HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP 1

THỰC TẬP CƠ SỞ

Giảng viên hướng dẫn: GV. Trần Đình Quế

Họ và tên sinh viên : Lưu Ngọc Anh
Mã sinh viên : B21DCCN148
Lớp : D21CNPM02

Hà Nội - 2025

Họ và tên: Lưu Ngọc Anh
Mã sinh viên : B21DCCN148

I. Kỹ thuật cleaning data

Làm sạch dữ liệu là bước quan trọng trong bất kỳ dự án Machine Learning nào. Các kỹ thuật quan trọng bao gồm:

✅ Xử lý giá trị bị thiếu.
✅ Loại bỏ hoặc điều chỉnh dữ liệu ngoại lai.
✅ Chuẩn hóa dữ liệu để cải thiện độ chính xác của mô hình.
✅ Chuyển đổi kiểu dữ liệu để đảm bảo tính nhất quán.
✅ Cân bằng dữ liệu để tránh mô hình bị thiên lệch.

Sau khi làm sạch dữ liệu, mô hình có thể hoạt động chính xác và hiệu quả hơn!

1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

Dữ liệu bị thiếu là một vấn đề phổ biến trong mọi tập dữ liệu. Nếu không xử lý đúng cách, nó có thể ảnh hưởng đến độ chính xác của mô hình.

- Kiểm tra dữ liệu bị thiếu

Có thể kiểm tra dữ liệu bị thiếu trong Pandas bằng cách:

```
import pandas as pd

df = pd.read_csv("dataset.csv")
print(df.isnull().sum())  # Đếm số lượng giá trị bị thiếu trong mỗi cột
```

- Cách xử lý dữ liệu bị thiếu
    - Loại bỏ dữ liệu bị thiếu: Nếu số lượng dữ liệu bị thiếu nhỏ, bạn có thể xóa các dòng hoặc cột đó:

```
df.dropna(inplace=True)  # Xóa tất cả dòng có giá trị bị thiếu
df.drop(columns=['column_name'], inplace=True)  # Xóa cột nếu nó có quá nhiều giá trị bị thiế
```

    - Điền giá trị thay thế (Imputation): Điền bằng giá trị trung bình, trung vị hoặc mode:

```
df['Glucose'].fillna(df['Glucose'].mean(), inplace=True)  # Điền bằng trung bình
df['BMI'].fillna(df['BMI'].median(), inplace=True)  # Điền bằng trung vị
df['Age'].fillna(df['Age'].mode()[0], inplace=True)  # Điền bằng mode
```

- Dự đoán giá trị bị thiếu bằng mô hình Machine Learning:

```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=3)
df[['Glucose', 'BMI', 'Age']] = imputer.fit_transform(df[['Glucose', 'BMI', 'Age']])
```

- Khi nào nên dùng phương pháp nào?
  ✔ Nếu dữ liệu bị thiếu ít → có thể xóa dòng/cột.
  ✔ Nếu dữ liệu bị thiếu nhiều → nên điền bằng giá trị trung bình, trung vị hoặc mode.
  ✔ Nếu dữ liệu phức tạp → có thể dùng mô hình ML để dự đoán giá trị bị thiếu.

2. Xử lý dữ liệu ngoại lai (Handling Outliers)

Ngoại lai là các giá trị cực đoan không phù hợp với xu hướng chung của dữ liệu. Nếu không xử lý, chúng có thể làm sai lệch mô hình.

2.1. Kiểm tra ngoại lai

Có nhiều phương pháp phát hiện ngoại lai:

- Sử dụng Boxplot

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=df['Glucose'])
plt.show()
```

Nếu có điểm quá xa so với phần còn lại → có thể là ngoại lai.

- Sử dụng Z-score

```
from scipy import stats

z_scores = stats.zscore(df['Glucose'])
df = df[(z_scores < 3)]  # Loại bỏ điểm có Z-score > 3
```

- Sử dụng IQR (Interquartile Range)

