3 Fun Machine Learning Problems for Big Data

John Langford (Yahoo! $\longrightarrow$ Microsoft)

STOC Workshop on Big Data and Streaming Algorithms, May 19, 2012

What's big data?

The practical viewpoint:

1. $O(n^2)$ algorithm feasible: small data
2. Fits on one machine: medium data
3. Doesn't fit on one machine: big data

What's big data?

The practical viewpoint:

1. $O(n^2)$ algorithm feasible: small data
2. Fits on one machine: medium data
3. Doesn't fit on one machine: big data

An example: predicting which ad is interesting. [ACDL11]

2.1T sparse features

17B Examples

16M parameters

1K nodes

What's big data?

The practical viewpoint:

1. $O(n^2)$ algorithm feasible: small data
2. Fits on one machine: medium data
3. Doesn't fit on one machine: big data

An example: predicting which ad is interesting. [ACDL11]

2.1T sparse features

17B Examples

16M parameters

1K nodes

We train an optimal linear predictor in 70 minutes = 500M features/second: faster than the IO bandwidth of a single machine⇒ we beat <u>all possible</u> single machine linear learning algorithms.

Algorithm

The algorithm (sketch: many details):

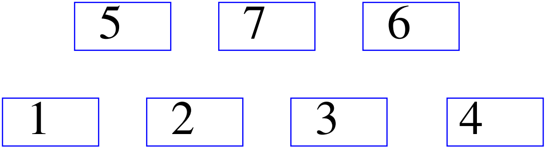
1. On each node use online learning independently to find a parameter vector.
2. Use AllReduce to average the weights.
3. On each node, compute the sum of the gradient for each example.
4. Use AllReduce to add the gradients at each node.
5. Use L-BFGS to update the weight vector, goto (3) 20 times.

Algorithm

The algorithm (sketch: many details):

1. On each node use online learning independently to find a parameter vector.
2. Use AllReduce to average the weights.
3. On each node, compute the sum of the gradient for each example.
4. Use AllReduce to add the gradients at each node.
5. Use L-BFGS to update the weight vector, goto (3) 20 times.

Allreduce initial state

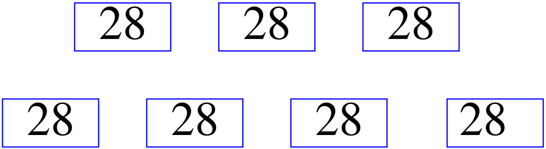


Algorithm

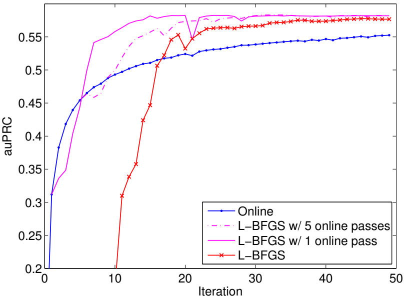
The algorithm (sketch: many details):

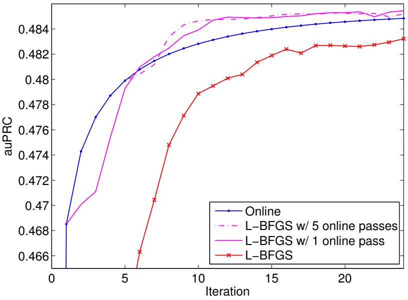
1. On each node use online learning independently to find a parameter vector.
2. Use AllReduce to average the weights.
3. On each node, compute the sum of the gradient for each example.
4. Use AllReduce to add the gradients at each node.
5. Use L-BFGS to update the weight vector, goto (3) 20 times.

