Gradient Boosting for Classification

`from sklearn.ensemble import GradientBoostingClassifier`

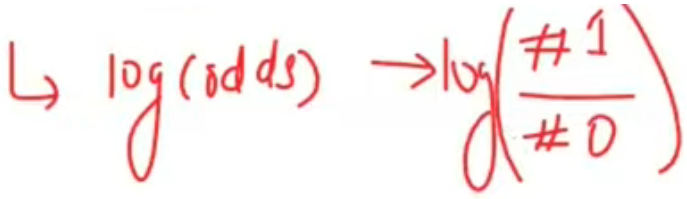
- Used for classification tasks

`loss` :

- The loss function to optimize. For classification, common choices are:
  - `'log_loss'` (logistic regression loss, default).
  - `'exponential'` (AdaBoost loss).

How Gradient Boosting Works for Classification:

1. **Initialize the Model**:
   - Start with an initial prediction (e.g., the **log-odds** of the target class probabilities).

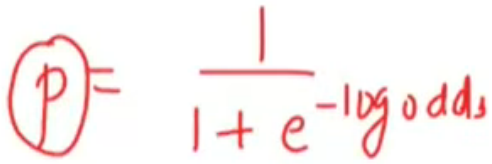


👆 ***log (No. of times 1 occurred / No. of times 0 occurred)***

- **Example:**
  - Suppose the probability of an event is 0.75.
  - Odds = 0.75 / (1 - 0.75) = 0.75 / 0.25 = 3

- Log-odds = ln(3) ≈ ***1.0986***

- Calculate probability from **logit**👆



	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)
0	6.82	118	0	0.510826	0.625
1	6.36	125	1	0.510826	0.625
2	5.39	99	1	0.510826	0.625
3	5.50	106	1	0.510826	0.625
4	6.39	148	0	0.510826	0.625
5	9.13	148	1	0.510826	0.625
6	7.17	147	1	0.510826	0.625
7	7.72	72	0	0.510826	0.625

2. **Compute Residuals**:

- Calculate the residuals (errors) between the actual class labels and the predicted probabilities.

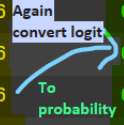
```
# calculating residual for stage 1
df['res1'] = df['is_placed'] - df['pre1(probability)']
df
```

	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)	res1
0	6.82	118	0	0.510826	0.625	-0.625
1	6.36	125	1	0.510826	0.625	0.375
2	5.39	99	1	0.510826	0.625	0.375
3	5.50	106	1	0.510826	0.625	0.375
4	6.39	148	0	0.510826	0.625	-0.625
5	9.13	148	1	0.510826	0.625	0.375
6	7.17	147	1	0.510826	0.625	0.375
7	7.72	72	0	0.510826	0.625	-0.625

3. **Fit a Weak Learner**:
    - Train a weak learner (e.g., a decision tree) to predict the residuals.
    - The DT is **regression tree** because your output is a number.
    - Convert the ***probability*** values of residuals → ***log(odds)*** with the following formula:

$$\frac{\sum Residual}{\sum[PreviousProb * (1 - PreviousProb)]}$$

	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)	res1	leaf_entry1	pre2(log-odds)	pre2(probability)
0	6.82	118	0	0.510826	0.625	-0.625	3	-2.159174	0.103477
1	6.36	125	1	0.510826	0.625	0.375	1	2.110826	0.891951
2	5.39	99	1	0.510826	0.625	0.375	1	2.110826	0.891951
3	5.50	106	1	0.510826	0.625	0.375	1	2.110826	0.891951
4	6.39	148	0	0.510826	0.625	-0.625	4	0.690826	0.666151
5	9.13	148	1	0.510826	0.625	0.375	4	0.690826	0.666151
6	7.17	147	1	0.510826	0.625	0.375	4	0.690826	0.666151
7	7.72	72	0	0.510826	0.625	-0.625	3	-2.159174	0.103477



- Calculate the residue of the 2nd model

4. **Update the Model**:
    - Add the predictions of the weak learner to the current model, scaled by a learning rate.

5. **Repeat**:
    - Repeat steps 2-4 for a specified number of iterations or until the residuals are minimized.
6. **Final Prediction**:
    - The final prediction is the sum of the initial prediction and all the scaled predictions from the weak learners, transformed into class probabilities using a logistic function.

