

Data Mining Term Project

第12組 吳奐萱、戴翊帆、陳昀萱、陳姿妤

一、題目

根據消費行為預測購買金額的分布

二、動機

我們根據消費金額的多寡分成三組(8k 以下、8k~16k、16k 以上)，利用客戶的資訊以及消費行為預測客戶的消費金額會落在哪一組，賣場可以根據預測結果鎖定廣告族群。

三、資料集敘述

1. 資料集來源：從 kaggle 抓取資料

<https://www.kaggle.com/mehdidag/black-Friday>

2. 資料集簡介：此資料集是零售商店中黑色星期五進行的交易樣本，商店希望更好地了解客戶的購買行為。客戶的資訊包含性別、年齡、職業、居住地類別、婚姻狀況，另外還包含了產品的資訊，以及客戶消費的金額。

四、分析工具

使用Python以及Scikit-Learn的四種模型，分別是SVM、GDBT、DecisionTree、RandomForest，來進行分析，看哪一種模型可以產生最好的結果。

五、實作與評估方法

(一) 資料前處理

1. 匯入模組

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import numpy as np
```

2. 讀檔

```
df = pd.read_csv("C:/Users/acer/.spyder-py3/BlackFriday.csv")
```

3. 補齊NA值

```
df.fillna(0,inplace=True)
```

4. 捨棄不相關的欄位 (User_ID, Product_ID)

```
df = df.drop(['User_ID', 'Product_ID'], axis=1)
```

5. 將categorical data且**沒有大小關係**的欄位轉為one-hot

```
df = df.replace("F", 0)
df = df.replace("M", 1)
occ = pd.get_dummies(df['Occupation'])
city = pd.get_dummies(df['City_Category'])
pc1 = pd.get_dummies(df['Product_Category_1'])
pc2 = pd.get_dummies(df['Product_Category_2'])
pc3 = pd.get_dummies(df['Product_Category_3'])
```

```
df = df.drop(['Occupation', 'City_Category',
'Product_Category_1', 'Product_Category_2',
'Product_Category_3'], axis=1)
df1 = pd.concat([df, occ, city, pc1, pc2, pc3], axis=1)
```

6. 處理categorical data且有大小關係的欄位

6-1 Age 分成 9, 22, 32, 42, 48, 53, 65共七個族群

```
df1 = df1.replace("0-17", 9)
df1 = df1.replace("18-25", 22)
df1 = df1.replace("26-35", 32)
df1 = df1.replace("36-45", 42)
df1 = df1.replace("46-50", 48)
df1 = df1.replace("51-55", 53)
df1 = df1.replace("55+", 65)
scaler = MinMaxScaler()
scaler.fit(df1[['Age']])
MinMaxScaler(copy=True, feature_range=(0, 1))
df1[['Age']] = scaler.transform(df1[['Age']])
```

6-2 Stay in current city

```
df1 = df1.replace("4+", 10)
scaler2 = MinMaxScaler()
scaler2.fit(df1[['Stay_In_Current_City_Years']])
MinMaxScaler(copy=True, feature_range=(0, 1))
df1[['Stay_In_Current_City_Years']] =
scaler2.transform(df1[['Stay_In_Current_City_Years']])
```

6-3 Purchase (最小值==185, 最大值==23961)

```
Purchase = np.array(df1['Purchase'].values.tolist())
for i in range(0, len(Purchase)):
    if Purchase[i] < 8000:
        Purchase[i] = 0
    elif Purchase[i] < 16000:
        Purchase[i] = 1
    else:
        Purchase[i] = 2
Purchase = pd.DataFrame(Purchase, dtype=np.float)
```

7. 匯出處理好的資料，省去之後測試model的時間

```
df1 = df1.drop(['Purchase'], axis=1)
df1.to_csv('hw6_data_modified1.csv')
Purchase.to_csv('hw6_label1.csv')
```

8. 匯入資料

```
X =
pd.read_csv("C:/Users/acer/.spyder-py3/hw6_data_modified1.csv")
X = X.drop(['Unnamed: 0'], axis=1)
Y = pd.read_csv("C:/Users/acer/.spyder-py3/hw6_label1.csv")
Y = Y.drop(['Unnamed: 0'], axis=1)
x = np.array(X)
y = np.array(Y)
```

9. 亂數拆成訓練集(75%)與測試集(25%)

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.25, shuffle=True, random_state = 41)
```

