# HW1 Report

10550097 王若禎

## I. Introduction

In this project, I use both the Naïve Bayes classifier and Regression-based classifiers to build classification models. To evaluate their performance, I apply various metrics including the confusion matrix, ROC curve, Accuracy, AUC, F1-score, support, and feature importance.

I selected four different datasets for prediction tasks in this project: the Online Shoppers Purchasing Intention Dataset, Titanic Survival Prediction, Seeds, and Dry Bean datasets. The first two are binary classification problems, while the latter two involve multi-class classification. These datasets are used to perform classification, evaluation, and prediction analyses.

## II. Dataset

In this project, I use four datasets to do classification, evaluation, and prediction analyses.

### i. Shoppers Purchasing Intention Dataset

This dataset contains information from 12,330 unique user sessions collected over the span of one year. Each session represents a distinct user to avoid biases related to specific marketing campaigns, special dates, user profiles, or seasonal patterns. The dataset consists of 18 attributes—10 numerical and 8 categorical.

For this project, I use 17 of these attributes as input features for classification. These include: *Administrative*, *Administrative_Duration*, *Informational*, *Informational_Duration*, *ProductRelated*, *ProductRelated_Duration*, *BounceRates*, *ExitRates*, *PageValues*, *SpecialDay*, *Month*, *OperatingSystems*, *Browser*, *Region*, *TrafficType*, *VisitorType*, and *Weekend*. The prediction target is the "*Revenue*" attribute, a boolean indicating whether the user completed a purchase or not.

**ii. Titanic Survival Prediction**

The dataset contains detailed information about passengers aboard the RMS Titanic and is widely used for binary classification tasks—particularly to predict whether a passenger survived or not. While the dataset includes 11 columns, I selected 8 key features for classification and prediction.

In this project, I use *PassengerId, Pclass, Sex, Age, SibSp, Parch, and Fare* as input features. The target variable is "*Survived*", which indicates whether the passenger survived the disaster.

**iii. Seeds**

This dataset contains information about the physical characteristics of seeds, focusing on their appearance. It is used to classify seeds into three distinct categories. The dataset consists of 8 numerical features.

In this project, I use all 8 features as input for prediction. These include: *area, perimeter, compactness, length_of_kernel, width_of_kernel, asymmetry_coefficient,* and *length_of_kernel_groove*. The prediction target is the *"class"* attribute, which represents the category of the seed, classified into one of three types.

**iv. Dry_Bean_DataSet**

This dataset contains information about dry bean seeds, with most features describing their physical appearance. The beans are categorized into seven different classes, and the dataset includes 13,611 entries.

There are 17 features in total, including:

*Area, Perimeter, MajorAxisLength, MinorAxisLength, AspectRatio, Eccentricity, ConvexArea, EquivDiameter, Extent, Solidity, Roundness, Compactness, ShapeFactor1, ShapeFactor2, ShapeFactor3, and ShapeFactor4.*

The prediction target is the *"Class"* attribute, which classifies each bean into one of seven categories.

# III. Method

## i. Data Preprocessing

First, I converted all target class values into numerical format, since some of them were represented as headings or text labels.

Second, I observed a significant data imbalance problem across the four datasets. To address this, I calculated the size of the majority class and applied oversampling to balance the class distribution. I then compared the prediction results with and without oversampling to evaluate its impact and attempt to improve model performance.

## ii. Classification model - Naive Bayes

In this project, I use NumPy to build a Gaussian Naive Bayes classification model and implement two versions: one for binary (two-class) datasets and another for multi-class datasets.

The model is designed to handle continuous input features by assuming that each feature follows a normal (Gaussian) distribution. During training, the fit function computes the mean, standard deviation, and prior probability for each class based on the training data. The _likelihood function calculates the probability of a feature value given a class using the Gaussian distribution formula, while the _posterior function computes the log-posterior probability for each class. The predict function applies the _posterior method to each input sample and returns the class with the highest posterior probability.

To ensure numerical stability, the model also clips likelihood values and adds a small constant to the standard deviation to avoid issues like division by zero or taking the logarithm of zero.

## iii. Classification function – Logistic Regression

I used NumPy to build a Logistic Regression model, which is a supervised learning algorithm. It uses the logistic function to map real-valued inputs into values between 0 and 1, representing the probability of a given class. The model learns optimal parameters by minimizing a loss function—typically cross-entropy—to find the best fit.

In the fit function, the model initializes weights and biases to zero, converts the target labels into one-hot encoded vectors, and updates the weights and biases based on the cross-entropy loss using the predicted class probabilities from the softmax function. The softmax function ensures that the model outputs a valid probability distribution for each input sample. The predict function returns the class with the highest predicted probability, while predict_proba returns the complete probability distribution across all classes.

**iv. Evaluation Modules**

In this project, I use four different methods to construct the evaluation module, including:

1. Confusion Matrix:

A table that compares the predicted labels with the true labels, showing the counts of correct and incorrect predictions across different classes. This helps evaluate the classification performance in detail.

2. ROC Curve:

Used for binary (two-class) datasets, the ROC curve plots the True Positive Rate (TPR) or Recall on the y-axis against the False Positive Rate (FPR) on the x-axis at various classification thresholds. It helps assess the model's ability to distinguish between classes.

3. Feature Importance:

After training a logistic regression model, I evaluate how much each feature contributes to the model's predictions. This provides insight into which features have the most impact on the model's performance.

