EX1實作紀錄:Random forest

參數調整:

```
X = np.asarray(data.iloc[:, ~data.columns.isin(['Class'])])
Y = np.asarray(data.iloc[:, data.columns == 'Class'])

# split training set and data set
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=TEST_SIZE, random_state=RANDOM_SEED)

# build Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=114,max_depth=14,random_state=RANDOM_SEED)
rf_model.fit(X_train, y_train)
```

[7]

```
c:\Users\USER\Desktop\程式\machinelearning\.venv\Lib\site-packages\sklearn\base.py:1389: DataConversionWarning:
  return fit_method(estimator, *args, **kwargs)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=14, n_estimators=114, random_state=42)
```

調整了初建立的樹的數量為114以提升表現，並限制單顆樹的深度為14防止overfitting

```
# predict and print result
y_proba = rf_model.predict_proba(X_test)[:, 1]
threshold = 0.4888
y_pred = (y_proba >= threshold).astype(int)
print(classification_report(y_test, y_pred))
```

將threshold調整為0.4888 (測試出來在0.485左右會是最佳結果

結果產出:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.93      0.84      0.88       136

    accuracy                           1.00     85443
   macro avg       0.97      0.92      0.94     85443
weighted avg       1.00      1.00      1.00     85443
```

調整後三項micro avg為 0.97 0.92 0.94

```
Random Forest Evaluation:
================================================
          Accuracy: 0.9996371850239341
   Precision Score: 0.9411764705882353
      Recall Score: 0.8235294117647058
          F1 Score: 0.8784313725490196

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.94      0.82      0.88       136

    accuracy                           1.00     85443
   macro avg       0.97      0.91      0.94     85443
weighted avg       1.00      1.00      1.00     85443
```

範例micro avg為 0.97 0.91 0.94

EX1實作紀錄:Kmeans

參數調整:

由於只單純加上n_init定義Cluster中心點的更新次數和max_iter更新次數的上限對於模型優化沒太大幫助，於是我想到使用了PCA來將資料降維希望能優化分群效果到優化的目的

```
pca = PCA(n_components=0.01919810)
n_x_train_pca = pca.fit_transform(n_x_train)
x_test_pca = pca.transform(x_test)
```

✓ 0.2s

其實正常好像要保留9成到9成5的數值最好，但我試著試著只保留0.02左右的資料反而預測結果是最好的,我不太清楚為什麼qq

```
scores = []
for k in range(2,12):
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=RANDOM_SEED)
    kmeans.fit(n_x_train_pca)
    score = silhouette_score(n_x_train_pca, kmeans.labels_)
    scores.append(score)

optimal_k = np.argmax(scores) + 2
kmeans = KMeans(n_clusters=optimal_k, init='k-means++',max_iter=1145,n_init=14,random_state=RANDOM_SEED)
kmeans.fit(n_x_train_pca)
y_pred_test = kmeans.predict(x_test_pca)

def align_labels(y_true, y_pred, n_clusters):
    labels = np.zeros_like(y_pred)
    for i in range(n_clusters):
        mask = (y_pred == i)
        if np.sum(mask) > 0:
            labels[mask] = np.bincount(y_true[mask]).argmax()
        else:
            labels[mask] = 0  # Default to normal class
    return labels

y_pred_aligned = align_labels(y_test, y_pred_test, optimal_k)
```

✓ 0.1s

因加入了PCA,所以有些資料改成_pca的新變數,而參數max_iter更新次數上限設為1145而n_init更新次數設為14

結果產出:

```
KMeans (Unsupervised) Evaluation:
=============================================
         Accuracy: 0.9987477031471274
  Precision Score: 0.8153846153846154
     Recall Score: 0.3581081081081081
         F1 Score: 0.49765258215962443

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.82      0.36      0.50       148

    accuracy                           1.00     85443
   macro avg       0.91      0.68      0.75     85443
weighted avg       1.00      1.00      1.00     85443
```

調整後在犧牲一點點F1 score和Recall Score後提升了Accuracy和Precision Score

Accuracy 0.99872->0.99874

Precision Score 0.7826 ->0.8153

雖然並沒有調整到全部都高於範例，但考慮到最重要的準確率和精確率都有提升便保存了下來

