

Yggdrasil: Faster Decision Trees Using Column Partitioning in Spark

Firas Abuzaid, Joseph Bradley, Feynman Liang,
Andrew Feng, Lee Yang,
Matei Zaharia, Ameet Talwalkar

MIT, UCLA, Databricks, Yahoo!

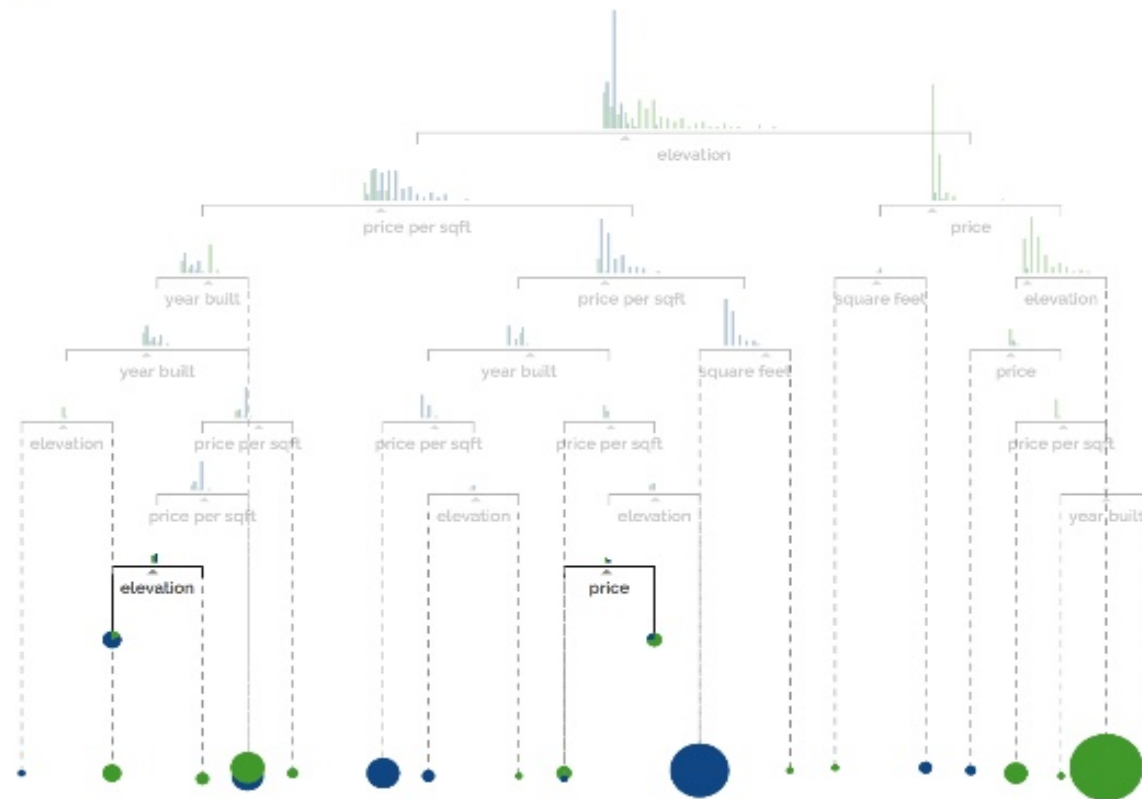


SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

A (Brief) Decision Tree Tutorial

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



Why Use Decision Trees?

- Arbitrarily expressive, but easy to interpret
- Simple to tune
- Natural support for different feature types
- Can easily extend to ensembles and boosting techniques



