PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Luyu Wang
SciCom Group, UW
Content

- Background
- Methodology
- Results
- Conclusion
Google’s Bounce Rate Prediction Problem

- 'Bounce'
Google’s Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience
Google’s Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience
- Task: to predict bounce rate with data on hand
Google’s Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience
- Task: to predict bounce rate with data on hand
The Data Mining Tasks

- Discovering patterns in large data sets (knowledge)
- Data mining vs machine learning?
  - Lines are blurred
The Data Mining Tasks

- Supervised learning
  - Classification
  - Regression
- Unsupervised learning
  - Clustering
  - Compression
  - Outlier detection
- Reinforce learning
  ......
The Data Mining Tasks

• Supervised learning
  - Classification
  - Regression
• Unsupervised learning
  - Clustering
  - Compression
  - Outlier detection
• Reinforce learning
  ......
Google’s Bounce Rate Prediction Problem

- Go back to ‘bounce’
Google’s Bounce Rate Prediction Problem

• Go back to ‘bounce’

• One Click:
  - 6 attributes
  - 1 label
Google’s Bounce Rate Prediction Problem

- 6 attributes
  - search query of the click
  - advertiser chosen keyword
  - ad text
  - estimated clickthrough rate of the ad click
  - numeric similarity score
  - whether the ad matches the query

- 1 label
  - bounce or not
Supervised Learning - Data Model

- Set of attributes
  \[ \chi = \{ X_1, X_2, ..., X_N \} \]
- Output
  \[ Y \]
- Training data set (the i\textsuperscript{th} vector)
  \[ D^* = \{ (x_i, y_i) \mid x_i \in D_{x_1} \times D_{x_2} \times ... \times D_{x_N} \} \]
Supervised Learning - Task

• Given the training dataset

\[ D^* = \{(x_i, y_i) \mid x_i \in D_{x_1} \times D_{x_2} \times \ldots \times D_{x_N} \} \]

• Goal: to learn a mapping model

\[ F : D_{x_1} \times D_{x_2} \times \ldots \times D_{x_N} \rightarrow D_y \]
Google’s Bounce Rate Prediction Problem

• Given 6 attributes
  - search query of the click
  - advertiser chosen keyword
  - ad text
  - estimated clickthrough rate of the ad click
  - numeric similarity score
  - whether the ad matches the query

• Want to know if it is going to be a bounce?
Supervised Learning - Task

- Given the training dataset
  \[ D^* = \{(x_i, y_i) \mid x_i \in D_{x_1} \times D_{x_2} \times \cdots \times D_{x_N} \} \]

- Goal: to learn a mapping model
  \[ F : D_{x_1} \times D_{x_2} \times \cdots \times D_{x_N} \rightarrow D_y \]

- Tree models
  - Capable of modeling complex tasks
Tree Model

- Goal: to learn a mapping model
  \[ F : D_{x_1} \times D_{x_2} \times \ldots \times D_{x_N} \to D_y \]

- Recursively partitioning the input data space into non-overlapping regions

- Simple model each region
  - constant
  - simple function
Tree Model

- Easy to interpret; thus popular

Learning Tree Model

- Greedy learning algorithm

Input: node $n$, training dataset $D$

1) fully scan $D$, find the *best split*, by maximizing ‘purity’

$$|D| \times Var(D) - (|D_L| \times Var(D_L) + |D_R| \times Var(D_R))$$

2) for either branch
   - if stopping criteria satisfied: *pure* region
   - else: advance a level
Google’s Bounce Rate Prediction Problem

- 'Bounce'
- High bounce rate = poor user experience
- Task: predicting bounce rate with data on hand

What if big data?
Learning Tree Model

• Greedy learning algorithm

Input: node $n$, training dataset $D$

1) **fully scan** $D$, find the *best split*

2) for either branch
   - if stopping criteria satisfied: *pure* region
   - else: build a higher-level node

- out of memory
- hard disk slow
Solution - Scaling Up Tree Learning

• Fully scan $D$, find the best split
  - out of memory
  - hard disk slow

• By Google Research, 2009
  - Computer Cluster
  - MapReduce
  - Tree learning

Content

- Background
- **Methodology**
- Results
- Conclusion
Computer Cluster

- Controller and workers

Source: Wikepedia
MapReduce Framework

- Objective: to easily handle data too large to fit in memory
MapReduce Framework

- Objective: to easily handle data too large to fit in memory

- It does all the dirty work:
  - distribute the data
  - parallelize the computation
  - handle failures
MapReduce Framework

• Objective: to easily handle data too large to fit in memory

• It does all the dirty work:
  - distribute the data
  - parallelize the computation
  - handle failures

• User simply writes Map and Reduce functions
Computer Cluster

- Core: Controller

Job of Controller

- Keeps model file (M), containing the entire tree constructed so far
- Partitions the whole training dataset, across a set of mappers
Job of Controller

• Each tree node, detects size of data set
  
  if single machine ok?
    -> push to ‘SmallData Queue’
  else
    -> push to ‘LargeData Queue’

• Schedules jobs in both queues for workers
Job of Workers

- Map and Reduce functions

MapReduce Work - SmallData Queue

- Map function
  - input:
    partitioned training set $D_k$
    node $n$
    Model file M
  - check if an instance input to $n \rightarrow$ emits
  - output (list):
    key = node $n$
    value = subset of $D_k$ input to $n$
MapReduce Work - SmallData Queue

• Reduce function
  - input:
    key = node n
    value = subset of $D_k$ input to $n$
  - loads training records in memory
  - single-machine algorithm to find the split

