Machine Learning over Big Data

Presented by Fuhao Zou
fuhao@hust.edu.cn

Jue 16, 2014

Huazhong University of Science and Technology
Contents

1. Role of Machine learning
2. Challenge of Big Data Analysis
3. Distributed Machine learning
4. Conclusion
Big Data already Happened

1 Billion Tweets Per Week

750 Million Facebook Users

6 Billion Flickr Photos

48 Hours of Video Uploaded every Minute
4V in Big Data

Characteristics

1. Volume
2. Variety
3. Velocity
4. Value
Data≠Knowledge

"If a tree falls in a forest and no one is around to hear it, does it make a sound?" --- George Berkeley

Nobody Knows What Is In All These Data Unless Having Them Processed and Analyzed
Why do Machine Learning on Big Data?

Machine learning incorporates data analyses covering predictive analytics, data mining, pattern recognition, and multivariate statistics.

<table>
<thead>
<tr>
<th>Basic Analytics</th>
<th>Advanced Analytics (Machine Learning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts and Averages</td>
<td>Discovering Patterns</td>
</tr>
<tr>
<td>Data Preprocessing</td>
<td>Making Predictions</td>
</tr>
<tr>
<td>SQL Queries</td>
<td>Multivariate Queries</td>
</tr>
<tr>
<td>Human Scale</td>
<td>Machine Scale</td>
</tr>
<tr>
<td>Hand Crafted</td>
<td>Data Driven</td>
</tr>
</tbody>
</table>

Unfortunately traditional analytics tools are not well suited to capturing the **value hidden** in Big Data.
Contents

1. Role of Machine learning
2. Challenge of Big Data Analysis
3. Distributed Machine learning
4. Conclusion
Challenge #1 – Massive Data Scale

~1B node, do not fitting into the main memory of a single machine, a familiar problem!
Challenge #2 – Gigantic Model Size

$>10^{11}$ parameters, do not fitting into the main memory of a single machine either!
Challenge #3 – Huge Cognitive Space

1M ~ 1B categories are seen in modern extreme classification problems, kNN? SVM?
Distributed ML is a necessity
Contents

1. Role of Machine learning
2. Challenge of Big Data Analysis
3. Distributed Machine learning
4. Conclusion
Data-Parallel ML

Linear regression with gradient descent

\[ h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \]

\[ J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \]

Repeat { 
\[ \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \]

(for every \( j = 0, \ldots, n \))
}
Batch gradient descent:

\[ \theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \]

**Machine 1:** Use \((x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})\).

\[ temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \]

**Machine 2:** Use \((x^{(101)}, y^{(101)}), \ldots, (x^{(200)}, y^{(200)})\).

\[ temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \]

**Machine 3:** Use \((x^{(201)}, y^{(201)}), \ldots, (x^{(300)}, y^{(300)})\).

\[ temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \]

**Machine 4:** Use \((x^{(301)}, y^{(301)}), \ldots, (x^{(400)}, y^{(400)})\).

\[ temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \]

**Combine:**

\[ \theta_j := \theta_j - \alpha \frac{1}{400} \left( temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)} \right) \]
Map-reduce

Training set

Computer 1

Computer 2

Computer 3

Computer 4

Combine results

Hadoop
Graph-Parallel ML
Label Propagation

- **Social Arithmetic:**
  50% What I list on my profile
  40% Sue Ann Likes
  + 10% Carlos Like

- **Recurrence Algorithm:**
  \[ \text{Likes}[i] = \sum_{j \in \text{Friends}[i]} W_{ij} \times \text{Likes}[j] \]
  - iterate until convergence

- **Parallelism:**
  - Compute all \text{Likes}[i] in parallel

---

- Sue Ann
  80% Cameras
  20% Biking

- Profile
  50% Cameras
  50% Biking

- Carlos
  30% Cameras
  70% Biking

Me

50%

80%

40%

10%

50%

10%
PageRank (Centrality Measures)

• **Iterate:**

\[ R[i] = \alpha + (1 - \alpha) \sum_{(j,i) \in E} \frac{1}{L[j]} R[j] \]

• **Where:**
  
  – \( \alpha \) is the random reset probability
  
  – \( L[j] \) is the number of links on page \( j \)
Matrix Factorization
Alternating Least Squares (ALS)

