HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO THỰC TẬP CƠ SỞ

CHỦ ĐỀ: TÌM HIỂU TRIỂN KHAI MÔ HÌNH MACHINE LEARNING

Giảng viên hướng dẫn : **PGS.TS TRẦN ĐÌNH QUẾ**
Họ và tên sinh viên : **ĐINH QUANG HƯNG**
Mã sinh viên : **B22DCCN407**
Lớp : **D22CQCN11-B**
Nhóm : **30**

Hà Nội – 2025

TÓM TẮT

Báo cáo được chia thành 3 phần với phần 1 là tìm hiểu kỹ thuật cleaning data. Phần 2 là tìm hiểu về kỹ thuật học có giám sát trong machine learning với các thuật toán như : Logistic Regression, K-Nearest Neighbors (KNN) và Support Vector Machines (với Linear Kernel và Radial Basic Function). Phần 3 là case study với việc chạy mã, giải thích và lưu mô hình.

NỘI DUNG

PHẦN 1: KỸ THUẬT CLEANING DATA

1. Cleaning data là gì?

Cleaning data hay làm sạch dữ liệu là quá trình xử lý, sửa chữa, điều chỉnh hoặc loại bỏ các dữ liệu không chính xác, không đầy đủ, sai định dạng, trùng lặp hoặc không liên quan trong tập dữ liệu.

Đây là một bước quan trọng trong quy trình phân tích dữ liệu và học máy nhằm đảm bảo dữ liệu sạch, có chất lượng cao và đáng tin cậy.

2. Thực hiện cleaning data với tệp cụ thể

Đầu tiên, ta đọc dữ liệu từ tệp dữ liệu. Tệp dữ liệu có tên là "diabetes.csv" - một tập dữ liệu y tế chứa thông tin về các bệnh nhân nữ thuộc nhóm người da đỏ Pima (Pima Indians) để dự đoán khả năng mắc bệnh tiểu đường.

```
1
2    import numpy as np
3    import pandas as pd
4    import matplotlib.pyplot as plt
5    import seaborn as sns
6
7    df = pd.read_csv('diabetes.csv')
8    df.info()
```

Kết quả:

```
<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
```

Tiếp theo kiểm tra số lượng giá trị NaN trong mỗi cột:

```
print("Nulls")
print("=====")
print(df.isnull().sum())
```

Kết quả:

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

Nhận xét: không có giá trị NaN trong tập dữ liệu.

Bước tiếp kiểm tra có bao nhiêu giá trị 0 trong mỗi cột.

```
# check 0s
print("0s")
print("+++")
print(df.eq(0).sum())
```

Kết quả:

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

Các cột Glucose, BloodPressure, SkinThickness, Insulin, BMI có giá trị 0 không hợp lý (vì không thể có huyết áp, đường huyết hay insulin bằng 0). Cần thay thế các giá trị 0 này bằng NaN để xử lý sau.

```
df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = \
    df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].replace(0, np.nan)
df.fillna(df.mean(),inplace=True)
print(df.eq(0).sum())
```

Kết quả:

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

Nhận xét: Các cột được thay thế giá trị thì không còn giá trị 0 nên quá trình làm sạch dữ liệu đã xong.

Giải thích:

Câu lệnh *df.fillna(df.mean(),inplace=True)* là để thay thế các giá trị NaN bằng giá trị trung bình cột. Cụ thể thì *df.mean()* với ý nghĩa là tính giá trị trung bình mỗi cột, *inplace=True* có nghĩa là cập nhật trực tiếp vào DataFrame mà không cần tạo bản sao mới.

Điền giá trị trung bình cho các giá trị NaN sẽ giúp giữ nguyên số lượng bộ dữ liệu mà không làm mất thông tin.

Câu lệnh *df.eq(0).sum()* sẽ kiểm tra số lượng giá trị bằng 0 trong từng cột của DataFrame. Cụ thể hơn thì *df.eq(0)* sẽ so sánh giá trị trong DataFrame với 0 và trả về True hoặc False, *.sum()* sẽ tính tổng số True trong mỗi cột.

