Scaling up Decision Trees

Shannon Quinn

(with thanks to William Cohen of CMU, and B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo of IIT)
A decision tree

Dependent variable: PLAY

- **OUTLOOK?**
  - **sunny**
    - Play 2
    - Don't Play 3
  - **overcast**
    - Play 4
    - Don't Play 0
  - **rain**
    - Play 3
    - Don't Play 2

- **HUMIDITY?**
  - \(\leq 70\)
    - Play 2
    - Don't Play 0
  - \(> 70\)
    - Play 0
    - Don't Play 3

- **WINDY?**
  - **TRUE**
    - Play 0
    - Don't Play 2
  - **FALSE**
    - Play 3
    - Don't Play 0
A regression tree

Dependent variable: PLAY

Play ~= 33

OUTLOOK?

sunny

Play ~= 24

HUMIDITY?

<= 70

Play ~= 37

Play = 30m, 45min

> 70

Play ~= 5

Play = 0m, 0m, 15m

overcast

Play ~= 48

WINDY?

TRUE

Play ~= 0

Play = 0m, 0m

rain

Play ~= 18

FALSE

Play ~= 32

Play = 20m, 30m, 45m,
Most decision tree learning algorithms

1. Given dataset D:
   - return \( \text{leaf}(y) \) if all examples are in the same class \( y \) ... or if blahblah...
   - pick the best split, on the best attribute \( a \)
     - \( a < \theta \) or \( a \geq \theta \)
     - \( a \) or \( \neg(a) \)
     - \( a = c_1 \) or \( a = c_2 \) or ...
     - \( a \ in \{c_1, ..., c_k\} \) or not
   - split the data into \( D_1, D_2, ... D_k \) and recursively build trees for each subset

2. “Prune” the tree
Most decision tree learning algorithms

1. Given dataset D:
   - return \( \text{leaf}(y) \) if all examples are in the same class \( y \) ... or if blahblah...
   - pick the best split, on the best attribute \( a \)
     - \( a < \theta \) or \( a \geq \theta \)
     - \( a \) or \( \text{not}(a) \)
     - \( a = c_1 \) or \( a = c_2 \) or ...
     - \( a \) in \( \{c_1, ..., c_k\} \) or not
   - split the data into \( D_1, D_2, ... D_k \) and recursively build trees for each subset

2. “Prune” the tree

\[
H(D) = \sum_k \text{count}_D(Y = y_k) \log \Pr_D(Y = y_k)
\]
Most decision tree learning algorithms

• Find Best Split
  – Choose split point that minimizes weighted impurity (e.g., variance (regression) and information gain (classification) are common)

• Stopping Criteria (common ones)
  – Maximum Depth
  – Minimum number of data points
  – Pure data points (e.g., all have the same Y label)
Most decision tree learning algorithms

- “Pruning” a tree
  - avoid overfitting by removing subtrees somehow
  - trade variance for bias
Most decision tree learning algorithms

1. Given dataset $D$:
   - return $\text{leaf}(y)$ if all examples are in the same class $y$ ... or if blahblah...
   - pick the best split, on the best attribute $a$
     - $a < \theta$ or $a \geq \theta$
     - $a$ or $\text{not}(a)$
     - $a = c_1$ or $a = c_2$ or ...
     - $a \in \{c_1, \ldots, c_k\}$ or not
   - split the data into $D_1, D_2, \ldots, D_k$ and recursively build trees for each subset

2. “Prune” the tree

Same idea
Decision trees: plus and minus

• Simple and fast to learn
• Arguably easy to understand (if compact)
• Very fast to use:
  – often you don’t even need to compute all attribute values
• Can find interactions between variables (play if it’s cool and sunny or ....) and hence non-linear decision boundaries
• Don’t need to worry about how numeric values are scaled
Decision trees: plus and minus

- Hard to prove things about
- Not well-suited to probabilistic extensions
- Don’t (typically) improve over linear classifiers when you have lots of features
- Sometimes fail badly on problems that linear classifiers perform well on
Another view of a decision tree
Another view of a decision tree
Another view of a decision tree
Another picture...
Fixing decision trees....