(二) SVM

10. 建模型 && 調整參數

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
#調整參數
parameters = {'kernel':('linear', 'rbf'), 'C':[10, 100, 1000],
'gamma':[0.001, 0.0001]}
clf = GridSearchCV(estimator=svm.SVC(), param_grid=parameters, cv=5)
#選擇前面5000筆來尋找最佳參數 速度會比較快
find_bestx = x_train[0:5000]
find_besty = y_train[0:5000]
find_besty = find_besty.ravel()
clf.fit(find_bestx, find_besty)

# Print out the results
print('Best `C`: ',clf.best_estimator_.C)
print('Best kernel:',clf.best_estimator_.kernel)
print('Best `gamma`: ',clf.best_estimator_.gamma)

# Create the SVC model
svc_model = svm.SVC(gamma=0.001, C=100., kernel='rbf')
svc_model.fit(x_train, y_train.ravel())
p1_test = svc_model.predict(x_test)
p2_train = svc_model.predict(x_train)
```

11. 計算Precision, Recall, Accuracy

```
from sklearn.metrics import confusion_matrix
def confusionmatrix(true, predict):
    cnf_matrix = confusion_matrix(true, predict)
    al = 0
    for i in range(3):
        for j in range(3):
            al = al + cnf_matrix[i][j]
    P1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[1][0]+cnf_matrix[2][0])
    P2 = cnf_matrix[1][1]/(cnf_matrix[0][1]+cnf_matrix[1][1]+cnf_matrix[2][1])
    P3 = cnf_matrix[2][2]/(cnf_matrix[0][2]+cnf_matrix[1][2]+cnf_matrix[2][2])

    R1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[0][1]+cnf_matrix[0][2])
    R2 = cnf_matrix[1][1]/(cnf_matrix[1][0]+cnf_matrix[1][1]+cnf_matrix[1][2])
    R3 = cnf_matrix[2][2]/(cnf_matrix[2][0]+cnf_matrix[2][1]+cnf_matrix[2][2])

    precision = (P1+P2+P3)/3
    recall = (R1+R2+R3)/3
    accuracy = (cnf_matrix[0][0] + cnf_matrix[1][1] + cnf_matrix[2][2])/ al
    return precision, recall, accuracy
p_test, r_test, a_test = confusionmatrix(y_test, p1_test)
p_train, r_train, a_train = confusionmatrix(y_train, p2_train)
```

	precision	recall	accuracy
test	0.8262	0.6182	0.7755

(三) GDBT

10. 建模型

```
from sklearn import ensemble
clf = ensemble.GradientBoostingClassifier()
clf.fit(train_X, train_Y)
predict = clf.predict(test_X)
```

11. 計算Precision, Recall, Accuracy

```
from sklearn.metrics import confusion_matrix
def confusionmatrix(true, predict):
    cnf_matrix = confusion_matrix(true, predict)
    al = 0
    for i in range(3):
        for j in range(3):
            al = al + cnf_matrix[i][j]

    P1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[1][0]+cnf_matrix[2][0])
    P2 = cnf_matrix[1][1]/(cnf_matrix[0][1]+cnf_matrix[1][1]+cnf_matrix[2][1])
    P3 = cnf_matrix[2][2]/(cnf_matrix[0][2]+cnf_matrix[1][2]+cnf_matrix[2][2])

    R1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[0][1]+cnf_matrix[0][2])
    R2 = cnf_matrix[1][1]/(cnf_matrix[1][0]+cnf_matrix[1][1]+cnf_matrix[1][2])
    R3 = cnf_matrix[2][2]/(cnf_matrix[2][0]+cnf_matrix[2][1]+cnf_matrix[2][2])

    precision = (P1+P2+P3)/3
    recall = (R1+R2+R3)/3
    accuracy = (cnf_matrix[0][0] + cnf_matrix[1][1] + cnf_matrix[2][2])/ al
    return precision, recall, accuracy

precision, recall, accuracy = confusionmatrix(test_Y, predict)
```

	precision	recall	accuracy
test	0.82137	0.62958	0.77672

(四) DecisionTree

10. 建模型

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(max_depth = 9)
classifier = classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

11. 計算Precision, Recall, Accuracy

```
from sklearn.metrics import confusion_matrix
def confusionmatrix(true, predict):
```

```

cnf_matrix = confusion_matrix(true, predict)
al = 0
for i in range(3):
    for j in range(3):
        al = al + cnf_matrix[i][j]

P1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[1][0]+cnf_matrix[2][0])
P2 = cnf_matrix[1][1]/(cnf_matrix[0][1]+cnf_matrix[1][1]+cnf_matrix[2][1])
P3 = cnf_matrix[2][2]/(cnf_matrix[0][2]+cnf_matrix[1][2]+cnf_matrix[2][2])

