HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP 1

THỰC TẬP CƠ SỞ

Giảng viên hướng dẫn : **Trần Đình Quế**
Họ và tên sinh viên : **Nguyễn Đức Trường**
Mã sinh viên : **B22DCCN883**
Lớp : **D22CQCN07**

Hà Nội – 2025

I. Chạy case study

1. Loading the data

```
import numpy as np
import pandas as pd
df = pd.read_csv('C://DATA//diabetes.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

2. Cleaning data

- Kiểm tra các giá trị null

```
#---check for null values---
print("Nulls")
print("=====")
print(df.isnull().sum())
```

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

- Kiểm tra giá trị 0

```
#---check for 0s---
print("0s")
print("==")
print(df.eq(0).sum())
```

```
0s
==
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

- Thay đổi giá trị 0 bằng NaN

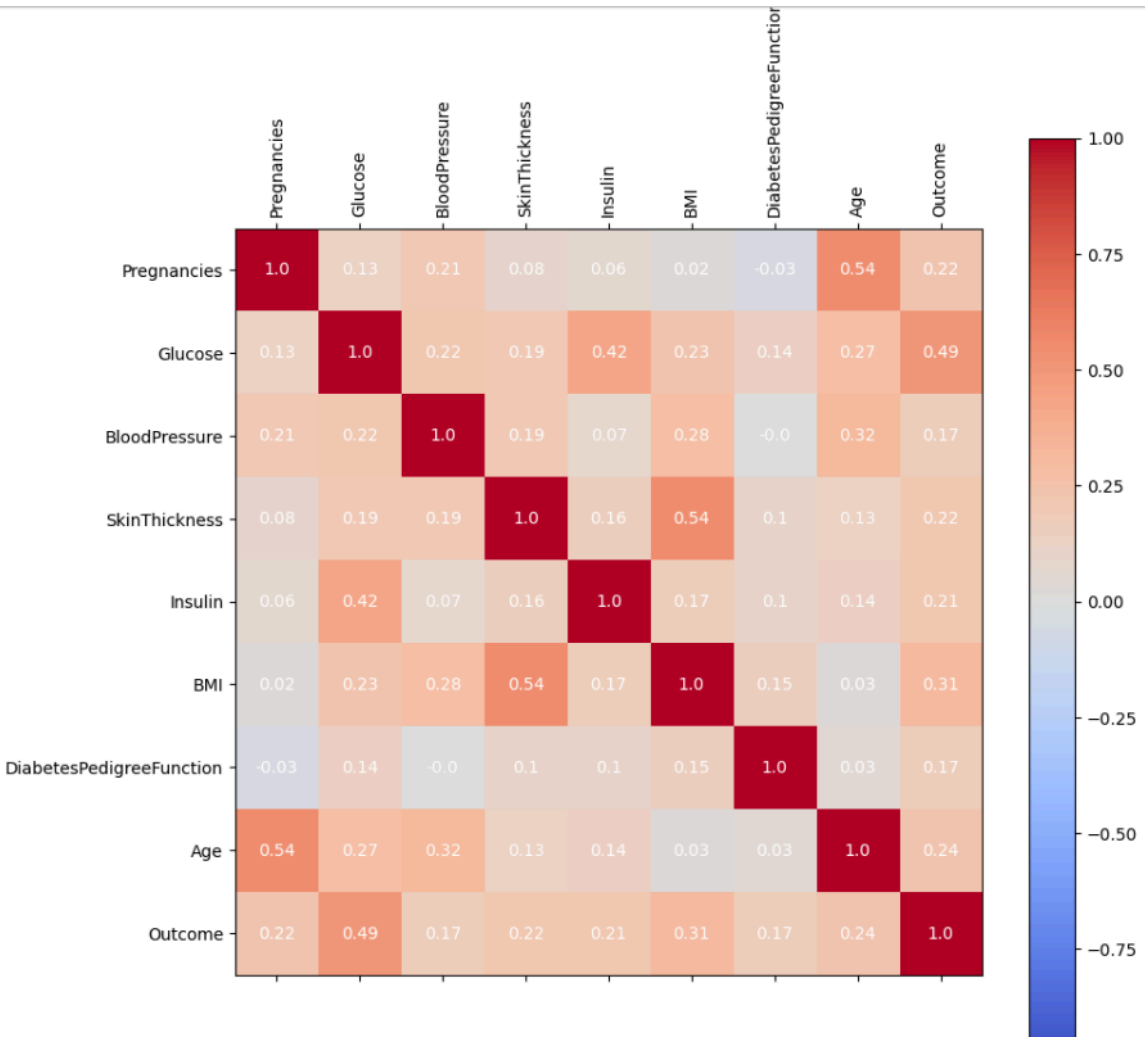
```
df[['Glucose','BloodPressure','SkinThickness',
 'Insulin','BMI','DiabetesPedigreeFunction','Age']] = \
 df[['Glucose','BloodPressure','SkinThickness',
 'Insulin','BMI','DiabetesPedigreeFunction','Age']].replace(0,np.nan)
```

