Ensemble Methods: Bagging and Boosting

FA590 Statistical Learning in Finance

Dr. Zonghao Yang

Stevens Institute of Technology

Previous Lecture

- Understand the fundamental components and structure of decision trees, including nodes, branches, and terminal nodes
- Learn the processes of training a decision tree, including feature selection, splitting, and pruning to prevent overfitting
- Explore metrics like Gini impurity and cross-entropy to evaluate and select optimal splits in classification tasks
- Recognize the differences between decision trees and other models, particularly in handling missing data, nonlinear relationships, and categorical features

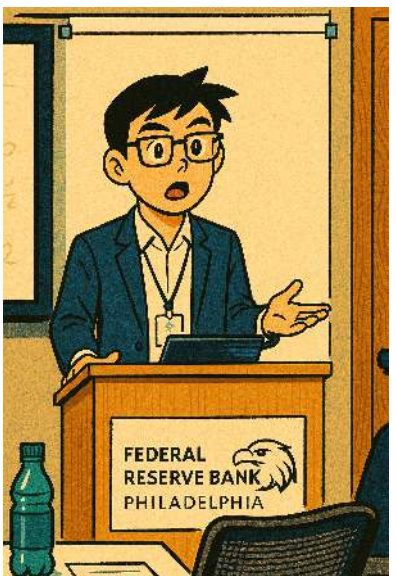
Learning Objectives

- Understand the fundamentals of ensemble methods, including bagging and boosting
- Learn the mechanics of bagging and random forests
- Gain insight into boosting techniques, including algorithms such as AdaBoost and Gradient Boosting, and comprehend how each model corrects the errors of its predecessors
- Apply ensemble methods to regression and classification tasks

Part I

Ensemble Methods

Height Guessing Game



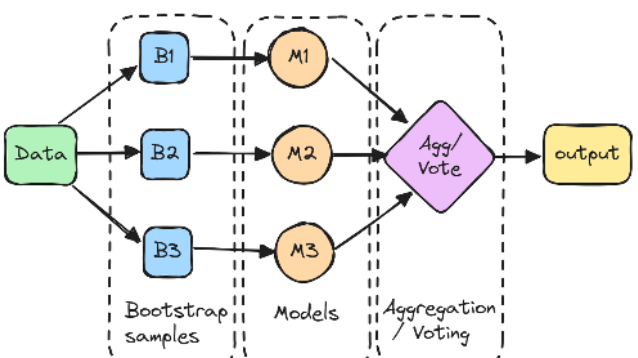
How tall is the instructor?

Ensemble Methods

- **Ensemble** means combining multiple supervised models into a "super-model"
- The principle of combining methods is popular for reducing risk
  - For example, in finance, portfolios are created to reduce investment risk
- In predictive modeling, "risk" is equivalent to variation in prediction error
  - The more our prediction errors vary, the more volatile our predictive model
- Using an average of two predictions can potentially lead to smaller error variance, and therefore better predictive power

BAGGING

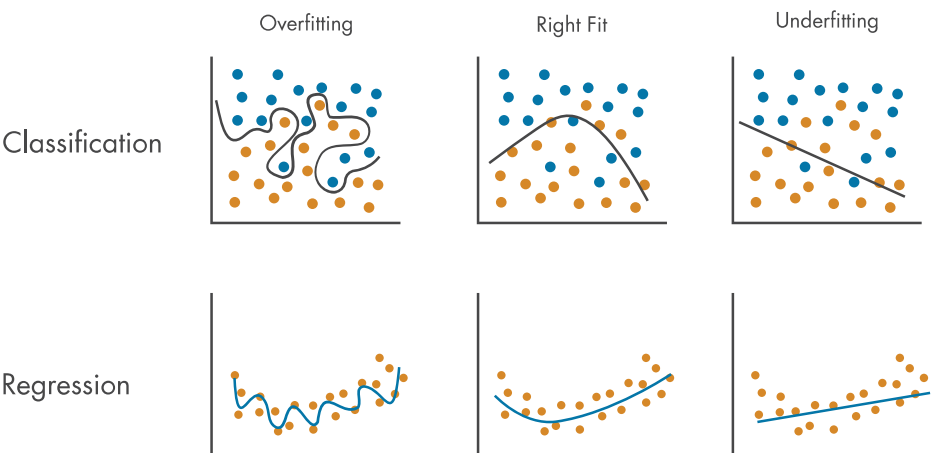
- **Bagging** is short for “bootstrap aggregating”; bagging is an ensemble algorithm
  - Bootstrap: Fit multiple models on different subsets of the training dataset
  - Aggregating: Combine the predictions of all models



