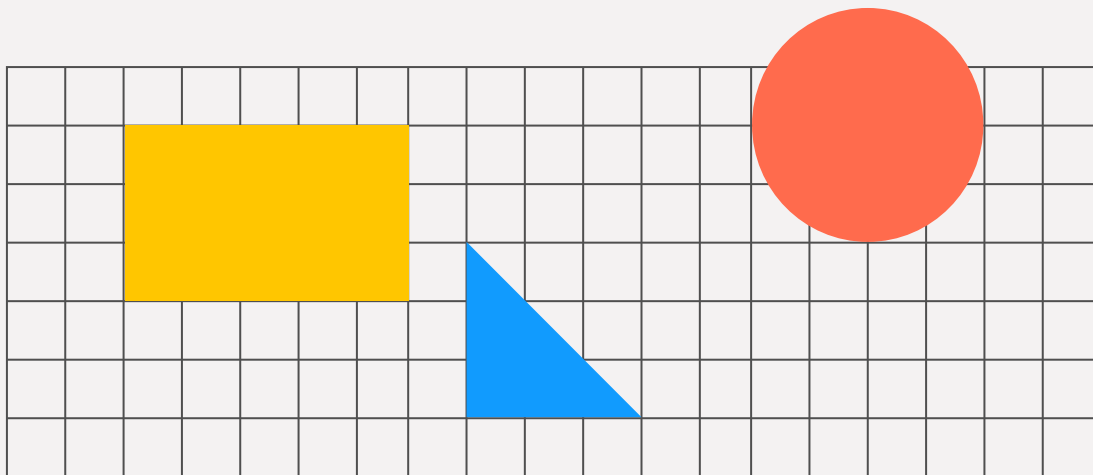
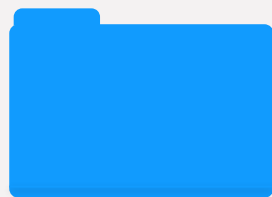


Stochastic Gradient Boosting

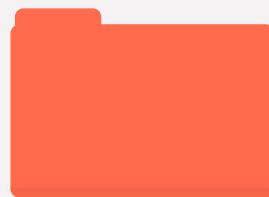




Introduction



The Math



Numerical
Techniques



Comparison

Introduction

Our goal is to find a function that maps X to Y such that the loss function is minimized over the distribution D .

Boosting algorithms solve this problem by find such a function as the form of additive sum of some base learners:

$$F(\mathbf{x}) = \sum_{m=0}^M \beta_m h(\mathbf{x}; \mathbf{a}_m)$$

Expansion coefficient

Base learner

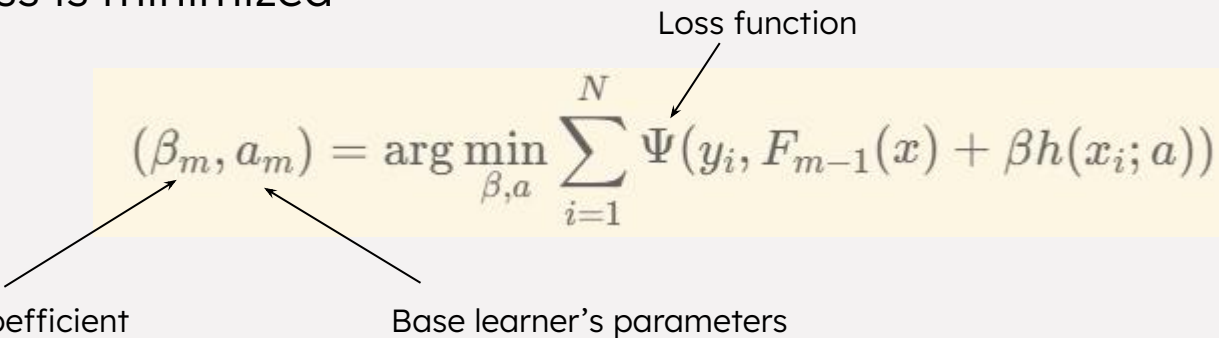
Base learner's parameters

Boosting

We start with an initial guess F_0 , then we fit the m 'th base learner and its expansion coefficient in a stage-wise manner.