3. Kiểm tra mối tương quan giữa các thuộc tính

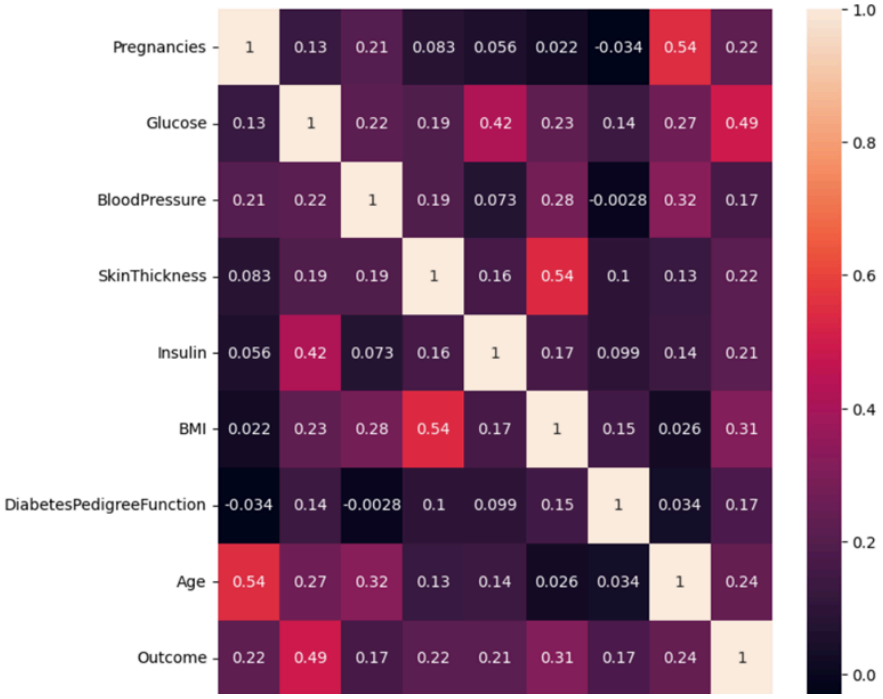
```
corr = df.corr()
print(corr)
```

```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.127911       0.208522       0.082989
Glucose                      0.127911  1.000000       0.218367       0.192991
BloodPressure                0.208522  0.218367       1.000000       0.192816
SkinThickness                0.082989  0.192991       0.192816       1.000000
Insulin                      0.056027  0.420157       0.072517       0.158139
BMI                          0.021565  0.230941       0.281268       0.542398
DiabetesPedigreeFunction    -0.033523  0.137060      -0.002763       0.100966
Age                          0.544341  0.266534       0.324595       0.127872
Outcome                      0.221898  0.492928       0.166074       0.215299

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               0.056027  0.021565                 -0.033523
Glucose                   0.420157  0.230941                  0.137060
BloodPressure             0.072517  0.281268                 -0.002763
SkinThickness             0.158139  0.542398                  0.100966
Insulin                   1.000000  0.166586                  0.098634
BMI                       0.166586  1.000000                  0.153400
DiabetesPedigreeFunction  0.098634  0.153400                  1.000000
Age                       0.136734  0.025519                  0.033561
Outcome                   0.214411  0.311924                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266534  0.492928
BloodPressure             0.324595  0.166074
SkinThickness             0.127872  0.215299
Insulin                   0.136734  0.214411
BMI                       0.025519  0.311924
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

4. Vẽ ma trận



- use Seaborn's heatmap():



5. Traning and Saving the Model

