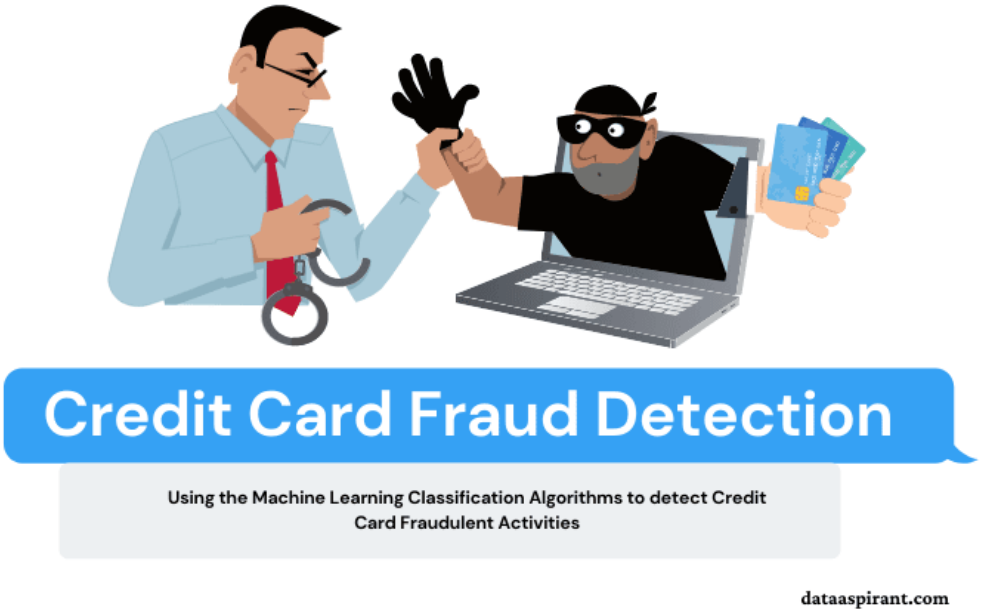
信用卡詐欺偵測

Credit Card Fraud Detection



製作人：甯文駿

摘要

本作品旨在透過機器學習模型偵測高度不平衡的**信用卡詐欺交易資料**，該資料集包含 **284,807 筆交易**，其中僅有 **492 筆為詐欺交易（佔 0.172%）**。考量資料不平衡，模型主要以**精確度-召回率曲線下面積（AUPRC）進行評估**。經過初始模型訓練後，**XGBoost 展現最佳表現，在 ROC 曲線下面積（AUC）達 0.974，平均精確率（AP）達 0.878**。為進一步優化少數類別的偵測能力，使用了多種 SMOTE 採樣技術，其中 **SMOTEENN 表現最佳，成功將 XGBoost 模型的 AUC 從 0.9743 提升至 0.9873，並將詐欺交易的召回率從 0.81 提高到 0.86**，雖然精確率略有下降，但顯著提升了模型對詐欺的捕捉能力。最終，透過 SHAP 特徵重要性分析，發現 **V14 是區分詐欺樣本的關鍵特徵，其對詐欺樣本的平均影響力（SHAP 值）顯著高於非詐欺樣本，其次為 V4**。

後續研究方向可從**模型超參數調優、探索其他進階的採樣或不平衡處理技術、採用 Ensemble 方法等等**，來進一步優化模型的性能。

資料來源

Credit Card Fraud Detection

連結：
https://storage.googleapis.com/download.tensorflow.org/data/creditcard.csv

此資料集包含歐洲持卡人在 2013 年 9 月使用信用卡進行的交易。 該資料集呈現了兩天內發生的交易，在 **284,807 筆交易**中，我們發現了 **492 筆詐欺交易**。此資料集高度不平衡，**正類（詐欺）交易占所有交易的 0.172%**。

特徵 V1、V2、…V28 是透過 PCA 獲得的主成分，唯一未經過 PCA 轉換的特徵是「時間」和「金額」。特徵「時間」包含資料集中每筆交易與第一筆交易之間相隔的秒數。特徵「金額」是交易金額，此特徵可用於依賴範例的成本敏感學習。特徵「類別」是回應變量，在**有詐欺的情況下取值為 1，否則取值為 0**。