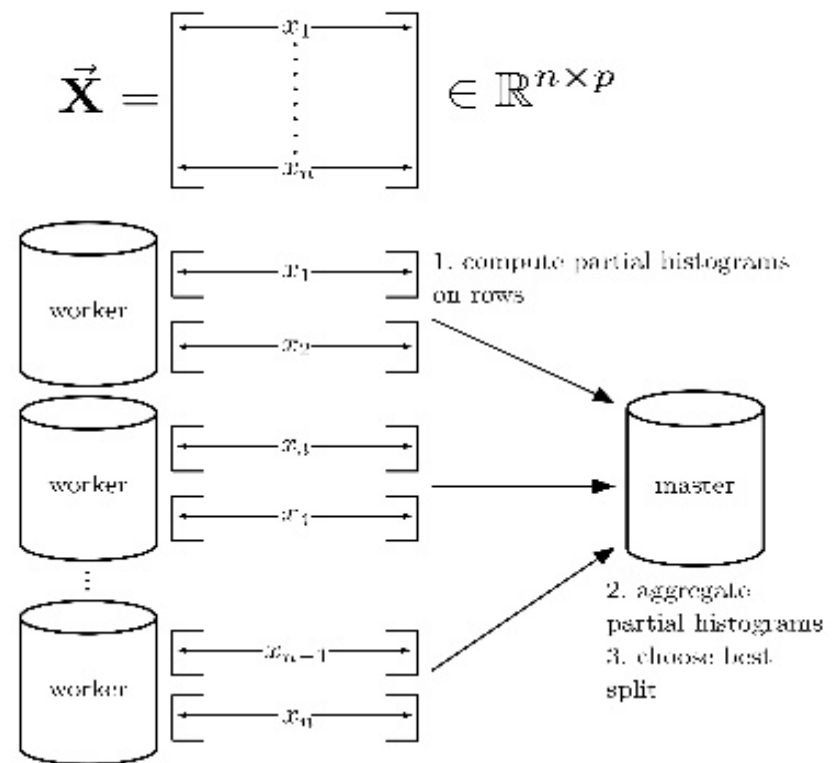
Why Use Deep Decision Trees?

- Today's 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 \Rightarrow deeper trees**



Decision Trees in Spark

- Partition training set by row
- Histograms used to compute splits
- Workers compute partial histograms on subsets of rows
- Master aggregates partial histograms, picks best feature to split on



What's wrong with 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
2. To reduce communication, small number of thresholds are considered
 - Approximate split-finding using histograms
 - User specifies # thresholds
 - **NB:** Optimal split may not always be found



PROBLEM:

PARTITIONING BY ROW

**⇒ INEFFICIENT FOR DEEP TREES
AND MANY FEATURES**



PROBLEM:
PARTITIONING BY ROW
⇒ TRADEOFF BETWEEN
ACCURACY AND EFFICIENCY



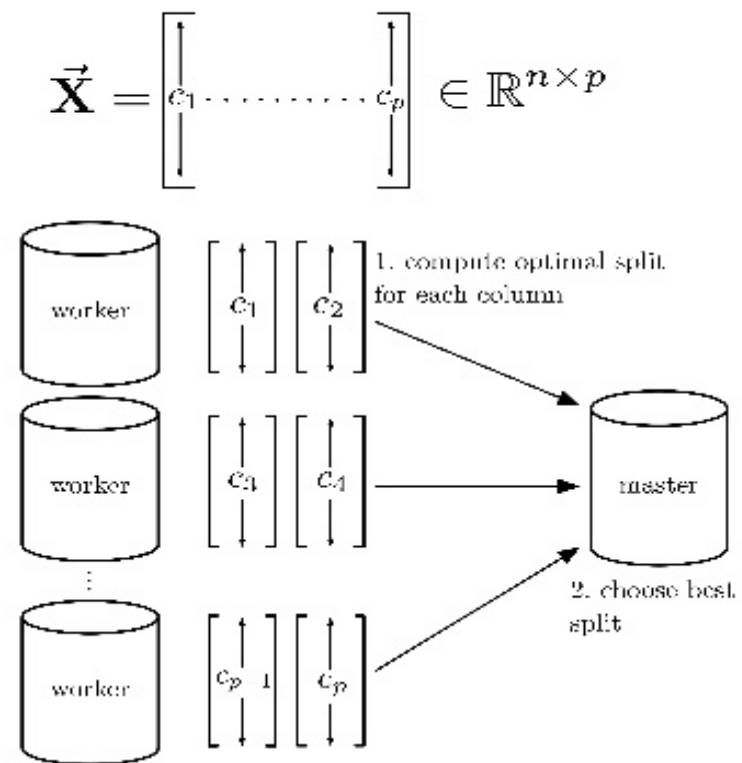
SOLUTION: PARTITION BY COLUMN



SPARK SUMMIT 2016

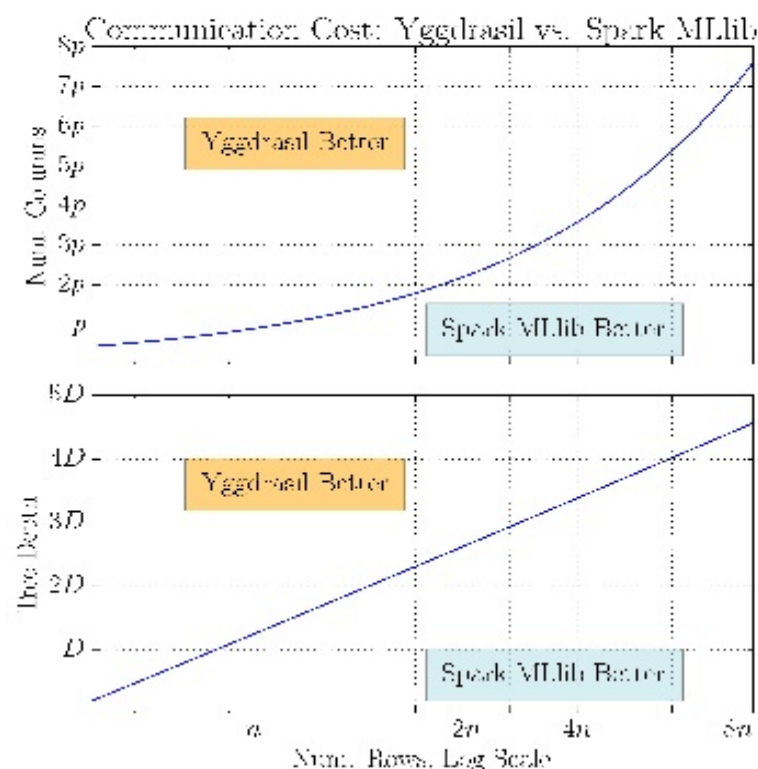
Yggdrasil: The New Approach

- Workers compute sufficient stats on entire columns
- Master has one job: pick best global split
- No approximation; all thresholds considered



