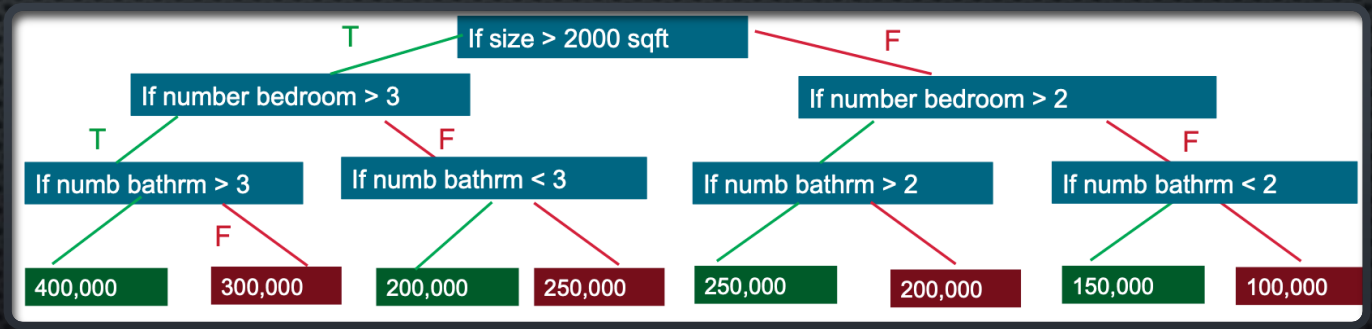
XGBOOST

Group 4

Nick Kartschoke, Alex Mora, Chris Ng, Jakob Thunen

WHAT IS XGBOOST?

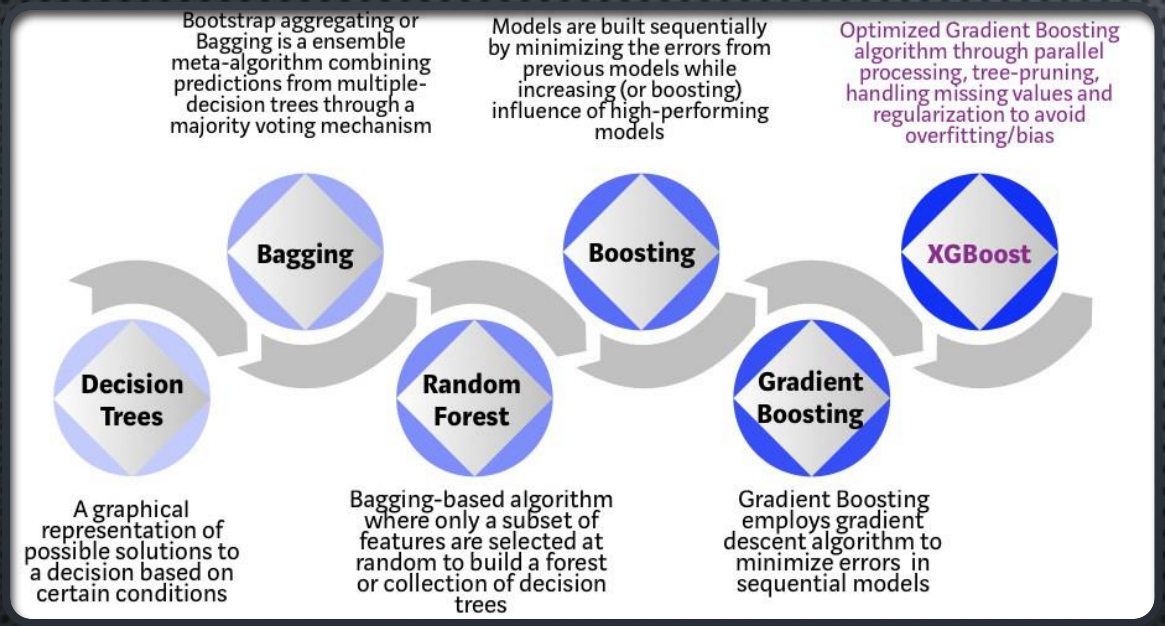
XGBoost, short for "Extreme Gradient Boosting", is a machine learning algorithm based on collections of random decision trees, like the one pictured here.



How does it work?

XGBoost is similar to other random decision tree-based algorithms like Random Forest in that it uses a collection of random decision trees to generate its predictive model. Unlike a random forest, however, it "boosts" the forest by combining smaller, weaker trees into larger, stronger ones.

BAGGING AND BOOSTING



"Bagging" in a random forest algorithm refers to choosing random samples of data to construct decision trees, minimizing variance and over-fitting.

"Boosting" refers to the combination of weaker trees to create a stronger one, as in Gradient-boosted Decision Tree models. This reduces bias and under-fitting.

XGBoost utilizes both of these techniques to get the best of both algorithms, leading it to be the model of choice for numerous ML competitions.

What else does it do?

Unlike GBDTs, XGBoost trees are built in parallel to each other instead of one at a time. The model is analyzed per layer instead of per tree, ensuring the optimal quality of each split in the decision trees.

Building the decision trees in parallel also increases efficiency and scalability by allowing for parallel processing. This effect is especially profound when using GPUs with thousands of processing cores to create models.

Advantages/Disadvantages

Advantages

- Tree Pruning
- Handling of missing/Sparsity-aware: XGBoost contains a sparsity-aware split finding algorithm and handles different types of sparsity patterns in the data
- Built-in cross validation
- Good job Limiting overfitting Trees
  - Uses Lasso and Ridge regularization

Disadvantages

- Has a lot of hyper parameters, thus hard to tune
- Can be overfitted if parameters are not in tune
- Sensitive to outliers
- Partially a "black box"
  - Hard to interpret the inner workings

Processing steps

- In order for xgboost to be able to use the desired dataset, a transformation is needed.
  - DMatrix format
  - Transforms a NumPy array to dmatrix format
- With xgboost (sparse aware), will create a default direction for missing value nodes. At the prediction time, if the path goes through a missing value, the default path is followed. It is good practice to clean missing data, but xgboost does allow for missing variables.
- Tree based classifier - the need to standardize features is not required, but if there is a want for high performance, it is encouraged.

```
D_train = xgb.DMatrix(X_train, label=Y_train)
D_test = xgb.DMatrix(X_test, label=Y_test)
```

HYPERPARAMETERS

- General parameters – Guide the overall functioning of the model and relate to which booster is being used
- Booster parameters – Guide each boosting step
- Task parameters – Used to define the optimization objective by specifying the learning task and the corresponding learning objective

Common Hyperparameters

- Booster – default is gbtree, could also be gblinear or dart
- objective – determines the loss function to be used i.e. reg:linear or reg:logistic
- colsample_bytree – percentage of features used per tree (high value can lead to overfitting)
- learning_rate (eta) - [0-1] a higher learning rate means faster computation, a lower learning rate means you can get closer to the perfect model fit
- max_depth – determines how deep each tree can grow while boosting
- alpha – [0-∞] the higher the alpha, the more conservative the model/the more regularization
- n_estimators – number of trees you want to build