Distributed Machine Learning on Spark

Evan Sparks
UC Berkeley
January 31st, 2014
Collaborators: Ameet Talwalkar, Xiangrui Meng, Virginia Smith, Xinghao Pan, Shivaram Venkataraman, Matei Zaharia, Rean Griffith, John Duchi, Joseph Gonzalez, Michael Franklin, Michael I. Jordan, Tim Kraska

www.mlbase.org
Problem: Scalable implementations difficult for ML Developers...
Problem: Scalable implementations difficult for ML Developers...
Problem: Scalable implementations difficult for ML Developers...

VOWPAL WABBIT

\[ \text{mahout} \]

GraphLab
Problem: ML is difficult for End Users...

Too many algorithms...
Problem: ML is difficult for End Users...

Too many algorithms...

Too many knobs...
Problem: ML is difficult for End Users...

- Too many algorithms...
- Too many knobs...
- Difficult to debug...
Problem: ML is difficult for End Users...

Too many algorithms...
Too many knobs...
Difficult to debug...
Doesn’t scale...
Problem: ML is difficult for End Users...

Too many algorithms...
Too many knobs...
Difficult to debug...
Doesn’t scale...

ML Developer
1. Easy scalable ML development (*ML Developers*)
2. User-friendly ML at scale (*End Users*)
Matlab Stack
Matlab Stack

Single Machine
Matlab Stack

- Lapack: low-level Fortran linear algebra library
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
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
MLbase Stack

Matlab Interface
Lapack
Single Machine
MLbase Stack

Matlab Interface
Lapack
Single Machine
Runtime(s)
Spark: cluster computing system designed for iterative computation
MLbase Stack

Spark: cluster computing system designed for iterative computation
MLlib: production-quality ML library in Spark
Spark: cluster computing system designed for iterative computation

MLlib: production-quality ML library in Spark

MLI: experimental API for simplified feature extraction and algorithm development
**Spark**: cluster computing system designed for iterative computation

**MLlib**: production-quality ML library in Spark

**MLI**: experimental API for simplified feature extraction and algorithm development

**ML Optimizer**: a declarative layer to simplify access to large-scale ML
Overview

MLlib

Collaborative Filtering

ALS Details
**MLlib**

**Classification:** Logistic Regression, Linear SVM (+L1, L2), Decision Trees, Naive Bayes

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

**Collaborative Filtering:** Alternating Least Squares

**Clustering / Exploration:** K-Means, SVD

**Optimization Primitives:** SGD, Parallel Gradient

**Interoperability:** Scala, Java, PySpark (0.9)
MLlib

**Classification:** Logistic Regression, Linear SVM (+L1, L2), Decision Trees, Naive Bayes

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

**Collaborative Filtering:** Alternating Least Squares

**Clustering / Exploration:** K-Means, SVD

**Optimization Primitives:** SGD, Parallel Gradient

**Interoperability:** Scala, Java, PySpark (0.9)

Included within Spark codebase

- Unlike Mahout/Hadoop
- Part of Spark 0.8 release
- Continued support via Spark project
- Community involvement has been terrific: ALS with implicit feedback (0.8.1), Naive Bayes (0.9), SVD (0.9), Decision Trees (soon!)
MLlib Performance
MLlib Performance

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

- **Walltime**: elapsed time to execute task
- **Weak scaling**
  - fix problem size *per processor*
  - ideally: constant walltime as we grow cluster
MLlib 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
MLlib 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
Logistic Regression - Weak Scaling

✦ Full dataset: 200K images, 160K dense features
Logistic Regression - Weak Scaling

- Full dataset: 200K images, 160K dense features
- Similar weak scaling
Logistic Regression - Weak Scaling

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

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

- Fixed Dataset: 50K images, 160K dense features
Logistic Regression - Strong Scaling

- Fixed Dataset: 50K images, 160K dense features
- MLlib exhibits better scaling properties
Logistic Regression - Strong Scaling

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

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
## ALS - Walltime

<table>
<thead>
<tr>
<th>System</th>
<th>Walltime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>15443</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 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>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 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>MLlib</td>
<td>481</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
- **MLlib an order of magnitude faster than Mahout**
## 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>MLlib</td>
<td>481</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
- **MLlib** an order of magnitude faster than Mahout
- **MLlib** within factor of 2 of GraphLab
Deployment Considerations
Deployment Considerations

