

## 1. Classifier Used

這次的實驗作業我建立了兩種 supervised 模型去預測我的資料，其一就是 Logistic Regression，另外一個就是 K-Nearest-Neighbors (KNN)，以下簡述一下兩者的運作原理：

### a. Logistic Regression

Logistic Regression 是利用 Linear Regression 的方式先將資料點做運算再代入 Logistic Function，像是 sigmoid function 然後將資料根據機率做預測分類。而訓練的過程就是利用 gradient descent 的方式去調整 Linear Regression 的參數，最後在預測階段就將訓練好的參數代入式子中對資料點進行預測。

### b. K-Nearest-Neighbors (KNN)

K-Nearest-Neighbors 這個模型特別的一點就是他沒有訓練過程，它對資料進行預測純粹就以樣本空間(以這次作業而言為  $X_{train}$ )中和受測點距離最近的  $k$  個鄰居，針對這  $k$  個鄰居的標籤取眾數去決定它的預測結果，而計算點與點之間距離的方法有很多種，而我利用的是最典型的方式：Euclidean Distance，也就是平方合開根號距離。

## 2. Data Preprocessing

在討論 data pre-processing 以前，先在此簡單描述一下資料的存取方式，根據這次作業目標，我們要針對 12 個特徵(一些骨頭角度等資料)去預測有沒有背痛(0 或 1 的二分類)， $X_{train}$ 、 $X_{test}$  為二為矩陣，每個 row 存一筆資料，而每個 column 為一個 feature(一共 12 個欄位)， $y_{train}$  則為一個一為陣列，每個位置對應  $X_{train}$  的資料列位置，存取 0 或 1。  
針對這次的資料，我有嘗試進行了以下步驟：

### (1) Shuffle

因為因應後面會提到的 cross validation，為避免 raw data 有分布不均的可能性，我在讀入資料後有將一筆一筆的資料重新洗牌增加隨機分佈的特性，以免影響資料的訓練和模型的優化。

### (2) Normalize

根據每一行(每一個特徵)，將資料點減去最小值除以極值差(最大值減去最小值)，如下公式：

$$\text{normalized } x = \frac{x - \text{column min}}{(\text{column max} - \text{column min})}$$

這樣可以將資料分佈於 0 和 1 之間，這樣是避免不同特徵在數值上面有不同的區間範圍(例如正負號之差)差異影響訓練的參數和結果。

### (3) Standardize

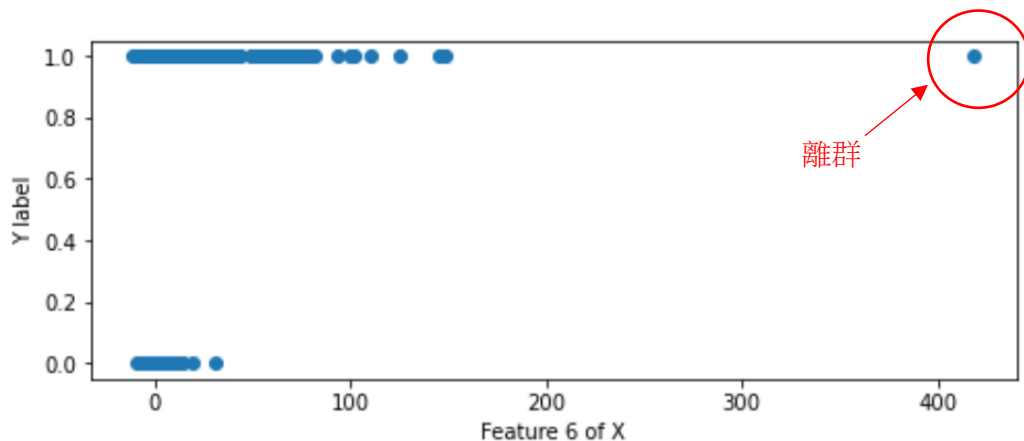
根據每一行(每一個特徵)，將資料點減去行平均值除以行標準差，如以下公式：

$$\text{standardized } x = \frac{x - \text{mean of column}}{\text{standard deviation of column}}$$

將資料標準化後，每一個欄位的資料平均會是 0，標準差呈 1，之所以會對資料進行標準化是要避免不同的特徵單位(例如數值大小呈倍數差異)影響訓練的參數和結果。

### (4) Remove Outliers

在進行訓練以前的另一關鍵要點就是替除資料中的離群值，避免離群資料(像是錯誤資料、奇異資料點)對訓練有影響造成偏差，因為這次的資料有 13 個維度有點難視覺化，所以我就產出二維圖形——觀測單一特徵中資料和分類的分佈關係，舉下圖為例：



可以從第六特徵和分類關係圖中看到右上角有一數據點和其他資料點相距比較遠，這樣可以看出  $x$  中的這一點以第六特徵而言有離群的可能性。為了進一步分辨離群值，我建了一個  $z$  score 的矩陣，把 **threshold** 設為 3， $Z$  值大於 3 的資料都會被塞選出來，再把這些選出來的資料剔除就將資料集中的離群值在初步的處理階段移除了。

### 3. Improve Performance

#### (1) Train-test split

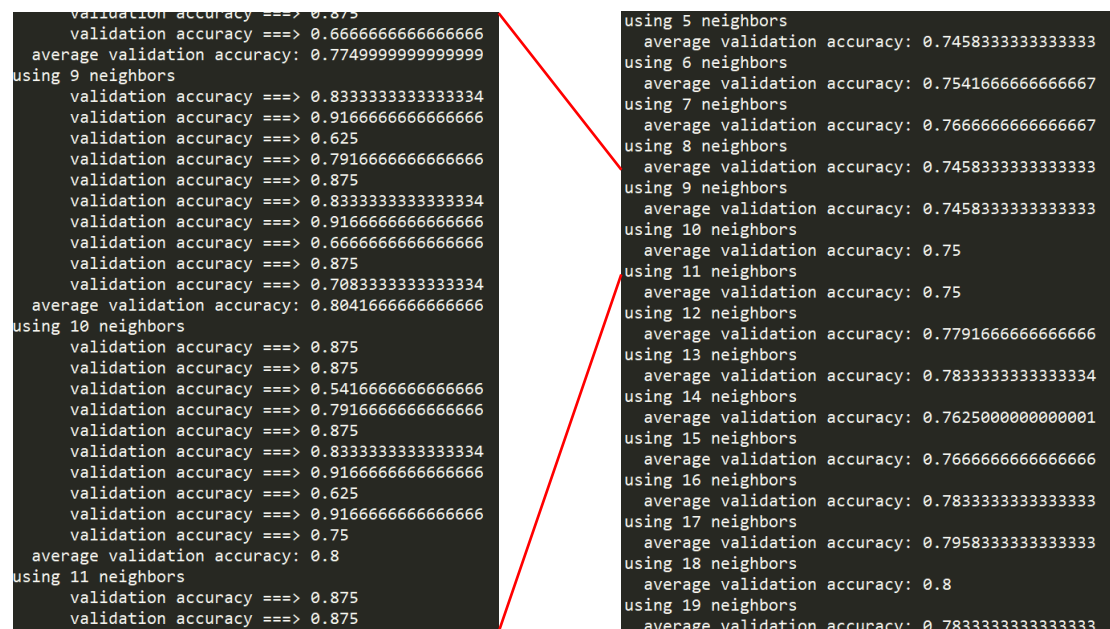
一開始我將 `X_train` 資料以 3:1 的比例再切成訓練和測試資料條參數，但後來發覺這樣的模式太慢了，而且一旦測試和訓練資料有不均的分配就很容易照呈誤差，所以改用了接下來會提到的 `cross validation`。

#### (2) Cross-validation

這個方法適合用在資料量少的資料集上(像我們這次的訓練資料就只有兩百多筆)，原則上就是將資料切成  $n$  等份，輪流拿一等份當測試資料集 (validation set)，剩下的  $n-1$  等份當訓練資料集。我這次就是利用這個方法進行模型的比較以及模型參數的優化。(本次作業我將  $n$  設為 10)

##### (i) Cross Validation on K-Nearest-Neighbors

下圖為調  $k$  參數的優化過程，適用取不同鄰居數看會不會稱加預測命中機率，過程可以看到跑十次分別的測試命中率以及最終的平均命中率，以右圖得比較可看出，整體平均數大約介於 75~80% 之間，而取  $k=16$ 、 $17$  會得到較高的預測命中率。

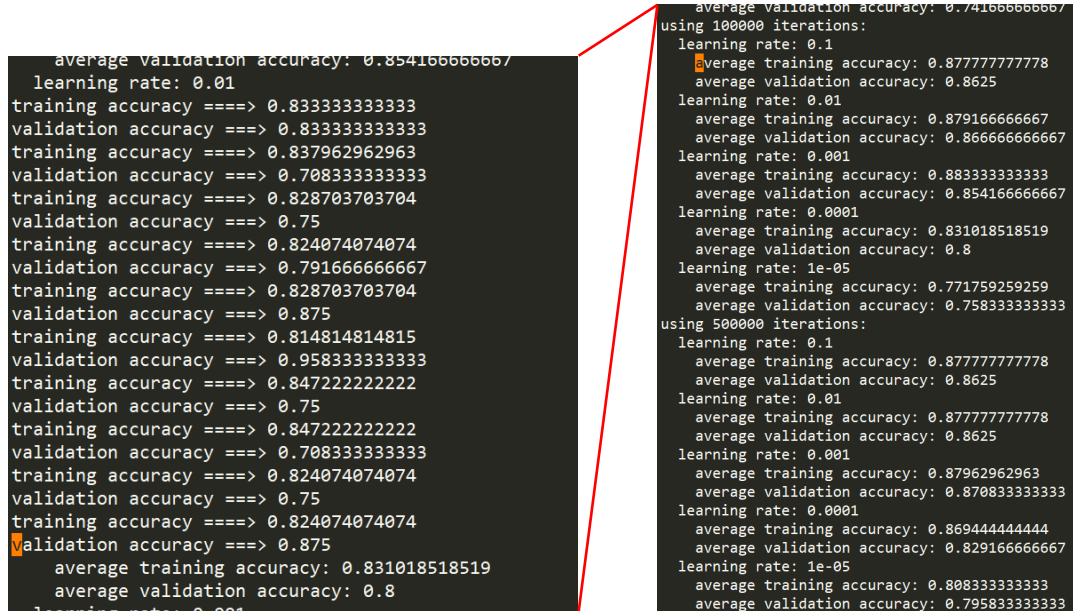


```
validation accuracy ==> 0.875
validation accuracy ==> 0.6666666666666666
average validation accuracy: 0.7749999999999999
using 9 neighbors
validation accuracy ==> 0.8333333333333334
validation accuracy ==> 0.9166666666666666
validation accuracy ==> 0.625
validation accuracy ==> 0.7916666666666666
validation accuracy ==> 0.875
validation accuracy ==> 0.8333333333333334
validation accuracy ==> 0.9166666666666666
validation accuracy ==> 0.6666666666666666
validation accuracy ==> 0.875
validation accuracy ==> 0.7083333333333334
average validation accuracy: 0.8041666666666666
using 10 neighbors
validation accuracy ==> 0.875
validation accuracy ==> 0.875
validation accuracy ==> 0.5416666666666666
validation accuracy ==> 0.7916666666666666
validation accuracy ==> 0.875
validation accuracy ==> 0.8333333333333334
validation accuracy ==> 0.9166666666666666
validation accuracy ==> 0.625
validation accuracy ==> 0.9166666666666666
validation accuracy ==> 0.75
average validation accuracy: 0.8
using 11 neighbors
validation accuracy ==> 0.875
validation accuracy ==> 0.875

using 5 neighbors
average validation accuracy: 0.7458333333333333
using 6 neighbors
average validation accuracy: 0.7541666666666667
using 7 neighbors
average validation accuracy: 0.7666666666666667
using 8 neighbors
average validation accuracy: 0.7458333333333333
using 9 neighbors
average validation accuracy: 0.7458333333333333
using 10 neighbors
average validation accuracy: 0.75
using 11 neighbors
average validation accuracy: 0.75
using 12 neighbors
average validation accuracy: 0.7791666666666666
using 13 neighbors
average validation accuracy: 0.7833333333333334
using 14 neighbors
average validation accuracy: 0.7625000000000001
using 15 neighbors
average validation accuracy: 0.7666666666666666
using 16 neighbors
average validation accuracy: 0.7833333333333333
using 17 neighbors
average validation accuracy: 0.7958333333333333
using 18 neighbors
average validation accuracy: 0.8
using 19 neighbors
average validation accuracy: 0.7833333333333333
```

## (ii) Cross Validation on Logistic Regression

我對於 Logistic Regression 相較 KNN 調的參數比較多，一樣的，我先對 learning rate 還有 training iterations 進行優化，普遍上來說預測命中率可以落在 80~88%之間，略高於 KNN 的 cross validation。



```
average validation accuracy: 0.85416666667
learning rate: 0.01
training accuracy ==> 0.833333333333
validation accuracy ==> 0.833333333333
training accuracy ==> 0.837962962963
validation accuracy ==> 0.708333333333
training accuracy ==> 0.828703703704
validation accuracy ==> 0.75
training accuracy ==> 0.824074074074
validation accuracy ==> 0.791666666667
training accuracy ==> 0.828703703704
validation accuracy ==> 0.875
training accuracy ==> 0.814814814815
validation accuracy ==> 0.958333333333
training accuracy ==> 0.847222222222
validation accuracy ==> 0.75
training accuracy ==> 0.847222222222
validation accuracy ==> 0.708333333333
training accuracy ==> 0.824074074074
validation accuracy ==> 0.75
training accuracy ==> 0.824074074074
validation accuracy ==> 0.875
average training accuracy: 0.831018518519
average validation accuracy: 0.8
learning rate: 0.001

average validation accuracy: 0.74166666667
using 100000 iterations:
learning rate: 0.1
average training accuracy: 0.877777777778
average validation accuracy: 0.8625
learning rate: 0.01
average training accuracy: 0.879166666667
average validation accuracy: 0.866666666667
learning rate: 0.001
average training accuracy: 0.883333333333
average validation accuracy: 0.854166666667
learning rate: 0.0001
average training accuracy: 0.831018518519
average validation accuracy: 0.8
learning rate: 1e-05
average training accuracy: 0.771759259259
average validation accuracy: 0.758333333333
using 500000 iterations:
learning rate: 0.1
average training accuracy: 0.877777777778
average validation accuracy: 0.8625
learning rate: 0.01
average training accuracy: 0.877777777778
average validation accuracy: 0.8625
learning rate: 0.001
average training accuracy: 0.87962962963
average validation accuracy: 0.870833333333
learning rate: 0.0001
average training accuracy: 0.869444444444
average validation accuracy: 0.829166666667
learning rate: 1e-05
average training accuracy: 0.808333333333
average validation accuracy: 0.795833333333
```

## (iii) Finalize and Choose Model

KNN : 上 Kaggle 跑得到最好的結果為 0.72093%

Logistic Regression : 上 Kaggle 跑得到最好的結果為 0.83720%

從上面做 cross validation 比較以及 Kaggle 的結果來看，Logistic Regression 具有比較高的命中機率，原因或許是因為 KNN 需要龐大的資料集材可以做較準確的預測，而我們這次的資料集資料量較少，所以或許這是為甚麼 LR 會有較好結果的原因。

## 4. Difficulties and How I Overcame Them

### (1) Building My Own Models from Scratch

本次作業遇到的困難首先當然就是要先了解模型，因為有規定不能呼叫 scikit learn 所以很多模型都必須手刻，但好家在 numpy 的函示庫夠給力，在建立模型和做運算的過程中給了不少幫助，網路上也有不少機器學習文章有講解模型背後運作原理，讓我可以比較快上手。

## (2) Inconsistent Data Pre-processing

很多資料處理的細節上一開始都會忽略，就導致自己的預測模型很多 bug，像是 training data 有做 standardization 而 testing 沒有，就讓 training 的 accuracy 和 Kaggle 上的差 10~20%，一開始以為是 overfit 的原因，但做完 cross validation 後才發現原來是自己資料處理前後不一貫，疏忽所導致，也讓我理解到做是最好齊一，先將 data 都一併做處理再進行訓練才不會落掉。

## (3) Python Versions

Python 版本也是個要注意的點，姑且不論 python 2 和 python 3 的差別，很多 3.\* 的版本就有很多細節不大一樣，像是 `statistics.mode()` 這個 function，它的功能是選出矩陣中出現次數最頻繁的值，而當有兩個以上一樣最多的值在 python3.8 以前的版本會丟出一個 error，但 3.8 之後就不會，像這樣細節上的差異容易造成在不同平台上跑出現 bug (自己電腦 v.s server)，所以我發覺要多利用 try、except 去解決類似的問題。