10-605/10-805: Machine Learning with Large Datasets

FALL 2022

Distributed Trees

Key course topics

Data preparation
- Data cleaning
- Data summarization
- Visualization
- Dimensionality reduction

Training
- **Distributed ML**
- Large-scale optimization
- Scalable deep learning
- Efficient data structures
- Hyperparameter tuning

Inference
- Hardware for ML
- Techniques for low-latency inference (compression, pruning, distillation)

Infrastructure / Frameworks
- Apache Spark
- TensorFlow
- AWS / Google Cloud / Azure

Advanced topics
- Federated learning
- Neural architecture search
- Productionizing ML

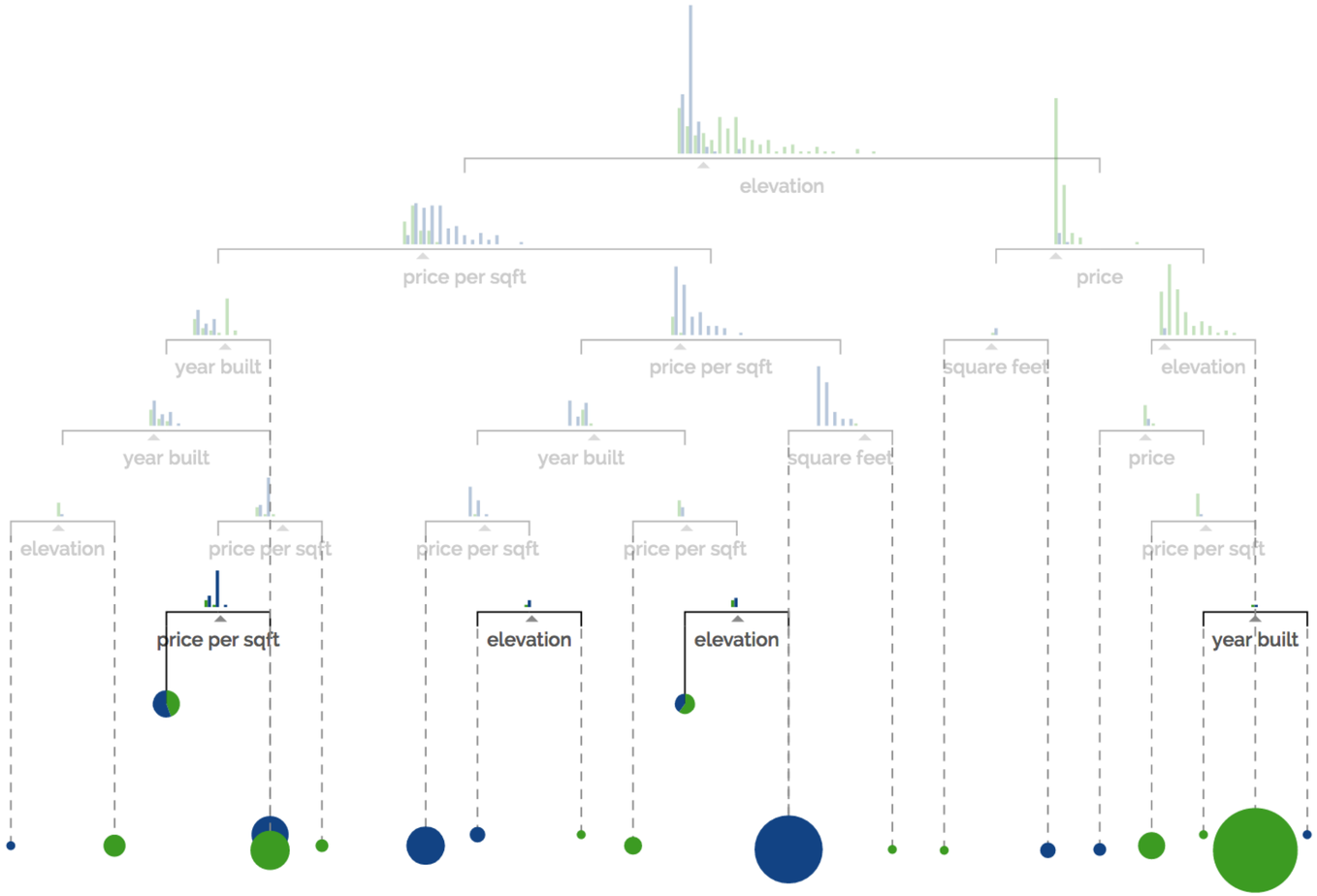
Outline

1. **Review of Decision Trees**
2. Distributed Decision Trees — Row Partitioning
3. Distributed Decision Trees — Column Partitioning

Decision Tree Review - Training Time

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

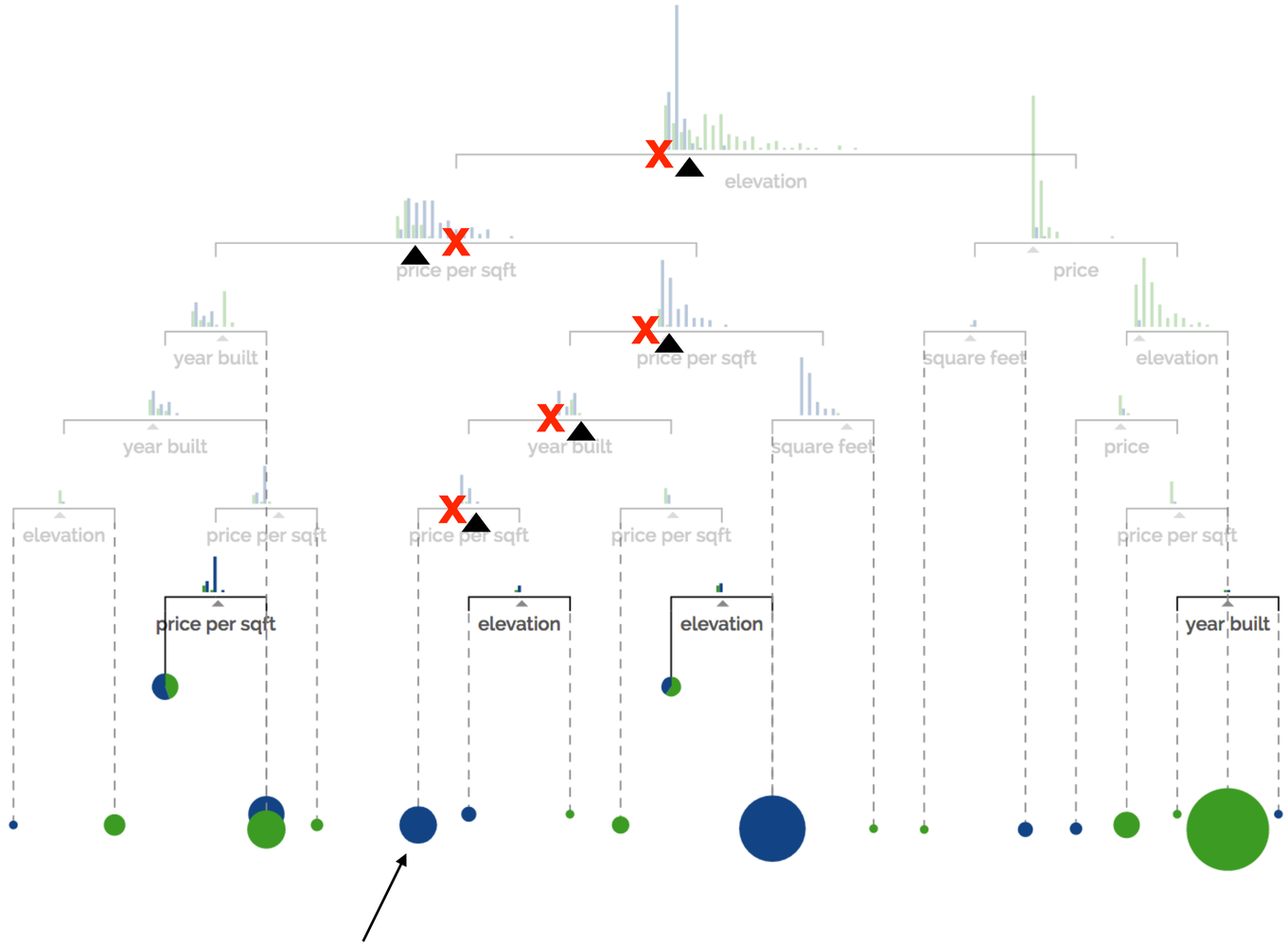
- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms



Decision Tree Review - Test Time

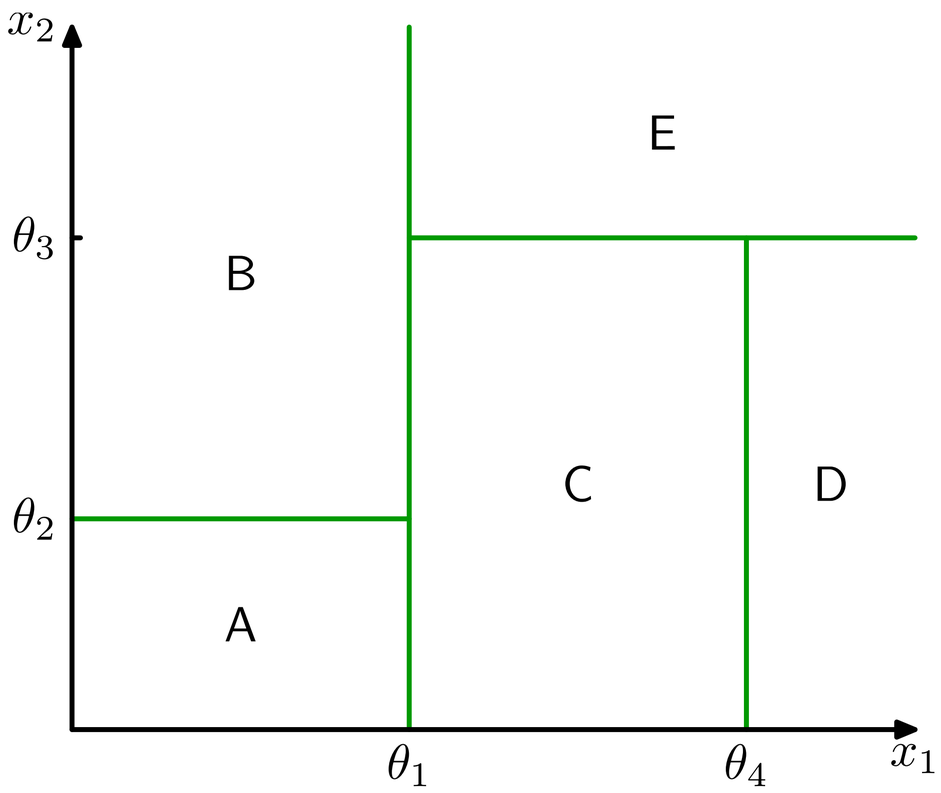
Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

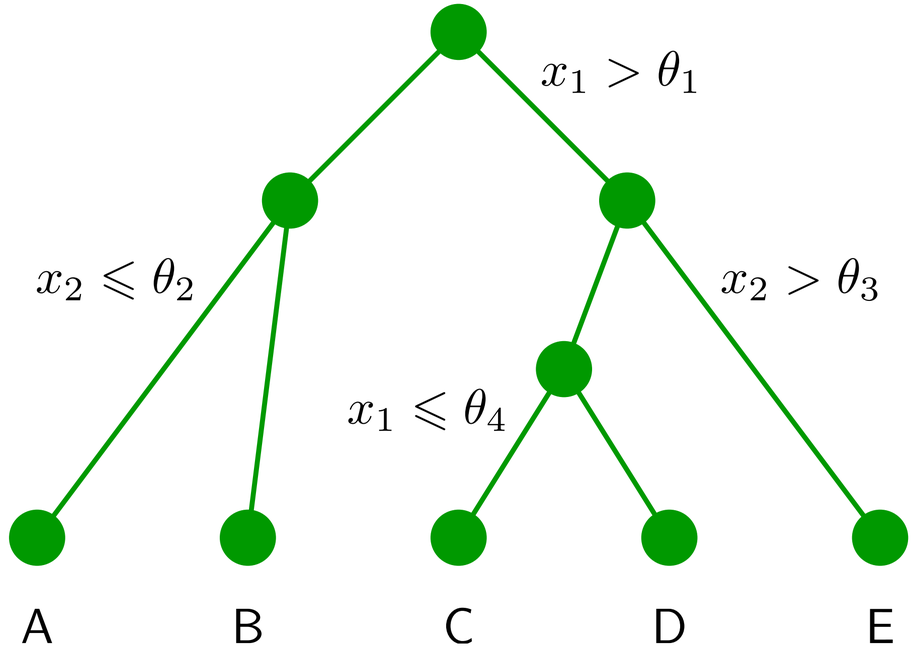
- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms



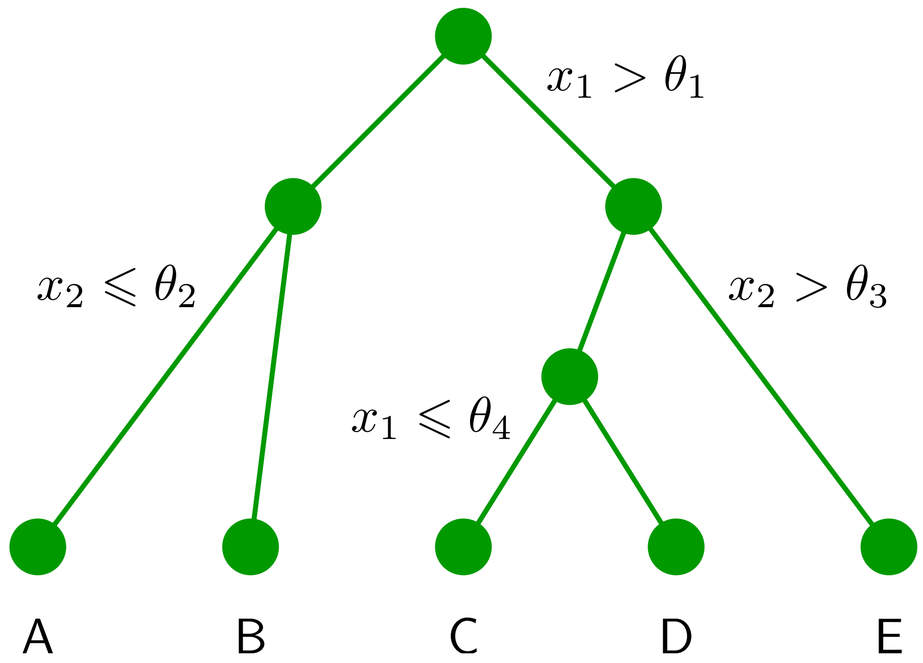
Predict *New York*

A tree partitions the feature space





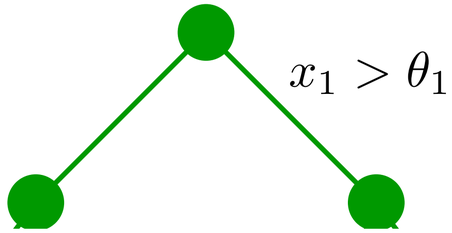
Learning a tree model



Three things to learn:

1. The structure of the tree.
2. The threshold values (θ_i).
3. The values for the leafs $(A, B, \ldots)$.

Goal: Learn threshold or “split" to maximize “purity” of child nodes



Common measures of impurity

- Gini and Entropy (classification)
- Variance (regression)

Learning strategy: brute force search over space of split candidates (i.e., what feature to split on and with what threshold value)

$$Split(i) = \arg\max_{s \in \mathcal{S}}$$

split candidate maximizing node purity at node i

Set of possible split values

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute our impurity metric of interest

Computational Considerations

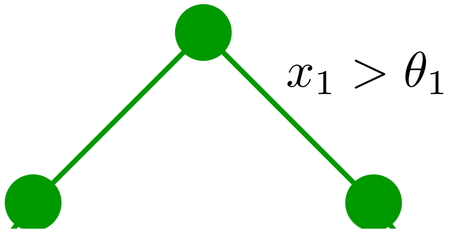
Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute our impurity metric of interest

Categorical Features

- Assuming q distinct categories, there are $2^{q-1} - 1$ possible partitions
- Things simplify in the case of binary classification or regression,
  - suffices to consider only $q - 1$ possible splits (see Section 9.2.4 in ESL)

How to compute purity for a given split candidate?



Common measures of impurity
- Gini and Entropy (classification)
- Variance (regression)

Goal: compute purity for given split candidate s
- Split candidate partitions all training points in a node into two groups
- We want to compute purity of *labels* in each group

A useful definition of variance: $E[\mathbf{x}^2] - E[\mathbf{x}]^2$
- Can separately compute sufficient statistics for each $\mathbf{x}$ (i.e., $\mathbf{x}$ and $\mathbf{x}^2$) and then aggregate these statistics

Putting it together…

$$Split(i) =$$

split candidate maximizing
node purity at node i

E.g., computing variance

- Sufficient statistics for each point $\mathbf{x}$ are $\mathbf{x}$ and $\mathbf{x^2}$
- Compute variance for each child node induced by s via $E[\mathbf{x}^2] - E[\mathbf{x}]^2$

Putting it together…

$$Split(i) = \arg\max_{s \in \mathcal{S}}$$

split candidate maximizing node purity at node i

Set of possible split values

E.g., computing variance

- Sufficient statistics for each point $\mathbf{x}$ are $\mathbf{x}$ and $\mathbf{x}^2$
- Compute variance for each child node induced by s via $E[\mathbf{x}^2] - E[\mathbf{x}]^2$

Putting it together…

computes c sufficient statistics for each point $\mathbf{x}$

$$Split(i) = \arg\max_{s \in \mathcal{S}} \left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s) \right)$$

split candidate maximizing node purity at node i

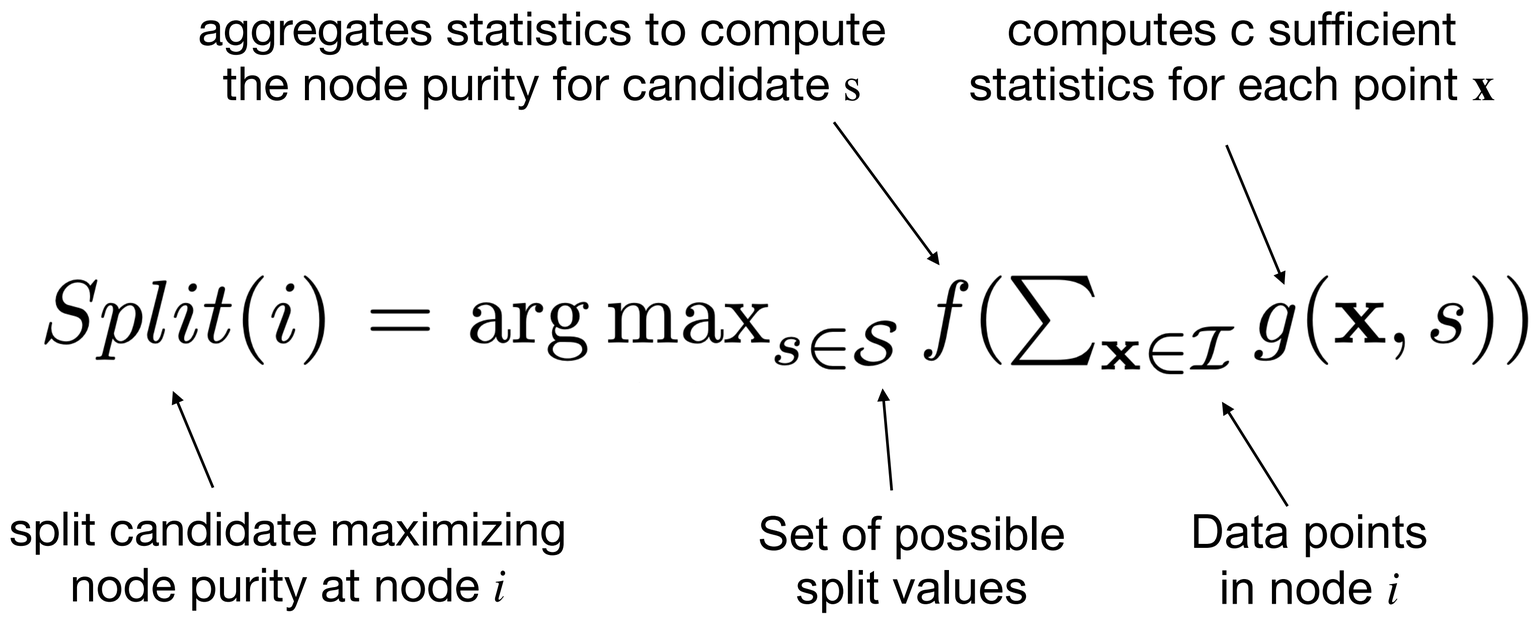
Set of possible split values

Data points in node i

E.g., computing variance

- Sufficient statistics for each point $\mathbf{x}$ are $\mathbf{x}$ and $\mathbf{x}^2$
- Compute variance for each child node induced by s via $E[\mathbf{x}^2] - E[\mathbf{x}]^2$

Putting it together…



E.g., computing variance

- Sufficient statistics for each point $\mathbf{x}$ are $\mathbf{x}$ and $\mathbf{x}^2$
- Compute variance for each child node induced by s via $E[\mathbf{x}^2] - E[\mathbf{x}]^2$

Outline

1. Review of Decision Trees
2. **Distributed Decision Trees — Row Partitioning**
3. Distributed Decision Trees — Column Partitioning

aggregates statistics to compute the node purity for candidate s

computes c sufficient statistics for each point **x**

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

split candidate maximizing node purity at node i

Set of possible split values

Data points in node i

aggregates statistics to compute the node purity for candidate s

computes c sufficient statistics for each point **x**

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

split candidate maximizing node purity at node i

Set of possible split values

Data points in node i

# of worker nodes

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{j=1}^{m} g_j(s)\right)$$

aggregates statistics to compute the node purity for candidate s

computes c sufficient statistics for each point **x**

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

split candidate maximizing node purity at node i

Set of possible split values

Data points in node i

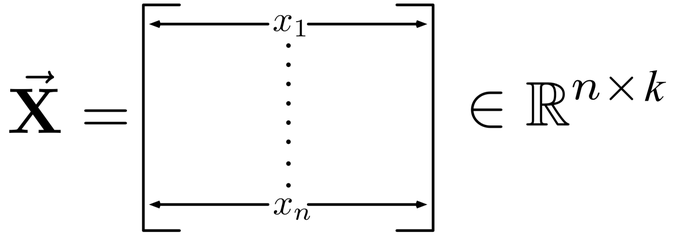
# of worker nodes

Data points in worker j

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{j=1}^{m} g_j(s)\right) \quad \text{where} \quad g_j(s) = \sum_{x \in \mathcal{I} \cap \mathcal{J}} g(\mathbf{x}, s)$$