Update Function computes:

\[ u_i = \arg \min_w \sum_{j \in N[i]} (r_{ij} - m_j \cdot w)^2 \]

\[ m_j = \arg \min_w \sum_{i \in N[j]} (r_{ij} - u_i \cdot w)^2 \]
Other Examples

• **Statistical Inference in Relational Models**
  – Belief Propagation
  – Gibbs Sampling

• **Network Analysis**
  – Centrality Measures
  – Triangle Counting

• **Natural Language Processing**
  – CoEM
  – Topic Modeling
Pregel: Bulk Synchronous Parallel

Compute

Communicate

Barrier
Open Source Implementations

- **PREGEL**: [https://plus.google.com/+Pregel/](https://plus.google.com/+Pregel/)
Curse of the Slow Job

Iterations

CPU 1 → Data → CPU 1
CPU 2 → Data → CPU 2
CPU 3 → Data → CPU 3

Barrier

Iterations

CPU 1 → Data → CPU 1
CPU 2 → Data → CPU 2
CPU 3 → Data → CPU 3

Barrier

Iterations

CPU 1 → Data → CPU 1
CPU 2 → Data → CPU 2
CPU 3 → Data → CPU 3

Barrier

CPU 1 → Data → CPU 1
CPU 2 → Data → CPU 2
CPU 3 → Data → CPU 3

Barrier
Tradeoffs of the BSP Model

• **Pros:**
  – *Graph Parallel*
  – Relatively easy to build
  – Deterministic execution

• **Cons:**
  – Doesn’t exploit the graph structure
  – Can lead to inefficient systems
Asynchronous Computation Model

Threads proceed to next iteration without waiting.

An asynchronous variant:

GraphLab: [http://graphlab.org/](http://graphlab.org/)
What is GraphLab?
The GraphLab Framework

Graph Based Data Representation

Update Functions User Computation

Scheduler

Consistency Model
Data Graph

A graph with arbitrary data (C++ Objects) associated with each vertex and edge.

Graph:
- Social Network

Vertex Data:
- User profile text
- Current interests estimates

Edge Data:
- Similarity weights
New Perspective on Partitioning

- Natural graphs have poor edge separators
  - Classic graph partitioning tools (e.g., ParMetis, Zoltan …) fail

- Natural graphs have good vertex separators

CPU 1

Must synchronize many edges

CPU 2

CPU 1

Must synchronize a single vertex

CPU 2
An update function is a user defined program which when applied to a **vertex** transforms the data in the **scope** of the vertex.

```latex
code
\text{pagerank}(i, \text{scope})\
\begin{align*}
  &\text{// Get Neighborhood data} \\
  & (R[i], W_{ij}, R[j]) \leftarrow \text{scope}; \\
  &\text{// Update the vertex data} \\
  & R[i] \leftarrow \alpha + (1 - \alpha) \sum_{j \in N(i)} W_{ij} \times R[j]; \\
  &\text{// Reschedule Neighbors if needed} \\
  & \text{if } R[i] \text{ changes then} \\
  & \text{reschedule_neighbors_of}(i); \\
\end{align*}
```
The Scheduler

The scheduler determines the order that vertices are updated.

The process repeats until the scheduler is empty.
Consistency Through Scheduling

• **Edge Consistency Model:**
  – Two vertices can be *updated simultaneously* if they do not share an edge.

• **Graph Coloring:**
  – Two vertices can be assigned the same color if they do not share an edge.
Dynamically Changing Graphs

- **Example: Social Networks**
  - New users → New Vertices
  - New Friends → New Edges

- **How do you adaptively maintain computation:**
  - Trigger computation with changes in the graph
  - Update “interest estimates” only where needed
  - Exploit asynchrony
  - Preserve consistency
Graph Partitioning

- How can you quickly place a large data-graph in a distributed environment:
- Edge separators fail on large power-law graphs
  - Social networks, Recommender Systems, NLP
- Constructing vertex separators at scale:
  - No large-scale tools!
  - How can you adapt the placement in changing graphs?
New Computation model is needed

**BSP**: safe by slow

**Async**: fast but risky
Thank You!