考慮到**類別不平衡率**，建議使用**精確度-召回率曲線下面積**（AUPRC）來衡量準確率。對於不平衡的分類，**混淆矩陣**準確率意義不大。

Credit Card Fraud Detection.ipynb

1. 讀取與確認資料

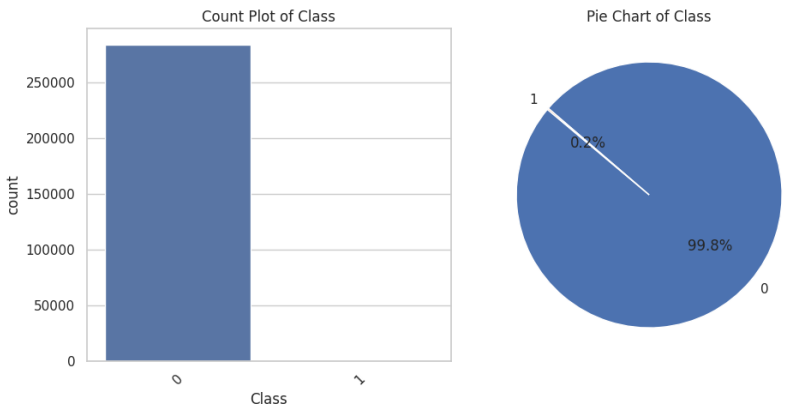
先讀取資料可看到資料集的內容為 284807 rows × 30 columns,檔案大小約為 67.4 MB,且資料完整並無缺失值的存在。

```
df = pd.read_csv('https://storage.googleapis.com/download.tensorflow.org/data/creditcard.csv')
df
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	0.0	-1.36	-0.07	2.54	1.38	-0.34	0.46	0.24	0.10	0.36	0.09	-0.55	-0.62	-0.99	-0.31	1.47	-0.47	0.21	0.03	0.40	2.51e-01	-1.83e-02	2.78e-01	-0.11	6.69e-02	0.13	-0.19	1.34e-01	-0.02	149.62
1	0.0	1.19	0.27	0.17	0.45	0.06	-0.08	-0.08	0.09	-0.26	-0.17	1.61	1.07	0.49	-0.14	0.64	0.46	-0.11	-0.18	-0.15	-6.91e-02	-2.26e-01	-6.39e-01	0.10	-3.40e-01	0.17	0.13	-8.98e-03	0.01	2.69
2	1.0	-1.36	-1.34	1.77	0.38	-0.50	1.80	0.79	0.25	-1.51	0.21	0.62	0.07	0.72	-0.17	2.35	-2.89	1.11	-0.12	-2.26	5.25e-01	2.48e-01	7.72e-01	0.91	-6.89e-01	-0.33	-0.14	-5.54e-02	-0.06	378.66
3	1.0	-0.97	-0.19	1.79	-0.86	-0.01	1.25	0.24	0.38	-1.39	-0.05	-0.23	0.18	0.51	-0.29	-0.63	-1.06	-0.68	1.97	-1.23	-2.08e-01	-1.08e-01	5.27e-03	-0.19	-1.18e+00	0.65	-0.22	6.27e-02	0.06	123.50
4	2.0	-1.16	0.88	1.55	0.40	-0.41	0.10	0.59	-0.27	0.82	0.75	-0.82	0.54	1.35	-1.12	0.18	-0.45	-0.24	-0.04	0.80	4.09e-01	-9.43e-03	7.98e-01	-0.14	1.41e-01	-0.21	0.50	2.19e-01	0.22	69.99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.88	10.07	-9.83	-2.07	-5.36	-2.61	-4.92	7.31	1.91	4.36	-1.59	2.71	-0.69	4.63	-0.92	1.11	1.99	0.51	-0.68	1.48e+00	2.13e-01	1.12e-01	1.01	-5.09e-01	1.44	0.25	9.44e-01	0.82	0.77
284803	172787.0	-0.73	-0.06	2.04	-0.74	0.87	1.06	0.02	0.29	0.58	-0.98	-0.15	0.92	1.21	-0.68	1.16	-0.71	-0.03	-1.22	-1.55	5.96e-02	2.14e-01	9.24e-01	0.01	-1.02e+00	-0.61	-0.40	6.85e-02	-0.05	24.79
284804	172788.0	1.92	-0.30	-3.25	-0.56	2.63	3.03	-0.30	0.71	0.43	-0.48	0.41	0.06	-0.18	-0.51	1.33	0.14	0.31	0.40	-0.58	1.40e-03	2.32e-01	5.78e-01	-0.04	6.40e-01	0.27	-0.09	4.45e-03	-0.03	67.88
284805	172788.0	-0.24	0.53	0.70	0.69	-0.38	0.62	-0.69	0.68	0.39	-0.40	-1.93	-0.96	-1.04	0.45	1.96	-0.61	0.51	1.11	2.90	1.27e-01	2.65e-01	8.00e-01	-0.16	1.23e-01	-0.57	0.55	1.09e-01	0.10	10.00
284806	172792.0	-0.53	-0.19	0.70	-0.51	-0.01	-0.65	1.58	-0.41	0.49	-0.92	-1.04	-0.03	-0.19	-0.08	0.04	-0.30	-0.66	0.17	-0.26	3.83e-01	2.61e-01	6.43e-01	0.38	8.80e-03	-0.47	-0.82	-2.42e-03	0.01	217.00

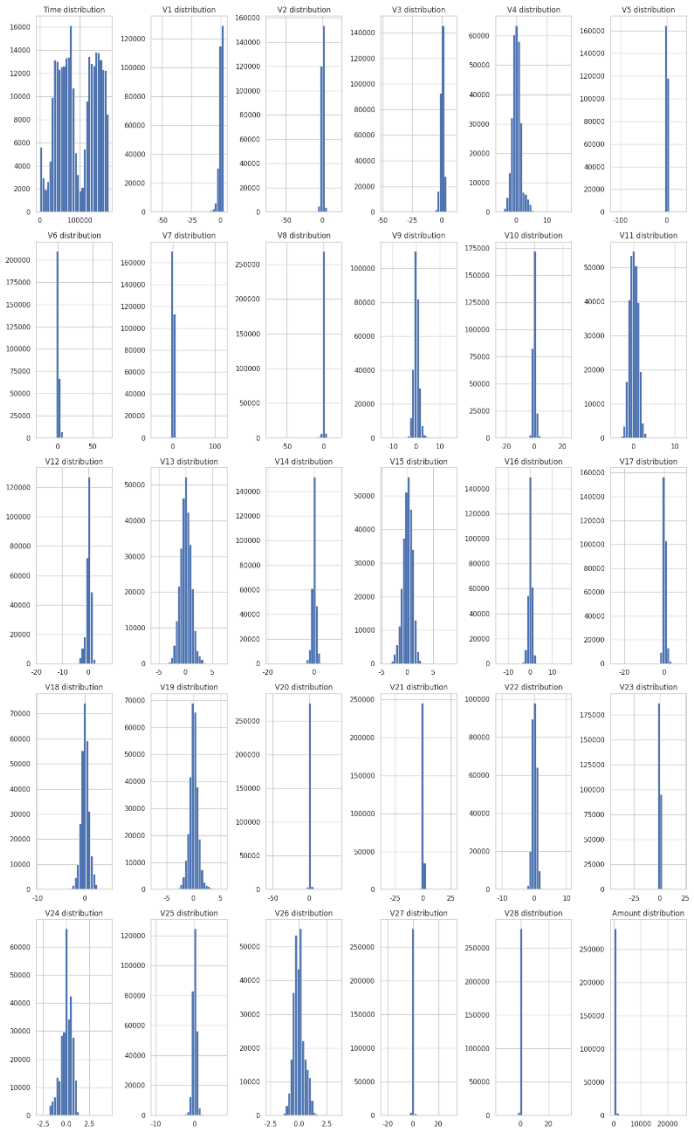
284807 rows × 31 columns

