Data Mining HW2 《P76111262 林晨鈞》

Classification Problem:Who is good teacher?

Feature Information:

Age: 25-65	#年齡
Gender: M,F	#性別
Height: 140-190	#身高
Weight: 40-100	#體重
Attitude: Serious,Free	#教學態度(嚴肅/自由)
Prepare_hours: 0-6(hours)	#備課時間(單位:小時)
Class_type: Online,Physical	#授課方式(線上/實體)
Pass_rate: Low,High	#班上學生 Pass 課程比例(高/低)
Glasses: 0,1	#是否戴眼鏡
Classhw: Low,Middle,High	#課堂作業(低/中/高)
Care: Yes,No	#是否會關心學生(是/否)
Label: good,bad	#好老師/壞老師

Absolute right rules:

Attitude: Serious	#教學態度嚴肅
Prepare_hours: 3-6	#備課時間達 3-6 小時
Pass_rate: High	#學生通過課程比例高
Care: Yes	#會關心學生
Class_type: Physical	#實體授課

資料集說明:

此分類問題為”判斷該老師是否為一位好老師”，此資料集總共有 6000 筆資料，12 個不同的 features，其中”好老師”的資料數占 3107 筆，”壞老師”的資料數占 2893 筆。Absolute right rules 也列在上面，只要完全符合上述 5 點，即為”好老師”。(此資料僅為想像，並無惡意詆毀之意)

生成資料檔案:data_produce.py

資料集:teacher_data.csv

```
[6000 rows x 12 columns]
positive 3107 / negative 2893
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	age	gender	height	weight	attitude	prepare_hc	class_type	pass_rate	glasses	classhw	care	label
2	43	F	180	67	Serious	5	Physical	High	0	High	Yes	good
3	52	M	157	50	Serious	5	Physical	High	0	Low	Yes	good
4	36	M	161	42	Serious	3	Physical	High	1	High	Yes	good
5	52	M	162	54	Serious	6	Physical	High	1	High	Yes	good
6	28	M	179	41	Serious	0	Online	Low	1	Low	No	bad
7	48	F	159	48	Serious	6	Physical	High	1	High	Yes	good
8	36	F	140	88	Serious	5	Physical	High	1	Middle	Yes	good
9	55	F	179	62	Serious	4	Online	Low	1	Low	Yes	bad

1. Decision Tree (Train_set:0.75,Test_set:0.25)

Comparison :

當樹的最大深度設為 5 時，可以發現無論是訓練集還是測試集的預測結果都是 1.0，可以看到重要特徵都有被挑出來，且都是 Absolutely right rules，分別是:

1. Attitude:0.28852774
2. Prepare_hours:0.04443353
3. Class_type:0.36432398
4. Pass_rate:0.19989458
5. Care:0.10282016

PS: heatmap 為測試集的預測結果