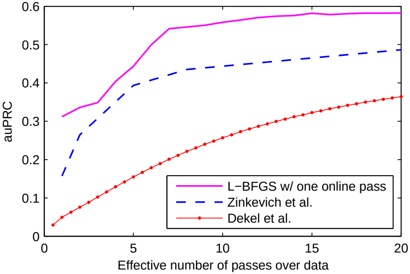
Allreduce final state



Empirical Results







Analysis

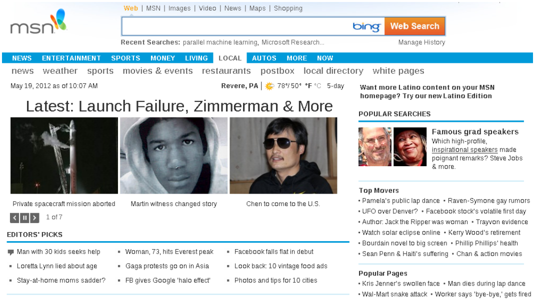
Theorem: ??

Analysis

Theorem: ??

We can prove that the algorithm makes sense as an optimization. We don't know how to prove anything meaningful about generalization.

Where does data come from?



Repeatedly:

1. A user comes to ~~Yahoo!~~MSN (with history of previous visits, IP address, data related to his ~~Yahoo!~~MSN account)
2. ~~Yahoo!~~MSN chooses information to present (urls, ads, news stories)
3. The user reacts to the presented information (clicks on something, clicks, comes back and clicks again,...)

~~Yahoo!~~MSN wants to interactively choose content and use the observed feedback to improve future content choices.

The Contextual Bandit Setting

For $t = 1, \ldots, T$:

1. The world produces some context $x \in X$
2. The learner chooses an action $a \in A$
3. The world reacts with reward $r_a \in [0, 1]$

Goal: Efficiently compete with a large reference class of policies $\Pi = \{\pi : X \to A\}$:

$$\text{Regret} = \max_{\pi \in \Pi} \text{average}_t (r_{\pi(x)} - r_a)$$

The Contextual Bandit Setting

For $t = 1, \ldots, T$:

1. The world produces some context $x \in X$
2. The learner chooses an action $a \in A$
3. The world reacts with reward $r_a \in [0, 1]$

Goal: Efficiently compete with a large reference class of policies $\Pi = \{\pi : X \to A\}$:

$$\text{Regret} = \max_{\pi \in \Pi} \text{average}_t (r_{\pi(x)} - r_a)$$

Examples of Π:

- Context-free policies prescribing the same treatment to all.
- A machine learning system (e.g., all linear predictors)
- A discrete set based on domain-specific hunches or hypotheses

Randomized UCB

[DHKKLRZ11]

Given an oracle finding $\arg\max_{\pi \in \Pi} \sum_{(x,\vec{r})} r_{\pi(x)}$:

Randomized_UCB

For each $t = 1, 2, \ldots$

1. Choose distribution P over Π minizing variance for empirical good policies and limiting variance for empirical bad policies.
2. observe x
3. Let $p(a) =$ fraction of P choosing a given x.
4. Choose $a \sim p$ and observe reward r

Theorem: For all sets of policies Π, for all distributions $D(x, \vec{r})$, if the world is IID w.r.t. D, with high probability Randomized_UCB has regret $O\left(\sqrt{\frac{K \ln |\Pi|}{T}}\right)$ in time $\mathrm{Poly}(t, K, \log |\Pi|)$

All other approaches require $O(|\Pi|)$ time!

The Problem

...It uses the ellipsoid algorithm for convex programming.

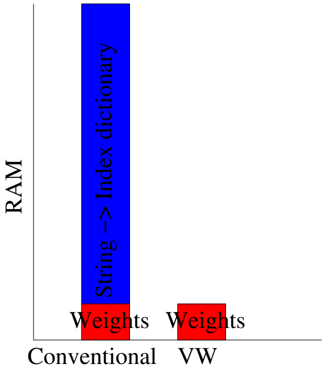
The Problem

...It uses the ellipsoid algorithm for convex programming.

The grand challenge: Can we make a quasilinear time algorithm given an optimization oracle?

Some evidence: We succeeded with active learning [BHLZ10] [KL11]

Feature Hashing



Most algorithms use a hashmap to change a word into an index for a weight.
A hash function takes almost no RAM, is x10 faster, and is easily parallelized.

