Decision Trees on MapReduce

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu
Give one attribute (e.g., lifespan), try to predict the value of new people’s lifespans by means of some of the other available attribute attribute

Input attributes:
- d features/attributes: \( x^{(1)}, x^{(2)}, \ldots x^{(d)} \)
- Each \( x^{(j)} \) has domain \( O_j \)
  - Categorical: \( O_j = \{ \text{red}, \text{blue} \} \)
  - Numerical: \( H_j = (0, 10) \)
- \( Y \) is output variable with domain \( O_Y \):
  - Categorical: Classification, Numerical: Regression

Data \( D \):
- \( n \) examples \((x_i, y_i)\) where \( x_i \) is a \( d \)-dim feature vector, \( y_i \in O_Y \) is output variable

Task:
- Given an input data vector \( x \) predict \( y \)
A Decision Tree is a tree-structured plan of a set of attributes to test in order to predict the output.
Decision Trees (1)

- **Decision trees:**
  - Split the data at each internal node
  - Each leaf node makes a prediction

- **Lecture today:**
  - Binary splits: $X^{(i)} < v$
  - Numerical attrs.
  - Regression
How to make predictions?

- **Input:** Example $x_i$
- **Output:** Predicted $y_i'$

- “Drop” $x_i$ down the tree until it hits a leaf node
- Predict the value stored in the leaf that $x_i$ hits
Decision Trees Vs. SVM

- **Alternative view:**

![Diagram showing decision tree and data points in different regions defined by decision boundaries](image-url)
How to construct a tree?
How to construct a tree?

- Training dataset $D^*$, $|D^*| = 100$ examples

![Diagram of a tree structure with labeled nodes and edges, showing the number of examples traversing each edge.]

2/18/2015

How to construct a tree?

- Imagine we are currently at some node $G$
  - Let $D_G$ be the data that reaches $G$
- There is a decision we have to make: Do we continue building the tree?
  - If yes, which variable and which value do we use for a split?
    - Continue building the tree recursively
  - If not, how do we make a prediction?
    - We need to build a “predictor node”
3 steps in constructing a tree

Algorithm 1: BuildSubtree

Require: Node $n$, Data $D \subseteq D^*$

1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$ (1)
2: if $\text{StoppingCriteria}(D_L)$ then (2)
3: $n \rightarrow \text{left prediction} = \text{FindPrediction}(D_L)$ (3)
4: else
5: $(n \rightarrow \text{left}, D_L)$
6: if $\text{StoppingCriteria}(D_R)$ then
7: $n \rightarrow \text{right prediction} = \text{FindPrediction}(D_R)$
8: else
9: $(n \rightarrow \text{right}, D_R)$

- Requires at least a single pass over the data!
How to construct a tree?

(1) How to split? Pick attribute & value that optimizes some criterion

- Regression: Purity
  - Find split \((X^{(i)}, v)\) that creates \(D, D_L, D_R\): parent, left, right child datasets and maximizes:
    \[
    |D| \cdot Var(D) - \left( |D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R) \right)
    \]
  - \(Var(D) = \frac{1}{n} \sum_{i \in D} (y_i - \bar{y})^2 \) ... variance of \(y_i\) in \(D\)
(1) How to split? Pick attribute & value that optimizes some criterion

- **Classification:** Information Gain
  - Measures how much a given attribute $X$ tells us about the class $Y$
  - $IG(Y \mid X)$: We must transmit $Y$ over a binary link. How many bits on average would it save us if both ends of the line knew $X$?
**Why Information Gain? Entropy**

- **Entropy**: What’s the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from X’s distribution?