PHẦN 2: CÁC KỸ THUẬT MACHINE LEARNING

Trong phần này kỹ thuật sẽ được đề cập đến là học có giám sát (Supervised Learning) trong Machine Learning khi dùng cho phân loại hoặc hồi quy với các thuật toán như: Logistic Regression, K-Nearest Neighbors (KNN) và Support Vector Machines (với Linear Kernel và Radial Basic Function)

1. Logistic Regression (Hồi quy Logistic)

Logistic Regression là một kỹ thuật học máy thuộc nhóm học có giám sát (supervised learning), được sử dụng chủ yếu cho bài toán phân loại (classification). Mục tiêu của kỹ thuật này là dự đoán 1 khả năng có thể xảy ra hay không xảy ra.

Cách hoạt động: Logistic Regression dựa trên ý tưởng biến đổi bài toán phân loại thành một bài toán xác suất, sử dụng hàm Sigmoid với các bước sau:

+ Đầu tiên là mô hình hóa dữ liệu:

- Dữ liệu đầu vào là các biến số $x_1, x_2,..., x_n$ (ví dụ: tuổi, thu nhập).
- Mô hình gán một trọng số w_i cho mỗi biến và thêm một hằng số bias b.
- Công thức tuyến tính ban đầu: $z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$ (z là giá trị tuyến tính)

+ Tiếp theo đó là hàm Sigmoid sẽ chuyển z thành xác suất P trong khoảng [0, 1], để mô hình có thể diễn giải kết quả dưới dạng xác suất:

$$P = \frac{1}{1 + e^{-z}}$$

Trong đó: $e \approx 2{,}718$

+ Sau đó là quyết định nhãn lớp: Ta sẽ chọn một ngưỡng nào đó ,thông thường thì là 0,5. Điều này có nghĩa là $P \geq 0{,}5$ thì nhãn là 1(sự kiện có xảy ra) , $P < 0{,}5$ thì nhãn là 0(sự kiện không xảy ra).

2. K-Nearest Neighbors (K láng giềng gần nhất-KNN)

K-Nearest Neighbors là một thuật toán học máy dựa trên sự tương đồng giữa các điểm dữ liệu. Nó thuộc nhóm học có giám sát khi dùng cho phân loại hoặc hồi quy, và đôi khi được dùng trong học không giám sát (unsupervised learning) để phân cụm.

Cơ chế hoạt động: Nó hoạt động bằng cách so sánh khoảng cách từ điểm cần phân loại đến các điểm trong tập huấn luyện, sau đó chọn K-láng giềng gần nhất. Sau đó, thuật toán sẽ dựa vào đa số phiếu bầu của các điểm láng giềng này để đưa ra dự đoán cho điểm cần phân loại. Cụ thể hơn thì nó qua các bước sau:

+ Bước 1: Tính khoảng cách.

- Để tìm láng giềng gần nhất , KNN cần đo khoảng cách giữa điểm cần dự đoán và tất cả các điểm trong tập huấn luyện. Các cách tính khoảng cách phổ biến gồm:
  - Euclidean Distance**:** (khoảng cách Euclid): $\sqrt[2]{\sum(x_i - y_i)^2}$ - phổ biến nhất.
  - Manhattan Distance:$\sum|x_i - y_i|$ - hữu ích khi dữ liệu có dạng lưới.
  - Minkowski Distance: Tổng quát hóa của Euclidean và Manhattan.

+ Bước 2: Chọn K láng giềng gần nhất.

- Sau khi tính khoảng cách, thuật toán chọn K điểm gần nhất. K thường là số lẻ để tránh trường hợp hòa phiếu trong phân loại.