The spam example [WALS09]

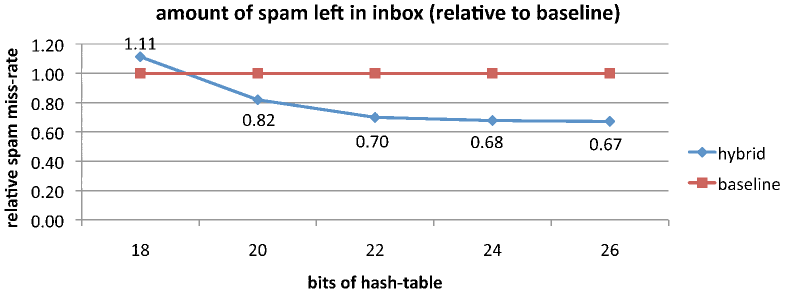
1. $3.2 * 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 * 10^6$ unique features.

Construct a personalized spam filter using hashing:
$\langle w, \phi(x) \rangle + \langle w, \phi_u(x) \rangle$

The spam example [WALS09]

1. $3.2 * 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 * 10^6$ unique features.

Construct a personalized spam filter using hashing:
$\langle w, \phi(x) \rangle + \langle w, \phi_u(x) \rangle$



(baseline = global only predictor)

Hashing in learning $\neq$ linear projection

Experiment 1: Hash features, then Learn
Experiment 2: Learn, then Hash features and weights.
Under linear projection these are equivalent.

Hashing in learning $\neq$ linear projection

Experiment 1: Hash features, then Learn

Experiment 2: Learn, then Hash features and weights.

Under linear projection these are equivalent.

The truth: 1 beats 2.

Label	Hashed Features
1	1 1 0
0	1 0 1
Weights	0.33 0.67 -0.33

Average Squared loss: 0

Label	Features	Hashed
1	1 1 0 0	1 1 0
0	0 0 1 1	1 0 1
Weights	0.5 0.5 0 0	0.5 0.5 0

Average Squared Loss 0.25

The Bloom Filter(ish) view [SPDLSV09]

N identical features $+$ infinite examples $\Rightarrow$ performance like Bloom filter with N hashes.

The Bloom Filter(ish) view [SPDLSV09]

N identical features + infinite examples $\Rightarrow$ performance like Bloom filter with N hashes.
All N features unique and necessary + infinite examples $\Rightarrow$ need multiple hashes of each feature like Bloom filter.

The Bloom Filter(ish) view [SPDLSV09]

N identical features + infinite examples ⇒ performance like Bloom filter with N hashes.

All N features unique and necessary + infinite examples ⇒ need multiple hashes of each feature like Bloom filter.

Reality is messier: finite examples + problem dependent partial redundancy in features.

The Bloom Filter(ish) view [SPDLSV09]

N identical features + infinite examples ⇒ performance like Bloom filter with N hashes.

All N features unique and necessary + infinite examples ⇒ need multiple hashes of each feature like Bloom filter.

Reality is messier: finite examples + problem dependent partial redundancy in features.

What is an efficient and effective algorithm to determine the optimal hash size online?

What is an efficient and effective algorithm to determine the optimal redundancy online?

Other Problems

\$1K reward: Efficient Robust Conditional Probability Estimation
`http://hunch.net/?p=1253`
\$0.5K reward: Cross Validation Analysis
`http://hunch.net/?p=29`

Papers

[ACDL11] Alekh Agarwal, Olivier Chapelle, Miroslav Dudik, John Langford, A Reliable Effective Terascale Linear Learning System, http://arxiv.org/abs/1110.4198

[DHKKLRZ11] Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang, Efficient Optimal Leanring for Contextual Bandits, UAI 2011.

[BHLZ10] Alina Beygelzimer, Daniel Hsu, John Langford, and Tong Zhang Agnostic Active Learning Without Constraints NIPS 2010.

[KL11] Nikos Karampatziakis and John Langford, Importance Weight Aware Gradient Updates, UAI 2011.

[WALS09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, Josh Attenberg, Feature Hashing for Large Scale Multitask Learning, ICML 2009

[SPDLSV09] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan, Hash Kernels for Structured Data, AISTAT 2009 and JMLR 2009.