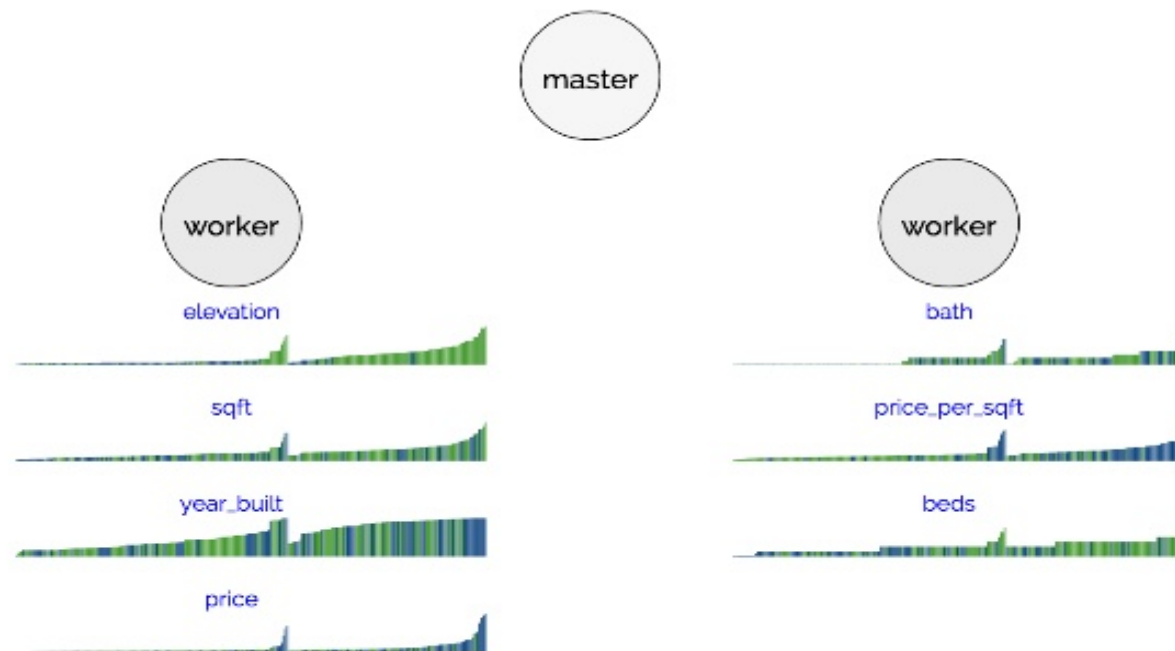
Yggdrasil: The New Approach

Communication cost much lower for deep trees & many features



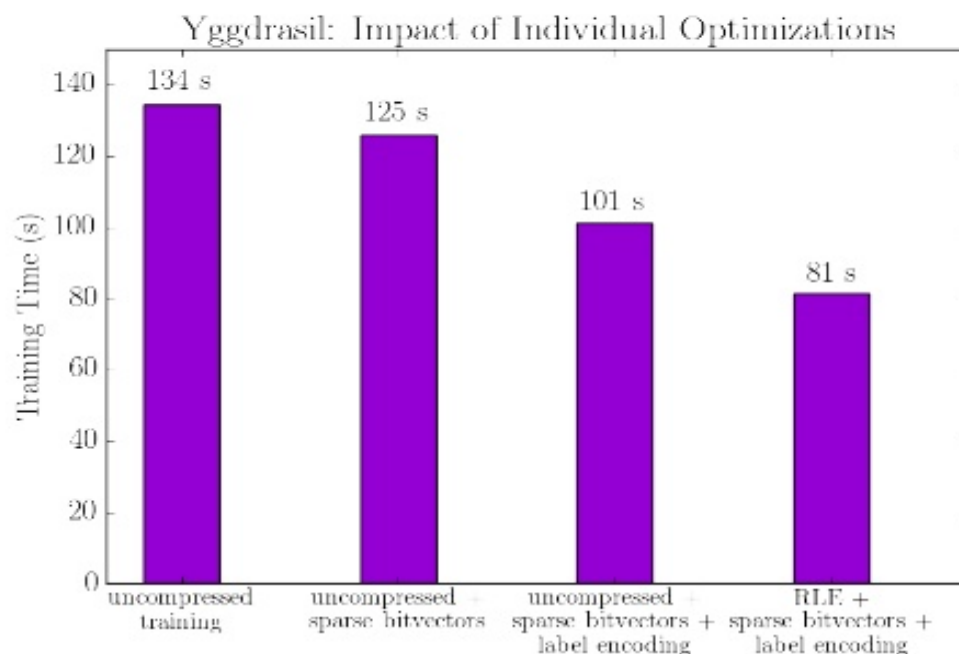
Yggdrasil in Action

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 master
5. Master selects best global split among the candidates, requests bit vector from worker
6. Master broadcasts bit vector for best global split to all workers
7. Workers re-partition their features into sorted sub-arrays

