Decision Trees on MapReduce

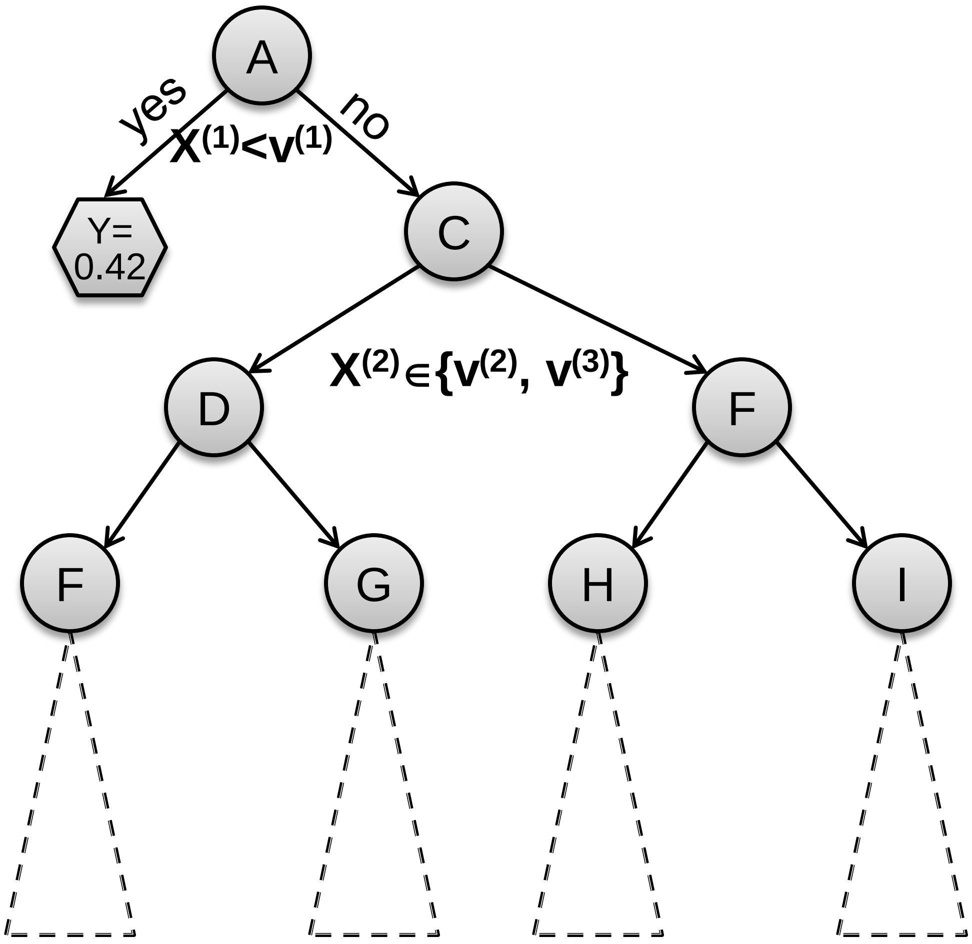
CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

Decision Tree Learning

- **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**
- **Input attributes:**
  - **d** features/attributes: $\mathbf{x^{(1)}, x^{(2)}, \ldots x^{(d)}}$
  - Each $\mathbf{x^{(j)}}$ has **domain** $\mathbf{O_j}$
    - **Categorical:** $\mathbf{O_j}$ = {red, blue}
    - **Numerical:** $\mathbf{H_j}$ = (0, 10)
  - $\boldsymbol{Y}$ is output variable with domain $\boldsymbol{O_Y}$:
    - **Categorical:** Classification, **Numerical:** Regression
- **Data D:**
  - $\boldsymbol{n}$ examples $(\boldsymbol{x_i}, \boldsymbol{y_i})$ where $\boldsymbol{x_i}$ is a $\boldsymbol{d}$-dim feature vector, $\boldsymbol{y_i} \in \boldsymbol{O_Y}$ is output variable
- **Task:**
  - Given an input data vector $\boldsymbol{x}$ predict $\boldsymbol{y}$

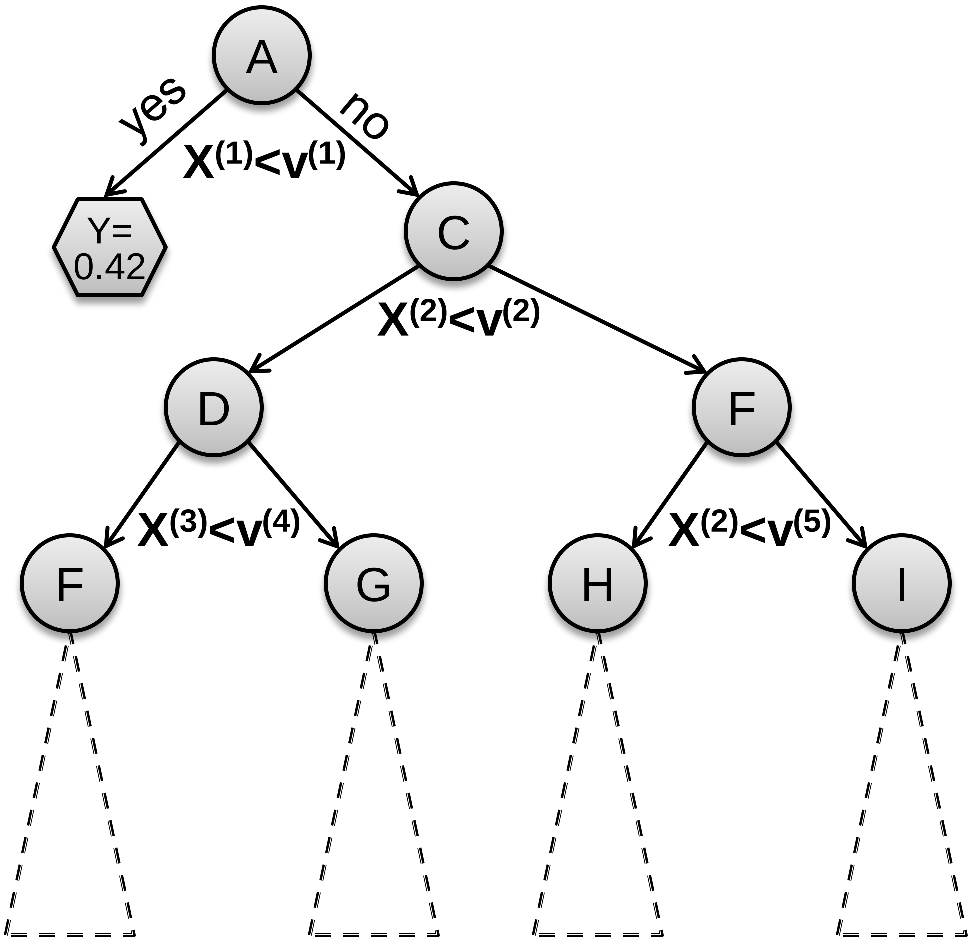
Decision Trees

- 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: $\mathbf{X^{(j)}<v}$
  - Numerical attrs.
  - Regression