$$F_m(x) = F_{m-1}(x) + \beta_m h(x; a_m)$$

At each stage, we find the base learner and its expansion coefficient such that the loss is minimized

$$(\beta_m, a_m) = \arg \min_{\beta, a} \sum_{i=1}^N \Psi(y_i, F_{m-1}(x) + \beta h(x_i; a))$$


The diagram illustrates the minimization of the loss function. The equation is shown in a yellow box. An arrow labeled "Loss function" points to the Ψ term in the summation. Two arrows point from the text labels "Expansion coefficient" and "Base learner's parameters" to the β_m and a_m terms in the $\arg \min$ expression, respectively.

Expansion coefficient

Base learner's parameters

Gradient Boosting

For each stage $m=1,\dots,M$, we fit the base learner and its expansion coefficients in two steps.

A.K.A. current pseudo residual

First step, we compute the negative gradient of the loss function w.r.t. the current predictor, and fit the base learner to it (with least squares)

$$a_m = \arg \min_{a,p} \sum_{i=1}^N [\tilde{y}_{im} - ph(x_i; a)]^2$$

$$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

Second step, given the base learner, we find the optimal value of the expansion coefficient of it (to minimize the loss w.r.t. True values and new predicted values with the base learner)

$$\beta_m = \arg \min_{\beta} \sum_{i=1}^N \Psi(y_i, F_{m-1}(x_i) + \beta h(x_i; a_m))$$

Gradient Tree Boosting

With regression tree as a base learner, we can specialize the former approach as such:

At each stage $m=1,\dots,M$:

First step, fit a regression tree to the current pseudo residual

$$h(x; \{R_{lm}\}_1^L) = \sum_{l=1}^L \bar{y}_{lm} 1_{x \in R_{lm}}$$

l 'th region in m 'th tree

Constant value represents the mean of pseudo residuals in each region

which will partition the x -space into L -disjoint regions and predicts a constant value in each one region.

Gradient Tree Boosting cont.

Second step, with the l 'th regions of the m 'th tree defined, we can separately solve for the expansion coefficient in each region:

$$\gamma_{lm} = \arg \min_{\gamma} \sum_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(x_i) + \gamma)$$

The current function (estimator) is then updated separately in each region:

$$F_m(x) = F_{m-1}(x) + lr \cdot \gamma_{lm} 1_{x \in R_{lm}}$$

Learning rate

Gradient Tree Pseudo code and weak learner

How gradient tree boosting works

Initialize

$F_m = \text{mean}(Y)$ # Start with the mean of Y

For each boosting iteration

for m in range($n_estimators$):

 # Sample data

$X_batch, Y_batch = \text{random_sample}(X, Y, \text{size}=\text{subsample} * \text{len}(Y))$

 # Compute residuals

$\text{residuals} = Y_batch - F_m[\text{indices_of}(X_batch)]$

Train weak learner on residuals



$\text{tree} = \text{train_decision_tree}(X_batch, \text{residuals})$

$\text{save}(\text{tree})$

 # Update predictions

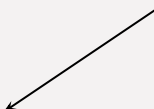
 for leaf in tree.leaves :

$\text{gamma} = \text{mean}(\text{residuals_in_leaf}(\text{leaf}))$

$\text{gamma_m.append}(\text{gamma})$

$F_m[\text{region_mask}] += \text{self.learning_rate} * \text{gamma}$

We iterate over weak learner's leaf nodes, compute the average residuals for samples in each leaf, and update their predictions using this value scaled by the learning rate.



save gamma

How weak learner works

Splitting the Data:

- Finding the Best Split: one that minimizes the sum of squared errors (SSE) in the two resulting subsets (left and right).
- Split stops if: The number of samples in a node is less than `min_samples`. The tree depth exceeds `max_depth`.

