# Distributed Data Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at Workshop on Scalable Data Analytics, PAKDD, May 2014

# Outline

1. Introduction: why distributed classification
2. Example: a distributed Newton method
3. Discussion from the viewpoint of the application workflow
4. Conclusions

# Outline

1. Introduction: why distributed classification
2. Example: a distributed Newton method
3. Discussion from the viewpoint of the application workflow
4. Conclusions

# Why Distributed Data Classification?

- The usual answer is that data are too big to be stored in one computer
- However, we will show that the whole issue is more complicated

# Let's Start with An Example

- Using a linear classifier LIBLINEAR (Fan et al., 2008) to train the rcv1 document data sets (Lewis et al., 2004).
- # instances: 677,399, # features: 47,236
- On a typical PC

  ```
  $time ./train rcv1_test.binary
  ```
- Total time: 50.88 seconds

  Loading time: 43.51 seconds

- For this example

$$\text{loading time} \gg \text{running time}$$

- In fact, two seconds are enough $\Rightarrow$ test accuracy becomes stable

# Loading Time Versus Running Time

- To see why this happens, let's discuss the complexity
- Assume the memory hierarchy contains only disk and number of instances is $l$
- Loading time: $l \times$ (a big constant)

  Running time: $l^q \times$ (some constant), where $q \geq 1$.
- Running time is often larger than loading because $q > 1$ (e.g., $q = 2$ or $3$)

  Example: kernel methods

# Loading Time Versus Running Time (Cont'd)

- Therefore,

$$l^{q-1} > \text{ a big constant}$$

  and traditionally machine learning and data mining papers consider only running time
- When $l$ is large, we may use a linear algorithm (i.e., $q = 1$) for efficiency

# Loading Time Versus Running Time (Cont'd)

- An important conclusion of this example is that computation time may not be the only concern

  - If running time dominates, then we should design algorithms to reduce number of operations

  - If loading time dominates, then we should design algorithms to reduce number of data accesses

- This example is on one machine. Situation on distributed environments is even more complicated

# Possible Advantages of Distributed Data Classification

Parallel data loading

- Reading several TB data from disk is slow
- Using 100 machines, each has 1/100 data in its local disk ⇒ 1/100 loading time
- But moving data to these 100 machines may be difficult!

Fault tolerance

- Some data replicated across machines: if one fails, others are still available

# Possible Disadvantages of Distributed Data Classification

- More complicated (of course)
- Communication and synchronization

# Going Distributed or Not Isn't Easy to Decide

- Quote from Yann LeCun (KDnuggets News 14:n05)
  "I have seen people insisting on using Hadoop for datasets that could easily fit on a flash drive and could easily be processed on a laptop."
- Now disk and RAM are large. You may load several TB of data once and conveniently conduct all analysis
- The decision is application dependent

# Outline

# Logistic Regression

- Training data $\{y_i, \boldsymbol{x}_i\}, \boldsymbol{x}_i \in R^n, i = 1, \ldots, l,\ y_i = \pm 1$
- $l$: # of data, $n$: # of features
- Regularized logistic regression

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}),$$

where

$$f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\log\left(1 + e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i}\right).$$

- $C$: regularization parameter decided by users
- Twice differentiable, so we can use Newton methods

# Newton Methods

- Newton direction

$$\min_{\boldsymbol{s}} \quad \nabla f(\boldsymbol{w}^k)^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T \nabla^2 f(\boldsymbol{w}^k)\boldsymbol{s}$$

- This is the same as solving Newton linear system

$$\nabla^2 f(\boldsymbol{w}^k)\boldsymbol{s} = -\nabla f(\boldsymbol{w}^k)$$

- Hessian matrix $\nabla^2 f(\boldsymbol{w}^k)$ too large to be stored

$$\nabla^2 f(\boldsymbol{w}^k) : n \times n, \quad n : \text{ number of features}$$

- But Hessian has a special form

$$\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + CX^T DX,$$

# Newton Methods (Cont'd)

- $X$: data matrix. $D$ diagonal with

$$D_{ii} = \frac{e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}}{(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i})^2}$$

- Using Conjugate Gradient (CG) to solve the linear system. Only Hessian-vector products are needed

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{s} = \boldsymbol{s} + C \cdot X^T(D(X\boldsymbol{s}))$$

- Therefore, we have a Hessian-free approach
- Other details; see Lin et al. (2008) and the software LIBLINEAR