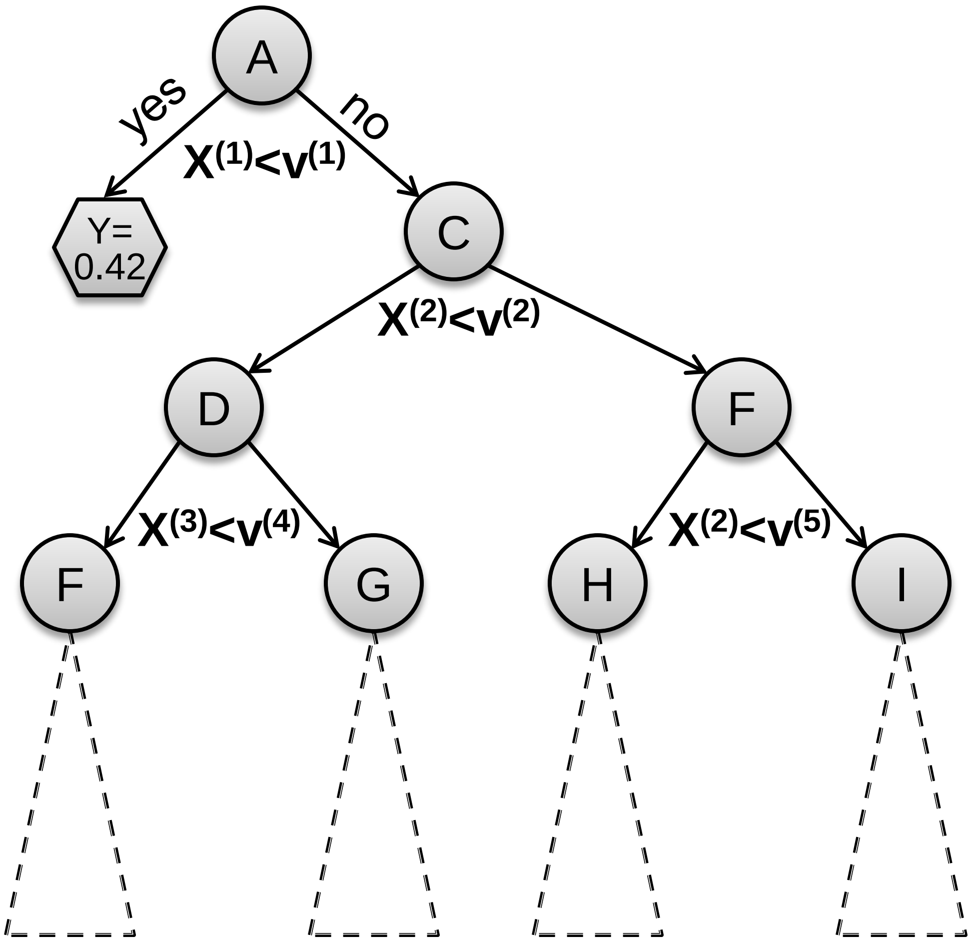


How to make predictions?

- **Input:** Example $\boldsymbol{x_i}$
- **Output:** Predicted $\boldsymbol{y_i'}$

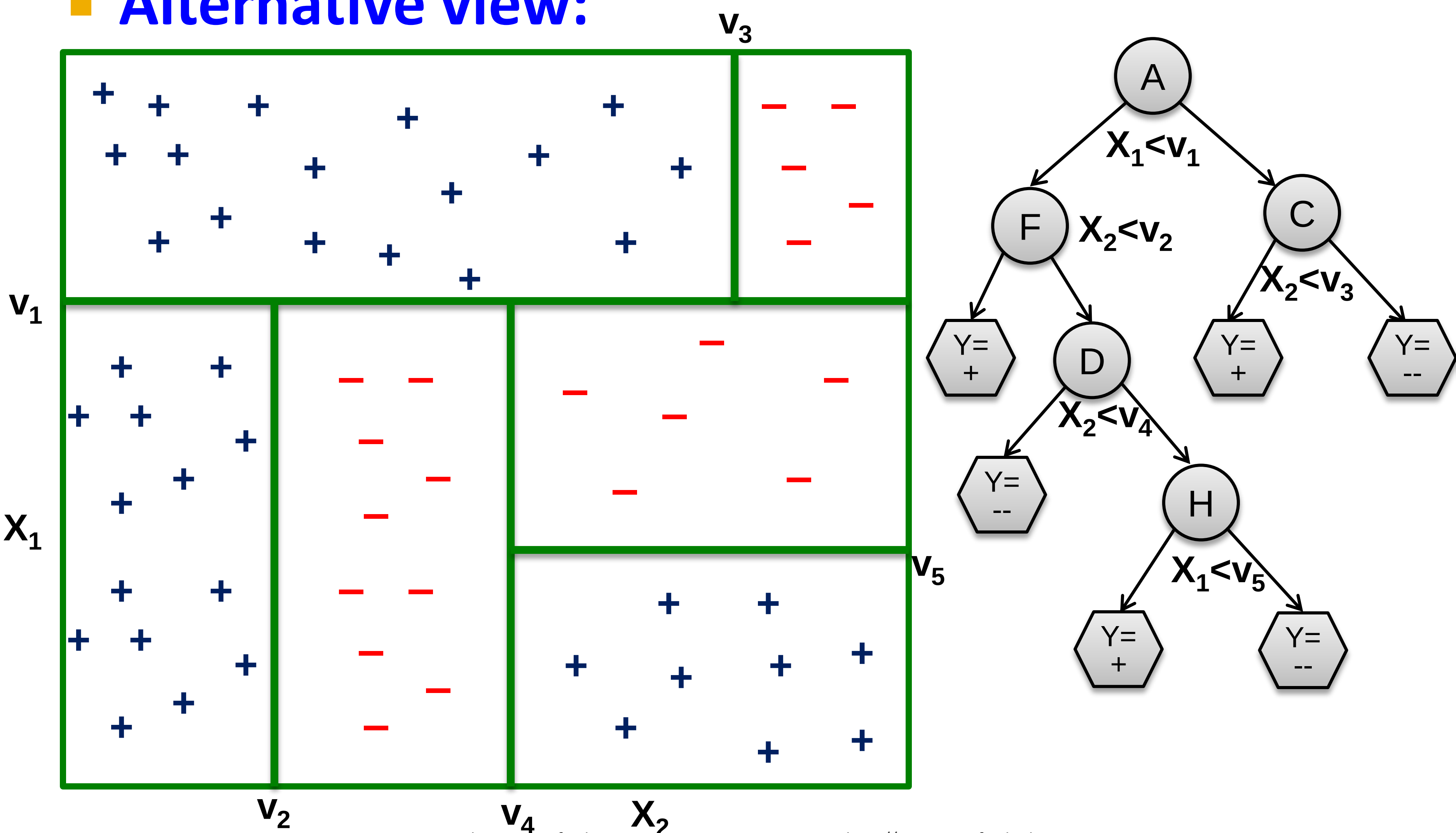
- "Drop" $\boldsymbol{x_i}$ down the tree until it hits a leaf node

