Fast, General Parallel Computation for Machine Learning

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

P2PS Workshop, ICPP 2018

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

Outline

Outline

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

Motivation

Motivation

Characteristics of machine learning (ML) algorithms:

Motivation

Characteristics of machine learning (ML) algorithms:

- Big Data: in $n \times p$ $(\text{cases} \ \times \text{features})$ 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 $n \times p$ $(\text{cases} \ \times \text{features})$ 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 $n \times p$ $(\text{cases} \ \times \text{features})$ 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 $n \times p$ $(\text{cases } \times \text{features})$ 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).

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

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.)

- "Alchemical": Converts non-EP problems to *statistically equivalent* EP problems.

Software Alchemy (cont'd.)

- "Alchemical": Converts non-EP problems to *statistically equivalent* EP problems.
- Developed independently by (Matloff, JSS, 2013) and several others.

Software Alchemy (cont'd.)

- "Alchemical": Converts non-EP problems to *statistically equivalent* EP problems.
- Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)

Software Alchemy (cont'd.)

- "Alchemical": Converts non-EP problems to *statistically equivalent* EP problems.
- Developed independently by (Matloff, JSS, 2013) and several others. EP: No programming challenge. :-)
- Not just Embarrassingly Parallel but also Embarrassingly Simple. :-)

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

Software Alchemy (cont'd)

Software Alchemy (cont'd)

- Break the data into chunks, one chunk per process.

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

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."

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

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."
- 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.

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.

- Conditions of theorem could be relaxed.

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.

- 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.

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.
- If $c > 1$, get a superlinear speedup!

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.
- 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)

Theory (cont.d)

Although...

Theory (cont.d)

Although...

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

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.

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.

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.
- Above analysis ignores overhead time for distributing the data.

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.
- Above analysis ignores overhead time for distributing the data. However, we advocate permanently distributed data anyway (Hadoop, Spark, our **partools** package).

Other Issues

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

chunks	scatter	train.	pred.	mean abs. error
full	-	34.046	0.346	0.649
2	13.49	18.679	0.647	0.647
4	21.86	10.444	1.113	0.656

Table: NMF Model, MovieLens Data, 20M

Approaching linear speedup.

k-NN, Jester Data

k-NN, Jester Data

# of chunks	time (sec)	mean abs. error
full	259.601	4.79
2	76.440	4.60
4	58.133	4.36
8	81.185	3.89

Table: k-NN Model, Jester Data

Superlinear speedup for 2, 4 chunks.

k-NN, Jester Data

# of chunks	time (sec)	mean abs. error
full	259.601	4.79
2	76.440	4.60
4	58.133	4.36
8	81.185	3.89

Table: k-NN Model, Jester Data

Superlinear speedup for 2, 4 chunks. Note improved accuracy, probably due to nonoptimal *k* in full set.

MLE, Book Crossings

MLE, Book Crossings

chunks	scatter	train.	pred.	mean abs. error
full	-	1114.155	0.455	2.67
2	5.101	685.757	0.455	2.72
4	11.134	423.018	1.173	2.77
8	10.918	246.668	1.470	2.82

Table: MLE Model, Book Crossings Data

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

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

MLE, MovieLens Data

MLE, MovieLens Data

chunks	scatter	train.	pred.	mean abs. error
full	-	99.028	0.267	0.710
2	4.503	100.356	0.317	0.737
4	2.596	73.055	0.469	0.752
8	8.408	100.356	0.483	0.764

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

Logit, NYC Taxi Data

# of chunks	time	prop. correct class.
full	40.641	0.694
2	38.753	0.694
4	23.501	0.694
8	14.320	0.694

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).

Robin Elizabeth Yancey and **Norm Matloff**
University of California at Davis

NNs, Last.fm Data

NNs, Last.fm Data

# of chunks	time	mean abs. error
full	486.259	221.41
2	325.567	211.94
4	254.306	210.15
8	133.495	221.41

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

k-NN, NYC Taxi Data

# of chunks	time	mean abs. error
full	87.463	456.00
2	48.110	451.08
4	25.75	392.13
8	17.413	424.36

Table: k-NN, NYC TaxiData

Superlinear speedup at 4 chunks, with better prediction error; k too large in full?

RF, Forest Cover Data

RF, Forest Cover Data

# of chunks	time	prob. correct class.
full	841.884	0.955
2	485.171	0.941
4	236.518	0.919
6	194.803	0.911

Table: Random Forests, Forest Cover Data

As noted, EP anyway, but still interesting.

GPU Settings

GPU Settings

Use of Software Alchemy with GPUs.

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

Conclusions, Comments

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.