PGE 383 Subsurface Machine Learning

Lecture 15b: Gradient Boosting

Lecture outline:

- **Decision Tree Review**
- **Boosting Methods**
- **Tree-based Gradient Boosting Regression**
- **Tree-based Gradient Boosting Methods Hands-on**

Motivation

Gradient Boosting Models Build on the Concepts,

- ensemble approach – combining multiple estimators

- optimization by gradient descent optimization

Gradient boosting are quite competitive in many applications.

Double Tree of Casorzo in Asti, Piemonte, Italy, cherry tree growing on another tree, image from https://urnabios.com/double-tree-casorzo/.

PGE 383 Subsurface Machine Learning

Lecture 15b: Gradient Boosting

Lecture outline:

- **Decision Tree Review**

Decision Tree

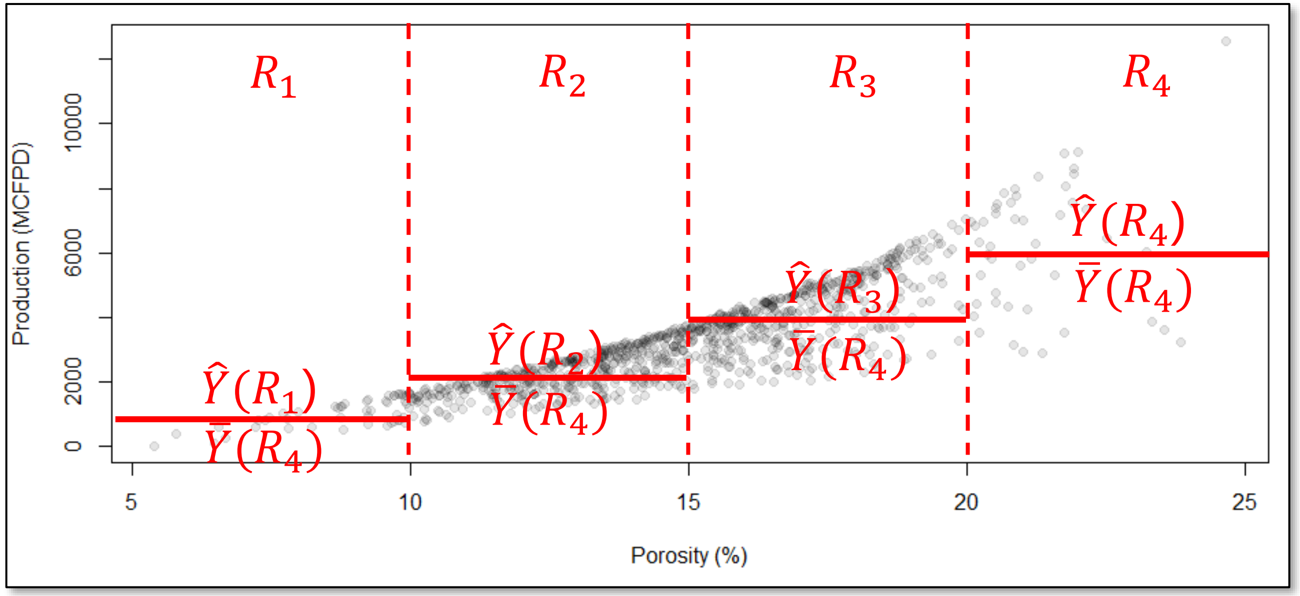
Predictor Feature Space Segmentation-based Prediction

The fundamental idea is to divide the predictor space, $X_1, \ldots, X_m$, into J mutually exclusive, exhaustive regions

- **mutually exclusive** – any combination of predictors only belongs to a single region, R_j
- **exhaustive** – all combinations of predictors belong a region, R_j, regions cover entire feature space (range of the variables being considered)

The same prediction in each region, mean of training data in region, $\hat{Y}(R_j) = \bar{Y}(R_j)$

For example, predict production, $\hat{Y}$, from porosity, X_1,



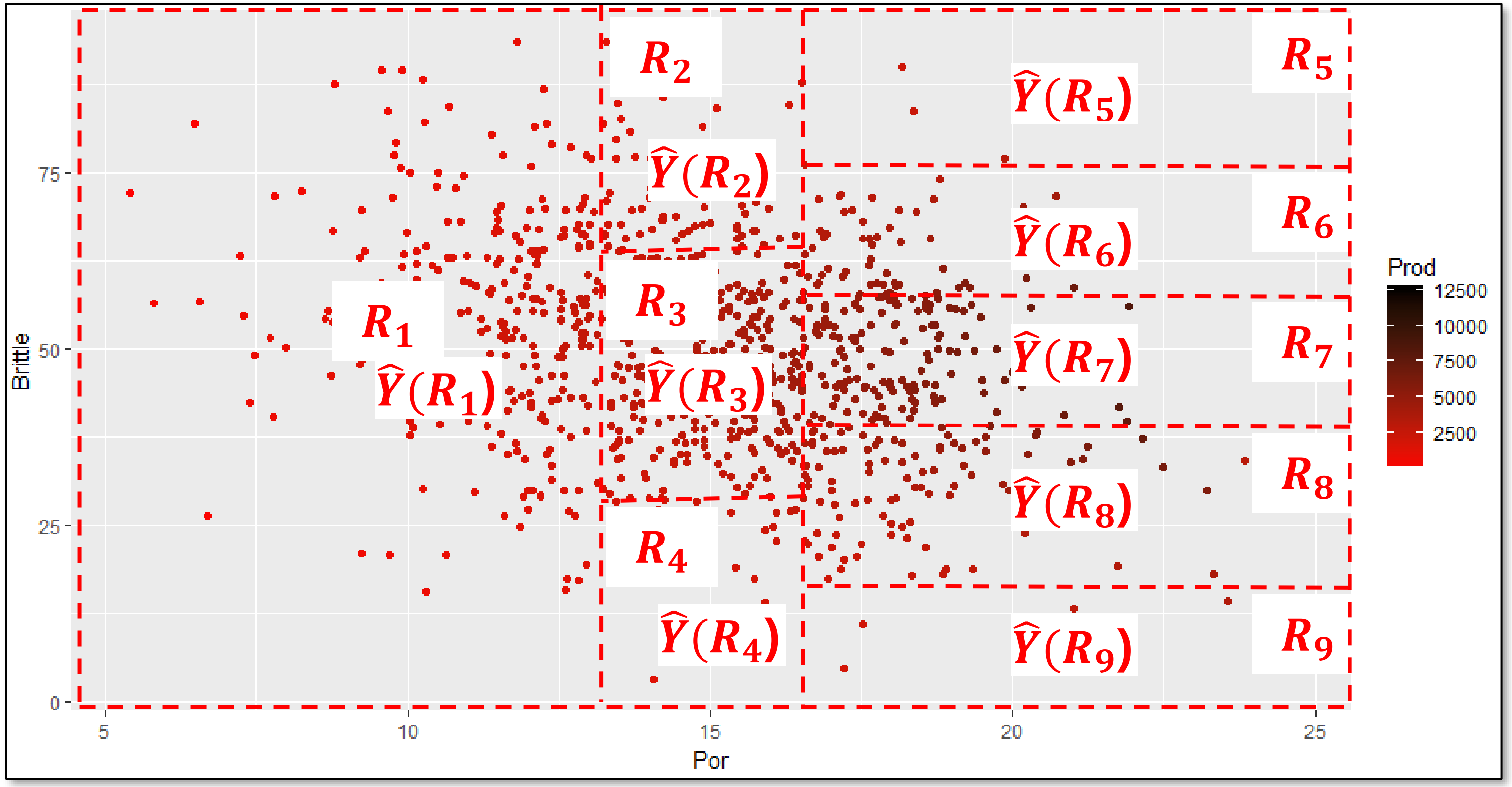
4 region decision tree with data and predictions by region, $R_j, j = 1, \ldots, J$.

Decision Tree

How do we construct regions, $R_1, R_2, \ldots, R_J$, for our predictions?

- They could be any shape!
- We decide to use high-dimensional cuboid → simple interpretation / rules

Hierarchical segmentation over the features – somewhat **flexible, compact model!**



Predict production from porosity and brittleness with 9 cuboid regions with data and predictions by region, $R_j, j = 1, \ldots, J$.

Decision Trees Example

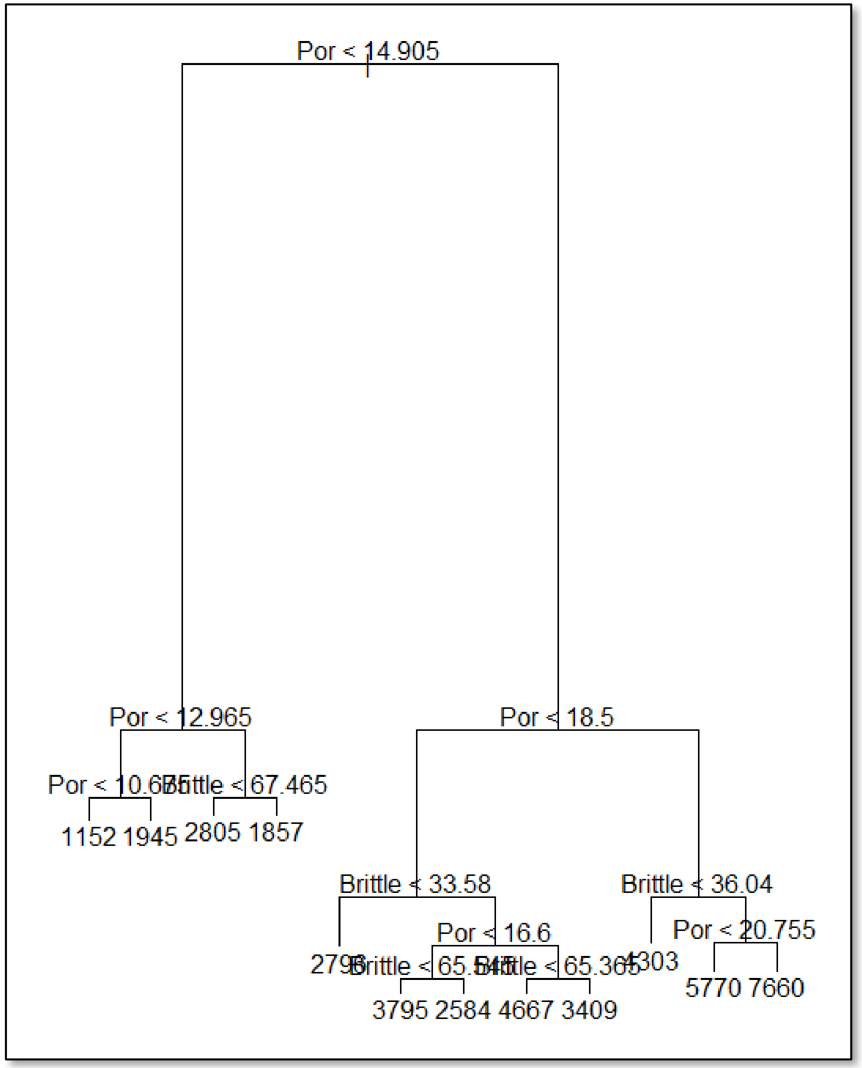
Build the initial reasonably complicated tree,

