Python Data Science
&
Machine Learning

Weighted and Regularized kNN for Classification

Challenge 1

Group 7

Team Members:

109613051 蘇品瑜　110550143 洪巧芸

110511097 孫書恆　111511223 羊政宇

K-最近鄰居（K-Nearest Neighbors，簡稱 KNN）是一種用於分類和回歸的監督式機器學習方法。它的主要思想是基於鄰近的點來做出預測或分類。

1. 主要運作模式分為 3 步驟:

步驟 1. 計算每個點之間的距離

收集數據：首先，你需要收集包含特徵和對應標籤（類別或數值）的訓練數據。這些特徵是描述資料點的屬性，而標籤是我們要預測或分類的目標。

步驟 2. 用 K 值決定鄰居數目，並進行投票（在連續型資料中，則是計算平均數）

選擇一個適當的 K 值：K 代表你將使用多少個最接近的鄰居來進行預測。選擇正確的 K 值很重要，因為它會影響模型的性能。預測或分類：對於分類問題，KNN 將根據這 K 個鄰居的多數類別來預測新數據點的類別。對於回歸問題，KNN 將根據這 K 個鄰居的平均值或加權平均值來預測新數據點的數值。

步驟 3. 以投票結果決定類別

評估模型性能：使用測試數據來評估模型的性能，通常使用評估指標如精確度、均方誤差（MSE）、或其他適合的指標。

KNN 的數學原理相對簡單，主要涉及到距離計算和鄰居的選擇。以下是 KNN 的一些數學原理：

歐基里德距離 (Euclidean distance)

這其實就是我們平時最熟悉的距離計算模式:

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

曼哈頓距離 (Manhattan distance)

曼哈頓距離的計算方式是將所有特徵的個別距離加總在一起:

$$D = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n|$$

$$= \sum_{i=1}^{n} |x_i - y_i|$$

明氏距離 (Minkowski distance)

有點像是歐基里德距離與曼哈頓距離的推廣，仔細對照公式會發現，當p=1時其實就是曼哈頓距離，而當p=2時則為歐基里德距離。在這邊，p為任意常數。

$$D = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

2. 決定 K 值，並進行投票：

根據計算的距離，我們選擇與測試數據點距離最近的 K 個鄰居。

計算完距離以後，我們可以決定常數 K，也就是有多少個鄰居被我們視為最接近的鄰居。舉例來說，如果我們選擇 K=3，那我們就會以距離最接近的 3 個點當作鄰居，並看這 3 個鄰居當中有多少個鄰居是屬於哪一種類別，並以數量最多的類別取勝。以下圖來說，當 K=3 時，因為藍色水滴有 2 個，黃色倒藍水滴只有 1 個，因此我們會判定星星為藍色水滴。

總的來說，KNN 是一個簡單但有效的機器學習方法，特別適用於小型數據集和非線性問題。但它對大型數據集的計算成本較高，並且對 K 值的選擇敏感，因此需要謹慎調參以達到最佳性能。

- **Code Explanation**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline, FeatureUnion
```

Import 所有需要的 libraries。

```
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

分別將 train data 和 test data 兩個 csv 檔案轉為 pandas 的格式，以進行之後 numpy 的運算。

```
# Extract numerical features only
x_train = train_data.select_dtypes(include=['float64'])
x_test = test_data.select_dtypes(include=['float64'])
y_train = train_data['class_num']  #specify the label that we want to predict
y_test = test_data['class_num']
```

利用 select_dtypes 此一 funciton 將 train data 和 test data 兩筆資料中包含 string 格式的特徵接去除，僅留下以 float 表示之特徵值，方便進行資料的標準化[註 1]。其中以 x 表示用來訓練 model 的特徵資料，y 則為想要預測的目標特徵，以此次作業來說 y 就是 class number。

```
# replace NaN value
for column in x_train:
    x_train[column] = x_train[column].replace(0, np.NaN)
    mean = int(x_train[column].mean(skipna=True))
    x_train[column] = x_train[column].replace(np.NaN, mean)

for column in x_test:
    x_test[column] = x_test[column].replace(0, np.NaN)
    mean = int(x_test[column].mean(skipna=True))
    x_test[column] = x_test[column].replace(np.NaN, mean)
```

若是資料庫的資料不完整，包含缺失的數值(NaN)，將影響整體 KNN 計算距離之結果。因此我們將所有 NaN 值以平均值填入，避免影響計算結果。

```
# Feature scaling
std_scaler = StandardScaler()
std_x_train = std_scaler.fit_transform(x_train)
std_x_test = std_scaler.fit_transform(x_test)
```

因為每個特徵的尺度大小不一，因此需要將資料進行標準化的動作。例如，特徵 H 是小行星的絕對幅度參數(Absolute magnitude parameters)，其數值由 1~20 不等；而特徵 e，偏心率(Eccentricity)則只有 0.0 幾的大小。因此我們需要對資料進行標準化，將每筆數據都利用標準化調整成只有 0~1 的大小，如此進行 KNN 距離計算時，才不會有特徵的比重被加深或是有特徵被遺忘。

```
# 創建 KNN 模型並訓練
knn = KNeighborsClassifier(n_neighbors=5,p=1,weights='distance')
knn.fit(X_train, y_train)
```

