2019 ECT Project

第1- 4題

☒ **從 data_new.csv 選取 Crossing ~ GKReflexes 欄位（共 34 個屬性）**

☒ **計算以上所有欄位的平均**

☒ **加標籤（大於平均：' Above-average Players'，小於平均：' Below-average Players'）**

☒ **訓練模型，可針對所需的模型進行屬性挑選**

☒ **切分資料集（test_size=0.33），並用測試集測試模型**

☒ **分析結果需印出 accuracy 、classification report、confusionmatrix**

☒ **調整模型，讓 accuracy 達到 0.9 以上**

☒ **加分題（每大題至多 2.5%）嘗試使用 matplotlib 等套件將各個演算法結果視覺化**

步驟過程

1. 首先，import 所需套件。

```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

2. 依各步驟進行實作。

```
# 讀取CSV檔案
data = pd.read_csv('datanew.csv', index_col=0)

# 從 data_new.csv 選取Crossing ~ GKReflexes欄位（共34個屬性）
df = data.loc[:,'Crossing':'GKReflexes']

# 計算以上所有欄位的平均
array_data = np.array(df)
column_data_mean = np.mean(array_data, axis =0)
all_data_mean = np.mean(column_data_mean)

# 加標籤（大於平均：' Above-average Players'，小於平均：' Below-average Players'）
df['all_mean'] = df[:].mean(axis=1)
df.loc[df.all_mean > all_data_mean, 'label']  = 'Above-average Players'
df.loc[df.all_mean <= all_data_mean, 'label'] = 'Below-average Players'
df.drop('all_mean', axis=1, inplace=True)

data['label'] = df['label']

feature = df.iloc[:,0:34]

#將屬性轉為數字label
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
target = le.fit_transform(data['label'])

# 切分訓練與測試資料
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(feature, target, test_size = 0.33, random_state=1)

# 定義 target_name 用於顯示圖表使用
target_names = ['Above-average Players', 'Below-average Players']
```

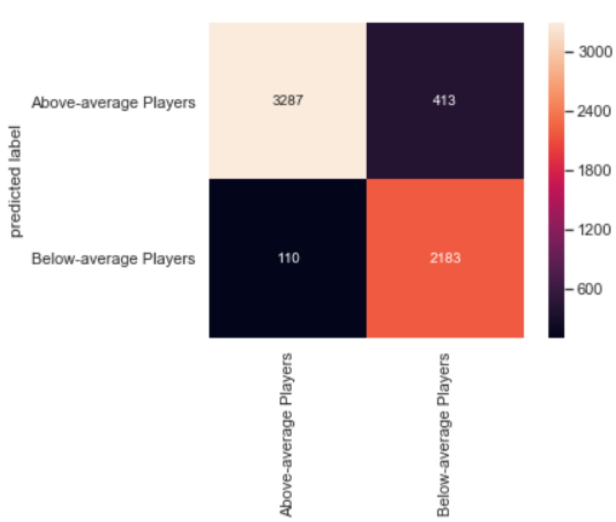
1. Naive Bayes（20%）

模型建立與訓練，並進行預測，透過 seaborn 顯示 confusion matrix

```python
from sklearn.naive_bayes import GaussianNB

# 建立 Naive Bayes 模型
nb = GaussianNB()

# 僅挑選 Crossing ~ SlidingTackle 的屬性
feature = data.loc[:,'Crossing':'SlidingTackle']
X_train, X_test, y_train, y_test = train_test_split(feature, target, test_size = 0.33, random_state=1)

datanew_nb = nb.fit(X_train, y_train)

# 預測
y_test_pred = datanew_nb.predict(X_test)
y_train_pred = datanew_nb.predict(X_train)
# 績效
test_accuracy = accuracy_score(y_test ,y_test_pred)
train_accuracy = accuracy_score(y_train ,y_train_pred)

cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

print('訓練集準確度為：', train_accuracy)
print('測試集準確度為：', test_accuracy)
print('\nclassification_report:\n', classification_report(y_test, y_test_pred, target_names=target_names))

mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',
            xticklabels=target_names, yticklabels=target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