+ Bước 3: Dự đoán

- Phân loại (Classification): Dựa trên đa số phiếu bầu của K láng giềng. Ví dụ, nếu K = 5, và 3/5 láng giềng thuộc lớp A, điểm mới sẽ được gán nhãn là A.
- Hồi quy (Regression): Lấy trung bình (hoặc trung vị) giá trị của K láng giềng. Ví dụ, nếu dự đoán giá nhà và 5 láng giềng có giá [100, 110, 120, 130, 140], giá dự đoán có thể là 120 (trung bình).

3. Support Vector Machines (SVM)

SVM là một thuật toán học máy có giám sát dùng để phân loại và hồi quy, nhưng phổ biến nhất trong bài toán phân loại nhị phân.

SVM hoạt động bằng cách tìm một siêu phẳng tối ưu để phân tách các nhóm dữ liệu tốt nhất. Khi dữ liệu không thể phân tách tuyến tính, SVM sử dụng Kernel Trick để ánh xạ dữ liệu sang không gian có nhiều chiều, giúp phân tách dễ hơn.

a) SVM với Linear Kernel

Linear Kernel là kernel đơn giản nhất trong SVM, giả định rằng dữ liệu có thể được phân tách tuyến tính bằng một siêu phẳng trong không gian đặc trưng ban đầu mà không cần ánh xạ lên không gian chiều cao hơn.

Công thức: $K(x, y) = x \cdot y$

Trong đó: + x,y là hai vector đặc trưng.

+ $x \cdot y$ là tích vô hướng của chúng.

Cơ chế hoạt động: Tìm 1 siêu phẳng $w \cdot x + b = 0$ sao cho:

+ w là vector pháp tuyến của siêu phẳng

+ b là hằng số bias

Khi đã có *w* và *b* bằng cách giải bài toán toán tối ưu hóa, SVM dùng siêu phẳng để:

+ Phân loại dữ liệu mới (x trong X_test với X_test là một ma trận, trong đó mỗi hàng là một điểm dữ liệu, và mỗi cột là một đặc trưng):

- Tính $w \cdot x + b$.
- Nếu > 0: Lớp 1.
- Nếu < 0: Lớp -1.

+ Trong *scikit-learn*, điều này được thực hiện bằng *predict()*

b) SVM với Radial Basis Function (RBF) Kernel

RBF Kernel (hay Gaussian Kernel) là một kernel phi tuyến, cho phép SVM phân tách dữ liệu phức tạp bằng cách ánh xạ dữ liệu lên không gian chiều cao hơn (thường là vô hạn chiều) thông qua hàm Gaussian.

Công thức:

$$K(x, y) = \exp(-\gamma \|x - y\|^2)$$

Trong đó:

+ $\|x - y\|^2$: Bình phương khoảng cách Euclidean giữa x và y.

+ γ: Tham số điều chỉnh độ rộng của hàm Gaussian ($\gamma = \frac{1}{2\sigma^2}$, với σ là độ lệch chuẩn).

Giá trị K(x,y) là giá trị tương đồng giữa x và y nằm trong khoảng [0,1] :

+ Gần 1 nếu x và y rất gần nhau.

+ Gần 0 nếu x và y xa nhau.

Giá trị K(x,y) thay thế tích vô hướng x·y (như Linear Kernel) trong quá trình tối ưu hóa.

Trong dự đoán: $f(x) = \sum w_i K(x, x_i) + b$ dựa trên K(x, x_i) để phân loại mà không cần biết tọa độ trong không gian ánh xạ. Trong đó x là một điểm dữ liệu (từ X_test) và $x_i$là một điểm dữ liệu trong tập huấn luyện (X_train). Nếu $f(x) > 0$ thì dự đoán lớp 1. Nếu $f(x) < 0$ thì dự đoán lớp -1

PHẦN 3: CASE STUDY

Ở phần này ta sẽ tiếp tục với tệp dữ liệu có tên là “diabetes.csv” mà đã được làm sạch dữ liệu ở Phần 1.