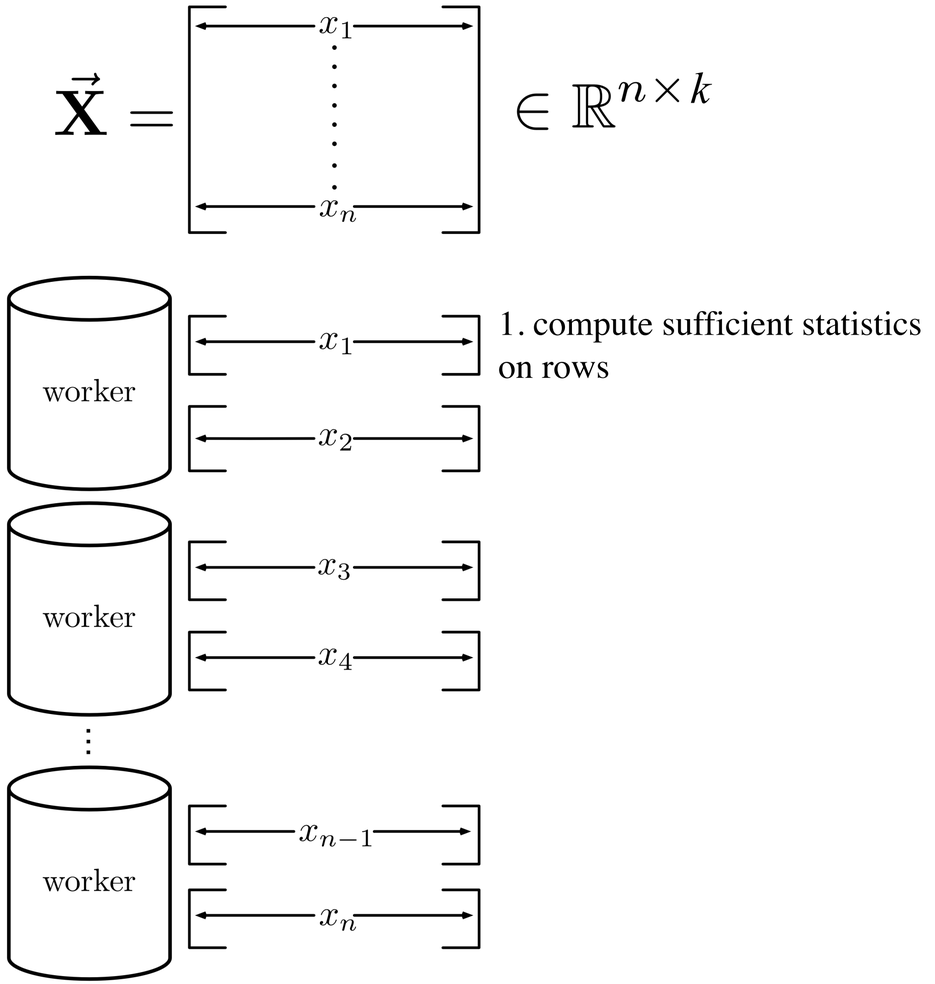
PLANET: “Row Partitioned” Decision Trees

- Partition training set ***by row***



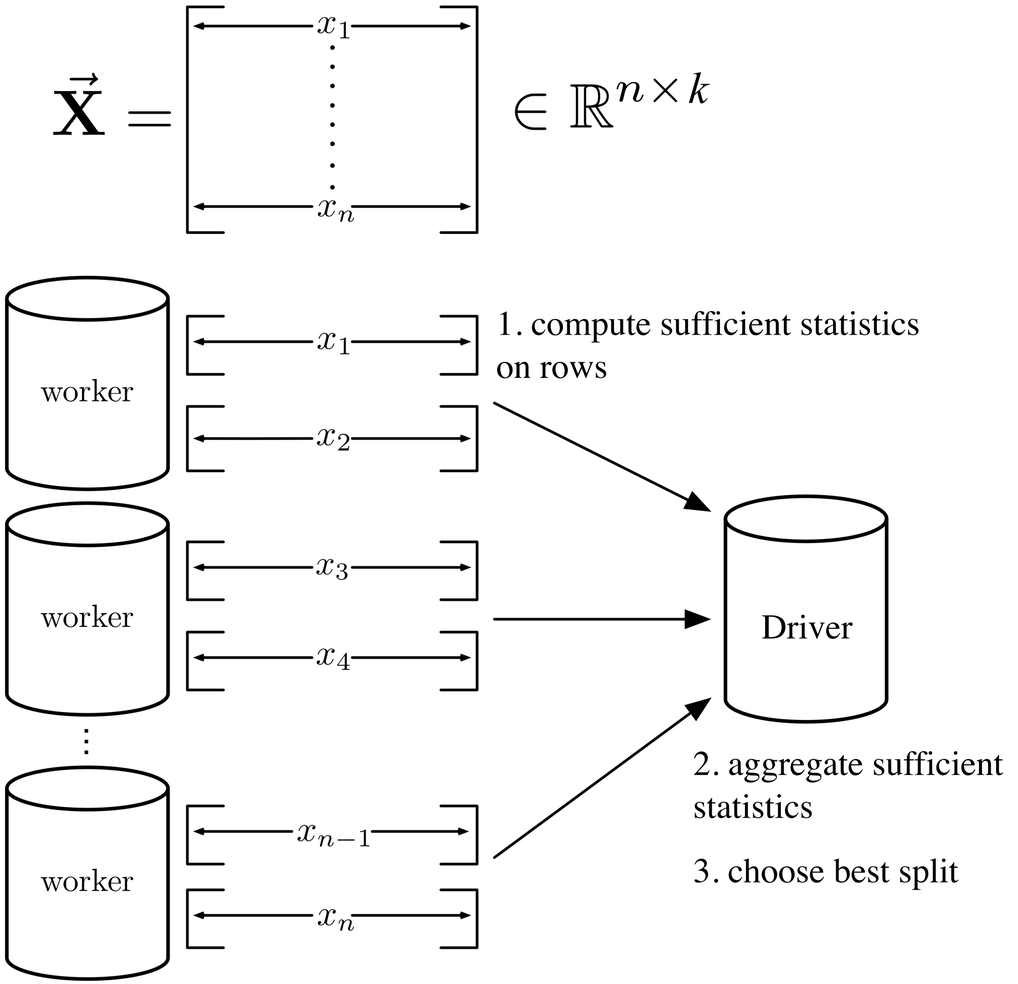
PLANET: “Row Partitioned” Decision Trees

- Partition training set ***by row***
- Workers: compute sufficient statistics on subsets of rows