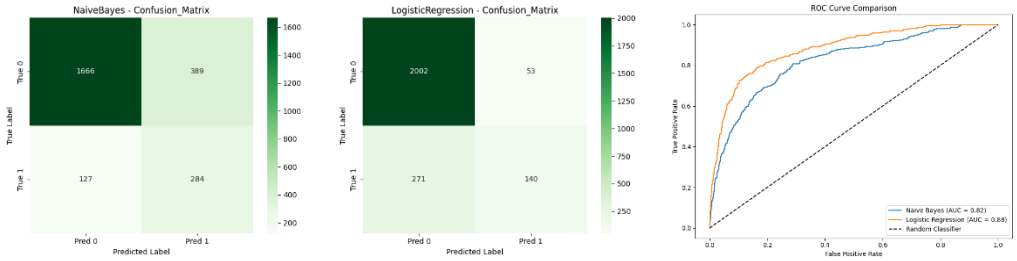
4. Accuracy, AUC, F1-score, support:

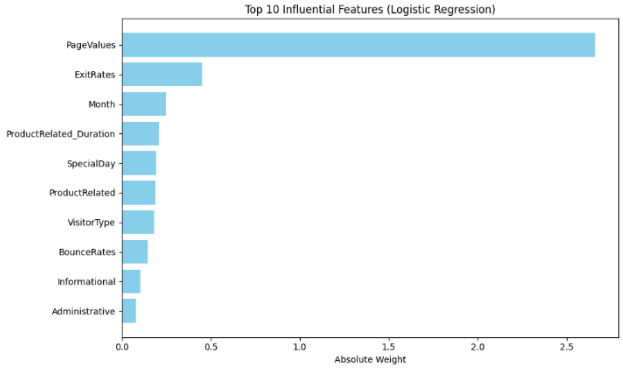
Accuracy indicates the overall correctness of the model. AUC means the area under the ROC curve. F1-score combines both precision and recall into a single metric. Support shows the number of actual instances for each class. Together, these metrics provide a comprehensive understanding of the model's performance.

# IV. Experiment and Result

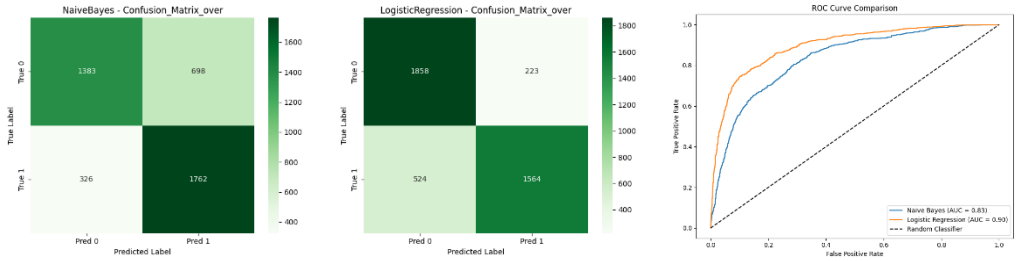
## i. Shoppers Purchasing Intention Dataset

After conducting the experiment on the dataset, I found that the Logistic Regression model produced better prediction results compared to the Gaussian Naive Bayes model. The accuracy rate of Logistic Regression also increased more rapidly. Below are the confusion matrix and ROC curve for both models. Additionally, based on the feature importance chart, PageValues was identified as the most influential feature among all.




After applying oversampling, I observed a significant improvement in prediction accuracy. Compared to the results without oversampling, the model showed a noticeable enhancement in predicting label 1 (i.e., users who made a purchase). I believe this improvement is due to the resolution of the data imbalance problem — the number of samples in class 1 increased substantially, allowing the model to learn better and thus produce more accurate predictions.

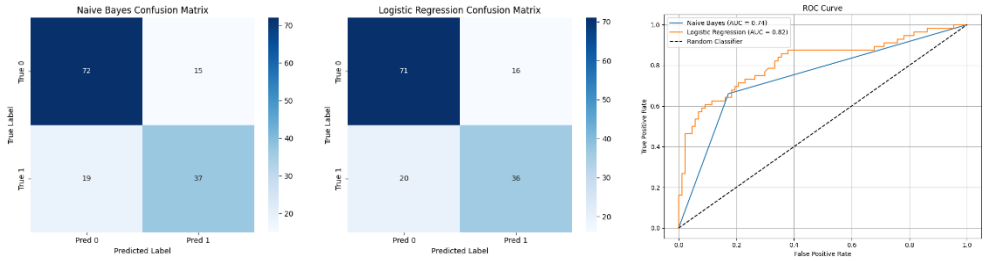


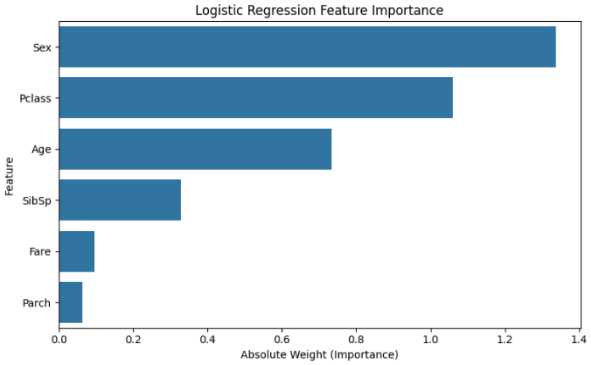
The prediction results are summarized in the table below. NB represents Naive Bayes, LR stands for Logistic Regression, NB_over refers to Naive Bayes with oversampling, and LR_over refers to Logistic Regression with oversampling.

| Model_name | Accuracy | AUC | f1-score | Support(1, 0) |
|---|---|---|---|---|
| NB | 0.7908 | 0.8217 | 0.5204 | (2055, 411) |
| LR | 0.8686 | 0.8813 | 0.4636 | (2055, 411) |
| NB_over | 0.7544 | 0.8339 | 0.7748 | (2081, 2088) |
| LR_over | 0.8208 | 0.8962 | 0.8072 | (2081, 2088) |