```
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```



Class

- 0：284315
- 1：492



2. 資料分割

- **訓練集：80%**
- **測試集：20%**
- random_state=42：設定隨機種子以便結果可重現
- stratify=y：表示在拆分時保持標籤的比例與原始資料一致

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42 ,stratify=y)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(pd.Series(y_train).value_counts())
print(pd.Series(y_train).value_counts(normalize=True))
print('='*40)
print(pd.Series(y_test).value_counts())
print(pd.Series(y_test).value_counts(normalize=True))
```

```
(227845, 30) (56962, 30) (227845,) (56962,)
Class
0    227451
1       394
Name: count, dtype: int64
Class
0    9.98e-01
1    1.73e-03
Name: proportion, dtype: float64
========================================
Class
0    56864
1       98
Name: count, dtype: int64
Class
0    9.98e-01
1    1.72e-03
Name: proportion, dtype: float64
```

3. 設計評估指標 def() 函式

def plot_combined_roc_pr_curves(models, X_test, y_test)

作用：

對多個已訓練好的分類模型，在同一圖中繪製其測試集上的 ROC **曲線**與 Precision-Recall **曲線**，用以比較各模型在二元分類問題中的性能表現（AUC 和 AUPRC）。

```python
# 各 模型的 ROC 曲線 和 PR 曲線
def plot_combined_roc_pr_curves(models, X_test, y_test):
    plt.figure(figsize=(12, 5))

    # ═══ ROC 曲線 ═══
    plt.subplot(1, 2, 1)
    for name, model in models.items():
        y_score = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_score)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.3f})')

    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve (All Models)')
    plt.legend(loc='lower right')
    plt.grid()

    # ═══ PR 曲線 ═══
    plt.subplot(1, 2, 2)
    for name, model in models.items():
        y_score = model.predict_proba(X_test)[:, 1]
        precision, recall, _ = precision_recall_curve(y_test, y_score)
        avg_precision = average_precision_score(y_test, y_score)
        plt.plot(recall, precision, lw=2, label=f'{name} (AP = {avg_precision:.3f})')

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve (All Models)')
    plt.legend(loc='best')
    plt.grid()
    plt.tight_layout()
    plt.show()
```

def evaluate_model(model, X_train, y_train, X_test, y_test)

作用：

對輸入的分類模型在**訓練集**與**測試集**上進行評估，輸出**分類報告**（Precision, Recall, F1）、AUC 分數，並繪製**混淆矩陣**，快速了解模型性能與預測表現。

```python
# 分類報告和混淆矩陣
def evaluate_model(model, X_train, y_train, X_test, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    print("Train Set:")
    print(classification_report(y_train, y_train_pred))
    print("Test Set:")
    print(classification_report(y_test, y_test_pred))

    # 計算 AUC（二元分類）
    train_auc = roc_auc_score(y_train, model.predict_proba(X_train)[:, 1])
    test_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])

    print(f"Train AUC: {train_auc:.6f}")
    print(f"Test AUC: {test_auc:.6f}")

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    sns.heatmap(confusion_matrix(y_train, y_train_pred), annot=True, cmap='Blues', fmt='d', cbar=False)
    plt.title('Confusion Matrix (Train Set)')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.subplot(1, 2, 2)
    sns.heatmap(confusion_matrix(y_test, y_test_pred), annot=True, cmap='Blues', fmt='d', cbar=False)
    plt.title('Confusion Matrix (Test Set)')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.tight_layout()
    plt.show()
evaluate_model(xgb_model, X_train, y_train, X_test, y_test)
```

def plot_learning_curve(estimator, X, y, cv=5, train

sizes=np.linspace(0.1, 1.0, 10))

作用：

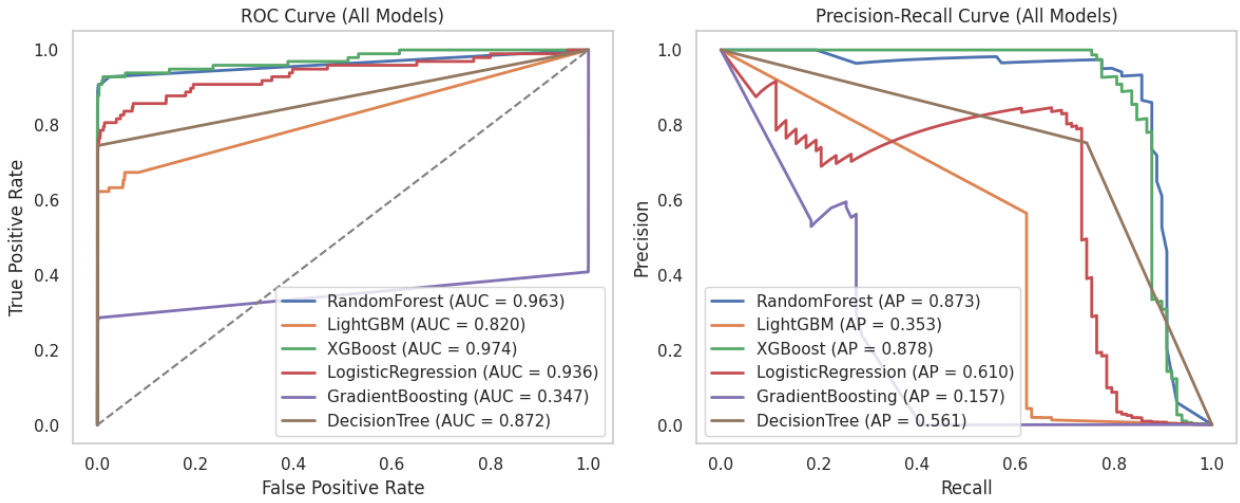
根據模型的訓練結果，繪製**學習曲線**圖，展示**模型在不同訓練樣本數下的訓練與交叉驗證表現**，藉此**評估模型是否欠擬合或過擬合**，並協助調整模型結構與資料大小。

