Fast, General Parallel Computation for Machine Learning

Robin Elizabeth Yancey and Norm Matloff
University of California at Davis

P2PS Workshop, ICPP 2018
Outline

• Motivation.
• Software Alchemy.
• Theoretical foundations.
• Empirical investigation.
Outline

- Motivation.
- Software Alchemy.
- Theoretical foundations.
- Empirical investigation.
Motivation

Characteristics of machine learning (ML) algorithms:
• Big Data: in $n \times p$ (cases $\times$ features) dataset, both $n$ AND $p$ large.
• Compute-intensive algorithms: sorting, k-NN, matrix inversion, iteration.
• Not generally embarrassingly parallel (EP). (An exception: Random Forests – grow different trees within different processes.)
• Memory problems: The computation may not fit on a single machine (esp. in R or GPUs).
Motivation

Characteristics of machine learning (ML) algorithms:

- **Big Data**: $n \times p$ (cases $\times$ features) dataset, both $n$ AND $p$ large.
- **Compute-intensive algorithms**: sorting, k-NN, matrix inversion, iteration.
- **Not generally embarrassingly parallel (EP)**. (An exception: Random Forests – grow different trees within different processes.)
- **Memory problems**: The computation may not fit on a single machine (esp. in R or GPUs).
Motivation

Characteristics of machine learning (ML) algorithms:

- **Big Data**: in $n \times p$ (cases $\times$ features) dataset, both $n$ AND $p$ large.
Motivation

Characteristics of machine learning (ML) algorithms:

- Big Data: in $n \times p$ (cases $\times$ features) dataset, both $n$ AND $p$ large.
- Compute-intensive algorithms: sorting, k-NN, matrix inversion, iteration.
Motivation

Characteristics of machine learning (ML) algorithms:

- **Big Data**: in $n \times p$ (cases × features) dataset, both $n$ AND $p$ large.
- **Compute-intensive algorithms**: sorting, k-NN, matrix inversion, iteration.
- **Not generally embarrassingly parallel (EP)**.
Motivation

Characteristics of machine learning (ML) algorithms:

- **Big Data:** in $n \times p$ (cases $\times$ features) dataset, both $n$ AND $p$ large.
- **Compute-intensive algorithms:** sorting, k-NN, matrix inversion, iteration.
- **Not generally embarrassingly parallel (EP).** (An exception: Random Forests – grow different trees within different processes.)
Motivation

Characteristics of machine learning (ML) algorithms:

- **Big Data**: in $n \times p$ (cases $\times$ features) dataset, both $n$ AND $p$ large.
- **Compute-intensive algorithms**: sorting, k-NN, matrix inversion, iteration.
- **Not generally embarrassingly parallel (EP)**. (An exception: Random Forests – grow different trees within different processes.)
- **Memory problems**: The computation may not fit on a single machine (esp. in R or GPUs).
Parallel ML: Desired Properties
Parallel ML: Desired Properties

• Simple, easily implementable.
Parallel ML: Desired Properties

- Simple, easily implementable. (And easily understood by non-techies.)
Parallel ML: Desired Properties

- Simple, easily implementable. (And easily understood by non-techies.)
- As general in applicability as possible.
Software Alchemy
Software Alchemy

alchemy:

*The medieval forerunner of chemistry... concerned particularly with attempts to convert base metals into gold... a seemingly magical process of transformation...*
Software Alchemy (cont’d.)
Software Alchemy (cont’d.)

Software Alchemy (cont’d.)

• Developed independently by (Matloff, JSS, 2013) and several others.
Software Alchemy (cont’d.)

- Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)

Software Alchemy (cont’d.)
Software Alchemy (cont’d.)

• Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)
• Not just Embarrassingly Parallel but also Embarrassingly Simple. :-)

Software Alchemy (cont’d.)

• Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)
• Not just Embarrassingly Parallel but also Embarrassingly Simple. :-)

Software Alchemy (cont’d.)

• Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)
• Not just Embarrassingly Parallel but also Embarrassingly Simple. :-)

Software Alchemy (cont’d.)

• Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)
• Not just Embarrassingly Parallel but also Embarrassingly Simple. :-)

Software Alchemy (cont’d.)
Software Alchemy (cont’d)
Software Alchemy (cont’d)

• Break the data into chunks, one chunk per process.
Software Alchemy (cont’d)

- Break the data into chunks, one chunk per process.
- Apply the procedure, e.g. neural networks (NNs), to each chunk,
Software Alchemy (cont’d)

- Break the data into chunks, one chunk per process.
- Apply the procedure, e.g. neural networks (NNs), to each chunk, using off-the-shelf SERIAL algorithms.
Software Alchemy (cont’d)

- Break the data into chunks, one chunk per process.
- Apply the procedure, e.g. neural networks (NNs), to each chunk, using off-the-shelf SERIAL algorithms.
- In regression case (continuous response variable) take final estimate as average of the chunked estimates.
Software Alchemy (cont’d)

- Break the data into chunks, one chunk per process.
- Apply the procedure, e.g. neural networks (NNs), to each chunk, using off-the-shelf SERIAL algorithms.
- In regression case (continuous response variable) take final estimate as average of the chunked estimates.
- In classification case (categorical response variable), do “voting.”
Break the data into chunks, one chunk per process.

Apply the procedure, e.g. neural networks (NNs), to each chunk, using off-the-shelf SERIAL algorithms.

In regression case (continuous response variable) take final estimate as average of the chunked estimates.

In classification case (categorical response variable), do “voting.”

If have some kind of parametric model (incl. NNs), can average the parameter values across chunks.
Theory
Theory

- Theorem:
Theory

- Theorem:

  Say rows of data matrix are i.i.d., output of procedure asymptotically normal. Then the Software Alchemy estimator is fully statistically efficient, i.e. has the same asymptotic variance.
• **Theorem:**

   *Say rows of data matrix are i.i.d., output of procedure asymptotically normal. Then the Software Alchemy estimator is fully statistically efficient, i.e. has the same asymptotic variance.*

• **Conditions of theorem could be relaxed.**
• Theorem:

Say rows of data matrix are i.i.d., output of procedure asymptotically normal. Then the Software Alchemy estimator is fully statistically efficient, i.e. has the same asymptotic variance.

• Conditions of theorem could be relaxed.
• Can do some informal analysis of speedup (next slide).
Theory (cont’d.)
Theory (cont’d.)

Say original algorithm has time complexity $O(n^c)$.
Theory (cont’d.)

Say original algorithm has time complexity $O(n^c)$.

- Then Software Alchemy time for $q$ processes is $O((n/q)^c) = O(n^c/q^c)$, a speedup of $q^c$.
Say original algorithm has time complexity $O(n^c)$. 

- Then Software Alchemy time for $q$ processes is $O((n/q)^c) = O(n^c/q^c)$, a speedup of $q^c$.
- If $c > 1$, get a superlinear speedup!
Say original algorithm has time complexity $O(n^c)$.

- Then Software Alchemy time for $q$ processes is $O((n/q)^c) = O(n^c/q^c)$, a speedup of $q^c$.
- If $c > 1$, get a superlinear speedup!
- In fact, even if the chunked computation is done serially, time is $O(q(n/q)^c) = O(n^c/q^{c-1})$, a speedup of $q^{c-1}$, a win if $c > 1$. 
Theory (cont.d)
Although...
Although...

- SA time is technically $\max_i \text{chunktime}_i$. If large variance, this would may result in speedup of $< q^c$. 
Although...

- SA time is technically $\max_i$ chunktime$_i$. If large variance, this would may result in speedup of $< q^c$.
- If number of features $p$ is a substantial fraction of $n$, the asymptotic convergence may not have quite kicked in yet.
Theory (cont.d)

Although...

- SA time is technically $\max_i \text{chunktime}_i$. If large variance, this would may result in speedup of $< q^c$.
- If number of features $p$ is a substantial fraction of $n$, the asymptotic convergence may not have quite kicked in yet.
- If full algorithm time is not just $O(f(n)))$ but $O(g(n, p))$, e.g. need $p \times p$ matrix inversion, then speedup is limited.
Although...

- SA time is technically $\max_i \text{chunktime}_i$. If large variance, this would may result in speedup of $< q^c$.
- If number of features $p$ is a substantial fraction of $n$, the asymptotic convergence may not have quite kicked in yet.
- If full algorithm time is not just $O(f(n))$ but $O(g(n, p))$, e.g. need $p \times p$ matrix inversion, then speedup is limited.
- Above analysis ignores overhead time for distributing the data.
Although...