**ii. Titanic Survival Prediction dataset**

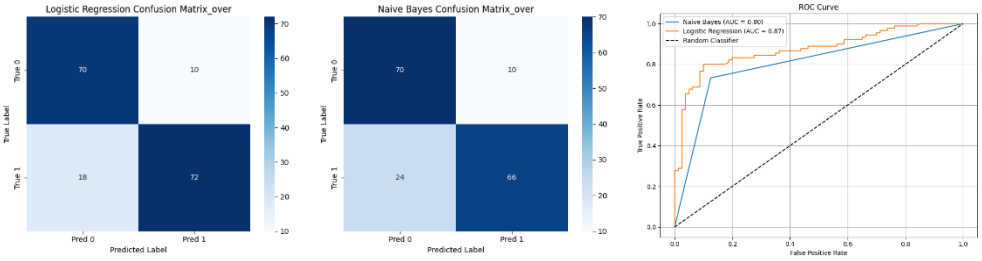
In this dataset experiment, I found that the prediction performance of the Logistic Regression model was better than that of the Naïve Bayes model. The ROC curve of the Naïve Bayes model showed a noticeable sharp corner, which I believe is an interesting and potentially problematic characteristic of this model's prediction behavior. Based on the results from the Logistic Regression model, I also found that the “Sex” feature had a significant impact on the prediction outcome — indicating that females had a higher chance of survival.




This dataset still suffers from a significant class imbalance problem, so I applied oversampling to address it by balancing the number of survivors and non-survivors. After applying oversampling, I observed a notable improvement

in the overall prediction results, especially in the model's ability to correctly predict survivors.



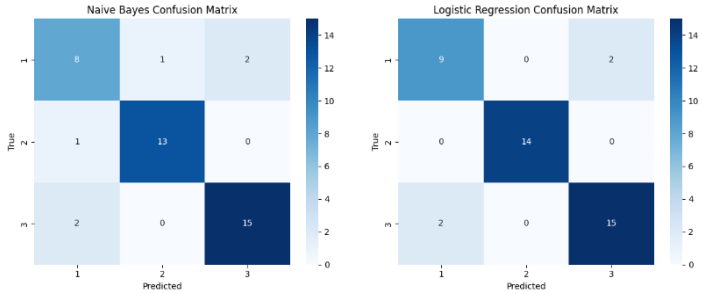

The prediction results are summarized in the table below.

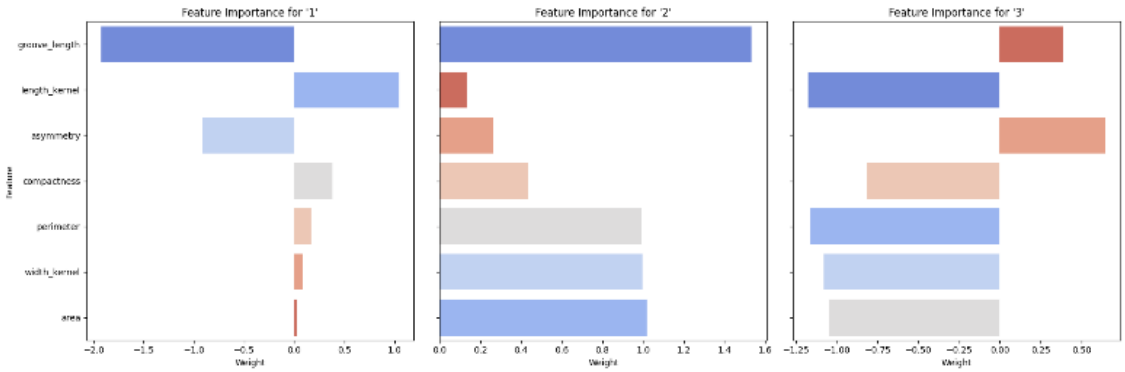
| Model_name | Accuracy | AUC | f1-score | Support |
|---|---|---|---|---|
| NB | 0.7622 | 0.7442 | 0.6852 | 143 |
| LR | 0.7483 | 0.8162 | 0.6667 | 143 |
| NB_over | 0.8000 | 0.8042 | 0.7952 | 170 |
| LR_over | 0.8353 | 0.8749 | 0.8372 | 170 |

**iii. Seeds dataset**

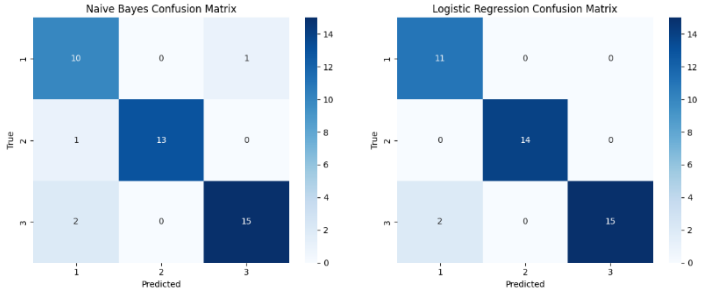
Since this is a multi-class dataset, I did not use ROC curves or AUC to evaluate the prediction results. Based on the results, I found that the Logistic Regression model performed better than the Naïve Bayes model.

In the feature importance analysis, an interesting observation emerged: for class 1 and class 2, the "groove_length" feature had the greatest influence on the prediction. However, for class 3, the most important feature was "length_kernel." I believe this is because "length_kernel" plays a key role in distinguishing class 3 from classes 1 and 2.






In this dataset, I also applied oversampling to address the class imbalance problem. After oversampling, I observed a significant improvement in the prediction performance for class 1. Additionally, the performance gap between the two methods (with and without oversampling) became even more noticeable.




The prediction results are summarized in the table below.