```python
# 學習曲線
def plot_learning_curve(estimator, X, y, cv=5, train_sizes=np.linspace(0.1, 1.0, 10)):
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, train_sizes=train_sizes, n_jobs=-1)
    # Calculate the mean and standard deviation of training and test scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    # Plot the learning curve
    plt.figure(figsize=(8, 4))
    plt.title("Learning Curve")
    plt.xlabel("Training Examples")
    plt.ylabel("Score")
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color="g")

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training Score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation Score")
    plt.legend(loc="best")
    plt.show()

    # Print additional information
    print("Train Sizes:", train_sizes)
    print("Train Scores Mean:", train_scores_mean)
    print("Test Scores Mean:", test_scores_mean)
plot_learning_curve(xgb_model, X_train, y_train)
```

4. 設計迴圈，個別模型訓練

使用 XGBoost、RandomForest、LogisticRegression、LightGBM、DecisionTree 來做**初始模型**的訓練，並針做表現最好的模型，做後續的採樣、特徵分析等等。

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
import lightgbm as lgb
import xgboost as xgb

models = {
        "RandomForest": RandomForestClassifier(random_state=42),
        "LightGBM": lgb.LGBMClassifier(random_state=42, verbose=-1),
        "XGBoost": xgb.XGBClassifier(random_state=42),
        "LogisticRegression": LogisticRegression(),
        "DecisionTree": DecisionTreeClassifier(random_state=42),
}

for name, model in models.items():
        print(f"\n===== 訓練 {name} 模型 =====")
        model.fit(X_train, y_train)
        evaluate_model(model, X_train, y_train, X_test, y_test)
print('='*50)
plot_combined_roc_pr_curves(models, X_test, y_test)
```



左圖：ROC Curve

- **橫軸：** False Positive Rate（偽陽性率）
- **縱軸：** True Positive Rate（真正率）
- **意義：** 越接近左上角的曲線表示模型表現越好。
- **AUC（曲線下面積）：值越接近 1 越好**，代表模型越能區分正負類。

各模型 AUC 表現：

- XGBoost：0.974（表現最佳）
- RandomForest：0.963
- LogisticRegression：0.936
- DecisionTree：0.872

- LightGBM：0.820（表現最差）

右圖：Precision-Recall Curve

- **橫軸**： Recall（召回率 / 真陽性率）
- **縱軸**： Precision（精確率）
- **意義**： 曲線越往右上方延伸，代表在召回更多正樣本的同時仍維持高精確率，模型越優秀。

AP（平均精確率）： 越高越好。

各模型 AP 表現：

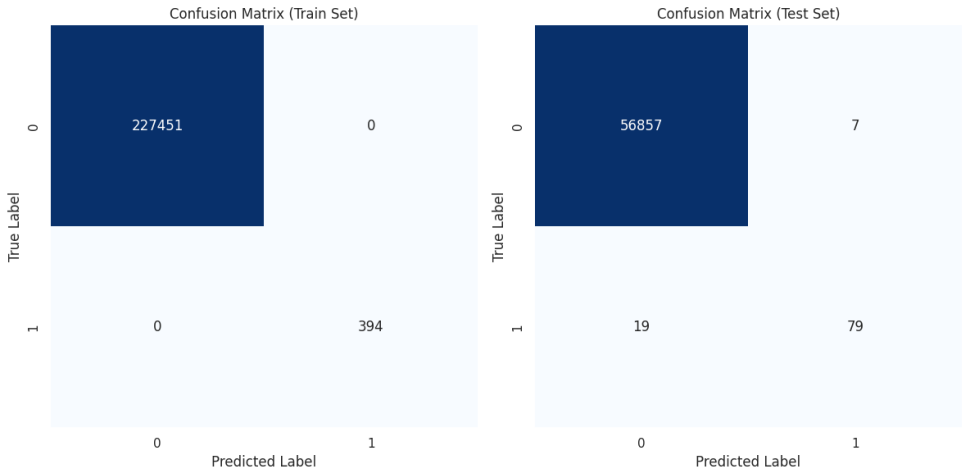
- XGBoost：0.878（表現最佳）
- RandomForest：0.873
- LogisticRegression：0.610
- DecisionTree：0.561
- LightGBM：0.353（表現最差）

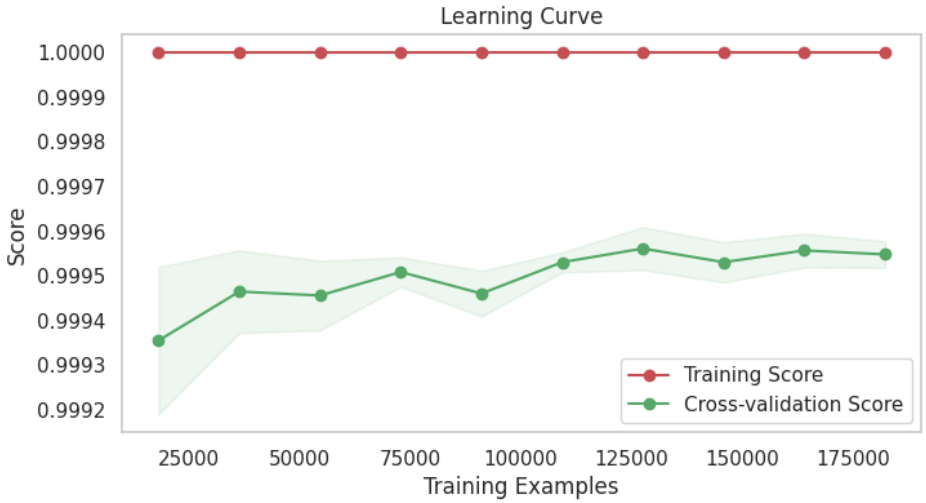
總結：

- **最佳整體表現模型**：XGBoost（在 ROC 和 PR 曲線都表現最佳）
- **次佳模型**：RandomForest
- **表現較差模型**：LightGBM

XGBoost 的分類報告、混淆矩陣、學習曲線

```
Train Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227451
           1       1.00      1.00      1.00       394

    accuracy                           1.00    227845
   macro avg       1.00      1.00      1.00    227845
weighted avg       1.00      1.00      1.00    227845

Test Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.92      0.81      0.86        98

    accuracy                           1.00     56962
   macro avg       0.96      0.90      0.93     56962
weighted avg       1.00      1.00      1.00     56962

Train AUC: 1.000000
Test AUC: 0.974323
```





