Gradient Boosting: inside a black box

Vitalii Duk

Agenda

- Decision Trees
- Ensembles: bootstrap aggregating
- Random Forest
- Ensembles: boosting
- AdaBoost
- Gradient Boosting
- Gradient Boosted Trees
- XGBoost

Decision Trees: overview

- **Supervised** learning algorithm
- **Binary tree** as the underlying data structure
- Works for both **regression** and **classification** tasks
- Uses **impurity measure** to decide where split and how to build a tree

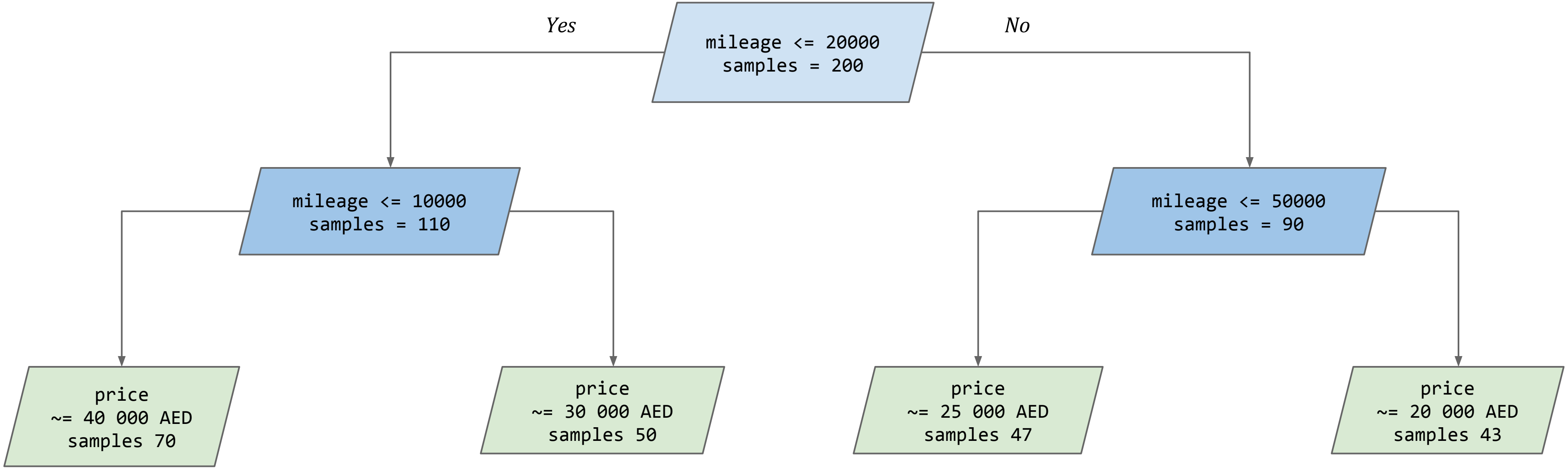
Advantages:

- Simple to understand
- Can be easily visualized
- Don’t require heavy computations while predicting results
- Can handle categorical and numerical data

Disadvantages:

- Can be easily overfitted
- Can be biased if the dataset is skewed
- Learning decision tree structure is NP-complete task
- Can’t guarantee finding globally optimal decision tree

Decision Trees: visualization



tree depth = 2

Decision Trees: split

How do we know where to split?

15	31	83	19	27	33	56	23	12	78
1	0	0	1	1	0	1	0	1	0

12	15	19	23	27	31	33	56	78	83
1	1	1	0	1	0	0	1	0	0

12	15	19	23	27	31	33	56	78	83
1	1	1	0	1	0	0	1	0	0

1. Sort unique features values in ascending order.
2. Try to split at each unique value and calculate the impurity of left and right leaf.
3. Select the threshold value with the lowest impurity score.