• Hard to prove things about
• Don’t (typically) improve over linear classifiers when you have lots of features
• Sometimes fail badly on problems that linear classifiers perform well on

• Solution is two build ensembles of decision trees
Most decision tree learning algorithms

- “Pruning” a tree
  - avoid overfitting by removing subtrees somehow
  - trade variance for bias

Alternative - build a big ensemble to reduce the variance of the algorithms - via boosting, bagging, or random forests
Example: random forests

• Repeat T times:
  – Draw a bootstrap sample S (\(n\) examples taken with replacement) from the dataset D.
  – Select a subset of features to use for S
    • Usually half to 1/3 of the full feature set
  – Build a tree considering only the features in the selected subset
    • Don’t prune
• Vote the classifications of all the trees at the end

• Ok - how well does this work?
Tree Induction vs. Logistic Regression: A Learning-Curve Analysis

Claudia Perlich
Foster Provost
Jeffrey S. Simonoff

Leonard N. Stern School of Business
New York University
44 West 4th Street
New York, NY 10012

CPERLICH@STERN.NYU.EDU
FPROVOST@STERN.NYU.EDU
JSIMONOF@STERN.NYU.EDU
Scaling up decision tree algorithms

1. Given dataset D:
   - return $\text{leaf}(y)$ if all examples are in the same class $y$
   - pick the best split, on the best attribute $a$
     - $a < \theta$ or $a \geq \theta$
     - $a$ or $\neg(a)$
     - $a = c_1$ or $a = c_2$ or ...
     - $a \in \{c_1, \ldots, c_k\}$ or not
   - split the data into $D_1, D_2, \ldots, D_k$ and recursively build trees for each subset

2. “Prune” the tree
Scaling up decision tree algorithms

1. Given dataset D:
   - return \( \text{leaf}(y) \) if all examples are in the same class \( y \)
   - pick the best split, on the best attribute \( a \)
     - \( a < \theta \) or \( a \geq \theta \)
     - \( a \) or not(\( a \))
     - \( a = c_1 \) or \( a = c_2 \) or ...
     - \( a \) in \( \{c_1, ..., c_k\} \) or not
   - split the data into \( D_1, D_2, \ldots, D_k \) and recursively build trees for each subset

2. “Prune” the tree
Scaling up decision tree algorithms

1. Given dataset D:
   - return $leaf(y)$ if all examples are in the same class $y$
   - pick the best split, on the best attribute $a$
     - $a < \theta$ or $a \geq \theta$
     - $a$ or $not(a)$
     - $a = c_1$ or $a = c_2$ or ...
     - $a \ in \ {c_1, ..., c_k}$ or not
   - split the data into $D_1, D_2, ... D_k$ and recursively build trees for each subset
2. “Prune” the tree

   • Numeric attribute:
     - sort examples by $a$
     - retain label $y$
     - scan thru once and update the histogram of $y|a < \theta$ and $y/a \geq \theta$ at each point $\theta$
     - pick the threshold $\theta$ with the best entropy score
     - $O(n \log n)$ due to the sort
     - but repeated for each attribute
Scaling up decision tree algorithms

1. Given dataset D:
   - return \( \text{leaf}(y) \) if all examples are in the same class \( y \)
   - pick the best split, on the best attribute \( a \)
     - \( a < \theta \) or \( a \geq \theta \)
     - \( a \) or \( \text{not}(a) \)
     - \( a = c_1 \) or \( a = c_2 \) or ...
     - \( a \in \{c_1, ..., c_k\} \) or not
   - split the data into \( D_1, D_2, ..., D_k \) and recursively build trees for each subset

2. “Prune” the tree

   • Numeric attribute:
     - or, fix a set of possible split-points \( \theta \) in advance
     - scan through once and compute the histogram of \( y \)'s
     - \( O(n) \) per attribute
Scaling up decision tree algorithms