Vowpal Wabbit, GraphLab

✦ Data preparation specific to each program
✦ Non-trivial setup on cluster
✦ No fault tolerance
Deployment Considerations

Vowpal Wabbit, GraphLab
- Data preparation specific to each program
- Non-trivial setup on cluster
- No fault tolerance

MLlib
- Reads files from HDFS
- Launch/compile/run on cluster with a few commands
- RDD’s provide fault tolerance naturally
Deployment Considerations

**Vowpal Wabbit, GraphLab**
- Data preparation specific to each program
- Non-trivial setup on cluster
- No fault tolerance

**MLlib**
- Reads files from HDFS
- Launch/compile/run on cluster with a few commands
- RDD’s provide fault tolerance naturally
- Part of Spark’s ‘swiss army knife’ ecosystem
Deployment Considerations

Vowpal Wabbit, GraphLab
- Data preparation specific to each program
- Non-trivial setup on cluster
- No fault tolerance

MLlib
- Reads files from HDFS
- Launch/compile/run on cluster with a few commands
- RDD’s provide fault tolerance naturally
- Part of Spark’s ‘swiss army knife’ ecosystem
  - Shark, Spark Streaming, Graph-X, BlinkDB, etc.
Vision
MLlib
**Collaborative Filtering**
ALS Details
# Matrix Completion

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>★</td>
<td>★★★★</td>
<td></td>
</tr>
<tr>
<td></td>
<td>★★★★</td>
<td>★★</td>
</tr>
<tr>
<td>★</td>
<td></td>
<td>★★</td>
</tr>
<tr>
<td>★★★</td>
<td></td>
<td>★★</td>
</tr>
<tr>
<td>★★★★</td>
<td>★★★</td>
<td>★★</td>
</tr>
</tbody>
</table>

Matrix Completion (MCM) is a technique used to fill in missing values in a matrix. This figure illustrates a simple example of MCM, where the goal is to predict the missing values in the matrix based on the observed values.
Matrix Completion

**Goal**: Recover a matrix from a subset of its entries
Matrix Completion

**Goal:** Recover a matrix from a subset of its entries

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>★</td>
<td>★★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>★</td>
<td>★★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>★★★★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>★</td>
<td>★★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>★★★★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>★</td>
<td>★★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>★★★★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>★</td>
<td>★★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>★★★★</td>
<td>★</td>
<td></td>
</tr>
</tbody>
</table>

[Netflix logo]
### Matrix Completion

**Goal:** Recover a matrix from a subset of its entries

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>⭐</td>
<td>⭐⭐⭐⭐</td>
</tr>
<tr>
<td>⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐</td>
<td>⭐</td>
</tr>
</tbody>
</table>

Companies: [last.fm], [Netflix], [Pandora], [Facebook], [LinkedIn], [Amazon]
Reducing Degrees of Freedom

<table>
<thead>
<tr>
<th>🎥</th>
<th>🎥</th>
<th>🎥</th>
</tr>
</thead>
<tbody>
<tr>
<td>✫</td>
<td>✫</td>
<td>✫</td>
</tr>
<tr>
<td>✫</td>
<td>✫</td>
<td>✫</td>
</tr>
<tr>
<td>✫</td>
<td>✫</td>
<td>✫</td>
</tr>
<tr>
<td>✫</td>
<td>✫</td>
<td>✫</td>
</tr>
<tr>
<td>✫</td>
<td>✫</td>
<td>✫</td>
</tr>
<tr>
<td>✫</td>
<td>✫</td>
<td>✫</td>
</tr>
</tbody>
</table>
Reducing Degrees of Freedom

**Problem:** Impossible without additional information

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>★</td>
<td>★ ★ ★ ★</td>
<td></td>
</tr>
<tr>
<td>★</td>
<td>★ ★ ★</td>
<td>★ ★</td>
</tr>
<tr>
<td>★</td>
<td>★ ★ ★</td>
<td>★ ★</td>
</tr>
<tr>
<td>★</td>
<td>★ ★ ★</td>
<td>★ ★</td>
</tr>
<tr>
<td>★</td>
<td>★ ★ ★</td>
<td>★ ★</td>
</tr>
</tbody>
</table>