顯示結果如下

```
classification_report:
                       precision    recall  f1-score   support

Above-average Players       0.89      0.97      0.93      3397
Below-average Players       0.95      0.84      0.89      2596

             accuracy                           0.91      5993
            macro avg       0.92      0.90      0.91      5993
         weighted avg       0.92      0.91      0.91      5993
```



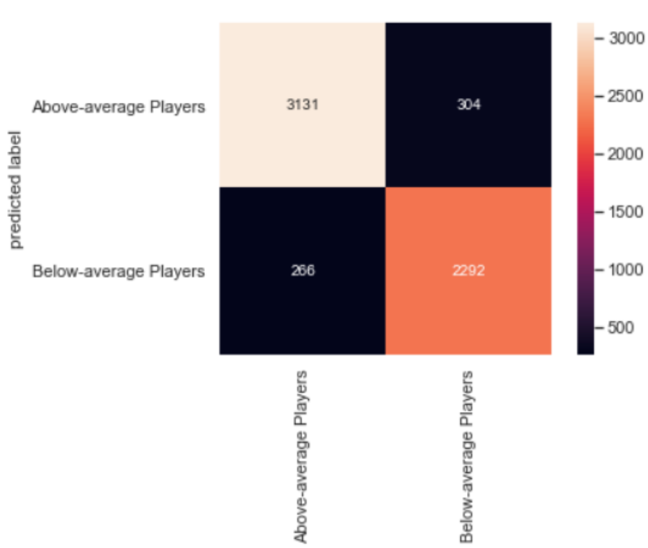
2. Decision Trees（20%）

模型建立與訓練，並進行預測，透過 seaborn 顯示 confusion matrix

```python
from sklearn.tree import DecisionTreeClassifier

# 建立 DecisionTree 模型
clf = DecisionTreeClassifier(criterion = 'gini', max_depth=7)

datanew_clf = clf.fit(X_train, y_train)
# 預測
y_test_pred = datanew_clf.predict(X_test)
y_train_pred = datanew_clf.predict(X_train)
# 績效
test_accuracy = accuracy_score(y_test ,y_test_pred)
train_accuracy = accuracy_score(y_train ,y_train_pred)

cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

print('訓練集準確度為：', train_accuracy)
print('測試集準確度為：', test_accuracy)
print('\nclassification_report:\n', classification_report(y_test, y_test_pred, target_names=target_names))

mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',
            xticklabels=target_names, yticklabels=target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

顯示結果如下

```
訓練集準確度為： 0.9473943777741246
測試集準確度為： 0.9048890372100784

classification_report:
                        precision    recall  f1-score   support

Above-average Players       0.91      0.92      0.92      3397
Below-average Players       0.90      0.88      0.89      2596

             accuracy                           0.90      5993
            macro avg       0.90      0.90      0.90      5993
         weighted avg       0.90      0.90      0.90      5993
```



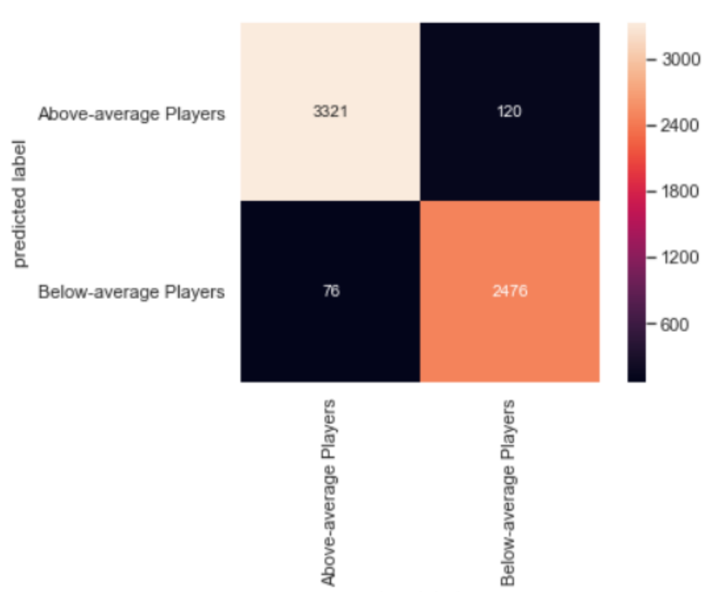
3. Logistic Regression（20%）

模型建立與訓練，並進行預測，透過 seaborn 顯示 confusion matrix

