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

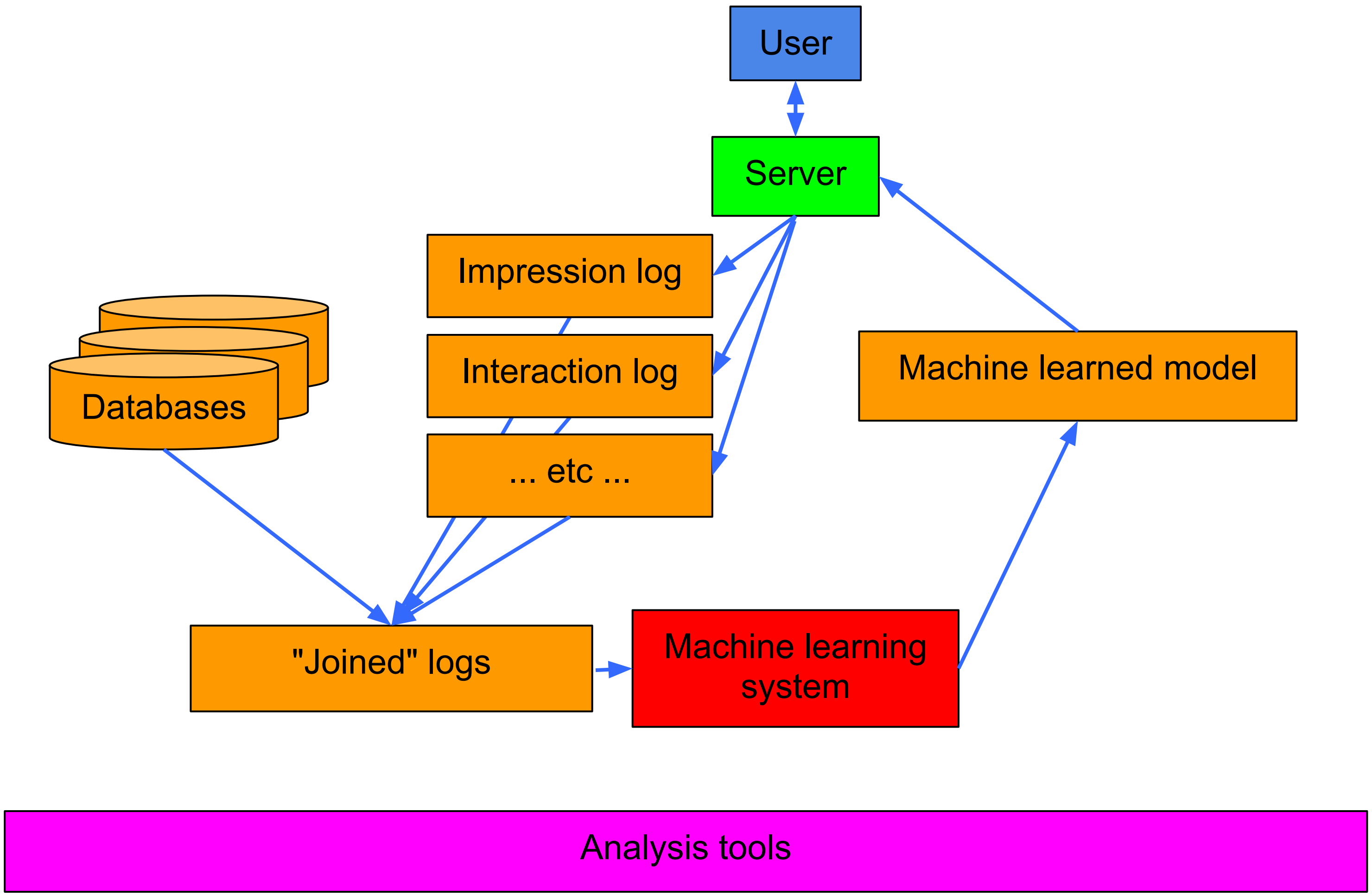
- ML typically generates 10+% improvements

=> 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



Sibyl spec

Google

- 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 ϵ 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)

$\textsc{Input:}$ Training set $S = \{(\boldsymbol{x}_i, y_i) \mid \boldsymbol{x}_i \in \{0,1\}^n, y_i \in \{-1,+1\}\}_{i=1}^m$

