Hoàng Sơn Lâm

B22DCCN477

BÁO CÁO BÀI TẬP TTCS

1. Kỹ thuật cleaning data

- Kiểm tra các giá trị null trong tập dữ liệu bằng hàm **isnull()** và hàm **sum()** Trong tập dữ liệu các giá trị null được coi là không hợp lệ nên ta cần thống kê số lượng 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
```

check for 0s

- Kiểm tra các giá trị bằng 0 trong tập dữ liệu bằng hàm **eq(0)** và **sum()**
  Theo kiểm tra phía trên tuy không có giá trị null nhưng những giá trị 0 đối với một số chỉ số ở người cũng được coi là không hợp lệ nên ta cũng cần kiểm tra các 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
```

replace the 0 values with NaN

- Thay thế các giá trị 0 trong bảng dữ liệu:
  Phương thức replace(0, np.NaN) thay thế tất cả giá trị 0 trong các cột này bằng giá trị NaN (Not a Number)

  Đây là một bước tiền xử lý quan trọng vì trong dữ liệu y tế, giá trị 0 thường không hợp lệ (ví dụ :đường huyết không thể bằng 0 ở người sống, BMI cũng không thể bằng 0 , chỉ trừ số lần mang thai và kết quả là có thể bằng 0)

Sau khi chuyển giá trị 0 thành NaN, dòng này điền các giá trị NaN bằng giá trị trung bình của mỗi cột.
Tham số inplace=True cho biết thay đổi được áp dụng trực tiếp vào dataframe df mà không cần gán lại.

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

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

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
```

2. Các kỹ thuật Machine Learning

- Thuật toán Logistic Regression

Logistic Regression là một thuật toán học có giám sát (Supervised Learning) được sử dụng để phân loại nhị phân (Binary Classification). Thuật toán này dựa trên mô hình hồi quy tuyến tính, nhưng thay vì dự đoán một giá trị liên tục, nó sử dụng hàm **Sigmoid** để đưa đầu ra về khoảng [0,1][0, 1][0,1], phù hợp cho các bài toán phân loại.

1. Import thư viện

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

linear_model: Thư viện chứa các mô hình hồi
cross_val_score: Hàm dùng để đánh giá hiệu suất mô hình bằng phương pháp cross-validation

2. Chuẩn bị dữ liệu

```
X = df[['Glucose','BMI','Age']]
y = df.iloc[:,8]
```

X: Biến đặc trưng (features) được chọn từ DataFrame df, bao gồm 3 cột: Glucose, BMI, và Age.
y: Biến mục tiêu (label) được lấy từ cột thứ 9 (index 8) của DataFrame df. Đây là cột chứa kết quả phân loại (1 là mắc tiểu đường và 0 là không mắc).

3. Khởi tạo mô hình

```
log_regress = linear_model.LogisticRegression()
```

Tạo một đối tượng mô hình hồi quy logistic (**LogisticRegression**) để phân loại dữ liệu.

4. Đánh giá mô hình bằng Cross-Validation

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

cross_val_score: Thực hiện cross-validation với các tham số:

- **log_regress**: Mô hình cần đánh giá.
- **X, y**: Dữ liệu đặc trưng và nhãn.
- **cv=10**: Chia dữ liệu thành 10 phần (10-fold cross-validation), huấn luyện 10 lần, mỗi lần dùng 9 phần để train và 1 phần để test.
- **scoring='accuracy'**: Đánh giá bằng độ chính xác (tỉ lệ dự đoán đúng).

mean(): Tính giá trị trung bình của độ chính xác từ 10 lần kiểm tra.

5. In kết quả và lưu vào danh sách

```
print(log_regress_score)

