**Machine Learning Algorithms Part\_4**

* + **1.1.1.5 Gradient Boosting Regressor**

<https://www.youtube.com/watch?v=fbKz7N92mhQ>

<https://www.youtube.com/watch?v=nMNiTZm-qY0&t=36s>

<https://www.youtube.com/watch?v=nMNiTZm-qY0&t=32s>

<https://www.youtube.com/watch?v=JmXnztjULnQ>

<https://www.youtube.com/watch?v=a20TaKNsriE>

<https://www.youtube.com/watch?v=gmp2tS2joaA>

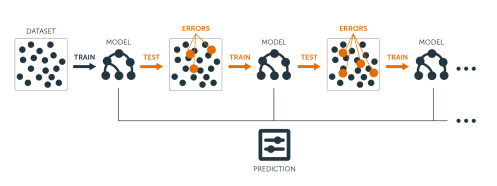
**Gradient Boosting: Intuition behind the algorithm**

**Source - https://medium.com/analytics-vidhya/gradient-boosting-intuition-behind-the-algorithm-e567ed908100**

**Steps Involved in Gradient Boosting Regression**

To build a Gradient Boosting model, follow these general steps:

1. **Initialize the model** with a simple prediction, often the mean of the target values.
2. **Calculate residuals** for each data point by finding the difference between the actual values and the predictions.
3. **Fit a weak model** (usually a decision tree) to these residuals.
4. **Update the predictions** by adding the new model’s predictions, scaled by a learning rate, to the existing predictions.
5. **Repeat steps 2–4** for a set number of iterations or until the residuals are sufficiently minimized.



*Gradient boosting is very popular and widely used in the field of ML. It is a boosting technique which can be applied with classification or regression problem.*

*Boosting is a method for creating an ensemble. It starts by fitting an initial model (e.g. a tree or linear regression) to the data. Then a second model is built that focuses on accurately predicting the cases where the first model performs poorly. The combination of these two models is expected to be better than either model alone. Then you repeat this process of boosting many times. Each successive model attempts to correct for the shortcomings of the combined boosted ensemble of all previous models.*

The main intuition behind the algorithm is that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model to minimize the error. Let’s understand this with an example of data for the regression problem.



In this problem, we want to predict Salary (Target) based on Experience and Degree (Independent Variables) of a candidate.

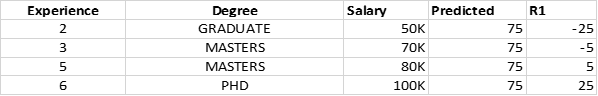
In the case of regression we first create our base model which will be the average of all the actual output.

In our case it would be 50 + 70 + 80 + 100 / 4 = 75

So this base model will give output 75k for any next prediction.

Next, we will calculate Pseudo Residual which would be

**actual(Salary) — Predicted.**



In the next step, we will create a decision tree by taking independent variables (Experience, Degree) as input and Residual R1 as output.

After this step we will have two models. First is the base model and the second is we created by residual R1. Now we can do current prediction by adding values from both models. Let’s look at the predicted value for (Experience =2 AND Degree = GRADUATE)

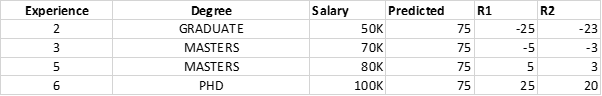
**Base Model + M(R1) = 75 + (-25) = 50 (Equals to Actual)**

As we can see that predicted value is equal to the actual value means the model is overfitting or we can say our model has low bias and high variance.

To overcome this problem algorithm use parameter alpha(@) which is called learning rate. The value of the learning rate lies between (0,1). So applying this our next prediction assuming (@ = 0.1) would be: -

**Base Model + (@) \* M(R1) = 75 + (0.1)(-25) = 72.5 (Actual = 50)**

In the next step, we will again calculate residuals and then predict values using all the weak learners. Let’s look at the next step below



From this state, our prediction would be –

**Base Model + (@) \* M(R1) + (@) \* M(R2) = 75 + (0.1)(-25) + (0.1)(-23)= 70.2 (Actual = 50)**

You have observed it that as we are going further and adding more weak learners our residuals are getting decreased and we are predicted value is approaching actual value which is our motto for this problem. To generalize this we can write an equation –

F(x) = h0(x) + @1 \* h1(x) + @2 \* h2(x)…………………. + @n \* hn(x).