- SA time is technically $\max_i \text{chunktime}_i$. If large variance, this would may result in speedup of $< q^c$.
- If number of features $p$ is a substantial fraction of $n$, the asymptotic convergence may not have quite kicked in yet.
- If full algorithm time is not just $O(f(n)))$ but $O(g(n, p))$, e.g. need $p \times p$ matrix inversion, then speedup is limited.
- Above analysis ignores overhead time for distributing the data. However, we advocate permanently distributed data anyway (Hadoop, Spark, our partools package).
Other Issues

• How many chunks? Having too many means chunks are too small for the asymptotics.

• Impact of tuning parameters.
  E.g. in neural nets, user must choose number of hidden layers, number of units per layer, etc. (Feng, 2016) has so many tuning parameters that the paper has a separate table to summarize them.

• Performance may depend crucially on the settings for those parameters.

• What if best tuning parameter settings for chunks are not the same as the best for the full data?
Other Issues

- How many chunks? Having too many means chunks are too small for the asymptotics.
Other Issues

- How many chunks? Having too many means chunks are too small for the asymptotics.
- Impact of tuning parameters.
  - E.g. in neural nets, user must choose number of hidden layers, number of units per layer, etc.
Other Issues

- How many chunks? Having too many means chunks are too small for the asymptotics.
- Impact of tuning parameters.
  - E.g. in neural nets, user must choose number of hidden layers, number of units per layer, etc. (Feng, 2016) has so many tuning parameters that the paper has a separate table to summarize them.
Other Issues

- How many chunks? Having too many means chunks are too small for the asymptotics.
- Impact of tuning parameters.
  - E.g. in neural nets, user must choose number of hidden layers, number of units per layer, etc. (Feng, 2016) has so many tuning parameters that the paper has a separate table to summarize them.
  - Performance may depend crucially on the settings for those parameters.
Other Issues

• How many chunks? Having too many means chunks are too small for the asymptotics.

• Impact of tuning parameters.
  • E.g. in neural nets, user must choose number of hidden layers, number of units per layer, etc. (Feng, 2016) has so many tuning parameters that the paper has a separate table to summarize them.
  • Performance may depend crucially on the settings for those parameters.
  • What if best tuning parameter settings for chunks are not the same as the best for the full data?
Empirical Investigation
Empirical Investigation

- Recommender systems
  - Famous example: Predict rating user $i$ would give to movie $j$, based on what $i$ has said about other movies, and what ratings $j$ got from other users.
  - Maximum Likelihood
  - Matrix factorization
  - k-NN model

- General ML applications
  - Logistic
  - Neural networks
  - Random forests
  - k-NN
Recommender Systems Datasets
Recommender Systems Datasets

- **Movie Lens:** User ratings of movies. We used the 1 million- and 20 million-record versions.
- **Book Crossings:** Book reviews, about 1 million records.
- **Jester:** Joke reviews, about 6 million records.
- No optimization of tuning parameters; focus is just on run time.
- No data cleaning.
- Timings on a quad core machine with hyperthreading.
Prediction Methods
Prediction Methods

- **MLE:** Rating of item $i$ by user $j$ is

  \[ Y_{ij} = \mu + \gamma' X_i + \alpha_i + \beta_j + \epsilon_{ij} \]

  where $X_i$ is a vector of covariates for user $i$ (e.g. age), and $\mu + \alpha_i$ and $\mu + \beta_j$ are overall means.

- **Nonnegative matrix factorization:** Find low-rank matrices $W$ and $H$ such that the matrix $A$ of all $Y_{ij}$, observed or not, is approx. $WH$. Fill in missing values from the latter.

- **k-Nearest Neighbor:** The $k$ users with ratings patterns closest to that of user $i$ and who have rated item $j$ are collected, and the average of their item-$j$ ratings computed.