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

Mục đích: Đoạn code này đánh giá hiệu suất của mô hình hồi quy logistic trong việc dự đoán biến mục tiêu phát hiện bệnh tiểu đường dựa trên 3 đặc trưng: Glucose, BMI, và Age.
Phương pháp: Sử dụng **10-fold cross-validation** để đảm bảo kết quả đáng tin cậy và tránh overfitting.
Kết quả: Độ chính xác trung bình (accuracy) của mô hình được in ra và lưu vào danh sách result

```
from sklearn import linear_model
from sklearn.model_selection import cross_val_score
#---features---
X = df[['Glucose','BMI','Age']]
#---label---
y = df.iloc[:,8]
log_regress = linear_model.LogisticRegression()
log_regress_score = cross_val_score(log_regress, X, y, cv=10,
scoring='accuracy').mean()
print(log_regress_score)

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

```
0.7669856459330144
```

Logistic Regression đạt **76.17% độ chính xác**.

- **Thuật toán K-Nearest Neighbors (KNN)**

KNN là một thuật toán phân loại đơn giản nhưng hiệu quả, hoạt động dựa trên khoảng cách giữa một điểm dữ liệu mới với các điểm dữ liệu trong tập huấn luyện. Thuật toán này không yêu cầu huấn luyện mô hình trước (lazy learning).

Cách hoạt động:

Chọn một số **k** (số lượng hàng xóm gần nhất).

Tính khoảng cách giữa điểm dữ liệu mới và tất cả các điểm dữ liệu trong tập huấn luyện.

Chọn **k điểm dữ liệu gần nhất** và xác định nhãn phổ biến nhất trong số đó (đa số thắng).

Gán nhãn phổ biến nhất cho điểm dữ liệu mới.

1. Import thư viện và khởi tạo biến

```
from sklearn.neighbors import KNeighborsClassifier

cv_scores = []
folds = 10
```

KNeighborsClassifier: Mô hình KNN từ thư viện scikit-learn.
cv_scores: Danh sách rỗng để lưu độ chính xác trung bình của từng giá trị k.
folds = 10: Cross-validation với 10 fold (chia dữ liệu thành 10 phần).

2. Tạo danh sách giá trị k cần thử nghiệm

```
ks = list(range(1,int(len(X) * ((folds - 1)/folds)), 2))
```

Giá trị k được chọn từ **1** đến giá trị lớn nhất là int(len(X) * 0.9) (với folds = 10).
(folds - 1)/folds: Vì trong cross-validation, mỗi fold train sử dụng 90% dữ liệu (do 1 fold dùng để test). Giá trị k không thể vượt quá kích thước tập train.
Bước nhảy 2: Chỉ chọn giá trị k lẻ để tránh trường hợp "hòa" khi phân loại nhị phân (ví dụ: 4 vs 4 láng giềng).

3. Thực hiện k-fold cross-validation cho từng k

```
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)
```

Vòng lặp qua từng k:
Khởi tạo mô hình KNN với n_neighbors = k.
cross_val_score: Tính độ chính xác (accuracy) trung bình qua 10 fold.
Lưu kết quả vào danh sách cv_scores.

4. Tìm giá trị k tối ưu

```
#---get the maximum score---
knn_score = max(cv_scores)
#---find the optimal k that gives the highest score---
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)
```

knn_score: Độ chính xác cao nhất từ danh sách cv_scores.
optimal_k: Giá trị k tương ứng với độ chính xác cao nhất (dùng cv_scores.index() để tìm vị trí).

Mục đích: Tìm giá trị k tối ưu cho mô hình KNN để dự đoán biến mục tiêu dựa trên các đặc trưng Glucose, BMI, Age.
Phương pháp: 10-fold cross-validation, thử nghiệm các giá trị k lẻ từ 1 đến 90% số lượng mẫu.
Kết quả: In ra k tối ưu và độ chính xác tương ứng, sau đó lưu vào danh sách result.