# Parallel Hessian-vector Product

- Hessian-vector products are the computational bottleneck

$$X^T D X \boldsymbol{s}$$

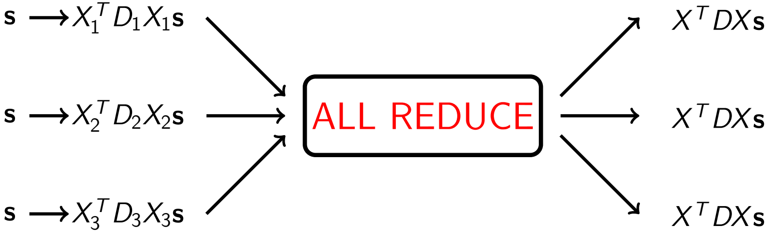
- Data matrix $X$ is now distributedly stored

$$\begin{array}{rl}
\text{node } 1 \longrightarrow & \boxed{X_1} \\
\text{node } 2 \longrightarrow & \boxed{X_2} \\
 & \cdots \\
\text{node } p \longrightarrow & \boxed{X_p}
\end{array}$$

$$X^T D X \boldsymbol{s} = X_1^T D_1 X_1 \boldsymbol{s} + \cdots + X_p^T D_p X_p \boldsymbol{s}$$

# Parallel Hessian-vector Product (Cont'd)

We use allreduce to let every node get $X^T D X \mathbf{s}$

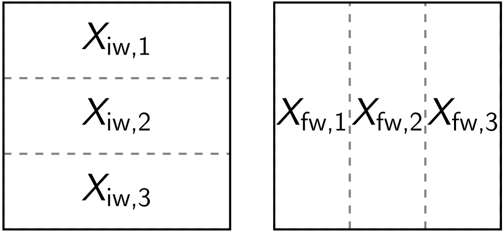



Allreduce: reducing all vectors $(X_i^T D_i X_i)$ to a single vector $(X^T D X)$ and then sending the result to every node

# Parallel Hessian-vector Product (Cont'd)

- Then each node has all the information to finish a Newton method
- We don't use a master-slave model because implementations on master and slaves become different

# Instance-wise and Feature-wise Data Splits




Instance-wise Feature-wise

- Feature-wise: each machine calculates part of the Hessian-vector product

$$(\nabla^2 f(\boldsymbol{w})\mathbf{v})_{\mathrm{fw},1} = \boldsymbol{v}_1 + C X_{\mathrm{fw},1}^T D (X_{\mathrm{fw},1}\boldsymbol{v}_1 + \cdots + X_{\mathrm{fw},p}\boldsymbol{v}_p)$$

# Instance-wise and Feature-wise Data Splits (Cont'd)

- $X_{\text{fw},1}\boldsymbol{v}_1, \ldots, X_{\text{fw},p}\boldsymbol{v}_p$ must be available on all nodes (by allreduce)
- Data moved per Hessian-vector product

  Instance-wise: $O(n)$, Feature-wise: $O(l)$

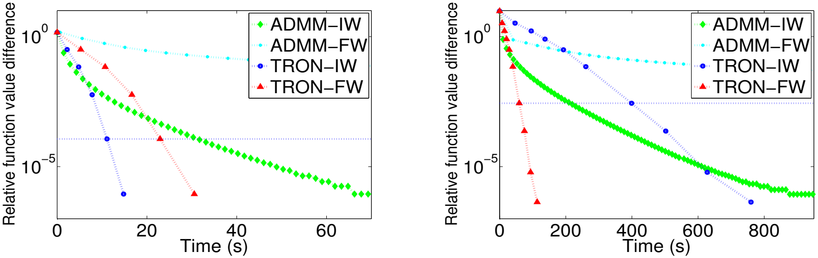
# Experiments

- Two sets:

| Data set | $l$ | $n$ | #nonzeros |
|---|---|---|---|
| epsilon | 400,000 | 2,000 | 800,000,000 |
| webspam | 350,000 | 16,609,143 | 1,304,697,446 |

- For results of more sets, see Zhuang et al. (2014)
- We use Amazon AWS
- We compare
  - TRON: Newton method
  - ADMM: alternating direction method of multipliers (Boyd et al., 2011; Zhang et al., 2012)

# Experiments (Cont'd)




epsilon webspam

- 16 machines are used
- Horizontal line: test accuracy has stabilized
- TRON has faster convergence than ADMM
- Instance-wise and feature-wise splits useful for $l \gg n$ and $l \ll n$, respectively

# Outline

1. Introduction: why distributed classification
2. Example: a distributed Newton method
3. Discussion from the viewpoint of the application workflow
4. Conclusions

# Training Is Only Part of the Workflow

- Previous experiments show that for a set with

  0.35M instances and 16M features,

  distributed training using 16 machines takes 50 seconds
- This looks good, but is not the whole story
- Copying data from Amazon S3 to 16 local disks takes more than 150 seconds
- Distributed training may not be the bottleneck in the whole workflow