- first choice is porosity < or > 14.9%

- we get to the 3rd decision before brittleness is considered

Length of the branches is proportional to decrease in model error.

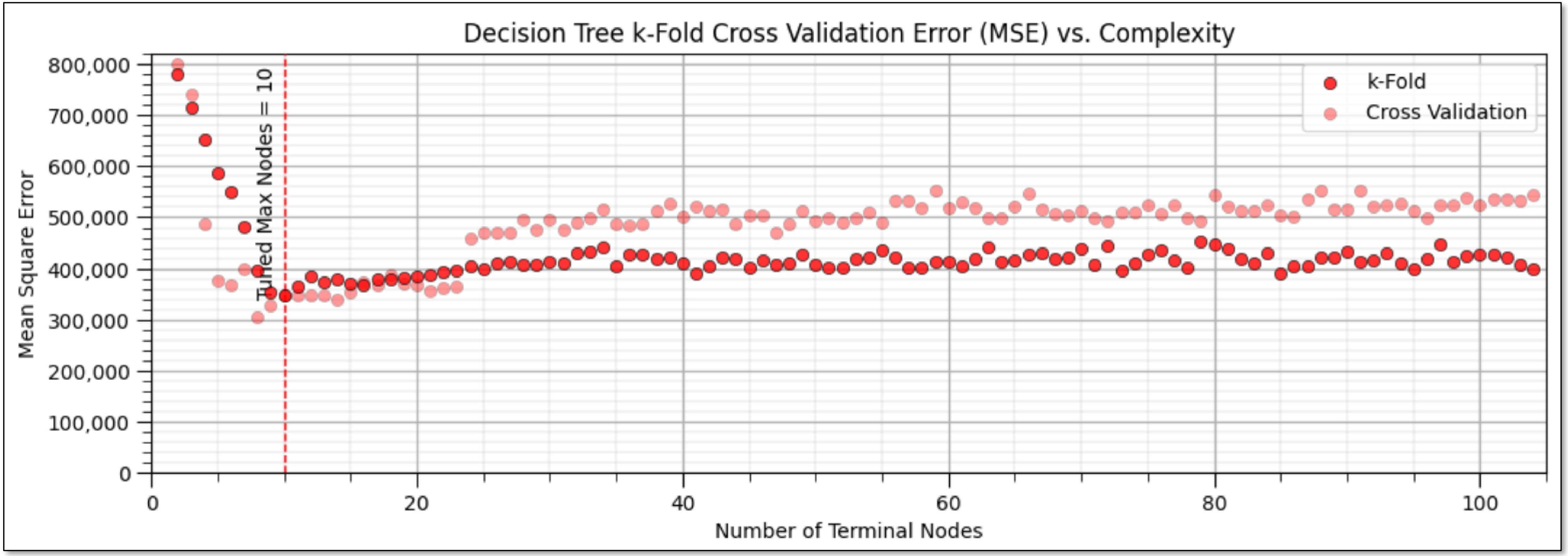
- decrease in RSS of the model for regression tree



Decision tree plot from Rstats, length of branches is proportional to the decrease in model error.

Decision Trees Example

Hyperparameter Tuning

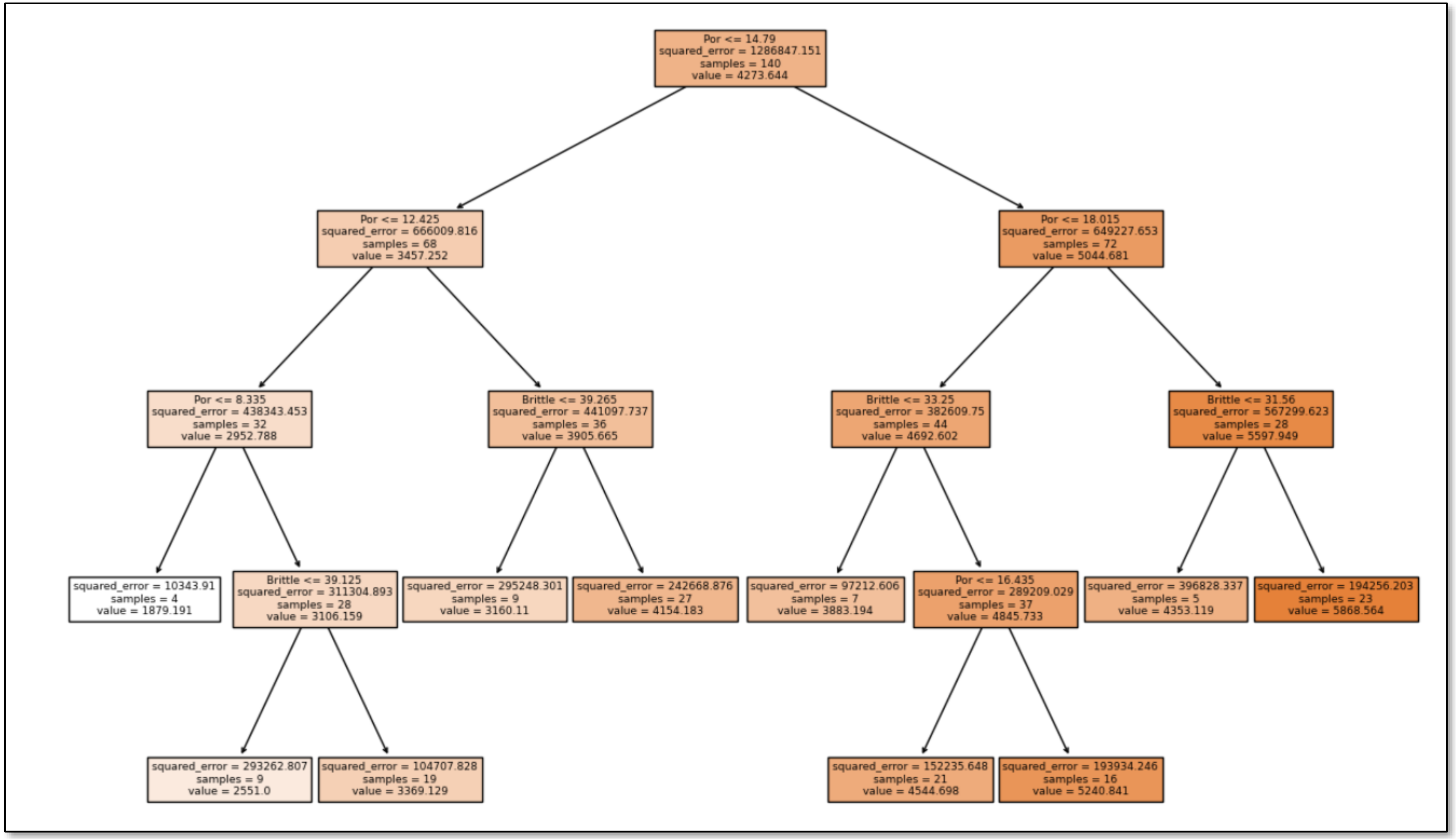


Decision tree hyperparameter tuning, simple to complicated with number of region hyperparameter, from Decision Tree chapter of Applied Machine Learning e-book.

Decision Trees Example

Hyperparameter Tuning

- tuned decision tree



Tuned decision tree model, from Decision Tree chapter of Applied Machine Learning e-book.

Lecture 15b: Gradient Boosting

Lecture outline:

- Boosting Methods

Boosting

General method that may be applied a many machine learning methods for regression and classification

- Build a model sequentially
- Next model is dependent on the previous models

The multiblade razor (here a 12 blade razor), 1st blade cuts, 2nd gets some of what the 1st blade missed, the 3rd blade gets some of what the 1st and 2nd blade missed, …, image from Image from https://thinkingoregon.files.wordpress.com/2015/10/15_blade_razor.png.

Recall Bagging Ensemble Models

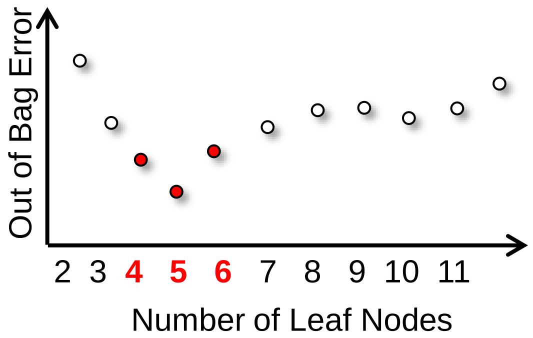
What is the bagging estimation model?