```
knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(X, y)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=19)
```

- Lưu vào ổ đĩa để sử dụng sau

```
import pickle
#---save the model to disk---
filename = 'diabetes.sav'
#---write to the file using write and binary mode---
pickle.dump(knn, open(filename, 'wb'))
```

- Load model

```
#---load the model from disk---
loaded_model = pickle.load(open(filename, 'rb'))
```

- Dự đoán

```
Glucose = 65
BMI = 70
Age = 50
prediction = loaded_model.predict([[Glucose, BMI, Age]])
print(prediction)
if (prediction[0]==0):
 print("Non-diabetic")
else:
 print("Diabetic")
```

```
[0]
Non-diabetic
```

+ Kết quả trả về: prediction:0 ứng với Non-diabetic

- Dự đoán xác xuất cao nhất(%):

```
proba = loaded_model.predict_proba([[Glucose, BMI, Age]])
print(proba)
print("Confidence: " + str(round(np.amax(proba[0]) * 100 ,2)) + "%")
```

```
[[0.94736842 0.05263158]]
Confidence: 94.74%
```

II. Cleaning data

1. Cleaning data là gì?

- Cleaning data là quá trình sửa chữa hoặc loại bỏ dữ liệu không chính xác, bị hỏng, định dạng sai, trùng lặp hoặc không đầy đủ trong một tập dữ liệu. Khi kết hợp nhiều nguồn dữ liệu, có nhiều khả năng dữ liệu sẽ bị trùng lặp hoặc gán nhãn sai.

2. Các vấn đề thường gặp trong dữ liệu

- Dữ liệu thô thường gặp phải một số vấn đề phổ biến như:
    - Dữ liệu thiếu (Missing Data): Một số giá trị bị thiếu trong tập dữ liệu.
    - Dữ liệu trùng lặp (Duplicate Data): Các bản ghi xuất hiện nhiều lần không cần thiết.
    - Dữ liệu không hợp lệ (Invalid Data): Dữ liệu có định dạng không đúng hoặc không hợp lý.
    - Dữ liệu không nhất quán (Inconsistent Data): Sự khác biệt trong cách nhập dữ liệu, ví dụ như định dạng ngày tháng khác nhau.
    - Dữ liệu chứa nhiễu (Noisy Data): Dữ liệu có các giá trị bất thường hoặc ngoại lai (outliers).
    - Dữ liệu sai chính tả (Typographical Errors): Lỗi nhập liệu do con người gây ra.

3. Các phương pháp làm sạch dữ liệu

3.1. Xử lý dữ liệu bị thiếu

- Loại bỏ hàng/cột chứa nhiều giá trị bị thiếu nếu chúng không ảnh hưởng lớn đến phân tích.
- Điền giá trị thiếu bằng cách:
    + Điền giá trị trung bình, trung vị hoặc mode.
    + Sử dụng mô hình dự đoán để ước lượng giá trị bị thiếu.
    + Nội suy dữ liệu (Interpolation) dựa trên xu hướng của tập dữ liệu.

3.2. Xử lý dữ liệu trùng lặp

- Sử dụng các phương pháp nhận diện trùng lặp dựa trên khóa chính hoặc các thuộc tính quan trọng.
- Loại bỏ các bản ghi trùng lặp nhưng vẫn đảm bảo giữ lại dữ liệu quan trọng

3.3. Chuẩn hóa và chuyển đổi dữ liệu

- Định dạng dữ liệu nhất quán (ví dụ: chuẩn hóa ngày tháng, số thập phân).

- Chuyển đổi kiểu dữ liệu phù hợp với yêu cầu phân tích.
- Loại bỏ ký tự không mong muốn trong các cột văn bản.

3.4. Xử lý dữ liệu ngoại lai (Outliers)

- Sử dụng các phương pháp thống kê như IQR (Interquartile Range), Z-score để phát hiện outliers.
- Loại bỏ hoặc điều chỉnh các giá trị ngoại lai dựa trên ảnh hưởng của chúng đến phân tích.

3.5. Xử lý dữ liệu không nhất quán

- Kiểm tra và đồng bộ hóa cách nhập liệu.
- Sử dụng các thuật toán gán nhãn để chuẩn hóa dữ liệu (ví dụ: chuẩn hóa tên sản phẩm, địa chỉ,...)