1. Given dataset D:
   - return $leaf(y)$ if all examples are in the same class $y$
   - pick the best split, on the best attribute $a$
     - $a<\theta$ or $a \geq \theta$
     - $a$ or $not(a)$
     - $a=c_1$ or $a=c_2$ or ...
     - $a$ in \{c_1, ..., c_k\} or not
   - split the data into $D_1, D_2, ... D_k$ and recursively build trees for each subset

2. “Prune” the tree

• Subset splits:
  - expensive but useful
  - there is a similar sorting trick that works for the regression case
Scaling up decision tree algorithms

1. Given dataset D:
   - return $\text{leaf}(y)$ if all examples are in the same class $y$
   - pick the best split, on the best attribute $a$
     - $a < \theta$ or $a \geq \theta$
     - $a$ or $\text{not}(a)$
     - $a = c_1$ or $a = c_2$ or ...
     - $a \in \{c_1, ..., c_k\}$ or not
   - split the data into $D_1, D_2, ..., D_k$ and recursively build trees for each subset

2. “Prune” the tree

   - Points to ponder:
     - different subtrees are distinct tasks
     - once the data is in memory, this algorithm is fast.
     - each example appears only once in each level of the tree
     - depth of the tree is usually $O(\log n)$
     - as you move down the tree, the datasets get smaller
Scaling up decision tree algorithms

1. Given dataset D:
   - return $\text{leaf}(y)$ if all examples are in the same class $y$
   - pick the best split, on the best attribute $a$
     - $a < \theta$ or $a \geq \theta$
     - $a$ or $\text{not}(a)$
     - $a = c_1$ or $a = c_2$ or ...
     - $a$ in $\{c_1, ..., c_k\}$ or not
   - split the data into $D_1, D_2, ..., D_k$ and recursively build trees for each subset

2. “Prune” the tree

   • Bottleneck points:
     - what’s expensive is picking the attributes
       - especially at the top levels
     - also, moving the data around
       - in a distributed setting
PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Biswanath Panda, Joshua S. Herbach, Sugato Basu, Roberto J. Bayardo
Google, Inc.

[bpanda, jsherbach, sugato]@google.com, bayardo@alum.mit.edu
**Key ideas**

- A controller to generate Map-Reduce jobs
  - distribute the task of building subtrees of the main decision trees
  - handle “small” and “large” tasks differently
    - small:
      - build tree in-memory
    - large:
      - build the tree (mostly) depth-first
      - send the whole dataset to all mappers and let them classify instances to nodes on-the-fly
Greedy top-down tree assembly

Algorithm 1 InMemoryBuildNode

Require: Node $n$, Data $D \subseteq D^*$
1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$
2: if StoppingCriteria($D_L$) then
3: \hspace{1em} $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$
4: else
5: \hspace{1em} InMemoryBuildNode($n \rightarrow \text{left}, D_L$)
6: \hspace{1em} if StoppingCriteria($D_R$) then
7: \hspace{2em} $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8: \hspace{1em} else
9: \hspace{2em} InMemoryBuildNode($n \rightarrow \text{right}, D_R$)

This algorithm does not scale well for large data sets.
FindBestSplit

• Continuous attributes
  – Treating a point as a boundary (e.g., <5.0 or >=5.0)

• Categorical attributes
  – Membership in a set of values (e.g., is the attribute value one of {Ford, Toyota, Volvo}?)
**FindBestSplit(D)**

- Let $\text{Var}(D)$ be the variance of the output attribute $Y$ measured over all records in $D$ ($D$ refers to the records that are input to a given node). At each step the tree learning algorithm picks a split which maximizes

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

- $\text{Var}(D)$ is the variance of the output attribute $Y$ measured over all records in $D$
- $D_L$ and $D_R$ are the training records in the left and right subtree after splitting $D$ by a predicate
StoppingCriteria(D)

• A node in the tree is not expanded if the number of records in D falls below a threshold.
FindPrediction(D)

- The prediction at a leaf is simply the average of all the Y values in D
What about large data sets?