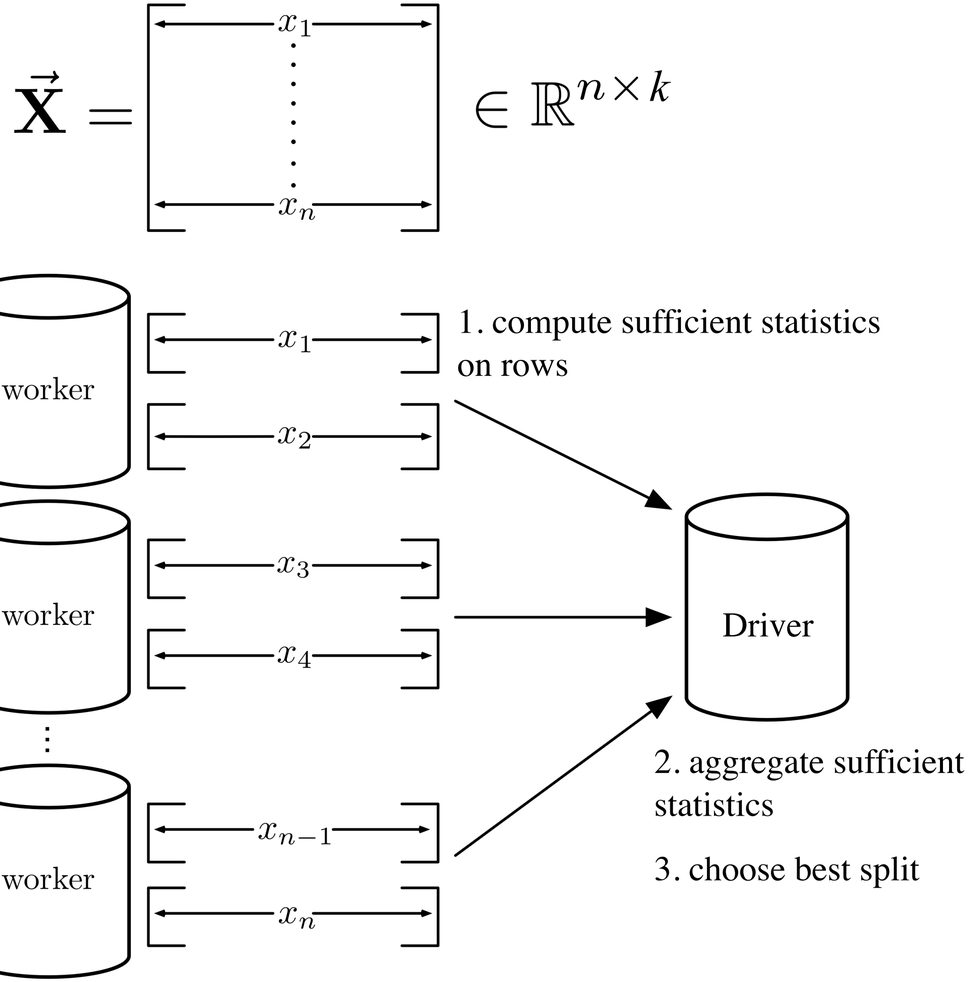
PLANET: “Row Partitioned” Decision Trees

- Partition training set ***by row***
- Workers: compute sufficient statistics on subsets of rows
- Driver:
  - aggregates sufficient statistics
  - picks best split



PLANET: “Row Partitioned” Decision Trees

- Partition training set ***by row***
- Workers: compute sufficient statistics on subsets of rows
- Driver:
  - aggregates sufficient statistics
  - picks best split
  - broadcasts result to all workers (who update local state to keep track of which datapoints are assigned to which child nodes)

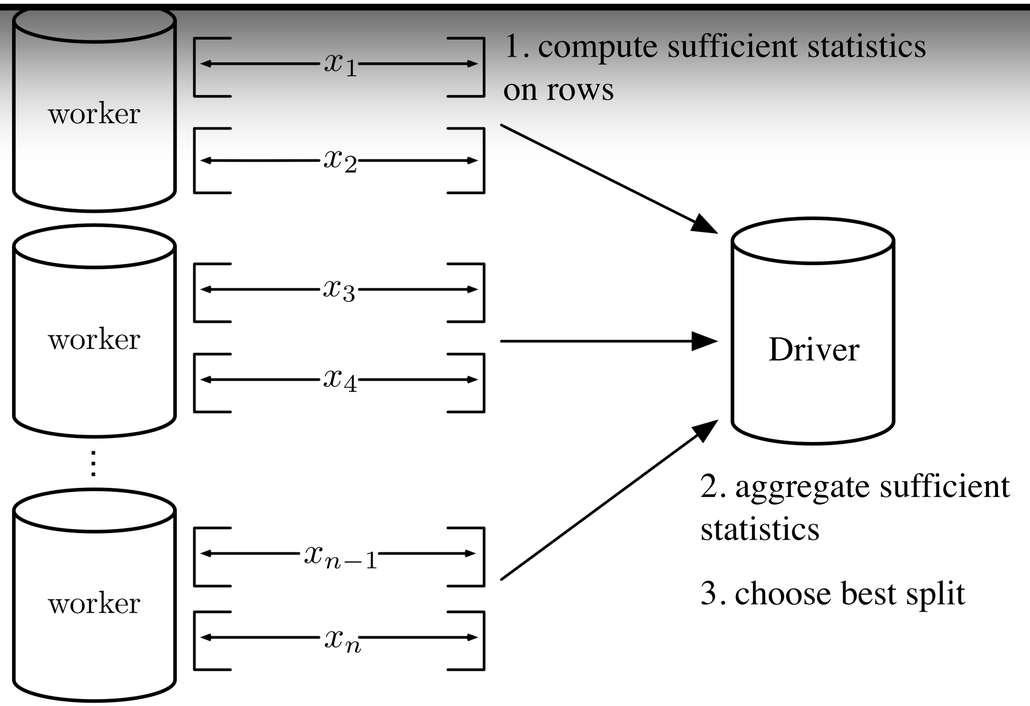


# of worker nodes

Data points in worker j

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{j=1}^{m} g_j(s)\right) \quad \text{where} \quad g_j(s) = \sum_{x \in \mathcal{I} \cap \mathcal{J}} g(\mathbf{x}, s)$$

- Driver:
  - aggregates sufficient statistics
  - picks best split
  - **broadcasts result to all workers (who update local state to keep track of which datapoints are assigned to which child nodes)**



Planet costs?

Notation

- n observations with k features
- B is number of split candidates for each feature
- D is depth of tree
- m workers

Distributed Computation: linear in n, k, and D

Communication

- For each node, workers communicate m B k tuples total
- Given 2^D total nodes, total communication is **$O(2^D$ m B k)**

Downsides of row-partitioning?

1. High communication costs, esp. for deep trees & many features
   - Exponential in the depth of the tree
   - Linear in the number of features, thresholds (B)

2. To reduce communication, small number of thresholds are considered, i.e., $B \ll n$
   - User specifies # thresholds
   - Provably suboptimal splits are selected

Why Use *Deep* Decision Trees?

