Distributed Data Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at ICML workshop on New Learning Frameworks and Models for Big Data, June 25, 2014
Outline

1 Introduction: why distributed classification

2 Example: a distributed Newton method for logistic regression

3 Discussion from the viewpoint of the application workflow

4 Conclusions
Outline

1. Introduction: why distributed classification
2. Example: a distributed Newton method for logistic regression
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

loading time $\gg$ 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: $l \times (a \text{ big constant})$
- Running time: $l^q \times (\text{some constant})$, where $q \geq 1$.
- Running time is often larger than loading because $q > 1$ (e.g., $q = 2$ or 3)
- Example: kernel methods
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.
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.
Introduction: why distributed classification

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 \( \Rightarrow \) 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

Everybody says moving computation to data, but this isn’t that easy
Introduction: why distributed classification

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

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

- Training data \( \{y_i, x_i\}, x_i \in \mathbb{R}^n, i = 1, \ldots, l, y_i = \pm 1 \)
- \( l \): # of data, \( n \): # of features
- Regularized logistic regression
  \[
  \min_w f(w),
  \]
  where
  \[
  f(w) = \frac{1}{2} w^T w + C \sum_{i=1}^{l} \log \left( 1 + e^{-y_i w^T x_i} \right).
  \]
- \( C \): regularization parameter decided by users
- Twice differentiable, so we can use Newton methods.
Newton Methods

- Newton direction

\[
\min_s \quad \nabla f(w^k)^T s + \frac{1}{2} s^T \nabla^2 f(w^k) s
\]

- This is the same as solving Newton linear system

\[
\nabla^2 f(w^k)s = -\nabla f(w^k)
\]

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

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

- But Hessian has a special form

\[
\nabla^2 f(w) = I + CX^T DX,
\]
Newton Methods (Cont’d)

- $X$: data matrix. $D$ diagonal with
  \[ D_{ii} = \frac{e^{-y_iw^T x_i}}{(1 + e^{-y_iw^T x_i})^2} \]

- Using Conjugate Gradient (CG) to solve the linear system. Only Hessian-vector products are needed
  \[ \nabla^2 f(w)s = s + C \cdot X^T(D(Xs)) \]

- 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

\[ X^TDXs \]

- Data matrix \( X \) is now distributedly stored

\[
X^TDXs = X_1^TD_1X_1s + \cdots + X_p^TD_pX_p s
\]
Parallel Hessian-vector Product (Cont’d)

We use allreduce to let every node get $X^T DXs$

\[ s \rightarrow X_1^T D_1 X_1 s \]
\[ s \rightarrow X_2^T D_2 X_2 s \]
\[ s \rightarrow X_3^T D_3 X_3 s \]

Allreduce: reducing all vectors $(X_i^T D_i X_i x, \forall i)$ to a single vector $(X^T DXs \in \mathbb{R}^n)$ and then sending the result to every node
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.

We use MPI here, but will discuss other programming frameworks later.
Example: a distributed Newton method for logistic regression

Instance-wise and Feature-wise Data Splits

**Instance-wise**

| \(X_{iw,1}\) | \(X_{iw,2}\) | \(X_{iw,3}\) |

**Feature-wise**

\(X_{fw,1} | X_{fw,2} | X_{fw,3}\)

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

\[
(\nabla^2 f(w)v)_{fw,1} = v_1 + CX_{fw,1}^T D(X_{fw,1}v_1 + \cdots + X_{fw,p}v_p)
\]
Instance-wise and Feature-wise Data Splits (Cont’d)

- \(X_{fw,1}v_1 + \cdots + X_{fw,p}v_p \in \mathbb{R}^l\) must be available on all nodes (by allreduce)

- Amount of data moved per Hessian-vector product:
  - Instance-wise: \(O(n)\), Feature-wise: \(O(l)\)
Experiments

- Two sets:

<table>
<thead>
<tr>
<th>Data set</th>
<th>$l$</th>
<th>$n$</th>
<th>#nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsilon</td>
<td>400,000</td>
<td>2,000</td>
<td>800,000,000</td>
</tr>
<tr>
<td>webspam</td>
<td>350,000</td>
<td>16,609,143</td>
<td>1,304,697,446</td>
</tr>
</tbody>
</table>

- 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)

- 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
Other Distributed Classification Methods

- We give only an example here (distributed Newton)
- There are many other methods
- For example, distributed quasi Newton, distributed random forests, etc.
- Existing software include, for example, Vowpal_Wabbit (Langford et al., 2007)
Discussion from the viewpoint of the application workflow

Outline

1. Introduction: why distributed classification
2. Example: a distributed Newton method for logistic regression
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
  - Clicked
  - Not clicked
  - \ldots

- A binary classification problem. We use the distributed Newton method described above
Example: CTR Prediction (Cont’d)

System Architecture

<table>
<thead>
<tr>
<th>Data Storage</th>
<th>Amazon S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Transfer</td>
<td>Amazon EC2</td>
</tr>
<tr>
<td>Local Disk</td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td></td>
</tr>
<tr>
<td>Train</td>
<td></td>
</tr>
<tr>
<td>(Amazon EC2)</td>
<td></td>
</tr>
</tbody>
</table>

Collecting data

Model

Web UI

User

Log
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 an 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.
- See also an essay by Jimmy Lin (2012).
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

<table>
<thead>
<tr>
<th>Data set</th>
<th>$l$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsilon</td>
<td>400,000</td>
<td>2,000 dense features</td>
</tr>
<tr>
<td>rcv1</td>
<td>677,399</td>
<td>47,236 sparse features</td>
</tr>
</tbody>
</table>

Discussion from the viewpoint of the application workflow
Discussion from the viewpoint of the application workflow

A Comparison Between MPI and Spark (Cont’d)

8 nodes in a local cluster (not AWS) are used. Spark is slower, but in general competitive.

Some issues may cause the time differences:
- C versus Scala
- Allreduce versus master-slave setting
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. We are working hard to improve the Spark version

Your comments are very welcome.
Outline

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

- Designing distributed training algorithm isn’t easy. You can parallelize existing algorithms or create new ones.
- Issues such as communication cost must be solved.
- We also need to know that distributed training is only one component of the whole workflow.
- System issues are important because many programming frameworks are still being developed.
- Overall, distributed classification is an active and exciting research topic.