Decision Trees: tree construction

Alghorithm	**Impurity measure**	*Impurity measure equation*	*Features*
CART	Gini Index	$1 - \sum_{i}^{K} p_i^2$	Use complexity-based pruning, handles numerical and categorical values, handle missing values.
ID3	Information Gain	$\mathrm{H} = -\sum_{i=1}^{K} p_k \log_2 p_k$ $\Delta H = H - \frac{m_L}{m} H_L - \frac{m_R}{m} H_R$	No pruning is done, handles only categorical values, can't handle missing values.
C4.5	Information Gain Ratio	$GR(S, A) = \frac{Gain(S, A)}{IntI(S, A)}$	Use error-based pruning, handles numerical and categorical values, handle missing values.

Decision Trees: regularization

- **Pre-pruning** - stop growing tree when:
  - number of elements in a leaf node is less than user-defined threshold
  - current node don't improve impurity measure
  - maximum tree depth or maximum number of leafs is reached
  - leaf node contains same/similar items

- **Post-pruning** - grow decision tree as much as possible:
  - start trimming tree nodes from the bottom
  - check if generalization of the model improves after each pruning iteration using cross-validation

Decision Trees: tuning parameters

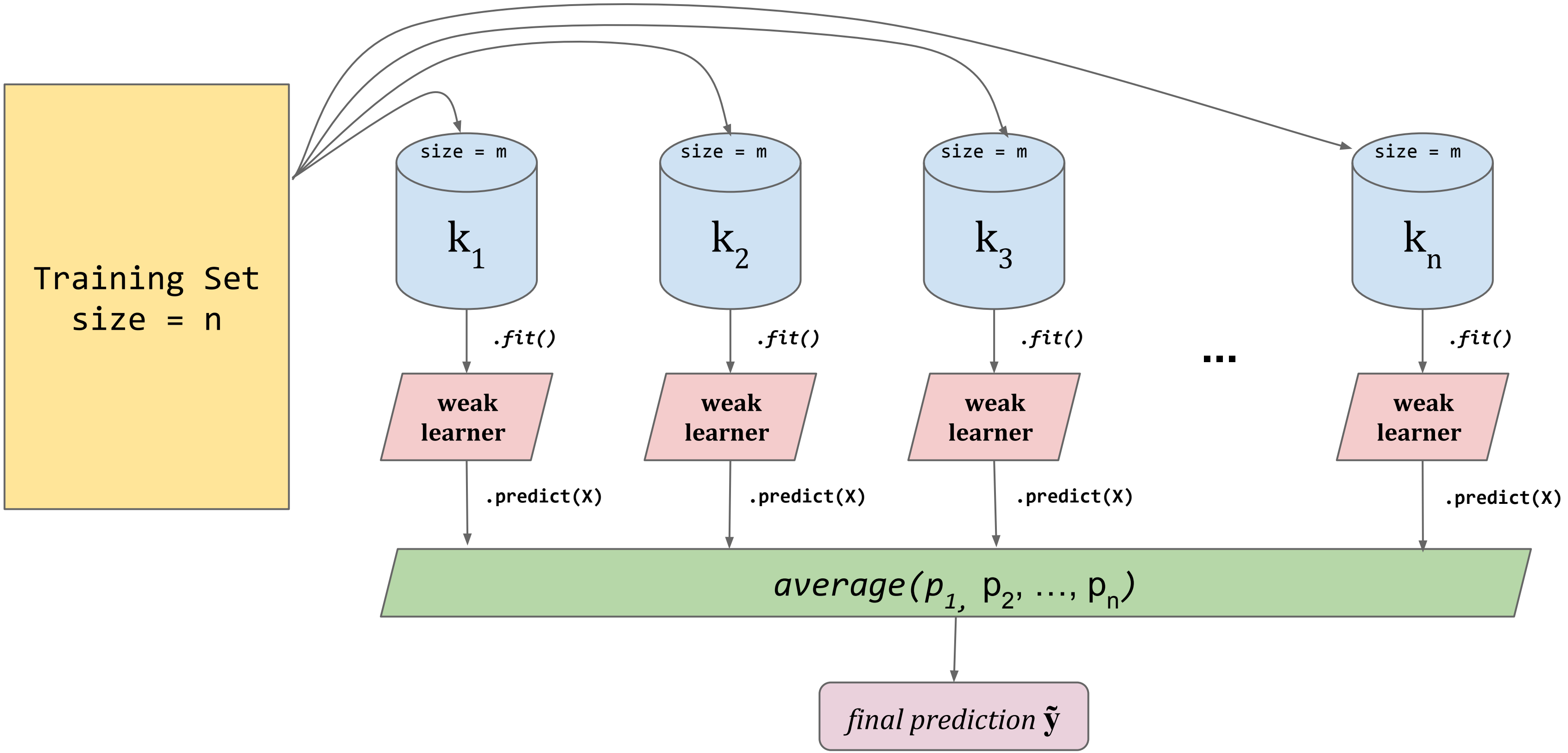
- `Parameters to tune:`
  - `max_depth` - maximum depth of the tree
  - `max_leaf_nodes` - maximum number of nodes of each leaf
  - `min_samples_split` - minimum number of samples to split the leaf
  - `min_impurity_split` - continue to split if this value is less than node's impurity value