$m \times n$
## Reducing Degrees of Freedom

- **Problem**: Impossible without additional information
- \( mn \) degrees of freedom

\[
\begin{array}{ccc}
\star & \star\star\star\star & \\
\star & \star\star\star\star & \star\star \\
\star\star\star\star & \star & \\
\star\star\star\star & \star\star & \\
\star\star\star\star & \star\star & \\
\star\star\star\star & \star\star & \\
\end{array}
\]

\[ m \times n = \text{\textit{Low-rank}} \]
Reducing Degrees of Freedom

✦ **Problem**: Impossible without additional information
  - $mn$ degrees of freedom

✦ **Solution**: Assume small # of factors determine preference
  - $O(m + n)$ degrees of freedom

$m \times n$ matrix:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>★</td>
<td>★★</td>
<td>★★</td>
</tr>
<tr>
<td>★</td>
<td>★★</td>
<td>★</td>
</tr>
<tr>
<td>★★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>★</td>
<td>★★</td>
<td></td>
</tr>
<tr>
<td>★★</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$m \times r$ matrix:

$n \times r$ matrix:

$m = m$

‘Low-rank’
Alternating Least Squares
Alternating Least Squares
Alternating Least Squares
Alternating Least Squares
Alternating Least Squares

training error for first user = \((\square - \square) + (\triangle - \square)\)
Alternating Least Squares

training error for first user = \((\boxed{\text{user factors}}) - \boxed{\text{movie factors}}) + \boxed{\text{ratings}}$

ALS: alternate between updating user and movie factors
Alternating Least Squares

training error for first user = \((\mathbf{U} - \mathbf{H}) + (\mathbf{V} - \mathbf{W})\)

ALS: alternate between updating user and movie factors

update first user by finding \(\mathbf{U}\) that minimizes training error
Alternating Least Squares

\[
\text{training error for first user} = (\bullet - \color{red}{\blacksquare}) + (\bullet - \color{green}{\blacksquare})
\]

ALS: alternate between updating user and movie factors

update first user by finding \( \color{green}{\blacksquare} \) that minimizes training error

reduces to standard linear regression problem
Alternating Least Squares

\[
\text{training error for first user} = (\text{•} - \text{□}) + (\text{•} - \text{□})
\]

ALS: alternate between updating user and movie factors

update first user by finding \(\text{□}\) that minimizes training error

reduces to standard linear regression problem

can update all users in parallel!
Alternating Least Squares

training error for first user = \((\square - \blacksquare) + (\blacksquare - \blacksquare)\)
Alternating Least Squares

\[ M \overset{\rightarrow}{=} W H^\top \]

training error for first user = \((\text{red} - \text{white}) + (\text{blue} - \text{green})\)
Alternating Least Squares

\[
\text{training error for first user} = \left( \overbrace{\text{ }}^{M_{1,j}} - \overbrace{\text{ }}^{W_1} \right) + \left( \overbrace{\text{ }}^{M_{1,j}} - \overbrace{\text{ }}^{W_1} \right)
\]

\[
= \sum_{(1,j) \in \Omega} (M_{1,j} - W_1 H_{j}^\top)^2
\]
Alternating Least Squares

\[
\text{training error for first user} = (\begin{bmatrix} \bigcirc \bigcirc \bigcirc \\ \bigcirc \bigcirc \bigcirc \end{bmatrix} - \begin{bmatrix} \bigcirc \bigcirc \\ \bigcirc \bigcirc \end{bmatrix}) + (\begin{bmatrix} \bigcirc \bigcirc \\ \bigcirc \bigcirc \end{bmatrix} - \begin{bmatrix} \bigcirc \bigcirc \\ \bigcirc \bigcirc \end{bmatrix})
\]

\[
= \sum_{(1,j) \in \Omega} (M_{1j} - W_1 H_j^\top)^2
\]

\[
W_1^* = (H_{\Omega_1}^\top H_{\Omega_1})^{-1} H_{\Omega_1}^\top M_{1\Omega_1}^\top
\]
Exercise Today
Exercise Today

• Load 1,000,000 ratings from MovieLens.
Exercise Today

- Load 1,000,000 ratings from MovieLens.
- Get YOUR ratings.
Exercise Today

• Load 1,000,000 ratings from MovieLens.
• Get YOUR ratings.
• Split into training/validation.
Exercise Today

• Load 1,000,000 ratings from MovieLens.
• Get YOUR ratings.
• Split into training/validation.
• Fit a model.
Exercise Today

• Load 1,000,000 ratings from MovieLens.
• Get **YOUR** ratings.
• Split into training/validation.
• Fit a model.
• Validate and tune hyperparameters.
Exercise Today