```python
from sklearn.linear_model import LogisticRegression

# 建立 LinearRegression 模型
lr = LogisticRegression(solver='liblinear')

datanew_lr = lr.fit(X_train, y_train)
# 預測
y_test_pred = datanew_lr.predict(X_test)
y_train_pred = datanew_lr.predict(X_train)
# 績效
test_accuracy = accuracy_score(y_test ,y_test_pred)
train_accuracy = accuracy_score(y_train ,y_train_pred)

cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

print('訓練集準確度為：', train_accuracy)
print('測試集準確度為：', test_accuracy)
print('\nclassification_report:\n', classification_report(y_test, y_test_pred, target_names=target_names))

mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',
            xticklabels=target_names, yticklabels=target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

顯示結果如下

```
訓練集準確度為： 0.9660529344073648
測試集準確度為： 0.9672951777073252

classification_report:
                         precision    recall  f1-score   support

Above-average Players       0.97      0.98      0.97      3397
Below-average Players       0.97      0.95      0.96      2596

             accuracy                           0.97      5993
            macro avg       0.97      0.97      0.97      5993
         weighted avg       0.97      0.97      0.97      5993
```



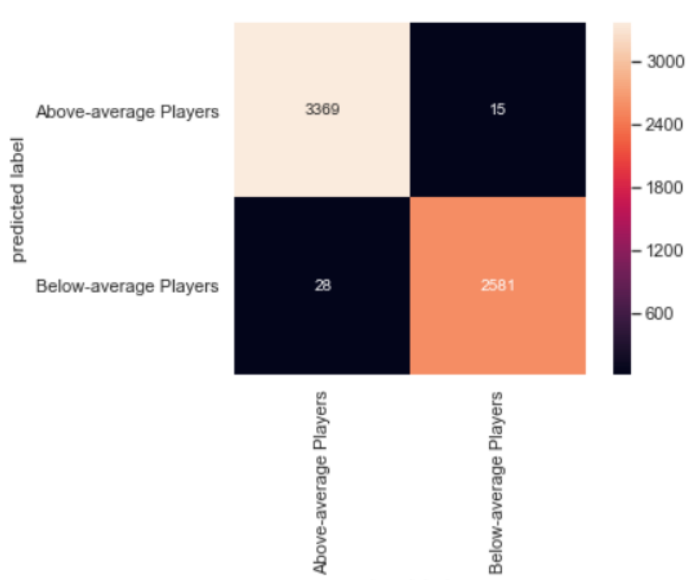
4. SVM（20%）

模型建立與訓練， 並進行預測， 透過 seaborn 顯示 confusion matrix

```python
from sklearn.svm import SVC

# 建立 SVM 模型
svc = SVC(gamma = 0.0001, C=1.0)

datanew_svc = svc.fit(X_train, y_train)
# 預測
y_test_pred = datanew_svc.predict(X_test)
y_train_pred = datanew_svc.predict(X_train)
# 績效
test_accuracy = accuracy_score(y_test ,y_test_pred)
train_accuracy = accuracy_score(y_train ,y_train_pred)

cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

print('訓練集準確度為：', train_accuracy)
print('測試集準確度為：', test_accuracy)
print('\nclassification_report:\n', classification_report(y_test, y_test_pred, target_names=target_names))

mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',
            xticklabels=target_names, yticklabels=target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

顯示結果如下

```
訓練集準確度為： 0.9958901857636034
測試集準確度為： 0.9928249624561989

classification_report:
                        precision    recall  f1-score   support

Above-average Players       1.00      0.99      0.99      3397
Below-average Players       0.99      0.99      0.99      2596

             accuracy                           0.99      5993
            macro avg       0.99      0.99      0.99      5993
         weighted avg       0.99      0.99      0.99      5993
```



第5題

取data_new.csv，進行KNN分析

可針對所需的模型進行屬性挑選

選取Crossing ~ GKReflexes欄位，並加上Skill Moves欄位。

