Machine Learning

HW1：Maximum A Posteriori probability

姓名：陳毅

學號：111063577

老師：孫民

日期：3/27

首先先匯入 pandas 與 numpy 模組並且設定成縮寫以便後面的程式撰寫，其中 pandas 讀取資料，Numpy 用來陣列運算處理。

1. train_test_split(x,y,random = None)

由於資料集的數據雜亂且隨機，我們必須定義一個函式去把資料分成訓練集與測試集，而題目中有 483 種資料，我們必須分割成 60 個測試資料與 423 訓練資料，其中每種測試資料必須包含 20 筆資料。

我們先定義一個叫 **train_test_split(x,y,random = None)** 的函式，首先我們設定隨機種子以便產生重複的隨機數序列，確保每次運行的隨機數序列是相同的，接著我們把資料打亂，使用 **np.random.permutation(len(x))**來生成長度為 x(特徵標籤)的隨機排列序列，接著使用 **np.unique(y)**去取得 y(目標標籤)的種類，並且定義每個類別所需要選取的測試樣本數，後面我們使用 **np.where(y==s)[0]**去獲取 y 中該類別的索引，接著使用 **np.delete(x_sum,slice(each_test_type, None), axis=0)**從總樣

本數中刪除[20:]當作測試資料，訓練資料也相同概念，最後合併回傳。

接著將資料轉為 pandas 數據框以便合併特徵標籤與目標標籤，最後合併且匯出.csv 檔。

2. MyGaussianNB

在這個函數中我初始化了四個變數，分別為：

self.classes：記錄區分類型標籤。

self.prior：記錄對應到的先驗機率。

self.means：記錄各類型的平均數。

self.variances：記錄各類型的變異數。

其中在我定義的 MyGaussianNB 中，實作了兩個函式，分別為 fit(self, x, y)與 predict(self, x)。

fit(self, x, y)：先計算各類別組數，接著初始化各項參數與定義各項長度與數組，再用 for 去歷遍所有類別，其中 i 為索引 c 為標籤，prior 的計算為(當前類別的數量/總樣本數)，平均數與變異數的計算為按照各個特徵計算平均數和變異數，最後存取在第 i 個位置上。

predict(self, x)：我們先獲取了輸入資料 X 的樣本數量。接下來創建一個大小為 (n_samples, len(self.classes)) 的 posteriors 矩陣，其中 n_samples 是輸入資料 X 的樣本數量，而 len(self.classes) 是分類器中總共有幾個類別。

然後進入 for 迴圈，對每個類別進行以下操作：

先計算出該類別的先驗機率（即 self.prior[i]），並取自然對數（np.log()），而此步驟可以利用取 log 來使的數據正規化，較難出現 underflow 的問題，再分別取出該類別的平均值（self.means[i, :]）和方差（self.variances[i, :]）。

接著計算該類別對應每個輸入資料點的似然值，也就是高斯分布的機率密度函數，其中 x 是輸入資料點，mean 是該類別的平均值，var 是該類別的方差。這部分的計算使用了簡單的高斯分布公式。

最後，將先驗機率和似然值相加，得到後驗機率，並存入 posteriors 矩陣中對應的位置（posteriors[:, i]）。

最後，使用 np.argmax() 函數找到 posteriors 矩陣每個輸入資料點最大值對應的類別索引，並返回對應的類別。

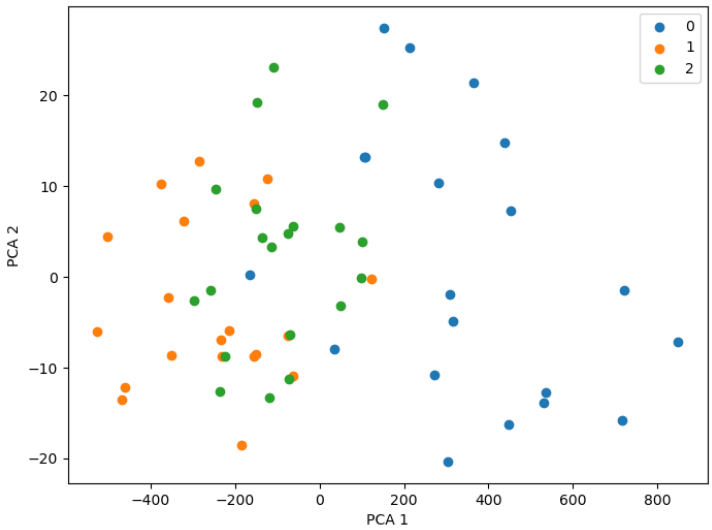
```
# 讀取數據
data = pd.read_csv('D:/wine.csv')
x = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

x_test,x_train,y_test,y_train = train_test_split(x,y,random = None)

clf = MyGaussianNB()
clf.fit(x_train, y_train)

# 計算測試集的預測分數
accuracy = clf.score(x_test, y_test)
print('test:', accuracy)

# 將訓練資料與測試資料轉化為pandas數據框
train_df = pd.DataFrame(x_train)
train_df.insert(0, 'class', y_train)
test_df = pd.DataFrame(x_test)
test_df.insert(0, 'class', y_test)

# 存取資料為csv檔
train_df.to_csv('train_data.csv', index=False)
test_df.to_csv('test_data.csv', index=False)
```

```
test: 0.9833333333333333
```

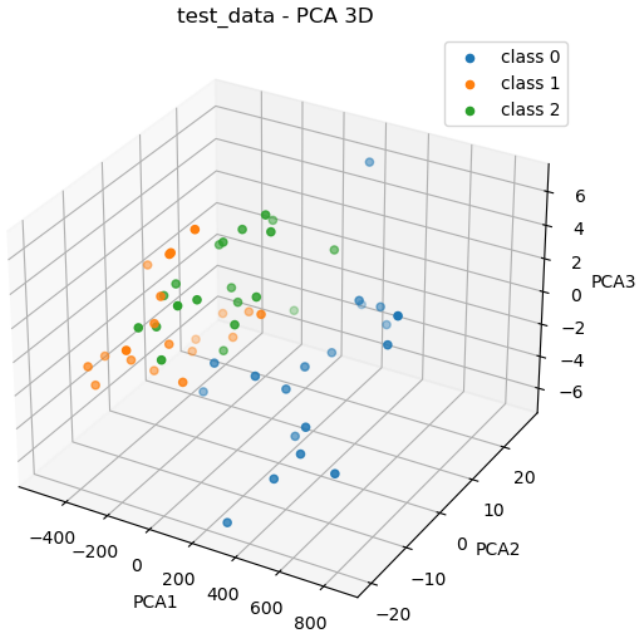
Test_score：0.983

3. Visualized result of testing data

此部分我們利用 scikit-learn 的 PCA 將 testing data 從 13 維降成 2 維及 3 維並顯示出來。從圖中我們可以發現大部分的種類會匯聚在一塊，其中有些點幾乎重疊，那是因為預測的結果不是 100%準確，但大部分的點會明顯區分出來。


0
1
2
PCA 2
PCA 1
20
10
0
−10
−20
−400
−200
0
200
400
600
800

2 維作圖


test_data - PCA 3D
class 0
class 1
class 2
PCA1
PCA2
PCA3
−400
−200
0
200
400
600
800
20
10
0
−10
−20
6
4
2
0
−2
−4
−6

3 維作圖

4. The effect of prior distribution

由結果可以看到少了 prior 那項所計算出的 score 稍微低了點，推測因為 MAP 方法引入了先驗分佈的知識，因此在估計參數時可以更充分地利用先驗分佈的信息。因此 MAP 的優點在於可以減少受到資料集 outlier 的影響，可以更加 robust，但是缺點是如果我們沒有好的 prior，估計出來的值會偏差得更嚴重。

```
    def predict(self, x):
        n_samples = x.shape[0]
        posteriors = np.zeros((n_samples, len(self.classes)))
        for i, c in enumerate(self.classes):
            prior = np.log(self.prior[i])
            mean = self.means[i, :]
            var = self.variances[i, :]
            likelihood = np.sum(
                -0.5 * ((x - mean) ** 2 / var) - 0.5 * np.log(2 * np.pi * var), axis=1)
            posteriors[:, i] =  likelihood
        return self.classes[np.argmax(posteriors, axis=1)]

    def score(self, x, y):
        y_pred = self.predict(x)
        return accuracy_score(y, y_pred)

# 读取数据
data = pd.read_csv('D:/wine.csv')
x = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(x, y, random=None)

no = noprior()
no.fit(x_train, y_train)

# 计算测试集的预测分数
accuracy = no.score(x_test, y_test)
print('test:', accuracy)
```

```
test: 0.9408983451536643
```

Test_score：0.94

參考資料：

1.https://playround.site/?p=632

2. https://ithelp.ithome.com.tw/articles/10282668

3. https://pyecontech.com/2020/03/06/python_bayesian_classifier/