```
Q1 = df['Glucose'].quantile(0.25)
Q3 = df['Glucose'].quantile(0.75)
IQR = Q3 - Q1
df = df[(df['Glucose'] >= (Q1 - 1.5 * IQR)) & (df['Glucose'] <= (Q3 + 1.5 * IQR))]
```

2.2. Cách xử lý ngoại lai

- Loại bỏ giá trị ngoại lai (nếu số lượng nhỏ và không quan trọng).
- Thay thế bằng trung bình hoặc trung vị (nếu dữ liệu bị méo).
- Dùng log transformation để giảm ảnh hưởng của ngoại lai:

```
df['Glucose'] = np.log1p(df['Glucose'])
```

3. Chuẩn hóa dữ liệu (Data Normalization & Scaling)

Trong chương 12, thuật toán KNN và SVM đã được sử dụng. Các thuật toán này nhạy cảm với sự chênh lệch giá trị giữa các cột, nên cần chuẩn hóa dữ liệu.

- Min-Max Scaling (đưa dữ liệu về khoảng [0,1])

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df[['Glucose', 'BMI', 'Age']] = scaler.fit_transform(df[['Glucose', 'BMI', 'Age']])
```

- Standardization (Z-score scaling)

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['Glucose', 'BMI', 'Age']] = scaler.fit_transform(df[['Glucose', 'BMI', 'Age']])
```

- Khi nào dùng gì?
  ✔ Min-Max Scaling → nếu dữ liệu không có phân phối chuẩn.
  ✔ Standardization → nếu dữ liệu có phân phối chuẩn.

4. Chuyển đổi kiểu dữ liệu (Data Type Conversion)

Một số cột có thể bị sai kiểu dữ liệu, gây lỗi khi huấn luyện mô hình.

- Kiểm tra kiểu dữ liệu

```
print(df.dtypes)
```

- Chuyển đổi kiểu dữ liệu
- Chuyển object thành int/float:

```
df['Age'] = df['Age'].astype(int)
```

- Chuyển categorical thành one-hot encoding:

```
df = pd.get_dummies(df, columns=['Gender'], drop_first=True)
```

- Chuyển đổi dữ liệu ngày tháng:

```
df['Date'] = pd.to_datetime(df['Date'])
```

5. Cân bằng dữ liệu (Handling Imbalanced Data)

Nếu dữ liệu có sự chênh lệch lớn giữa các nhãn (Ví dụ: quá nhiều mẫu không bị tiểu đường so với mẫu bị tiểu đường), mô hình có thể bị thiên lệch.

- Over-sampling (Tạo thêm dữ liệu thiểu số bằng SMOTE)

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(sampling_strategy=0.5)  # Tạo thêm mẫu cho lớp thiểu số
X_resampled, y_resampled = sm.fit_resample(X, y)
```

- Undersampling (Giảm số lượng mẫu đa số)

```python
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(sampling_strategy=0.5)
X_resampled, y_resampled = rus.fit_resample(X, y)
```

- Cost-sensitive learning (Tăng trọng số lớp thiểu số)

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(class_weight="balanced")  # Tăng trọng số lớp thiểu số
model.fit(X, y)
```

II. Các kỹ thuật Machine Learning

Các kỹ thuật Machine learning được sử dụng trong chương 12 bao gồm:

- Hồi quy Logistic (*Logistic Regression*)
- K-láng giềng gần nhất (*K-Nearest Neighbors - KNN*)
- Máy vector hỗ trợ (SVM) với kernel tuyến tính và kernel RBF (*Support Vector Machines - Linear and RBF Kernels*)

1. Hồi quy Logistic (Logistic Regression)

1.1. Mô tả kỹ thuật

Hồi quy Logistic là một thuật toán phân loại nhị phân, sử dụng hàm sigmoid để đưa ra xác suất dự đoán một đối tượng thuộc về một trong hai lớp.

Công thức của hàm sigmoid:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)}}$$

1.2. Cách triển khai trong Python

```python
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

1.3. Ưu và nhược điểm

✅ **Ưu điểm:**

- Đơn giản, dễ hiểu và dễ triển khai.
- Hiệu quả khi dữ liệu có mối quan hệ tuyến tính.

❌ **Nhược điểm:**

- Không hoạt động tốt nếu dữ liệu không tuyến tính.
- Dễ bị ảnh hưởng bởi ngoại lai.