R1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[0][1]+cnf_matrix[0][2])
R2 = cnf_matrix[1][1]/(cnf_matrix[1][0]+cnf_matrix[1][1]+cnf_matrix[1][2])
R3 = cnf_matrix[2][2]/(cnf_matrix[2][0]+cnf_matrix[2][1]+cnf_matrix[2][2])

precision = (P1+P2+P3)/3
recall = (R1+R2+R3)/3
accuracy = (cnf_matrix[0][0] + cnf_matrix[1][1] + cnf_matrix[2][2])/ al
return precision, recall, accuracy

precision, recall, accuracy = confusionmatrix(y_test, y_pred)

```

	precision	recall	accuracy
test	0.6901	0.6411	0.7134

(五) Randomforest

10. 建模型

```

clf = RandomForestClassifier(n_estimators=100, max_depth=30,
random_state=41)
modelrf = clf.fit(x_train, y_train.ravel())

```

```

y_predict = modelrf.predict(x_test)
y_predict.reshape(len(y_predict), 1)
y_predict1 = modelrf.predict(x_train)
y_predict1.reshape(len(y_predict1), 1)

```

11. 計算Precision, Recall, Accuracy

```

from sklearn.metrics import confusion_matrix
def confusionmatrix(true, predict):
    cnf_matrix = confusion_matrix(true, predict)
    al = 0
    for i in range(3):
        for j in range(3):
            al = al + cnf_matrix[i][j]

    P1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[1][0]+cnf_matrix[2][0])
    P2 = cnf_matrix[1][1]/(cnf_matrix[0][1]+cnf_matrix[1][1]+cnf_matrix[2][1])
    P3 = cnf_matrix[2][2]/(cnf_matrix[0][2]+cnf_matrix[1][2]+cnf_matrix[2][2])
    R1 = cnf_matrix[0][0]/(cnf_matrix[0][0]+cnf_matrix[0][1]+cnf_matrix[0][2])
    R2 = cnf_matrix[1][1]/(cnf_matrix[1][0]+cnf_matrix[1][1]+cnf_matrix[1][2])
    R3 = cnf_matrix[2][2]/(cnf_matrix[2][0]+cnf_matrix[2][1]+cnf_matrix[2][2])
    precision = (P1+P2+P3)/3
    recall = (R1+R2+R3)/3
    accuracy = (cnf_matrix[0][0] + cnf_matrix[1][1] + cnf_matrix[2][2])/ al
    return precision, recall, accuracy

```

```
precision, recall, accuracy = confusionmatrix(y_test, y_predict)
```

	precision	recall	accuracy
test	0.67979	0.64328	0.71346

六、流程(分析流程圖、結果截圖等)

1.結果截圖

1-1 SVM

```
In [30]: print(p_test)
0.826208941002091

In [31]: print(r_test)
0.6182289913053811

In [32]: print(a_test)
0.7754997437211686
```

1-2 GDBT (並非最好的結果，後來為了截圖特地重train一次，所以和上面最好的結果不一樣)

accuracy	float64	1	0.7215819040886938
precision	float64	1	0.6970496052543481
predict	float64	(134395,)	[1. 0. 1. ... 1. 2. 1.]
recall	float64	1	0.6477981670483577

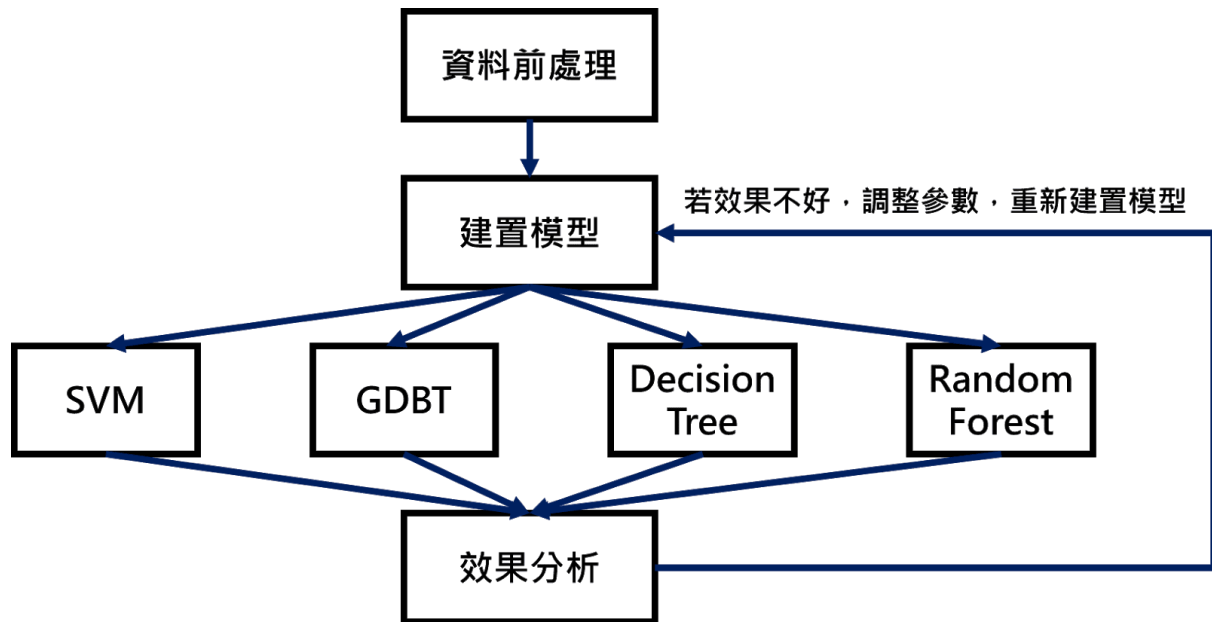
1-3 DecisionTree