I hope this will clarify the intuition behind the Gradient boosting algorithm. We can further explore mathematical implementation for getting in-depth.

Must read - <https://medium.datadriveninvestor.com/gradient-boosting-for-regression-539fa8aa4b00>

<https://medium.com/data-science/gradient-boosting-regressor-explained-a-visual-guide-with-code-examples-c098d1ae425c>

Gradient Boosting is a ensemble learning method used for classification and regression tasks. It is a[**boosting**](https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/) algorithm which combine multiple weak learner to create a strong predictive model. It works by sequentially training models where each new model tries to correct the errors made by its predecessor.

In gradient boosting each new model is trained to minimize the loss function such as mean squared error or cross-entropy of the previous model using [gradient descent](https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/). In each iteration the algorithm computes the gradient of the loss function with respect to predictions and then trains a new weak model to minimize this gradient. Predictions of the new model are then added to the ensemble (all models prediction) and the process is repeated until a stopping criterion is met.

**Shrinkage and Model Complexity**

A key feature of Gradient Boosting is shrinkage which scales the contribution of each new model using **learning rate** (denoted as η*η*).

* **Smaller learning rates:** mean the contribution of each tree is smaller which reduces the risk of overfitting but requires more trees to achieve the same performance.
* **Larger learning rates:** mean each tree has a more significant impact but this can lead to overfitting.

There's a trade off between the learning rate and the number of estimators (trees) a smaller learning rate usually means more trees are required to achieve optimal performance.

Let's now use a more **realistic dataset** with **4 rows and 3 features**, simulating a problem like **predicting house prices** based on features like:

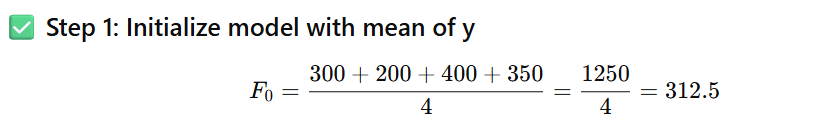
* 🏠 Size (sqft)
* 🛏️ Bedrooms
* 📍 LocationScore (a numerical proxy for neighborhood quality)

**📊 Sample Dataset**

| **ID** | **Size (sqft)** | **Bedrooms** | **LocationScore** | **Target Price (y)** |
| --- | --- | --- | --- | --- |
| 1 | 1500 | 3 | 7 | 300 |
| 2 | 1200 | 2 | 5 | 200 |
| 3 | 1800 | 4 | 8 | 400 |
| 4 | 1700 | 3 | 6 | 350 |

* **Learning Rate (η)**: 0.1
* **Model**: Gradient Boosting Regressor
* **# Trees**: 3
* **Loss Function**: Mean Squared Error
* **Goal**: Learn to predict house prices step by step

**🌟 Gradient Boosting Regression: Step-by-Step**

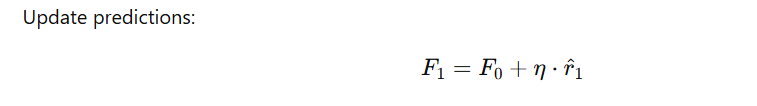
****

| **ID** | **y** | **F₀ = Initial Prediction** | **Residual r₁ = y - F₀** |
| --- | --- | --- | --- |
| 1 | 300 | 312.5 | -12.5 |
| 2 | 200 | 312.5 | -112.5 |
| 3 | 400 | 312.5 | +87.5 |
| 4 | 350 | 312.5 | +37.5 |

