PLANET
Massively Parallel Learning of Tree Ensembles with MapReduce

Joshua Herbach*
Google Inc., AdWords

*Joint work with Biswanath Panda, Sugato Basu, Roberto J. Bayardo
Outline

• PLANET – infrastructure for building trees
• Decision trees
• Usage and motivation
• MapReduce
• PLANET details
• Results
• Future Work
Tree Models

- Classic data mining model
- Interpretable
- Good when built with ensemble techniques like bagging and boosting
Construction

\[
\begin{align*}
A & \quad |D|=100 \\
C & \quad |D|=90 \\
D & \quad |D|=45 \\
F & \quad |D|=20 \\
G & \quad |D|=25 \\
H & \quad |D|=15 \\
I & \quad |D|=30 \\

X_1 & < v_1 \\
X_2 & \in \{v_2, v_3\}
\end{align*}
\]
Find Best Split
Trees at Google

• Large Datasets
  ▪ Iterating through a large dataset (10s, 100s, or 1000s of GB) is slow
  ▪ Computing values based on the records in a large dataset is really slow

• Parallelism!
  ▪ Break up dataset across many processing units and then combine results
  ▪ Super computers with specialized parallel hardware to support high throughput are expensive
  ▪ Computers made from commodity hardware are cheap

• Enter MapReduce
Can use a secondary key to control ordering reducers see key-value pairs

*http://labs.google.com/papers/mapreduce.html
PLANET

• Parallel Learner for Assembling Numerous Ensemble Trees

• PLANET is a learner for training decision trees that is built on MapReduce
  ▪ Regression models (or classification using logistic regression)
  ▪ Supports boosting, bagging and combinations thereof
  ▪ Scales to very large datasets
System Components

- Master
  - Monitors and controls everything

- MapReduce Initialization Task
  - Identifies all the attribute values which need to be considered for splits

- MapReduce FindBestSplit Task
  - MapReduce job to find best split when there is too much data to fit in memory

- MapReduce InMemoryGrow Task
  - Task to grow an entire subtree once the data for it fits in memory

- Model File
  - A file describing the state of the model
Master

• Controls the entire process
• Determines the state of the tree and grows it
  ▪ Decides if nodes should be leaves
  ▪ If there is relatively little data entering a node; launch an InMemory MapReduce job to grow the entire subtree
  ▪ For larger nodes, launches a MapReduce job to find candidate best splits
  ▪ Collects results from MapReduce jobs and chooses the best split for a node
  ▪ Updates Model
• Periodically checkpoints system
• Maintains status page for monitoring
Status page

Status overview:
- Initialize Boosting: Done!
- Mapreduce: In Progress
- Clean Model: Pending
- Napping at desk: Skipped

Error vs number of trees

Active Workers:

To write: 071113beef7bcca4/building/grow_output_86

Tasks:
- tree_id: 21
- node(s): 3, 4
- model_file: 071113beef7bcca4/model
- nodes_in_model_file: 473
- active_trees: 21
Initialization MapReduce

- Identifies all the attribute values which need to be considered for splits
- Continuous attributes
  - Compute an approximate equi-depth histogram*
  - Boundary points of histogram used for potential splits
- Categorical attributes
  - Identify attribute's domain
- Generates an “attribute file” to be loaded in memory by other tasks

*G. S. Manku, S. Rajagopalan, and B. G. Lindsay, SIGMOD, 1999.
FindBestSplit MapReduce

- MapReduce job to find best split when there is too much data to fit in memory
- Mapper
  - Initialize by loading attribute file from Initialization task and current model file
  - For each record run the Map algorithm
  - For each node output to all reducers
    <Node.Id, <Sum Result, Sum Squared Result, Count>>
  - For each split output <Split.Id, <Sum Result, Sum Squared Result, Count>>

Map(data):
  Node = TraverseTree(data, Model)
  if Node to be grown:
    Node.stats.addData(data)
  for feature in data:
    Split = FindSplitForValue(Node.Id, feature)
    Split.stats.addData(data)
FindBestSplit MapReduce

- MapReduce job to find best split when there is too much data to fit in memory
- Reducer (Continuous Attributes)
  - Load in all the `<Node_Id, List<Sum Result, Sum Squared Result, Count>>` pairs and aggregate the per_node statistics.
  - For each `<Split_Id, List<Sum Result, Sum Squared Result, Count>>` run the Reduce algorithm
  - For each Node_Id, output the best split found

```python
Reduce(Split_Id, values):
    Split = NewSplit(Split_Id)
    best = FindBestSplitSoFar(Split.Node.Id)
    for stats in values
        split.stats.AddStats(stats)
        left = ComputeImpurity(split.stats)
        right = ComputeImpurity(split.node.stats – split.stats)
        split.impurity = left + right
        if split.impurity < best.impurity:
            UpdateBestSplit(Split.Node.Id, split)
```
FindBestSplit MapReduce

- MapReduce job to find best split when there is too much data to fit in memory
  - Reducer (Categorical Attributes)
    - Modification to reduce algorithm:
      - Compute the aggregate stats for each individual value
      - Sort values by average target value
      - Iterate through list and find optimal subsequence in list*

---

InMemoryGrow MapReduce

- Task to grow an entire subtree once the data for it fits in memory
- Mapper
  - Initialize by loading current model file
  - For each record identify the node it falls under and if that node is to be grown, output `<Node_Id, Record>`
- Reducer
  - Initialize by loading attribute file from Initialization task
  - For each `<Node_Id, List<Record>>` run the basic tree growing algorithm on the records
  - Output the best splits for each node in the subtree
Ensembles

- **Bagging**
  - Construct multiple trees in parallel, each on a sample of the data
  - Sampling without replacement is easy to implement on the Mapper side for both types of MapReduce tasks
    - Compute a hash of `<Tree_Id, Record_Id>` and if it's below a threshold then sample it
  - Get results by combining the output of the trees

- **Boosting**
  - Construct multiple trees in a series, each on a sample of the data*
  - Modify the target of each record to be the residual of the target and the model's prediction for the record
    - For regression, the residual $z$ is the target $y$ minus the model prediction $F(x)$
    - For classification, $z = y - 1 / (1 + \exp(-F(x)))$
  - Get results by combining output from each tree

---

Performance Issues

- Set up and Tear down
  - Per-MapReduce overhead is significant for large forests or deep trees
  - Reduce tear-down cost by polling for output instead of waiting for a task to return
  - Reduce start-up cost through forward scheduling
    - Maintain a set of live MapReduce jobs and assign them tasks instead of starting new jobs from scratch

- Categorical Attributes
  - Basic implementation stored and tracked these as strings
    - This made traversing the tree expensive
  - Improved latency by instead considering fingerprints of these values

- Very high dimensional data
  - If the number of splits is too large the Mapper might run out of memory
  - Instead of defining split tasks as a set of nodes to grow, define them as a set of nodes grow and a set of attributes to explore.
Results
Conclusions

• Large-scale learning is increasingly important
• Computing infrastructures like MapReduce can be leveraged for large-scale learning
• PLANET scales efficiently with larger datasets and complex models.

Future work

- Adding support for sampling with replacement
- Categorical attributes with large domains
  - Might run out of memory
    - Only support splitting on single values
    - Area for future exploration
Thank You!

Q&A