- Multiple models trained to different bootstrap data realizations, all with the same hyperparameter(s).

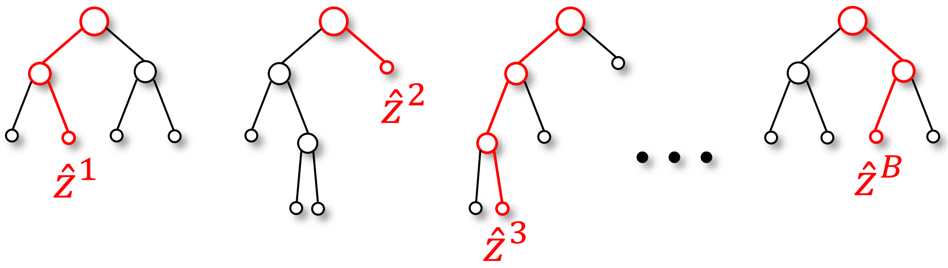
$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B}\left[\boxed{\hat{f}^1} + \boxed{\hat{f}^2} + \cdots + \boxed{\hat{f}^B}\right]$$

← bootstrap →

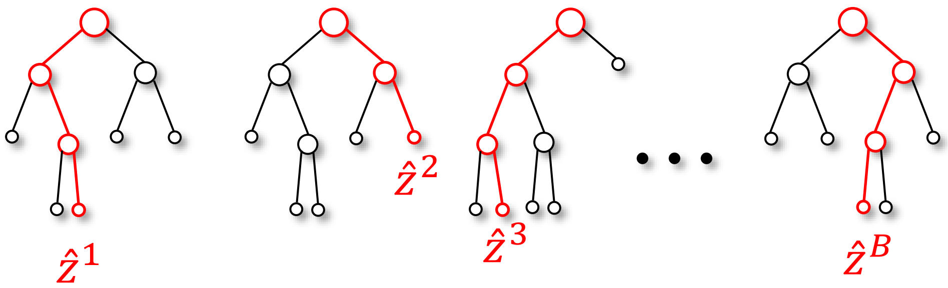
Bagging predictions by **averaging** multiple prediction models



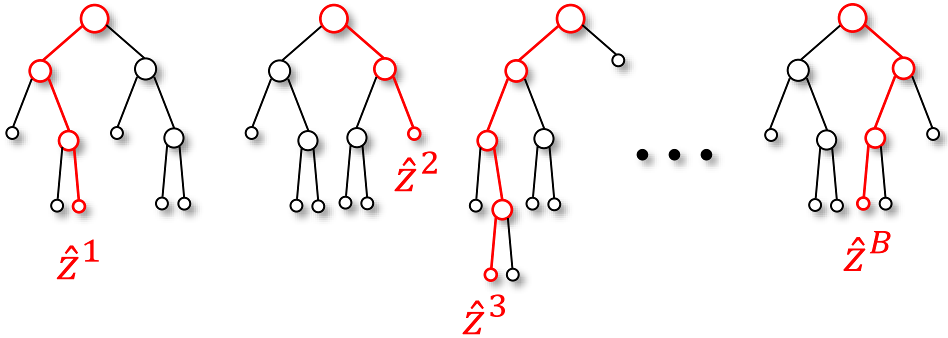
Hyperparameter – Number of Leaf Nodes = 4



Hyperparameter – Number of Leaf Nodes = 5



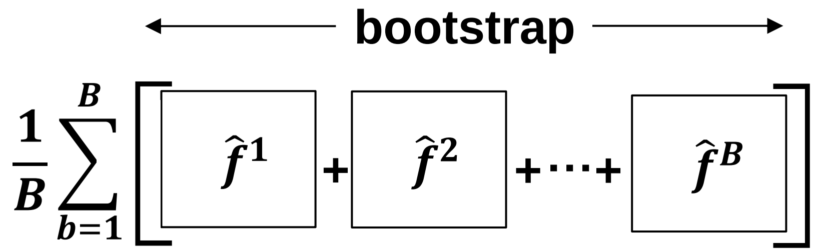
Hyperparameter – Number of Leaf Nodes = 6



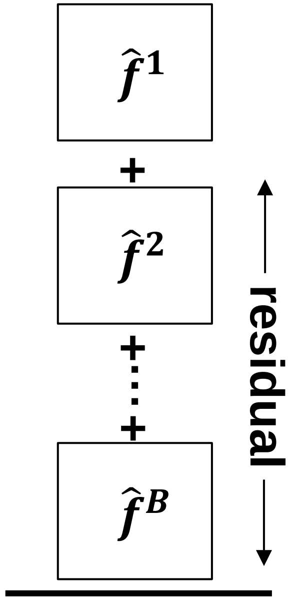
Boosting Models

Comparison Between Bagging and Boosting

- bagging uses bootstrap version of the data to build multiple estimators and averages the estimates to reduce model variance
- boosting uses a modified version, the error residual, of the data to 'slowly learn' to reduce model variance
- boosting does not use bootstrap!



Bagging predictions by averaging multiple prediction models



Boosting predictions by summing multiple prediction models

Boosting

The General Boosting Workflow:

Iterate

1. Build a ‘weak learner’ model for the training data

2. Calculate the residual (error) at the training data

3. Update the training data as the residual (error)

Weak Learner Definition

A Simple Model with High Error Rate, the 'Weak Learner'

- the prediction model performs only marginally better than random predicton!

$$\hat{Y} = \hat{F}_k(X_1, \dots, X_m)$$

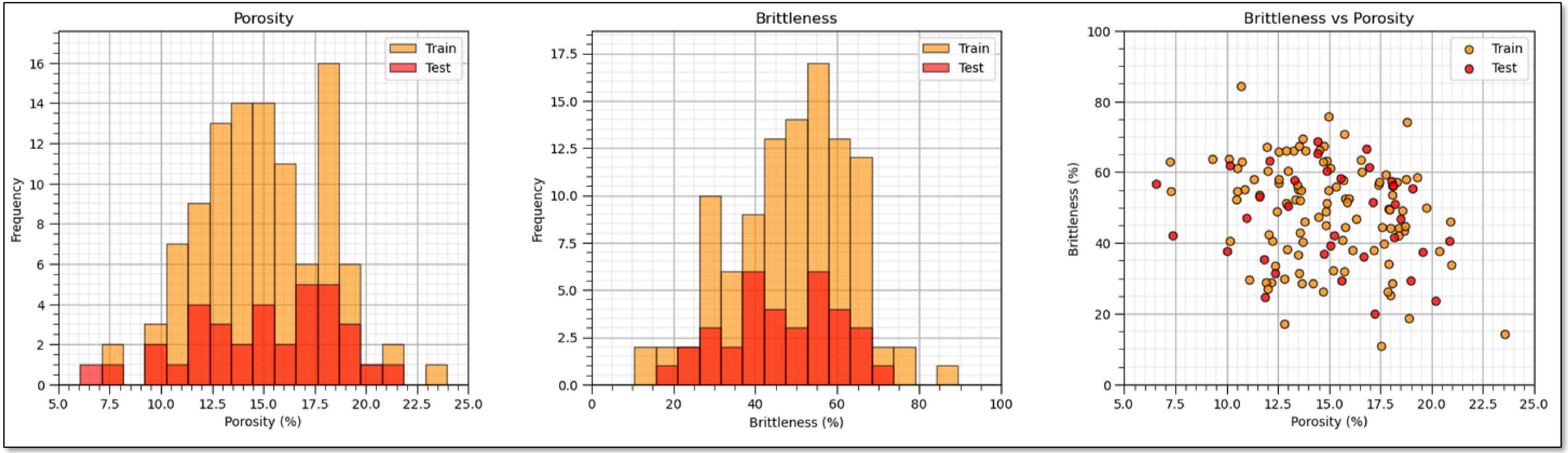
- $\hat{F}_k$ is the k^{th} weak learner
- $X_1, \dots, X_m$ are the predictor features
- $\hat{Y}$ is the prediction of the response feature

The term 'weak predictor' is often used, and specifically 'weak classifier' in the case of classification models.

Boosting Workflow

Train and Test Data Split

- unlike bootstrap-based, bagging methods (including random forest), boosting uses all the provided data to build each model
- **no out-of-bag samples** to perfrom out-of-bag cross validation as the modeling is being trained.
- we must perform standard cross validation for hyperparameter tuning



Example of training and testing data split, histograms (left and center) and scatter plot (right).

Boosting Workflow

Calculate the Residual with the Training Data

$$h_k(X_1, \dots, X_m) = Y - \hat{F}_k(X_1, \dots, X_m)$$

- $\hat{F}_k$ is a weak learner
- $X_1, \dots, X_m$ are the predictor features
- $\hat{Y}$ is the prediction of the response feature at the training data
- h_k is the k^{th} residual at the training data

Boosting Workflow

Fit a model to the residual / error from the first model

Another weak learner (building a model sequentially),

$$\hat{h}_k(X_1, \dots, X_m) = \hat{F}_{k+1}(h_k(X_1, \dots, X_m))$$

- $\hat{F}_{k+1}$ is a weak learner trained on the error of the previous model
- $X_1, \dots, X_m$ are the predictor features
- $\hat{Y}$ is the prediction of the response feature
- h_k is the k^{th} residual
- $\hat{h}_k$ is the estimated model of the k^{th} residual

Boosting Workflow

Calculate the Second Residual with the Training Data

$$h_{k+1}(X_1,\ldots,X_m) = h_k(X_1,\ldots,X_m) - \hat{F}_{k+1}(h_k(X_1,\ldots,X_m))$$

- $\hat{F}_{k+1}$ is a weak learner trained on the error of the previous model
- $X_1,\ldots,X_m$ are the predictor features
- $\hat{Y}$ is the prediction of the response feature at the training data
- h_k is the k^{th} residual at the training data