```
accuracy : 0.713456601808103
precision : 0.6901830932638542
recall : 0.6411733106982963
```

1-4 Randomforest

```
precision:0.6797912651734839
recall:0.6432760169121293
accuracy:0.7134640425611072
```

2.分析流程圖



七、分析結果與結論(或其他補充內容)

1.結果與比較

(test data)

precision	SVM>GDBT>DecisionTree>Randomforest
recall	DecisionTree>Randomforest>GDBT>SVM
accuracy	GDBT>SVM>Randomforest>DecisionTree

可以看出越複雜的模型越容易有良好的Accuracy，像是DecisionTree的模型構造較簡單，所以算出的Accuracy是最低的，但是不代表Precision和Recall也是最低的，DecisionTree反而意外的擁有最好的Recall，而GDBT擁有最好的結果的原因可能是因為會參考前面的模型，所以表現得叫良好，而SVM的結果也不錯。選擇參數的方法除了自己隨機挑選一個一個測試以外，也可以使用GridSearchCV，自己選擇一些參數後，電腦會使用暴力破解，一個一個嘗試，最後可以直接印出擁有最高Accuracy的結果，其中的cv這個參數，代表著使用幾層的cross-validation，就不需要自己一個一個手動調，只要等train完之後的結果就可以了，其中SVM所耗費的時間最多。

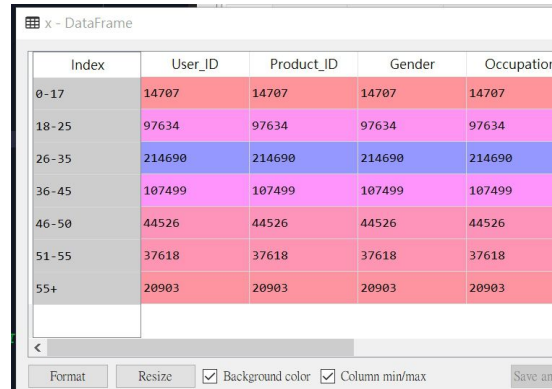
2. 資料前處理對於accuracy的影響

2-1 label

一開始的設定是希望可以將output分成10大類，但是accuracy連50%都過不了，我們猜想或許是資料量不夠所以無法切太細。將output改成切3大類後，accuracy就會大於70%，這就是我們選擇分三種output的原因。

2-2 Age

Age在原始資料裡是分成7大類，分布如下圖：



Index	User_ID	Product_ID	Gender	Occupation
0-17	14707	14707	14707	14707
18-25	97634	97634	97634	97634
26-35	214690	214690	214690	214690
36-45	107499	107499	107499	107499
46-50	44526	44526	44526	44526
51-55	37618	37618	37618	37618
55+	20903	20903	20903	20903

直接使用randomforest train的話，accuracy是0.7739。

以調高accuracy為目標，我們將Age分成三類：0-25設成21，26-35設成32，36以上都是50。

```
25
26 #Age
27 df1 = df1.replace("0-17", 21)
28 df1 = df1.replace("18-25", 21)
29 df1 = df1.replace("26-35", 32)
30 df1 = df1.replace("36-45", 50)
31 df1 = df1.replace("46-50", 50)
32 df1 = df1.replace("51-55", 50)
33 df1 = df1.replace("55+", 50)
34 scaler = MinMaxScaler()
```

結果跟第一項的Purchase結果不一樣：accuracy並沒有升高。我們猜想是因為Age並不是分類門檻的主要原因。