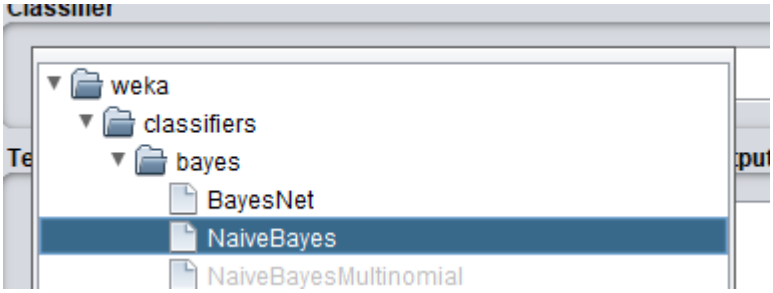
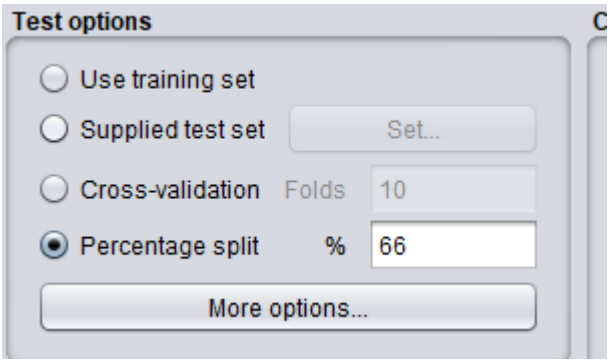
ECT HW6

Weka Part:

前置處理:

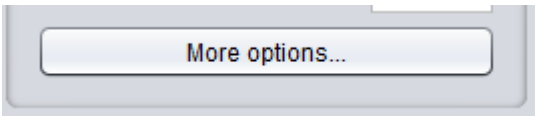


➔ 依題目要求使用 NaiveBayes

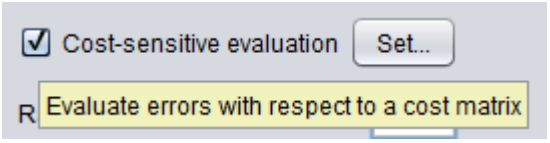


➔ 依題目要求設定 Percentage Split = 66%

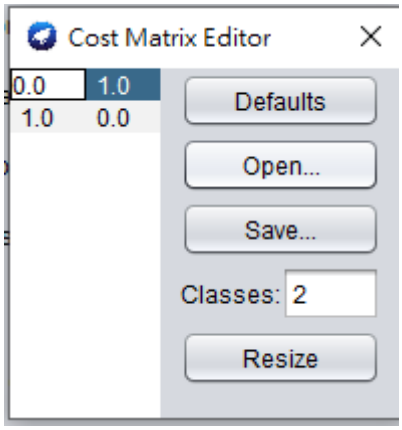
(a)



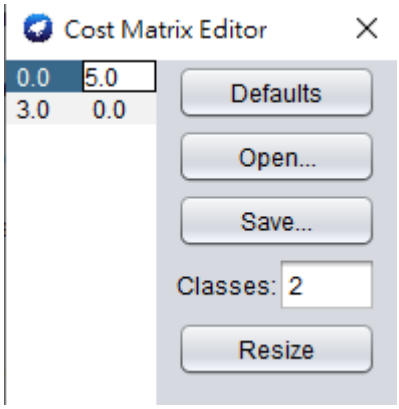
➔ 點選 More Options



➔ 如圖所示，將框框打勾，並點擊「Set...」



➔ 會跑出以上的圖片，把 Cost Matrix 設定成題目要求的數值



➔ 如圖所示，雙擊之後便可以修改成所需的數值

```
Total Cost                              70
Average Cost                            0.5147

                       a  b   <-- classified as
                     40 11 |  a = 1
                      5 80 |  b = 0
```

Confusion Matrix:

Cost Matrix:

0.0	5.0
3.0	0.0

➔ 兩個數值分別如上圖所示，必須配合 Confusion Matrix 來解釋

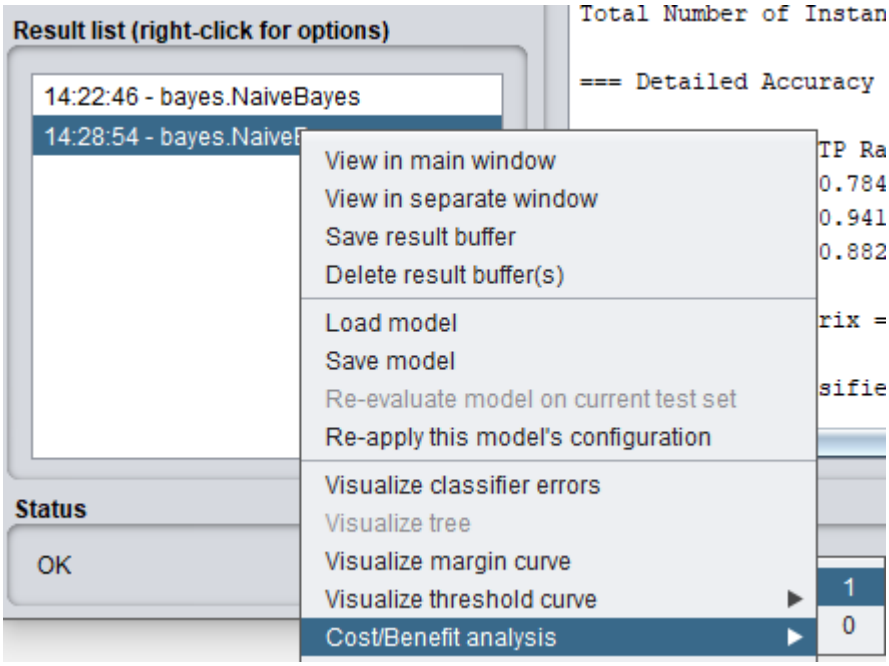
➔ For Total Cost

- 將兩個 Matrix 相乘，即可得到 Total Cost
- 0*40 + 5*11 + 3*5 + 0*80 = 55+15 = 70

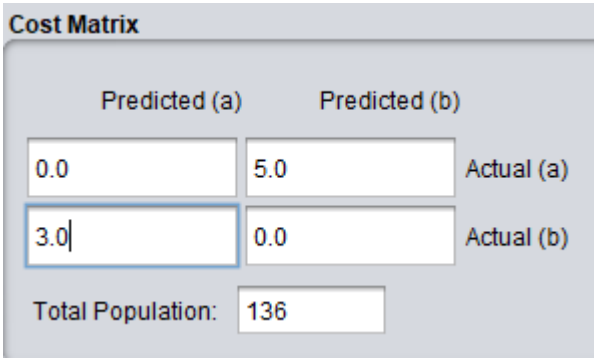
➔ For Average Cost

- 純粹 Total Cost/ 個數 = 700 / (40+11+5+80) = 70 / 136 ≒ 0.5147

(b)



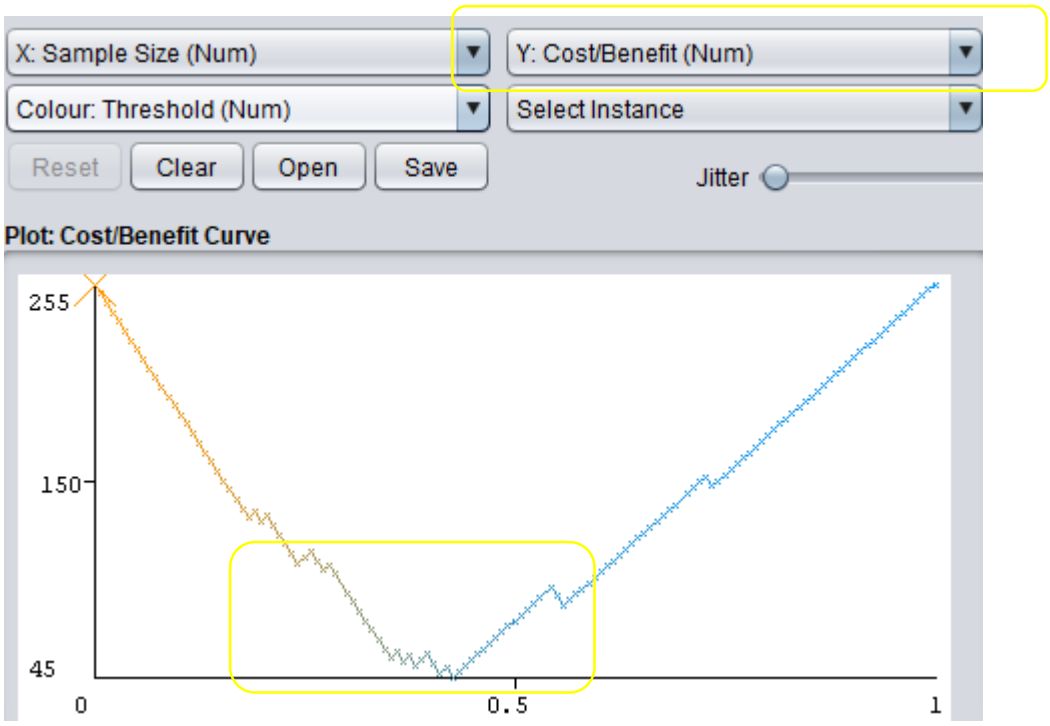
➔ 在 Result list 的紀錄終點擊右，找到 Cost/Benefit analysis 並依照題目要求，選擇 class = 1