```
from sklearn.tree import DecisionTreeClassifier
DecidionTreeModel=DecisionTreeClassifier(criterion='gini',max_depth=5,random_state=42) #建立模型
DecidionTreeModel.fit(X_train,y_train) #使用訓練資料訓練模型
print("Test set Accuracy:",DecidionTreeModel.score(X_test,y_test)) #使用測試資料預測分類，並印出準確率
print("Train set Accuracy:",DecidionTreeModel.score(X_train,y_train)) #使用訓練資料預測分類，並印出準確率
```

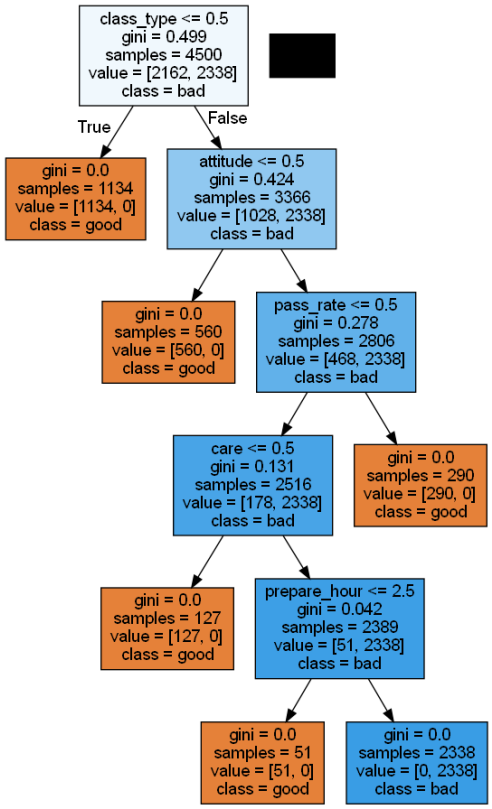
✓ 0.3s

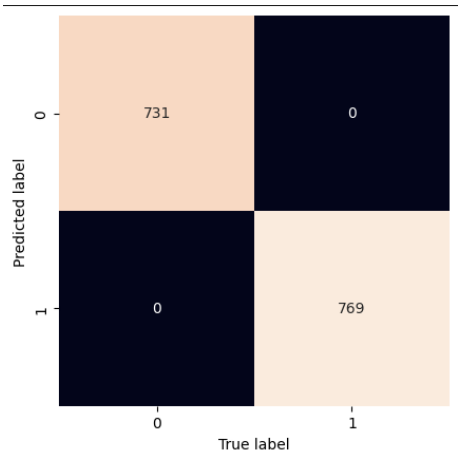
```
Test set Accuracy: 1.0
Train set Accuracy: 1.0
```

```
print('特徵重要程度: ',DecidionTreeModel.feature_importances_)
```

✓ 0.6s

```
特徵重要程度:  [0.         0.         0.         0.         0.28852774 0.04443353
 0.36432398 0.19989458 0.         0.         0.10282016]
```





當樹的最大深度設為 4 時，發現訓練集的預測準確率為 0.989333，測試集的預測準確率為 0.988666，其中重要的特徵 Prepare_hours 被省略了，但準確率還是不錯。

```
from sklearn.tree import DecisionTreeClassifier
DecidionTreeModel=DecisionTreeClassifier(criterion='gini',max_depth=4,random_state=42) #建立模型
DecidionTreeModel.fit(X_train,y_train) #使用訓練資料訓練模型
print("Test set Accuracy:",DecidionTreeModel.score(X_test,y_test)) #使用測試資料預測分類，並印出準確率
print("Train set Accuracy:",DecidionTreeModel.score(X_train,y_train)) #使用訓練資料預測分類，並印出準確率
```

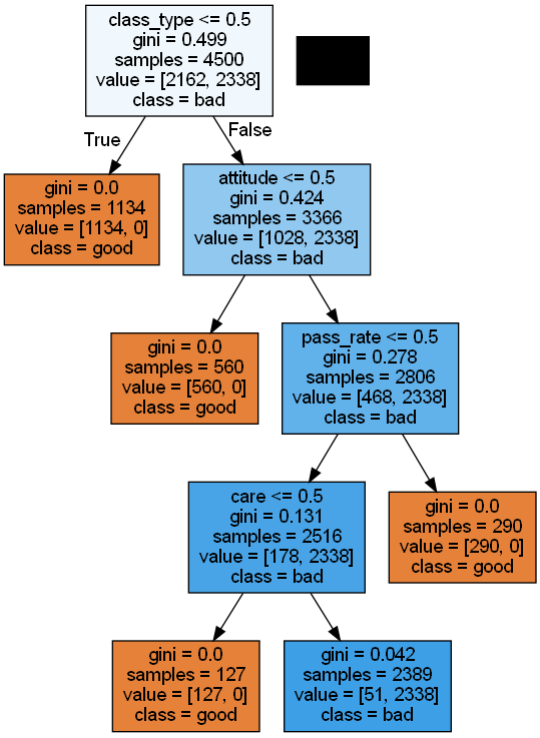
✓ 0.3s

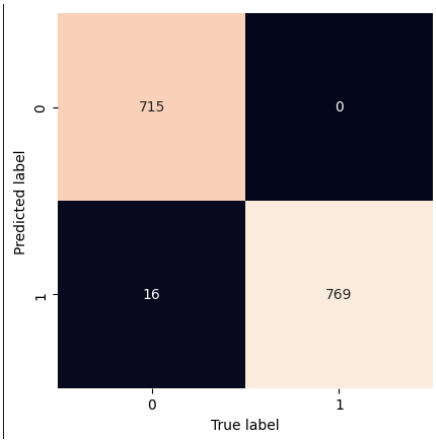
```
Test set Accuracy: 0.9893333333333333
Train set Accuracy: 0.9886666666666667
```

```
print('特徵重要程度: ',DecidionTreeModel.feature_importances_)
```

✓ 0.3s

```
特徵重要程度:  [0.         0.         0.         0.         0.30194419 0.
 0.38126493 0.20918962 0.         0.         0.10760126]
```





當樹的最大深度設為 3 時，發現訓練集的預測準確率為 0.96，測試集的預測準確率為 0.960444，其中重要的特徵 Prepare_hours, Care 被省略了。

```
from sklearn.tree import DecisionTreeClassifier
DecidionTreeModel=DecisionTreeClassifier(criterion='gini',max_depth=3,random_state=42) #建立模型
DecidionTreeModel.fit(X_train,y_train) #使用訓練資料訓練模型
print("Test set Accuracy:",DecidionTreeModel.score(X_test,y_test)) #使用測試資料預測分類，並印出準確率
print("Train set Accuracy:",DecidionTreeModel.score(X_train,y_train)) #使用訓練資料預測分類，並印出準確率
```

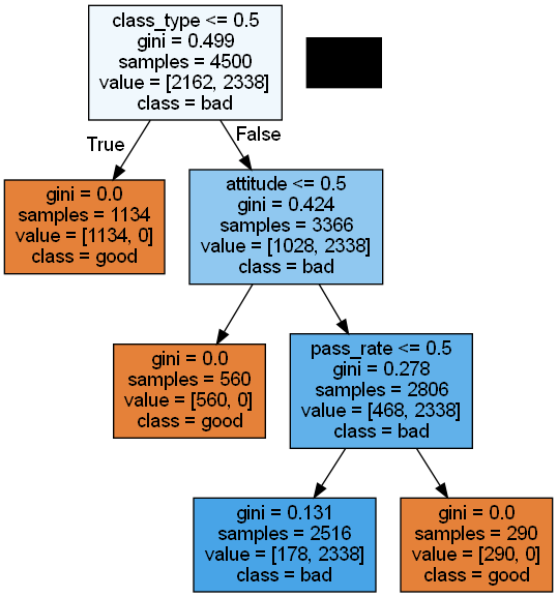
✓ 0.3s

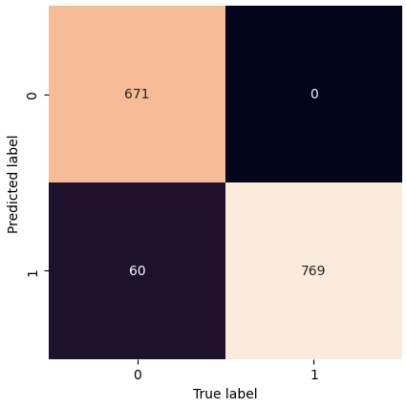
```
Test set Accuracy: 0.96
Train set Accuracy: 0.9604444444444444
```

```
print('特徵重要程度: ',DecidionTreeModel.feature_importances_)
```

✓ 0.3s

```
特徵重要程度:  [0.         0.         0.         0.         0.33835121 0.
 0.42723607 0.23441272 0.         0.         0.         ]
```





2. KNN

Comparison :

KNN 相較於 Decision Tree，預測的準確率偏低很多，將 n_neighbors 設定為 1 會得到以下結果，訓練集的準確率為 1.0，但測試集的準確率為 0.625333

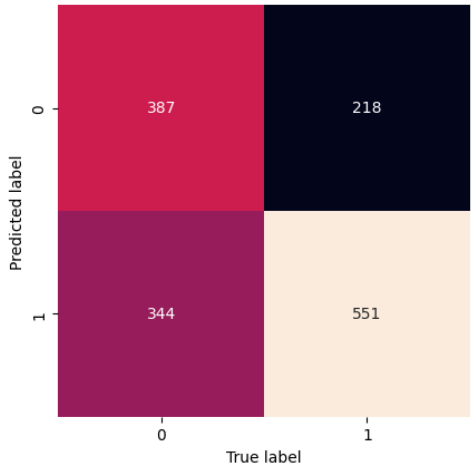
PS: heatmap 為測試集的預測結果

```
from sklearn.neighbors import KNeighborsClassifier

# 建立KNN模型
knnModel = KNeighborsClassifier(n_neighbors=1)
# 使用訓練資料訓練模型
knnModel.fit(X_train,y_train)
# 預測成功的比例
print('Training Set Accuracy: ',knnModel.score(X_train,y_train))
print('Testing Set Accuracy: ',knnModel.score(X_test,y_test))
```

✓ 0.9s

```
Training Set Accuracy:  1.0
Testing Set Accuracy:  0.6253333333333333
```



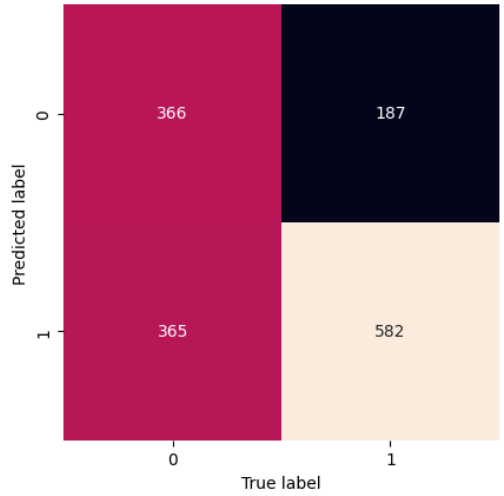
n_neighbors 設定為 3 會得到以下結果，訓練集的準確率為 0.826，但測試集的準確率為 0.632

```
from sklearn.neighbors import KNeighborsClassifier

# 建立KNN模型
knnModel = KNeighborsClassifier(n_neighbors=3)
# 使用訓練資料訓練模型
knnModel.fit(X_train,y_train)
# 預測成功的比例
print('Training Set Accuracy: ',knnModel.score(X_train,y_train))
print('Testing Set Accuracy: ',knnModel.score(X_test,y_test))
```

✓ 0.1s

```
Training Set Accuracy:  0.826
Testing Set Accuracy:  0.632
```

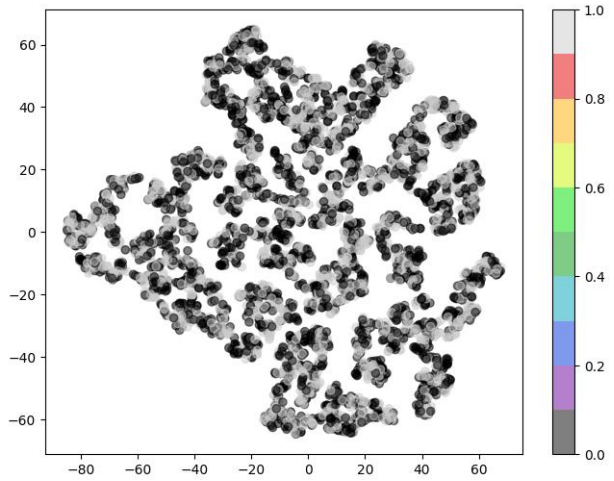


因為上述結果不盡理想，所以嘗試進行降維，我採用 TSNE 降維，並降成 2 維，以下第二張圖為降維後的分布圖，最後 n_neighbors 設定為 2，訓練集的準確率為 0.80，但測試集的準確率為 0.48，效果仍不盡理想，甚至更差。

```python
from sklearn.manifold import TSNE # 進行TSNE降維，降成2維

tsneModel = TSNE(n_components=2, random_state=42,n_iter=1000)
train_reduced = tsneModel.fit_transform(X_train)
test_reduced = tsneModel.fit_transform(X_test)
```

✓ 7.6s




```python
from sklearn.neighbors import KNeighborsClassifier

# 建立KNN模型
knnModel = KNeighborsClassifier(n_neighbors=2)
# 使用訓練資料訓練模型
# knnModel.fit(X_train,y_train)
knnModel.fit(train_reduced,y_train) # 降維資料
# 預測成功的比例
# print('Training Set Accuracy: ',knnModel.score(X_train,y_train))
# print('Testing Set Accuracy: ',knnModel.score(X_test,y_test))
print('Training Set Accuracy: ',knnModel.score(train_reduced,y_train))
print('Testing Set Accuracy: ',knnModel.score(test_reduced,y_test))
```

✓ 0.9s

```
Training Set Accuracy:  0.8004444444444444
Testing Set Accuracy:  0.48133333333333334
```

3. NaiveBayes

使用 Gaussian Naïve Bayes 所得到的結果分別是:

訓練集準確率:0.9886666

測試集準確率:0.9893333

```python
# Gaussian Naive Bayes: 主要用於連續變數，比方說特徵長度為幾公分、重量為幾公斤等等。
from sklearn.naive_bayes import GaussianNB
# 建立naive模型
NBModel = GaussianNB()
# 使用訓練資料訓練模型
NBModel.fit(X_train,y_train)
# 預測成功的比例
print('Training Set Accuracy: ',NBModel.score(X_train,y_train))
print('Testing Set Accuracy: ',NBModel.score(X_test,y_test))
```

✓ 0.4s

```
Training Set Accuracy:  0.9886666666666667
Testing Set Accuracy:  0.9893333333333333
```

使用 Bernoulli Naïve Bayes 所得到的結果分別是:
訓練集準確率:0.991555555555
測試集準確率:0.994

```python
# Bernoulli Naive Bayes: 主要用在二元特徵，比方說特徵是否出現、特徵大小、特徵長短等等這種的二元分類。
from sklearn.naive_bayes import BernoulliNB
# 建立naive模型
BernModel = BernoulliNB()
# 使用訓練資料訓練模型
BernModel.fit(X_train,y_train)
# 預測成功的比例
print('Training Set Accuracy: ',BernModel.score(X_train,y_train))
print('Testing Set Accuracy: ',BernModel.score(X_test,y_test))
```

✓ 0.7s

```
Training Set Accuracy:  0.9915555555555555
Testing Set Accuracy:  0.994
```

使用 Multinomial Naïve Bayes 所得到的結果分別是:
訓練集準確率:0.902666666666
測試集準確率:0.907333333333

```
# Multinomial Naive Bayes: 主要用在離散變數，比方說次數、類別等等。
from sklearn.naive_bayes import MultinomialNB
# 建立naive模型
MultiModel = MultinomialNB()
# 使用訓練資料訓練模型
MultiModel.fit(X_train,y_train)
# 預測成功的比例
print('Training Set Accuracy: ',MultiModel.score(X_train,y_train))
print('Testing Set Accuracy: ',MultiModel.score(X_test,y_test))
```

✓ 0.4s

```
Training Set Accuracy:  0.9026666666666666
Testing Set Accuracy:  0.9073333333333333
```

結論:

可看出 Bernoulli Naïve Bayes 所得到的結果較好，因為它主要適用於二元特徵，比如說特徵是否出現等等，跟我設計的資料集相符度甚高。

4. Random Forest

訓練集準確率:1.0　測試集準確率:1.0

使用 Random Forest 可以發現，每個特徵都被列為參考對象，只是重要程度的區別，最重要的特徵是 Class_type:0.2495，再來我們的 Absolute right rules 也都有被篩選出來，依重要程度依序為:Attitude:0.21212888、Pass_rate:0.21175923、Care:0.17424444、Prepare_hours:0.13904489，最不重要的特徵是 glasses:0.00058，最後一張圖呈現前 4 重要的樹。

```python
from sklearn.ensemble import RandomForestClassifier

# 建立 Random Forest Classifier 模型
randomForestModel = RandomForestClassifier(n_estimators=100, criterion = 'gini')
# 使用訓練資料訓練模型
randomForestModel.fit(X_train, y_train)
# 預測成功的比例
print('訓練集準確率: ',randomForestModel.score(X_train,y_train))
print('測試集準確率: ',randomForestModel.score(X_test,y_test))
```

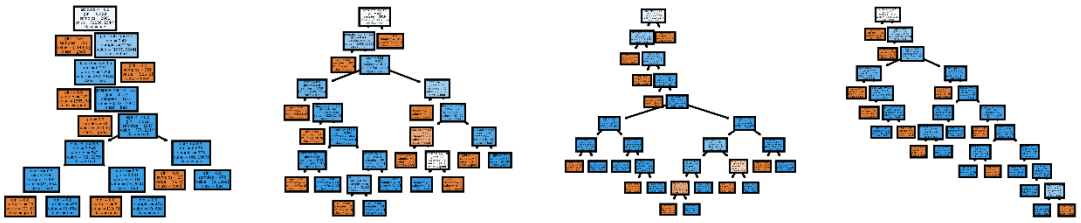
✓ 0.2s

```
訓練集準確率:  1.0
測試集準確率:  1.0
```

```python
print('特徵重要程度: ',randomForestModel.feature_importances_)
```

✓ 0.3s

```
特徵重要程度:  [0.00334569 0.0006175  0.00386551 0.00400958 0.21212888 0.13904489
 0.24955932 0.21175923 0.00058743 0.00083753 0.17424444]
```



5. XGBoost

可看出 Boosting 方法跑出來的結果相當好，且觀察第二張圖，Absolute right rules 都有被篩選出來，分別是陣列中第 4,5,6,7,10 個特徵，依序為: Attitude, Prepare_hours, Class_type, Pass_rate,Care

```python
from xgboost import XGBClassifier

# 建立 XGBClassifier 模型
xgboostModel = XGBClassifier(n_estimators=100, learning_rate= 0.3)
# 使用訓練資料訓練模型
xgboostModel.fit(X_train, y_train)
# 預測成功的比例
print('訓練集準確率: ',xgboostModel.score(X_train,y_train))
print('測試集準確率: ',xgboostModel.score(X_test,y_test))
```

✓ 0.7s

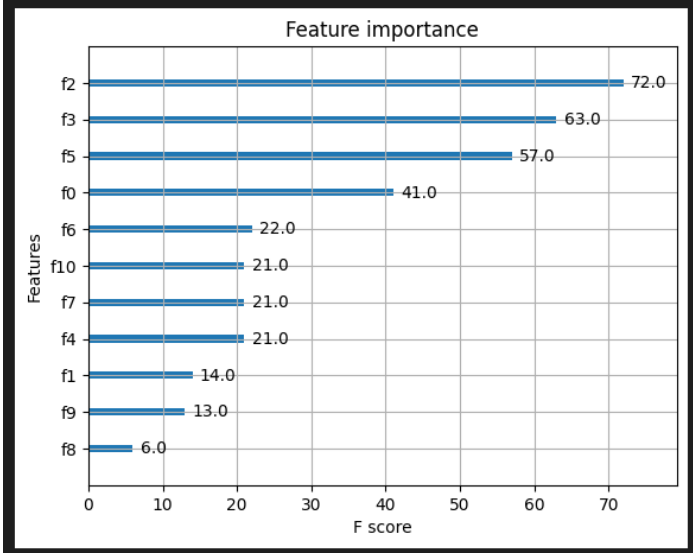
```
訓練集準確率:  1.0
測試集準確率:  1.0
```

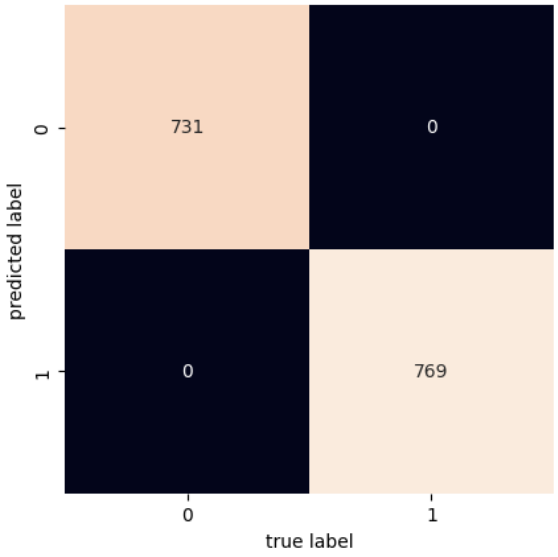
```python
from xgboost import plot_importance
from xgboost import plot_tree

plot_importance(xgboostModel)
print('特徵重要程度: ',xgboostModel.feature_importances_)
```

✓ 0.1s

```
特徵重要程度:  [3.6895433e-06 3.5382461e-06 2.7120802e-06 2.7072338e-06 2.8607836e-01
 1.8437011e-02 3.5273343e-01 2.2070520e-01 2.9075502e-06 4.5679467e-06
 1.2202578e-01]
```





Discussion:

Slightly alter the absolutely-right rules and generate another set of data; run the classifcation models on this set of data and include your observations in this section.

原本的 **Absolutely-right rules:**

Attitude: Serious	#教學態度嚴肅
Prepare_hours: 3-6	#備課時間達 3-6 小時
Pass_rate: High	#學生通過課程比例高
Care: Yes	#會關心學生
Class_type: Physical	#實體授課

微改後的 **Absolutely-right rules:**

Attitude: Serious	#教學態度嚴肅
Prepare_hours: 2-6	#備課時間達 2-6 小時
Pass_rate: High	#學生通過課程比例高
Care: Yes	#會關心學生
Class_hw:Middle/High	#回家作業比例(中/多)
Class_type: Physical	#實體授課

共 1000 筆資料，其中”好老師”資料佔 513 筆，”壞老師”資料佔 487 筆

生成資料檔案:data_produce_alter.py

資料集:teacher_data_alter.csv

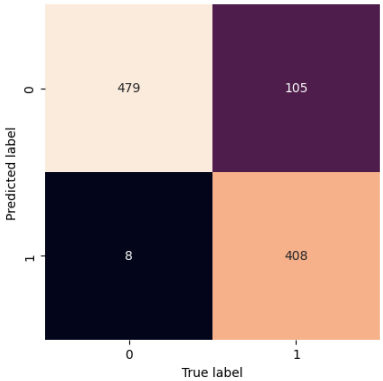
```
[1000 rows x 12 columns]
positive 513 / negative 487
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	age	gender	height	weight	attitude	prepare_hc	class_type	pass_rate	glasses	classhw	care	label
2	33	M	166	40	Serious	2	Physical	High	1	Middle	Yes	good
3	35	M	167	96	Serious	2	Physical	High	1	High	Yes	good
4	53	F	179	63	Free	4	Physical	High	1	Low	Yes	bad
5	39	F	159	55	Free	4	Physical	Low	1	Low	Yes	bad
6	63	F	181	56	Serious	5	Online	High	1	Low	Yes	bad

1. Decision Tree

   準確率:0.887

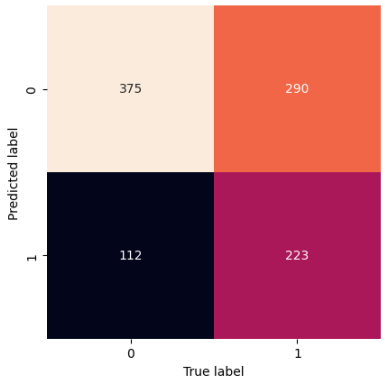
```
微調後資料集準確率: 0.887
```



2. KNN

   準確率:0.598

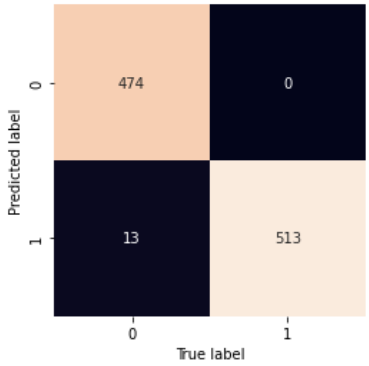
```
微調後資料集準確率: 0.598
```



3. Naïve Bayes (Bernoulli Naïve Bayes)

   準確率:0.987

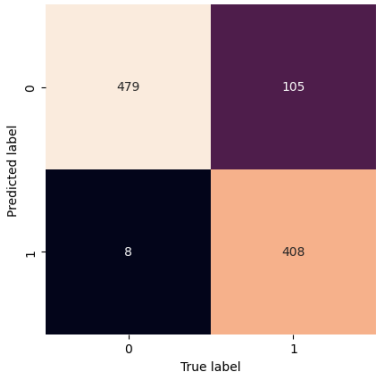
```
微調後資料集準確率: 0.987
```



4. Random Forest

   準確率:0.887

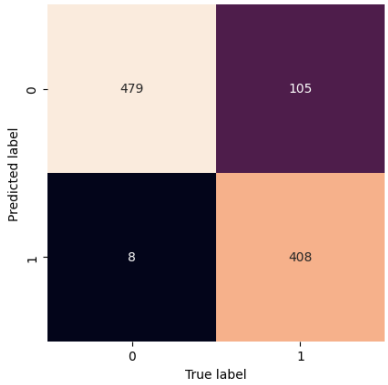
```
微調後資料集準確率: 0.887
```



5. XGBoost

   準確率:0.887

```
微調後資料集準確率: 0.887
```



結論:

我微調了 Absolutely-right rules 並生成對應資料集，再各自丟到了不同的 classify model 做分類，原本預期 XGBoost 會是 5 個裡面最好的，【因為 Bagging 透過隨機抽樣的方式生成每一棵樹，每棵樹彼此獨立並無關聯。隨機森林就是 Bagging 的實例。而 Boosting 則是透過序列的方式生成樹，後面所生成的樹會與前一棵樹相關。XGBoost 就是 Boosting 方法的其中一種實例。正是每棵樹的生成都改善了上一棵樹學習不好的地方，因此 Boosting 的模型通常會比 Bagging 還來的精準。】，但是可以看到 Decision Tree、Random Forest、XGBoost 這 3 個分類模型的準確率和 heatmap 跑出來的結果都一模一樣，我個人覺得他們學到的特徵和最後建出來的樹應該都是相差不遠的，造成此結果的原因可能是我本身設計的資料集的 feature 就不夠複雜，大部分影響結果的重要的 feature 都只有 2-3 類，所以即便我微調了 Absolutely-right rules，結果也是一樣的。另外，不難看出 KNN 因為微調了 Absolutely-right rules，所以資料被混得更亂了，導致準確率很低。最讓我意外的是 Naïve Bayes 準確率高達 98%，應該是運氣好，為此，我另外設計了一個 Absolutely-right rules(這邊沒有呈現出來)，Naïve Bayes 的準確率就沒有比其他來的好。