Bootstrap aggregating: overview

Bootstrapping in statistics means ***random sampling with replacement***

- Assume we have a dataset of size n
- Let’s create k subsets of original dataset
- Each subset k_i consist of m elements uniformly sampled from original dataset with replacements
- Train separate “base models/learners” on each of k subsets
- Regression: output average value of k learners
- Classification: output most common label of k learners

Bootstrap aggregating: visualization



Random Forest

- ***Supervised*** learning algorithm
- Based on idea of ***bagging*** (bootstrap aggregating)
- Uses ***Decision Trees*** as a base learner
- Train each base learner using a ***subsample*** of data with replacements

Tuning parameters:

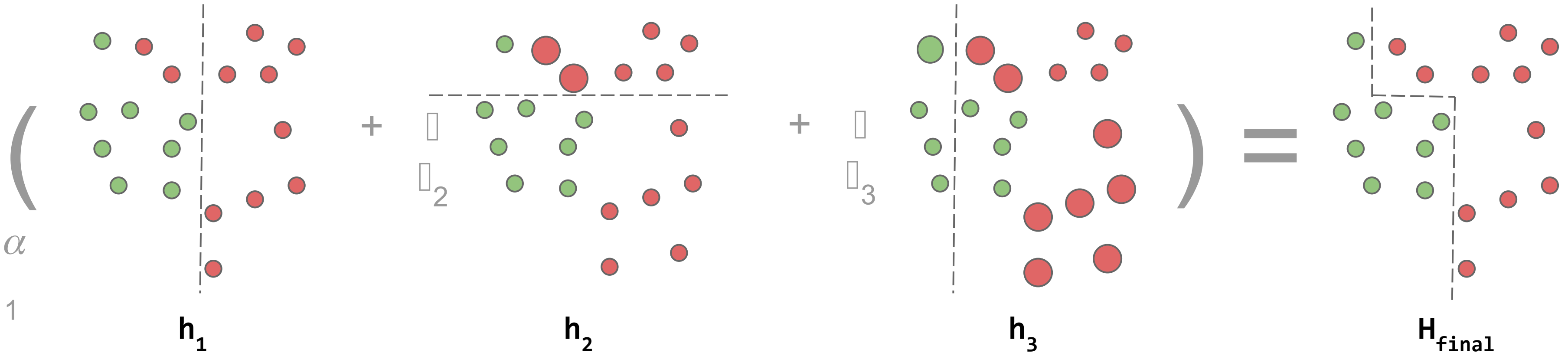
  - **`n_estimators`** - number of base learners to fit
  - **`max_features`** - percentage of features to consider while building a tree
  - **`max_depth`** - maximum depth of the tree
  - **`max_leaf_nodes`**- maximum number of nodes of each leaf