Report: Scatter, train and test times, MAPE or prop. correct class.
NMF, MovieLens 20M
NMF, MovieLens 20M

<table>
<thead>
<tr>
<th>chunks</th>
<th>scatter</th>
<th>train.</th>
<th>pred.</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>-</td>
<td>34.046</td>
<td>0.346</td>
<td>0.649</td>
</tr>
<tr>
<td>2</td>
<td>13.49</td>
<td>18.679</td>
<td>0.647</td>
<td>0.647</td>
</tr>
<tr>
<td>4</td>
<td>21.86</td>
<td>10.444</td>
<td>1.113</td>
<td>0.656</td>
</tr>
</tbody>
</table>

Table: NMF Model, MovieLens Data, 20M

Approaching linear speedup.
k-NN, Jester Data

Superlinear speedup for 2, 4 chunks. Note improved accuracy, probably due to nonoptimal $k$ in full set.
k-NN, Jester Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time (sec)</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>259.601</td>
<td>4.79</td>
</tr>
<tr>
<td>2</td>
<td>76.440</td>
<td>4.60</td>
</tr>
<tr>
<td>4</td>
<td>58.133</td>
<td>4.36</td>
</tr>
<tr>
<td>8</td>
<td>81.185</td>
<td>3.89</td>
</tr>
</tbody>
</table>

Table: k-NN Model, Jester Data

Superlinear speedup for 2, 4 chunks.
Table: k-NN Model, Jester Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time (sec)</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>259.601</td>
<td>4.79</td>
</tr>
<tr>
<td>2</td>
<td>76.440</td>
<td>4.60</td>
</tr>
<tr>
<td>4</td>
<td>58.133</td>
<td>4.36</td>
</tr>
<tr>
<td>8</td>
<td>81.185</td>
<td>3.89</td>
</tr>
</tbody>
</table>

Superlinear speedup for 2, 4 chunks. Note improved accuracy, probably due to nonoptimal $k$ in full set.
MLE, Book Crossings
MLE, Book Crossings

<table>
<thead>
<tr>
<th>chunks</th>
<th>scatter</th>
<th>train.</th>
<th>pred.</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>-</td>
<td>1114.155</td>
<td>0.455</td>
<td>2.67</td>
</tr>
<tr>
<td>2</td>
<td>5.101</td>
<td>685.757</td>
<td>0.455</td>
<td>2.72</td>
</tr>
<tr>
<td>4</td>
<td>11.134</td>
<td>423.018</td>
<td>1.173</td>
<td>2.77</td>
</tr>
<tr>
<td>8</td>
<td>10.918</td>
<td>246.668</td>
<td>1.470</td>
<td>2.82</td>
</tr>
</tbody>
</table>

Table: MLE Model, Book Crossings Data

Sublinear speedup due to matrix inversion, but still faster at 8 chunks.
MLE, MovieLens Data

<table>
<thead>
<tr>
<th>Speedup</th>
<th>full</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean abs. error</td>
<td>99.028</td>
<td>0.267</td>
<td>0.710</td>
<td>0.737</td>
</tr>
</tbody>
</table>

Table: MLE Model, MovieLens Data, 1M

Speedup limited due to matrix inversion.
MLE, MovieLens Data

<table>
<thead>
<tr>
<th>chunks</th>
<th>scatter</th>
<th>train.</th>
<th>pred.</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>-</td>
<td>99.028</td>
<td>0.267</td>
<td>0.710</td>
</tr>
<tr>
<td>2</td>
<td>4.503</td>
<td>100.356</td>
<td>0.317</td>
<td>0.737</td>
</tr>
<tr>
<td>4</td>
<td>2.596</td>
<td>73.055</td>
<td>0.469</td>
<td>0.752</td>
</tr>
<tr>
<td>8</td>
<td>8.408</td>
<td>100.356</td>
<td>0.483</td>
<td>0.764</td>
</tr>
</tbody>
</table>

*Table: MLE Model, MovieLens Data, 1M*

Speedup limited due to matrix inversion.
General ML Applications
General ML Applications

Methods: Logistic regression; neural nets; k-NN; random forests.