**✅ Step 2: Train Tree T1 on residuals (r₁)**

Suppose Tree T1 learns the following prediction for residuals:

| **ID** | **True Residual r₁** | **Predicted r̂₁** |
| --- | --- | --- |
| 1 | -12.5 | -10 |
| 2 | -112.5 | -90 |
| 3 | +87.5 | +80 |
| 4 | +37.5 | +30 |



| **ID** | **F₀** | **r̂₁** | **F₁ = F₀ + 0.1 × r̂₁** |
| --- | --- | --- | --- |
| 1 | 312.5 | -10 | 311.5 |
| 2 | 312.5 | -90 | 303.5 |
| 3 | 312.5 | +80 | 320.5 |
| 4 | 312.5 | +30 | 315.5 |

**✅ Step 3: Train Tree T2 on residuals r₂ = y - F₁**

| **ID** | **y** | **F₁** | **r₂ = y - F₁** |
| --- | --- | --- | --- |
| 1 | 300 | 311.5 | -11.5 |
| 2 | 200 | 303.5 | -103.5 |
| 3 | 400 | 320.5 | +79.5 |
| 4 | 350 | 315.5 | +34.5 |
|  |  |  |  |

Suppose Tree T2 predicts:

| **ID** | **r₂ (true)** | **r̂₂ (Tree T2)** |
| --- | --- | --- |
| 1 | -11.5 | -9 |
| 2 | -103.5 | -85 |
| 3 | +79.5 | +70 |
| 4 | +34.5 | +25 |



| **ID** | **F₁** | **r̂₂** | **F₂ = F₁ + 0.1 × r̂₂** |
| --- | --- | --- | --- |
| 1 | 311.5 | -9 | 310.6 |
| 2 | 303.5 | -85 | 295.0 |
| 3 | 320.5 | +70 | 327.5 |
| 4 | 315.5 | +25 | 318.0 |

**✅ Step 4: Train Tree T3 on residuals r₃ = y - F₂**

| **ID** | **y** | **F₂** | **r₃ = y - F₂** |
| --- | --- | --- | --- |
| 1 | 300 | 310.6 | -10.6 |
| 2 | 200 | 295.0 | -95.0 |
| 3 | 400 | 327.5 | +72.5 |
| 4 | 350 | 318.0 | +32.0 |
|  |  |  |  |

Tree T3 predictions:

| **ID** | **r₃ (true)** | **r̂₃ (Tree T3)** |
| --- | --- | --- |
| 1 | -10.6 | -8 |
| 2 | -95.0 | -75 |
| 3 | +72.5 | +65 |
| 4 | +32.0 | +20 |



| **ID** | **F₂** | **r̂₃** | **F₃ = Final Prediction** |
| --- | --- | --- | --- |
| 1 | 310.6 | -8 | 309.8 |
| 2 | 295.0 | -75 | 287.5 |
| 3 | 327.5 | +65 | 334.0 |
| 4 | 318.0 | +20 | 320.0 |

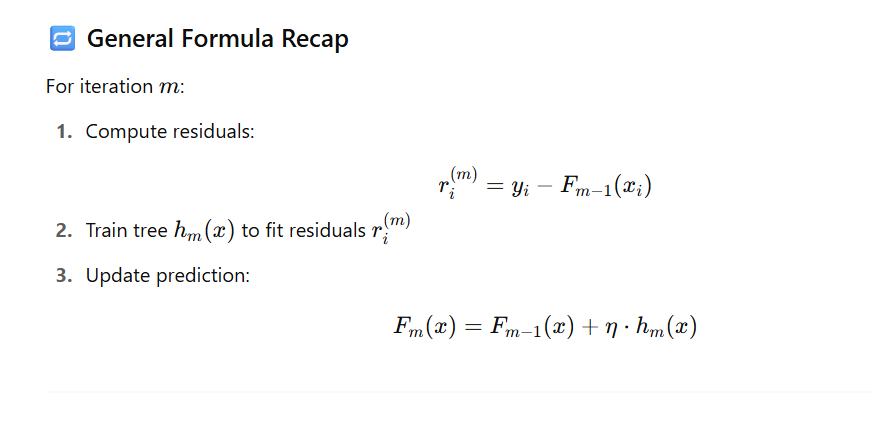
**🧾 Final Model Output**

| **ID** | **Actual y** | **Final Prediction (F₃)** | **Residual (Error)** |
| --- | --- | --- | --- |
| 1 | 300 | 309.8 | -9.8 |
| 2 | 200 | 287.5 | -87.5 |
| 3 | 400 | 334.0 | +66.0 |
| 4 | 350 | 320.0 | +30.0 |