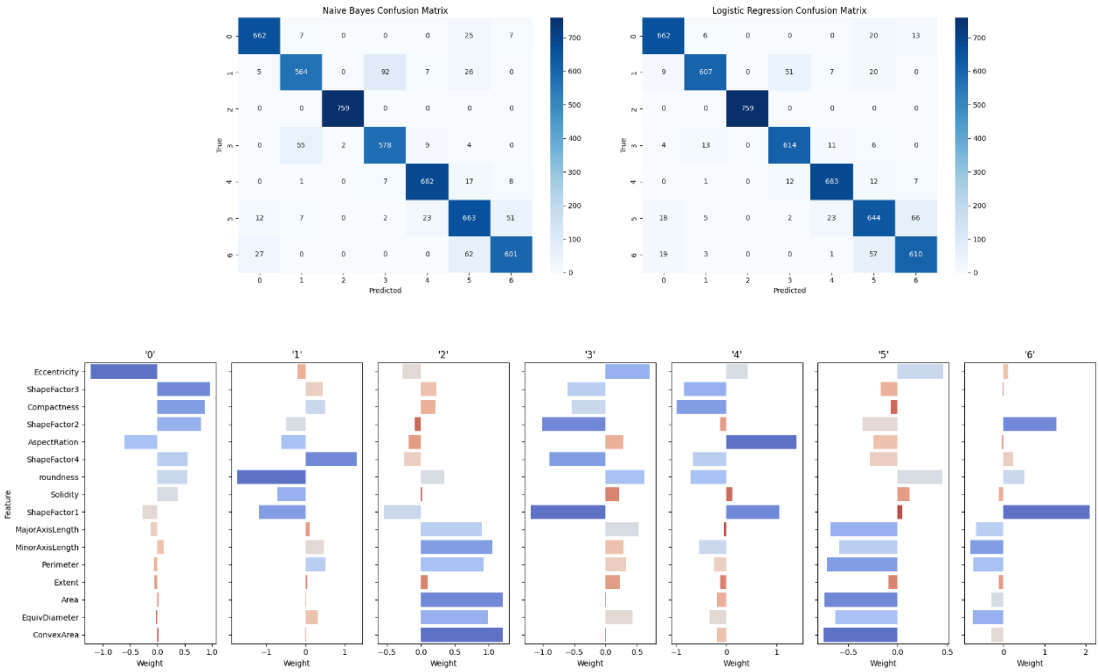
| Model_name | Accuracy | f1-score | Support |
|---|---|---|---|
| NB | 0.8571 | 0.8461 | 42 |
| LR | 0.9048 | 0.9002 | 42 |
| NB_over | 0.8095 | 0.8056 | 42 |
| LR_over | 0.8571 | 0.8591 | 42 |

**iv. Dry_Bean_DataSet**

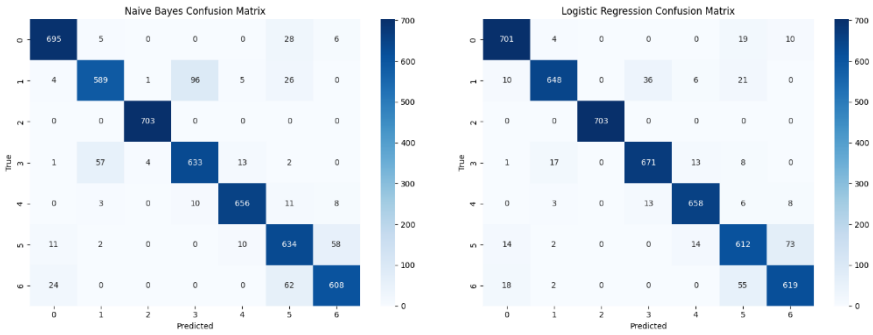
This is the second multi-class dataset, and again, I did not use ROC curves or AUC to evaluate the prediction results due to the nature of the task. In this case, the Logistic Regression model still outperformed the Naïve Bayes model, although the performance gap between the two was relatively small.

According to the results from Logistic Regression and the feature importance chart, there isn't a single dominant feature that can clearly separate all classes. Instead, each class has its own most important feature, and these

features work together to classify the data into seven different classes. From this observation, I concluded that as the number of classes increases, the relationships among features become more complex.




In this dataset, I also applied oversampling to address the data imbalance issue. After oversampling, I observed a significant improvement in prediction performance, particularly in class 0 and class 3.




The prediction results are summarized in the table below.

| Model_name | Accuracy | f1-score | Support |
|---|---|---|---|
| NB | 0.9082 | 0.9069 | 4965 |
| LR | 0.9223 | 0.9220 | 4965 |
| NB_over | 0.9100 | 0.9102 | 4965 |
| LR_over | 0.9289 | 0.9290 | 4965 |

## V. Analysis

In this project, I observed several key findings:

1. The Logistic Regression model consistently outperformed the Naïve Bayes model in terms of prediction performance.

2. In binary classification datasets, there is often a single dominant feature that plays a crucial role in distinguishing the classes. However, in multi-class datasets, each target class tends to have its own most important feature. These features can serve as strong indicators for roughly distinguishing between different classes.

3. Across all four datasets, oversampling significantly improved prediction performance. This demonstrates that data imbalance has a clear negative impact on model accuracy, and addressing it leads to better results.

Overall, the results of this project aligned with my expectations: the Logistic Regression model consistently delivered better performance, and oversampling proved to be an effective technique for improving classification outcomes.

