JLBoost.jl is a gradient tree-boosting library

What is gradient tree-boosting?

- This “poster” is too short to do it justice
- You fit an ensemble of tree models in an additive manner by minimizing
  - *loss(existing_trees + new_tree)*
- Most commonly used to predict categories
  - Survived Titanic vs Not
  - Adelie vs Chinstrap vs Gentoo
  - Other types of predictions too!

Isn’t XGBoost and LightGBM juggernauts in this area?

- Yes

Are’t there XGBoost.jl and LightGBM.jl already?

- Yes

Why not just use XGBoost.jl/LightGBM.jl?

- You can but...
- They don’t play nice with the rest of the Julia ecosystem
- You can’t use DataFrames.jl nor Tables.jl with it
- It’s not coded in Julia, so not customizable and not composable

Why create JLBoost.jl?

- A pure 100%-Julia implementation of gradient boosted trees
- Coding it in a “high”-level language like Julia = easier to experiment
- Coding it in Julia makes it “hackable” (see slide after)

How to use JLBoost.jl?

JLBoost.jl plays nice with Tables.jl (so DataFrames.jl)

```
using JLBoost, RDatasets
iris = dataset("datasets", "iris")

iris[!, :is_setosa] = iris[!, :Species] .== "setosa"
target = :is_setosa

features = setdiff(names(iris), ["Species", "is_setosa"])

# fit one tree
# ?jlboost for more details
xgtreemodel = jlboost(iris, target)
```

Fit larger-than-RAM datasets with JDF.jl

Fit model on `JDF.JDFFile` - enabling larger-than-RAM model fit

Sometimes, you may want to fit a model on a dataset that is too large to fit into RAM. You can convert the dataset to JDF format and then use `JDF.JDFFile` functionalities to fit the models. The interface `jlbosst` for `DataFrame` and `JDFFiLe` are the same.

The key advantage of fitting a model using `JDF.JDFFile` is that not all the data need to be loaded into memory. This is because `JDF` can load the columns one at a time. Hence this will enable larger models to be trained on a single computer.

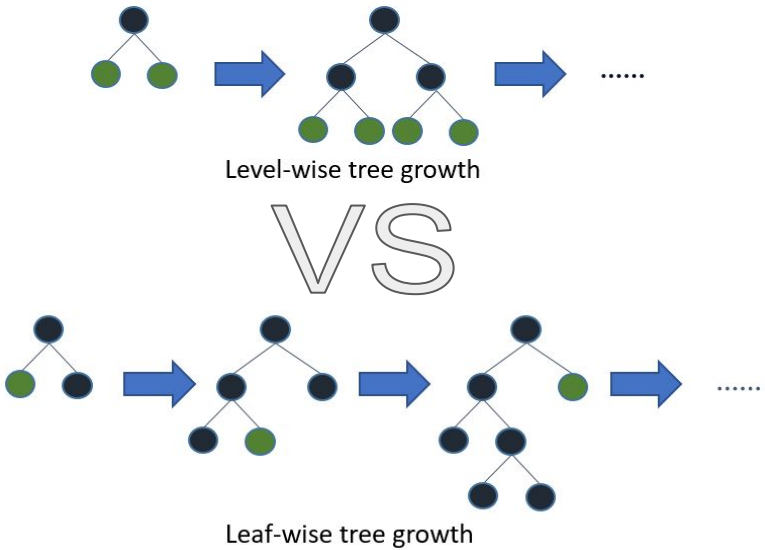
```
using JLBoost, RDatasets, JDF
iris = dataset("datasets", "iris")

iris[!, :is_setosa] = iris[!, :Species] .== "setosa"
target = :is_setosa

features = setdiff(names(iris), [:Species, :is_setosa])

savejdf("iris.jdf", iris)
irisdisk = JDFFile("iris.jdf")

# fit using on disk JDF format
xgtree1 = jlboost(irisdisk, target, features)
xgtree2 = jlboost(iris, target, features; nrounds = 2, max_depth = 2)

# predict using on disk JDF format
iris.pred1 = predict(xgtree1, irisdisk)
iris.pred2 = predict(xgtree2, irisdisk)

# AUC
AUC(-predict(xgtree1, irisdisk), irisdisk[!, :is_setosa])

# gini
gini(-predict(xgtree1, irisdisk), irisdisk[!, :is_setosa])

# clean up
rm("iris.jdf", force=true, recursive=true)
```

JLBoost.jl is *hackable*

What is *hackable*?

- You can *customize* all key aspects
- E.g. easy to define custom loss functions

A concrete example of *hackable*-ness



* Images froms https://github.com/microsoft/LightGBM/blob/master/docs/Features.rst

Making tree growth *hackable*

- First split
  - find the *best* splitting point for all features
  - Split on feature that gives best *gain*
- After first split, there are many way to proceed
  - XGBoost uses level-wise but (see left)
  - LightGBM uses leaf-wise (see left)
- Why not let the user *customize* the choice?
  - JLBoost.jl allows the user to pass a function to choose how to proceed - *grow_policy*.
  - By passing a function. The user can *experiment* and design novel methods to choose how to proceed!
- This level of customization is NOT possible with XGBoost nor LightGBM because their choices are fixed strings not functions!

What's next?

Near Future

- Upcoming features and enhancements
  - Need to add support for *Union{T, Missing}*
  - Need to add support for *Categorical*
- Performance
  - Quite slow compared to XGBoost.jl
  - But lots of low hanging fruit available
- GPU support
  - GPU support via CUDA.jl needs to be investigated

Ultimate aim

- Make everything *hackable*
  - Make it *easy* to experiment with different tree building techniques
- Easy to deploy models
- Easy to manipulate the tree after fitting
  - E.g. pruning, editing the tree. These are difficult to achieve in C++ based libraries like XGBoost