- Predict the value stored in the leaf that $\boldsymbol{x_i}$ hits



Decision Trees Vs. SVM

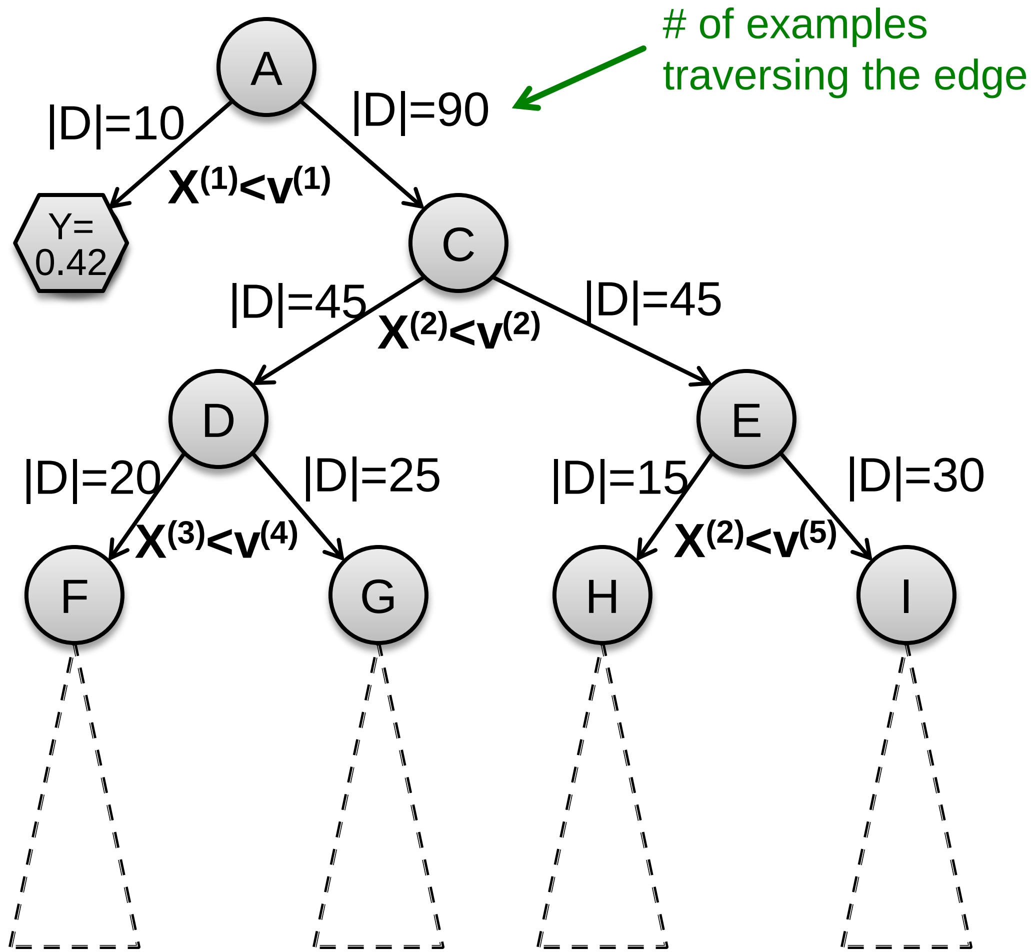
- **Alternative view:**



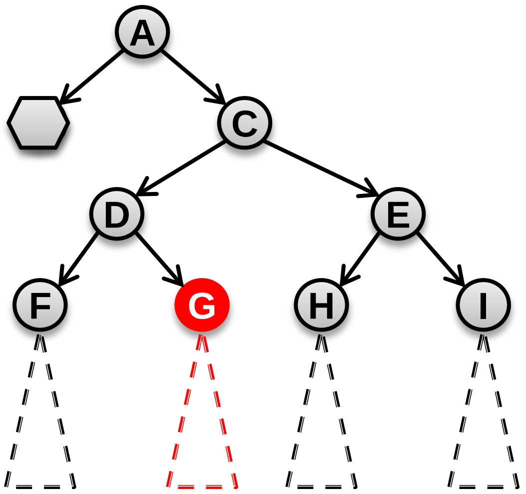
How to construct a tree?

How to construct a tree?

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



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 $\boxed{\textbf{BuildSubtree}}$

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

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

- **Requires at least a single pass over the data!**

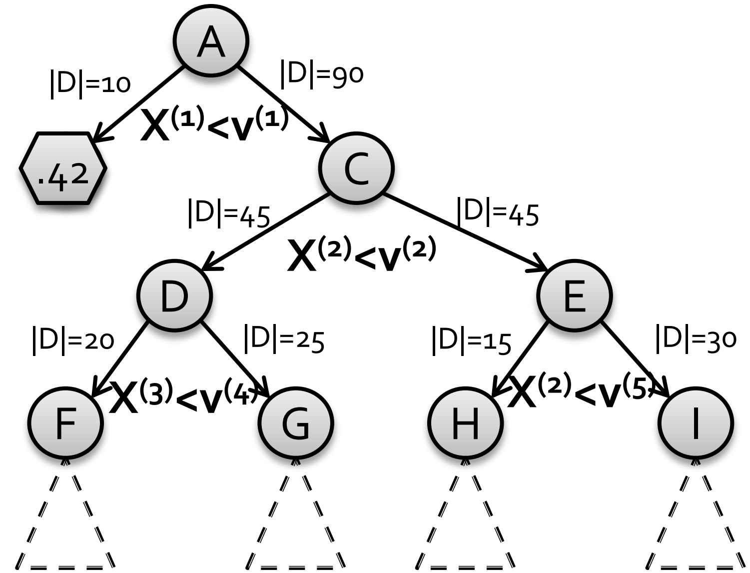
How to construct a tree?

(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**:

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

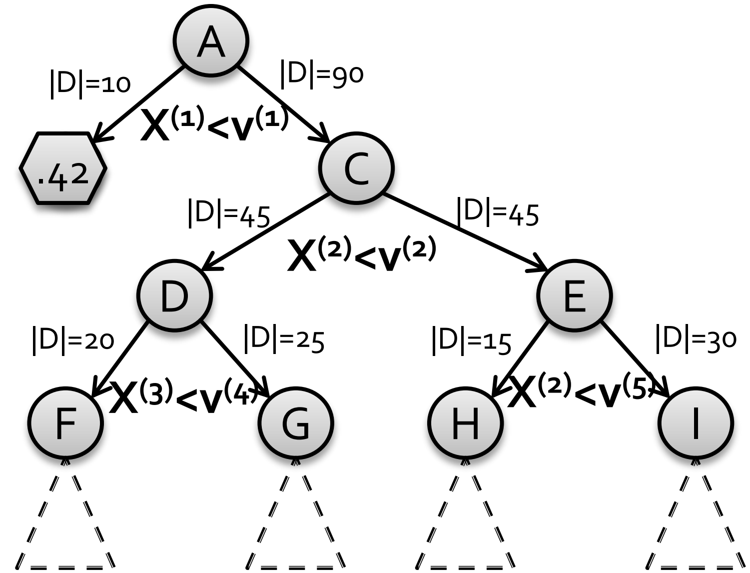
    - $Var(D) = \frac{1}{n}\sum_{i \in D}(y_i - \overline{y})^2$ … variance of y_i in D



How to construct a tree?