**✅ Summary of Each Tree Step**

| **Step** | **Model Used** | **Target** | **Update Formula** |
| --- | --- | --- | --- |
| 1 | Mean Predictor | — | F₀ = mean(y) = 312.5 |
| 2 | Tree T1 | r₁ = y - F₀ | F₁ = F₀ + 0.1 × T1(x) |
| 3 | Tree T2 | r₂ = y - F₁ | F₂ = F₁ + 0.1 × T2(x) |
| 4 | Tree T3 | r₃ = y - F₂ | F₃ = F₂ + 0.1 × T3(x) |

Would you like the **Python code** version of this example to test or visualize it?



Code example same for above to understand working

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeRegressor

import matplotlib.pyplot as plt

from sklearn import tree

# Sample dataset

data = pd.DataFrame({

"Size": [1500, 1200, 1800, 1700],

"Bedrooms": [3, 2, 4, 3],

"LocationScore": [7, 5, 8, 6],

"Price": [300, 200, 400, 350]

})

X = data[["Size", "Bedrooms", "LocationScore"]]

y = data["Price"]

# Initialize prediction with mean

F0 = np.full\_like(y, y.mean(), dtype=float)

learning\_rate = 0.1

predictions = [F0]

trees = []

residuals\_list = []

# Function to display model step summary

def print\_model\_step(step, y\_true, pred\_before, tree\_model, step\_name):

residuals = y\_true - pred\_before

residuals\_list.append(residuals)

tree\_pred = tree\_model.predict(X)

updated\_pred = pred\_before + learning\_rate \* tree\_pred

predictions.append(updated\_pred)

result\_df = pd.DataFrame({

"y": y\_true,

f"F\_{step-1}": pred\_before.round(2),

f"r\_{step}": residuals.round(2),

f"r̂\_{step} (Tree Prediction)": tree\_pred.round(2),

f"F\_{step}": updated\_pred.round(2),

})

print(f"\n🌳 Step {step}: {step\_name}")

print(result\_df)

return updated\_pred

# Step 1: Train tree on residuals from F0

r1 = y - F0

tree1 = DecisionTreeRegressor(max\_depth=2, random\_state=42)

tree1.fit(X, r1)

trees.append(tree1)

F1 = print\_model\_step(1, y, F0, tree1, "Train on r1 = y - F0")

# Step 2: Train tree on residuals from F1

r2 = y - F1

tree2 = DecisionTreeRegressor(max\_depth=2, random\_state=43)

tree2.fit(X, r2)

trees.append(tree2)

F2 = print\_model\_step(2, y, F1, tree2, "Train on r2 = y - F1")

# Step 3: Train tree on residuals from F2

r3 = y - F2

tree3 = DecisionTreeRegressor(max\_depth=2, random\_state=44)

tree3.fit(X, r3)

trees.append(tree3)

F3 = print\_model\_step(3, y, F2, tree3, "Train on r3 = y - F2")

# Final Results

final\_df = pd.DataFrame({

"ID": data.index + 1,

"Actual y": y,

"Final Prediction F3": F3.round(2),

"Residual (y - F3)": (y - F3).round(2)

})

print("\n📘 Final Model Output After 3 Trees:")

print(final\_df)

# Plot trees

fig, axs = plt.subplots(1, 3, figsize=(18, 5))

for i, tr in enumerate(trees, 1):

tree.plot\_tree(tr, feature\_names=X.columns, filled=True, ax=axs[i-1])

axs[i-1].set\_title(f"Tree {i}")

plt.tight\_layout()

plt.show()