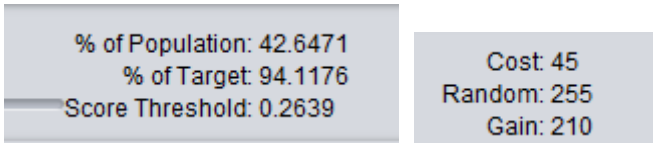
➔ 依照題目要求，設 Cost Matrix 如上圖

➔ 因題目要求要找最佳的 sample size rate，在此先定義何謂最佳的 sample size Rate。

- 根據 Cost/Benefit analysis 評分方式，「Gain」越高就代表越好，因此需要調整 sample size rate 找到 Gain 最高的數值

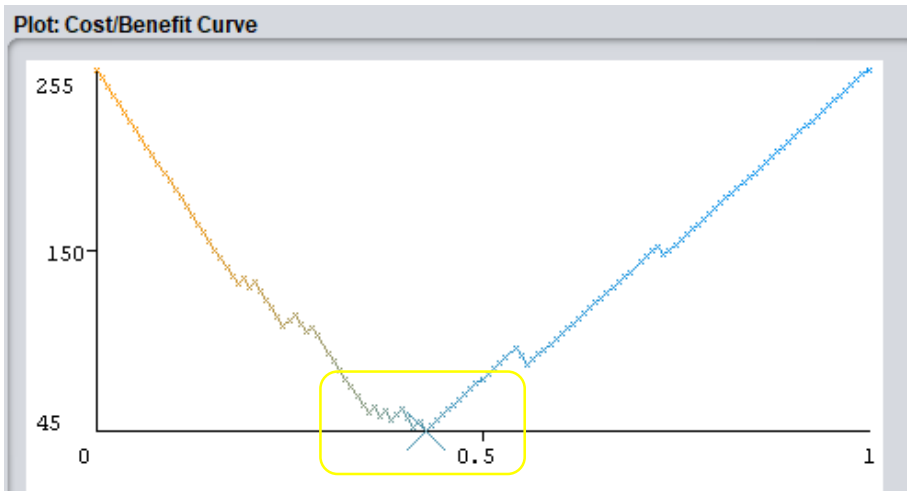


➔ 可以把 Y 軸設為 Cost/Benefit，找 cost 較低的部分，幫助尋找 gain 的最大值。



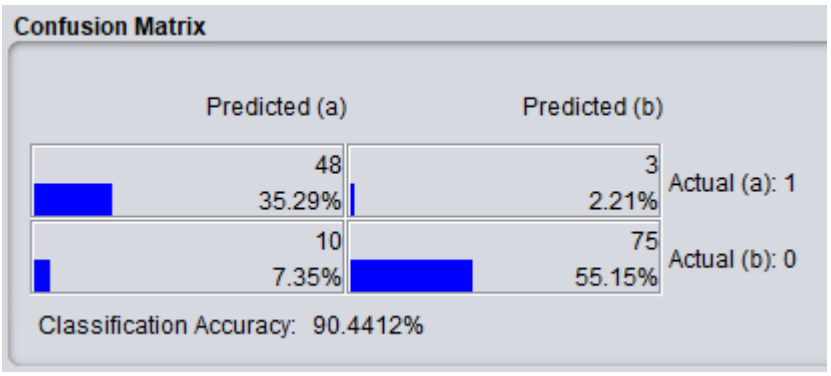
➔ 在 sample size rate = 42.6471%時找到最大的 Gain = 210