Fit a 3rd model to the 2nd residual, etc.,

$$\hat{h}_{k+1}(X_1,\ldots,X_m) = \hat{F}_{k+2}(h_{k+1}(X_1,\ldots,X_m))$$

Boosting Workflow

Calculate Residual, Model Residual Repeat - Results in an Additive Regression Model

$$\hat{Y} = \hat{F}(X_1, \ldots, X_m) = \sum_{k=1}^{K} \hat{F}_k\,(X_1, \ldots, X_m)$$

- for regression problems we just sum the estimator ensemble - additive ensemble models

In Geostatistics, Additive Trend and Residual Models are Common,

Commonly using multiple model type to best model the phenomenon,

- e.g., smooth polynomial regression to capture large scale trends and then kriging to capture the details of the residual to the trend - these methods often learn too fast, because each model is not a weak learner, i.e., NOT boosting

Tree-based Boosting Hyperparameters

Tree-based Boosting Hyperparameters,

1. **number of trees**, $\boldsymbol{K}$, in the additive model

- Unlike bagging and random forest, too large an ensemble of models (large $\boldsymbol{K}$) will eventually result in overfit
- Although, the slow learner is more resistant to overfit

2. **learning rate**, ρ, controls the rate of updating with each new model

- slow down learning for a more robust model, balanced to ensure good performance, too small rate will require very large number ot trees

3. **complexity of the individual trees**, e.g., d, number of levels

- $d = 1$, stumps, addition of simple models without feature interactions (one feature at a time)
- d is known as interaction depth, max. # features in each estimator
- modern XGBoost uses 3-10 levels and relies on limiting learning rate to manage overfit

Tree-based Boosting Hyperparameters

Tree-based Boosting Hyperparameter Tuning:

- Unlike bagging trees and random forest there is no bootstrap and out-of-bag samples

Hyperparameter tuning requires the previous discussed cross validation approaches, for example,

- hold out cross validation
- k-fold cross validation

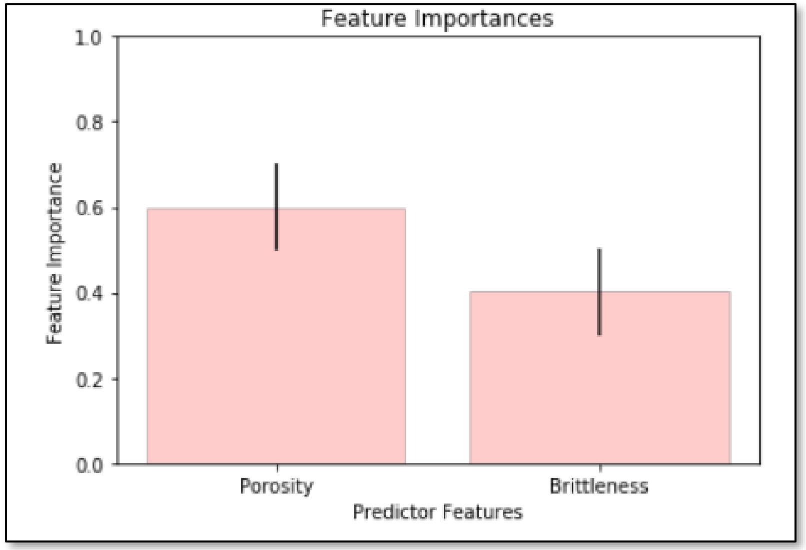
Tree-based Boosting Feature Importance

Tree-based Boosting Feature Importance,

- During ensemble model construction, track the RSS reduction with each split for each feature in each tree.

- Average feature importance across all decision trees in the ensemble (scikit-learn)

- Force the sum over the predictor features to 1.0 (L1 normalizer).



Feature importance with 95% confidence interval indicated.

Tree-based Boosting Guidance

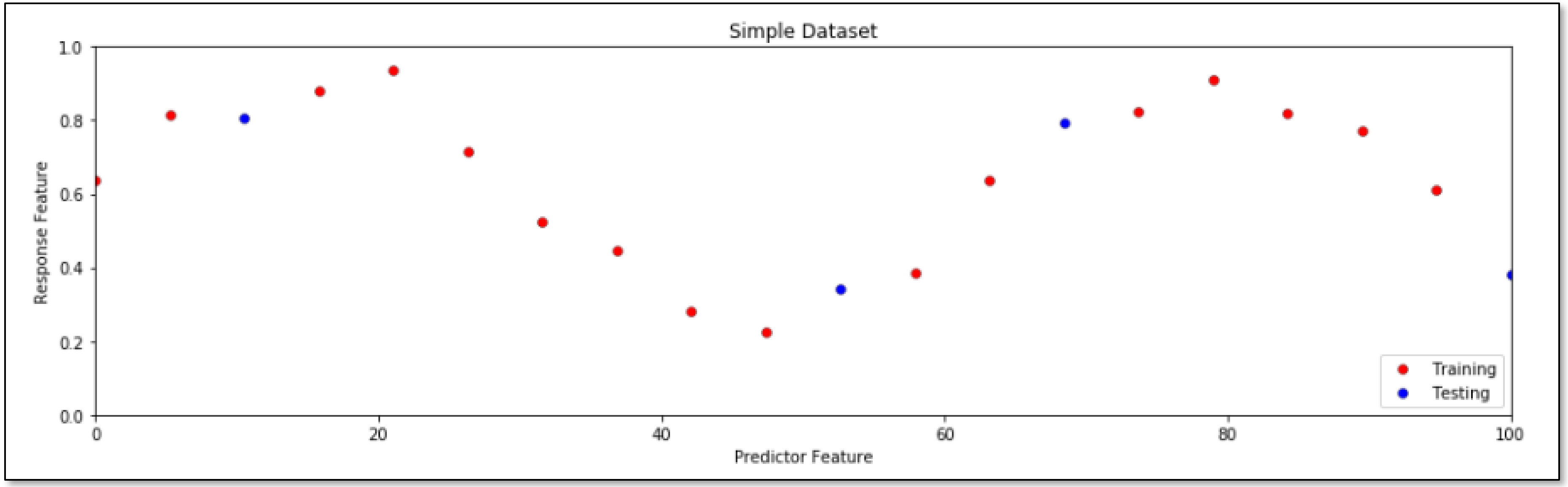
Tree-based Boosting Guidance

- Slow learning, more robust to overfit, but overfit is still possible with too many estimators, $\boldsymbol{K}$.
- Cross validation needed to select the number of estimators, $\boldsymbol{K}$, learning rate, ρ, and the complexity of the individual estimators, d, number of levels.
- Smaller trees, d, are often sufficient due to the sequential learning and provide a more interpretable model

Boosting Illustration

Let's Take a Simple Model

- 1 predictor feature and 1 response feature
- some irregularity and highly non-linear



Simple 1 predictor and 1 response training and testing data.

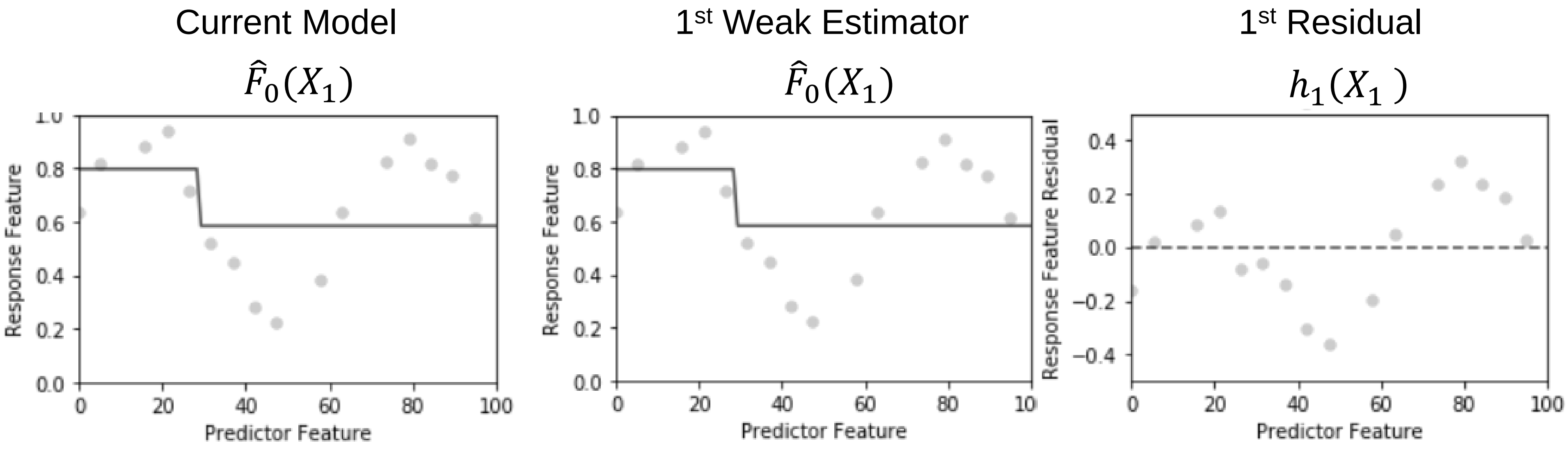
- we will use decision stumps for ease of visualization / interrogation (trees with only 1 decision – 1 level trees).

Boosting Illustration

Step 1: Fit Weak Predictor $\hat{F}_0(X_1, \ldots, X_m)$

- a decision stump (tree with 1 level) with one decision and 2 regions

Step 2: Calculate the Residual $h_0(X_1, \ldots, X_m)$



First weak predictor and residual and boosting model.