```
KMeans (Unsupervised) Evaluation:
=============================================
         Accuracy: 0.9987242957293166
  Precision Score: 0.782608695652174
     Recall Score: 0.36486486486486486
         F1 Score: 0.4976958525345622

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.78      0.36      0.50       148

    accuracy                           1.00     85443
   macro avg       0.89      0.68      0.75     85443
weighted avg       1.00      1.00      1.00     85443
```

範例數值

EX2實作紀錄:isolation+XGBoost

選擇isolation+XGBoost是因為範例也是使用這兩種融合，若出來成果不佳聽講解時會有比較直觀的感受。

```python
  # Run Isolation Forest
  iso = IsolationForest(n_estimators=100, contamination=0.00173, random_state=RANDOM_SEED)
  iso.fit(x_train_scaled)
  anomaly_scores_train = iso.decision_function(x_train_scaled)
  anomaly_labels_train = iso.predict(x_train_scaled)  # -1: anomaly, 1: normal

  anomaly_scores_test = iso.decision_function(x_test_scaled)
  anomaly_labels_test = iso.predict(x_test_scaled)
```

設定contamination異常樣本的比例，由下圖得出

```python
fraud = data[data['Class'] == 1]
nonfraud = data[data['Class'] == 0]
print(f'Fraudulent:{len(fraud)}, non-fraudulent:{len(nonfraud)}')
print(f'the positive class (frauds) percentage: {len(fraud)}/{len(fraud) + len(nonfraud)} ({len(fraud)/(len(fraud) + len(nonfraud))*100:.3f}%)')
```

```
✓ 0.0s
Fraudulent:492, non-fraudulent:284315
the positive class (frauds) percentage: 492/284807 (0.173%)
```

```python
# Training XGBoost classifier
xgb_model = XGBClassifier(n_estimators=90, max_depth=10, learning_rate=0.09, gamma=0.6, scale_pos_weight=60, random_state=RANDOM_SEED)
xgb_model.fit(x_train_with_iso, y_train)


# Prediction and evaluation
threshold = 0.55
y_proba = xgb_model.predict_proba(x_test_with_iso)[:, 1]
y_pred = (y_proba >= threshold).astype(int)
```

(以下各參數皆為多次測試下所得出的最佳結果)

樹的數量n_estimator=90

最大深度max_depth=10

學習率learning_rate=0.09

節點分裂所需的最小損失函數下降值gamma=0.6

用於平衡類別不平衡的問題scale_pos_weight=60

threshold=0.55

結果產出:

```
Hybrid model Evaluation:
=============================================
        Accuracy: 0.9995552590615966
 Precision Score: 0.9435483870967742
    Recall Score: 0.7905405405405406
        F1 Score: 0.8602941176470589

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.94      0.79      0.86       148

    accuracy                           1.00     85443
   macro avg       0.97      0.90      0.93     85443
weighted avg       1.00      1.00      1.00     85443
```

這是我能調出來最好的結果了，嘗試過加上PCA降維但結果還不如不加，不太確定問題出在哪，我的Accuracy怎麼改都提不上去了，但還是有調到全部高於0.9且precision是有高於範例的0.96。

```
Hybrid Model Evaluation:
=============================================
         Accuracy: 0.9996722961506501
  Precision Score: 0.9285714285714286
     Recall Score: 0.8602941176470589
         F1 Score: 0.8931297709923665

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.93      0.86      0.89       136

    accuracy                           1.00     85443
   macro avg       0.96      0.93      0.95     85443
weighted avg       1.00      1.00      1.00     85443
```

範例數值

參考資料

https://hackmd.io/@CynthiaChuang/Common-Evaluation-MetricAccuracy-Precision-Recall-F1-ROCAUC-and-PRAUC

https://medium.com/ai反斗城/learning-model-random-forest-ca4e3f8a63d3

https://medium.com/@jason8410271027/學習筆記-k-means實作篇-5c3fb9faf17

https://ithelp.ithome.com.tw/m/articles/10299735

https://cyeninesky3.medium.com/xgboost-a-scalable-tree-boosting-system-論文筆記與實作-2b3291e0d1fe

https://blog.csdn.net/wzk4869/article/details/128738001

https://blog.csdn.net/qq_34160248/article/details/124538485