3 Fun Machine Learning Problems for Big Data

John Langford (Yahoo! → 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 $\Rightarrow$ we beat all possible single machine linear learning algorithms.
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

\[ 2 \ 3 \ 4 \]

Allreduce final state

\[ 28 \ 28 \ 28 \ 28 \]
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

![Graphs showing empirical results with respect to iteration and effective number of passes over data. The graphs compare various optimization methods, including Online, L-BFGS with 5 online passes, L-BFGS with 1 online pass, and L-BFGS. The x-axis represents iteration and effective number of passes over data, while the y-axis shows the auPRC (Area Under the Precision-Recall Curve). The graphs illustrate the performance of these methods over different iterations and passes.](https://example.com/graphs)
Analysis

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

Theorem: ??
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 $\Pi = \{\pi : X \rightarrow A\}$:

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

Examples of $\Pi$:

- 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 \( \Pi \) minimizing 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 \( \Pi \), 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 \( \text{Poly}(t, K, \log |\Pi|) \)

All other approaches require \( O(|\Pi|) \) time!
...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]
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. $3.2 \times 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 \times 10^6$ 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. $3.2 \times 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 \times 10^6$ unique features.

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

(amount of spam left in inbox (relative to baseline))

(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.

<table>
<thead>
<tr>
<th>Label</th>
<th>Hashed Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>

Weights: 0.33 0.67 -0.33

Average Squared loss: 0

<table>
<thead>
<tr>
<th>Label</th>
<th>Features</th>
<th>Hashed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1 0 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0</td>
<td>0 0 1 1</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>

Weights: 0.5 0.5 0 0

Average Squared Loss 0.25
$N$ identical features + infinite examples $\Rightarrow$ performance like Bloom filter with $N$ hashes.
$N$ 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.
$N$ 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.
Reality is messier: finite examples + problem dependent partial redundancy in features.
The Bloom Filter(ish) view [SPDLSV09]

\[ N \text{ identical features} + \text{infinite examples} \Rightarrow \text{performance like Bloom filter with } N \text{ hashes.} \]

All \( N \) features unique and necessary + infinite examples \( \Rightarrow \) 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
[WALS09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, Josh Attenberg, Feature Hashing for Large Scale Multitask Learning, ICML 2009