5 折交叉驗證

使用 Stratified KFold 搭配 XGBoost 模型對二分類資料進行交叉驗證，並計算每個 fold 的準確率和 AUC 分數，再平均匯總結果。

```python
from sklearn.model_selection import StratifiedKFold

# 設定 KFold 的參數
n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

# 儲存每次交叉驗證的結果
fold_accuracies = []
fold_aucs = []
fold_reports = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
        print(f"Fold {fold + 1}")

        X_train_fold, X_val_fold = X.iloc[train_idx], X.iloc[val_idx]
        y_train_fold, y_val_fold = y.iloc[train_idx], y.iloc[val_idx]

        # 建立 XGBoost 模型
        xgb_model = xgb.XGBClassifier(random_state=42)
        xgb_model.fit(X_train_fold, y_train_fold)

        # 預測與評估
        y_val_pred = xgb_model.predict(X_val_fold)
        y_val_pred_proba = xgb_model.predict_proba(X_val_fold)

        acc = accuracy_score(y_val_fold, y_val_pred)

        # 二分類 AUC 計算
        if len(np.unique(y_val_fold)) == 2:
                auc = roc_auc_score(y_val_fold, y_val_pred_proba[:, 1])
        else:
                auc = np.nan

        report = classification_report(y_val_fold, y_val_pred, output_dict=True)

        fold_accuracies.append(acc)
        fold_aucs.append(auc)
        fold_reports.append(report)

        print(classification_report(y_val_fold, y_val_pred))
        print(f"AUC: {auc:.4f}")
        print("=" * 50)

# 顯示整體結果
print(f"\n平均準確率 (Accuracy): {np.mean(fold_accuracies):.4f}")
print(f"平均 AUC: {np.nanmean(fold_aucs):.4f}")
```

```
Fold 1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.95      0.76      0.84        99

    accuracy                           1.00     56962
   macro avg       0.97      0.88      0.92     56962
weighted avg       1.00      1.00      1.00     56962

AUC: 0.9798
==================================================
Fold 2
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       1.00      0.84      0.91        99

    accuracy                           1.00     56962
   macro avg       1.00      0.92      0.96     56962
weighted avg       1.00      1.00      1.00     56962

AUC: 0.9752
==================================================
Fold 3
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.96      0.82      0.88        98

    accuracy                           1.00     56961
   macro avg       0.98      0.91      0.94     56961
weighted avg       1.00      1.00      1.00     56961

AUC: 0.9926
Fold 4
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.96      0.80      0.87        98

    accuracy                           1.00     56961
   macro avg       0.98      0.90      0.94     56961
weighted avg       1.00      1.00      1.00     56961

AUC: 0.9695
==================================================
Fold 5
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.92      0.82      0.86        98

    accuracy                           1.00     56961
   macro avg       0.96      0.91      0.93     56961
weighted avg       1.00      1.00      1.00     56961

AUC: 0.9760
==================================================

平均準確率 (Accuracy): 0.9996
平均 AUC: 0.9786
```

XGBoost 模型在五折交叉驗證中表現非常**穩定**，平均 AUC 高達 0.9786，平均準確率也接近 1.0，說明模型能有效辨識少數類別；但 f1-score 在少數類別（1 類）仍略低於 1，代表模型在少數類別上有小幅漏判的空間。

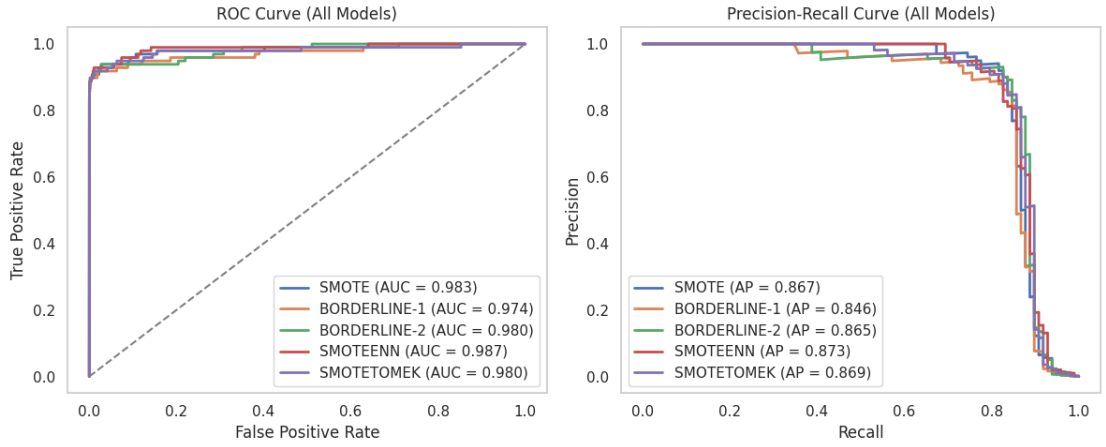
5. 採樣方法

def run_smote_pipeline(smote_method, X_train, y_train, X_test, y_test, model_dict, random_state=42)