Boosting

- Based on a ensemble of base learners
- Base learners are stacked, one after another
- Any input item $\boldsymbol{X}$ has a weight, initially uniformly distributed
- Each base learner is trying to predict the output value $\boldsymbol{y}$ based on input $\boldsymbol{X}$
- Weights of poorly predicted items are increased
- Next base learner is taking re-weighted input $\mathbf{X}$ from his predecessor and is trying to improve predictions

AdaBoost

- ***AdaBoost*** stands for ***Adaptive Boosting***
- Minimizes exponential loss
- Works only for binary classification *
- Desired outputs should be ***{-1, 1}***



AdaBoost

- Take a dataset $x_1, x_2, \ldots, x_n$ as a input
- Set initial weights $w_1, w_2, \ldots, w_n$ of each training example as $1/n$
- Define k as a number of ***weak learners*** (usually: decision tree of depth 1)

for each weak learner ***do:***

- Train a weak learner using weighted samples
- Make predictions using current trained learner
- Calculate the error made by current learner
- Add more weight to wrongly predicted items

end for

Gradient Boosting

Same idea as behind **Boosting**:

- build ensemble of ***k*** stacked base learners
- additive learning approach to predict final output

Differences:

- start with the initial guess $f_0(x)$ which is usually 0, next steps will continuously improve that
- on each step base learner predicts pseudo-residuals instead of original target value

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

- any differentiable loss function $L(y_i, F(x_i))$ can be used

Gradient Boosted Trees

- How can we use Decision Trees as a base learner now?
- How we can calculate gradient of a Decision Tree?

- Few changes to make:
  - regression trees used internally to predict pseudo-residuals for both classification and regression
  - select split point which minimizes the loss

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

Gradient Boosted Trees: residuals fitting

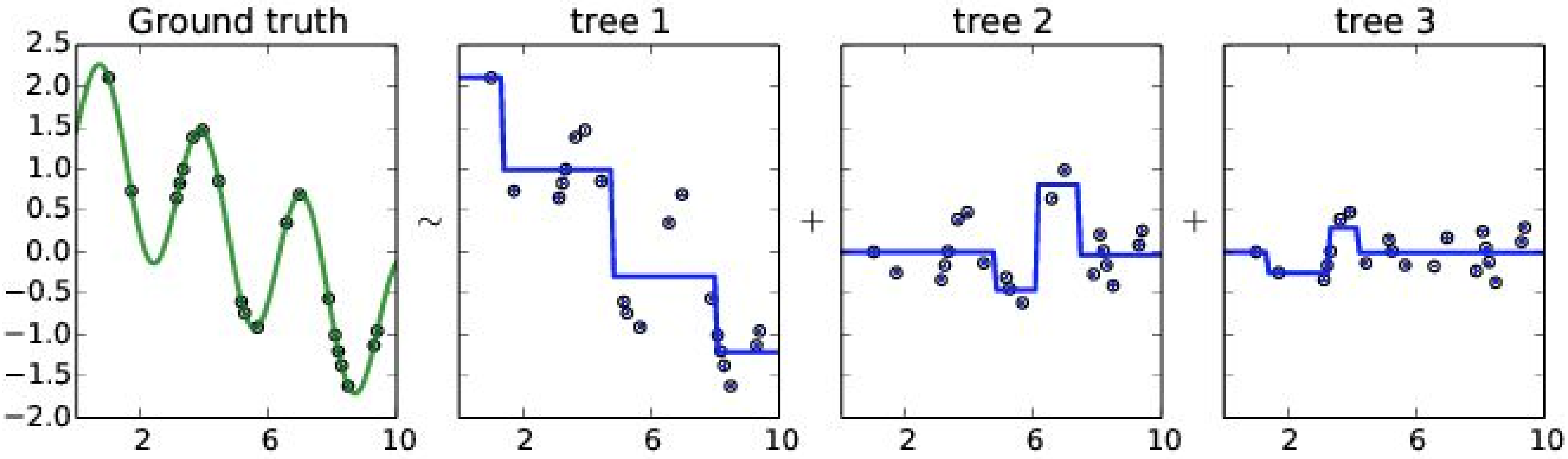
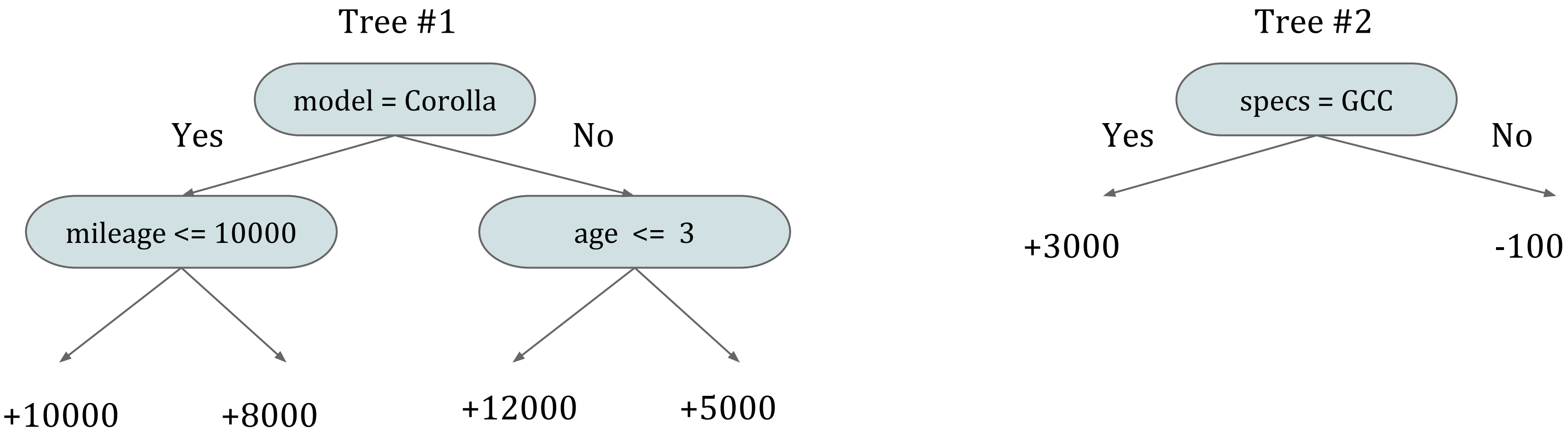