# Example: CTR Prediction

- CTR prediction is an important component of an advertisement system

$$\text{CTR} = \frac{\#\ \text{clicks}}{\#\ \text{impressions}}.$$

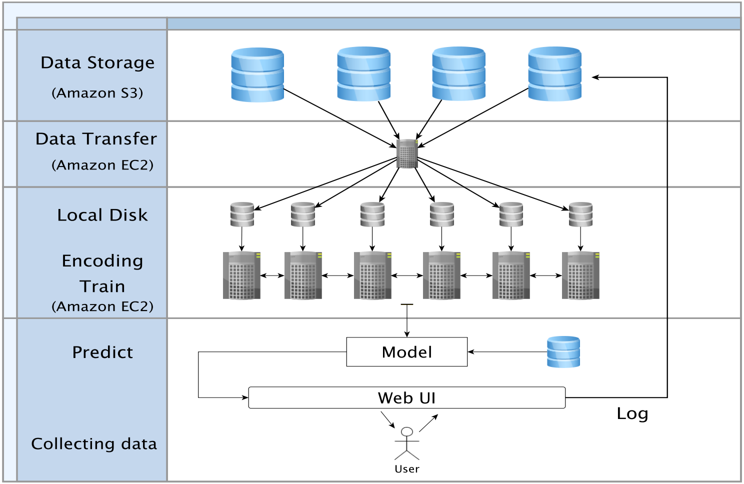
- A sequence of events

| | |
|---|---|
| Not clicked | Features of user |
| Clicked | Features of user |
| Not clicked | Features of user |
| ... | ... |

- A binary classification problem. We use the distributed Newton method described above

# Example: CTR Prediction (Cont'd)

## System Architecture



# Example: CTR Prediction (Cont'd)

- We use data in a sliding window. For example, data of past week is used to train a model for today's prediction
- We keep renting local disks
- A coming instance is immediately dispatched to a local disk
- Thus data moving is completed before training
- For training, we rent machines to mount these disks
- Data are also constantly removed

# Example: CTR Prediction (Cont'd)

- This design effectively alleviates the problem of moving and copying data before training
- However, if you want to use data 3 months ago for analysis, data movement becomes a issue
- This is an example showing that distributed training is just part of the workflow
- It is important to consider all steps in the whole application

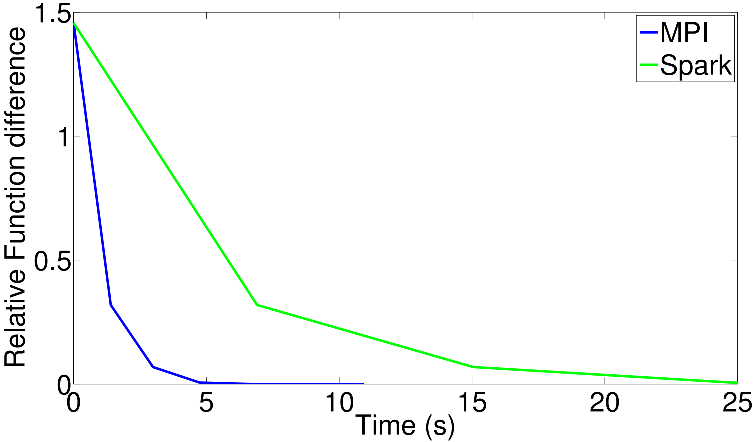
# What if We Don't Maintain Data at All?

- We may use an online setting so an instance is used only once
- Advantages: the classification implementation is simpler than methods like distributed Newton
- Disadvantage: you may worry about accuracy
- The situation may be application dependent

# Programming Frameworks

- We use MPI for the above experiments
- How about others like MapReduce?
- MPI is more efficient, but has no fault tolerance
- In contrast, MapReduce is slow for iterative algorithms due to heavy disk I/O
- Many new frameworks are being actively developed
  1. Spark (Zaharia et al., 2010)
  2. REEF (Chun et al., 2013)
- Selecting suitable frameworks for distributed classification isn't that easy!

# A Comparison Between MPI and Spark




We use the data set epsilon. Spark is slower, but in general competitive

# Distributed LIBLINEAR

- We recently released an extension of LIBLINEAR for distributed classification
- See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear`
- We support both MPI and Spark
- The development is still in an early stage. Your comments are very welcome.

# Outline

# Conclusions

- Distributed training is only one component of the whole workflow
- In a big-data environment, every component can be a bottleneck
- System issues are important because many programming frameworks are still being developed
- Overall, distributed classification is an active and exciting research topic