作用：

對訓練資料使用**不同的 SMOTE 技術**進行資料平衡，再用 XGBoost 訓練模型，並透過分類報告與 ROC/PR 曲線，比較不同資料增強方法下的模型表現。

```
def run_smote_pipeline(smote_method, X_train, y_train, X_test, y_test, model_dict, random_state=42):
    # 建立重抽樣器
    if smote_method == 'smote':
        sampler = SMOTE(random_state=random_state)
    elif smote_method == 'borderline-1':
        sampler = BorderlineSMOTE(kind='borderline-1', random_state=random_state)
    elif smote_method == 'borderline-2':
        sampler = BorderlineSMOTE(kind='borderline-2', random_state=random_state)
    elif smote_method == 'smoteenn':
        sampler = SMOTEENN(random_state=random_state)
    elif smote_method == 'smotetomek':
        sampler = SMOTETomek(random_state=random_state)
    else:
        raise ValueError("Invalid SMOTE method. Choose from: 'smote', 'borderline-1', 'borderline-2', 'smoteenn', 'smotetomek'")

    # 執行重抽樣
    X_resampled, y_resampled = sampler.fit_resample(X_train, y_train)
    print(f"\n=== {smote_method.upper()} label distribution ===")
    print(pd.Series(y_resampled).value_counts())

    # 建立並訓練模型
    model = xgb.XGBClassifier(random_state=42)
    model.fit(X_resampled, y_resampled)

    # 評估模型表現
    evaluate_model(model, X_resampled, y_resampled, X_test, y_test)

    # 可選：畫學習曲線
    # plot_learning_curve(model, X_resampled, y_resampled)

    # 儲存模型以供畫圖
    model_dict[smote_method.upper()] = model

# 要跑的所有方法
methods = ['smote', 'borderline-1', 'borderline-2', 'smoteenn', 'smotetomek']

# 跑整個流程
for method in methods:
    run_smote_pipeline(method, X_train, y_train, X_test, y_test, all_models)
plot_combined_roc_pr_curves(all_models, X_test, y_test)
```

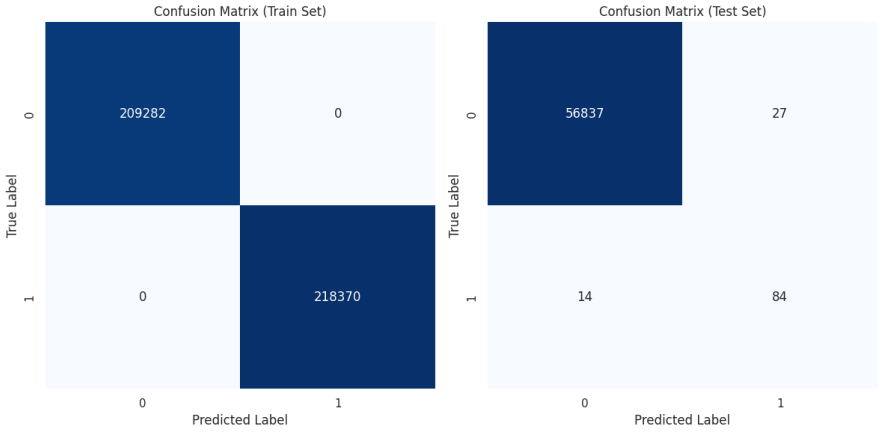


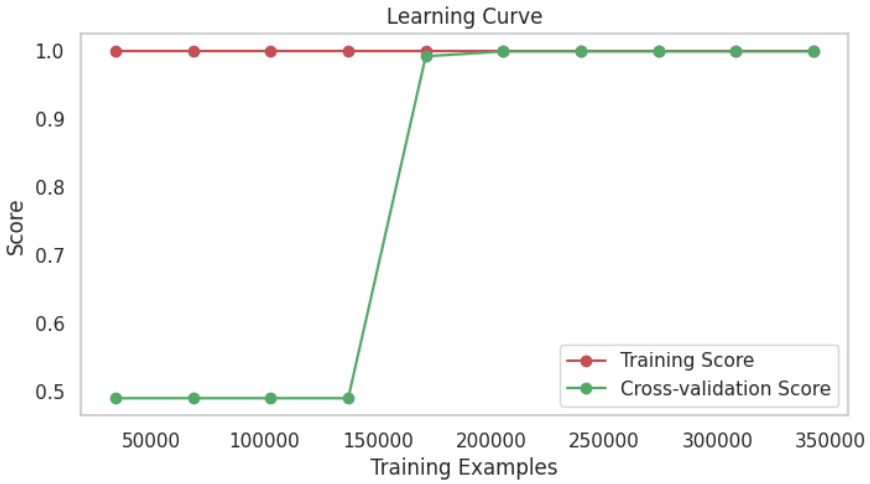
總結：

- **最佳整體表現模型：SMOTEENN**（在 ROC 和 PR 曲線都表現最佳）

XGBoost + SMOTEENN 的分類報告、混淆矩陣、學習曲線

```
Train Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209282
           1       1.00      1.00      1.00    218370

    accuracy                           1.00    427652
   macro avg       1.00      1.00      1.00    427652
weighted avg       1.00      1.00      1.00    427652

Test Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.76      0.86      0.80        98

    accuracy                           1.00     56962
   macro avg       0.88      0.93      0.90     56962
weighted avg       1.00      1.00      1.00     56962

Train AUC: 1.000000
Test AUC: 0.987259
```





6. 採樣前後分析

🔎 模型效能比較 (XGBoost)

指標	採樣前 (原始資料)	採樣後 (SMOTEENN)
Precision (Class 1)	0.92	0.76
Recall (Class 1)	0.81	0.86
F1-Score (Class 1)	0.86	0.80
Test AUC	0.9743	0.9873
PR Curve AP	0.8777	0.8728

AUC 提升

- SMOTEENN 的 AUC 從 **0.9743 → 0.9873**，顯示模型對整體分類能力的提升（模型區分能力更好）

Recall 提升

- 在少數類別（1）的 **Recall 從 0.81 → 0.86**，表示模型在測試集中更擅長抓到正類別的樣本（更少漏判）

Precision 下降

- Precision 降到了 **0.76**（從 0.92），說明有更多的誤報（False Positive）被引入。這是使用 SMOTEENN 的常見現象：它透過混合欠抽樣把資料分佈平衡化，但同時也可能導致邊界更模糊，容易多預測一些假陽性