將資料進行標準化後即可創建 KNN 的模型，利用 sklearn.neighbors library 中的 KNeighborsClassifier 創建模型，並且可指定 K 的數值(n_neighbors)，在圖中即是指定 K=5，而 p 則是 Minkowski metric 的次方參數，當 p=1 時，使用的是 manhattan_distance，p=2 時則是

euclidean_distance。若未指定則是使用預設值 minkowski_distance。
我們選擇使用的是，Manhattan_distance，其計算方式為將所有特徵的個別距離加總在一起。公式如下：

$$D = |x1 - y1| + |x2 - y2| + \cdots + |x_n - y_{n|} = \sum_{i=1}^{n} |x_i - y_i|$$

創建完成 KNN 模型後將 X_train 和 y_train 丟入模型中，進行訓練。

```
# 預測測試數據中的 class_num
y_pred = knn.predict(X_test)

# 提取 id 和預測結果
ids = test_data['id']
predicted_class_num = y_pred
```

利用 test data 之中的其他數據丟入已經用 train data 訓練完成的 KNN 模型之中，預測 test data 之中我們想要預測的 class_num 此一特徵。

最後為了輸出想要的 csv 檔格式，提取 test data 中的 index 和預測出來的結果。

```
# 添加 "IDX, Target" 到結果 DataFrame
result_df = pd.DataFrame({'IDX': test_data['idx'], 'Target': y_pred})

# 將結果保存為 CSV 文件
result_df.to_csv('predicted_results0.csv', index=False)
```

利用 pandas 以 index 和 Target 為 column 的標題，將結果輸出成一個 pandas 的 data frame。並利用 to_csv()此一函數輸出結果檔案。

- **Output Results**

```
# Evaluate Model
confusion_mat = confusion_matrix(y_test, y_pred)
print(confusion_mat)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    257041
           1       0.83      0.79      0.81      8380
           2       1.00      0.99      1.00      2443
           3       0.97      0.95      0.96      3792
           4       0.94      0.89      0.92      5556
           5       0.97      0.98      0.98      6068
           6       0.91      0.93      0.92      2521
           7       0.99      0.98      0.98       569
           8       0.98      1.00      0.99      1022
           9       0.96      0.83      0.89       136
          10       0.82      0.58      0.68        24
          11       0.75      0.75      0.75         4
          12       1.00      1.00      1.00         1

    accuracy                           0.98    287557
   macro avg       0.93      0.90      0.91    287557
weighted avg       0.98      0.98      0.98    287557
```

以 k=5 為例，利用 classification_report 輸出結果顯示，預測 class_num 的 accuracy 可以高達 0.985。而 k=1 時，準確率則僅有 0.97，若改變 k 值則準確率會隨之改變，經測試後 k>5 之後的準確率會大幅下降，因此我們選擇 k=5 作為最終想要保留的鄰居數量作為判斷標準。

- **改變訓練模型時使用的特徵數量**

```
# 提取特徵（semi-major-axis 'a' 和 'H'）和目標變數（class_num）
X_train = train_data[['a', 'H','q','i','om','w','ma','ad','n','tp','tp_cal','per','per_y','class_num']]
y_train = train_data['class_num']
X_test = test_data[['a', 'H','q','i','om','w','ma','ad','n','tp','tp_cal','per','per_y','class_num']]
```

```
# 提取特徵（semi-major-axis 'a' 和 'H'）和目標變數（class_num）
X_train = train_data[['a', 'H']]
y_train = train_data['class_num']
X_test = test_data[['a', 'H']]
```

藉由更改丟入訓練模型 KNN 的資料特徵數量，我們發現當能作為判斷依據的特徵值越少，預測的準確率會隨之下降。例如，上圖使用 14 種特徵訓練模型，其預測準確率為 0.98；而下圖僅使用 2 個特徵訓練模型時則準確率僅有 0.95。因此我們可知當訓練模型擁有越多的判斷數據則最後得到的模型可預測的數值則越加準確。

- **Code Optimization**
  1. **更改 weight 模式：**
     可指定 KNN 中的加權(weight)模式，例如，uniform weight 以及 distance weight。此次使用的是 uniform weight，意即所有最近的鄰居的權重都是相等的；而 distance weight 則會依照預測樣本到不同鄰居的距離去改變此樣本對預測結果的影響權重，越近的鄰居對預測結果的影響越大。
  2. **利用編碼，使用所有特徵值，包括非 float 之特徵：**
     我們可以利用不同的編碼方式將擁有字符的特徵數值轉換為二元表示方法，如此一來這些數值也同樣可以利用計算距離的方式成為訓練時的特徵值。但要注意的是，並不是每個特徵對於資料的預測都有幫助，可以多加測試、比較後決定訓練模型時欲使用哪些數據才能得到最好的預測結果。