- Datasets are growing both in size *and* dimension
  - More rows *and* more columns
- To model high-dimensional data, we need to consider more splits
  - **More splits ⇒ deeper trees**

PROBLEM: **PARTITIONING BY ROW**

⇒ INEFFICIENT FOR DEEP TREES AND MANY FEATURES

⇒ TRADEOFF BETWEEN ACCURACY AND EFFICIENCY

Outline

1. Review of Decision Trees
2. Distributed Decision Trees — Row Partitioning
3. **Distributed Decision Trees — Column Partitioning**

ALTERNATIVE APPROACH: PARTITION BY “*COLUMN*”

aggregates statistics to compute the node purity for candidate s

computes c sufficient statistics for each point **x**

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

split candidate maximizing node purity at node i

Set of possible split values

Data points in node i

aggregates statistics to compute the node purity for candidate s

computes c sufficient statistics for each point **x**

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

split candidate maximizing node purity at node i

Set of possible split values

Data points in node i

$$Split(i) = \text{Compare}\left(f_j\right)$$

Best split among features stored in worker j

aggregates statistics to compute the node purity for candidate s

computes c sufficient statistics for each point **x**

$$Split(i) = \arg\max_{s \in \mathcal{S}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

split candidate maximizing node purity at node i

Set of possible split values

Data points in node i

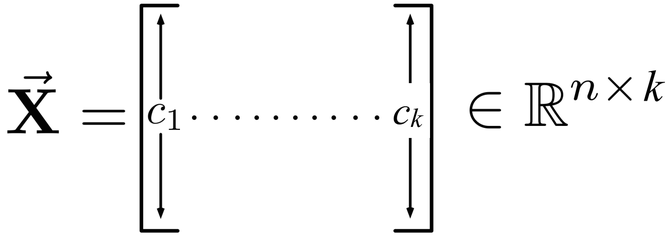
$$Split(i) = \text{Compare}\left(f_j\right) \text{ where } \quad f_j = \arg\max_{s \in \mathcal{J}} f\left(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s)\right)$$

Best split among features stored in worker j

Set of possible split values corresponding to features stored in worker j

Yggdrasil: Column-Partitioning

- Workers compute sufficient statistics on ***<u>all local features</u>***

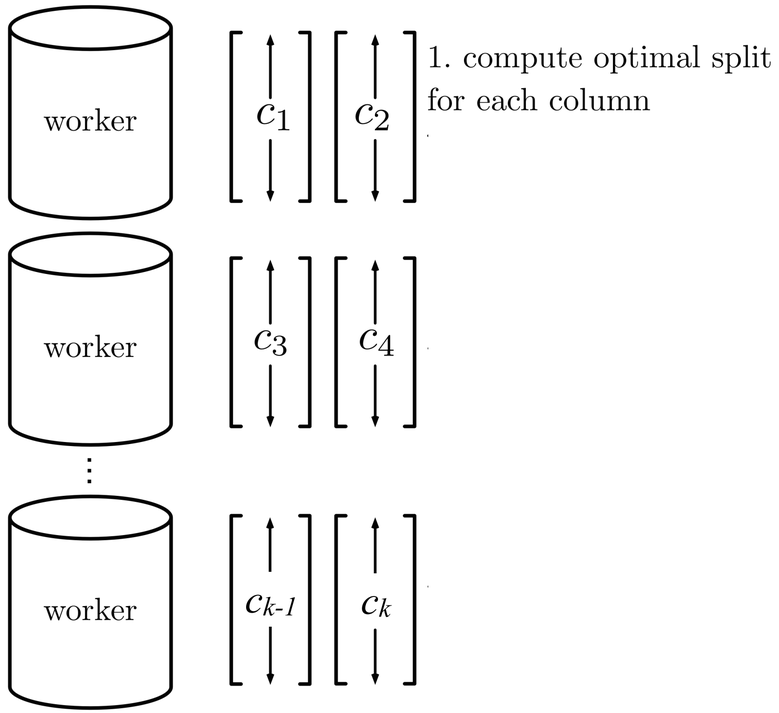


Yggdrasil: Column-Partitioning

- Workers compute sufficient statistics on ***all local features***

- No approximation; all thresholds considered

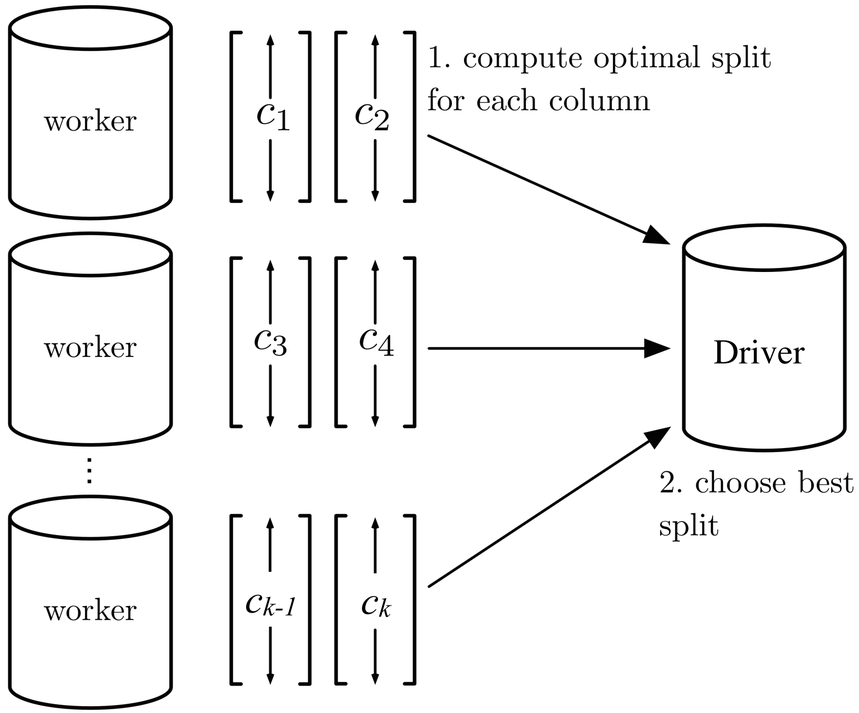
$$\vec{\mathbf{X}} = \left[c_1 \cdots\cdots\cdots c_k \right] \in \mathbb{R}^{n \times k}$$



Yggdrasil: Column-Partitioning

- Workers compute sufficient statistics on ***all local features***
- No approximation; all thresholds considered
- Driver picks best global split

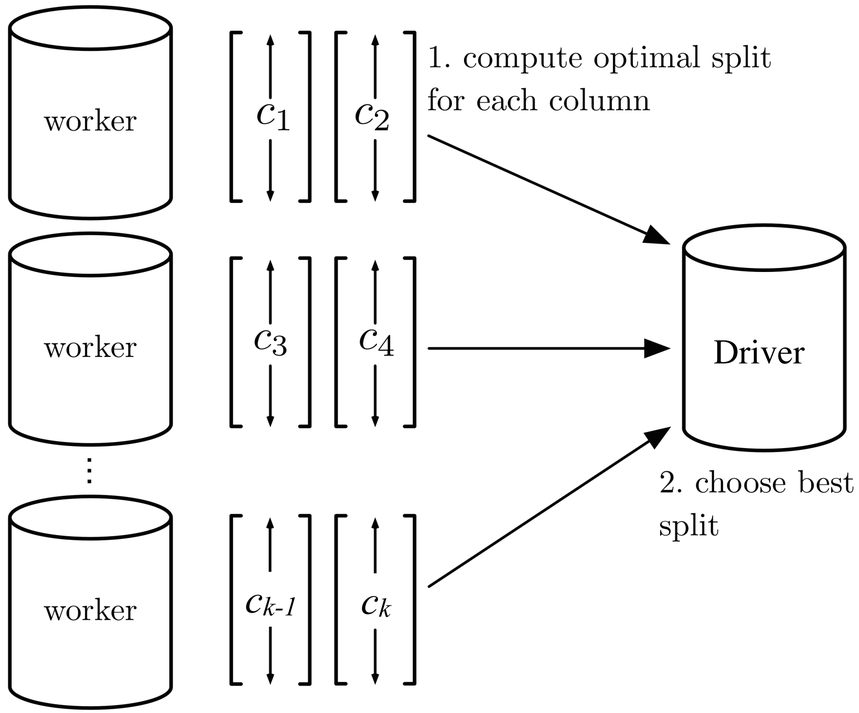
$$\vec{\mathbf{X}} = \left[c_1 \cdots\cdots\cdots c_k \right] \in \mathbb{R}^{n \times k}$$