Datasets:

- **NYC taxi data**: Trip times, fares, location etc.
- **Forest cover data**: Predict type of ground cover from satellite data.
- **Last.fm**: Popularity of songs.
Logit, NYC Taxi Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time</th>
<th>prop. correct class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>38.753</td>
<td>0.694</td>
</tr>
<tr>
<td>4</td>
<td>23.501</td>
<td>0.694</td>
</tr>
<tr>
<td>8</td>
<td>14.320</td>
<td>0.694</td>
</tr>
</tbody>
</table>

Have matrix inversion here too, but still getting speedup at 8 threads (and up to 32 on another machine, 16 cores).
Fast, General Parallel Computation for Machine Learning

Robin Elizabeth Yancey and Norm Matloff
University of California at Davis

Logit, NYC Taxi Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time</th>
<th>prop. correct class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>40.641</td>
<td>0.694</td>
</tr>
<tr>
<td>2</td>
<td>38.753</td>
<td>0.694</td>
</tr>
<tr>
<td>4</td>
<td>23.501</td>
<td>0.694</td>
</tr>
<tr>
<td>8</td>
<td>14.320</td>
<td>0.694</td>
</tr>
</tbody>
</table>

Table: Logistic Model, NYC Taxi Data

Have matrix inversion here too, but still getting speedup at 8 threads (and up to 32 on another machine, 16 cores).
NNs, Last.fm Data
NNs, Last.fm Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>486.259</td>
<td>221.41</td>
</tr>
<tr>
<td>2</td>
<td>325.567</td>
<td>211.94</td>
</tr>
<tr>
<td>4</td>
<td>254.306</td>
<td>210.15</td>
</tr>
<tr>
<td>8</td>
<td>133.495</td>
<td>221.41</td>
</tr>
</tbody>
</table>

Table: Neural nets, Last.fm data, 5 hidden layers

Sublinear, but still improving at 8 chunks. Better prediction with 2, 4 chunks; tuning thus suboptimal in full case.
k-NN, NYC Taxi Data

Superlinear speedup at 4 chunks, with better prediction error; k too large in full?
k-NN, NYC Taxi Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time</th>
<th>mean abs. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>87.463</td>
<td>456.00</td>
</tr>
<tr>
<td>2</td>
<td>48.110</td>
<td>451.08</td>
</tr>
<tr>
<td>4</td>
<td>25.75</td>
<td>392.13</td>
</tr>
<tr>
<td>8</td>
<td>17.413</td>
<td>424.36</td>
</tr>
</tbody>
</table>

Table: k-NN, NYC Taxi Data

Superlinear speedup at 4 chunks, with better prediction error; $k$ too large in full?
RF, Forest Cover Data

---

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time</th>
<th>prob. correct class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>841.884</td>
<td>0.955</td>
</tr>
<tr>
<td>2</td>
<td>485.171</td>
<td>0.941</td>
</tr>
<tr>
<td>4</td>
<td>236.518</td>
<td>0.919</td>
</tr>
<tr>
<td>6</td>
<td>194.803</td>
<td>0.911</td>
</tr>
</tbody>
</table>

Table: Random Forests, Forest Cover Data

As noted, EP anyway, but still interesting.
### RF, Forest Cover Data

<table>
<thead>
<tr>
<th># of chunks</th>
<th>time</th>
<th>prob. correct class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>841.884</td>
<td>0.955</td>
</tr>
<tr>
<td>2</td>
<td>485.171</td>
<td>0.941</td>
</tr>
<tr>
<td>4</td>
<td>236.518</td>
<td>0.919</td>
</tr>
<tr>
<td>6</td>
<td>194.803</td>
<td>0.911</td>
</tr>
</tbody>
</table>

**Table:** Random Forests, Forest Cover Data

As noted, EP anyway, but still interesting.
GPU Settings

• In a multi-GPU setting, chunking is a natural solution, hence SA.
• If GPU memory insufficient, use SA serially. Still may get a speedup (per earlier slide).
GPU Settings

Use of Software Alchemy with GPUs.

- In a multi-GPU setting, chunking is a natural solution, hence SA.
- If GPU memory insufficient, use SA serially. Still may get a speedup (per earlier slide).
Conclusions, Comments

• Software Alchemy extremely simple, statistically valid — same statistical accuracy.
• Generally got linear or even superlinear speedup on most recommender systems and other ML algorithms.
• We used our partools package, which is based on a "Leave It There" philosophy: Keep an object distributed as long as possible, including as a distributed file. Thus no scatter time needed.
Conclusions, Comments

- Software Alchemy extremely simple, statistically valid — same statistical accuracy.
- Generally got linear or even superlinear speedup on most recommender systems and other ML algorithms.
- We used our `partools` package, which is based on a “Leave It There” philosophy: Keep an object distributed as long as possible, including as a distributed file. Thus no scatter time needed.