# Appendix

GitHub Link:

https://github.com/Pandamachi/Pattern-Recognition-HW1

## 1. Shoppers Purchasing Intention Dataset Code

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import (
    confusion_matrix, roc_curve, auc,
    accuracy_score, precision_score, recall_score,
    f1_score, classification_report
)
from sklearn.model_selection import train_test_split

# ===== 1. data preprocessing =====
df = pd.read_csv("data.csv")

for col in df.select_dtypes(include='object').columns:
    df[col] = LabelEncoder().fit_transform(df[col])
for col in df.select_dtypes(include='bool').columns:
    df[col] = df[col].astype(int)

# # Over-sampling
# revenue_counts = df['Revenue'].value_counts()
# print("Revenue 分布：")
# print(revenue_counts)

# majority_class = df[df['Revenue'] == revenue_counts.idxmax()]
# minority_class = df[df['Revenue'] == revenue_counts.idxmin()]
# n_to_sample = len(majority_class) - len(minority_class)

# minority_oversampled = minority_class.sample(n=n_to_sample, replace=True, random_state=42)
# df_balanced = pd.concat([df, minority_oversampled], ignore_index=True).sample(frac=1, random_state=42)


X = df.drop("Revenue", axis=1).values
y = df["Revenue"].values
feature_names = df.drop("Revenue", axis=1).columns.tolist()

X = StandardScaler().fit_transform(X)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# ===== 2. Naive Bayes =====
class GaussianNB:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.var = {}
        self.priors = {}
        for c in self.classes:
            X_c = X[y == c]
            self.mean[c] = X_c.mean(axis=0)
            self.var[c] = X_c.var(axis=0) + 1e-9
            self.priors[c] = X_c.shape[0] / X.shape[0]

    def _pdf(self, class_idx, x):
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp(- (x - mean) ** 2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator

    def predict_proba(self, X):
        probs = []
        for x in X:
            class_probs = []
            for c in self.classes:
                prior = np.log(self.priors[c])
                cond = np.sum(np.log(self._pdf(c, x)))
                class_probs.append(prior + cond)
            probs.append(np.exp(class_probs) / np.sum(np.exp(class_probs)))
        return np.array(probs)

    def predict(self, X):
        return np.argmax(self.predict_proba(X), axis=1)
```

```python
# ===== 3. Logistic Regression =====
class LogisticRegression:
    def __init__(self, lr=0.1, n_iter=1000):
        self.lr = lr
        self.n_iter = n_iter

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        self.W = np.zeros(X.shape[1])
        self.b = 0
        for _ in range(self.n_iter):
            linear = np.dot(X, self.W) + self.b
            y_pred = self.sigmoid(linear)
            dw = np.dot(X.T, (y_pred - y)) / len(y)
            db = np.sum(y_pred - y) / len(y)
            self.W -= self.lr * dw
            self.b -= self.lr * db

    def predict_proba(self, X):
        return self.sigmoid(np.dot(X, self.W) + self.b)

    def predict(self, X):
        return (self.predict_proba(X) >= 0.5).astype(int)
```

```python
# ===== 4. Confusion Matrix =====
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
                xticklabels=["Pred 0", "Pred 1"],
                yticklabels=["True 0", "True 1"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(f"{title}.png")
    plt.close()
```

```python
# ===== 5. evulation module =====
def evaluate_and_save(name, model, X_val, y_val, probs=None):
    y_pred = model.predict(X_val)
    if probs is None:
        probs = model.predict_proba(X_val)

    cm_title = f"{name} - Confusion_Matrix"
    plot_confusion_matrix(y_val, y_pred, cm_title)
    cm_path = f"{cm_title}.png"

    fpr, tpr, _ = roc_curve(y_val, probs)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"{name} - ROC Curve")
    plt.legend()
    roc_path = f"{name}_roc_curve.png"
    plt.savefig(roc_path)
    plt.close()

    acc = accuracy_score(y_val, y_pred)
    prec = precision_score(y_val, y_pred)
    rec = recall_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred)
    support_0 = sum(y_val == 0)
    support_1 = sum(y_val == 1)

    print(f"\n{name} Classification Report:")
    print(classification_report(y_val, y_pred, target_names=["Class 0", "Class 1"]))

    return acc, roc_auc, f1, (support_0, support_1), cm_path, roc_path
```

```python
# ===== 6. output =====
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_acc, nb_auc, nb_f1, nb_support, nb_cm_path, nb_roc_path = evaluate_and_save(
    "NaiveBayes", nb_model, X_val, y_val, nb_model.predict_proba(X_val)[:, 1]
)

lr_model = LogisticRegression(lr=0.1, n_iter=1000)
lr_model.fit(X_train, y_train)
lr_acc, lr_auc, lr_f1, lr_support, lr_cm_path, lr_roc_path = evaluate_and_save(
    "LogisticRegression", lr_model, X_val, y_val
)
```

```python
# ===== 7. Feature Importance =====
abs_weights = np.abs(lr_model.W)
sorted_idx = np.argsort(abs_weights)[::-1]
top_features = np.array(feature_names)[sorted_idx]
top_weights = abs_weights[sorted_idx]

plt.figure(figsize=(10, 6))
plt.barh(top_features[:10][::-1], top_weights[:10][::-1], color='skyblue')
plt.xlabel("Absolute Weight")
plt.title("Top 10 Influential Features (Logistic Regression)")
plt.tight_layout()
plt.savefig("feature_importance_logistic_regression.png")
plt.close()

most_influential_feature = top_features[0]
print("\n===== Summary =====")
print(f"Naive Bayes - Accuracy: {nb_acc:.4f}, AUC: {nb_auc:.4f}, F1: {nb_f1:.4f}, Support: {nb_support}")
print(f"Logistic Reg - Accuracy: {lr_acc:.4f}, AUC: {lr_auc:.4f}, F1: {lr_f1:.4f}, Support: {lr_support}")
print(f"Most Influential Feature: {most_influential_feature}")

def plot_combined_roc(y_val, nb_probs, lr_probs):
    fpr_nb, tpr_nb, _ = roc_curve(y_val, nb_probs)
    auc_nb = auc(fpr_nb, tpr_nb)

    fpr_lr, tpr_lr, _ = roc_curve(y_val, lr_probs)
    auc_lr = auc(fpr_lr, tpr_lr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr_nb, tpr_nb, label=f"Naive Bayes (AUC = {auc_nb:.2f})")
    plt.plot(fpr_lr, tpr_lr, label=f"Logistic Regression (AUC = {auc_lr:.2f})")
    plt.plot([0, 1], [0, 1], 'k--', label="Random Classifier")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve Comparison")
    plt.legend()
    plt.tight_layout()
    plt.savefig("combined_roc_curve.png")
    plt.close()
```