In this way, cluster can process many nodes in parallel to grow the tree
MapReduce Work – LargeData Queue

• Ordered attribute vs. Unordered

Compare adjacent pairs

Troublesome; Breiman Method
MapReduce Work – LargeData Queue

• Map function

**Algorithm 2 MR.ExpandNodes::Map**

Require: NodeSet $N$, ModelFile $M$, Training record $(x, y) \in D^*$

```plaintext
1: n = TraverseTree(M, x)
2: if $n \in N$ then
3:   agg_tup_n $\leftarrow y$
4: for all $X \in \mathcal{X}$ do
5:   $v$ = Value on $X$ in $x$
6: if $X$ is ordered then
7:   for all Split point $s$ of $X$ s.t. $s < v$ do
8:     $T_n,X[s] \leftarrow y$
9: else
10:   $T_n,X[v] \leftarrow y$
```

**Algorithm 3 MR.ExpandNodes::Map_Finalize**

Require: NodeSet $N$

```plaintext
1: for all $n \in N$ do
2: Output to all reducers($n, \text{agg}_\text{tup}_n$)
3: for all $X \in \mathcal{X}$ do
4: if $X$ is ordered then
5: for all Split point $s$ of $X$ do
6: Output($n, X, s, T_n,X[s]$)
7: else
8: for all $v \in T_n,X$ do
9: Output($n, X, (v, T_n,X[v])$)
```
MapReduce Work – LargeData Queue

• Reduce function

```
Algorithm 4 MR_ExpandNodes::Reduce

Require: Key $k$, Value Set $V$
1: if $k == n$ then
2:   {Aggregate agg_tup_n’s from mappers}
3:   agg_tup_n = Aggregate($V$)
4: else if $k == n, X, s$ then
5:   {Split on ordered attribute}
6:   agg_tup_left = Aggregate($V$)
7:   agg_tup_right = agg_tup_n - agg_tup_left
8:   UpdateBestSplit($S[n], X, s, agg_tup_left, agg_tup_right)$
9: else if $k == n, X$ then
10:   {Split on unordered attribute}
11:   for all $v, agg_tup \in V$ do
12:     $T[v] \leftarrow agg_tup$
13:   UpdateBestSplit($S[n], BreimanSplit(X, T, agg_tup_n)$)
```
Walkthrough

• Training set $D^*$
  - 100 instances

• Memory constraint
  - 25 instances

• Stopping criteria
  - instances $\leq 10$
Walkthrough

Ordered attribute

A \rightarrow \text{LargeData Queue}
\rightarrow \text{Ordered node splitting}
Walkthrough

Stop

Average of Labels
Walkthrough

Unordered attribute

B -> LargeData Queue
- > Unordered node splitting
Walkthrough

LargeData Queue <- C

D -> LargeData Queue
Walkthrough

E, F, G -> SmallData Queue

H -> LargeData Queue
Walkthrough
Content

- Background
- Methodology
- Results
- Conclusion
Setup

• Bounce rate prediction problem

• 314 million records
  - 10 features
  - 1 label

• Each machine 768MB memory
Time to Train vs. Data Size

- Works well
  - 25 nodes
Time to Train vs. Data Size

- Works well
  - 25 nodes
  - 50
Time to Train vs. Data Size

- Works well
  - 25 nodes
  - 50
  - 100
  - 200
Time to Train vs. Data Size

- Works well
  - 25 nodes
  - 50
  - 100
  - 200
  - 400?
Time to Train vs. Data Size

- 400 workers worse than 200?

- Cluster Management
  - network overhead
  - failure watching
  - schedule backups
  - data distribution & collection
Time to Train vs. Tree Depth

- With/without ‘SmallData Queue’
Time to Train vs. Tree Depth

- With/without ‘SmallData Queue’
  - overhead of cluster management
Time to Train vs. Tree Depth

- With/without ‘SmallData Queue’
  - overhead of cluster management
  - sampling based method on single machine
Error Reduction vs Num of Trees

- Boosted tree model
  - a bundle of weighted weak learners: better performance
Error Reduction vs Num of Trees

- Boosted tree model
  - a bundle of weighted weak learners: better performance
  - better weak learners faster error reduction
Content

• Background
• Methodology
• Results
• Conclusion
Conclusion

• Successfully scales up tree learning with MapReduce
• Performs well
• Pioneered large-scale machine learning

## Related Work - Survey

<table>
<thead>
<tr>
<th>Learning Setting</th>
<th>Algorithm</th>
<th>Cluster Nodes</th>
<th>Parallelization Framework</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>Decision Tree</td>
<td>200</td>
<td>MapReduce</td>
<td>100</td>
</tr>
<tr>
<td>Classification</td>
<td>SVM</td>
<td>500</td>
<td>MPI</td>
<td>100</td>
</tr>
<tr>
<td>Ranking</td>
<td>LambdaMART</td>
<td>32</td>
<td>MPI</td>
<td>10</td>
</tr>
<tr>
<td>Inference</td>
<td>Loopy belief propagation</td>
<td>40</td>
<td>MPI</td>
<td>23</td>
</tr>
<tr>
<td>Inference</td>
<td>MCMC</td>
<td>1024</td>
<td>MPI</td>
<td>1000</td>
</tr>
<tr>
<td>Clustering</td>
<td>Spectral clustering</td>
<td>256</td>
<td>MapReduce, MPI</td>
<td>256</td>
</tr>
<tr>
<td>Clustering</td>
<td>Information-theoretic</td>
<td>400</td>
<td>MPI</td>
<td>100</td>
</tr>
</tbody>
</table>
Go on. I dare you.
References