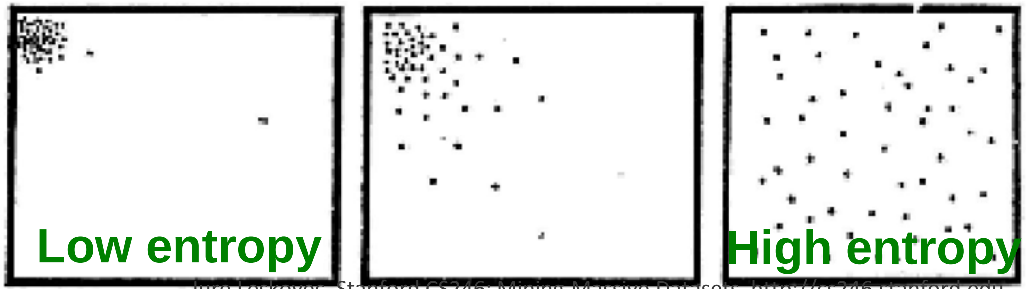
(1) How to split? **Pick attribute & value that optimizes some criterion**

- **Classification:** **Information Gain**
  - **Measures how much a given attribute X tells us about the class Y**
  - **IG(Y | 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



Why Information Gain? Entropy

- **Suppose I want to predict Y and I have input X**
  - X = College Major
  - Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

- **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:**
  - $H(Y) = -½\log_2(½) - ½\log_2(½) = \mathbf{1}$
  - $H(X) = 1.5$

Why Information Gain? Entropy

- **Suppose I want to predict Y and I have input X**
  - X = College Major
  - Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

- **Def: Specific Conditional Entropy**
  - **H(Y | X=v)** = The entropy of **Y** among only those records in which **X** has value **v**
  - **Example:**
    - $H(Y|X = Math) = 1$
    - $H(Y|X = History) = 0$
    - $H(Y|X = CS) = 0$

Why Information Gain?

- **Suppose I want to predict Y and I have input X**
  - X = College Major
  - Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

- **Def: Conditional Entropy**
  - $H(Y \mid X)$ $=$ The average specific conditional entropy of **Y**
    - **=** if you choose a record at random what will be the conditional entropy of **Y**, conditioned on that row's value of **X**
    - **=** Expected number of bits to transmit **Y** if both sides will know the value of **X**
  - **=** $\sum_j P(X = v_j) H(Y|X = v_j)$

Why Information Gain?

- **Suppose I want to predict Y and I have input X**
  - $H(Y \mid X)$ = The average specific conditional entropy of Y

$$= \sum_j P(X = v_j) H(Y|X = v_j)$$

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

  - **Example:**

V_j	$P(X=v_j)$	$H(Y\|X=v_j)$
Math	0.5	1
History	0.25	0
CS	0.25	0

  - **So:** H(Y|X)=0.5*1+0.25*0+0.25*0 = **0.5**

Why Information Gain?

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

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

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

$$IG(Y|X) = H(Y) - H(Y \mid X)$$

- **Example:**
  - H(Y) = 1
  - H(Y|X) = 0.5
  - Thus **IG(Y|X) = 1 – 0.5 = 0.5**

What is Information Gain used for?

- **Suppose you are trying to predict whether someone is going live past 80 years**
- From historical data you might find:
  - **IG(LongLife | HairColor) = 0.01**
  - **IG(LongLife | Smoker) = 0.3**
  - **IG(LongLife | Gender) = 0.25**
  - **IG(LongLife | LastDigitOfSSN) = 0.00001**
- **IG tells us how much information about Y is contained in X**
  - **So attribute X that has high IG(Y|X) is a good split!**

3 steps in constructing a tree

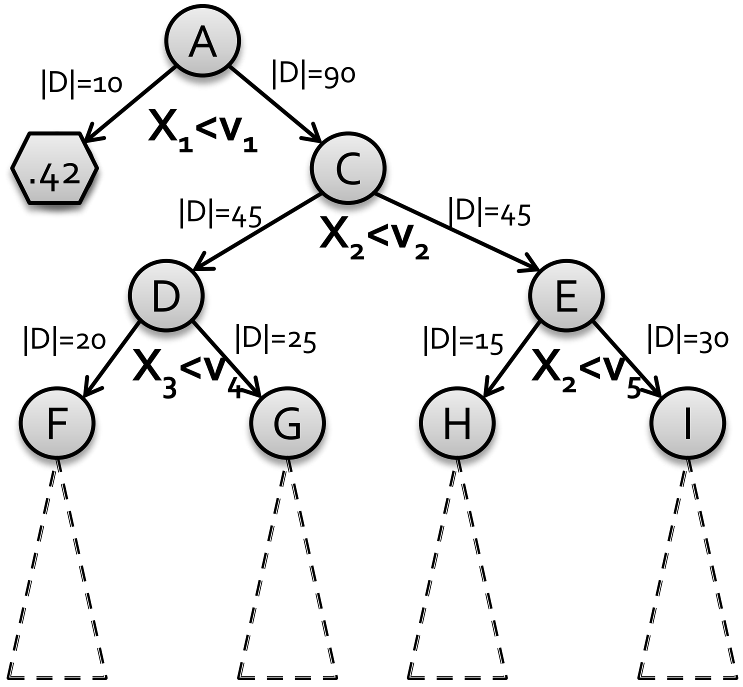
Algorithm 1 BuildSubtree

```
Require: Node n, Data D ⊆ D*
(n →split, D_L, D_R) = FindBestSplit(D)                (1)
if StoppingCriteria(D_L) then                          (2)
    n →left_prediction = FindPrediction(D_L)           (3)
else
    BuildSubtree (n →left, D_L)
if StoppingCriteria(D_R) then
    n →right_prediction = FindPrediction(D_R)
else
    BuildSubtree (n →right, D_R)
```

When to stop?

(2) When to stop?