```python
plot_combined_roc(
    y_val,
    nb_model.predict_proba(X_val)[:, 1],
    lr_model.predict_proba(X_val)
)
```

## 2. Titanic Survival Prediction dataset

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.utils import resample
from sklearn.metrics import accuracy_score, f1_score

# === 1. Data Preprocessing ===
df = pd.read_csv("train_cleaned.csv")
df = df.drop(columns=["PassengerId"])

df_majority = df[df.Survived == 0]
df_minority = df[df.Survived == 1]

# df_minority_upsampled = resample(
#     df_minority,
#     replace=True,
#     n_samples=len(df_majority),
#     random_state=42
# )

# df_balanced = pd.concat([df_majority, df_minority_upsampled])

X = df.drop("Survived", axis=1)
y = df["Survived"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
# === 2. Naive Bayes ===
class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = {}
        self.stds = {}
        self.priors = {}

        for c in self.classes:
            X_c = X[y == c]
            self.means[c] = X_c.mean(axis=0)
            self.stds[c] = X_c.std(axis=0) + 1e-6
            self.priors[c] = X_c.shape[0] / X.shape[0]

    def _calculate_likelihood(self, mean, std, x):
        exponent = np.exp(-0.5 * ((x - mean) / std) ** 2)
        return (1 / (np.sqrt(2 * np.pi) * std)) * exponent

    def _calculate_posterior(self, x):
        posteriors = []
        for c in self.classes:
            prior = np.log(self.priors[c])
            class_conditional = np.sum(np.log(self._calculate_likelihood(self.means[c], self.stds[c], x)))
            posterior = prior + class_conditional
            posteriors.append(posterior)
        return self.classes[np.argmax(posteriors)]

    def predict(self, X):
        return np.array([self._calculate_posterior(x) for x in X])


nb_model = GaussianNaiveBayes()
nb_model.fit(X_train_scaled, y_train)
y_pred_nb = nb_model.predict(X_test_scaled)
```

```python
# === 3. Logistic Regression ===
class LogisticRegressionScratch:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        self.m, self.n = X.shape
        self.weights = np.zeros(self.n)
        self.bias = 0

        for _ in range(self.epochs):
            linear_model = np.dot(X, self.weights) + self.bias
            y_pred = self.sigmoid(linear_model)
            dw = (1 / self.m) * np.dot(X.T, (y_pred - y))
            db = (1 / self.m) * np.sum(y_pred - y)
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict_proba(self, X):
        return self.sigmoid(np.dot(X, self.weights) + self.bias)

    def predict(self, X):
        return (self.predict_proba(X) >= 0.5).astype(int)


logreg_model = LogisticRegressionScratch(lr=0.1, epochs=1000)
logreg_model.fit(X_train_scaled, y_train)
y_pred_lr = logreg_model.predict(X_test_scaled)
y_proba_lr = logreg_model.predict_proba(X_test_scaled)
y_proba_nb = [1 if p == 1 else 0 for p in y_pred_nb]  # Naive Bayes is hard prediction only
```

```python
# === 4. Evaluation module ===
# Confusion Matrix & Classification Report
print("=== Naive Bayes ===")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))

print("\n=== Logistic Regression ===")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))

# ROC Curve
fpr_nb, tpr_nb, _ = roc_curve(y_test, y_proba_nb)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_proba_lr)
auc_nb = auc(fpr_nb, tpr_nb)
auc_lr = auc(fpr_lr, tpr_lr)

plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, label=f"Naive Bayes (AUC = {auc_nb:.2f})")
plt.plot(fpr_lr, tpr_lr, label=f"Logistic Regression (AUC = {auc_lr:.2f})")
plt.plot([0, 1], [0, 1], 'k--', label="Random Classifier")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("roc_curve.png")
plt.show()
```

```python
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
                xticklabels=["Pred 0", "Pred 1"],
                yticklabels=["True 0", "True 1"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(f"{title}.png")
    plt.show()

plot_confusion_matrix(y_test, y_pred_nb, "Naive Bayes Confusion Matrix")
plot_confusion_matrix(y_test, y_pred_lr, "Logistic Regression Confusion Matrix")


# Feature Importance for Logistic Regression
def plot_feature_importance(weights, feature_names, title):
    importance = np.abs(weights)
    sorted_idx = np.argsort(importance)[::-1]

    plt.figure(figsize=(8, 5))
    sns.barplot(x=importance[sorted_idx], y=np.array(feature_names)[sorted_idx])
    plt.title(title)
    plt.xlabel("Absolute Weight (Importance)")
    plt.ylabel("Feature")
    plt.tight_layout()
    plt.savefig(f"{title}.png")
    plt.show()
```

```python
feature_names = X.columns
plot_feature_importance(logreg_model.weights, feature_names, "Logistic Regression Feature Importance")

def print_metrics(y_true, y_pred, y_proba, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    auc_score = auc(*roc_curve(y_true, y_proba)[:2])
    support = len(y_true)

    print(f"\n=== {model_name} Performance Summary ===")
    print(f"Accuracy : {accuracy:.4f}")
    print(f"AUC      : {auc_score:.4f}")
    print(f"F1-score : {f1:.4f}")
    print(f"Support  : {support}")

print_metrics(y_test, y_pred_nb, y_proba_nb, "Naive Bayes")
print_metrics(y_test, y_pred_lr, y_proba_lr, "Logistic Regression")
```