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeRegressor

from sklearn import tree

import matplotlib.pyplot as plt

# Sample realistic dataset

data = pd.DataFrame({

"Size": [1500, 1200, 1800, 1700],

"Bedrooms": [3, 2, 4, 3],

"LocationScore": [7, 5, 8, 6],

"Price": [300, 200, 400, 350]

})

X = data[["Size", "Bedrooms", "LocationScore"]]

y = data["Price"]

# Initialize model

learning\_rate = 0.1

F0 = np.full\_like(y, y.mean(), dtype=float)

predictions = [F0]

trees = []

def print\_step\_info(step\_num, y, F\_prev, tree\_model, learning\_rate):

residuals = y - F\_prev

tree\_pred = tree\_model.predict(X)

F\_new = F\_prev + learning\_rate \* tree\_pred

print(f"\n🌳 Step {step\_num}: Train on residuals r{step\_num} = y - F{step\_num - 1}")

step\_df = pd.DataFrame({

"y": y,

f"F\_{step\_num-1}": F\_prev.round(2),

f"r\_{step\_num} = y - F\_{step\_num-1}": residuals.round(2),

f"r̂\_{step\_num} (Tree Prediction)": tree\_pred.round(2),

f"F\_{step\_num}": F\_new.round(2)

})

print(step\_df)

return F\_new

# Step 1

r1 = y - F0

tree1 = DecisionTreeRegressor(max\_depth=2, random\_state=42)

tree1.fit(X, r1)

trees.append(tree1)

F1 = print\_step\_info(1, y, F0, tree1, learning\_rate)

# Step 2

r2 = y - F1

tree2 = DecisionTreeRegressor(max\_depth=2, random\_state=43)

tree2.fit(X, r2)

trees.append(tree2)

F2 = print\_step\_info(2, y, F1, tree2, learning\_rate)

# Step 3

r3 = y - F2

tree3 = DecisionTreeRegressor(max\_depth=2, random\_state=44)

tree3.fit(X, r3)

trees.append(tree3)

F3 = print\_step\_info(3, y, F2, tree3, learning\_rate)

# Final result

final\_result = pd.DataFrame({

"ID": data.index + 1,

"Actual y": y,

"Final Prediction F3": F3.round(2),

"Residual (y - F3)": (y - F3).round(2)

})

print("\n📘 Final Model Output After 3 Trees:")

print(final\_result)

# Plot the decision trees

fig, axs = plt.subplots(1, 3, figsize=(18, 5))

for i, model in enumerate(trees):

tree.plot\_tree(model, feature\_names=X.columns, filled=True, ax=axs[i])

axs[i].set\_title(f"Tree {i+1}")

plt.tight\_layout()

plt.show()

**🔍 Real-World Use Cases**

| **Domain** | **Use Case** |
| --- | --- |
| 🏠 Real Estate | House price prediction |
| 📈 Finance | Stock market forecasting |
| ⚙️ Industry | Predictive maintenance costs |
| 🎮 Gaming | In-game purchase value prediction |
| 🛒 E-commerce | Predict customer spend/lifetime value |

**🟩 Pros**

* **High prediction accuracy**
* Reduces **bias** significantly
* Flexible — can optimize for **different loss functions**
* Works well even with **unclean data**

**🟥 Cons**

* **Slower** to train (especially with large datasets)
* Requires **careful tuning**
* Can **overfit** if n\_estimators is too high
* **Not easily interpretable**

**🧮 Key Hyperparameters**

| **Hyperparameter** | **Description** |
| --- | --- |
| n\_estimators | Number of boosting rounds (trees) |
| learning\_rate | Shrinks the contribution of each tree |
| max\_depth | Maximum depth of each regression tree |
| subsample | Fraction of samples to use for fitting each tree |
| loss | Loss function to optimize (ls, lad, huber) |

**🧠 Concept Recap**

* Gradient Boosting **minimizes loss** using the **gradient of the error**.
* Each tree learns **residuals** from previous trees.
* **Smaller learning rates** usually require **more estimators** for best performance.