1. Kiểm tra mối tương quan giữa các cột

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

Đoạn mã df.corr() để tính toán hệ số tương quan giữa các cột trong DataFrame. Kết quả sẽ là một ma trận tương quan, trong đó mỗi giá trị thể hiện mức độ tương quan giữa hai cột dữ liệu và các giá trị này nằm trong khoảng từ [-1;1].

```
                          Pregnancies   Glucose  ...       Age   Outcome
Pregnancies                  1.000000  0.127911  ...  0.544341  0.221898
Glucose                      0.127911  1.000000  ...  0.266534  0.492928
BloodPressure                0.208522  0.218367  ...  0.324595  0.166074
SkinThickness                0.082989  0.192991  ...  0.127872  0.215299
Insulin                      0.056027  0.420157  ...  0.136734  0.214411
BMI                          0.021565  0.230941  ...  0.025519  0.311924
DiabetesPedigreeFunction    -0.033523  0.137060  ...  0.033561  0.173844
Age                          0.544341  0.266534  ...  1.000000  0.238356
Outcome                      0.221898  0.492928  ...  0.238356  1.000000

[9 rows x 9 columns]
```

Từ bảng ta có thể thấy các giá trị tương quan như: Pregnancies và Glucose là 0.127911 nên có thể thấy 2 cột này không liên quan lắm; Glucose và Outcome là 0.4929, đây là tương quan khá mạnh; Pregnancies và Outcome là 0.2210, có chút liên quan, nhưng không mạnh.

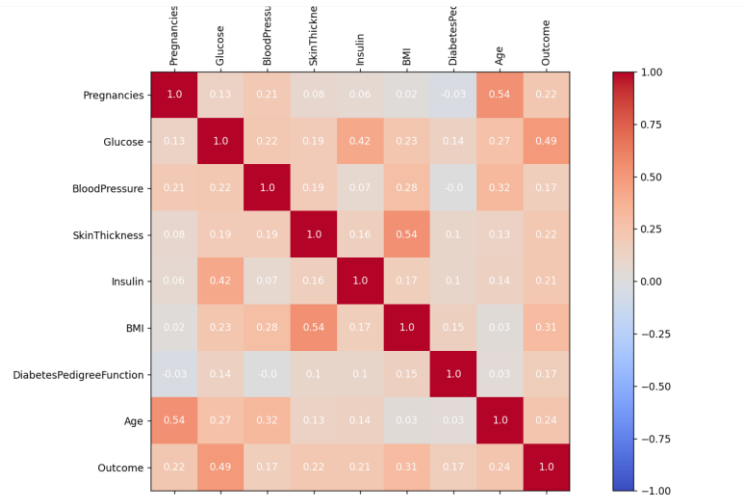
Các giá trị tương quan này có ý nghĩa: Glucose (0.4929) có tương quan khá mạnh với Outcome, nên nó là một đặc trưng quan trọng để dự đoán bệnh tiểu đường. Pregnancies (0.2210) có chút liên quan, có thể giữ lại nhưng không quá quan trọng. Nếu có một cột khác có tương quan chỉ 0.05, thì nó gần như không hữu ích và có thể loại bỏ.

2. Vẽ biểu đồ tương quan giữa các đặc trưng

Đoạn mã vẽ biểu đồ:

```
corr = df.corr()
fig, ax = plt.subplots(figsize=(10, 10))
cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,len(df.columns),1)
ax.set_xticks(ticks)
ax.set_xticklabels(df.columns)
plt.xticks(rotation = 90)
ax.set_yticklabels(df.columns)
ax.set_yticks(ticks)
#---print the correlation factor---
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()
```

Sau khi chạy sẽ có 1 biểu đồ như sau:



Màu sắc trong biểu đồ này để biểu diễn mức độ tương quan giữa các biến.