## 3. Seeds dataset

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report

# 1. Data Loading
df = pd.read_csv("seed.csv")
df.columns = [
    "area", "perimeter", "compactness", "length_kernel", "width_kernel",
    "asymmetry", "groove_length", "class"
]

# # 2. Oversampling
# max_count = df["class"].value_counts().max()
# df_oversampled = pd.DataFrame()

# for label in df["class"].unique():
#     subset = df[df["class"] == label]
#     upsampled = resample(subset, replace=True, n_samples=max_count, random_state=42)
#     df_oversampled = pd.concat([df_oversampled, upsampled])

# df = df_oversampled.sample(frac=1, random_state=42).reset_index(drop=True)  # 打亂

X = df.drop(columns=["class"])
y = df["class"]

X_train, X_test, y_train_raw, y_test_raw = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

le = LabelEncoder()
y_train = le.fit_transform(y_train_raw)
y_test = le.transform(y_test_raw)
```

```python
# 3. Naive Bayes
class GaussianNaiveBayesMulti:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = {}
        self.stds = {}
        self.priors = {}

        for c in self.classes:
            X_c = X[y == c]
            self.means[c] = X_c.mean(axis=0)
            self.stds[c] = X_c.std(axis=0) + 1e-6
            self.priors[c] = len(X_c) / len(X)

    def _likelihood(self, mean, std, x):
        exponent = np.exp(-0.5 * ((x - mean) / std) ** 2)
        return (1 / (np.sqrt(2 * np.pi) * std)) * exponent

    def _posterior(self, x):
        posteriors = []
        for c in self.classes:
            prior = np.log(self.priors[c])
            cond = np.sum(np.log(self._likelihood(self.means[c], self.stds[c], x)))
            posteriors.append(prior + cond)
        return self.classes[np.argmax(posteriors)]

    def predict(self, X):
        return np.array([self._posterior(x) for x in X])

nb_model = GaussianNaiveBayesMulti()
nb_model.fit(X_train_scaled, y_train)
y_pred_nb = nb_model.predict(X_test_scaled)
```

```python
# 4. Logistic Regression (Softmax)
class LogisticRegressionScratchMulti:
    def __init__(self, lr=0.1, epochs=1000):
        self.lr = lr
        self.epochs = epochs

    def _softmax(self, z):
        z_exp = np.exp(z - np.max(z, axis=1, keepdims=True))
        return z_exp / np.sum(z_exp, axis=1, keepdims=True)

    def fit(self, X, y):
        self.m, self.n = X.shape
        self.k = len(np.unique(y))
        self.weights = np.zeros((self.n, self.k))
        self.bias = np.zeros(self.k)

        y_onehot = np.eye(self.k)[y]

        for _ in range(self.epochs):
            logits = np.dot(X, self.weights) + self.bias
            probs = self._softmax(logits)
            error = probs - y_onehot

            dw = np.dot(X.T, error) / self.m
            db = np.sum(error, axis=0) / self.m

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        logits = np.dot(X, self.weights) + self.bias
        probs = self._softmax(logits)
        return np.argmax(probs, axis=1)

    def predict_proba(self, X):
        logits = np.dot(X, self.weights) + self.bias
        return self._softmax(logits)

logreg_model = LogisticRegressionScratchMulti()
logreg_model.fit(X_train_scaled, y_train)
y_pred_lr = logreg_model.predict(X_test_scaled)
```

```python
# 5. Confusion Matrix
def plot_conf_matrix(y_true, y_pred, labels, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(f"{title}.png")
    plt.show()

class_labels = le.classes_.astype(str)
plot_conf_matrix(y_test, y_pred_nb, class_labels, "Naive Bayes Confusion Matrix")
plot_conf_matrix(y_test, y_pred_lr, class_labels, "Logistic Regression Confusion Matrix")

print("=== Naive Bayes ===")
print(classification_report(y_test, y_pred_nb, target_names=class_labels))

print("\n=== Logistic Regression ===")
print(classification_report(y_test, y_pred_lr, target_names=class_labels))
```

```python
# 6. Feature Importance
def plot_all_class_feature_importance(weights, feature_names, class_labels):
    num_classes = weights.shape[1]
    fig, axes = plt.subplots(1, num_classes, figsize=(18, 6), sharey=True)

    for i in range(num_classes):
        importance = weights[:, i]
        sorted_idx = np.argsort(np.abs(importance))[::-1]

        sns.barplot(
            x=importance[sorted_idx],
            y=np.array(feature_names)[sorted_idx],
            ax=axes[i],
            palette="coolwarm"
        )
        axes[i].set_title(f"Feature Importance for '{class_labels[i]}'")
        axes[i].set_xlabel("Weight")
        if i == 0:
            axes[i].set_ylabel("Feature")
        else:
            axes[i].set_ylabel("")

    plt.tight_layout()
    plt.savefig("feature_importance.png")
    plt.show()

plot_all_class_feature_importance(logreg_model.weights, X.columns, class_labels)
```