```
feature = data.iloc[:,6:40]
feature['Skill Moves'] = data['Skill Moves']
```

5. KNN（20%）

(a) 推薦與 "Neymar Jr" 相像的前五名足球選手

```
from sklearn.neighbors import NearestNeighbors

# 找出 Neymar Jr 資料 的 index
Neymar_Jr = data[data['Name'] == 'Neymar Jr']
Neymar_Jr_index = Neymar_Jr.index.tolist()[0]
# 找出 Neymar Jr 的資料
Neymar_Jr = feature[Neymar_Jr_index:Neymar_Jr_index+1]

# 將 feature 進行標準化 (以 Neymar Jr 為中心)
normalized_feature=(feature-Neymar_Jr.mean())/(feature.std())

# 找出標準化後的 Neymar Jr 資料
normalized_Neymar_Jr = normalized_feature[Neymar_Jr_index:Neymar_Jr_index+1]

nbrs = NearestNeighbors(n_neighbors=6).fit(normalized_feature)
distances, indices = nbrs.kneighbors(normalized_Neymar_Jr)
for x in indices:
    print(data['Name'][x])
distances
```

```
2         Neymar Jr
5         E. Hazard
0          L. Messi
65    Douglas Costa
84         R. Mahrez
15         P. Dybala
Name: Name, dtype: object

array([[0.        , 2.51869119, 2.65644407, 3.00039668, 3.0243212 ,
        3.03490117]])
```

1. 先透過原始 data 找出 Neymar Jr 資料 的 index

2. 將feature以Neymar Jr為中心進行標準化，不同於一般標準化的方法，每筆feature中的資料會減去Neymar Jr的資料，再除以feature的標準差。

3. 利用步驟1. 找出的 index 找出 標準化後的feature中Neymar Jr的資料。(此資料將用於後續

kneighbors演算法中)

4. 得出E. Hazard、L. Messi、Douglas Costa、R. Mahrez、P. Dybala為與Neymar Jr相像的前五名足球選手

(b) 推薦與 " L. Messi " 相像的前五名足球選手

```python
from sklearn.neighbors import NearestNeighbors

# 找出 L. Messi 的 index
L_Messi = data[data['Name'] == 'L. Messi']
L_Messi_index = L_Messi.index.tolist()[0]
# 找出 L_Messi 的資料
L_Messi = feature[L_Messi_index:L_Messi_index+1]

# 將 feature 進行標準化 (以 Neymar Jr 為中心)
normalized_feature=(feature-L_Messi.mean())/(feature.std())

# 找出標準化後的 L_Messi 資料
normalized_L_Messi = normalized_feature[L_Messi_index:L_Messi_index+1]

nbrs = NearestNeighbors(n_neighbors=6).fit(normalized_feature)
distances, indices = nbrs.kneighbors(normalized_L_Messi)
for x in indices:
    print(data['Name'][x])
distances
```

```
0        L. Messi
2       Neymar Jr
5       E. Hazard
15      P. Dybala
154     A. Robben
68        M. Reus
Name: Name, dtype: object

array([[0.        , 2.65644407, 2.66905995, 2.7565991 , 3.26975065,
        3.43961592]])
```

1. 先透過原始 data 找出 L. Messi 資料 的 index

2. 將feature以L. Messi為中心進行標準化, 不同於一般標準化的方法, 每筆feature中的資料會減去L. Messi的資料, 再除以feature的標準差。

3. 利用步驟1. 找出的 index 找出 標準化後的feature中L. Messi的資料。(此資料將用於後續kneighbors演算法中)

4. 得出Neymar Jr、E. Hazard、P. Dybala、A. Robben、M. Reus為與L. Messi相像的前五名足球選手

第6題

6. 加分題(10%)

對資料額外進行有趣的分析

透過將 attribute 之間的相關性視覺化，可以更佳了解到屬性之間的相互關係，能做更多加深的應用

```
# 讀取CSV檔案
data = pd.read_csv('datanew.csv', index_col=0)

# correlation
corr = data.corr()

# 設定 figure 大小
fig = plt.figure(figsize=(20,10))

ax = fig.add_subplot()

# matshow 矩陣視覺化
cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,len(data.columns),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(data.columns)
ax.set_yticklabels(data.columns)
plt.show()
```