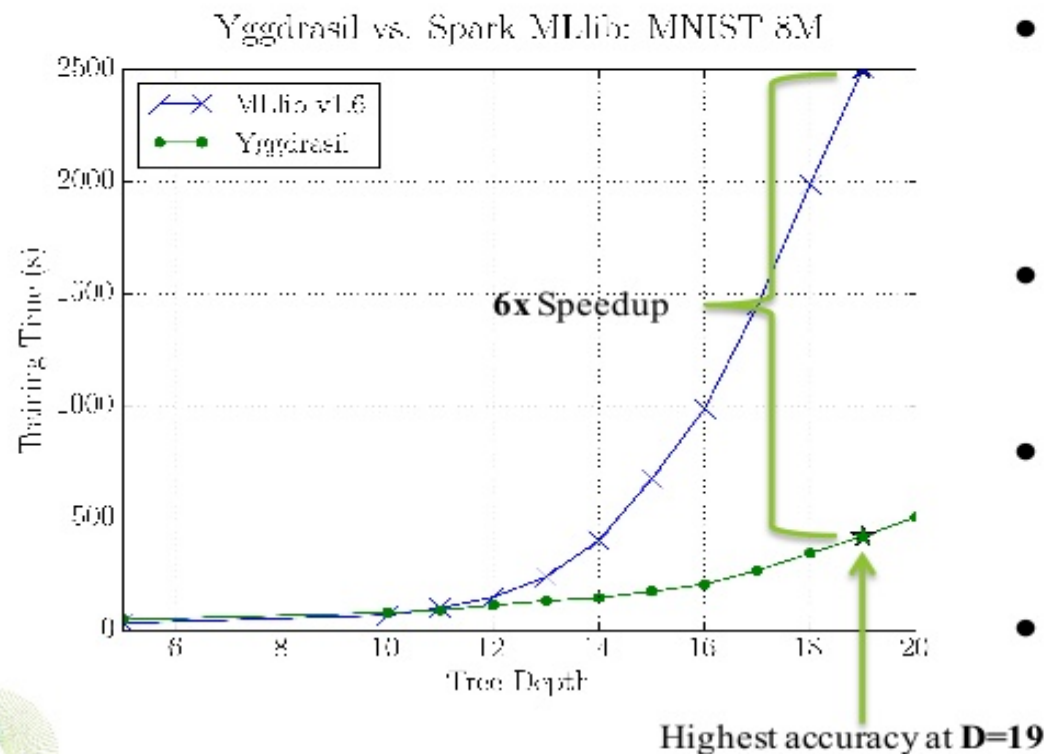


Optimizations in Yggdrasil

- Columnar compression using RLE
 - Train directly on columns without decompressing
- Label encoding for fewer cache misses
- Sparse bit vectors to reduce communication overheads



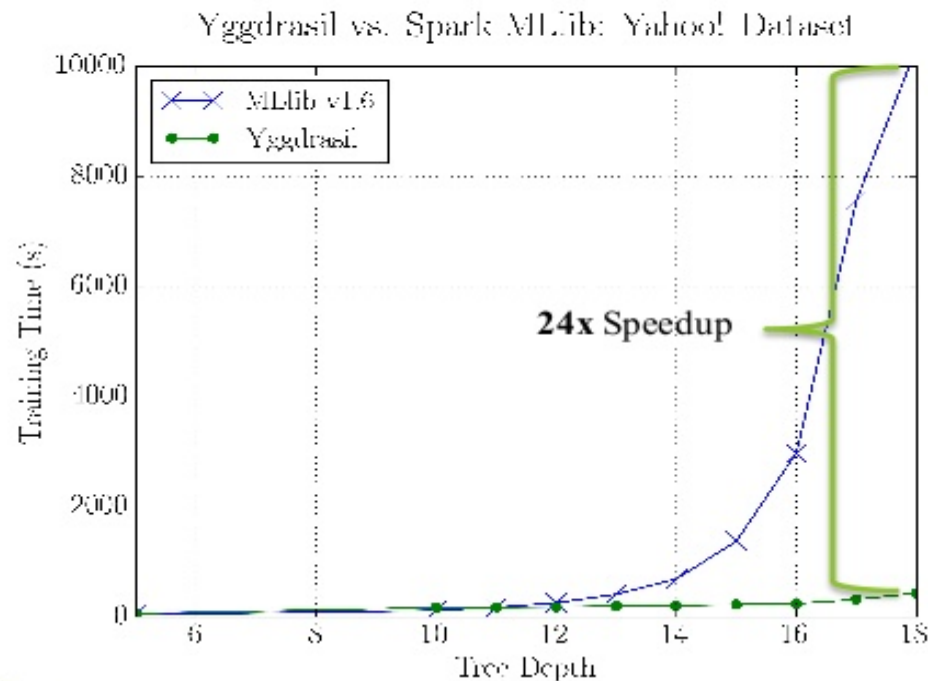
Results



- Classifying handwritten digits
- 8.1 million rows
- 784 columns
- 18.2 GB (< 1% non-zeros)