- 因此 42.6471%為最佳的 sample size rate



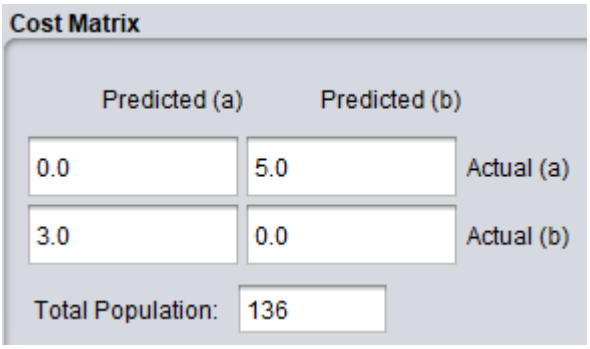
➔ 此時的確在 Cost 的低點，證明剛剛的分析是有用的

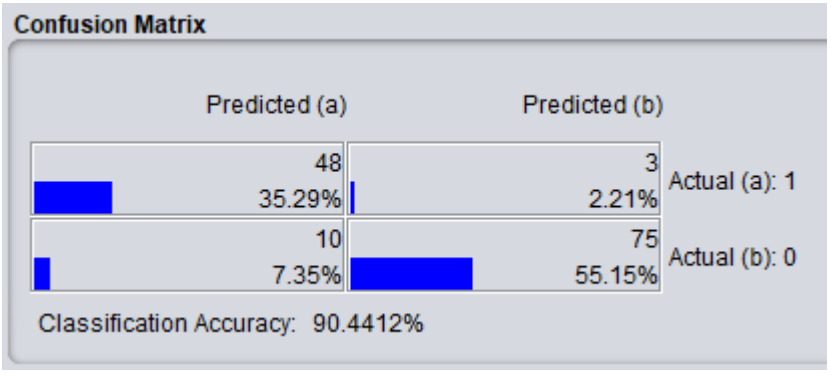
(c)



➔ Confusion Matrix 如上圖所示

➔ Cost = 45





➔ 必須配合這兩張圖來驗證 Cost，將兩個 Matrix 相乘即可得到 Cost

- Cost = 0*48 + 5*3 + 3*10 + 0*75 = 15 + 30 = 45

Python Part:

前置處理:

```
import pandas as pd
df = pd.read_csv('Social_Network_Ads.csv')
df
```

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0
...	...	...	...	...
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

➔ 先讀取資料，並觀看資料格式

```
# x:input
x = df.loc[:,"Gender":"EstimatedSalary"]
print(x)
# y:output
y = df.loc[:,["Purchased"]]
print(y)
```

➔ 將資料分成 input、output

(d)

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=1)
```

➔ 依照題目要求，設 test_size = 0.33，random_state = 1，把資料切割成 training、testing data

➔ 若是直接 fit 進 model 訓練會產生以下錯誤

- ValueError: could not convert string to float: 'Female'
- 因為其中有 nominal 的值無法處理，因此要做預處理

```
# 檢查是否為數字
import re
def is_number(num):
  pattern = re.compile(r'^[-+]?[-0-9]\d*\.\d*|[-+]?\.?[0-9]\d*$')
  result = pattern.match(num)
  if result:
    return True
  else:
    return False
```

➔ 等等要處理的只有 nominal 的部分，因此先寫個函數來判斷是否為數字

```
# Preprocessing
from sklearn import preprocessing
# 將nominal屬性轉為數字label，因為sklearn.naive_bayes.GaussianNB不接受nominal的input
le = preprocessing.LabelEncoder()


# 儲存編碼後的input
X_train_encoded = []
X_test_encoded = []

# 將input進行編碼，只有nominal需要編碼
for col_name in x:
    data = str(x[col_name][0]) # 先暫時轉為string，為了檢察是否為數字

    if(is_number(data) == False): # 若不是數字，代表為nominal
        encoded_train = le.fit_transform(x_train[col_name]) # 將各col的value轉成數字
        encoded_test = le.fit_transform(x_test[col_name]) # 將各col的value轉成數字
        X_train_encoded.append(encoded_train)
        X_test_encoded.append(encoded_test)
    else:
        X_train_encoded.append(x_train[col_name]) # 不用編碼直接放入list
        X_test_encoded.append(x_test[col_name])
```

➔ 對每一個欄位的第一個值進行判斷是否為數字。在此的方法為先把第一個欄位的值變成 String，再用我寫的函數去判斷。

- 不是數字 ： 用 LabelEncoder()編碼後放入 list 中
- 是數字 ： 直接放入 list 中