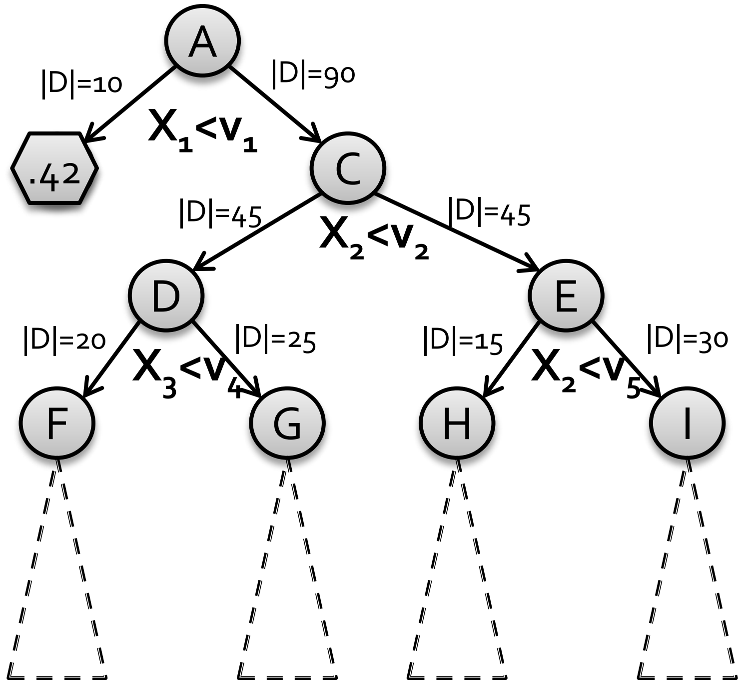
- Many different heuristic options
- **Two ideas:**
  - **(1) When the leaf is "pure"**
    - The target variable does not vary too much: $Var(y_i) < \varepsilon$
  - **(2) When # of examples in the leaf is too small**
    - For example, $|D| \leq 100$



How to predict?

(3) 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 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**

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

```
(n →split, D_L, D_R)=FindBestSplit(D)
if StoppingCriteria(D_L) then
    n →left_prediction=FindPrediction(D_L)
else
    BuildSubTree(n →left, D_L)
if StoppingCriteria(D_R) then
    n →right_prediction=FindPrediction(D_R)
else
    BuildSubTree(n →right, D_R)
```

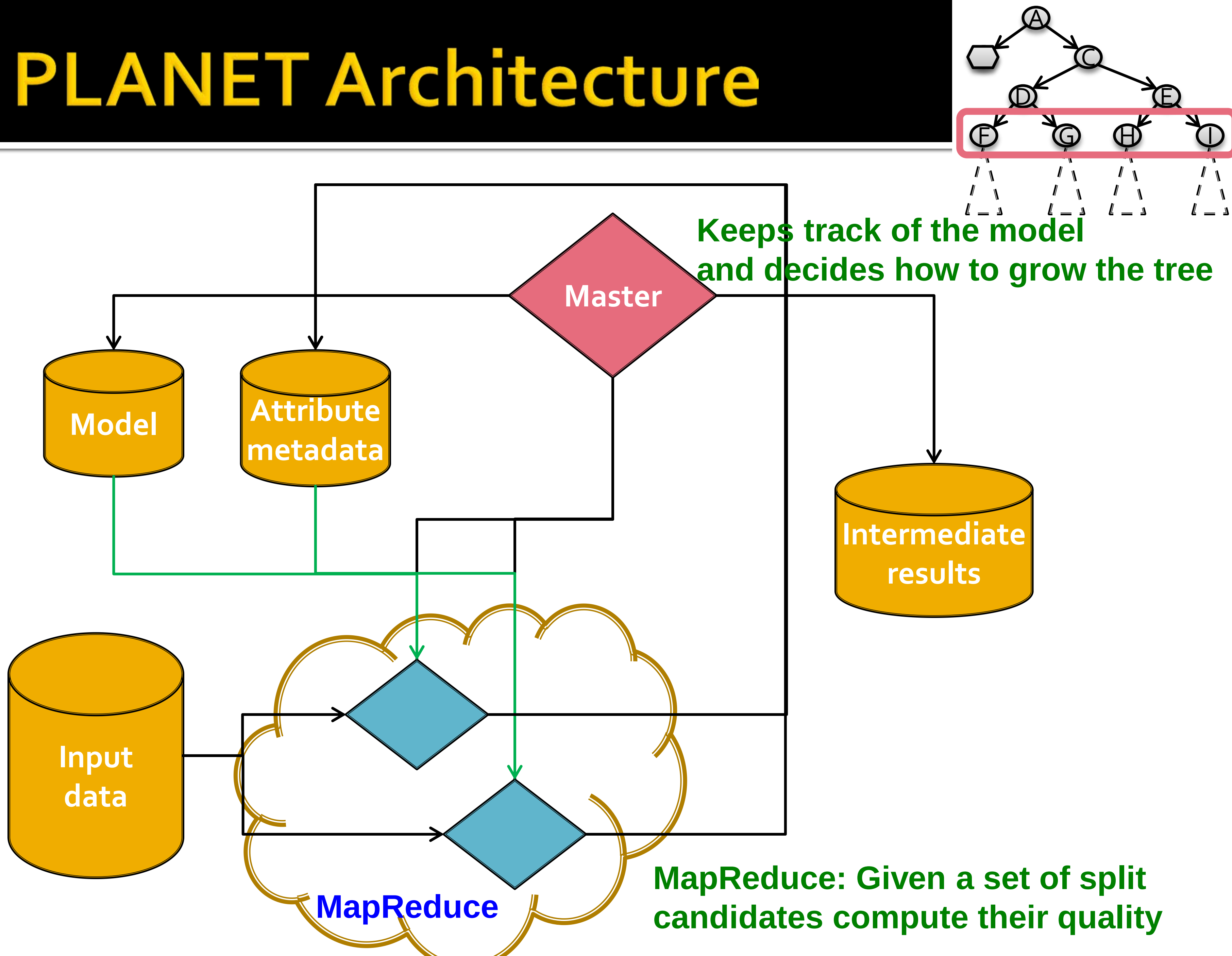
Today's Lecture: PLANET

Parallel **L**earner for **A**ssembling **N**umerous **E**nsemble **T**rees [Panda et al., VLDB '09]

- A **sequence** of MapReduce jobs that builds a decision tree

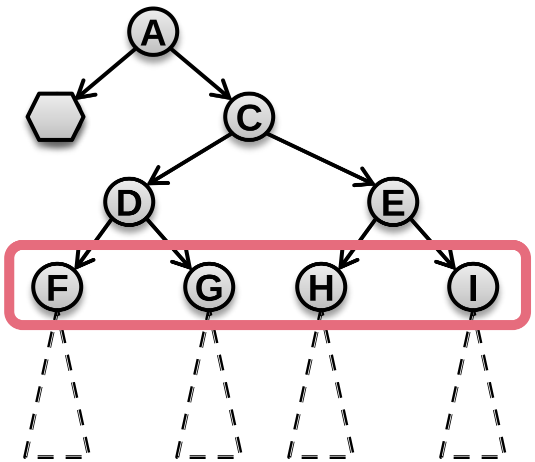
- **Setting:**
  - Hundreds of **numerical** (discrete & continuous, but not categorical) attributes
  - Target variable is **numerical**: **Regression**
  - Splits are **binary**: $X^{(j)} < v$
  - **Decision tree is small enough for each Mapper to keep it in memory**
  - **Data too large to keep in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