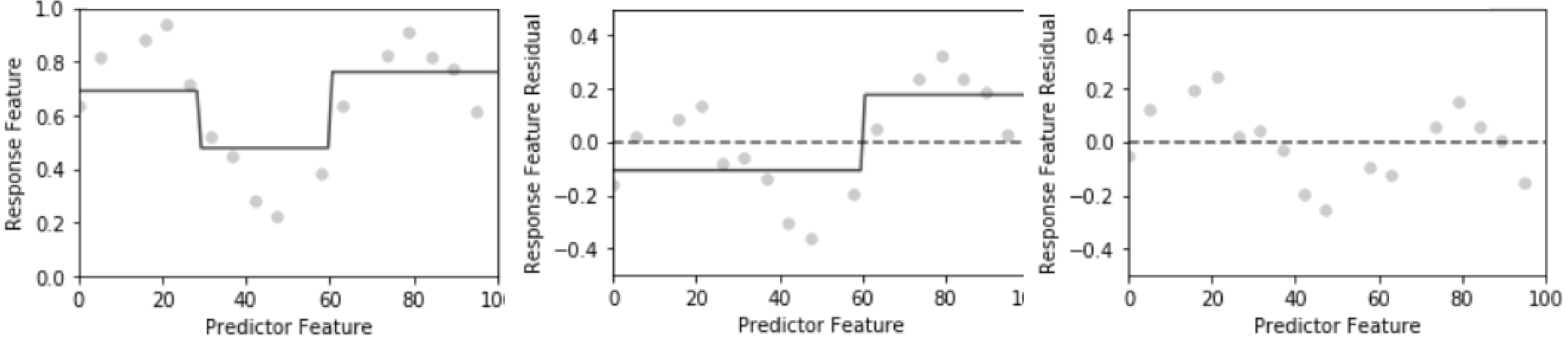
Boosting Illustration

Step 3: Fit Weak Predictor $\hat{F}_1(X_1, \ldots, X_m)$

- a decision stump with one decision and 2 regions

Step 4: Calculate the Residual $h_1(X_1, \ldots, X_m)$

Current Model	2^{nd} Weak Estimator	2^{nd} Residual
$\hat{F}_1(X_1) + \hat{F}_2(X_1)$	$\hat{F}_2(X_1)$	$h_2(X_1)$



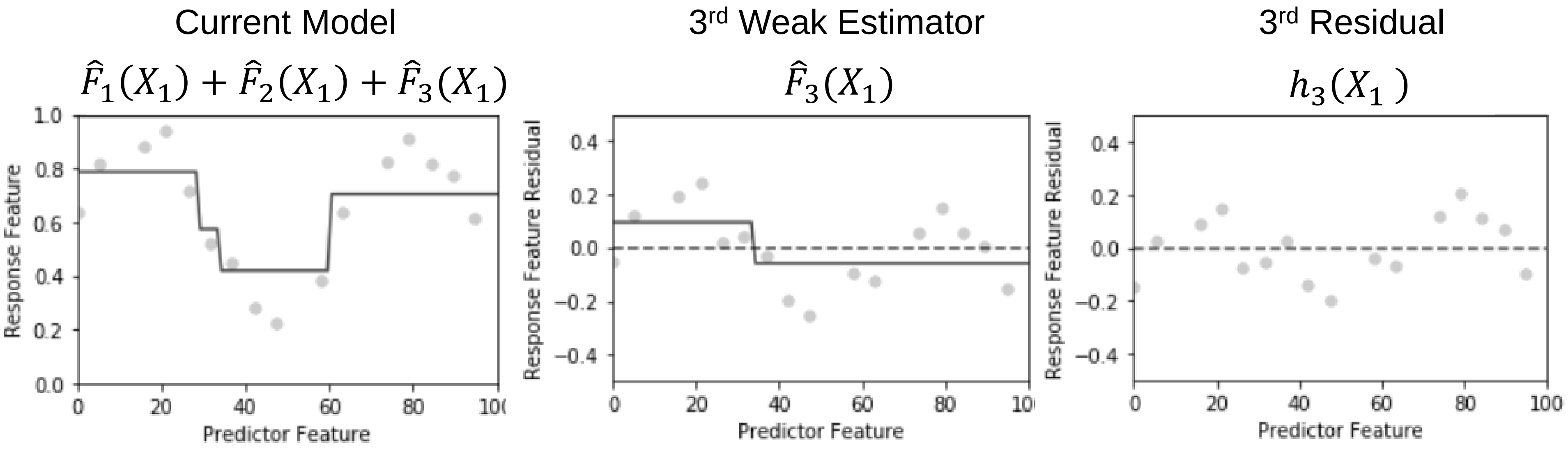
Second weak predictor and residual and boosting model.

Boosting Illustration

Step 5: Fit Weak Predictor $\hat{F}_3(X_1, \ldots, X_m)$

- a decision stump with one decision and 2 regions

Step 6: Calculate the Residual $h_3(X_1, \ldots, X_m)$



Third weak predictor and residual and boosting model.

Boosting Illustration

Step 7: Fit Weak Predictor $\hat{F}_4(X_1, \ldots, X_m)$

- a decision stump with one decision and 2 regions

Step 8: Calculate the Residual $h_4(X_1, \ldots, X_m)$

Current Model

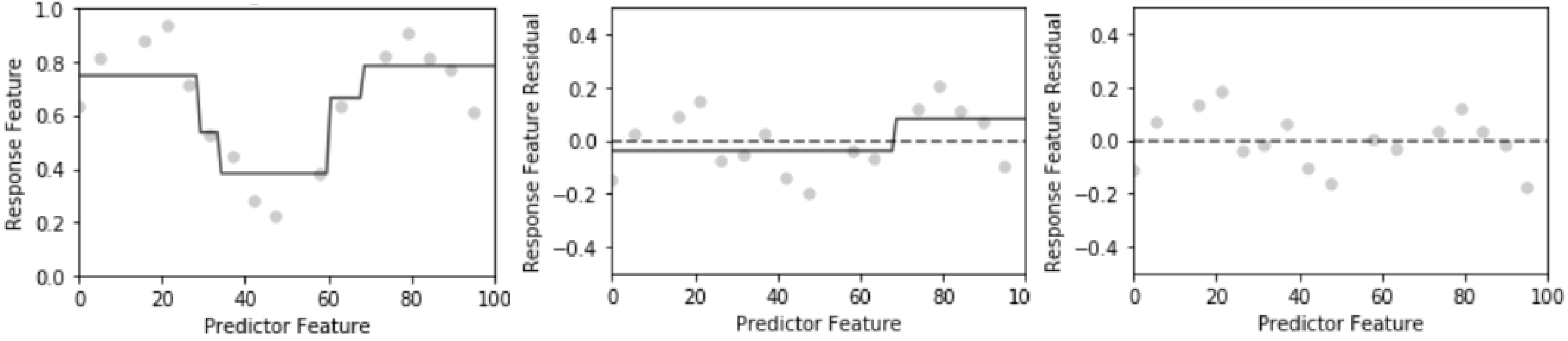
$\hat{F}_1(X_1) + \hat{F}_2(X_1) + \hat{F}_3(X_1) + \hat{F}_4(X_1)$

4th Weak Estimator

$\hat{F}_4(X_1)$

4th Residual

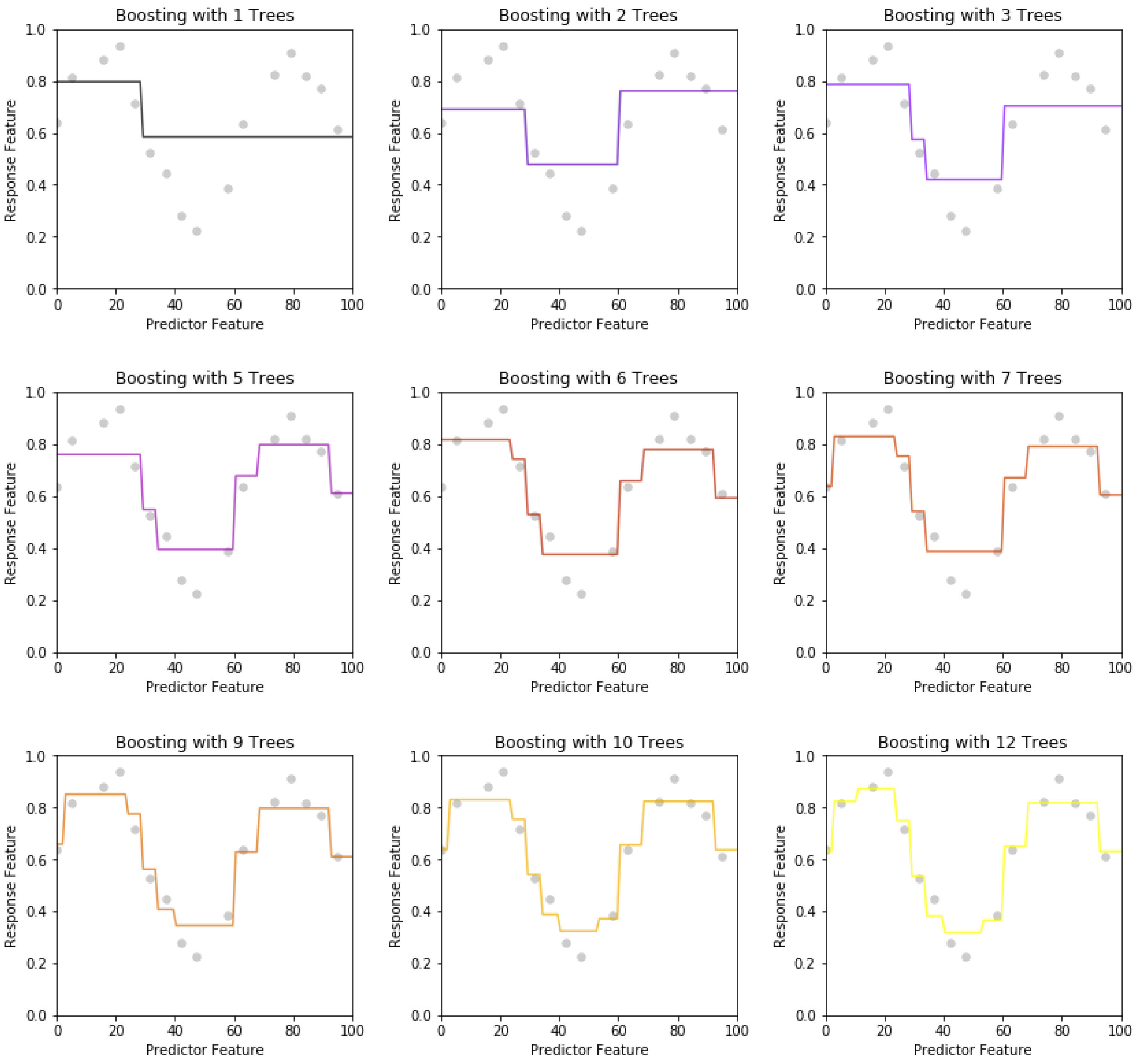
$h_4(X_1)$



Fourth weak predictor and residual and boosting model.