- **The entropy of X**: \( H(X) = - \sum_{j=1}^{m} p_j \log p_j \)
  - “High Entropy”: X is from a uniform (boring) distribution
    - A histogram of the frequency distribution of values of X is **flat**
  - “Low Entropy”: X is from a varied (peaks/valleys) distrib.
    - A histogram of the frequency distribution of values of X would have many lows and one or two highs
Suppose I want to predict $Y$ and I have input $X$

- $X = \text{College Major}$
- $Y = \text{Likes “Gladiator”}$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
</tbody>
</table>

From this data we estimate

- $P(Y = Yes) = 0.5$
- $P(X = Math \& Y = No) = 0.25$
- $P(X = Math) = 0.5$
- $P(Y = Yes \mid X = History) = 0$

Note:

- $H(Y) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$
- $H(X) = 1.5$
Why Information Gain? Entropy

- Suppose I want to predict $Y$ and I have input $X$
  - $X =$ College Major
  - $Y =$ Likes “Gladiator”

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
</tbody>
</table>

- **Def: Specific Conditional Entropy**
  - $H(Y \mid X=v) =$ The entropy of $Y$ among only those records in which $X$ has value $v$

- **Example:**
  - $H(Y \mid X = Math) = 1$
  - $H(Y \mid X = History) = 0$
  - $H(Y \mid X = CS) = 0$
Suppose I want to predict $Y$ and I have input $X$

- $X = \text{College Major}$
- $Y = \text{Likes “Gladiator”}$

**Def: Conditional Entropy**

$$H(Y \mid X) = \text{The average specific conditional entropy of } Y$$

- $= \text{if you choose a record at random what will be the conditional entropy of } Y$, conditioned on that row’s value of $X$
- $= \text{Expected number of bits to transmit } Y$ if both sides will know the value of $X$

$$= \sum_j P(X = v_j)H(Y \mid X = v_j)$$
Suppose I want to predict $Y$ and I have input $X$.

- **$H(Y \mid X)$** = The average specific conditional entropy of $Y$
  
  $$H(Y \mid X) = \sum_j P(X = v_j)H(Y \mid X = v_j)$$

**Example:**

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$v_j$</th>
<th>$P(X = v_j)$</th>
<th>$H(Y \mid X = v_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>History</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>CS</td>
<td>0.25</td>
<td>0</td>
</tr>
</tbody>
</table>

**So:** $H(Y \mid X) = 0.5 \times 1 + 0.25 \times 0 + 0.25 \times 0 = 0.5$
Why Information Gain?

- Suppose I want to predict $Y$ and I have input $X$

- Def: Information Gain
  - $IG(Y|X) = I \text{ must transmit } Y$. How many bits on average would it save me if both ends of the line knew $X$?
  - $IG(Y|X) = H(Y) - H(Y | X)$

### Table

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
</tbody>
</table>

- Example:
  - $H(Y) = 1$
  - $H(Y | X) = 0.5$
  - Thus $IG(Y | X) = 1 - 0.5 = 0.5$
Suppose you are trying to predict whether someone is going live past 80 years

From historical data you might find:

- \( IG(\text{LongLife} \mid \text{HairColor}) = 0.01 \)
- \( IG(\text{LongLife} \mid \text{Smoker}) = 0.3 \)
- \( IG(\text{LongLife} \mid \text{Gender}) = 0.25 \)
- \( IG(\text{LongLife} \mid \text{LastDigitOfSSN}) = 0.00001 \)

IG tells us how much information about \( Y \) is contained in \( X \)

- So attribute \( X \) that has high \( IG(Y \mid X) \) is a good split!
3 steps in constructing a tree

**Algorithm 1**

**BuildSubtree**

**Require**: Node $n$, Data $D \subseteq D^*$

1. $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$ **(1)**
2. if StoppingCriteria($D_L$) then **(2)**
3. $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$ **(3)**

4. else
5. $(n \rightarrow \text{left}, D_L)$
6. if StoppingCriteria($D_R$) then
7. $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8. else
9. $(n \rightarrow \text{right}, D_R)$

**BuildSubtree**
(2) When to stop?

- Many different heuristic options
- Two ideas:
  - (1) When the leaf is "pure"
    - The target variable does not vary too much: \( \text{Var}(y_i) < \varepsilon \)
  - (2) When # of examples in the leaf is too small
    - For example, \(|D| \leq 100\)
How to predict?

