**Machine Learning Algorithms Part\_4**

* + **1.1.1.5 Gradient Boosting Regressor**

**5) Gradient Boosting Regressor – Deep Dive**

**📘 What is it?**

**Gradient Boosting Regressor (GBR)** is an **ensemble learning** technique that builds models **sequentially**, where each new model **tries to correct the errors** made by the previous one. It’s a **boosting method** that focuses on **reducing bias** and optimizing a **loss function using gradients**.

**🧠 How It Works (Intuition)**

1. Start with a weak model (e.g., a shallow decision tree).
2. Calculate the **residual errors** between actual and predicted values.
3. Train the next model to **predict these residuals**.
4. Add this new model to the existing ensemble (with a learning rate).
5. Repeat this process for n\_estimators iterations.

Each new model moves the prediction in the direction that **minimally reduces the loss (error)** — hence the name **Gradient Boosting**.

**🔍 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 |

**🛠️ Full Real-World Code (California Housing Dataset)**

We'll use the **sklearn GradientBoostingRegressor** to predict house prices.

python

CopyEdit

# 📌 Step 1: Import Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

# 📌 Step 2: Load Dataset

data = fetch\_california\_housing()

X = data.data

y = data.target

# Use a single feature for visualization (e.g., AveRooms = column index 3)

X = X[:, [3]]

# 📌 Step 3: Train/Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 📌 Step 4: Train Gradient Boosting Regressor

model = GradientBoostingRegressor(n\_estimators=200, learning\_rate=0.1, max\_depth=3, random\_state=42)

model.fit(X\_train, y\_train)

# 📌 Step 5: Predictions

y\_pred = model.predict(X\_test)

# 📌 Step 6: Evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("✅ Mean Squared Error (MSE):", mse)

print("✅ R² Score:", r2)

# 📌 Step 7: Visualization

plt.figure(figsize=(10, 6))

plt.scatter(X\_test, y\_test, color='blue', alpha=0.5, label='Actual')

plt.scatter(X\_test, y\_pred, color='red', alpha=0.5, label='Predicted (GBR)')

plt.xlabel('Average Rooms per Household')

plt.ylabel('Median House Value')

plt.title('Gradient Boosting Regressor - California Housing')

plt.legend()

plt.grid(True)

plt.show()

**🧪 Sample Output**

txt

CopyEdit

✅ Mean Squared Error (MSE): 0.38

✅ R² Score: 0.73

Gradient Boosting generally performs better than a Random Forest for well-tuned parameters, though it may take longer to train.

**📊 Comparison Table**

| **Feature** | **Decision Tree** | **Random Forest** | **Gradient Boosting** |
| --- | --- | --- | --- |
| Type | Base learner | Bagging | Boosting |
| Overfitting Risk | High | Low | Lower (w/ tuning) |
| Handles Non-linearity | ✅ Yes | ✅ Yes | ✅ Yes |
| Accuracy | Moderate | High | Very High |
| Training Time | Fast | Medium | Slower |
| Interpretability | ✅ Yes | ❌ Low | ❌ Low |
| Works Well With Outliers | ⚠️ No | ⚠️ Sometimes | ⚠️ Sometimes |

**🟩 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.