Random Forest

- A problem with bagging: If a few features dominate (are very powerful), all trees will split on the same features
  - Hence, all trees look similar, and the averaging across trees is less useful
- In **random forest** , each node only gets to see a randomly selected subset of all available features, which guarantees heterogeneous trees
- The size of the feature subset is a hyperparameter of the algorithm
  - A typical choice is the square root of the total number of features
  - Given some k features, for each split-point, select $\sqrt{k}$ features at random, and only use those to find optimal split points

Overfitting

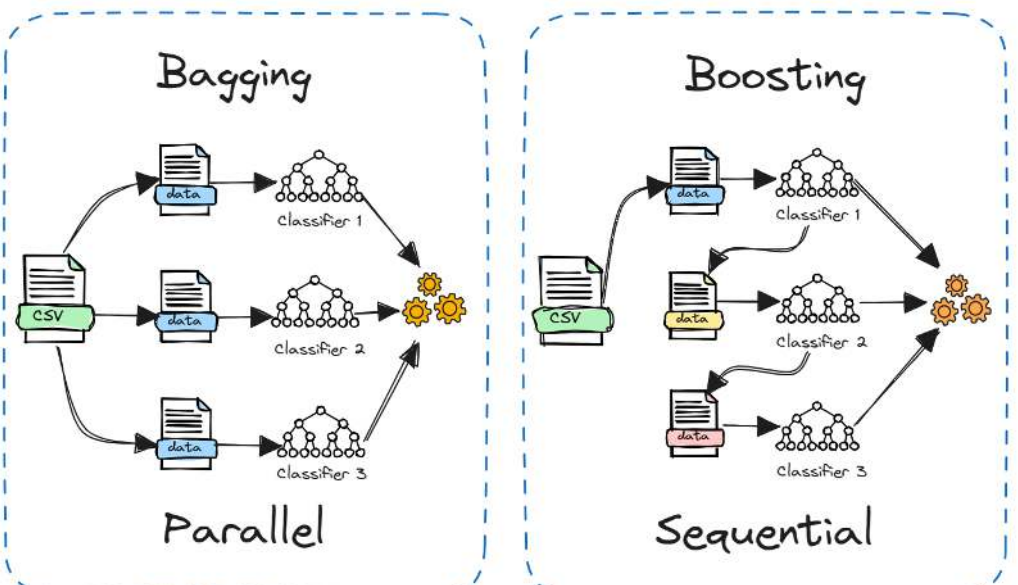


- Use a validation sample to tune the hyperparameters

Random Forest: Hyperparameters

- Tree depth (`max_depth`): Very important
- Fraction of data per final leaf (`min_samples_leaf`): Very important; correlated with tree depth)
- Fraction of feature per split (`max_features`): Somewhat important
- Minimum impurity decrease (`min_impurity_decrease`): Somewhat important
- Number of trees (`n_estimators`): Not so important, just choose large enough
- Decision criterion (`criterion`): e.g. Gini score or cross-entropy; not very important in practice

Bagging vs. Boosting



Boosting

- The idea of **boosting** is to train weak learners sequentially, each trying to correct its predecessor
- Unlike bagging (e.g., random forest), boosting trains trees sequentially rather than in parallel
- The most common form of boosting is **gradient boosting**
  - Other methods, such as **AdaBoost**
- Boosting methods are often winners of Kaggle competitions