Boosting Illustration

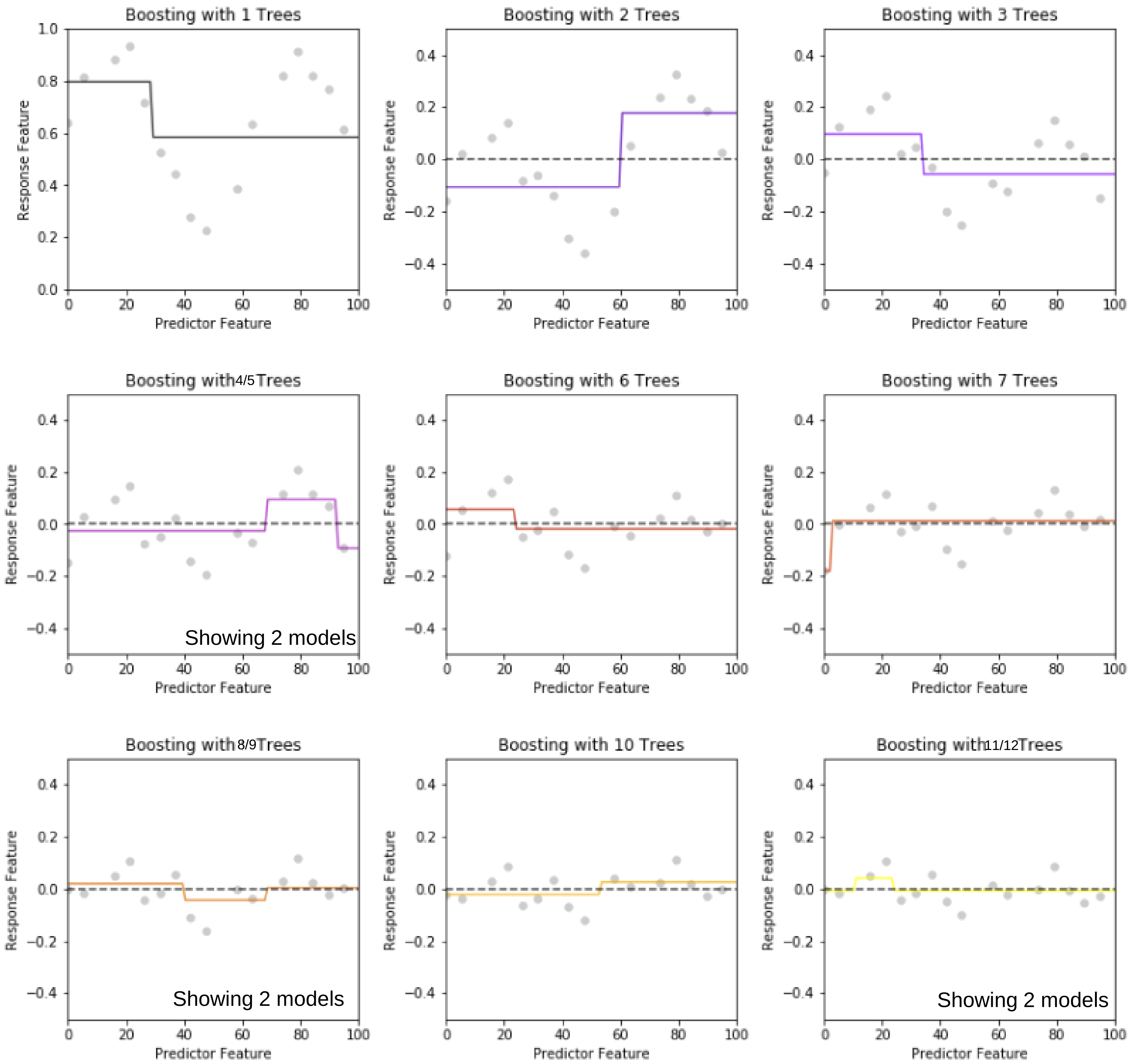
Evolving Model with Additional Estimators



Gradient Boosting chapter of the Applied Machine Learning in Python e-book.

Boosting Illustration

Residuals and Models



Gradient Boosting chapter of the Applied Machine Learning in Python e-book.

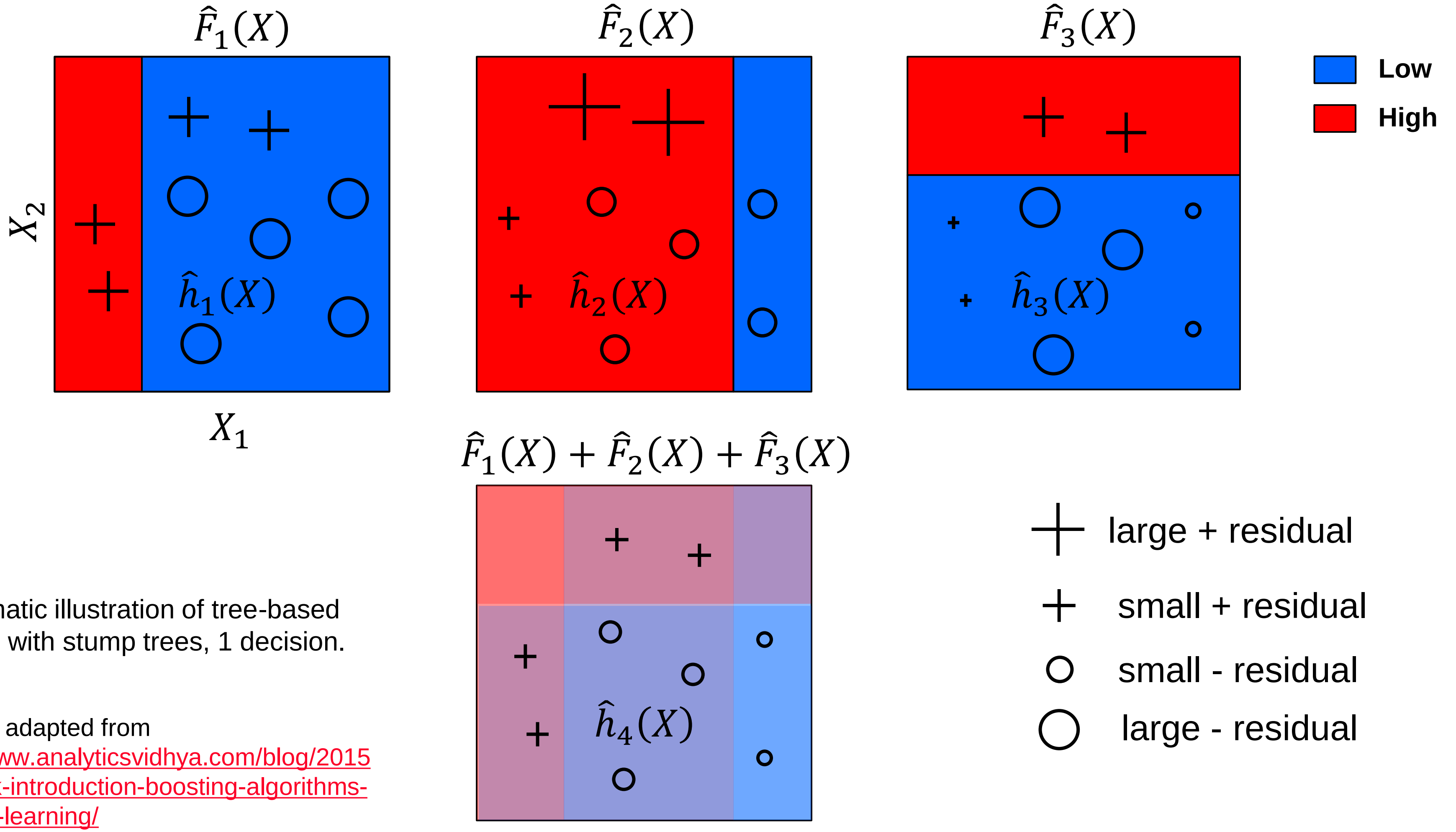
Boosting

Boosting is Another Strategy to Improve Model Accuracy While Avoiding Overfit

- a complicated model will fit the data with potential for overfit
- a boosting model learns slowly, avoids overfit
- it is a form of sequential learning, let’s show another (schematic) example

Boosting

Here another illustration demonstrating the concept of sequential learning.



Schmematic illustration of tree-based boosting with stump trees, 1 decision.