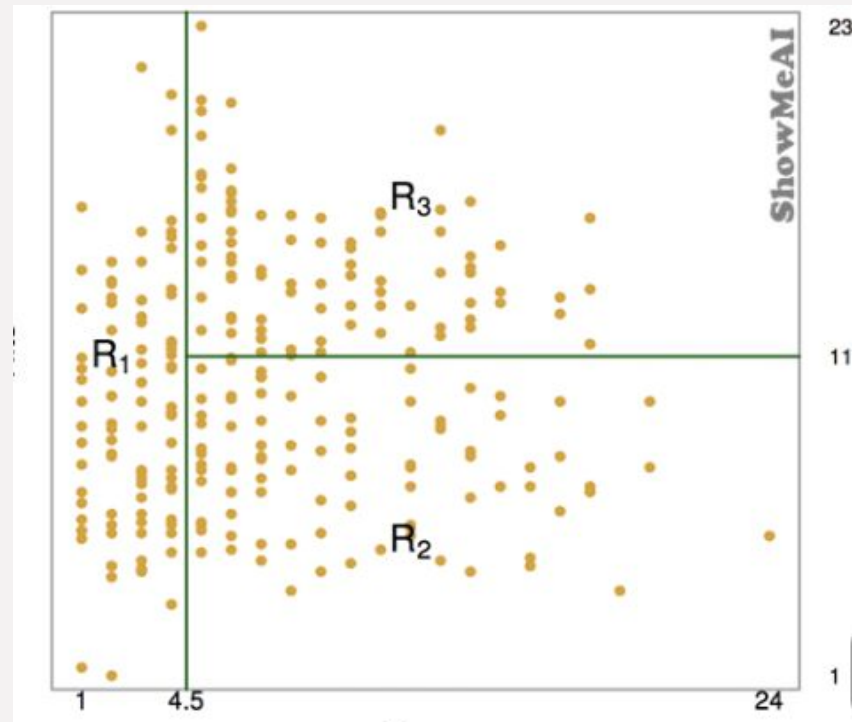
Growing the Tree:

- The data is recursively split into left and right subsets
- The left subset contains samples where the feature value is less than the split point.
- Nodes at the maximum depth or with insufficient samples become **leaf nodes**.

Leaf Nodes:

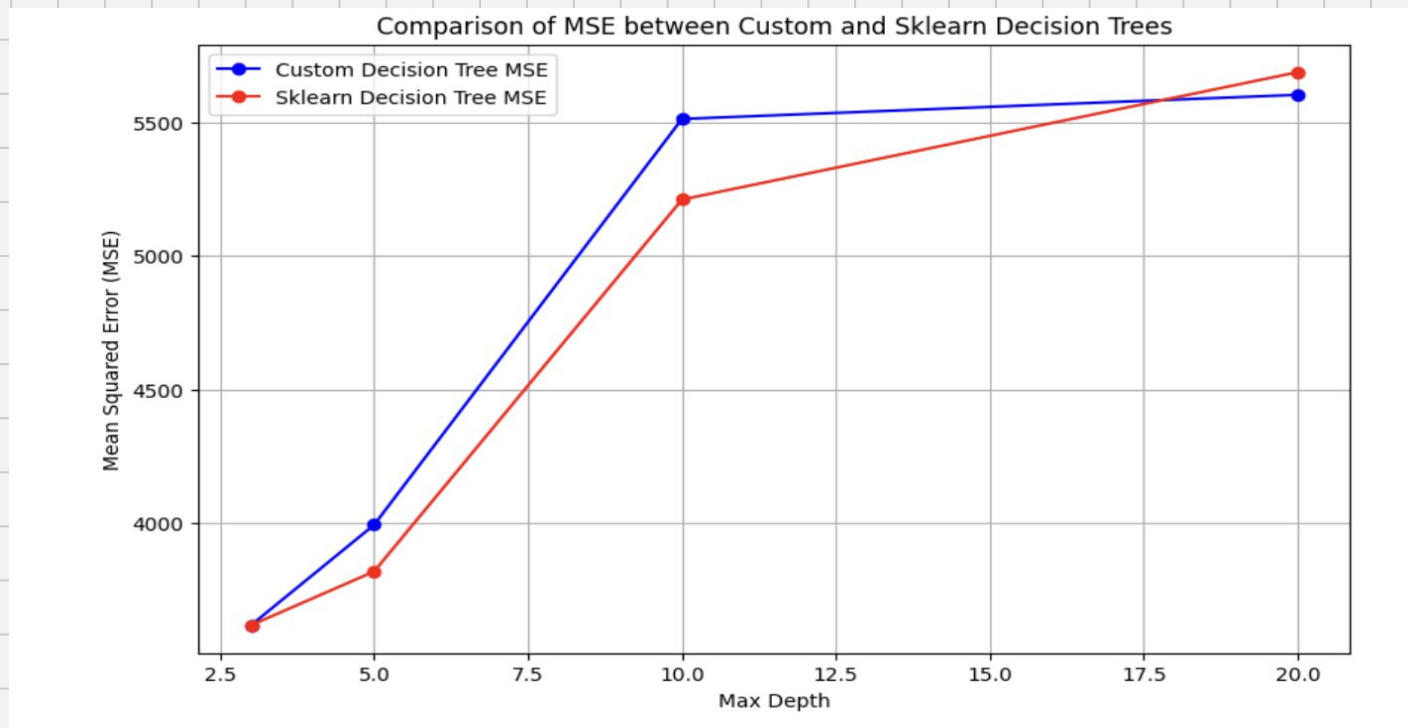
A leaf node contains the average of the target values of the samples in that node. This average serves as the prediction for any input sample that ends up in that leaf.