Many options

**Regression:**
- Predict average $y_i$ of the examples in the leaf
- Build a linear regression model on the examples in the leaf

**Classification:**
- Predict most common $y_i$ of the examples in the leaf
Building Decision Trees Using MapReduce
Problem: Building a tree

- Given a large dataset with hundreds of attributes
- Build a decision tree!

General considerations:

- **Tree is small** (can keep it in memory):
  - Shallow (~10 levels)
- Dataset too large to keep in memory
- Dataset too big to scan over on a single machine
- **MapReduce to the rescue!**

**Algorithm 1: BuildSubTree**

<table>
<thead>
<tr>
<th>Require: Node n, Data D ⊆ D*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: (n → split, D_L, D_R) = FindBestSplit(D)</td>
</tr>
<tr>
<td>2: if StoppingCriteria(D_L) then</td>
</tr>
<tr>
<td>3: n → left prediction = FindPrediction(D_L)</td>
</tr>
<tr>
<td>4: else</td>
</tr>
<tr>
<td>5: BuildSubTree(n → left, D_L)</td>
</tr>
<tr>
<td>6: if StoppingCriteria(D_R) then</td>
</tr>
<tr>
<td>7: n → right prediction = FindPrediction(D_R)</td>
</tr>
<tr>
<td>8: else</td>
</tr>
<tr>
<td>9: BuildSubTree(n → right, D_R)</td>
</tr>
</tbody>
</table>
Parallel Learner for Assembling Numerous Ensemble Trees [Panda et al., VLDB ‘09]

- A sequence of MapReduce jobs that builds a decision tree

**Setting:**

- Hundreds of **numerical** (discrete & continuous, but not categorical) attributes
- Target variable is **numerical**: Regression
- Splits are **binary**: $X^{(j)} < v$
- Decision tree is small enough for each Mapper to keep it in memory
- Data too large to keep in memory
MapReduce: Given a set of split candidates compute their quality

Keeps track of the model and decides how to grow the tree
The tree will be built in levels

One level at a time:

1) Master decides which nodes/splits to consider, MapReduce computes quality of those splits

2) Master then grows the tree for a level

Goto 1)
Decision trees on MapReduce

Hard part: Computing “quality” of a split
1) Master tells the Mappers which splits \((X^{(i)}, v)\) to consider
2) Each Mapper gets a subset of data and computes partial statistics for a given split
3) Reducers collect partial statistics and output the final quality for a given split \((X^{(i)}, v)\)
4) Master makes final decision where to split
We build the tree level by level
- One MapReduce step builds one level of the tree

Mapper
- Considers a number of possible splits \((X^{(i)},v)\) on its subset of the data
- For each split it stores partial statistics
- Partial split-statistics is sent to Reducers

Reducer
- Collects all partial statistics and determines best split

Master grows the tree for one level
**Mapper** loads the *model* and info about which *attribute splits* to consider
- Each mapper sees a subset of the data $D^*$
- Mapper “drops” each datapoint to find the appropriate leaf node $L$
- For each leaf node $L$ it keeps statistics about
  - (1) the data reaching $L$
  - (2) the data in left/right subtree under split $S$

**Reducer** aggregates the statistics (1), (2) and determines the best split for each tree node
PLANT: Components

- **Master**
  - Monitors everything (runs multiple MapReduce jobs)

- **Three types of MapReduce jobs:**
  - **(1) MapReduce Initialization** (run once first)
    - For each attribute identify values to be considered for splits
  
  - **(2) MapReduce FindBestSplit** (run multiple times)
    - MapReduce job to find best split (when there is too much data to fit in memory)
  
  - **(3) MapReduce InMemoryBuild** (run once last)
    - Similar to BuildSubTree (but for small data)
    - Grows an entire sub-tree once the data fits in memory

- **Model file**
  - A file describing the state of the model
**PLANET: Components**

1. Master Node
2. MapReduce **Initialization** (run once first)
3. MapReduce **FindBestSplit** (run multiple times)
4. MapReduce **InMemoryBuild** (run once last)
**PLANET: Master**

