionality

**Regression:** Linear Regression (+Lasso, Ridge)

**Collaborative Filtering:** Alternating Least Squares

**Clustering:** K-Means

**Classification:** Logistic Regression, Linear SVM (+L1, L2)

**Optimization Primitives:** Parallel Gradient
MLbase Stack Status

Goal 1: Summer Release
ML Developer

ML Optimizer
MLI
MLlib
Spark

Goal 2: Winter Release
End User
Future Directions

✧ **Identify minimal set of ML operators**
  ✧ Expose internals of ML algorithms to optimizer

✧ **Plug-ins to Python, R**

✧ **Visualization** for unsupervised learning and exploration

✧ **Advanced ML capabilities**
  ✧ Time-series algorithms
  ✧ Graphical models
  ✧ Advanced Optimization (e.g., asynchronous computation)
  ✧ Online updates
  ✧ Sampling for efficiency
Vision
MLI Details
Current Status
ML Workflow
Typical Data Analysis Workflow

1. **Spark, MLI**
   - Obtain / Load Raw Data

2. **Spark, [MLI]**
   - Data Exploration

3. **MLI**
   - Feature Extraction

4. **MLI, MLlib**
   - Learning

5. **MLI**
   - Evaluation

6. **Scala**
   - Deployment

Adapted from slides by Ariel Kleiner
**Goal:** Learn a mapping from entities to discrete labels

**Example:** Spam Classification
- Entities are emails
- Labels are \{spam, not-spam\}
- Given past labeled emails, we want to predict whether a new email is spam or not-spam

Adapted from slides by Ariel Kleiner
Binary Classification

**Goal:** Learn a mapping from entities to discrete labels

**Other Examples:**
- Click (and clickthrough rate) prediction
- Fraud detection
- Face detection
- Exercise: “ARTS” vs “LIFE” on Wikipedia
- Real data
Classification Pipeline

1. Randomly split full data into disjoint subsets
2. Featurize the data
3. Use training set to learn a classifier
4. Evaluate classifier on test set (avoid overfitting)
5. Use classifier to predict in the wild

Adapted from slides by Ariel Kleiner
E.g., Spam Classification

From: illegitimate@bad.com
"Eliminate your debt by giving us your money..."

From: bob@good.com
"Hi, it's been a while! How are you? ..."

Adapted from slides by Ariel Kleiner
Featurization

✦ Most classifiers require numeric descriptions of entities

✦ **Featurization**: Transform each entity into a vector of real numbers
  ✦ Opportunity to incorporate domain knowledge
  ✦ Useful even when original data is already numeric

Adapted from slides by Ariel Kleiner
E.g., “Bag of Words”

✦ Entities are documents
✦ Build Vocabulary

Example (Spam Classification):

From: illegitimate@bad.com
"Eliminate your debt by giving us your money..."

From: bob@good.com
"Hi, it's been a while! How are you? ..."

Vocabulary

been
debt
eliminate
giving
how
it's
money
while

Adapted from slides by Ariel Kleiner
E.g., “Bag of Words”

✧ Entities are documents
✧ Build Vocabulary
✧ Derive feature vectors from Vocabulary
  ✧ Exercise: we’ll use bigrams

From: illegitimate@bad.com
"Eliminate your debt by giving us your money..."
Support Vector Machines (SVMs)

- “Max-Margin”: find linear separator with the largest separation between the two classes
- Extensions:
  - non-separable setting
  - non-linear classifiers (kernels)

Credit: Foundations of Machine Learning
Mohri, Rostamizadeh, Talwalkar
Model Evaluation

- Test set simulates performance on new entity
  - Performance on training data overly optimistic!
  - “Overfitting”; “Generalization”
- Various metrics for quality; accuracy is most common
- Evaluation process
  - Train on training set (don’t expose test set to classifier)
  - Make predictions using test set (ignoring test labels)
  - Compute fraction of correct predictions on test set
- Other more sophisticated evaluation methods, e.g., cross-validation

Adapted from slides by Ariel Kleiner
Contributions encouraged!

www.mlbbase.org