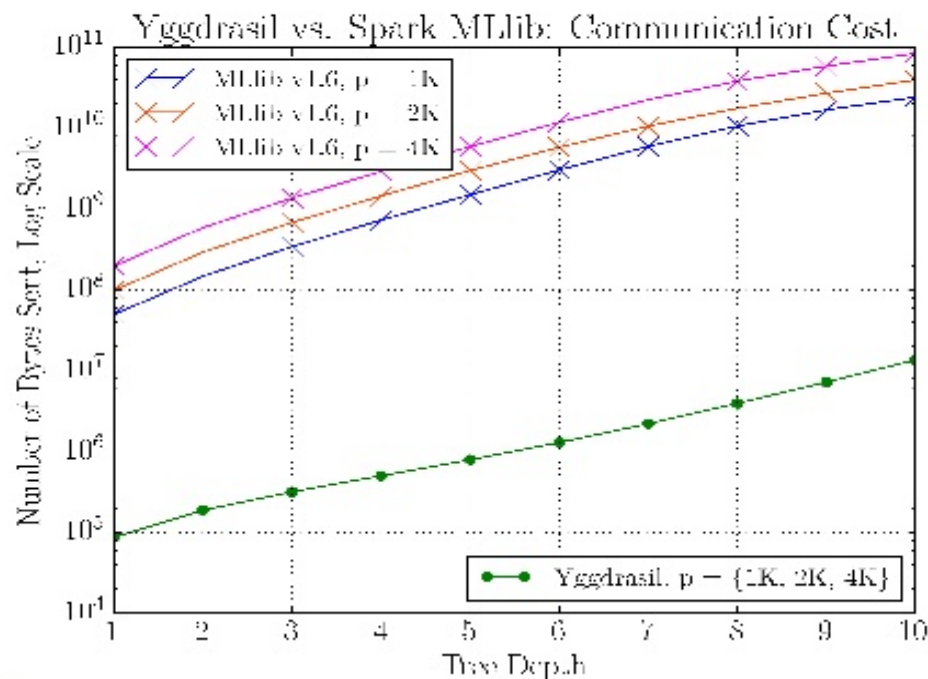
Results



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



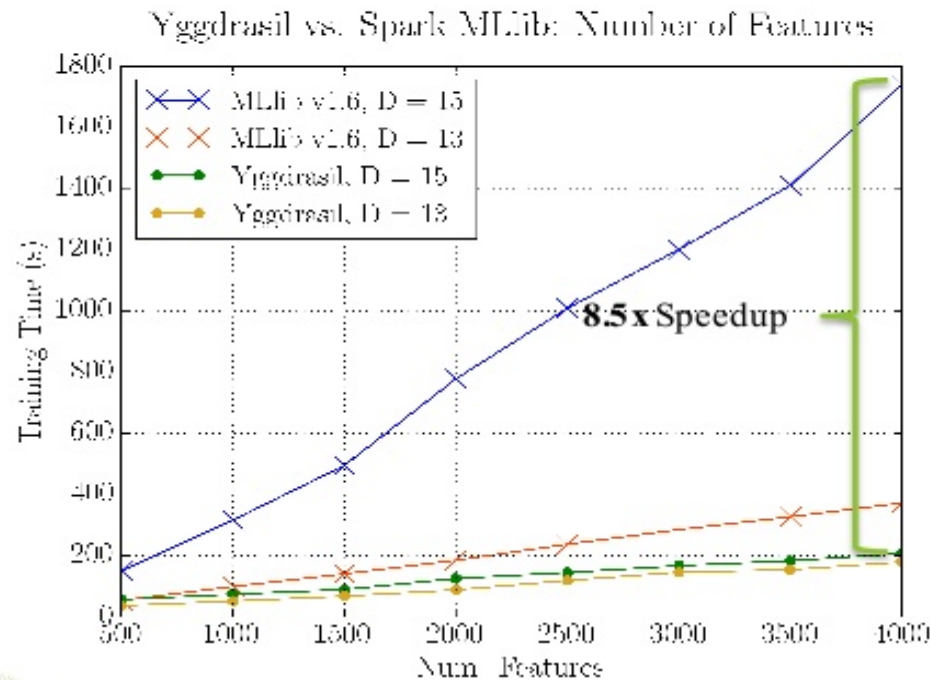
Results



- 2 million rows
- For Yggdrasil, communication cost is independent of the number of features