Example adapted from https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

PGE 383 Subsurface Machine Learning

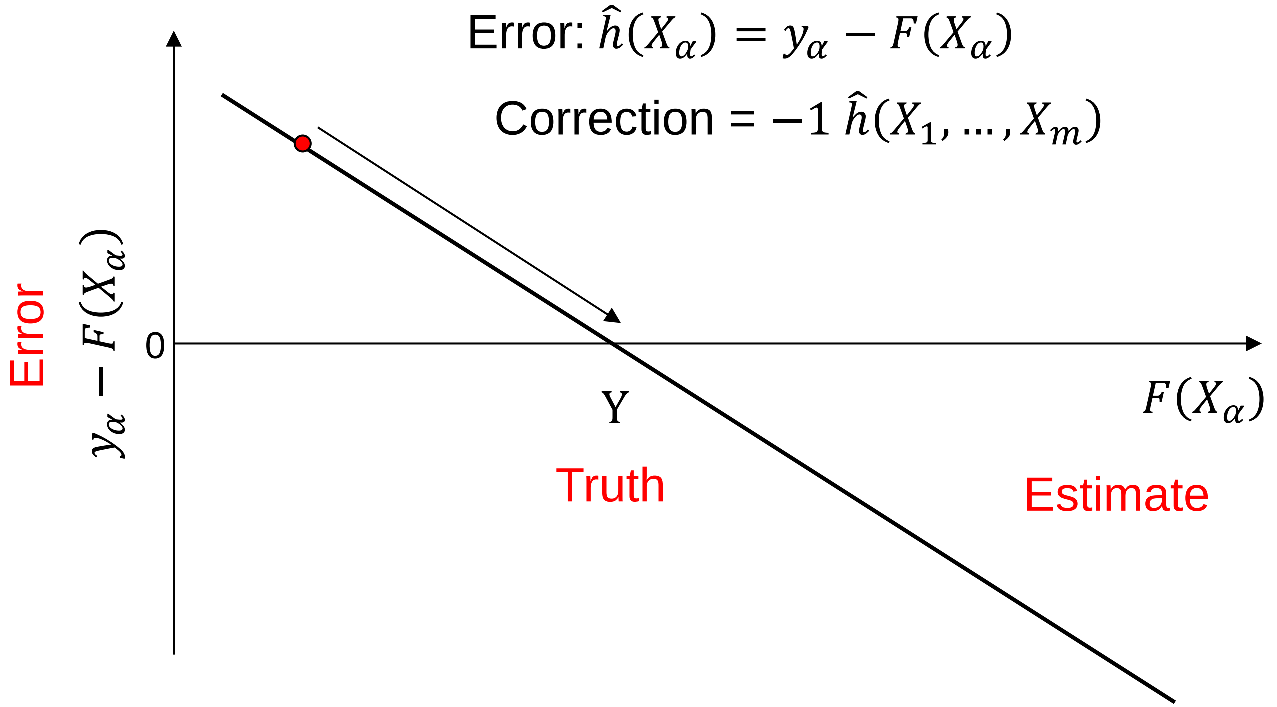
Lecture 15b: Gradient Boosting

Lecture outline:

- Tree-based Gradient Boosting Regression

Iterative Model Construction

For Each Model $\widehat{F}_k$ We Can Calculate the Error at the Training Data, $\alpha = 1, \dots, n$.



Then we correct the model by fitting the error with a new additive model.

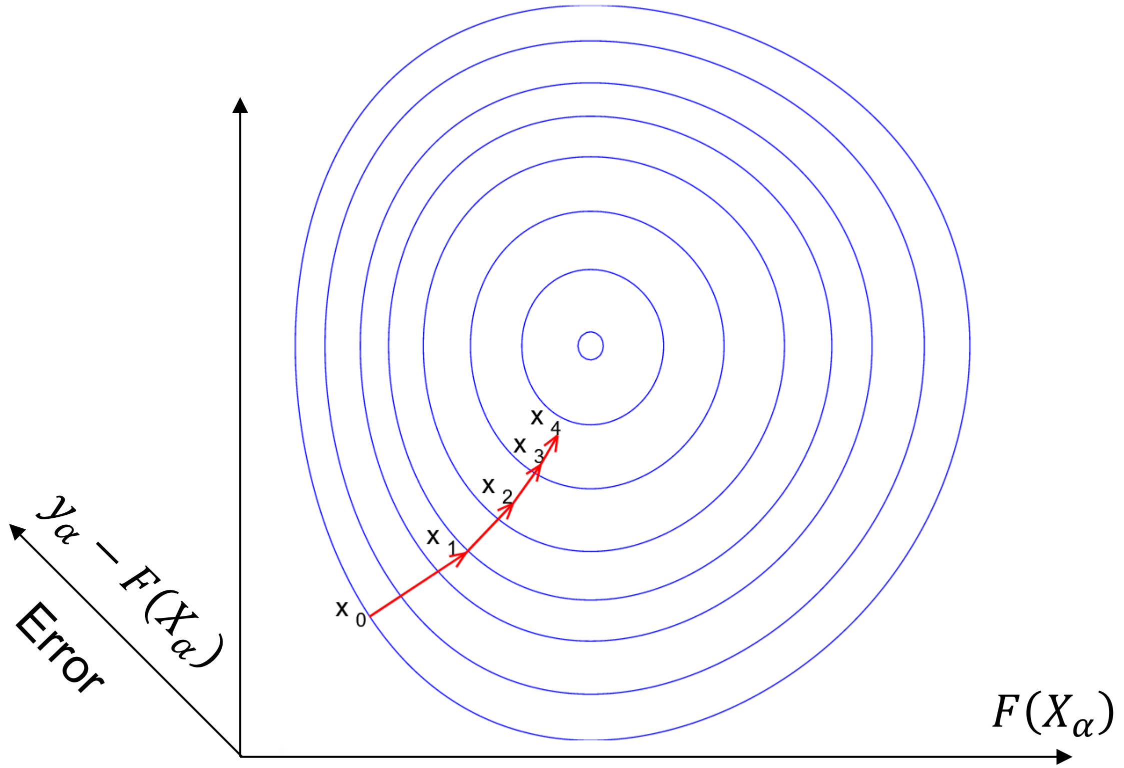
- Let's be more rigorous and pose as gradient descent optimization, this is gradient boosting

Gradient Descent Optimization

Iterative Solution Scheme to Find the Minimum of a Function.

- to find a local minimum take steps proportional to the negative of the gradient at each point

- Iteratively improving the prediction model



See the lecture on LASSO for more information on optimization.

Image modified from: https://upload.wikimedia.org/wikipedia/commons/f/ff/Gradient_descent.svg

Gradient Descent Optimization

Gradient Descent Workflow,

1. Evaluate the gradient of the loss function at state, $m - 1$

$$\alpha = 1, \ldots, n \quad \text{training data}$$

$$g_{m,\alpha} = \left[\frac{\partial L\left(y_\alpha, F(X_\alpha)\right)}{\partial F(X_\alpha)} \right]$$

2. Determine step length, ρ_m

3. Update the current solution, f_m

$$f_m = f_{m-1} - \rho_m \frac{\partial L\left(y_\alpha, F(X_\alpha)\right)}{\partial F(X_\alpha)}$$

$$f_m = f_{m-1} - \rho_m g_m$$

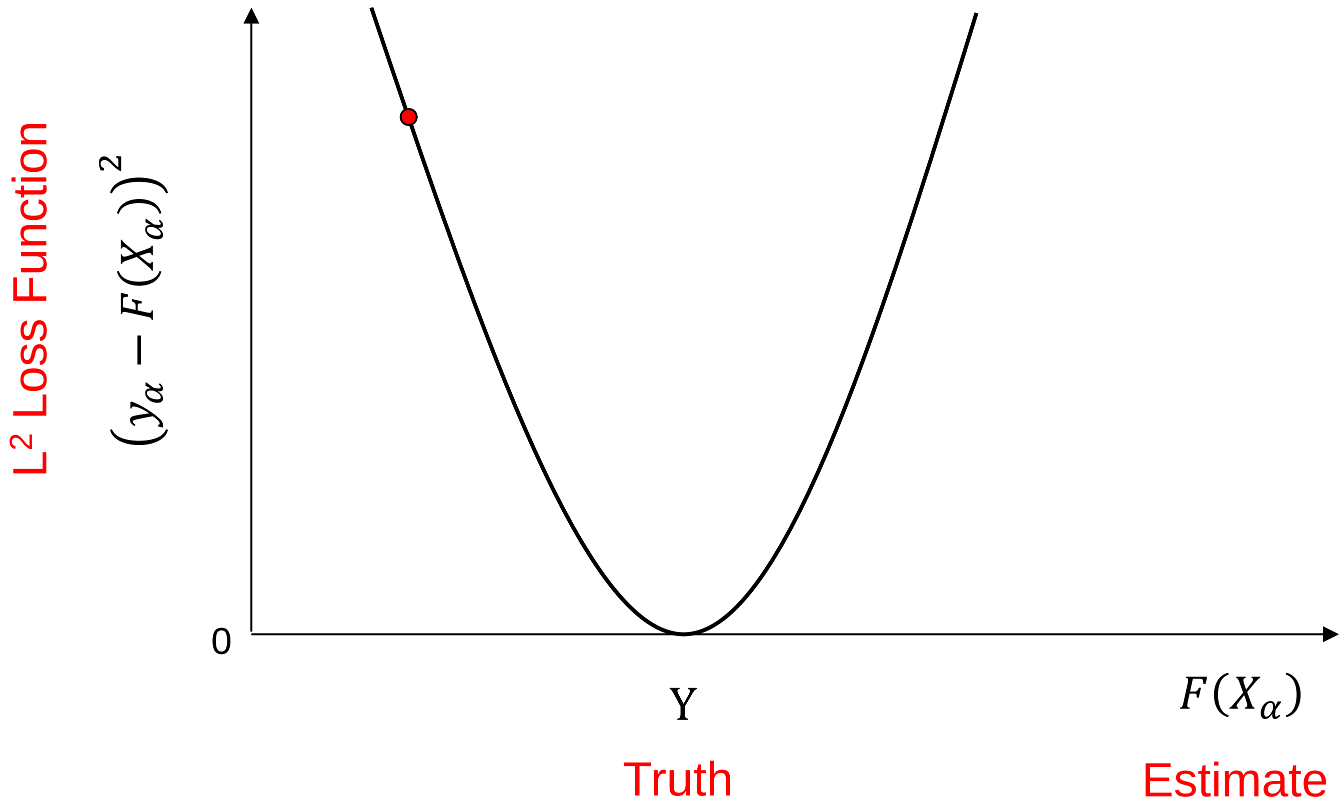
where g_m integrates the error over all training data, $\alpha = 1, \ldots, n$

4. Repeat.

Gradient Descent Optimization

First, Assign a Loss Function, $L\left(y_\alpha, F(X_\alpha)\right)$

Convert error to a loss function, we will assume L^2 norm.