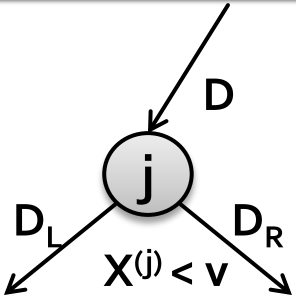
PLANET Architecture



PLANET: Building the Tree

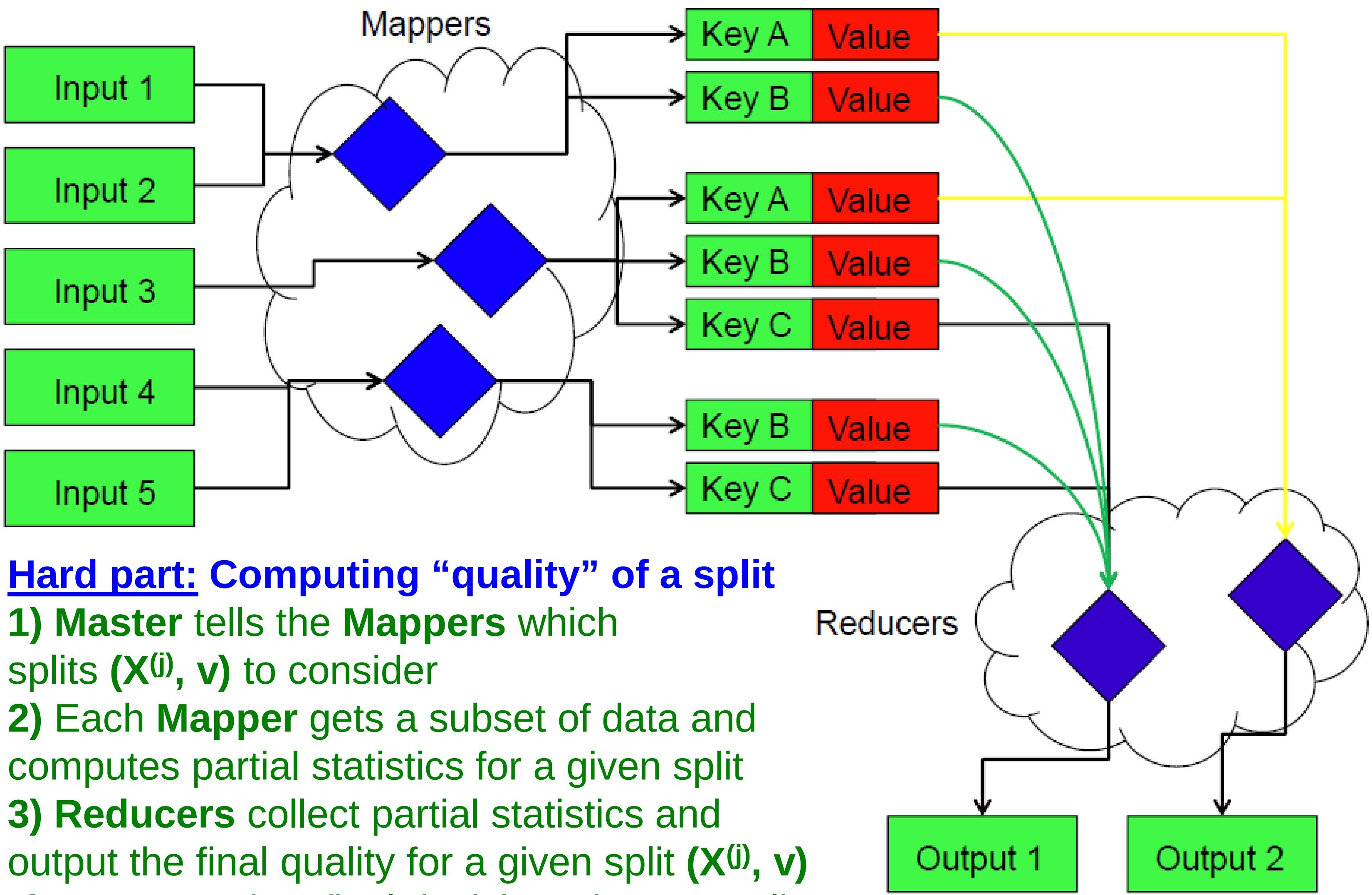
- **The tree will be built in levels**
  - **One level at a time:**





- **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

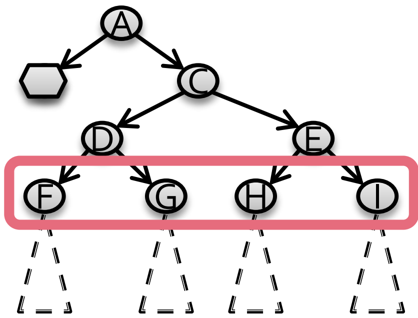
1) Master tells the **Mappers** which splits **($X^{(j)}$, v)** to consider

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 **($X^{(j)}$, v)**

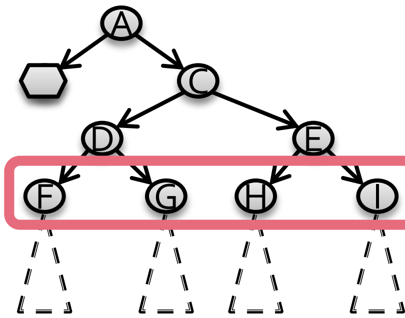
4) Master makes final decision where to split

PLANET Overview



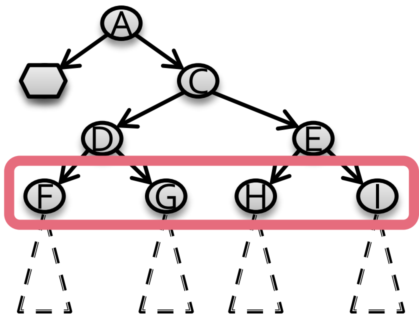
- **We build the tree level by level**
  - One MapReduce step builds **one level of the tree**
- **Mapper**
  - Considers a number of possible splits **($X^{(i)}$,v)** on its subset of the data
  - For each split it stores **partial statistics**
  - Partial split-statistics is sent to **Reducers**
- **Reducer**
  - Collects all partial statistics and determines best split
- **Master** grows the tree for one level

PLANET Overview



- **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

PLANET: 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

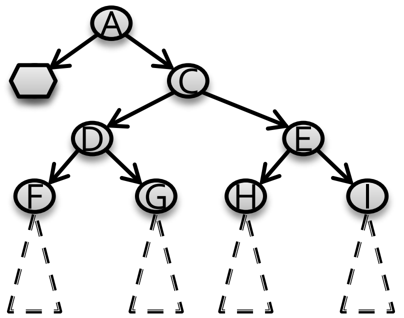
(1) Master Node

(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

PLANET: Components

(1) Master Node

(2) MapReduce Initialization (run once first)

(3) MapReduce FindBestSplit (run multiple times)

(4) MapReduce InMemoryBuild (run once last)

Initialization: Attribute metadata

- **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?