$\textsc{Parameters:}$ Regularization λ ; Number of rounds T

$\textsc{For}$ $t = 1$ to T

// *Compute importance weights*

$\textsc{For}$ $i = 1$ to m

$\textsc{Set}$ $q^t(i) = \frac{1}{1+e^{y_i(\boldsymbol{w}^t \cdot \boldsymbol{x}_i)}}$

$\textsc{For}$ $j = 1$ to n

// *Compute features statistics*

$$\mu_j^+ = \sum_{i:y_i=+1 \wedge x_{i,j}=1} q^t(i) \qquad \mu_j^- = \sum_{i:y_i=-1 \wedge x_{i,j}=1} q^t(i)$$

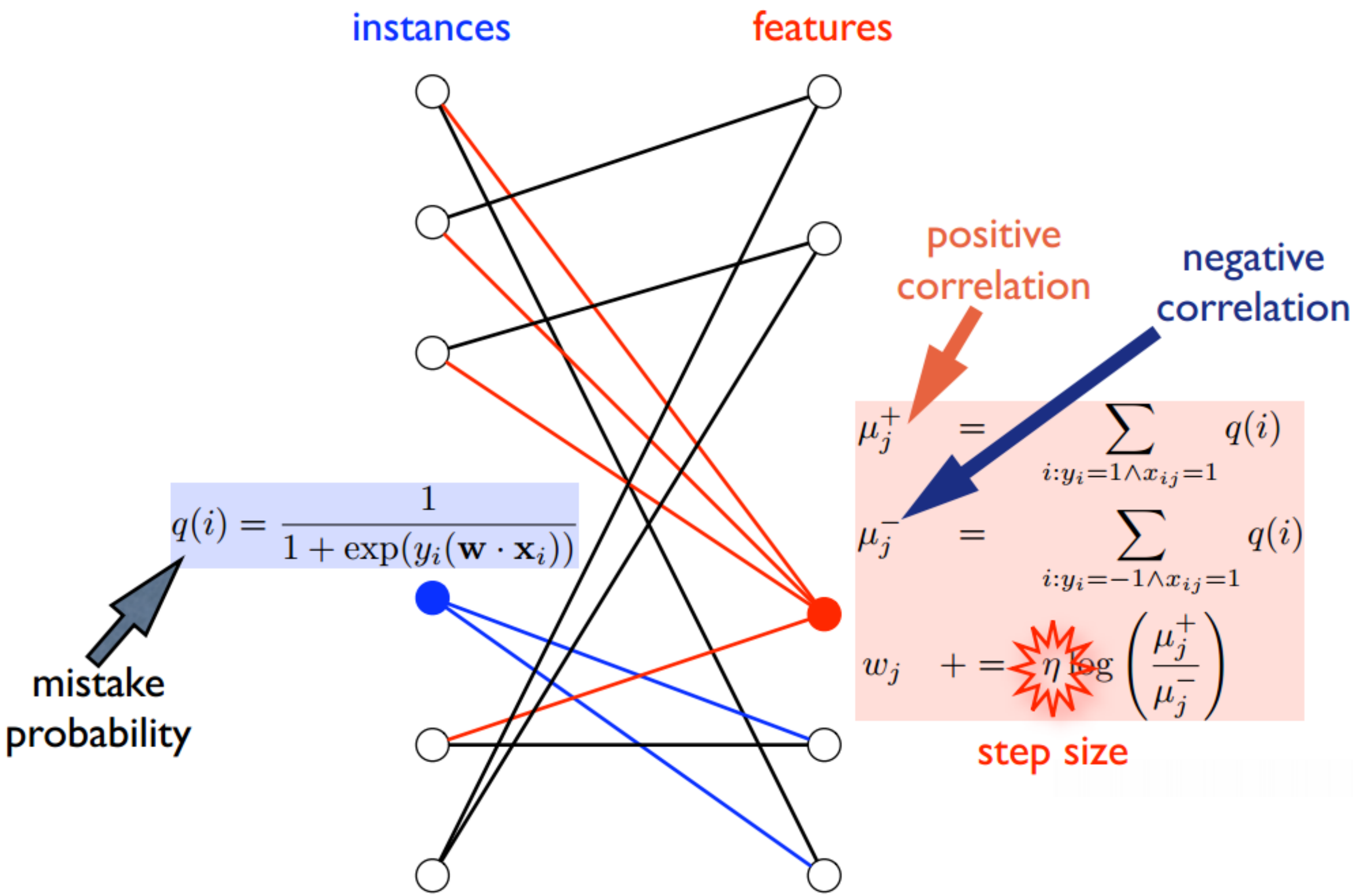
// *Compute change in weights*

$$\delta_j^t = \rho \log \frac{\mu_j^+}{\mu_j^-}$$

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \boldsymbol{\delta}^t$$

Parallel Boosting Algorithm

(Collins, Schapire, Singer 2001)