```
from sklearn.neighbors import KNeighborsClassifier
#---empty list that will hold cv (cross-validates) scores---
cv_scores = []
#---number of folds---
folds = 10
#---creating odd list of K for KNN---
ks = list(range(1,int(len(X) * ((folds - 1)/folds)), 2))
#---perform 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)
#---get the maximum score---
knn_score = max(cv_scores)
#---find the optimal k that gives the highest score---
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
```

Trong bài toán dự đoán bệnh tiểu đường, **k=19** đạt độ chính xác **77.21%** – cao nhất trong các mô hình thử nghiệm.

- **Thuật toán Support Vector Machine**

SVM là một thuật toán phân loại mạnh mẽ, hoạt động bằng cách tìm một **siêu phẳng tối ưu** (optimal hyperplane) để phân tách hai lớp dữ liệu. Siêu phẳng này có khoảng cách lớn nhất với các điểm dữ liệu gần nhất (support vectors).

1. Import thư viện:

```
from sklearn import svm
```

Import lớp svm từ scikit-learn để sử dụng các mô hình SVM.

2. Khởi tạo và đánh giá SVM với kernel linear

```
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)
```

kernel='linear': Sử dụng kernel tuyến tính, phù hợp cho dữ liệu phân tách tuyến tính.
cross_val_score:
Đánh giá mô hình bằng **10-fold cross-validation**.
Tính độ chính xác (accuracy) trung bình trên 10 lần chạy.

3. Khởi tạo và đánh giá SVM với kernel RBF

```
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)
```

kernel='rbf': Sử dụng kernel RBF (Radial Basis Function), phù hợp cho dữ liệu không phân tách tuyến tính.

cross_val_score: Tương tự như trên, đánh giá độ chính xác trung bình qua 10 fold.

4. So sánh kĩ thuật
   Kernel linear:
   Giả định dữ liệu có thể phân tách bằng một siêu phẳng tuyến tính.
   Phù hợp với dữ liệu có quan hệ tuyến tính rõ ràng.
   Ưu điểm: Đơn giản, ít tham số cần tối ưu.
   Kernel RBF:
   Ánh xạ dữ liệu vào không gian đặc trưng nhiều chiều để phân tách phi tuyến.
   Phù hợp với dữ liệu phức tạp, không phân tách tuyến tính.

```
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)

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)
```

```
0.7656527682843473
0.765704032809296
```

Hai kĩ thuật linear và RBF cho ra kết quả gần giống nhau

3. Chọn thuật toán có hiệu xuất 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

Thuật toán KNN có hiệu suất tốt nhất , các thuật toán còn lại có độ chính xác ngang nhau

4. Training và lưu model

1. Khởi tạo và huấn luyện mô hình

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

Đoạn code này tạo một mô hình KNN với tham số n_neighbors=19. Sau đó, mô hình được huấn luyện với dữ liệu X (đặc trưng) và y (nhãn).

2. Lưu mô hình

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

Đoạn này sử dụng thư viện pickle để tuần tự hóa (serialize) mô hình và lưu trữ vào file. Quá trình này giúp lưu lại mô hình đã được huấn luyện để sử dụng sau này mà không cần phải huấn luyện lại.

'wb' nghĩa là mở file ở chế độ ghi (write) và nhị phân (binary).

3. Tải lại mô hình

```
loaded_model = pickle.load(open(filename, 'rb'))
```

Đoạn này tải lại mô hình từ file vừa lưu. 'rb' nghĩa là mở file ở chế độ đọc (read) và nhị phân (binary). Sau khi thực hiện, loaded_model sẽ là một đối tượng có cùng trạng thái với knn.

4. Dự đoán và hiển thị kết quả

```
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")
```

Đoạn code này thực hiện dự đoán cho một người có:

- Glucose (đường huyết): 65
- BMI (chỉ số khối cơ thể): 70
- Age (tuổi): 50

Phương thức predict() cần một mảng 2 chiều. Vì vậy, các giá trị đặc trưng được đặt trong hai dấu ngoặc vuông [[...]]. Kết quả dự đoán được lưu trong biến prediction.

Mô hình sẽ trả về 0 nếu dự đoán là "Non-diabetic" (không mắc tiểu đường) hoặc 1 nếu dự đoán là "Diabetic" (mắc tiểu đường). Kết quả này được in ra màn hình, sau đó là thông báo dễ đọc tương ứng.

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

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

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
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py
  warnings.warn(
```

Kết quả chuẩn đoán của ví dụ là không bị mắc tiểu đường