Yggdrasil: Column-Partitioning

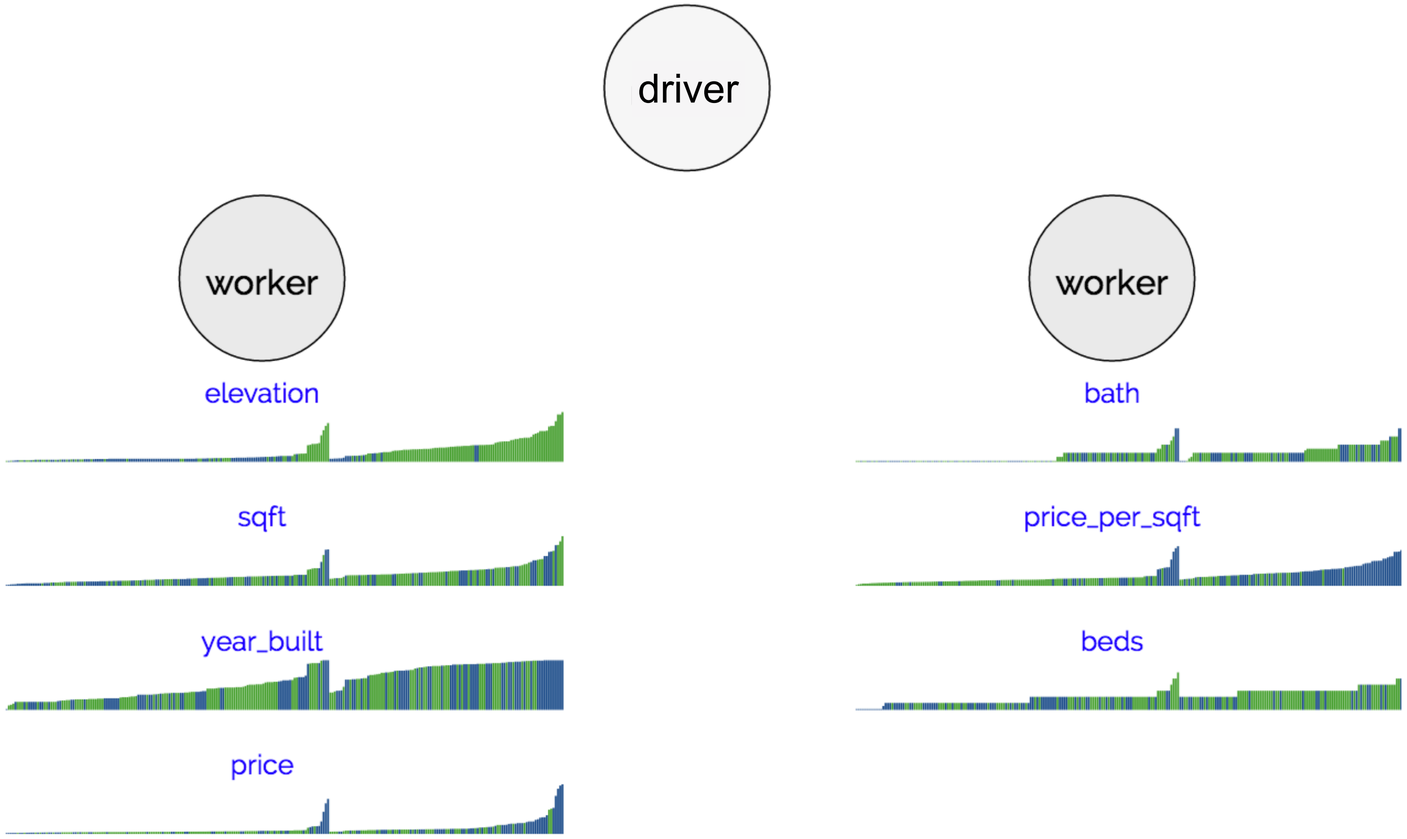
- Workers compute sufficient statistics on ***all local features***
- No approximation; all thresholds considered
- Driver picks best global split
- One round of communication to update worker state at each tree level
  - Driver asks relevant workers about how datapoints are split, then shares results with all workers

$$\vec{\mathbf{X}} = \left[c_1 \cdots\cdots\cdots c_k \right] \in \mathbb{R}^{n \times k}$$



Yggdrasil in Action

1. Partition features across workers
2. Workers sort each feature by value
3. Compute best split for each feature
4. Pick best split for each worker & send to driver
5. Driver selects best global split among the candidates, requests bit vector from worker
6. Driver broadcasts bit vector for best global split to all workers
7. Workers re-partition their features into sorted sub-arrays



Yggdrasil costs?

Notation

- n observations, k features, m workers
- B is number of split candidates for each feature (upper bounded by n)
- D is depth of tree (therefore roughly 2^D total tree nodes)

Computation: linear in n, k, and D, and is trivially parallelizable [same as PLANET]

Communication [much different than PLANET]

- workers communicate m tuples total per node, so **$O(2^D\ m)$** in total
- At each tree level, **O(n)** to compile and share results of splits to each worker
- Total communication is **$O(2^D\ m + D\ n\ m)$** [vs **$O(2^D\ B\ k\ m)$** for PLANET]
  - No dependence on k; for large n, **O(D n m)** dominates