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

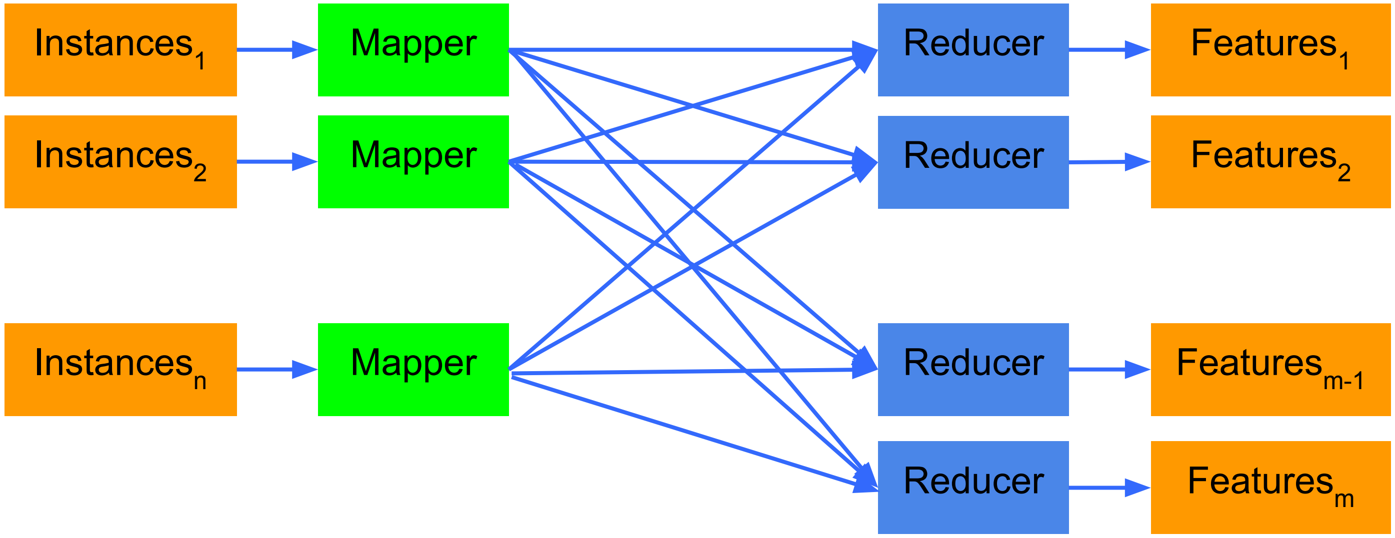
Extensions

- Add regularization
- Support other loss functions

A brief introduction to MapReduce

Programming model for processing large data sets

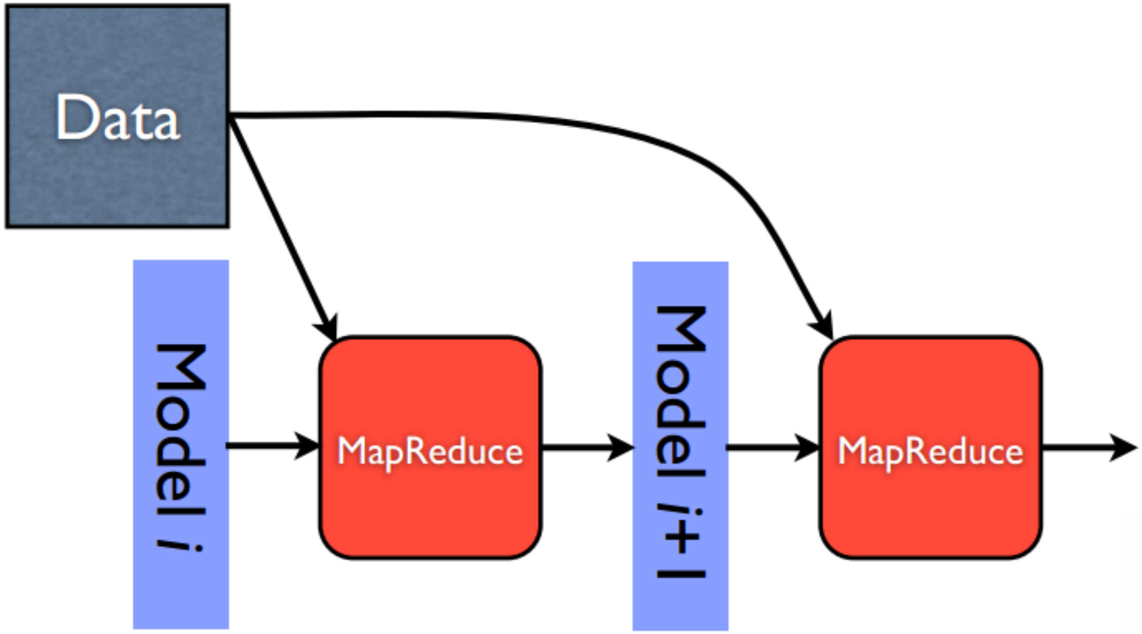
- Proven model and implementation



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

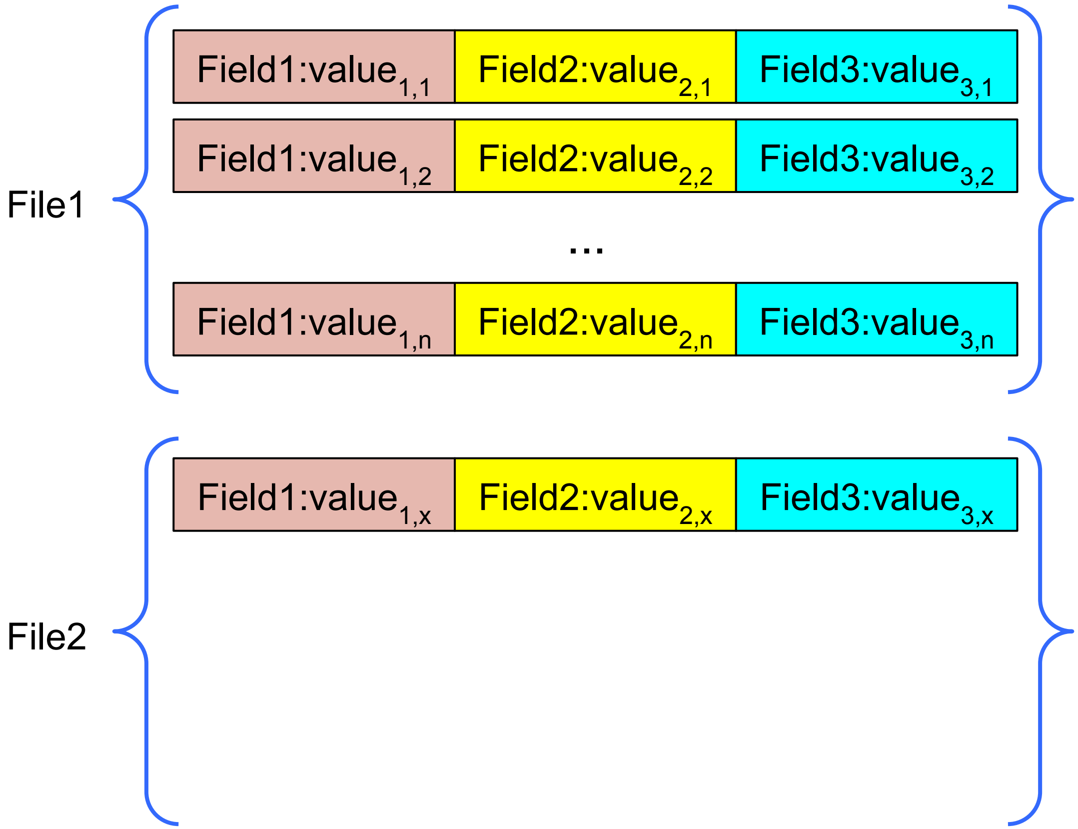


Some observations

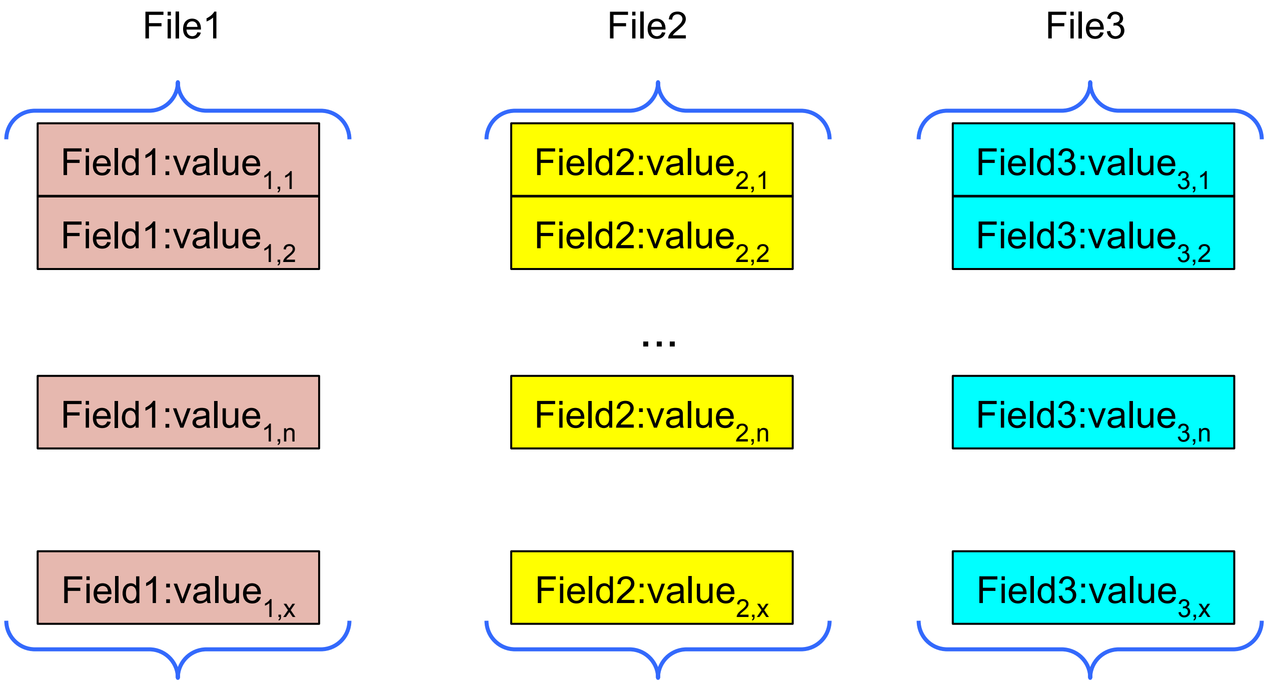
We typically train multiple models

- 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 ...



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 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

Product	Examples	Compressed Raw data	Training data	Compression	Features per example	bytes per feature
A	59.9B	9.98TB	2.00TB	4.99x	54.9	0.67
B	7.6B	2.67TB	0.71TB	3.78x	94.9	1.07
C	197.5B	66.66TB	15.54TB	4.29x	77.7	1.11
D	129.1B	61.93TB	17.24TB	3.59x	100.57	1.46

Processing throughput

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

Concurrency

Number of cores	Time per iteration (secs)	Cost per iteration (core x secs)
4 cores x 10 machines	15000	60000
8 cores x 10 machines	7500	60000
12 cores x 10 machines	4500	54000
16 cores x 10 machines	3900	62400

Impact of L1

Product	Number of features	Number of non-zero features	Fraction of non-zero features
A	868M	20.1M	2.31%
B	333M	7.9M	2.37%
C	1762M	251.8M	14.29%
D	2172M	371.6M	17.11%

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