AdaBoost for Classification

1. Initialize the observation weights $w_i = \frac{1}{N}$, $i = 1, 2, \ldots, N$
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i
   (b) Compute
   $$\text{err}_m = \sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq G_m(x_i))$$
   (c) Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
   (d) Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot \mathbb{I}(y_i \neq G_m(x_i))\right]$, $i = 1, 2, \ldots, N$
   (e) Normalize the weights w_i so that they sum to 1
3. Final prediction function: $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

AdaBoost for Regression

1. Initialize the observation weights $w_i = \frac{1}{N}$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a regression model $G_m(x)$ to the training data using weights w_i
   (b) Compute

$$\text{err}_m = \sum_{i=1}^{N} w_i \cdot \text{error}_i, \text{ where } \text{error}_i = |y_i - G_m(x_i)|$$

   (c) Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
   (d) Set $w_i \leftarrow w_i \cdot \exp\left(\alpha_m \cdot \frac{\text{error}_i}{\max_j \text{error}_j}\right)$, $i = 1, 2, \ldots, N$
   (e) Normalize the weights w_i so that they sum to 1
3. Normalize α_m so that they sum to 1
4. Final prediction function: $G(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$

Part II

Gradient-Tree Boosting Algorithm

Why Gradient Boosting?

- Assume an algorithm $f(\cdot)$ and the sum of squared errors for a sample x_i:

$$L(y_i, f(x_i)) = \frac{1}{2}(f(x_i) - y_i)^2$$

- The gradient for $f(\cdot)$:

$$\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial (f(x_i) - y_i)^2/2}{\partial f(x_i)} = f(x_i) - y_i$$

- The gradient is equivalent to the residuals
- For other loss functions, the gradient is a proxy for the residual
- **Gradient-tree boosting** Instead of training the new algorithm on the residuals, it trains it on the gradient of the loss function
  - This property makes the algorithm numerically fast, while flexible and applicable to any loss function (regression, classification, etc.)

Gradient-Tree Boosting Algorithm

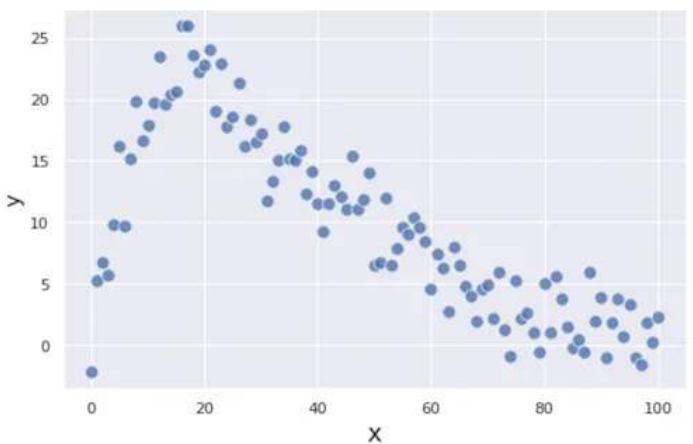
1. Initialize $f_0(x) = \gamma_0 = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$, compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   (b) Fit a regression tree h_m to the targets r_{im}
   (c) Update $f_m(x) = f_{m-1}(x) + \nu \cdot h_m(x)$, where ν is the learning rate
3. Output the final model $\hat{f}(x) = f_0(x) + \sum_{m=1}^{M} \nu \cdot h_m(x)$

A Regression Example

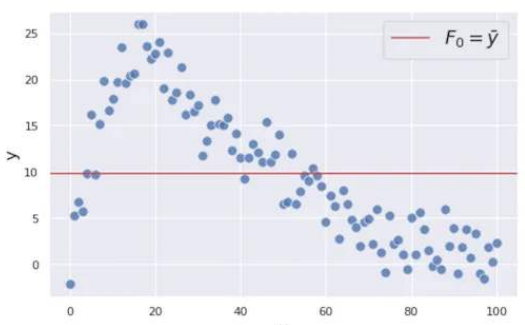
- Consider a regression example with the response y and just one feature x[1]
- There is a nonlinear relationship between x and y. Hence, linear regression is not appropriate.