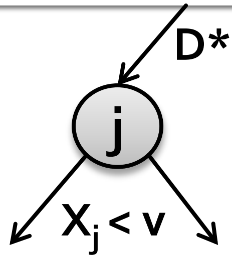
Initialization: Attribute metadata

- **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**

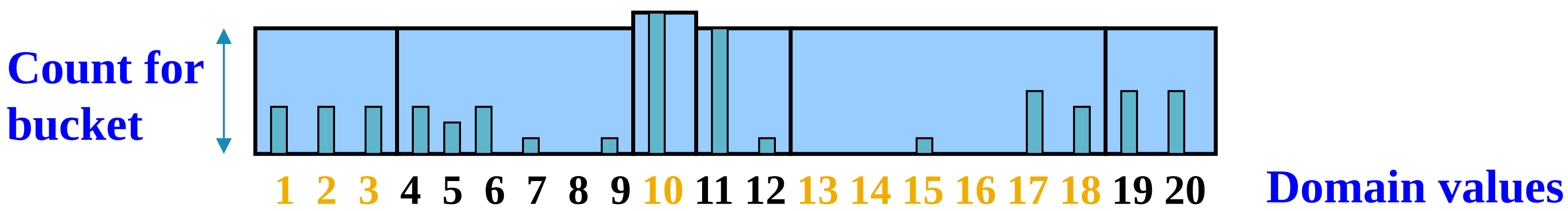
$X^{(j)}$: 1.2 1.3 1.4 1.6 2.1 7.2 8.1 9.8 10.1 10.2 10.3 10.4 11.5 11.7 12.8 12.9

- **Idea: Consider a limited number of splits such that splits "move" about the same amount of data**

Initialization: Attribute metadata

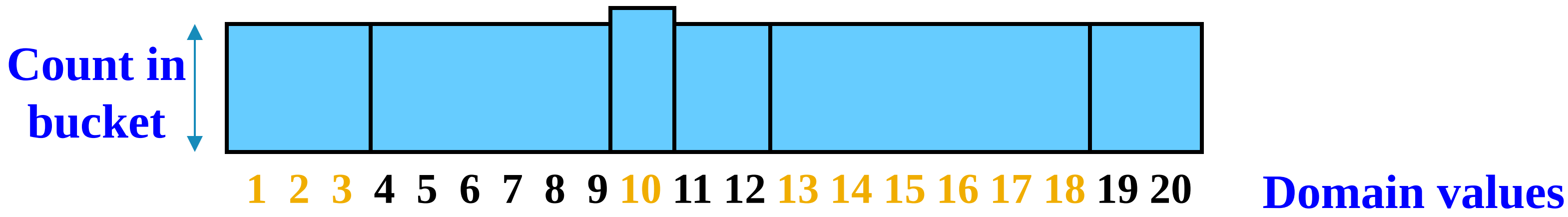


- **Splits for numerical attributes:**
  - **For attribute $X^{(j)}$ we would like to consider every possible value $v \in O_j$**
  - **Compute an approx. equi-depth histogram on *D****
    - **Idea:** Select buckets such that counts per bucket are equal



  - **Use boundary points of histogram as splits**

Side note: Computing Equi-Depth



- **Goal:** Equal number of elements per bucket (***B*** buckets total)
- Construct by first **sorting** and then taking **B-1** equally-spaced splits

1 2 2 3 4 7 8 9 10 10 10 10 11 11 12 12 14 16 16 18 19 20 20 20

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

PLANET: Components

(1) **Master Node**

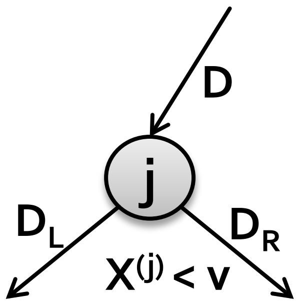
(2) **MapReduce <u>Initialization</u>** (run once first)

(3) **MapReduce <u>FindBestSplit</u>** (run multiple times)

(4) **MapReduce <u>InMemoryBuild</u>** (run once last)

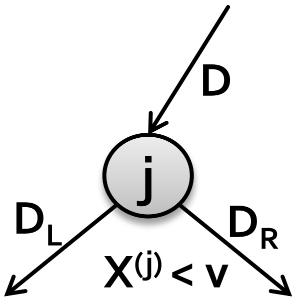
FindBestSplit

- **Goal:** For a particular split node ***j*** find attribute ***X^(j)*** and value ***v*** that **maximizes Purity**:
  - $|D| \cdot Var(D) - \left(|D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R)\right)$
    - **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 **$x_i^{(j)} \geq v$**
    - $Var(D) = \frac{1}{n}\sum_{i \in D} y_i^2 - \left(\frac{1}{n}\sum_{i \in D} y_i\right)^2$



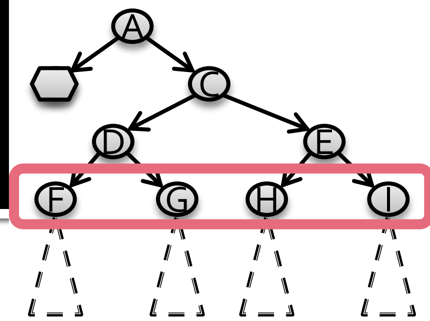
FindBestSplit

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



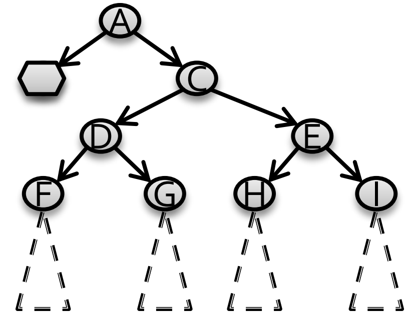
- **Important observation:** Variance can be computed from **sufficient statistics**:
  N, S=Σy$_i$, Q=Σy$_i^2$
  - Each **Mapper** processes subset of data **D$_m$**, and computes **N$_m$, S$_m$, Q$_m$** for its own **D$_m$**
  - **Reducer** combines the statistics and computes global variance and then Purity:
    - $Var(D) = \frac{1}{\sum_m N_m}\sum_m Q_m - \left(\frac{1}{\sum_m N_m}\sum_m S_m\right)^2$

FindBestSplit: Map



- **Mapper:**
  - Initialized by loading results of **Initialization task**
    - **Current model** (to find which node each datapoint $\mathbf{x}_i$ 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):**
      - **<NodeID, { S=Σy, Q=Σy², N=Σ1 } >**
    - **For each split store statistics and at the end emit:**
      - **<SplitID, { S, Q, N } >**
        - **SplitID = (node n, attribute $X^{(j)}$, split value *v*)**