2. K-Nearest Neighbors (KNN)

2.1. Mô tả kỹ thuật

KNN là một thuật toán phân loại dựa trên khoảng cách giữa một điểm mới và các điểm trong tập huấn luyện.

Công thức tính khoảng cách Euclidean:

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

2.2. Cách triển khai trong Python

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

2.3. Ưu và nhược điểm

✅ **Ưu điểm:**

- Dễ hiểu và không cần huấn luyện mô hình phức tạp.
- Hoạt động tốt với dữ liệu có biên dạng không tuyến tính.

❌ **Nhược điểm:**

- Tính toán chậm khi dữ liệu lớn.
- Cần chọn giá trị **k** phù hợp.

3. Hỗ trợ Vector Machine (SVM - Support Vector Machine)

3.1. Mô tả kỹ thuật

SVM là thuật toán phân loại tìm một siêu phẳng (hyperplane) để tách biệt hai nhóm dữ liệu tốt nhất.

Hàm tối ưu hóa:

$$\max \frac{2}{||w||} \quad \text{với điều kiện} \quad y_i(wX_i + b) \geq 1$$

3.2. Cách triển khai trong Python

```
from sklearn.svm import SVC

model = SVC(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

3.3. Ưu và nhược điểm

✅ Ưu điểm:

- Tốt cho dữ liệu có biên tách biệt rõ ràng.
- Hoạt động tốt với dữ liệu có số chiều cao.

❌ Nhược điểm:

- Chậm khi dữ liệu lớn.
- Nhạy cảm với nhiễu.

III. Case study chap 12

Cấu trúc project

1. Tải dữ liệu

Bộ dữ liệu đã được tải xuống cục bộ và được đặt tên là diabetes.csv. Đoạn mã sau đây sẽ tải bộ dữ liệu và in ra thông tin về DataFrame bằng cách sử dụng hàm info():

```
import numpy as np
import pandas as pd
df = pd.read_csv('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. Làm sạch dữ liệu

- Kiểm tra các giá trị null trong bộ dữ liệu

```
#---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
```

- Không có giá trị null trong bộ dữ liệu. Tiếp theo, hãy kiểm tra các giá trị bằng 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
```

- Đối với các đặc trưng như **Pregnancies** và **Outcome**, giá trị 0 là bình thường. Tuy nhiên, đối với các đặc trưng khác, giá trị 0 có thể cho thấy dữ liệu chưa được thu thập đầy đủ trong bộ dữ liệu. Có nhiều cách để xử lý các giá trị 0 này, nhưng để đơn giản, chúng ta sẽ thay thế các giá trị 0 bằng NaN (Not a Number):

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

- Sau khi các giá trị NaN đã thay thế các giá trị 0 trong DataFrame, bạn có thể thay thế chúng bằng giá trị trung bình của từng cột như sau:

```
df.fillna(df.mean(), inplace=True)  # replace NaN with the mean
```

- Bây giờ, bạn có thể kiểm tra lại DataFrame để đảm bảo không còn giá trị 0:

```
print(df.eq(0).sum())
```

```
Pregnancies                 111
Glucose                       0
BloodPressure                 0
SkinThickness                 0
Insulin                       0
BMI                           0
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

3. Kiểm tra mối tương quan giữa các đặc trưng

Bước tiếp theo là xem xét cách các đặc trưng độc lập ảnh hưởng đến kết quả (bệnh nhân có mắc tiểu đường hay không). Để làm điều đó, bạn có thể gọi hàm corr() trên DataFrame:

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

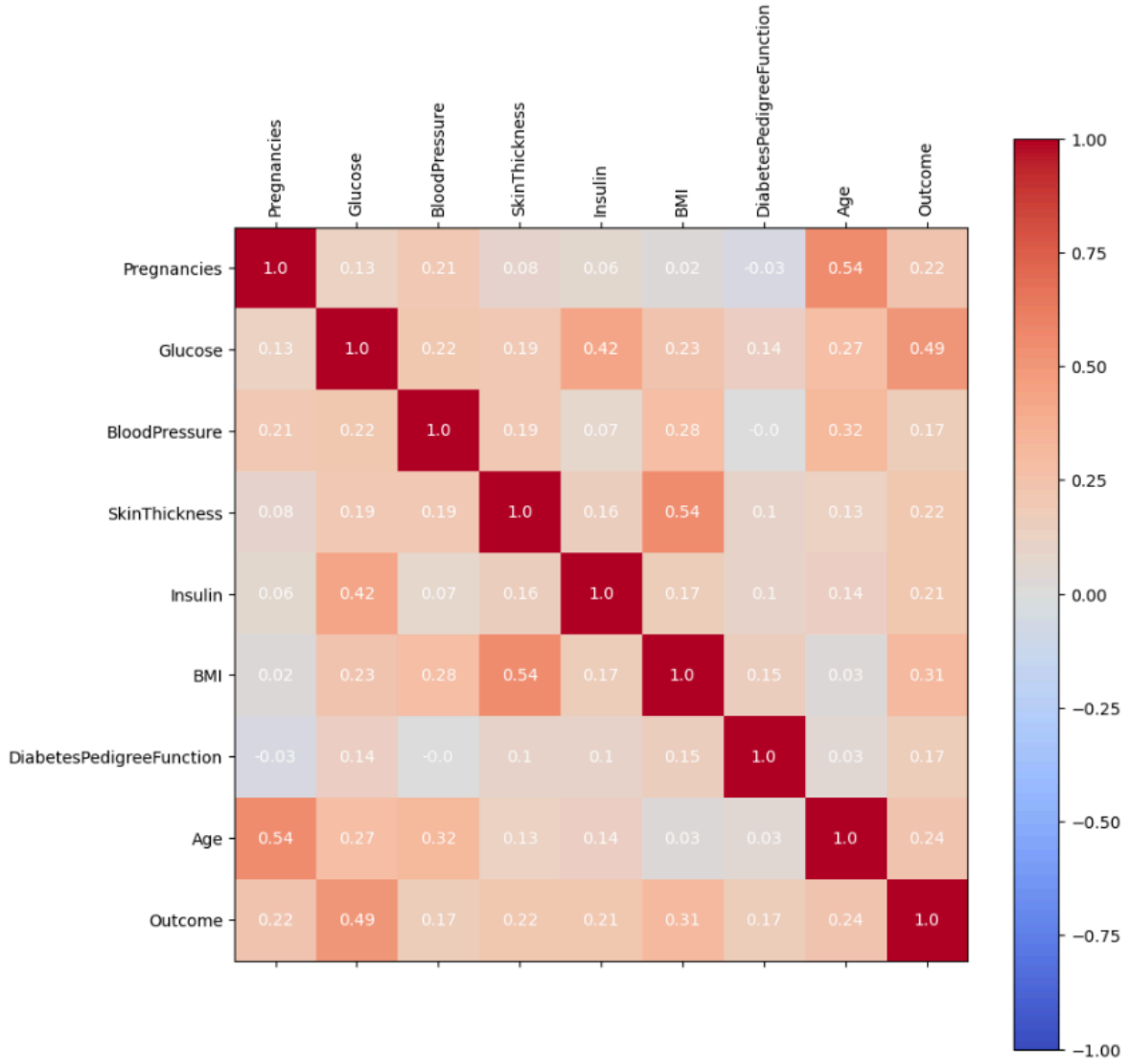
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.127911       0.208522       0.082989
Glucose                      0.127911  1.000000       0.218367       0.192991
BloodPressure                0.208522  0.218367       1.000000       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
```

Hàm corr() tính toán hệ số tương quan giữa các cột trong DataFrame. Ví dụ, đầu ra phía trên cho thấy mức glucose của bệnh nhân sau 2 giờ làm xét nghiệm dung nạp glucose đường uống có mối tương quan rất thấp với số lần mang thai (0.127911), nhưng lại có mối quan hệ đáng kể với kết quả (0.492928)

4. Trực quan hóa mối tương quan giữa các đặc trưng

Thay vì xem xét từng con số trong ma trận tương quan, chúng ta có thể hình dung dữ liệu trực quan để dễ dàng nhận thấy mối quan hệ giữa các đặc trưng. Đoạn mã sau sử dụng hàm matshow() để vẽ ma trận tương quan dựa trên kết quả của hàm corr(), đồng thời hiển thị các hệ số tương quan trong ma trận:

```
%matplotlib inline
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 10))
cax = ax.matshow(corr, cmap='coolwarm', vmin=-1, vmax=1)  # Hiển thị ma trận tương quan với màu sắc
fig.colorbar(cax)  # Thêm thanh màu để hiển thị giá trị tương quan

ticks = np.arange(0, len(df.columns), 1)  # Tạo danh sách chỉ mục cho trục x và y
ax.set_xticks(ticks)
ax.set_xticklabels(df.columns)  # Gán nhãn cho trục x
plt.xticks(rotation=90)  # Xoay nhãn trục x 90 độ để dễ đọc
ax.set_yticks(ticks)
ax.set_yticklabels(df.columns)  # Gán nhãn cho trục y

#--- Hiển thị hệ số tương quan trên từng ô của ma trận ---
for i in range(df.shape[1]):
    for j in range(9):
        text = ax.text(j, i, round(corr.iloc[i, j], 2), ha="center", va="center", color="w")

plt.show()
```



Trên biểu đồ heatmap trên, các ô có màu gần đỏ biểu thị các yếu tố có hệ số tương quan cao nhất, trong khi các ô có màu gần xanh dương thể hiện mối tương quan thấp nhất.

5. Đánh giá các thuật toán

```
#---Lựa chọn các đặc trưng---
X = df[['Glucose', 'BMI', 'Age']]
#---Nhãn---
y = df.iloc[:, 8]
```

a. Hồi quy Logistic (Logistic Regression)

Thuật toán đầu tiên mà chúng ta sử dụng là hồi quy logistic. Thay vì chia tập dữ liệu thành tập huấn luyện và tập kiểm tra, chúng ta sẽ sử dụng 10-fold cross-validation để tính điểm trung bình của thuật toán

```
from sklearn import linear_model
from sklearn.model_selection import cross_val_score

log_regress = linear_model.LogisticRegression()
log_regress_score = cross_val_score(log_regress, X, y, cv=10, scoring='accuracy').mean()
print(log_regress_score)
```

```
0.7669856459330144
```

Chúng ta cũng sẽ lưu lại kết quả này vào danh sách để so sánh với các thuật toán khác:

```
result = []
result.append(log_regress_score)
```

b. K-Nearest Neighbors (KNN)

Thuật toán tiếp theo mà chúng ta thử nghiệm là K-Nearest Neighbors (KNN). Ngoài việc sử dụng 10-fold cross-validation để tính điểm trung bình, chúng ta cũng cần thử các giá trị khác nhau của k để tìm ra k tối ưu nhằm đạt được độ chính xác cao nhất.

```
from sklearn.neighbors import KNeighborsClassifier

#---Danh sách trống để lưu điểm cross-validation---
cv_scores = []
#---Số lượng folds---
folds = 10
#---Tạo danh sách số lẻ cho k---
ks = list(range(1, int(len(X) * ((folds - 1) / folds)), 2))

#---Thực hiện k-fold cross-validation---
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)

#---Lấy điểm cao nhất---
knn_score = max(cv_scores)
#---Tìm k tối ưu---
optimal_k = ks[cv_scores.index(knn_score)]
print(f"The optimal number of neighbors is {optimal_k}")
print(knn_score)

result.append(knn_score)
```

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

c. Support Vector Machines (SVM)

Thuật toán tiếp theo mà chúng ta sử dụng là Support Vector Machine (SVM). Chúng ta sẽ thử nghiệm hai loại kernel:

1. Kernel tuyến tính (Linear Kernel)

```
from sklearn import svm
linear_svm = svm.SVC(kernel='linear')
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
```

2. Kernel RBF (Radial Basis Function)

```
rbf = svm.SVC(kernel='rbf')
rbf_score = cross_val_score(rbf, X, y, cv=10, scoring='accuracy').mean()
print(rbf_score)
result.append(rbf_score)
```

```
0.765704032809296
```

6. Lựa chọn thuật toán có hiệu suất tối ưu nhất

Sau khi đánh giá bốn thuật toán khác nhau, chúng ta có thể chọn thuật toán có hiệu suất tốt nhất:

```
algorithms = ["Logistic Regression", "K Nearest Neighbors", "SVM Linear Kernel", "SVM RBF Kernel"]
cv_mean = pd.DataFrame(result,index = algorithms)
cv_mean.columns=["Accuracy"]
cv_mean.sort_values(by="Accuracy",ascending=False)
```

	Accuracy
K Nearest Neighbors	0.772146
Logistic Regression	0.766986
SVM RBF Kernel	0.765704
SVM Linear Kernel	0.765653

7. Huấn luyện và lưu trữ mô hình

Vì KNN (k=19) là thuật toán có hiệu suất cao nhất, chúng ta sẽ huấn luyện mô hình bằng toàn bộ tập dữ liệu:

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

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

Sau khi mô hình được huấn luyện, chúng ta cần lưu nó vào ổ đĩa để có thể sử dụng lại trong tương lai cho việc dự đoán:

```
import pickle

#---Lưu mô hình vào file---
filename = 'diabetes.sav'
pickle.dump(knn, open(filename, 'wb'))
```

Mô hình đã được lưu vào file diabetes.sav. Giờ hãy thử tải mô hình lên để đảm bảo rằng nó đã được lưu trữ đúng cách:

```
#---Tải mô hình từ file---
loaded_model = pickle.load(open(filename, 'rb'))
```

8. Dự đoán với mô hình đã tải

Nếu kết quả dự đoán là 0, chương trình sẽ in ra "Non-Diabetic", ngược lại, nếu kết quả là 1, chương trình sẽ in ra "Diabetic".

```
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
```

9. Lấy xác suất dự đoán

Chúng ta cũng có thể xem mức độ tự tin của mô hình với kết quả dự đoán bằng cách lấy xác suất của từng lớp và chuyển nó thành phần trăm:

```
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%
```

10. Tạo REST_API

- Tạo mã nguồn REST_API.py:
    - Tạo một route API /diabetes/v1/predict bằng Flask route decorator.
    - Route này chỉ có thể truy cập bằng phương thức POST.
    - Để dự đoán, người dùng gửi yêu cầu POST đến route này kèm theo dữ liệu đầu vào dưới dạng JSON.
    - Kết quả dự đoán sẽ được trả về dưới dạng JSON.

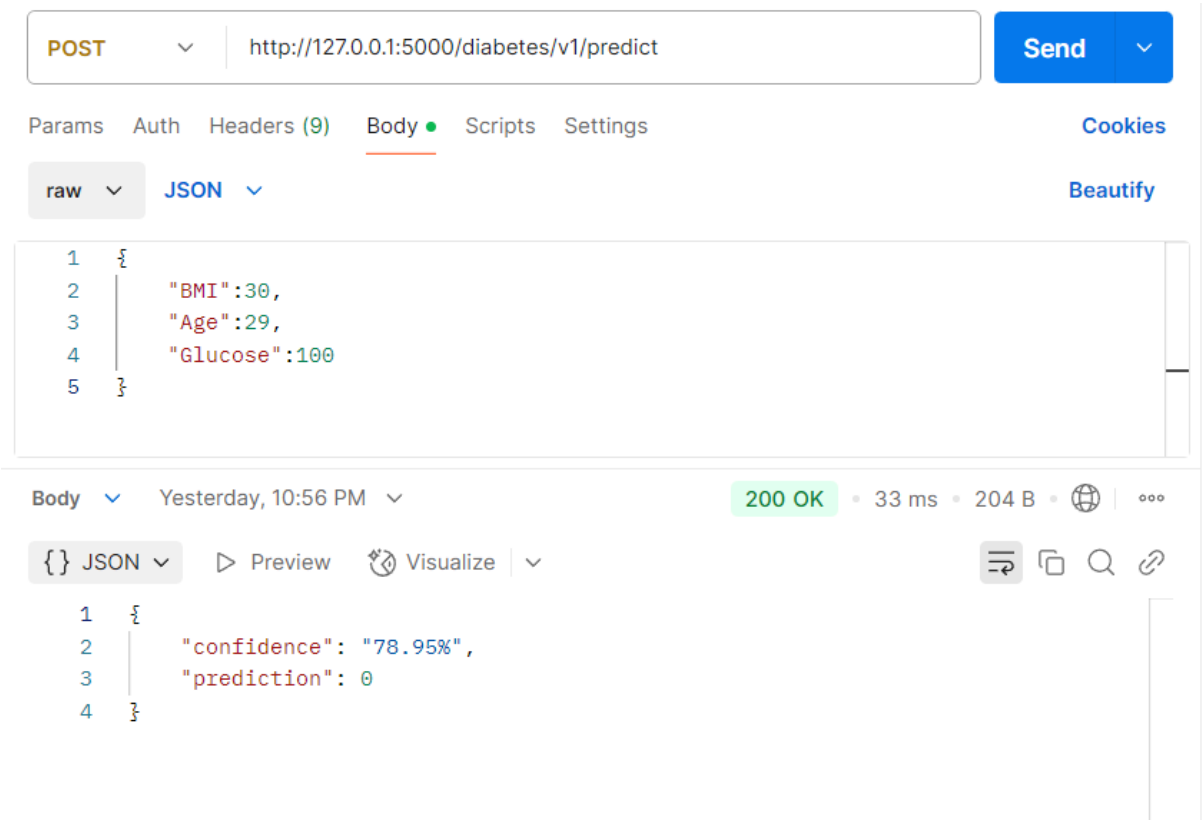
```python
import pickle
from flask import Flask, request, jsonify
import numpy as np