Giải thích: + Đoạn mã *plt.subplots(figsize=(10, 10))* sẽ khởi tạo biểu đồ với fig là đối tượng biểu đồ giúp quản lý toàn bộ biểu đồ và ax là trục tọa độ của biểu đồ để vẽ nội dụng trên đó

+ *ax.matshow(corr, cmap='coolwarm', vmin=-1, vmax=1)* với corr là ma trận tương quan giữa các cột *(df.corr())*,dòng *cmap='coolwarm'* có các màu: đỏ (tương quan dương mạnh nghĩa là gần 1), xanh(tương quan âm mạnh nghĩa là gần -1**)**, trắng (không có tương quan nghĩa là gần bằng 0)

+ *fig.colorbar(cax)*: Thanh màu bên cạnh giúp người xem biết màu nào ứng với giá trị bao nhiêu

+ Đoạn *mã ticks = np.arange(0, len(df.columns), 1)* sẽ tạo danh sách các chỉ số [0, 1, 2, ..., n-1], với n = số lượng cột trong df. Nó dùng để đánh dấu vị trí các nhãn trên trục X và Y. *ax.set_xticks(ticks)* sẽ xác định vị trí nhãn trên trục X. *ax.set_xticklabels(df.columns)* để gán tên các cột trong df làm nhãn trên trục X. Dòng *plt.xticks(rotation=90)* để xoay nhãn trục X 90 độ để tránh bị chồng chữ. Dòng *ax.set_yticklabels(df.columns)* để gán tên các cột làm nhãn trên trục Y. *ax.set_yticks(ticks)* để xác định vị trí nhãn trên trục Y. Ta làm như vậy do matshow() không hiển thị tên cột, nên ta phải thêm nhãn thủ công.

+ Dòng code *text = ax.text(j, i, round(corr.iloc[i][j], 2),ha="center", va="center", color="w")* có *corr.iloc[i][j]* là lấy giá trị tương quan tại vị trí [i,j] sau đó là Thêm chữ vào giữa ô *(ha="center", va="center")* với màu trắng *(color="w").*

+ Và cuối dùng là *plt.show()* là hiển thị biểu đồ

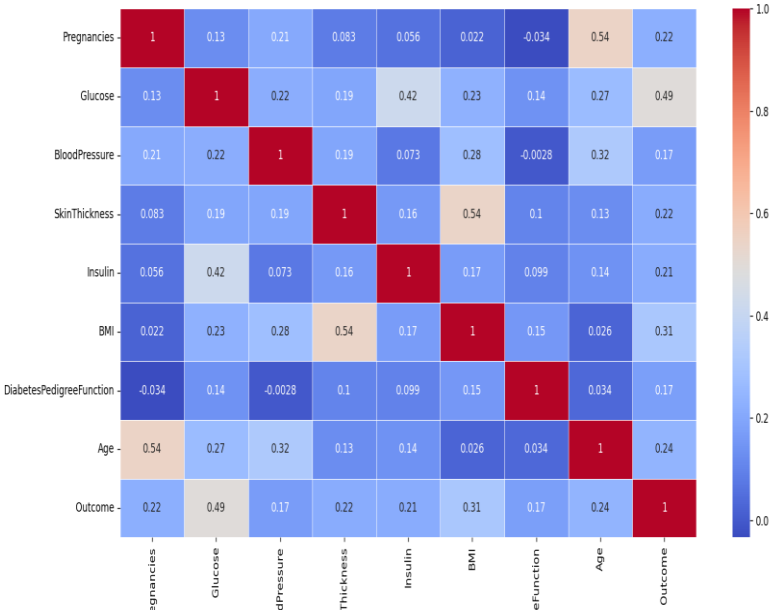
Ta có thể vẽ biểu đồ sử dụng 1 đoạn mã ngắn hơn khi dùng Seaborn:

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

plt.figure(figsize=(8, 8))  # Thiết lập kích thước mới
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", linewidths=0.5)

plt.show()  # Hiển thị heatmap mới
```

Biểu đồ:



3. Đánh giá các thuật toán

a) Hồi quy logictic

Đoạn mã để thực hiện:

```
#---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()
```

Kết quả khi chạy:

```
0.7669856459330144
```

Giải thích: + X là biến độc lập chứa 3 cột: Glucose (mức đường huyết), BMI (chỉ số khối cơ thể), Age (tuổi). Đây là các đặc trưng đầu vào mà mô hình sẽ dùng để dự đoán.

+ Dòng *df.iloc[:, 8]* dùng để lấy cột thứ 9 trong DataFrame (vì index bắt đầu từ 0). Đây là nhãn (0 hoặc 1) mà mô hình sẽ học để dự đoán có mắc bệnh tiểu đường hay không.

+ *log_regress = linear_model.LogisticRegression()*. Tạo một đối tượng log_regress thuộc lớp *LogisticRegression()*. Mô hình này sẽ tự động học trọng số (*w1, w2, ..., wn*) và bias (*b*) dựa trên dữ liệu đầu vào.

+ *log_regress_score = cross_val_score(log_regress, X, y, cv=10, scoring = 'accuracy').mean()*. Chia dữ liệu thành 10 phần bằng nhau chạy mô hình 10 lần, mỗi lần dùng 9 phần để huấn luyện, 1 phần để làm tập kiểm tra. Sau mỗi lần, đo độ chính xác (*scoring='accuracy'*) của mô hình. Thêm *.mean()* để lấy trung bình của 10 lần kiểm tra.

b) K láng giềng gần nhất (KKN)

Đoạn mã thực hiện:

```
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)
```

Kết quả khi chạy:

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

Giải thích: + *folds = 10* là số lượng tập con. Đòng *code ks = list(range(1, int(len(X) * ((folds - 1)/folds)), 2))* tạo danh sách các giá trị k lẻ (ks) để thử nghiệm với KNN , *int(len(X) * ((folds - 1)/folds))* đảm bảo giá trị k không quá lớn so với tập dữ liệu.

+ *knn = KNeighborsClassifier(n_neighbors=k)* .Khi khởi tạo *KNeighborsClassifier()*, mặc định nó sử dụng khoảng cách Euclid để tính khoảng cách giữa các điểm.

+ *score = cross_val_score(knn, X, y, cv=folds, scoring='accuracy').mean()*. *cross_val_score()* kiểm tra độ chính xác của KNN bằng cách dự đoán nhãn cho tập kiểm tra. Mô hình phân loại (*KNeighborsClassifier*) sẽ dựa trên đa số phiếu từ K hàng xóm để đưa ra dự đoán.

+ Đoạn mã *knn_score = max(cv_scores)* và *optimal_k = ks[cv_scores.index(knn_score)]* để tìm độ chính xác cao nhất trong danh sách *cv_scores* sau đó lấy chỉ số (*index()*) của *knn_score* trong danh sách *cv_scores* để tìm ra giá trị k tối ưu nhất

c) Support Vector Machines với Linear Kernel

Đoạn mã thực hiện:

```
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)
```

Kết quả khi chạy:

```
0.7656527682843473
```

Giải thích: + Dòng code svm.SVC(kernel='linear') tạo một mô hình SVM dùng kernel tuyến tính. Điều này tương ứng với công thức K(x,y)=x·y .Nó sẽ tìm siêu phẳng tối ưu $w \cdot x + b = 0$ để phân tách dữ liệu.

+ Dòng cross_val_score(linear_svm, X, y, cv=10, scoring='accuracy').mean() để chia thành 10 tập dữ liệu để đánh giá độ chính xác của mô hình. Tức là mô hình sẽ được huấn luyện 10 lần với tập dữ liệu chia khác nhau, sau đó lấy trung bình độ chính xác.

d) Support Vector Machines với RBF (Radial Basis Function)

Đoạn mã thực hiện:

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

Kết quả khi chạy:

```
0.765704032809296
```

Giải thích: + *svm.SVC(kernel='rbf')*: Tạo một mô hình SVM sử dụng Radial Basis Function (RBF) Kernel.

+ *rbf_score = cross_val_score(rbf, X, y, cv=10, scoring='accuracy').mean()* chia dữ liệu thành 10 phần (folds). Huấn luyện trên 9 phần, kiểm tra trên 1 phần và lặp lại 10 lần với mỗi phần là tập kiểm tra một lần. Sau đó lấy trung bình độ chính xác từ 10 lần chạy *(.mean())*

Nhận xét: Ta thấy thuật toán K láng giềng gần nhất (KKN) có độ chính xác cao nhất.

4. Huấn luyện và lưu mô hình

Vì thuật toán K láng giềng gần nhất (KKN) hoạt động tốt nhất cho tập dữ liệu của chúng ta là KNN với k = 19 nên ta có thể tiến hành huấn luyện mô hình bằng toàn bộ tập dữ liệu.

Đoạn mã thực hiện:

```
import pickle

knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(X, y)
filename = 'diabetes.sav'
pickle.dump(knn, open(filename, 'wb'))

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

Kết quả thực hiện là ta có thêm 1 file “disabetes.sav”

Giải thích đoạn mã : + *KNeighborsClassifier(n_neighbors=19)* sẽ khởi tạo mô hình KNN với k = 19, nghĩa là khi dự đoán một điểm dữ liệu mới, mô hình sẽ xem xét 19 điểm dữ liệu gần nhất trong tập huấn luyện.

+ *knn.fit(X, y)* sẽ huấn luyện mô hình KNN với dữ liệu với X là các đặc trưng của dữ liệu huấn luyện và y là nhãn tương ứng của dữ liệu huấn luyện (0: Không bị tiểu đường, 1: Bị tiểu đường).

+ *pickle.dump(knn, open(filename, 'wb'))* để lưu mô hình knn vào tệp “diabetes.sav” dưới dạng nhị phânpickle.load(open(filename, 'rb'))

+ *loaded_model = pickle.load(open(filename, 'rb'))* là đoạn mã mở tệp “diabetes.sav” ở chế độ đọc nhị phân và tải lại mô hình đã lưu. Biến loaded_model chứa mô hình KNN đã được huấn luyện trước đó.

Sau khi lưu mô hình và tải, ta sẽ áp dụng để dự đoán bệnh tiểu đường với các thông số Glucose = 65, BMI = 70, Age = 50

Đoạn mã thực hiện:

```
Glucose = 65
BMI = 70
Age = 50
prediction = loaded_model.predict(pd.DataFrame([[Glucose, BMI, Age]]))

if prediction[0]==0:
 print("Non-diabetic")
else:
 print("Diabetic")

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

Kết quả thực hiện:

```
Non-diabetic
[[0.94736842 0.05263158]]
Confidence: 94.74%
```

Giải thích: + Đoạn mã *pd.DataFrame([[Glucose, BMI, Age]])* sẽ chuyển danh sách [65, 70, 50] thành một DataFrame.

+ Đoạn mã *loaded_model.predict()* sẽ dùng mô hình đã huấn luyện (*loaded_model*) để dự đoán kết quả. Trả về 0 (Non-diabetic) hoặc 1 (Diabetic).

+ *prediction[0]*: Lấy giá trị dự đoán đầu tiên (vì kết quả là một mảng NumPy). Nếu là 0 thì in ra "Non-diabetic" (Không bị tiểu đường), nếu là 1 thì in ra "Diabetic" (Bị tiểu đường).

+ Dòng code có *loaded_model.predict_proba()* sẽ trả về xác suất của mỗi lớp

+ Đoạn mã *str(round(np.amax(proba[0]) * 100, 2))* sẽ Lấy giá trị lớn nhất trong mảng xác suất rồi nhân với 100 và làm tròn 2 chữ số, sau đó thì ta có kết quả như trên hình.

-HẾT-