FindBestSplit: Reducer

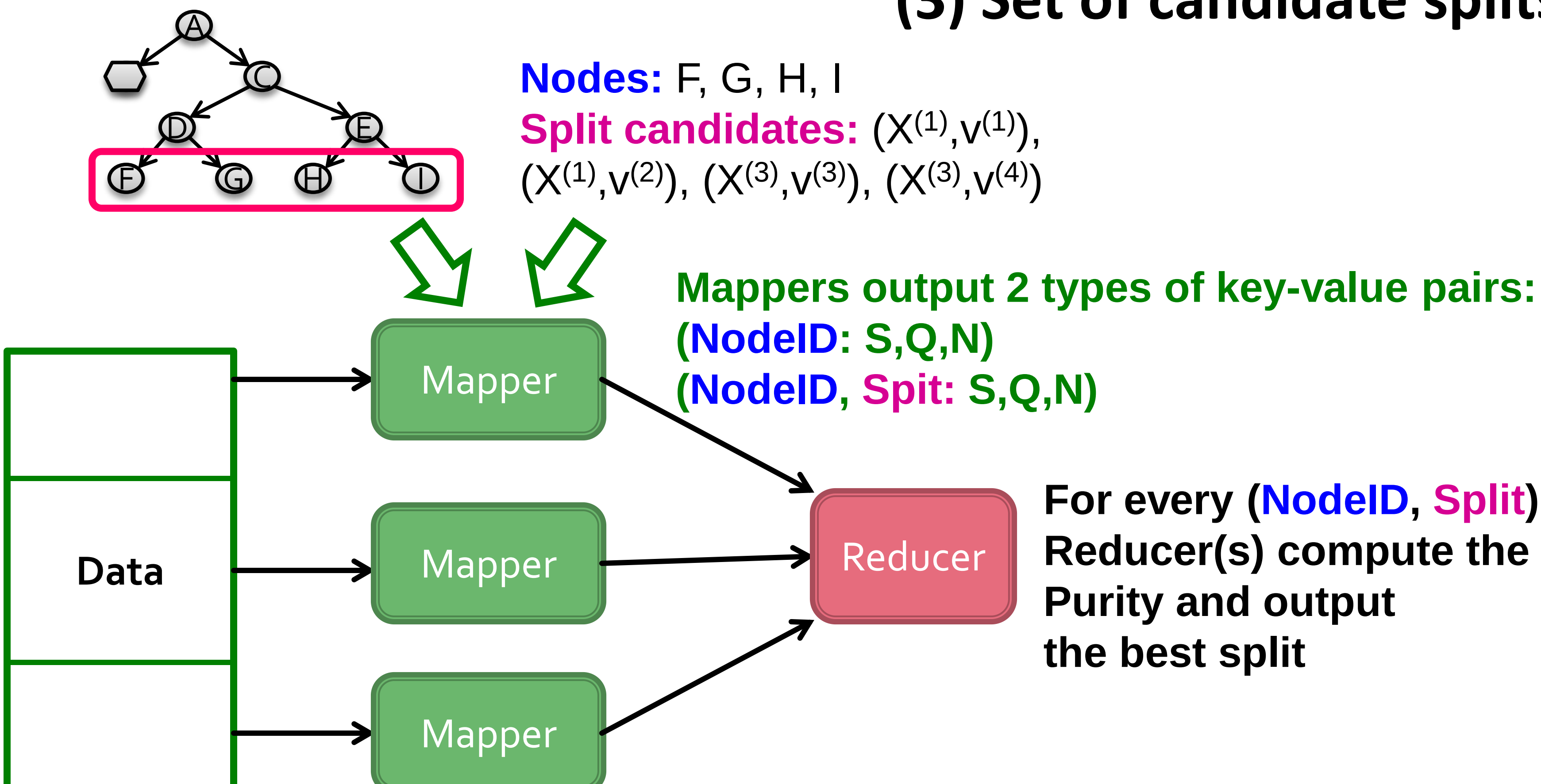


Reducer:

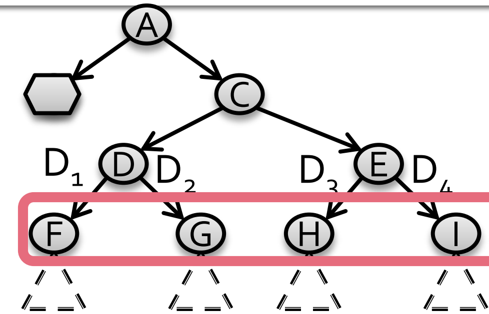
- **(1)** Load all the **<NodeID, <u>List</u> $\{S_m, Q_m, N_m\}$>** pairs and **aggregate** the per node statistics
- **(2)** For all the **<SplitID, <u>List</u> $\{S_m, Q_m, N_m\}$>** **aggregate** the statistics
  - $$Var(D) = \frac{1}{\sum_m N_m} \sum_m Q_m - \left(\frac{1}{\sum_m N_m} \sum_m S_m\right)^2$$
- **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

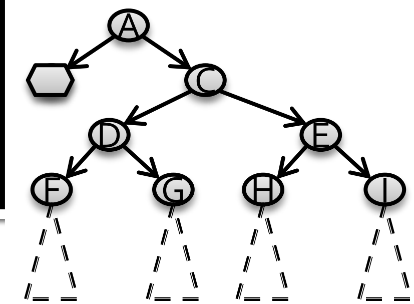


Overall system architecture



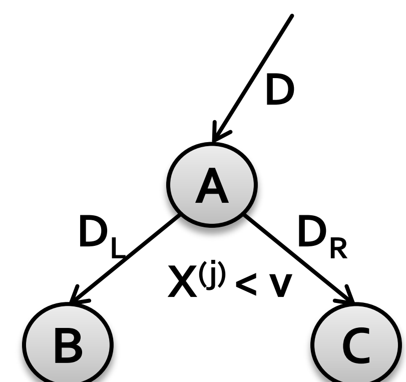
- **Example:** Need to split nodes **F, G, H, I**
- **Map and Reduce:**
  - **FindBestSplit::Map** (each mapper)
    - Load the current model **M**
    - Drop every example $\mathbf{x}_i$ down the tree
    - If it hits ***G*** or ***H***, update in-memory hash tables:
      - For each node: $\mathbf{T_n}$: **(Node)→{S, Q, N}**
      - For each **(Split, Node)**: $\mathbf{T_{n,j,s}}$: **(Node, Attribute, SplitValue)→{S, Q, N}**
    - **Map::Finalize**: output the key-value pairs from above hashtables
  - **FindBestSplit::Reduce** (each reducer)
    - Collect:
      - **T1:**<Node, List{S, Q, N} > → <Node, {Σ S, Σ Q, Σ N} >
      - **T2:**<(Node, Attr. Split), List{S, Q, N}> → <(Node, Attr. Split), {ΣS, ΣQ, ΣN}>
    - Compute impurity for each node using **T1, T2**
    - Return **best split** to Master (which then decides on globally best split)

Back to the Master



- **Collects outputs from FindBestSplit reducers**
  <Split.NodeID, Attribute, Value, Impurity>

- **For each node decides the best split**
  - If data in $\mathbf{D_L/D_R}$ is small enough, later run a MapReduce job **InMemoryBuild** on the node
  - Else run MapReduce **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!**

Learning Ensembles

- **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

Bagged Decision Trees

- **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

- B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. ***PLANET: Massively parallel learning of tree ensembles with MapReduce*****.** In Proc. *VLDB* 2009.
- J. Ye, J.-H. Chow, J. Chen, Z. Zheng. ***Stochastic Gradient Boosted Distributed Decision Trees***. In Proc. *CIKM* 2009.