```python
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

def print_multiclass_metrics(y_true, y_pred, y_proba, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    f1_macro = f1_score(y_true, y_pred, average='macro')
    support = len(y_true)

    try:
        auc_score = roc_auc_score(y_true, y_proba, multi_class='ovr')
    except:
        auc_score = "N/A (probabilities not available)"

    print(f"\n=== {model_name} Performance Summary ===")
    print(f"Accuracy : {accuracy:.4f}")
    print(f"F1-score : {f1_macro:.4f} (macro average)")
    print(f"AUC      : {auc_score}")
    print(f"Support  : {support}")

y_test_onehot = np.eye(len(class_labels))[y_test]

print_multiclass_metrics(y_test, y_pred_nb, None, "Naive Bayes")

y_proba_lr = logreg_model.predict_proba(X_test_scaled)
print_multiclass_metrics(y_test, y_pred_lr, y_proba_lr, "Logistic Regression")
```

## 4. Dry_Bean_DataSet

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report

# 1. Dry Bean Dataset
df = pd.read_csv("Dry_Bean_DataSet.csv")

# 2. Oversampling
max_count = df["Class"].value_counts().max()
df_oversampled = pd.DataFrame()

for label in df["Class"].unique():
    subset = df[df["Class"] == label]
    upsampled = resample(subset, replace=True, n_samples=max_count, random_state=42)
    df_oversampled = pd.concat([df_oversampled, upsampled])

df = df_oversampled.sample(frac=1, random_state=42).reset_index(drop=True)  # 打亂

# 3. Data Preprocessing
X = df_oversampled.drop(columns=["Class"])
y = df_oversampled["Class"]

X_train, X_test, y_train_raw, y_test_raw = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

le = LabelEncoder()
y_train = le.fit_transform(y_train_raw)
y_test = le.transform(y_test_raw)
```

```python
# 4. Naive Bayes
class GaussianNaiveBayesMulti:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = {}
        self.stds = {}
        self.priors = {}

        for c in self.classes:
            X_c = X[y == c]
            self.means[c] = X_c.mean(axis=0)
            self.stds[c] = X_c.std(axis=0) + 1e-6
            self.priors[c] = len(X_c) / len(X)

    def _likelihood(self, mean, std, x):
        exponent = np.exp(-0.5 * ((x - mean) / std) ** 2)
        return (1 / (np.sqrt(2 * np.pi) * std)) * exponent

    def _posterior(self, x):
        posteriors = []
        for c in self.classes:
            prior = np.log(self.priors[c])
            likelihood = self._likelihood(self.means[c], self.stds[c], x)
            likelihood = np.clip(likelihood, 1e-9, None)  # 避免 log(0)
            cond = np.sum(np.log(likelihood))
            posteriors.append(prior + cond)
        return self.classes[np.argmax(posteriors)]

    def predict(self, X):
        return np.array([self._posterior(x) for x in X])

nb_model = GaussianNaiveBayesMulti()
nb_model.fit(X_train_scaled, y_train)
y_pred_nb = nb_model.predict(X_test_scaled)
```

(variable) means: dict

```python
# 5. Logistic Regression (Softmax)
class LogisticRegressionScratchMulti:
    def __init__(self, lr=0.1, epochs=1000):
        self.lr = lr
        self.epochs = epochs

    def _softmax(self, z):
        z_exp = np.exp(z - np.max(z, axis=1, keepdims=True))
        return z_exp / np.sum(z_exp, axis=1, keepdims=True)

    def fit(self, X, y):
        self.m, self.n = X.shape
        self.k = len(np.unique(y))
        self.weights = np.zeros((self.n, self.k))
        self.bias = np.zeros(self.k)

        y_onehot = np.eye(self.k)[y]

        for _ in range(self.epochs):
            logits = np.dot(X, self.weights) + self.bias
            probs = self._softmax(logits)
            error = probs - y_onehot

            dw = np.dot(X.T, error) / self.m
            db = np.sum(error, axis=0) / self.m

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        logits = np.dot(X, self.weights) + self.bias
        probs = self._softmax(logits)
        return np.argmax(probs, axis=1)

    def predict_proba(self, X):
        logits = np.dot(X, self.weights) + self.bias
        return self._softmax(logits)

logreg_model = LogisticRegressionScratchMulti()
logreg_model.fit(X_train_scaled, y_train)
y_pred_lr = logreg_model.predict(X_test_scaled)
```

```python
# 6. Confusion Matrix
def plot_conf_matrix(y_true, y_pred, labels, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(f"{title}_over.png")
    plt.show()

class_labels = le.classes_.astype(str)
plot_conf_matrix(y_test, y_pred_nb, class_labels, "Naive Bayes Confusion Matrix")
plot_conf_matrix(y_test, y_pred_lr, class_labels, "Logistic Regression Confusion Matrix")

print("=== Naive Bayes ===")
print(classification_report(y_test, y_pred_nb, target_names=class_labels))

print("\n=== Logistic Regression ===")
print(classification_report(y_test, y_pred_lr, target_names=class_labels))
```

```python
# 7. Feature Importance
def plot_all_class_feature_importance(weights, feature_names, class_labels):
    num_classes = weights.shape[1]
    fig, axes = plt.subplots(1, num_classes, figsize=(20, 6), sharey=True)

    for i in range(num_classes):
        importance = weights[:, i]
        sorted_idx = np.argsort(np.abs(importance))[::-1]

        sns.barplot(
            x=importance[sorted_idx],
            y=np.array(feature_names)[sorted_idx],
            ax=axes[i],
            palette="coolwarm"
        )
        axes[i].set_title(f"'{class_labels[i]}'")
        axes[i].set_xlabel("Weight")
        if i == 0:
            axes[i].set_ylabel("Feature")
        else:
            axes[i].set_ylabel("")

    plt.tight_layout()
    plt.savefig("feature_importance_over.png")
    plt.show()

plot_all_class_feature_importance(logreg_model.weights, X.columns, class_labels)
```

```python
from sklearn.metrics import accuracy_score, f1_score

def print_summary_metrics(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred, average='macro')
    support = len(y_true)

    print(f"\n=== {model_name} Summary ===")
    print(f"Accuracy : {accuracy:.4f}")
    print(f"F1-score : {f1:.4f} (macro average)")
    print(f"Support  : {support}")

print_summary_metrics(y_test, y_pred_nb, "Naive Bayes")
print_summary_metrics(y_test, y_pred_lr, "Logistic Regression")
```