app = Flask(__name__)

#---Tên tệp chứa mô hình đã lưu---
filename = 'diabetes.sav'

#---Tải mô hình đã lưu---
loaded_model = pickle.load(open(filename, 'rb'))

@app.route('/diabetes/v1/predict', methods=['POST'])
def predict():
    #---Lấy dữ liệu đầu vào từ request---
    features = request.json

    #---Tạo danh sách các đặc trưng để dự đoán---
    features_list = [features["Glucose"],
                     features["BMI"],
                     features["Age"]]

    #---Dự đoán kết quả---
    prediction = loaded_model.predict([features_list])

    #---Lấy xác suất dự đoán---
    confidence = loaded_model.predict_proba([features_list])

    #---Tạo phản hồi gửi về client---
    response = {
        'prediction': int(prediction[0]),
        'confidence': str(round(np.amax(confidence[0]) * 100, 2)) + "%"
    }

    return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.2.16:5000
Press CTRL+C to quit
```

- Test API bằng postman
- Thiết lập Content-Type: application/json trong Header
- Chọn phương thức POST
- Nhập url: http://127.0.0.1:5000/diabetes/v1/predict
- Nhập body:

```
{
    "BMI":30,
    "Age":29,
    "Glucose":100
}
```

- Click Send



11. Tạo ứng dụng khách hàng để sử dụng mô hình

Sau khi REST API đã chạy thành công và được kiểm tra hoạt động đún cầnhể xây dựng phần **client** (ứng dụng khách) để sử dụng mô hìn Có thể xây dựng 1 ứng dụng Python đơn giản như sau:

- **Tạo chuỗi JSON** để gửi đến API.
- **Nhận kết quả dưới dạng JSON** từ API.
- **Trích xuất thông tin dự đoán** từ kết quả nhận được.h.

```
import json
import requests

def predict_diabetes(BMI, Age, Glucose):
    url = 'http://127.0.0.1:5000/diabetes/v1/predict'
    data = {"BMI": BMI, "Age": Age, "Glucose": Glucose}
    data_json = json.dumps(data)
    headers = {'Content-type': 'application/json'}
    response = requests.post(url, data=data_json, headers=headers)
    result = json.loads(response.text)
    return result

if __name__ == "__main__":
    BMI = input('BMI? ')
    Age = input('Age? ')
    Glucose = input('Glucose? ')

    predictions = predict_diabetes(float(BMI), float(Age), float(Glucose))

    print("Diabetic" if predictions["prediction"] == 1 else "Not Diabetic")
    print("Confidence: " + predictions["confidence"] + "%")
```

```
BMI?  55
Age?  29
Glucose?  120
Not Diabetic
Confidence: 52.63%%
```