Python code

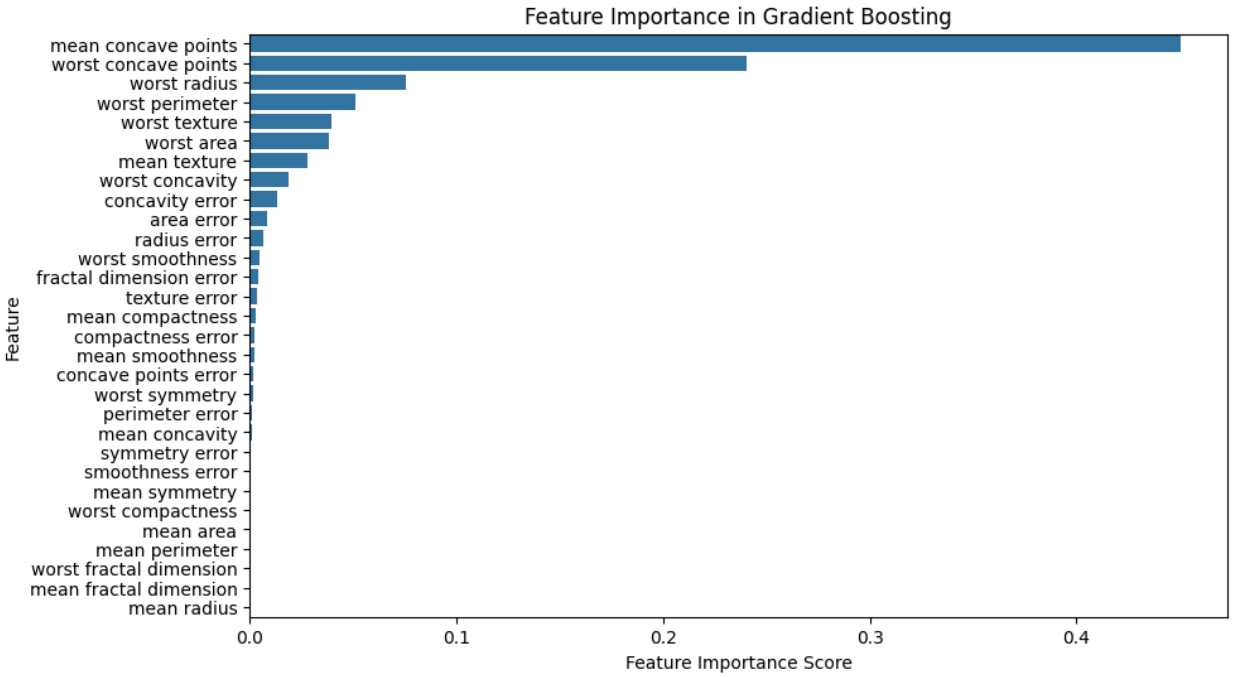
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_
matrix
```

```
# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Initialize GradientBoostingClassifier
model = GradientBoostingClassifier(random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=data.target_names))

# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.96

Classification Report:
              precision    recall  f1-score   support

   malignant       0.95      0.93      0.94        43
      benign       0.96      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114


Confusion Matrix:
[[40  3]
 [ 2 69]]
```

Feature Importance

```
# Get feature importance
feature_importance = model.feature_importances_

# Convert to DataFrame
importance_df = pd.DataFrame({'Feature': data.feature_names, 'Importance': feature_importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x=importance_df['Importance'], y=importance_df['Feature'])
plt.xlabel("Feature Importance Score")
plt.ylabel("Feature")
plt.title("Feature Importance in Gradient Boosting")
plt.show()
```




```
importance_df
```

	Feature	Importance
7	mean concave points	0.450528
27	worst concave points	0.240103
20	worst radius	0.075589
22	worst perimeter	0.051408
21	worst texture	0.039886
23	worst area	0.038245
1	mean texture	0.027805
26	worst concavity	0.018725
16	concavity error	0.013068
13	area error	0.008415
10	radius error	0.006870
24	worst smoothness	0.004811
19	fractal dimension error	0.004224
11	texture error	0.003604
5	mean compactness	0.002996
15	compactness error	0.002511
4	mean smoothness	0.002467
17	concave points error	0.002038
28	worst symmetry	0.001478
12	perimeter error	0.001157
6	mean concavity	0.000922
18	symmetry error	0.000703
14	smoothness error	0.000556
8	mean symmetry	0.000520
25	worst compactness	0.000450
3	mean area	0.000425