- **Master controls the entire process**
- **Determines the state of the tree and grows it:**
  - **(1)** Decides if nodes should be split
  - **(2)** If there is little data entering a tree node, Master runs an `InMemoryBuild` MapReduce job to grow the entire subtree below that node
  - **(3)** For larger nodes, Master launches MapReduce `FindBestSplit` to evaluate candidates for best split
    - Master also collects results from `FindBestSplit` and chooses the best split for a node
  - **(4)** Updates the model
PLANE T: Components

(1) Master Node

(2) MapReduce **Initialization** (run once first)

(3) MapReduce **FindBestSplit** (run multiple times)

(4) MapReduce **InMemoryBuild** (run once last)
Initialization job: Identifies all the attribute values which need to be considered for splits
- Initialization process generates “attribute metadata” to be loaded in memory by other tasks

Main question:
Which splits to even consider?
Which splits to even consider?

- For small data we can sort the values along a particular feature and consider every possible split.
- But data values may not be uniformly populated so many splits may not really make a difference.

\[ X^{(j)}: 1.2, 1.3, 1.4, 1.6, 2.1, 7.2, 8.1, 9.8, 10.1, 10.2, 10.3, 10.4, 11.5, 11.7, 12.8, 12.9 \]

Idea: Consider a limited number of splits such that splits “move” about the same amount of data.
Splits for numerical attributes:

- For attribute $X^{(i)}$ we would like to consider every possible value $v \in O_j$
- Compute an approx. equi-depth histogram on $D^*$
  - **Idea:** Select buckets such that counts per bucket are equal

Use boundary points of histogram as splits
**Goal:** Equal number of elements per bucket (\(B\) buckets total)

- Construct by first **sorting** and then taking **\(B-1\)** equally-spaced splits

**Faster construction:**
Sample & take equally-spaced splits in the sample
- Nearly equal buckets
PLANET: Components

(1) Master Node
(2) MapReduce Initialization (run once first)
(3) MapReduce FindBestSplit (run multiple times)
(4) MapReduce InMemoryBuild (run once last)
Goal: For a particular split node $j$ find attribute $X^{(j)}$ and value $v$ that maximizes Purity:

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

- $D$ ... training data $(x_i, y_i)$ reaching the node $j$
- $D_L$ ... training data $x_i$, where $x_i^{(j)} < v$
- $D_R$ ... training data $x_i$, where $x_i^{(j)} \geq v$

$Var(D) = \frac{1}{n} \sum_{i \in D} y_i^2 - \left(\frac{1}{n} \sum_{i \in D} y_i\right)^2$
To compute Purity we need

- \( \text{Var}(D) = \frac{1}{n} \sum_i y_i^2 - \left( \frac{1}{n} \sum_i y_i \right)^2 \)

Important observation: Variance can be computed from sufficient statistics: \( N, S=\sum y_i, Q=\sum y_i^2 \)

- Each Mapper processes subset of data \( D_m \), and computes \( N_m, S_m, Q_m \) for its own \( D_m \)
- Reducer combines the statistics and computes global variance and then Purity:

- \( \text{Var}(D) = \frac{1}{\sum_m N_m} \sum_m Q_m - \left( \frac{1}{\sum_m N_m} \sum_m S_m \right)^2 \)
FindBestSplit: Map

- **Mapper:**
  - Initialized by loading results of *Initialization task*
    - **Current model** (to find which node each datapoint $x_i$ ends up)
    - **Attribute metadata** (all split points for each attribute)
  - For each data record run the Map algorithm:
    - For each node store statistics of the data entering the node and at the end emit (to all reducers):
      - $<\text{NodeID}, \{ S=\Sigma y, Q=\Sigma y^2, N=\Sigma 1 \}>$
    - For each split store statistics and at the end emit:
      - $<\text{SplitID}, \{ S, Q, N \}>$
      - $\text{SplitID} = (\text{node } n, \text{ attribute } X^{(i)}, \text{ split value } v)$
Reducer:

1. Load all the $\langle$NodeID, List $\{S_m, Q_m, N_m\}$$\rangle$ pairs and aggregate the per node statistics.
2. For all the $\langle$SplitID, List $\{S_m, Q_m, N_m\}$$\rangle$ aggregate the statistics.

$$Var(D) = \frac{1}{\sum_m N_m} \sum_m Q_m - \left(\frac{1}{\sum_m N_m} \sum_m S_m\right)^2$$

For each NodeID, output the best split found.
**Overall system architecture**

- **Master gives the mappers:**
  1. Tree
  2. Set of nodes
  3. Set of candidate splits

**Nodes:** F, G, H, I

**Split candidates:**
- \((X^{(1)}, v^{(1)})\)
- \((X^{(1)}, v^{(2)})\)
- \((X^{(3)}, v^{(3)})\)
- \((X^{(3)}, v^{(4)})\)

Mappers output 2 types of key-value pairs:
- (NodeID: S, Q, N)
- (NodeID, Split: S, Q, N)

For every (NodeID, Split)
Reducer(s) compute the purity and output the best split.
Example: Need to split nodes F, G, H, I

Map and Reduce:

- **FindBestSplit::Map** (each mapper)
  - Load the current model \( M \)
  - Drop every example \( x_i \) down the tree
  - If it hits \( G \) or \( H \), update in-memory hash tables:
    - For each node: \( T_n: (\text{Node}) \rightarrow \{S, Q, N\} \)
    - For each (Split, Node): \( T_{n,j,s}: (\text{Node, Attribute, SplitValue}) \rightarrow \{S, Q, N\} \)
  - **Map::Finalize**: output the key-value pairs from above hashtables

- **FindBestSplit::Reduce** (each reducer)
  - Collect:
    - \( T_1: <\text{Node, List}\{S, Q, N\}> \rightarrow <\text{Node, \{\Sigma S, \Sigma Q, \Sigma N\}}> \)
    - \( T_2: <(\text{Node, Attr. Split}), \text{List}\{S, Q, N\}> \rightarrow <(\text{Node, Attr. Split}), \{\Sigma S, \Sigma Q, \Sigma N\}> \)
  - Compute impurity for each node using \( T_1, T_2 \)
  - Return **best split** to Master (which then decides on globally best split)
- Collects outputs from FindBestSplit reducers
  \(<\text{Split.NodeID, Attribute, Value, Impurity}>\)

- For each node decides the best split
  - If data in \(D_L/D_R\) is small enough, later run a MapReduce job \textbf{InMemoryBuild} on the node
  - Else run MapReduce \textbf{FindBestSplit} job for both nodes
Decision Trees: Conclusion
Decision Trees

- Decision trees are the single most popular data mining tool:
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap
  - It’s possible to get in trouble with overfitting
  - They do classification as well as regression!
Learn multiple trees and combine their predictions

- Gives better performance in practice

Bagging:

- Learns multiple trees over independent samples of the training data
  - For a dataset $D$ on $n$ data points: Create dataset $D'$ of $n$ points but sample from $D$ with replacement:
    - 33% points in $D'$ will be duplicates, 66% will be unique
  - Predictions from each tree are averaged to compute the final model prediction
How to create random samples of D*?

- Compute a hash of a training record’s id and tree id
- Use records that hash into a particular range to learn a tree
- This way the same sample is used for all nodes in a tree

**Note:** This is sampling D* without replacement (but samples of D* should be created with replacement)
SVM vs. DT

**SVM**
- **Classification**
  - Usually only 2 classes
- **Real valued features** (no categorical ones)
- **Tens/hundreds of thousands of features**
- **Very sparse features**
- **Simple decision boundary**
  - No issues with overfitting

**Example applications**
- Text classification
- Spam detection
- Computer vision

**Decision trees**
- **Classification & Regression**
  - Multiple (~10) classes
- **Real valued and categorical features**
- **Few (hundreds) of features**
- **Usually dense features**
- **Complicated decision boundaries**
  - Overfitting! Early stopping

**Example applications**
- User profile classification
- Landing page bounce prediction
References