🔷 混淆矩陣比較

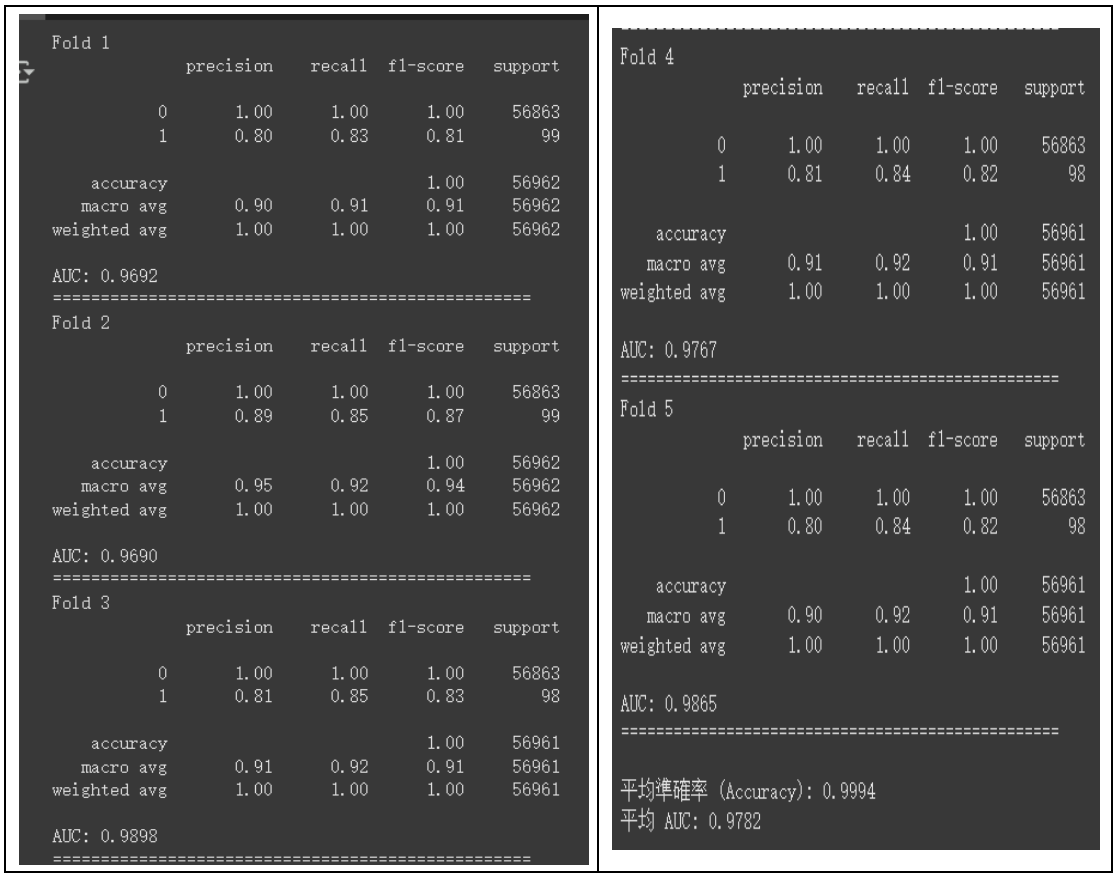
	0 (TN)	1 (FP)
採樣前	56857	7
採樣後	56837	27

	0 (FN)	1 (TP)
採樣前	19	79
採樣後	14	84

True Positive 增加了（79 → 84），模型對 Class 1 的敏感度提升。

False Positive 也明顯增加了（7 → 27），表示多預測了一些 1，但實際上是 0。

5 折交叉驗證



```
Fold 1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.80      0.83      0.81        99

    accuracy                           1.00     56962
   macro avg       0.90      0.91      0.91     56962
weighted avg       1.00      1.00      1.00     56962

AUC: 0.9692
==================================================
Fold 2
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.89      0.85      0.87        99

    accuracy                           1.00     56962
   macro avg       0.95      0.92      0.94     56962
weighted avg       1.00      1.00      1.00     56962

AUC: 0.9690
==================================================
Fold 3
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.81      0.85      0.83        98

    accuracy                           1.00     56961
   macro avg       0.91      0.92      0.91     56961
weighted avg       1.00      1.00      1.00     56961

AUC: 0.9898
==================================================
Fold 4
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.81      0.84      0.82        98

    accuracy                           1.00     56961
   macro avg       0.91      0.92      0.91     56961
weighted avg       1.00      1.00      1.00     56961

AUC: 0.9767
==================================================
Fold 5
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.80      0.84      0.82        98

    accuracy                           1.00     56961
   macro avg       0.90      0.92      0.91     56961
weighted avg       1.00      1.00      1.00     56961

AUC: 0.9865
==================================================

平均準確率 (Accuracy): 0.9994
平均 AUC: 0.9782
```

模型對正常樣本的判別非常好，對異常樣本的辨識能力也很強，但仍可努力提高 recall（減少漏判）。

7. 特徵重要性分析

使用 **SHAP 套件**，對 **xgb_model_smoteenn**（XGBoost 模型），創建一個專門用於解釋樹模型的 SHAP 解釋器。

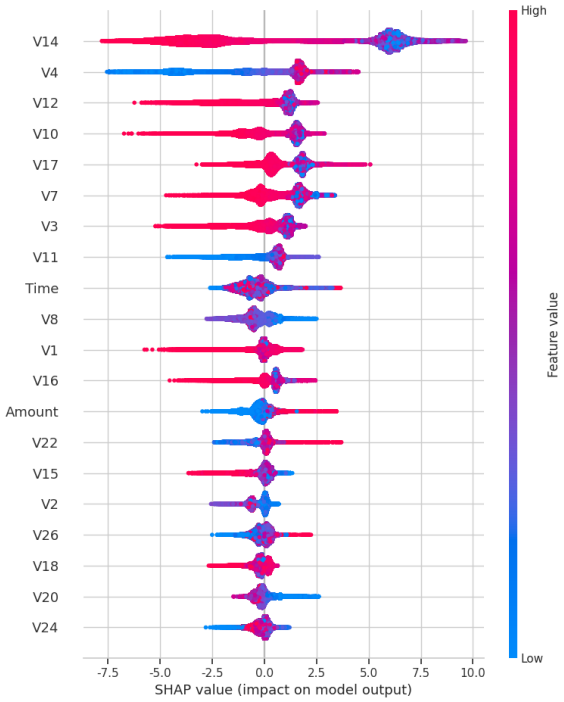
```
# 建立 SHAP Explainer (針對樹模型使用 TreeExplainer)
explainer = shap.TreeExplainer(xgb_model_smoteenn)
# 計算 SHAP 值
shap_values = explainer(X_train_resampled)

print("X_train_resampled shape:", X_train_resampled.shape)
print("SHAP values shape:", shap_values.shape) # (x, y, z), x: 樣本數, y: 特徵數量, z: Label的數量
```

```
X_train_resampled shape: (429712, 30)
SHAP values shape: (429712, 30)
```

