Sibyl: A system for large scale supervised machine learning

Kevin Canini, Tushar Chandra, Eugene Ie, Jim McFadden, Ken Goldman, Mike Gunter, Jeremiah Harmsen, Kristen LeFevre, Dmitry Lepikhin, Tomas Lloret Llinares, Indraneel Mukherjee, Fernando Pereira, Josh Redstone, Tal Shaked, Yoram Singer
Goal

- Users respond differently to different information in different contexts
- Learn model of what information gets the best user response in different contexts
- ... use model do decide what to present
Uses of machine learning

- Improve relevance
- Improve site monetization
- Reduce spam
- Improve advertiser return on investment
- ... etc ...
Problem scale

- 100M views per day (or more)
- Businesses worth $100M (or more)
Problem scope

- There are many such problems at Google
  - Search, YouTube, Gmail, Android, G+, etc
  - Relevance, monetization, spam, etc

- ML typically generates 10+% improvements

  => This is becoming an industry "best practice"

- 1% improvement is a big deal, e.g.:
  - Improves relevance for millions of users
  - Millions of dollars of revenue

  => accuracy is important
Machine learning architecture

- User
- Server
- Databases
- "Joined" logs
- Impression log
- Interaction log
- ... etc ...
- Machine learned model
- Machine learning system
- Analysis tools
Sibyl spec

- 100s of TB of joined logs (uncompressed)
- 100s of billions of training examples
- 100 billion unique features, 10s or 100s per example

=> Must train accurate models
   (should be able to train 100s of models Google-wide)

=> Need highly parallel algos that converge quickly
   (Algos should leverage Google's scalable infrastructure)
Results overview

Built principled large scale supervised ML system
- Using theoretically sound algorithms
- To solve internet scale problems
- Using reasonable resources
- For multiple loss functions and regularizations

Used techniques that are well known to the systems community
- MapReduce for scalability
- Multiple cores and threads per computer for efficiency
- Google File System (GFS) to store lots of data
- An integerized column-oriented data format for compression & performance
**Parallel Boosting Algorithm**
*(Collins, Schapire, Singer 2001)*

- Iterative algorithm, each iteration improves model
- Number of iterations to get within $\epsilon$ of the optimum:
  \[ \log(m)/\epsilon^2 \]
- Updates correlated with gradients, but not a gradient algorithm
- Self-tuned step size, large when instances are sparse
Parallel Boosting Algorithm
(Collins, Schapire, Singer 2001)

**Input:** Training set \( S = \{(x_i, y_i) \mid x_i \in \{0, 1\}^n, y_i \in \{-1, +1\}\}_{i=1}^m \)

**Parameters:** Regularization \( \lambda \); Number of rounds \( T \)

For \( t = 1 \) to \( T \)

// Compute importance weights

For \( i = 1 \) to \( m \)

Set \( q^t(i) = \frac{1}{1 + e^{y_i(w^t \cdot x_i)}} \)

For \( j = 1 \) to \( n \)

// Compute features statistics

\( \mu^+_j = \sum_{i : y_i = +1 \land x_{i,j} = 1} q^t(i) \)

\( \mu^-_j = \sum_{i : y_i = -1 \land x_{i,j} = 1} q^t(i) \)

// Compute change in weights

\( \delta^t_j = \rho \log \frac{\mu^+_j}{\mu^-_j} \)

\( w^{t+1} = w^t + \delta^t \)
Parallel Boosting Algorithm
(Collins, Schapire, Singer 2001)

\[ q(i) = \frac{1}{1 + \exp(y_i(w \cdot x_i))} \]

\[
\begin{align*}
\mu^+_j &= \sum_{i : y_i = 1 \land x_{ij} = 1} q(i) \\
\mu^-_j &= \sum_{i : y_i = -1 \land x_{ij} = 1} q(i) \\
w_j &= \eta \log \left( \frac{\mu^+_j}{\mu^-_j} \right)
\end{align*}
\]
Properties of parallel boosting

Embarrassingly parallel:
1. Computes feature correlations for each example in parallel
2. Feature are updated in parallel

We need to “shuffle” the outputs of Step 1 for Step 2

Step size inversely proportional to number of active features per example
● Not total number of features
● Good for sparse training data

Extensions
● Add regularization
● Support other loss functions
A brief introduction to MapReduce

Programming model for processing large data sets

- Proven model and implementation
Implementing parallel boosting

+ Embarrassingly parallel
+ Stateless, so robust to transient data errors
+ Each model is consistent, sequence of models for debugging

- 10-50 iterations to converge
Some observations

We typically train multiple models
  • To explore different types of features
    • Don’t read unnecessary features
  • To explore different levels of regularization
    • Amortize fixed costs across similar models
  • Computers have lots of RAM
    • Store the model and training stats in RAM at each worker
  • Computers have lots of cores
    • Design for multi-core
  • Training data is highly compressible
Instead of a row-oriented data store ...