```
# 將input組合起來
x_train =list(zip(X_train_encoded[0], X_train_encoded[1], X_train_encoded[2]))
x_test = list(zip(X_test_encoded[0], X_test_encoded[1], X_test_encoded[2]))

# 轉成array
import numpy as np
x_train = np.asarray(x_train)
x_test = np.asarray(x_test)
```

➔ 因為要放進 model 中訓練時 input 要變成一個一個的數組，因此用 zip 把每個數值組合起來，並且轉換成 array 來當作 input。

```
#Import Gaussian Naive Bayes 模型 (高斯樸素貝氏)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

# 訓練集訓練模型
# model.fit(x, y)
model.fit(x_train, y_train.values.ravel())
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

➔ 建立模型，並把 training data fit 進去訓練

```
# 用testing data去預測
predicted = model.predict(x_test) # 代表預測的答案
expected = y_test # 代表真正的答案
```

➔ 使用 predict()函數，把 testing data 的 x_test 放進去進行預測，並把實際答案 y_test 放進 expected 變數中。

```
print("準確率:", model.score(x_test, y_test))
```

```
準確率: 0.8333333333333334
```

➔ 使用內建的 score()函數找出 model 的準確率

```
def TP_Rate(confusion_matrix_2D, class_label):
    return confusion_matrix_2D[0][class_label]/(confusion_matrix_2D[0][0] + confusion_matrix_2D[0][1])

def FP_Rate(confusion_matrix_2D, class_label):
    return confusion_matrix_2D[1][class_label]/(confusion_matrix_2D[1][0] + confusion_matrix_2D[1][1])
```

➔ 因為題目只有說找 TP Rate / FP Rate，並未提及是針對哪一個 Class Label 做處理，因此我寫了一個函數，參數為 Confusion Matrix 和 Class Label，讓使用者可以自己決定要針對特定的 class 計算。

```
confusion_matrix_2D = metrics.confusion_matrix(expected, predicted)
print("準確率:", model.score(x_test, y_test))
print("For purchase = 0:")
print("TP Rate:", TP_Rate(confusion_matrix_2D, 0))
print("FP Rate:", FP_Rate(confusion_matrix_2D, 0))
print()
print("For purchase = 1:")
print("TP Rate:", TP_Rate(confusion_matrix_2D, 1))
print("FP Rate:", FP_Rate(confusion_matrix_2D, 1))
```

➔ 先用內建函數 confusion_matrix()取得 matrix，再用自定義函數去計算

purchase = 0 和 purchase = 1 時分別的 TP Rate / FP Rate

```
For purchase = 0:
TP Rate: 0.8641975308641975
FP Rate: 0.21568627450980393

For purchase = 1:
TP Rate: 0.13580246913580246
FP Rate: 0.7843137254901961
```

➔ 結果如圖所示

```
print(metrics.confusion_matrix(expected, predicted))
```

```
Confusion Matrix:
[[70 11]
 [11 40]]
```

➔ 為求保險，我把實際的 Confusion Matrix 印出來算一次

➔ For purchase = 0

- TP Rate = 70 / (70+11) = 0.8641975308641975
- FP Rate = 11 / (11+40) = 0.21568627450980393

➔ For purchase = 1

- TP Rate = 11 / (70+11) = 0.13580246913580246
- FP Rate = 40 / (11+40) = 0.7843137254901961

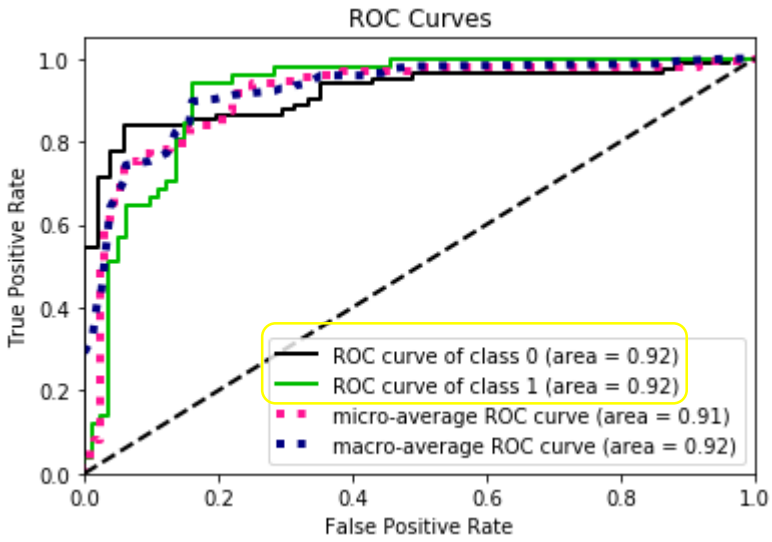
(e)

```
import scikitplot as skplt
import matplotlib.pyplot as plt
y_probas = model.predict_proba(x_test)

skplt.metrics.plot_roc(y_test, y_probas)
plt.show()
```