```
df[['Glucose','BloodPressure','SkinThickness',
 'Insulin','BMI','DiabetesPedigreeFunction','Age']] = \
 df[['Glucose','BloodPressure','SkinThickness',
 'Insulin','BMI','DiabetesPedigreeFunction','Age']].replace(0,np.nan)
```

VD: Thay các giá trị 0 thành Nan

```
df.fillna(df.mean(), inplace = True)
```

VD:Thay giá trị Nan bằng giá trị trung bình

4. Kết luận

- Cleaning data là một bước quan trọng nhằm đảm bảo tính chính xác và đáng tin cậy của dữ liệu trước khi phân tích. Việc áp dụng các phương pháp làm sạch phù hợp giúp cải thiện hiệu suất của các mô hình phân tích và hỗ trợ ra quyết định chính xác hơn. Sử dụng các công cụ và phương pháp tự động hóa có thể giúp tối ưu hóa quá trình làm sạch dữ liệu, tiết kiệm thời gian và nâng cao hiệu quả xử lý dữ liệu.

III. Logistic Regression

1. Tổng quan

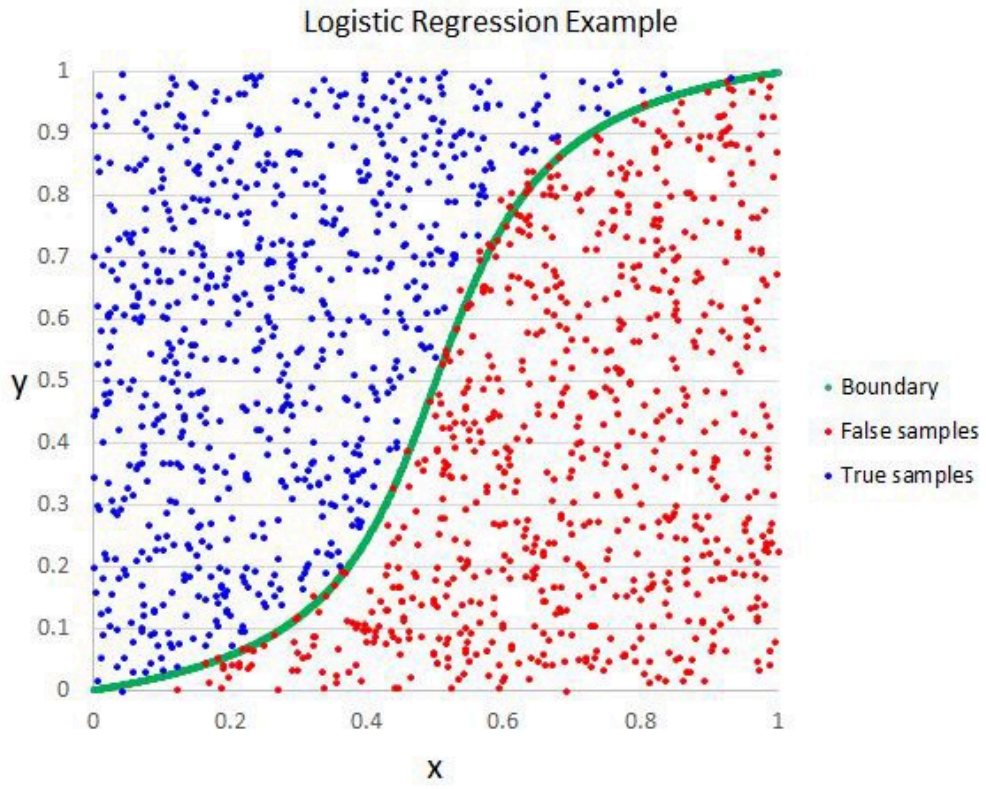
- Logistic Regression là một thuật toán **học có giám sát** (Supervised Learning), được sử dụng chủ yếu cho **bài toán phân loại** (Classification). Mặc dù có tên "Regression" (hồi quy), thuật toán này thực tế được dùng để phân loại nhị phân hoặc đa lớp.
- Logistic Regression dựa trên mô hình hồi quy tuyến tính nhưng sử dụng hàm **sigmoid** để biến đổi đầu ra thành xác suất thuộc một trong hai lớp.

2. Cơ sở lý thuyết

- Công thức tổng quát của mô hình Logistic Regression:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

- Trong đó:
  - t là tổ hợp tuyến tính của đầu vào:
  - σ là hàm sigmoid đưa giá trị về khoảng (0,1)



3. Ưu và nhược điểm

3.1. Ưu điểm

- **Đơn giản, dễ hiểu:** Logistic Regression là một trong những thuật toán học máy dễ triển khai nhất
- **Hiệu quả với dữ liệu tuyến tính:** Khi dữ liệu có thể phân tách tuyến tính, Logistic Regression hoạt động rất tốt.
- **Không yêu cầu nhiều tài nguyên:** Không cần quá nhiều bộ nhớ hay tính toán phức tạp.

- **Đưa ra xác suất dự đoán:** Có thể sử dụng ngưỡng khác nhau để điều chỉnh độ nhạy

3.2. Nhược điểm

- **Không hoạt động tốt với dữ liệu phi tuyến:** Nếu dữ liệu không thể phân tách tuyến tính, mô hình Logistic Regression hoạt động kém.
- **Dễ bị overfitting với dữ liệu có nhiều đặc trưng**
- **Nhạy cảm với dữ liệu mất cân bằng:** Nếu một lớp xuất hiện quá ít so với lớp còn lại, mô hình có thể bị lệch.

4. Ví dụ với bài toán dự đoán bệnh tiểu đường

Đối với thuật toán Logistic Regression ,thay vì chia bộ dữ liệu thành các bộ huấn luyện và thử nghiệm, chúng ta sẽ sử dụng xác thực chéo 10 lần để có được điểm trung bình của thuật toán được sử dụng

4.1. Code và giải thích(python)