- Full scan over the input data is slow (required by FindBestSplit)
- Large inputs that do not fit in memory (cost of scanning data from a secondary storage)
- Finding the best split on high dimensional data sets is slow ($2^M$ possible splits for categorical attribute with M categories)
PLANET

- Controller
  - Monitors and controls everything
- MapReduce initialization task
  - Identifies all feature values that need to be considered for splits
- **MapReduce FindBestSplit task (the main part)**
  - MapReduce job to find best split when there’s too much data to fit in memory (builds different parts of the tree)
- MapReduce inMemoryGrow task
  - Task to grow an entire subtree once the data for it fits in memory (in memory MapReduce job)
- ModelFile
  - A file describing the state of the model
PLANET – Controller

• Controls the entire process
• Periodically checkpoints system
• Determine the state of the tree and grows it
  – Decides if nodes should be split
  – If there’s relatively little data entering a node, launch an InMemory MapReduce job to grow the entire subtree
  – For larger nodes, launches MapReduce to find candidates for best split
  – Collects results from MapReduce jobs and chooses the best split for a node
  – Updates model
The controller constructs a tree using a set of MapReduce jobs that are working on different parts of the tree. At any point, the model file contains the entire tree constructed so far.

The controller checks with the model file the nodes at which split predicates can be computed next. For example, if the model has nodes A and B, then the controller can compute splits for C and D.
Two node queues

- MapReduceQueue (MRQ)
  - Contains nodes for which D is too large to fit in memory

- InMemoryQueue (InMemQ)
  - Contains nodes for which D fits in memory
Two MapReduce jobs

• MR_ExpandNodes
  – Process nodes from the MRQ. For a given set of nodes N, computes a candidate of good split predicate for each node in N.

• MR_InMemory
  – Process nodes from the InMemQ. For a given set of nodes N, completes tree induction at nodes in N using the InMemoryBuildNode algorithm.
Walkthrough

Using PLANET to construct the tree:

Assumptions:
1. D* with 100 records;
2. Tree induction stops once the # of training records at a node falls below 10;
3. A memory constraint limiting the algorithm to operating on inputs of size 25 records or less.
Walkthrough

1. Initially: M, MRQ, and InMemQ are empty.
   - Controller can only expand the root
   - Finding the split for the root requires the entire training set $D^*$ of 100 records (does not fit in memory)

2. A is pushed onto MRQ; InMemQ stays empty.
Walkthrough

1. Quality of the split
2. Predictions in L and R branches
3. Number of training records in L and R branches
Walkthrough

- Controller selects the best split
  - Do any branches match stopping criteria? If so, don’t add any new nodes
Walkthrough

- Expand C and D
  - The controller schedules a single MR_ExpandNodes for C and D that is done in a single step
  - PLANET expands trees breadth first as opposed to the depth first process used by the inMemory algorithm.
Walkthrough

• Scheduling jobs simultaneously

Once both jobs return, the subtrees rooted at G, F, and E are complete. The controller continues the process with the children of H only.
Figure 3: Running time vs data size for various numbers of machines.
Figure 4: Running time vs tree depth. Note: the Sampled R curve was trained on 1/30 of the data used for the other curves.
And now for something completely different...
What’s left?
Tomorrow

- Alternative distributed frameworks
  - Hadoop
  - Spark
  - ??? [tune in to find out!]
Next week

- Project presentations!
  - 25-30 minute talk
  - 5-10 minutes of questions
- Tuesday, April 21
  - Joey Ruberti and Michael Church
  - Zhaochong Liu
- Wednesday, April 22
  - Bita Kazemi and Alekhya Chennupati
- Thursday, April 23
  - Roi Ceren, Will Richardson, Muthukumaran Chandrasekaran
  - Ankita Joshi, Bahaa AlAila, Manish Ranjan
Finals week

• NO FINAL

• Final project write-up: Friday, May 1
  – NIPS format
  – [https://nips.cc/Conferences/2015/PaperInformation/StyleFiles](https://nips.cc/Conferences/2015/PaperInformation/StyleFiles)
  – 6-10 pages (not including references)
  – Email or BitBucket