image source: quora.com

Gradient Boosted Trees

How to calculate a final predicted value?



H(**model**=Toyota, **mileage**=6000, **age**=2, **specs**=GCC) = $(+10000 * \alpha_1) + (+3000 * \alpha_2)$

α_n - learning rate

Gradient Boosted Trees

- How to predict the output for a binary classification?
    - uses ***Regression Trees*** as a base learner
    - even for classification fits each base learner regressor to predict pseudo-residuals
    - use ***Logistic Loss*** instead of ***Squared Loss***
    - apply ***inverse logit*** transformation to the outputs to get probabilities

XGBoost

- What is XGBoost?
  - One of the implementations of **Gradient Boosting** algorithm
  - Written in **C++** with **bindings** in **Python, R, Julia, Scala**
  - Internal **DMatrix** data structure to speed-up computational process
  - A lot of **built-in objectives** for classification, regression and ranking
  - Supports **distributed training** on several machines in the cloud
  - A lot of performance tricks to speed-up learning process

XGBoost tuning parameters

`n_estimators` - number of stacked base learners
`eta` - learning rate which makes fitting process more conservative
`gamma` - minimum loss reduction required to make a new split in a tree
`max_depth` - control regression tree depth
`max_leaves` - limit maximum number of leaves instead of overall tree depth
`tree_method`:
- `approx` - apply binning of continuous variables into percentile buckets
- `exact` - use all unique values of each feature
- `hist`* - cache and reuse bins on each iteration

`subsample` - percentage of data to use while fitting new tree
`colsample_bytree` - percentage of features to use while fitting new tree
`colsample_bylevel` - percentage of features to use while making a new split

XGBoost: quick example

https://github.com/root-ua/gbm-intro-meetup

Thanks!