Gradient Descent Optimization

Calculate the Gradient of this Loss Function, $\boldsymbol{L(\,)}$ at the Training Data, $\boldsymbol{\alpha = 1, \ldots, n}$.

- For a regression model with a L^2 norm:

$$L\big(y_\alpha, F(X_\alpha)\big) = \frac{1}{2}\big(y_\alpha - F(X_\alpha)\big)^2$$

The gradient is:

$$\frac{-\partial L\big(y_\alpha, F(X_\alpha)\big)}{\partial F(X_\alpha)} = y_\alpha - F(X_\alpha)$$

Therefore, any model that attempts to fit (add) the error at the training data locations is a step along the steepest descent!

Gradient Descent Optimization

We Could Have Assumed a Variety of Other Loss Functions,

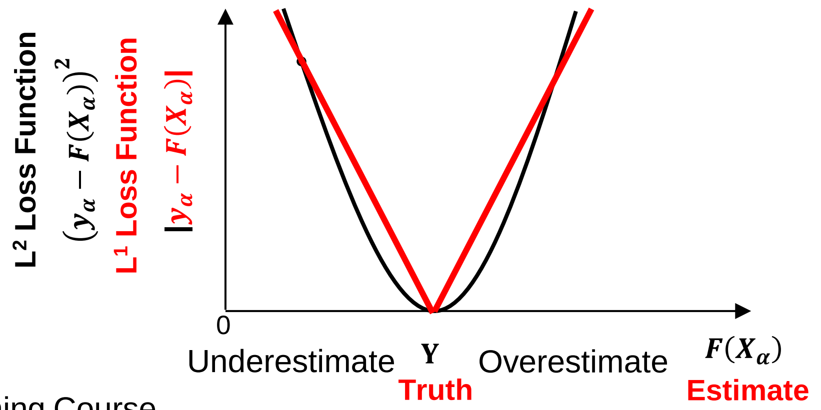
- For regression with a loss function based on a L^1 norm:

$$L\left(y_\alpha, F(X_\alpha)\right) = |y_\alpha - F(X_\alpha)|$$

The gradient is:

$$\frac{-\partial L\left(y_\alpha, F(X_\alpha)\right)}{\partial F(X_\alpha)} = sign[y_\alpha - F(X_\alpha)]$$

For the L^1 norm the rate of descent is only determined by the learning rate!



Gradient Descent Optimization

The Boosting Model Now Becomes Standard Gradient Descent Optimization

Based on our norm we calculate the loss function and its gradient. For the L^2 norm:

$$\frac{-\partial L\big(y_\alpha, F(X_\alpha)\big)}{\partial F(X_\alpha)} = y_\alpha - F(X_\alpha)$$

For the L^2 norm, the gradient of the loss function at the data locations relative to the model is the model error!

We sequentially update the model based on the negative of the gradient from the loss function.

$$F_{k+1}(X_\alpha) = F_k(X_\alpha) - \frac{\partial L\big(y_\alpha, F(X_\alpha)\big)}{\partial F(X_\alpha)}$$

Gradient Descent Optimization

We Often Getter a Better Result if We Slow the Rate of Change of the Model By Restricting The Learning Rate, $\boldsymbol{\rho}$.

$$F_{k+1}(X_\alpha) = F_k(X_\alpha) - \rho \frac{\partial L\big(y_\alpha, F(X_\alpha)\big)}{\partial F(X_\alpha)}$$

The learning rate constrains the size of the steps towards the solution.

- varies by problem, values of 0.01 - 0.1 are commonly used
- with high learning rate we risk overshooting the best solution
- with low learning rate the solution takes longer

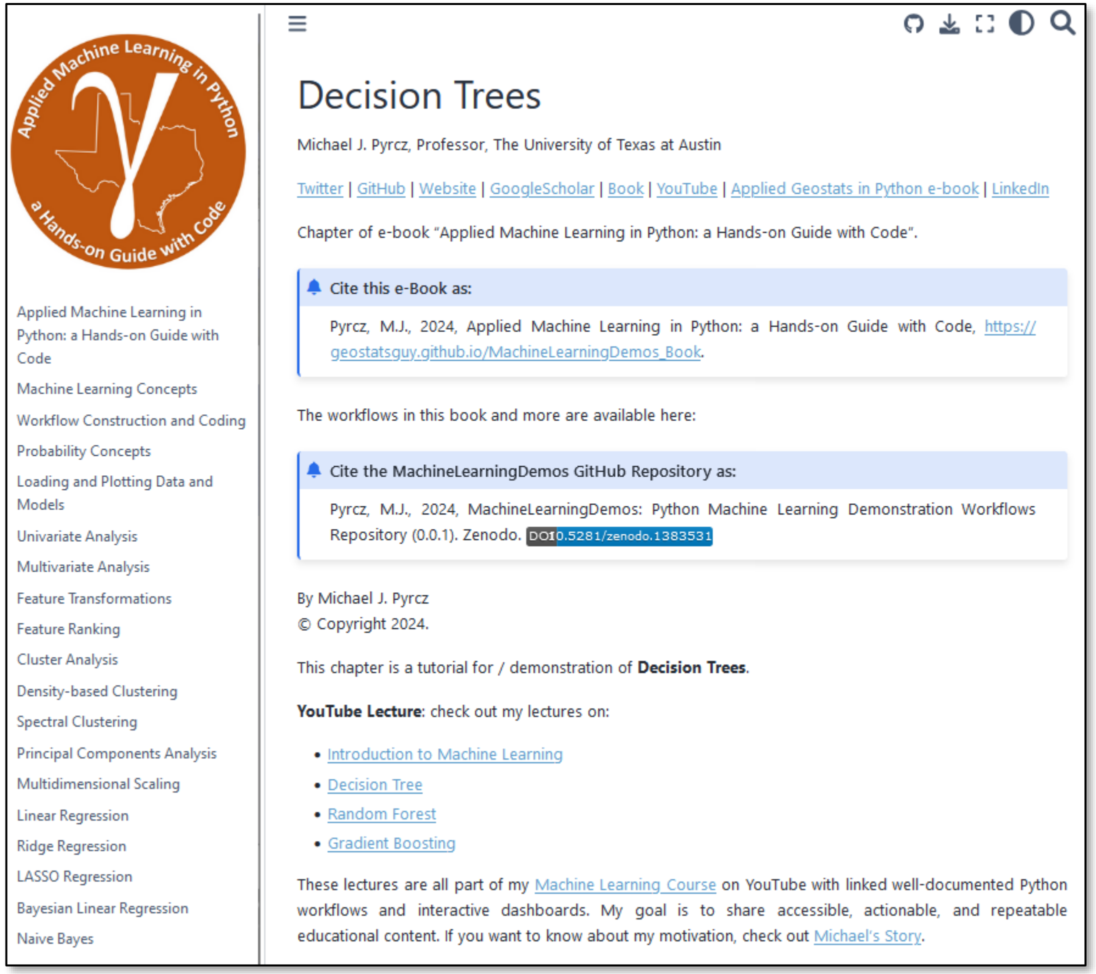
Lecture 15b: Gradient Boosting

Lecture outline:

- **Tree-based Gradient Boosting Methods Hands-on**

Gradient Boosting Demonstration in Python

Demonstration of gradient boosting with a well-documented workflow.



Applied Machine Learning in Python a Hands-on Guide with Code

Applied Machine Learning in Python: a Hands-on Guide with Code
Machine Learning Concepts
Workflow Construction and Coding
Probability Concepts
Loading and Plotting Data and Models
Univariate Analysis
Multivariate Analysis
Feature Transformations
Feature Ranking
Cluster Analysis
Density-based Clustering
Spectral Clustering
Principal Components Analysis
Multidimensional Scaling
Linear Regression
Ridge Regression
LASSO Regression
Bayesian Linear Regression
Naive Bayes

Decision Trees

Michael J. Pyrcz, Professor, The University of Texas at Austin

Twitter | GitHub | Website | GoogleScholar | Book | YouTube | Applied Geostats in Python e-book | LinkedIn

Chapter of e-book "Applied Machine Learning in Python: a Hands-on Guide with Code".

Cite this e-Book as:

Pyrcz, M.J., 2024, Applied Machine Learning in Python: a Hands-on Guide with Code, https://geostatsguy.github.io/MachineLearningDemos_Book.

The workflows in this book and more are available here:

Cite the MachineLearningDemos GitHub Repository as:

Pyrcz, M.J., 2024, MachineLearningDemos: Python Machine Learning Demonstration Workflows Repository (0.0.1). Zenodo. DOI 10.5281/zenodo.1383531

By Michael J. Pyrcz
© Copyright 2024.

This chapter is a tutorial for / demonstration of **Decision Trees**.

YouTube Lecture: check out my lectures on:

- Introduction to Machine Learning
- Decision Tree
- Random Forest
- Gradient Boosting

These lectures are all part of my Machine Learning Course on YouTube with linked well-documented Python workflows and interactive dashboards. My goal is to share accessible, actionable, and repeatable educational content. If you want to know about my motivation, check out Michael's Story.

Gradient Boosting chapter of Applied Machine Learning in Python e-book.

PGE 383 Subsurface Machine Learning

Lecture 15b: Gradient Boosting

Lecture outline:

- **Decision Tree Review**
- **Boosting Methods**
- **Tree-based Gradient Boosting Regression**
- **Tree-based Gradient Boosting Methods Hands-on**