```
from sklearn import linear_model
#---thêm thư viện linear_model chứa thuật toán Logistic Regression---
from sklearn.model_selection import cross_val_score
#---cross_val_scor để đánh giá mô hình
X = df[['Glucose','BMI','Age']]
#---X là tệp dữ liệu đầu vào, lấy 3 cột
y = df.iloc[:,8]
#---Y là label cần dự đoán. Biểu thị outcome (biểu thị có mắc bệnh tiểu đường k)
log_regress = linear_model.LogisticRegression()
#--- Tạo mô hình Logistic Regression--
log_regress_score = cross_val_score(log_regress, X, y, cv=10,
scoring='accuracy').mean()
#-- Đánh giá mô hình bằng Cross-validation
#-- cross_val_score() thực hiện 10-fold cross-validation (cv=10)
#--Mỗi lần, dữ liệu được chia thành 10 phần, 9 phần dùng để huấn luyện, 1 phần để kiểm tra. Quá trình này lặp lại 10 lần.
#--scoring='accuracy' đánh giá mô hình dựa trên độ chính xác.
#---.mean() lấy trung bình độ chính xác của 10 lần chạy.
print(log_regress_score)
```

4.2. Kết quả

```
0.7669856459330144
```

IV. K-Nearest Neighbors

1. Tổng quan

- Thuật toán K-Nearest Neighbors (KNN) là một trong những thuật toán học máy cơ bản nhất thuộc nhóm học có giám sát. Thuật toán này dùng để giải

quyết các bài toán phân loại và hồi quy dựa trên nguyên tắc các mẫu dữ liệu có đặc trưng tương đồng thường nằm gần nhau trong không gian dữ liệu.

2. Nguyên lý hoạt động

2.1. Các bước thực hiện

1. Xác định giá trị K (số lượng hàng xóm gần nhất cần xét).
2. Tính khoảng cách giữa mẫu dữ liệu cần dự đoán và tất cả các mẫu trong tập huấn luyện.
3. Tìm K mẫu gần nhất dựa trên khoảng cách đã tính.
4. Dự đoán nhãn của dữ liệu mới dựa trên đa số nhãn trong K hàng xóm gần nhất (nếu phân loại) hoặc tính giá trị trung bình (nếu hồi quy).

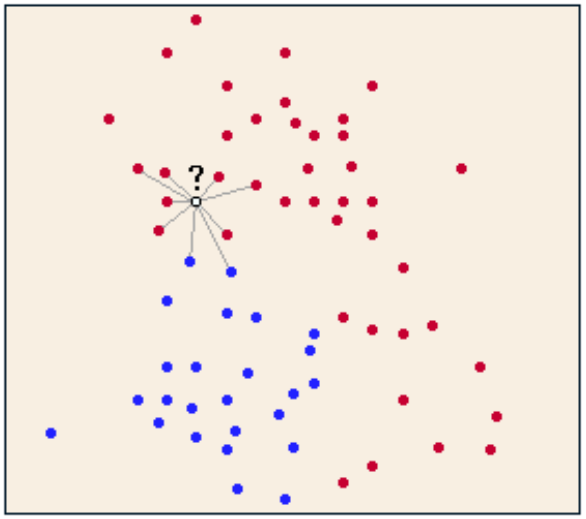
2.2. Các Phương Pháp Tính Khoảng Cách

- **Khoảng cách Euclidean (phổ biến nhất):**

$$\sqrt{\sum_{k=1}^{d}(x_{ik} - x_{jk})^2}$$

- **Khoảng cách Manhattan:**