![Diagram showing a different data structure](image-url)
Design principle: use column-oriented data store
Design principle: use column-oriented data store

Column for each field
Each learner only reads relevant columns

Benefits
  • Learners read much less data
  • Efficient to transform fields
  • Data compresses better
Design principle: use model sets

- Train multiple similar models together
- Benefit: amortize fixed costs across models
  - Cost of reading training data
  - Cost of transforming data
- Downsides
  - Need more RAM
  - Shuffle more data
Design principle: “Integerize” features

• Each column has its own dense integer space
• Encode features in decreasing order of frequency
• Variable-length encoding of integers
• Benefits:
  • Training data compression
  • Store in-memory model and statistics as arrays rather than hash tables
    • Compact, faster
Design principle: store model and stats in RAM

- Each worker keeps in RAM
  - A copy of the previous model
  - Learning statistics for its training data
- Boosting requires $O(10 \text{ bytes})$ per feature
- Possible to handle billions of features
Design principle: optimize for multi-core

- Share model across cores
- MapReduce optimizations
  - Multi-shard combiners
    - Share training statistics across cores
## Training data

<table>
<thead>
<tr>
<th>Product</th>
<th>Examples</th>
<th>Compressed Raw data</th>
<th>Training data</th>
<th>Compression</th>
<th>Features per example</th>
<th>bytes per feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>59.9B</td>
<td>9.98TB</td>
<td>2.00TB</td>
<td>4.99x</td>
<td>54.9</td>
<td>0.67</td>
</tr>
<tr>
<td>B</td>
<td>7.6B</td>
<td>2.67TB</td>
<td>0.71TB</td>
<td>3.78x</td>
<td>94.9</td>
<td>1.07</td>
</tr>
<tr>
<td>C</td>
<td>197.5B</td>
<td>66.66TB</td>
<td>15.54TB</td>
<td>4.29x</td>
<td>77.7</td>
<td>1.11</td>
</tr>
<tr>
<td>D</td>
<td>129.1B</td>
<td>61.93TB</td>
<td>17.24TB</td>
<td>3.59x</td>
<td>100.57</td>
<td>1.46</td>
</tr>
</tbody>
</table>
## Processing throughput

<table>
<thead>
<tr>
<th>Product</th>
<th>Examples</th>
<th>Features per example</th>
<th>Processing cores</th>
<th>Iteration time (secs)</th>
<th>Number of models</th>
<th>#features per sec per core</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>59.9B</td>
<td>26.59</td>
<td>195</td>
<td>2471</td>
<td>1</td>
<td>3.3M</td>
</tr>
<tr>
<td>B</td>
<td>7.6B</td>
<td>27.18</td>
<td>290</td>
<td>599</td>
<td>2</td>
<td>2.4M</td>
</tr>
<tr>
<td>C</td>
<td>197.5B</td>
<td>35.09</td>
<td>700</td>
<td>4523</td>
<td>1</td>
<td>2.2M</td>
</tr>
<tr>
<td>D</td>
<td>129.1B</td>
<td>54.61</td>
<td>970</td>
<td>3150</td>
<td>1</td>
<td>2.3M</td>
</tr>
</tbody>
</table>
# Concurrency

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Time per iteration (secs)</th>
<th>Cost per iteration (core x secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 cores x 10 machines</td>
<td>15000</td>
<td>60000</td>
</tr>
<tr>
<td>8 cores x 10 machines</td>
<td>7500</td>
<td>60000</td>
</tr>
<tr>
<td>12 cores x 10 machines</td>
<td>4500</td>
<td>54000</td>
</tr>
<tr>
<td>16 cores x 10 machines</td>
<td>3900</td>
<td>62400</td>
</tr>
</tbody>
</table>
## Impact of L1

<table>
<thead>
<tr>
<th>Product</th>
<th>Number of features</th>
<th>Number of non-zero features</th>
<th>Fraction of non-zero features</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>868M</td>
<td>20.1M</td>
<td>2.31%</td>
</tr>
<tr>
<td>B</td>
<td>333M</td>
<td>7.9M</td>
<td>2.37%</td>
</tr>
<tr>
<td>C</td>
<td>1762M</td>
<td>251.8M</td>
<td>14.29%</td>
</tr>
<tr>
<td>D</td>
<td>2172M</td>
<td>371.6M</td>
<td>17.11%</td>
</tr>
</tbody>
</table>
Other Sibyl features

- Multiple loss functions
- Sophisticated regularization scheme
- Template exploration
- Dynamic stepping for faster convergence
- Online setting
Lesson learnt (future direction):
Focus on ease of use

- Cleanly integrated machine learning pipeline
  - Log joining, training, serving, analysis

- Tools for analyzing TB of data

- Incorporate best practices
  - e.g., catch training/serving skew

- Incorporate other machine learning methods
  - e.g., unsupervised learning