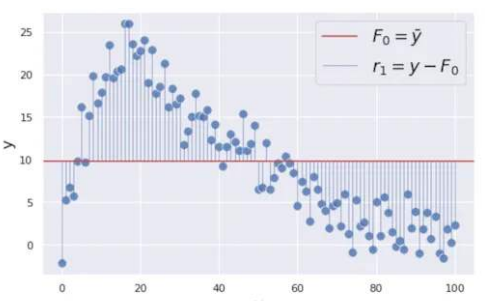


[1] "All You Need to Know about Gradient Boosting Algorithm" by Tomonori Masui: https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502

A Regression Example: Initialization

- Initialize $f_0(x) = \gamma_0 = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
- If L is the least-squares loss function, i.e., $L(\gamma) = \frac{1}{2}\sum_{i=1}^{n}(\gamma - y_i)^2$, the solution to the optimization problem is $\gamma_0 = \bar{y}$



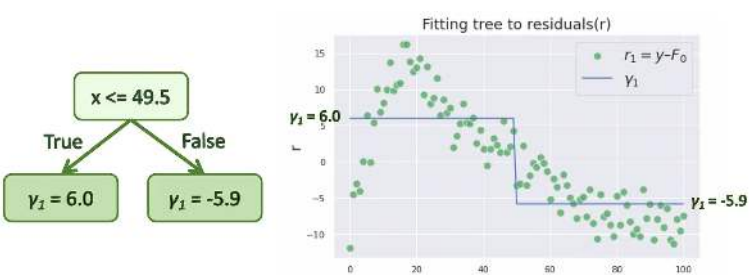


A Regression Example: First Iteration

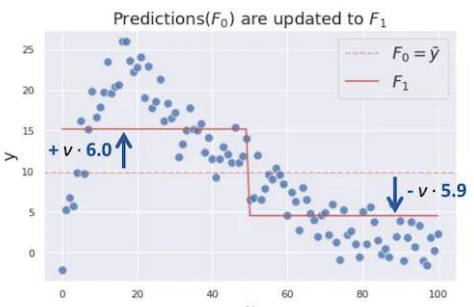
- Gradient: $-\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_0} = y - f_0(x)$
- Train the first decision tree (with depth 1 as illustration) on the gradients, and obtain the predictions $\gamma_1 = (\gamma_{1,1}, \gamma_{2,1})$
- Then, update the learning algorithm

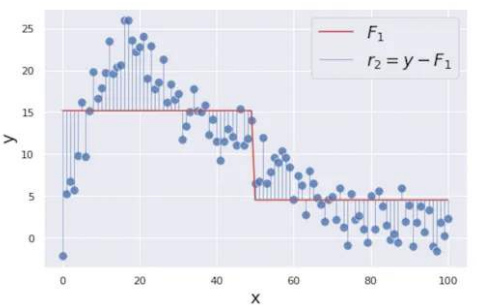
$$f_1(x) = \begin{cases} f_0(x) + \nu\gamma_{1,1} & \text{if } x \in R_{1,1} \\ f_0(x) + \nu\gamma_{2,1} & \text{if } x \in R_{2,1} \end{cases}$$

- Note that ν is the learning rate, and it is typically determined via cross-validation



A Regression Example: After First Iteration

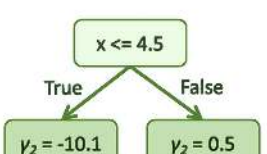


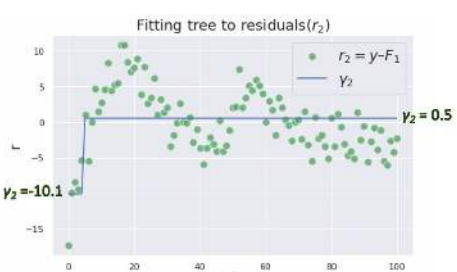


A Regression Example: Second Iteration

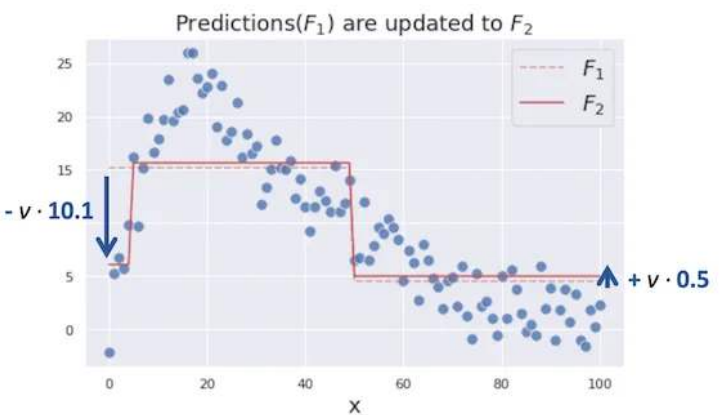
- Gradient: $y - f_1(x)$
- Train another decision tree on the gradients, and obtain the predictions $\gamma_2 = (\gamma_{1,2}, \gamma_{2,2})$
- Then, update the learning algorithm

$$f_2(x) = \begin{cases} f_1(x) + \nu\gamma_{1,2} & \text{if } x \in R_{1,2} \\ f_1(x) + \nu\gamma_{2,2} & \text{if } x \in R_{2,2} \end{cases}$$



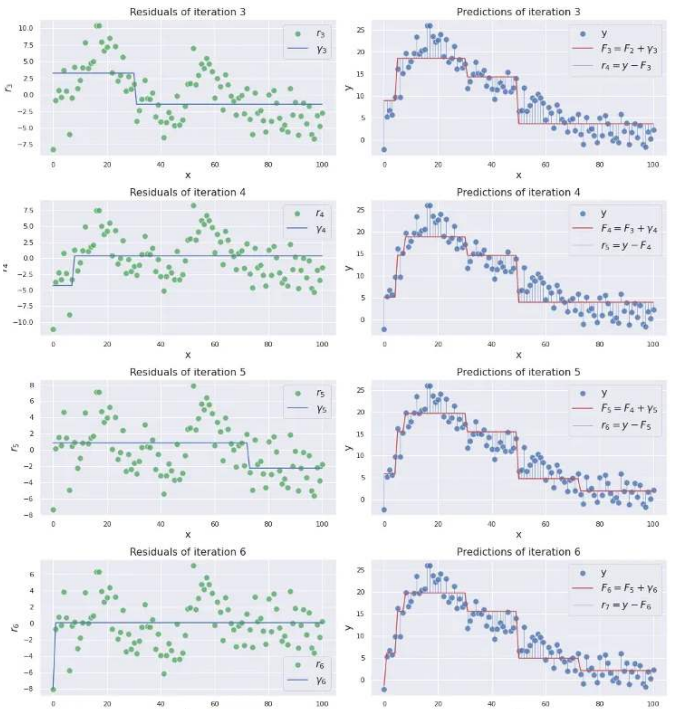


A Regression Example: After Second Iteration



A Regression Example: N^{th} Iteration

- Repeat this process M times, where M is a hyperparameter
- Or stop after some predefined criterion (e.g., when validation error does not improve for several iterations)



Revisit Gradient-Tree Boosting Algorithm

1. Initialize $f_0(x) = \gamma_0 = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$, compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   (d) Update $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$

XGBoost and LightGBM

- **XGBoost** and **LightGBM** are two popular variations of gradient-tree boosting[2]
- Both are state of the art for speed and performance
- The two algorithms differ in:
  - leaf-wise tree growth versus level-wise tree growth
  - how they handle residuals
  - how they handle categorical data

[2] "XGBoost vs LightGBM: How Are They Different" by Sumit Saha: https://neptune.ai/blog/xgboost-vs-lightgbm