➔ 使用 scikitplot 中的 plot_roc()來進行繪製，依照文檔需求，第一個參數為 y_test(y_label 的答案)、第二個參數為 y_probas，代表每個 x_test 估計出來的機率所形成的 vector。

- y_probas 可以使用內建函數 predict_proba()來取得



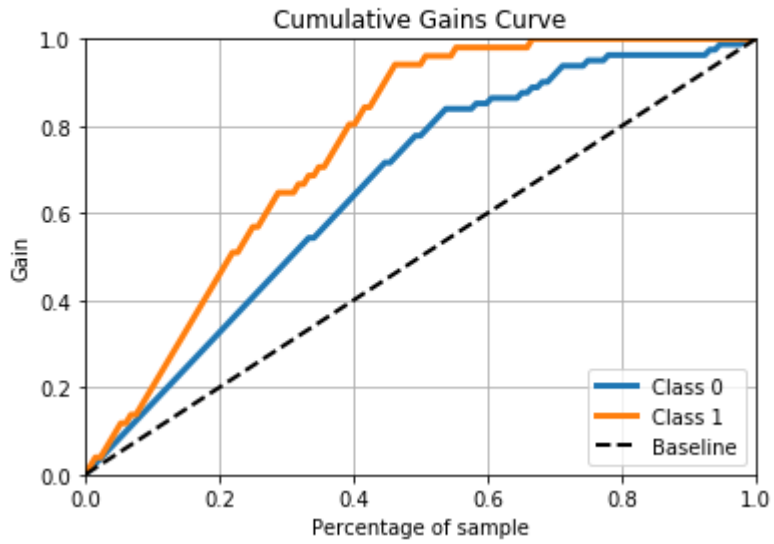
➔ 結果如上圖所示

➔ AUC 部分可分為 4 種答案，依照上課所學主要關注純粹 ROC curve 的部分

- Class(purchase) = 0 時 ：AUC = 0.92
- Class(purchase) = 1 時 ：AUC = 0.92
- Micro/Macro average : AUC = 0.91 和 0.92

(f)

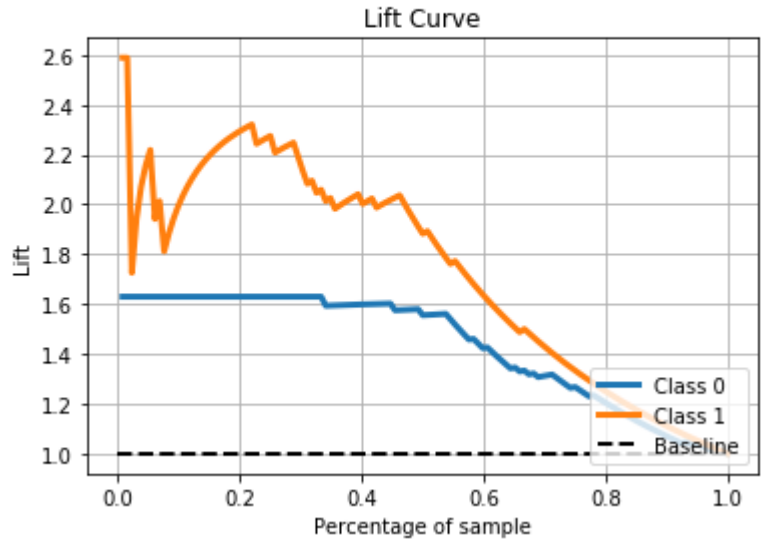
```
skplt.metrics.plot_cumulative_gain(y_test, y_probas)
plt.show()
```



➔ 使用內建函數 plot_cumulative_gain()，繪製出 lift chart。在此 Y 軸雖說標示為 Gain，但其實它就是 TP Rate。

(g)

```
skplt.metrics.plot_lift_curve(y_test, y_probas)
plt.show()
```



➔ 使用內建函數 plot_lift_curve()，繪製出 lift curve