$$\sum_{k=1}^{d} |x_{ik} - x_{jk}|$$



3. Ưu nhược điểm cảu KNN

3.1. Ưu điểm

- Dễ hiểu, dễ triển khai, không cần huấn luyện

- Hiệu quả tốt với dữ liệu nhỏ và không tuyến tính.
- Không yêu cầu giả định phân phối dữ liệu.

3.2. Nhược điểm

- Tốc độ chậm với tập dữ liệu lớn, do phải tính khoảng cách với tất cả các mẫu.
- Nhạy cảm với nhiễu và dữ liệu không cân bằng
- Chọn K thích hợp cần thử nghiệm nhiều lần.

4. Ví dụ với bài toán dự đoán bệnh tiểu đường

Thuật toán tiếp theo mà chúng ta sẽ sử dụng là K-Nearest Neighbors (KNN). Ngoài việc sử dụng xác thực chéo 10 lần để có được điểm trung bình của thuật toán, chúng ta cũng cần thử các giá trị khác nhau của k để có được k tối ưu để có thể có được độ chính xác tốt nhất:

4.1. Code và giải thích

Code dưới đây sử dụng thuật toán K-Nearest Neighbors (KNN) kết hợp với k-fold cross-validation để tìm giá trị K tối ưu (số lượng hàng xóm gần nhất) trong bài toán phân loại.

```
from sklearn.neighbors import KNeighborsClassifier
#---Thêm thư viện

cv_scores = []
#---Khởi tạo danh sách trống lưu điểm số cross-validation
folds = 10

ks = list(range(1,int(len(X) * ((folds - 1)/folds)), 2))
#---ks là danh sách các số lẻ (1, 3, 5, 7, ...) được dùng làm số lượng hàng xóm k cho KNN.
#--int(len(X) * ((folds - 1)/folds)): Giá trị lớn nhất của k được đặt là 90% kích thước tập dữ liệu


for k in ks:
 knn = KNeighborsClassifier(n_neighbors=k)
 score = cross_val_score(knn, X, y, cv=folds, scoring='accuracy').mean()
 cv_scores.append(score)
#---Tạo mô hình KNeighborsClassifier(n_neighbors=k)
#--Sử dụng cross_val_score để tính độ chính xác trung bình trên folds lần chạy
#--Lưu điểm số vào cv_scores

knn_score = max(cv_scores)

#--knn_score = max(cv_scores): Lấy độ chính xác cao nhất trong danh sách cv_scores.

optimal_k = ks[cv_scores.index(knn_score)]
#--optimal_k = ks[cv_scores.index(knn_score)]: Tìm giá trị k tương ứng với độ chính xác cao nhất.