Prediction result of regression decision tree



Our weak learner vs sklearn weak learner

We use diabetes dataset to make comparison:



Our Model vs. Sklearn

(GradientBoostingRegressor)

The Dataset: UCI wine

Features

- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- Chlorides
- free sulfur dioxide
- total sulfur dioxide
- Density
- pH
- Sulphates
- alcohol

- 4898 examples
- 11 features
- **Goal:** predict the quality of a wine (scored out of 10) given its various attributes

Target Variable

- Wine quality

```
X, Y = data[:, 1:], data[:, 0]
```

The Dataset

Hyperparameters

```
my_model = StochasticGradientBoosting(  
    learning_rate=0.1,  
    n_estimators=100,  
    subsample=0.8,  
    min_samples=2,  
    max_depth=3  
)
```

- **Estimators=100:** allows the model to achieve good performance and maintain reasonable computational cost.
- **Subsample=0.8:** Helps prevent overfitting, as each tree sees slightly different subsets of data. Improves generalization, similar to what happens with bagging.
- **min_samples_split=2:** default value, larger values lead to less complex trees, prevents overfitting but more likely to underfit.
- **max_depth=3:** simple, fast, less overfitting.

Results

- 10 random states
- Compared mean and standard deviation
- Long runtime
- Similar results with sklearn



----- Results Summary -----

My Model:

Training Loss - Mean: 0.3880, Std: 0.0049

Testing Loss - Mean: 0.4898, Std: 0.0344

Sklearn Model:

Training Loss - Mean: 0.3861, Std: 0.0073

Testing Loss - Mean: 0.4897, Std: 0.0332

----- Comparison -----

Training Loss Difference - Mean: 0.0019

Testing Loss Difference - Mean: 0.0002

Training Loss Difference - Std: -0.0023

Testing Loss Difference - Std: 0.0012

Summary

What's interesting



Stochasticity:

- The use of subsampling introduces randomness into training, reducing overfitting compared to standard Gradient Boosting.

Flexibility:

- Parameters like `learning_rate`, `n_estimators`, and `max_depth` control the balance between model complexity and generalization.

Math:

- Simple tricks like adding a $\frac{1}{2}$ to make later derivation easier.

Challenging



Weak learner implementation doesn't use optimized algorithms like CART (Classification and Regression Trees), and this results in a significant runtime cost during training.

Thank you for
listening