• Load 1,000,000 ratings from MovieLens.
• Get YOUR ratings.
• Split into training/validation.
• Fit a model.
• Validate and tune hyperparameters.
• Get YOUR recommendations.
Exercise Today

• Load 1,000,000 ratings from MovieLens.
• Get YOUR ratings.
• Split into training/validation.
• Fit a model.
• Validate and tune hyperparameters.
• Get YOUR recommendations.
• Great example of a Spark application!
Vision
MLlib
Collaborative Filtering
ALS Details
Three Kinds of ALS

- Broadcast Everything
- Data Parallel
- Fully Parallel
Three Kinds of ALS

- Broadcast Everything
- Data Parallel
- Fully Parallel
Broadcast Everything

Ratings

Movie Factors

User Factors

Master

Workers
Broadcast Everything

- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn’t scale well.
- Ships with Spark Examples
Broadcast Everything

- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn't scale well.
- Ships with Spark Examples
Broadcast Everything

- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn’t scale well.
- Ships with Spark Examples
Broadcast Everything

- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn’t scale well.
- Ships with Spark Examples
Broadcast Everything

- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn't scale well.
- Ships with Spark Examples
Broadcast Everything

- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn’t scale well.
- Ships with Spark Examples
Three Kinds of ALS

- Broadcast Everything
- Data Parallel
- Fully Parallel
Data Parallel

Movie Factors

User Factors

Master

Workers
Data Parallel

- **Workers** load data

- **Movie Factors**

- **User Factors**

- **Master**

- **Workers**
Data Parallel

- **Workers** load data
- Master broadcasts initial models
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
- Works on large datasets
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
- Works on large datasets
- Works well for smaller models. (low K)
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
- Works on large datasets
- Works well for smaller models. (low K)
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
- Works on large datasets
- Works well for smaller models. (low K)
Data Parallel

- **Workers** load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
- Works on large datasets
- Works well for smaller models. (low K)
Three Kinds of ALS

- Broadcast Everything
- Data Parallel
- Fully Parallel
Fully Parallel

Master

Workers
Fully Parallel

- **Workers** load data
Fully Parallel

- *Workers* load data
- Models are instantiated at workers.
Fully Parallel

- **Workers** load data
- Models are instantiated at workers.
- At each iteration, models are shared via *join* between workers.
Fully Parallel

- **Workers** load data
- Models are instantiated at workers.
- At each iteration, models are shared via join between workers.
- Much better scalability.

Diagram:

- Master
- Workers

Components:
- Movie Ratings
- User Factors
**Fully Parallel**

- **Workers** load data
- Models are instantiated **at workers**.
- At each iteration, models are shared via **join** between workers.
- Much better scalability.
- Works on large datasets

Master

Workers
Fully Parallel

- **Workers** load data
- Models are instantiated at workers.
- At each iteration, models are shared via join between workers.
- Much better scalability.
- Works on large datasets
- Works on big models (higher K)
Fully Parallel

- **Workers** load data
- Models are instantiated at workers.
- At each iteration, models are shared via join between workers.
- Much better scalability.
- Works on large datasets
- Works on big models (higher K)
Fully Parallel

- **Workers** load data
- Models are instantiated at workers.
- At each iteration, models are shared via *join* between workers.
- Much better scalability.
- Works on large datasets
- Works on big models (higher K)
Three Kinds of ALS

- Broadcast Everything
- Data Parallel
- Fully Parallel
Three Kinds of ALS

• Broadcast Everything
• Data Parallel
• Fully Parallel

Blocked
**ML Optimizer**: a declarative layer to simplify access to large-scale ML

**MLI**: experimental API for simplified feature extraction and algorithm development

**MLlib**: production-quality ML library in Spark

**Spark**: cluster computing system designed for iterative computation
### ML Optimizer
A declarative layer to simplify access to large-scale ML

### MLI
Experimental API for simplified feature extraction and algorithm development

### MLlib
Production-quality ML library in Spark

### Spark
Cluster computing system designed for iterative computation

---

**MLbase**
[www.mlbase.org](http://www.mlbase.org)
**ML Optimizer**: a declarative layer to simplify access to large-scale ML

**MLI**: experimental API for simplified feature extraction and algorithm development

**MLlib**: production-quality ML library in Spark

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

THANKS!

QUESTIONS?

www.mlbase.org