print(f"The optimal number of neighbors is {optimal_k}")
print(knn_score)
result.append(knn_score)
```

4.2. Kết quả

```
The optimal number of neighbors is 19
0.7721462747778537
```

V. Support Vector Machines

1. Tổng quan

- Support Vector Machines (SVM) là một thuật toán học máy mạnh mẽ, chủ yếu được sử dụng trong bài toán phân loại và hồi quy. SVM đặc biệt hữu ích khi dữ liệu có không gian đặc trưng cao và số lượng mẫu nhỏ.

2. Nguyên lý hoạt động

2.1. Siêu phẳng phân tách

- SVM tìm kiếm một siêu phẳng (hyperplane) tối ưu để phân tách các điểm dữ liệu thuộc các lớp khác nhau. Trong không gian hai chiều, siêu phẳng là một đường thẳng, còn trong không gian ba chiều, nó là một mặt phẳng.

2.2. Khoảng cách và Support Vectors

- Khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất của mỗi lớp được gọi là margin. Các điểm dữ liệu gần siêu phẳng nhất được gọi là Support Vectors, và chúng quyết định vị trí của siêu phẳng.
- Mục tiêu của SVM là tìm siêu phẳng tối ưu, sao cho margin là lớn nhất.

3. Các biến thể của SVM

3.1. SVM tuyến tính

- Nếu dữ liệu có thể phân tách tuyến tính, SVM chỉ cần tìm một siêu phẳng tối ưu để phân tách dữ liệu.

3.2. SVM phi tuyến tính và Kernel Trick

- Khi dữ liệu không thể phân tách tuyến tính, SVM sử dụng hàm kernel để ánh xạ dữ liệu sang không gian có số chiều cao hơn, nơi dữ liệu có thể phân tách tuyến tính
- Một số hàm kernel phổ biến:
    - Linear Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$$

    - Polynomial Kernel:

$$k(\mathbf{x}, \mathbf{z}) = (r + \gamma \mathbf{x}^T \mathbf{z})^d$$

    - RBF Kernel (Gaussian Kernel):

$$k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma||\mathbf{x} - \mathbf{z}||_2^2), \quad \gamma > 0$$

4. Ưu và nhược điểm

4.1. Ưu điểm

- Hiệu quả cao với dữ liệu có số chiều lớn
- Tốt trong trường hợp số lượng mẫu nhỏ
- Có thể xử lý dữ liệu phi tuyến bằng kernel

4.2. Nhược điểm

- Nhạy cảm với lựa chọn kernel
- Tốn thời gian tính toán với tập dữ liệu lớn
- Không hoạt động tốt khi dữ liệu có quá nhiều nhiễu hoặc chồng chéo

5. Ví dụ với bài toán dự đoán bệnh tiểu đường

Chúng ta sẽ sử dụng hai loại kernel cho SVM: linear và RBF.

5.1. Linear kernel:

```
#--Thêm thư viện
from sklearn import svm

#--svm.SVC() là Support Vector Classification, một thuật toán phân loại dựa trên SVM.
#-kernel='linear' chỉ định sử dụng hàm kernel tuyến tính
linear_svm = svm.SVC(kernel='linear')

#--cross_val_score() là một phương pháp Cross-Validation (CV) để đánh giá mô hình bằng cách chia tập dữ liệu thành 10 phần (cv=10).
linear_svm_score = cross_val_score(linear_svm, X, y,cv=10, scoring='accuracy').mean()

print(linear_svm_score)
result.append(linear_svm_score)
```

```
0.7656527682843473
```

5.2. RBF kernel:

```
rbf = svm.SVC(kernel='rbf')
#--svm.SVC() tạo một mô hình Support Vector Classification (SVC)
#--kernel='rbf' chỉ định sử dụng Radial Basis Function (RBF) kernel, là một loại hàm kernel phi tuyến tính

rbf_score = cross_val_score(rbf, X, y, cv=10, scoring='accuracy').mean()
#--cross_val_score() thực hiện Cross-Validation (CV) 10 lần trên dữ liệu
#--scoring='accuracy' sử dụng độ chính xác (accuracy) để đánh giá mô hình.
print(rbf_score)
result.append(rbf_score)
```

```
0.765704032809296
```