SHAP Summary Plot

```
# SHAP Summary Plot
shap.summary_plot(shap_values.values, X_train_resampled)
```

- **縱軸**：各個特徵（Feature），上方是影響最大的特徵，越往下影響力越小
- **顏色**：特徵值的大小（藍色 = 低，紅色 = 高）
- **橫軸**：SHAP 值（對模型預測的影響，左邊負向影響，右邊正向影響）

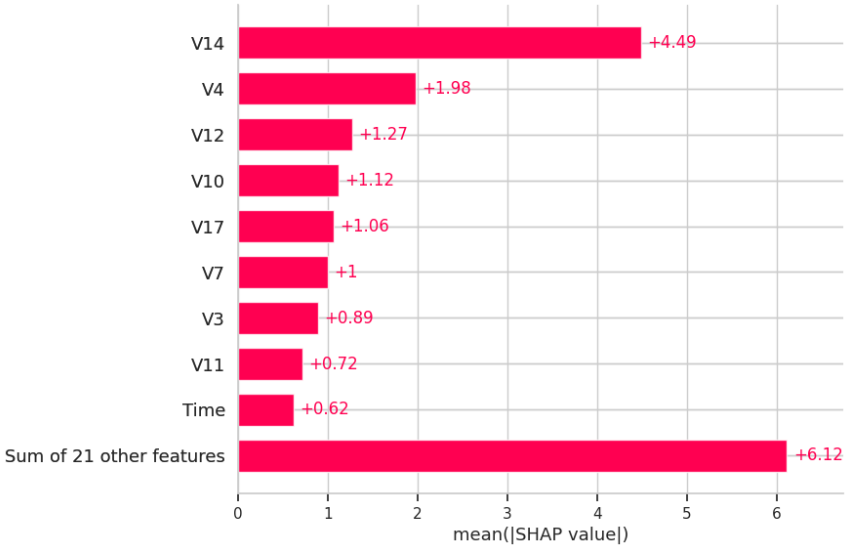


- **V14 越大，對於預測值的貢獻度越高，反之**
- **V4 越大，具有更大的總體模型影響**
- **Time 的大小,某種程度都會影響預測值**

Global bar plot

```
# Global bar plot
shap.plots.bar(shap_values)
```

- **縱軸**：各個特徵（Feature），上方是影響最大的特徵，越往下影響力越小
- **橫軸**：mean(|SHAP value|)，表示每個特徵對預測的重要程度（取絕對值後平均），值越大表示對模型貢獻越大



- **V14：平均 SHAP 值最高（+4.49），對模型解釋力最大**
- **其次是 V4、V12、V10、V17 等**

Cohort bar plot

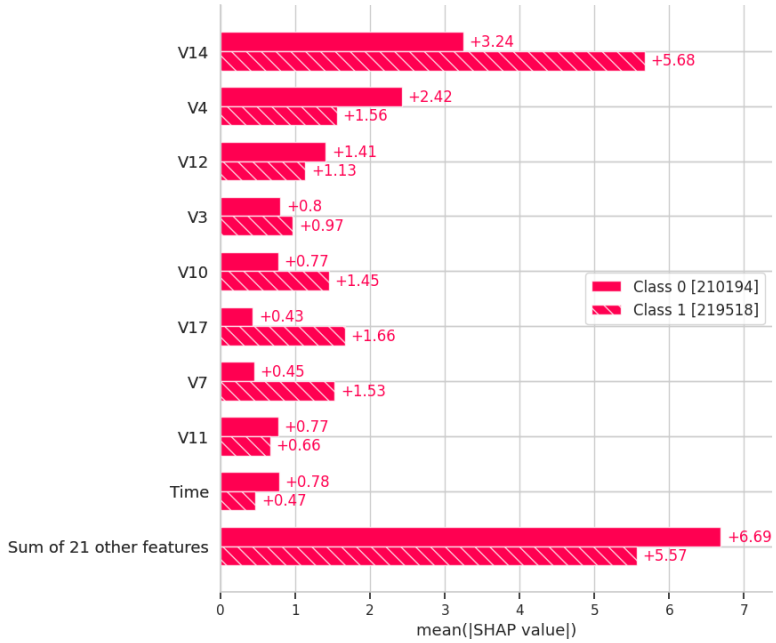
```
# Cohort bar plot
# 使用 SMOTE 後的 y_resampled 對應的分類標籤
cohorts = ["Class 0" if y_train_resampled.iloc[i] == 0 else "Class 1" for i in range(len(y_train_resampled))]

shap_cohorts = shap_values.cohorts(cohorts)
shap.plots.bar(shap_cohorts.abs.mean(0))
```

- **縱軸**：各個特徵（Feature），上方是影響最大的特徵，越往下影響力越小
- **橫軸**：mean(|SHAP value|)，表示每個特徵對預測的重要程度（取絕對值後平均），值越大表示對模型貢獻越大

紅色條形：

- **實心條：Class 0（正常樣本）**
- **斜線條：Class 1（異常樣本）**

這表示對於不同類別，該特徵的平均影響力（SHAP 值）是否存在差異。



V14：

- **對 Class 1（異常樣本）的平均影響力明顯高於 Class 0**
- Class 0: +3.24
- Class 1: +5.68

V14 是區分詐欺樣本的關鍵特徵，它的平均影響力在異常樣本上更大，V10, V12, V3, V7, V17, V8，這些特徵也有一定的重要性，但影響不如 V14 和 V4。

GitHub

連結：https://github.com/c110156247/Credit-card-Detection/blob/main/Credit_Card_Fraud_Detection.ipynb

參考連結

https://www.cnblogs.com/massquantity/p/8592091.html

https://www.cnblogs.com/massquantity/p/9382710.html