Results



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



Using Yggdrasil

- Available as a Spark package – [download here](#)
- Direct integration with Spark ML Pipeline API
- Support for various inputs: DataFrame, RDD[Labeled Point], or Parquet files



Using Yggdrasil

Before:

```
val dt = new DecisionTreeClassifier()  
    .setFeaturesCol("indexedFeatures")  
    .setLabelCol(labelColName)  
    .setMaxDepth(params.maxDepth)  
    .setMaxBins(params.maxBins)  
    .setMinInstancesPerNode(params.minInstancesPerNode)  
    .setMinInfoGain(params.minInfoGain)  
    .setCacheNodeIds(params.cacheNodeIds)  
    .setCheckpointInterval(params.checkpointInterval)
```



Using Yggdrasil

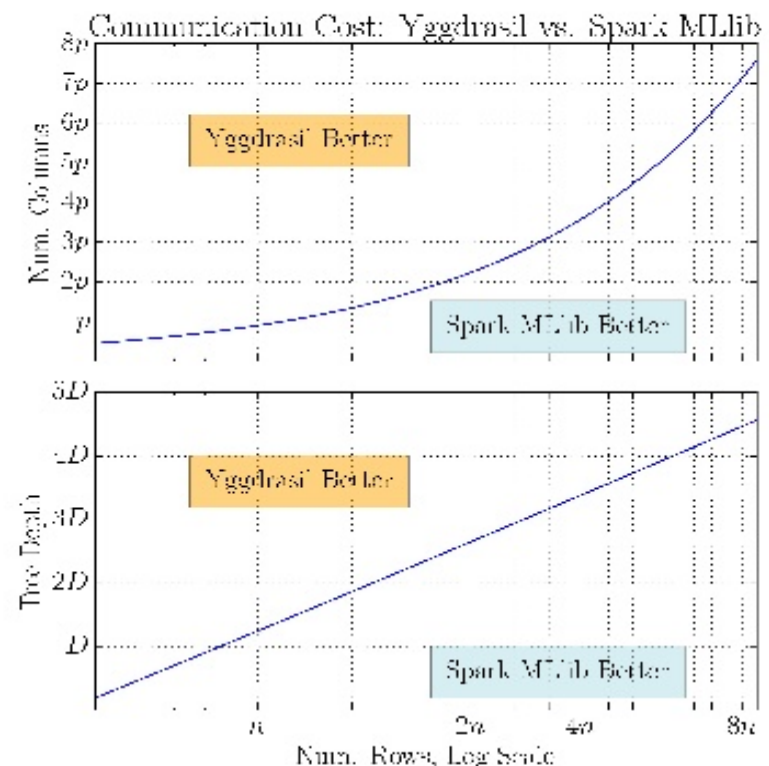
After:

```
val dt = new YggdrasilClassifier()  
    .setFeaturesCol("indexedFeatures")  
    .setLabelCol(labelColName)  
    .setMaxDepth(params.maxDepth)  
    .setMaxBins(params.maxBins)  
    .setMinInstancesPerNode(params.minInstancesPerNode)  
    .setMinInfoGain(params.minInfoGain)  
    .setCacheNodeIds(params.cacheNodeIds)  
    .setCheckpointInterval(params.checkpointInterval)
```



Why should I have to choose?

- If Spark MLlib is better for shallow trees and few features...
- And Yggdrasil is better for deeper trees and many features...
- Why can't I have both?



Future Work

- You should be able to have both!
- Next steps:
 - Merge Yggdrasil into Spark MLlib v2.x
 - Add decision rule that automatically chooses the best partitioning strategy for you:
 - by row (MLlib v1.6); or
 - by column (Yggdrasil)



THANKS!

Email: fabuzaid@csail.mit.edu

GitHub: <https://github.com/fabuzaid21/>

Twitter: [@FirasTheBoss](https://twitter.com/FirasTheBoss)

Website: <http://firasabuzaid.com>



SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO