# 智慧化企業整合

Project2-

只甜你口，不甜你尿

糖尿病預測篩檢服務

第 7 組

108034535 施美全

108034541 周郁淇

108034546 許家銘

108034533 李岱諭

目錄

## 一、 背景介紹

1、情境描述

隨著現代人對於飲食與健康的重視程度日益提升，因過分攝取精緻甜食的糖分所引發的糖尿病盛行問題也備受關注，民眾對於自身身體狀況的關心程度亦大幅增加，除了日常飲食的控制之外，罹患糖尿病的機率亦受到許多因素影響，如家族遺傳、生理狀態等，如何防範糖尿病找上身實為我們需要深入了解的問題。

然而，醫院的健康檢查報告往往過於複雜，一般民眾難以理解數據所代表的意義，也無法自行判斷自身患有糖尿病的可能。為了解決這樣的問題，同時協助民眾更加了解自己的健康狀況，我們和新竹在地知名甜點店—1000 種味道以及地方醫院進行合作，提供民眾糖尿病篩檢的服務，讓大家能夠沒有負擔地享用美味甜點。

2、問題描述（5W1H）

在國人的十大死因中，糖尿病已經攀上第四名，且每年或者有增加的趨勢，其對於健康所帶來的威脅不容忽視。我們藉由過往糖尿病患者的歷史資料來建立篩檢模型，分析用戶的個人資料來預測其潛在風險與罹病機率，以期望能讓用戶更早了解自身身體狀況，及早發現問題並改善生活習慣，進而達到預防糖尿病的問題。

以下將透過 5W1H 進行問題定義：

| | |
|---|---|
| What? | 糖尿病的篩檢服務 |
| When? | 疾病發生前與發生時 |
| Who? | 重視自身身體狀況的民眾 |
| Where? | 服務中心與網路 |
| Why? | 糖尿病危害民眾身體健康、難以自行判斷 |
| How? | 資料分析、深度學習、機器學習 |

我們提供針對糖尿病的篩檢服務，藉由過往的數據進行資料分析與處理後，透過深度學習的概念來訓練模型，讓在乎自身身體健康的用戶能夠便利地使用這項服務，不論是處在高患病風險或已有糖尿病而不自知的狀態，都可以針對自身狀況去調整生活習慣、預防疾病，避免糖尿病危害到身體。

## 二、 模型架構

1、資料前處理過程

- 資料來源

為了建立合適的模型，我們採用 Kaggle 內的數據作為資料來源。Kaggle 是一個數據建模和數據分析競賽平台。企業和研究者可在此發布數據，提供統計學家和數據挖掘專家以此資料為基礎，進行競賽以產

生最好的模型。

此次報告使用當中“National Institute of Diabetes and Digestive and Kidney Diseases”(國家糖尿病與消化及腎臟疾病研究所)所蒐集之皮馬族的糖尿病數據庫資料，共計 768 筆資料，作為模型的資料來源。

- 資料欄位

接著我們釐清各欄位資料所代表的資訊及意涵，以使後續資料處理與模型分析得以順利進行，本次資料欄位共有下列 9 項：

(1) Pregnancies：懷孕週數
(2) Glucose：血液中葡萄糖含量
(3) Blood Pressure：血壓
(4) Skin Thickness：皮褶厚度，是用來推斷全身脂肪含量、判斷皮下脂肪發育情況的一項重要指標
(5) Insulin：血清胰島素，是一種蛋白質類激素，由胰島 b 細胞分泌的一種唯一能使血糖降低之激素
(6) BMI：Body Mass Index，BMI 值原來的設計是一個用於公眾健康研究的統計工具，為體重(公斤)除以身高(公尺)的平方
(7) Diabetes Pedigree Function：糖尿病家族函數，此屬性為病患本身家族成員是否有得過糖尿病的病史
(8) Age：病患之年齡資料
(9) Outcome：病患是否患有糖尿病，以有跟無來區分(1 為有，0 為沒有)。

- 數據分析

(1)首先，將資料庫導入，並利用 pandas 的 head()方法來檢查數據是否正確匯入。

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
diabetes = pd.read_csv('diabetes2.csv')
diabetes.columns
```

```
diabetes.head()
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

(2)獲得資料的維度以及欄位資訊：

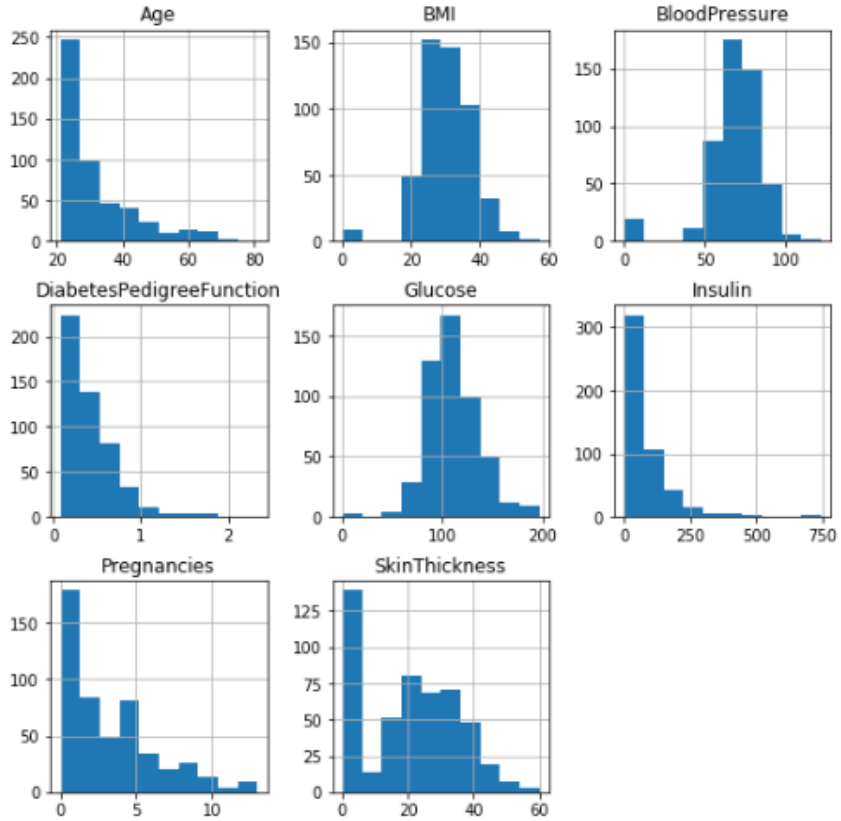
```
print("Diabetes data set dimensions : {}".format(diabetes.shape))
diabetes.groupby('Outcome').size()
```

```
Diabetes data set dimensions : (768, 9)
Outcome
0    500
1    268
dtype: int64
```

由此可知，此資料庫中共有 768 筆資料與 9 個欄位，當中欄位 Outcome 所代表的意義為使用者是否患有糖尿病(1 為有，0 為沒有)，從上列結果可知有 500 筆資料為糖尿病患者，268 筆無。

(3)利用 pandas 的 groupby 可視化工具，幫助資料進行分組，並得到各欄位之資料分布。

```
diabetes.groupby('Outcome').hist(figsize=(9, 9))
```




- 數據清理

進行數據清理時，應考量下列四項情形並視情況予以刪減：

(1) 是否有重複之資料

(2) 錯誤標示的資料

(3) 缺失或空的資料值

(4) 異常值

由於我們的數據來源為 “National Institute of Diabetes and

Digestive and Kidney Diseases”標準資料庫，經檢查後發現原資料已將(1)、(2)進行清理，故我們接著以(3)、(4)為主要清理目標。

首先，針對第三點是否有缺失或空的資料值，使用函數 isnull()和 isna()檢查資料是否有空值，並根據回傳之 boolean 值(0 或 1)進行空值刪除。

```
diabetes.isnull().sum()
diabetes.isna().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

由上列結果可得知各欄位皆無空值，因此這些資料皆以符合要求，我們並不需要再對其進行刪除與整理。接著，由先前所顯示之直方圖中可發現部分數據為異常值，故我們進行進一步的分析以找出異常值並刪除。

我們針對血壓（Blood pressure）、血糖（Glucose）、BMI 值進行分析，因此三個欄位的數值皆不可能為 0，一正常人並不會出現血壓為 0、血糖為 0 或 BMI 為 0 的情形，藉此判斷該資料是否為異常值。

```
print("Total : ", diabetes[diabetes.BloodPressure == 0].shape[0])
Total :  35
print(diabetes[diabetes.BloodPressure == 0].groupby('Outcome')['Age'].count())
```

```
Total :  35
Outcome
0    19
1    16
Name: Age, dtype: int64
```

由血壓的數據顯示，當中共有 35 筆血壓資料為 0。

```
print("Total : ", diabetes[diabetes.Glucose == 0].shape[0])
Total :  5
print(diabetes[diabetes.Glucose == 0].groupby('Outcome')['Age'].count())
Total :  5
```

```
Total :  5
Outcome
0    3
1    2
Name: Age, dtype: int64
```

由血糖的數據顯示，當中共有 5 筆血糖資料為 0。

```
print("Total : ", diabetes[diabetes.BMI == 0].shape[0])
Total :  11
print(diabetes[diabetes.BMI == 0].groupby('Outcome')['Age'].count())
```

```
Total :  11
Outcome
0    9
1    2
Name: Age, dtype: int64
```

由 BMI 的數據顯示，當中共有 11 筆血壓資料為 0。

針對前述所列之 3 種異常數值，為使模型準確度提升及更符合現實，我們採取移除異常值的動作，將這些資料刪除後得到新資料表(diabetes_mod)，資料筆數共計 724 筆，並以此資料量進行後續之模型訓練和測試。

- 特徵選取

透過將資料轉換成特徵，可以幫助建模並提升準確率，我們於建模前將資料裡所有的特徵欄位設為 X，而預測結果設為 y。

```
feature_names = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI',
                 'DiabetesPedigreeFunction','Age']
X = diabetes_mod[feature_names]
y = diabetes_mod.Outcome
```

- 標準化與樣本區分

最後，我們進行資料的標準化，主要是在機器學習時，為了避免某一特徵之標準差太大，而導致此特徵在目標函數中占主導地位，使其他特徵被淹沒之情形發生。同時，我們也將資料分為訓練組與測試組以建立後續的模型架構。

```
#標準化re-scale：整理使每一個特徵維度均值為0,變異數為1
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

```
#分訓練組與測試組
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = diabetes_mod.Outcome, random_state=66)
```

## 2、模型架構

為使模型的考慮更加全面性並且找到最適切的模型，我們分別以 Neural Network(NN)、NN+、K Nearest Neighbor(K-NN)、Multilayer perceptron(MLP)四種演算法去訓練我們的模型，以下將就各演算法進行說明：

- Neural Network(NN)

```
#設定NN模型
classifier = Sequential()
#設定輸入層,資料應為1維向量,若資料並非1維則需將其轉換成1維向量
classifier.add(Dense(input_dim = d, output_dim = 16 , init = 'uniform', activation = 'relu'))
#設定隱藏層
classifier.add(Dense(output_dim = 64 , init = 'uniform', activation = 'relu'))
#classifier.add(LeakyReLU(alpha=0.1))
#設定輸出層
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
#模型結構
classifier.summary()
```

首先介紹最基礎的 Neural Network，為一將 raw data 匯入模型經前處理後直接建立訓練模型，共有一輸入層、一隱藏層及一輸出層的基本模型。所給定之基本參數為:

(1)第一層 16 個節點(relu)，第二層 64 個節點(relu)，輸出層有一個節點(sigmoid)

(2)使用的 gradient descent optimizer 為 adam

(3)loss function 為 binary_crossentropy

```
#將資料集分割成訓練組與測試組(分配比率為3:1)
train_set,test_set, train_set_label, test_set_label = train_test_split(X, Y, stratify = diabetes_mod.Outcome)


for run in range(0,run):
#設定訓練組的情況(每步隨機抽樣batch_size個決定下一步的方向,共有epoch次學習取最高值視為該次最佳的學習方式)
#若使用steps_per_epoch 是決定每個epoch總共會走多少步 每步隨機抽樣1個資料決定下一步方向
#若使用batch_size 是決定總共會走 訓練組樣本數 步 每步 隨機以batch_size批資料共同研究找方向
    classifier.fit(train_set, train_set_label,batch_size=20 , epochs=100)
```

將 raw data 以預設比例 3:1 分類成訓練集與測試集，分次將資料放入訓練模型中，進行 100 個 epochs，總共運行五次並計算平均訓練成效。

- NN+

```
from keras.layers import Dropout

model = Sequential()
model.add(Dense(500, input_dim=7, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='sigmoid'))
model.add(Dense(2, activation='softmax'))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(x_train,y_train, epochs=1000, batch_size=70, validation_data=(x_test, y_test))
```

經過觀察上述基礎 NN 模型後，我們於文獻回顧時找到了另一種建立糖尿病預測模型的深度神經網絡(以下稱為 NN+)，其基本架構一樣為一輸入層(sigmoid)、一隱藏層(sigmoid)及一輸出層(softmax)的模

型，然而因觀察到 overfitting 現象(初始訓練之準確率為 0.98，但測試準確率僅有 0.70)，故又再加入了一層 dropout(alpha = 0.2)以忽略部分資料；而和 NN 的差別為，NN+將最後預測有無糖尿病之結果重新編碼為一個二維變數。 所給定之基本參數為:

(1)第一層改為 500 個節點，第二層改為 100 個節點

(2)使用 adam 做為 gradient descent optimizer

(3)loss function 改為 mse

(4)訓練集與測試集比例一樣為 3:1，

(5)方式為進行一次 1000 個 epochs 的訓練

- K Nearest Neighbor(K-NN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(X_train, y_train)
print('Train accuracy of kNN: {:.3f}'.format(knn.score(X_train, y_train)))
print('Test accuracy of kNN: {:.3f}'.format(knn.score(X_test, y_test)))
```

```
Train accuracy of kNN: 0.750
Test accuracy of kNN: 0.812
```

K 是一個用戶定義的常數，為一個沒有類別標籤的向量，會被歸類為最接近該點的 k 個樣本點中最頻繁使用的一類。

我們藉由改變 KNN 的 neighbor 數來訓練模型，以找出在不同 neighbor 時，對於模型準確度的影響並找出最佳值。

- Multilayer perceptron(MLP)

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(64,64,64),
                    activation='logistic',
                    solver='adam',
                    batch_size='auto',
                    learning_rate='constant',
                    learning_rate_init=0.001,
                    max_iter=1000,
                    random_state=0)
mlp.fit(X_train, y_train)
print("Train accuracy of MLP: {:.3f}".format(mlp.score(X_train, y_train)))
print("Test accuracy of MLP: {:.3f}".format(mlp.score(X_test, y_test)))
```

```
Train accuracy of MLP: 0.823
Test accuracy of MLP: 0.796
```

MLP 為一種前向傳遞類神經網路，利用倒傳遞的技術達到監督式學習，其中包含了三層結構：輸入層、隱藏層、輸出層，和 NN 類似，為 Deep NN 中的一個特例。在 sklearn 的套件中已有分類器，而我們藉由當中的各個參數進行調整來訓練模型，速度上較 NN 為快。

3、模型總結

根據前述所列的 4 種模型進行訓練後，我們將各模型依據最終準確度高低進行比較，所得到的結果如下表所示：

| | KNN | NN | MLP |
|---|---|---|---|
| **中文名稱** | 最近鄰居法 | 類神經網路 | 多層感知機 |
| **學習程度** | machine learning | deep learning | deep learning |
| **套件** | sklearn | keras | sklearn |
| **調整空間** | 少 | 多 | 中 |
| **調整速度** | 快 | 慢 | 中 |
| **準確度** | 81.8% | 66.9% | 79.6% |

我們從機器學習最基礎之 KNN 模型開始進行訓練，接著以深度學習的 NN 模型開始調整，雖然 NN 模型可以調整的空間相當多但速度非常的慢，準確度的表現也不佳，因此我們接著改以 MLP 模型進行訓練，反覆修正與調整參數後得到 79.6%的準確率。

## 三、 模型分析

在訓練模型的過程中，我們分別在各個模型嘗試了多種不同的方法以提高預測準確率，並試圖針對前處理內容進行調整及尋找更好的訓練模型建構方式，以提升整個學習模型的準確度。進行資料篩選時，將每種 independent variable 分別對 dependent variable 進行相關分析，發現資料類別 Pregnancies 對於是否罹患糖尿病之結果影響最小，故將原資料 Pregnancies 欄位刪去後，觀察個模型之表現，而我們發現各模型的準確率皆有較好的表現，比較結果顯示如下表：

| | original data | processed data | improvement |
|---|---|---|---|
| NN | 0.646 | 0.669 | +2.3% |
| NN+ | 0.734 | 0.760 | +2.6% |
| kNN | 0.779 | 0.785 | +0.6% |
| MLP | 0.702 | 0.74 | +3.8% |

由表可知，無論是在哪一個模型，刪除 Pregnancies 欄位後皆能使準確率提升，因此我們定義 Pregnancies 在此筆資料中為擾亂因子，將其從資料欄位刪除後再進行後續之模型訓練，以下將分別就 4 個模型之訓練過程進行分析：

1、Neural Network(NN)

在訓練 NN 模型時，我們發現不管調整 optimizer、loss function 或是 test_size 的比例配置，所訓練出之準確度結果皆相差無幾，如下表所示：

| | | |
|---|---|---|
| without stratified | 0.678 | 0.619 |
| with straitified | 0.661 | 0.669 |
| init = uniform | 0.661 | 0.669 |
| init = normal | 0.661 | 0.669 |
| test size = 0.2 | 0.663 | 0.662 |
| test size = 0.25 | 0.661 | 0.669 |

在經由多種參數調整後，最後模型計算出的訓練以及測試準確率如下:

```
 accuracy = 0.669 sensitivity = 0.113 specificity = 0.958
================================================================================

 Mean
 accuracy = 0.669 sensitivity = 0.113 specificity = 0.958

 Stand Error
 accuracy =  0.0 sensitivity = 0.0 specificity = 0.0
```

```
train_loss, train_acc=classifier.evaluate(train_set, train_set_label)
test_loss, test_acc=classifier.evaluate(test_set, test_set_label)
print('\nTrain accuracy: %.3f' %train_acc)
print('Test accuracy: %.3f' %test_acc)
```

```
543/543 [==============================] - 1s 924us/step
181/181 [==============================] - 0s 43us/step

Train accuracy: 0.661
Test accuracy: 0.669
```

NN 模型訓練準確度=0.661，測試準確度=0.669，模型訓練效果一般，儘管在訓練後的準確度有提升，但幅度仍顯不足，故尋找另外的模型建構方式以優化訓練績效。

2、NN+

```
train_loss, train_acc= model.evaluate(x_train, y_train)
test_loss, test_acc= model.evaluate(x_test, y_test)
print('\nTrain accuracy: %.3f' %train_acc)
print('Test accuracy: %.3f' %test_acc)
```

```
576/576 [==============================] - 0s 32us/step
192/192 [==============================] - 0s 39us/step

Train accuracy: 0.932
Test accuracy: 0.760
```

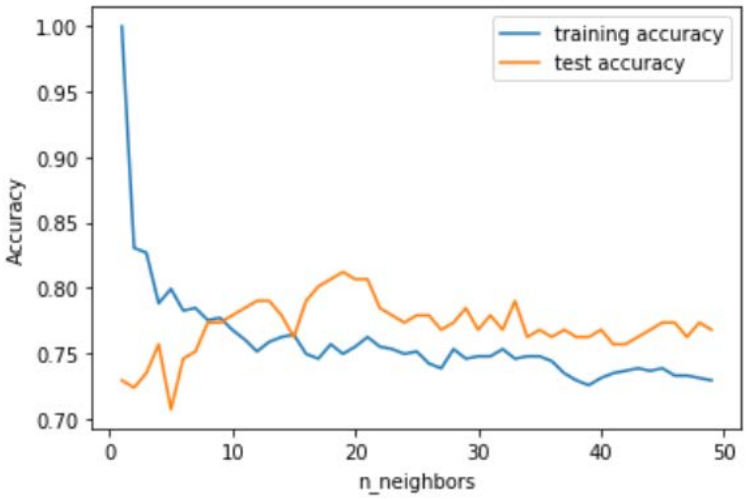
由 NN+的結果可得知其表現較 NN 佳，接著再進行調整模型內各項參

數，如 activation function 以及 optimizer、訓練測試比例或節點數量等，但觀察準確率之改動幅度都不大(變動率<1%)，故其他提升準確率方法將朝改變模型方向前進。

3、K Nearest Neighbor(K-NN)

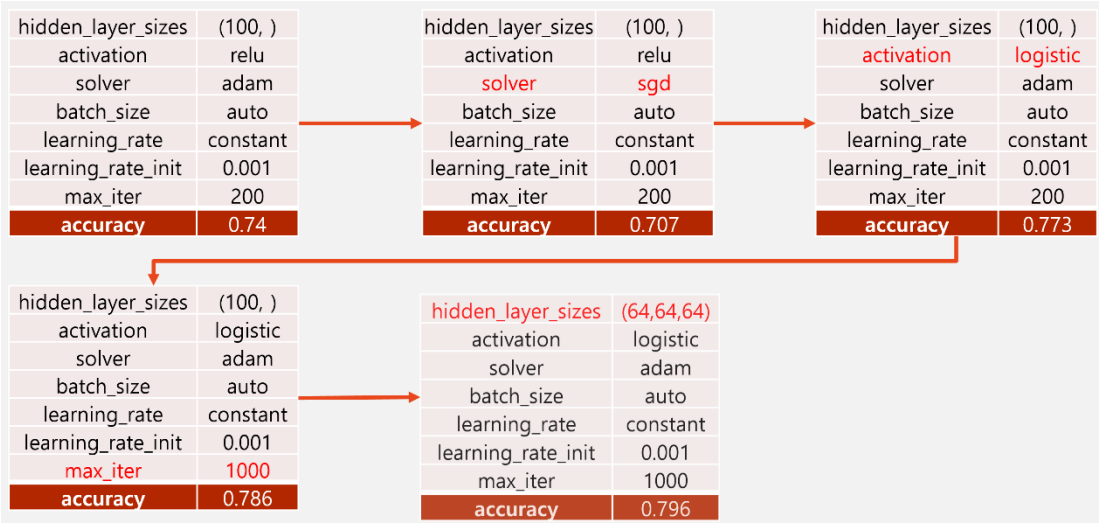
在 KNN 模型裡，我們以調整 n_neighbors 為主要的測試參數，利用迴圈測試 n_neighbors 數在 1-50 時的準確度，並輸出圖表以觀察其變化：

```
training_accuracy = []
test_accuracy = []

neighbors_settings = range(1, 50)
for n_neighbors in neighbors_settings:
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    training_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```




由表可看出，在 n_neighbors 為 19 時，測試資料的準確度最高，可達 0.818。

4、Multilayer perceptron(MLP)




MLP 模型我們以 sklearn 套件中分類器的各個參數進行修改，嘗試尋找不同參數的改變對於準確率的影響，並以最大化準確度為目標。

首先我們將 solver 從 adam 調整為 sgd，然而準確率從 0.74 降為 0.707，故我們將 solver 改回 adam 並調整 activation function 為 logistic，此改變讓準確率提升至 0.773，接著我們將 1000 設為最大 iteration 數，可得準確率為 0.786，並把隱藏層改為三層，使得準確率最終提升至 0.796。

從上述四個模型的訓練過程，我們分別在過程中根據不同模型調整了對應的參數以使準確率能提升，並不斷地思考是否能夠調整其他部分以優化模型，而在各模型的不同階段之準確率結果與進步幅度如下表所示：

| | Training process | Original data | Processed data | Optimal model | Improvement |
|---|---|---|---|---|---|
| **KNN** | n_neighbor<br>activation | 0.779 | 0.785 | 0.818 | **+3.9%** |
| **NN** | encoding<br>dropout | 0.646 | 0.669 | 0.75 | **+ 10.4%** |
| **MLP** | hidden_layer_sizes<br>activation<br>max_iter | 0.702 | 0.74 | 0.796 | **+9.4%** |

## 四、 結果與討論

### 1、結論

從最初之問題定義，我們發現罹患糖尿病的問題隨著現代人的生活習慣改變已不容忽視，然而並無一便利的方式提供民眾進行檢測，醫院的健檢報告又過於複雜，因此期望透過此預測模型來進行糖尿病的篩檢。

首先，我們採用 Kaggle 內的數據作為資料來源，一步步進行資料前處理以使模型的訓練能夠更加準確與順利，從釐清欄位、數據清理、特徵選取再到資料標準化，逐步刪去我們不需要以及異常數值，以使後續之模型訓練能夠更為準確。

接著，根據過往文獻以及此次資料來源的特徵，從機器學習到深度學習，決定由 NN、NN+、KNN、MLP 四種模型分別進行訓練，並針對不同的模型經由腦力激盪的過程來調整合適的參數以提升準確率，觀察每次訓練結果的變化後嘗試在當中找到最合適的訓練模型，由結果可得知各模型在訓練後均有顯著提升。

2、未來展望

此次模型僅針對糖尿病的篩檢進行訓練，若未來能夠將此模型推廣至其他民眾難以自行判斷的疾病類別，將可以使關心自身健康的使用者更準確地掌握自身身體狀況，醫療資源也能更有效率地被運用，及早發現疾病並準確治療，使國人於疾病防治的措施能夠更加完善。

此外、由於本次資料來源之種族與筆數有限，尚未能以大量資料進行模型準確度的訓練，亦尚未全面性地測試各族群的資料，若日後能與大型醫院進行合作，取得其資料庫內的資料作為訓練模型的來源，將能使分析結果更具可信度以及提升學習效率，同時，希望能夠架設網站或是 app 提供民眾更便利地服務，讓使用者能在自行輸入個人資料後就能透過預測模型判斷是否罹患該疾病，並再進一步至醫院進行治療與診斷，省去不必要的醫療資源浪費、隨時掌握自身身體狀況。