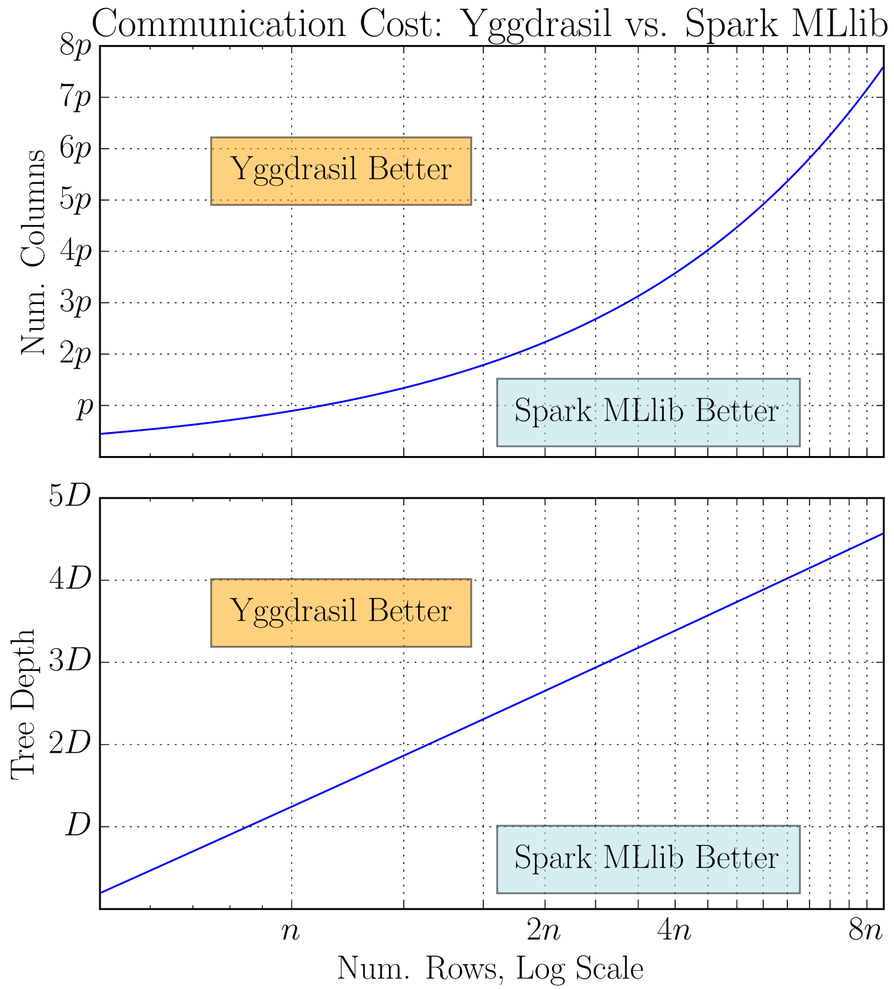
No Single Winner

Planet

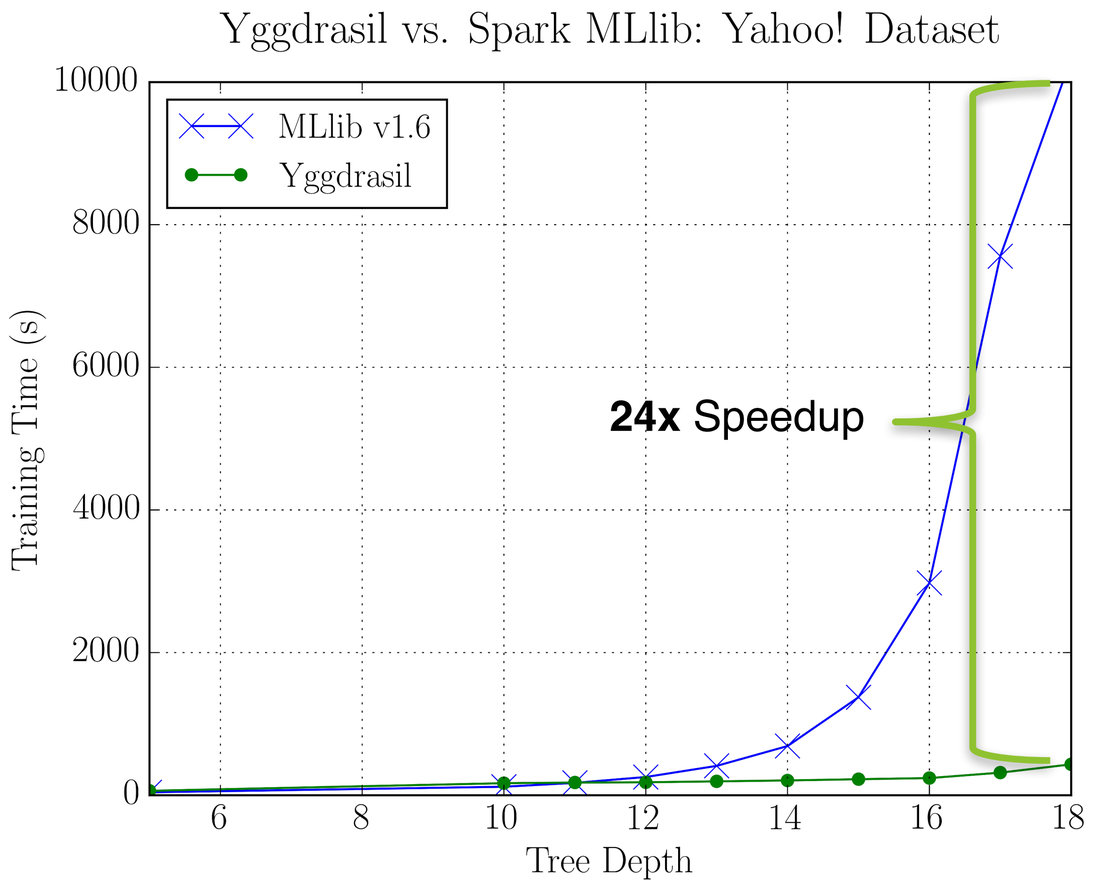
- Default in Spark MLlib
- **O(2^D B k m)** communication
- Better for small k / n and small D / n
- (also, row partitioning is the norm)

Yggdrasil

- **O(2^D m + D n m)** communication
- Better for large k / n and large D / n

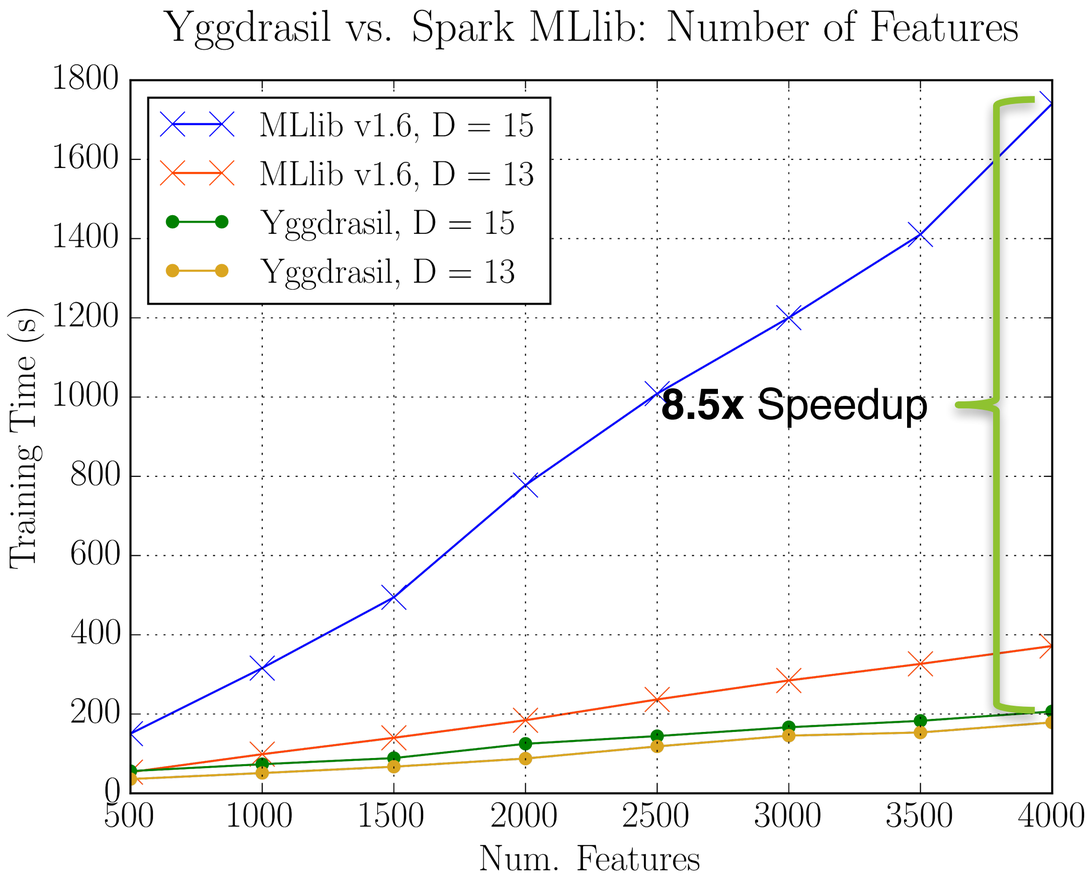


Results



- Regression task
- 2 million rows
- 3500 columns
- 52.2 GB (< 1% zeros)

Results



- 2 million rows
- Yggdrasil empirically outperforms Spark MLlib for deep trees and many features

Takeaways

- Understand cost of training decision trees
- Know how to distribute training of row partitioned vs. column partitioned trees
- Understand advantages of row vs. column partitioning

References

- Planet: Massively parallel learning of tree ensembles with mapreduce. B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. VLDB, 2009.
- Yggdrasil: An Optimized System for Training Deep Decision Trees at Scale. F. Abuzaid, J. Bradley, F. Liang, A. Feng, L. Yang, M. Zaharia, A. Talwalkar NeurIPS 2016