Buổi thực hành 4

Hồi quy tuyến tính, k-NN và các kỹ thuật đánh giá

1. Mục tiêu và Liên kết môn học:

- **Mục tiêu:**
  - Sinh viên thực hiện được quy trình học máy 5 bước (Get Data, Prepare, Train, Evaluate, Improve)[1].
  - Sinh viên vận dụng thành thạo các kỹ thuật tiền xử lý dữ liệu thiết yếu:
    - Xử lý dữ liệu thiếu (Missing values) sử dụng SimpleImputer[2].
    - Chuẩn hóa dữ liệu (Data Scaling) sử dụng StandardScaler [3] và hiểu rõ tầm quan trọng của nó với các thuật toán dựa trên khoảng cách (như k-NN).
    - Xử lý dữ liệu mất cân bằng (Imbalanced Data) sử dụng kỹ thuật SMOTE.
  - Sinh viên xây dựng và huấn luyện được 2 mô hình học có giám sát: LinearRegression (cho bài toán Hồi quy) [4]và KNeighborsClassifier (cho bài toán Phân loại)[5].
  - Sinh viên biết cách sử dụng các độ đo đánh giá đa dạng:
    - Hồi quy: **MAE**, **MSE**, **RMSE**, và **R2-Score**.
    - Phân loại: **Accuracy**, **Confusion Matrix**, và báo cáo chi tiết **Precision, Recall, F1-score** (đặc biệt cho lớp thiểu số).
  - Sinh viên biết cách áp dụng các kỹ thuật đánh giá mô hình nâng cao: **Cross-Validation** (cross_val_score) và **Tinh chỉnh tham số** (GridSearchCV).
- **Chuẩn đầu ra (CLOs) đáp ứng (từ Đề cương):**
  - **CLO3:** Phân tích và đánh giá được độ hiệu quả của các kỹ thuật máy học thông qua các độ đo thực nghiệm.
  - **CLO4:** Vận dụng được các kỹ thuật máy học (học có giám sát, học không giám sát) để giải quyết bài toán cụ thể.
  - **CLO5:** Sử dụng được các thư viện lập trình (Pandas, Scikit-learn) để triển khai các ứng dụng máy học.

2. Công cụ và Dữ liệu:

- **Công cụ:** Python, Jupyter Notebook/Google Colab, và các thư viện:
  - Pandas: Tải và thao tác dữ liệu.
  - Numpy: Hỗ trợ tính toán (tạo dữ liệu thiếu, tính RMSE).
  - Scikit-learn (sklearn): Cung cấp mọi công cụ chính (train_test_split, SimpleImputer, StandardScaler, LinearRegression, KNeighborsClassifier, các hàm metrics, Pipeline, cross_val_score, GridSearchCV).
  - Matplotlib / Seaborn: Trực quan hóa (tùy chọn).
  - imblearn (imbalanced-learn): Để cài đặt và sử dụng SMOTE. (Cần cài đặt bằng pip install -U imbalanced-learn).

- **Dữ liệu:**
  - **Phần 1 (Code-along):**
    - Hồi quy: **California Housing** (sklearn.datasets.fetch_california_housing).
    - Phân loại: **Breast Cancer** (sklearn.datasets.load_breast_cancer) (để minh họa xử lý mất cân bằng nhẹ).
  - **Phần 2 (Tự làm):**
    - Hồi quy: **Diabetes** (sklearn.datasets.load_diabetes).
    - Phân loại: **Wine** (sklearn.datasets.load_wine).

3. Nội dung thực hành:

Phần 1: Hướng dẫn thực hành (Code-along - 150 phút)

Bài toán 1: Hồi quy (Regression) & Tiền xử lý dữ liệu thiếu

Mục tiêu: Xây dựng mô hình dự đoán giá nhà (California Housing) với quy trình tiền xử lý chuẩn, bao gồm xử lý dữ liệu thiếu và đánh giá bằng nhiều độ đo.

Bước 1: Tải và Khám phá dữ liệu (Get Data)

Python

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing

# Tải dữ liệu
housing = fetch_california_housing()
X = pd.DataFrame(housing.data,
columns=housing.feature_names)
y = pd.Series(housing.target, name='MedHouseVal')

print("5 dòng dữ liệu X đầu tiên:")
print(X.head())

# Kiểm tra dữ liệu thiếu ban đầu (sẽ không có)
print("\nKiểm tra dữ liệu thiếu ban đầu:")
print(X.isnull().sum())
```

Bước 2: Chuẩn bị dữ liệu (Prepare Data) - Nâng cao

- **Giải thích:** Dữ liệu thực tế thường bị thiếu[6]. Ta sẽ *tạo dữ liệu thiếu giả* để học cách xử lý bằng SimpleImputer, vốn là một kỹ thuật điền giá trị thiếu tự động (ví dụ: điền bằng giá trị trung bình)[7]. Sau đó, ta chuẩn hóa dữ liệu bằng

StandardScaler[8].

Code:

```Python
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# --- Bước 2.1: Tạo dữ liệu thiếu giả ---
# Chọn 1% dữ liệu trong 2 cột đầu tiên và biến chúng thành NaN (Missing)
X_missing = X.copy()
mask1 = X_missing['MedInc'].sample(frac=0.01, random_state=42).index
mask2 = X_missing['HouseAge'].sample(frac=0.01, random_state=42).index
X_missing.loc[mask1, 'MedInc'] = np.nan
X_missing.loc[mask2, 'HouseAge'] = np.nan

print("\nKiểm tra dữ liệu thiếu sau khi tạo (giả lập):")
print(X_missing.isnull().sum())

# --- Bước 2.2: Phân chia Train/Test ---
X_train, X_test, y_train, y_test = train_test_split(X_missing, y, test_size=0.2, random_state=42)

# --- Bước 2.3: Xử lý dữ liệu thiếu (Imputation) ---
# Chiến lược 'mean': điền giá trị thiếu bằng giá trị trung bình của cột đó
imputer = SimpleImputer(strategy='mean')

# Chỉ 'fit' trên X_train để học giá trị trung bình
imputer.fit(X_train)

# 'transform' cho cả X_train và X_test
X_train_imputed = imputer.transform(X_train)
X_test_imputed = imputer.transform(X_test)

# --- Bước 2.4: Chuẩn hóa dữ liệu (Scaling) ---
scaler = StandardScaler()
```

```
# Chỉ 'fit' trên X_train_imputed
scaler.fit(X_train_imputed)

# 'transform' cho cả hai
X_train_scaled = scaler.transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)

print(f"\nĐã tiền xử lý xong. Kích thước tập huấn luyện:
{X_train_scaled.shape}")
```

Bước 3: Huấn luyện mô hình (Train Model)

- **Giải thích:** Huấn luyện mô hình LinearRegression [9] trên tập dữ liệu đã được làm sạch và chuẩn hóa.
  Code:
  Python
  ```
  from sklearn.linear_model import LinearRegression

  model_lr = LinearRegression()
  model_lr.fit(X_train_scaled, y_train)
  print("Đã huấn luyện xong mô hình Hồi quy tuyến tính.")
  ```

Bước 4: Đánh giá mô hình (Evaluate Model) - Nâng cao

- **Giải thích:** Sử dụng nhiều độ đo để có cái nhìn toàn diện về lỗi của mô hình.
  - **MAE (Mean Absolute Error):** Lỗi trung bình tuyệt đối. Dễ hiểu (ví dụ: mô hình dự đoán sai lệch trung bình $X đô la).
  - **MSE (Mean Squared Error):** Lỗi trung bình bình phương. Trừng phạt các lỗi lớn nặng hơn.
  - **RMSE (Root Mean Squared Error):** Căn bậc hai của MSE. Cùng đơn vị với biến mục tiêu (dễ hiểu như MAE).
  - **R2 Score:** Hệ số xác định. Cho biết mô hình giải thích được bao nhiêu % phương sai của dữ liệu (càng gần 1 càng tốt).

  Code:
  Python
  ```
  from sklearn.metrics import mean_absolute_error,
  mean_squared_error, r2_score

  y_pred_lr = model_lr.predict(X_test_scaled)

  mae = mean_absolute_error(y_test, y_pred_lr)
  mse = mean_squared_error(y_test, y_pred_lr)
  rmse = np.sqrt(mse) # Tính RMSE từ MSE
  ```

```
r2 = r2_score(y_test, y_pred_lr)

print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R-squared (R2) Score: {r2:.4f}")
```

Bài toán 2: Phân loại (k-NN) & Xử lý dữ liệu mất cân bằng

Mục tiêu: Xây dựng mô hình k-NN dự đoán ung thư vú, phát hiện vấn đề mất cân bằng và cải thiện bằng SMOTE.

Bước 1: Tải và Khám phá dữ liệu (Get Data)

Python

```
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
X_bc = pd.DataFrame(data.data, columns=data.feature_names)
y_bc = pd.Series(data.target, name='target')

# --- Bước 1.1: Kiểm tra độ mất cân bằng ---
print("Phân bố các lớp (0=Ác tính, 1=Lành tính):")
print(y_bc.value_counts())
# Kết quả: 1 (lành tính) nhiều hơn 0 (ác tính) -> Dữ liệu hơi mất cân bằng.
```

Bước 2: Tiền xử lý dữ liệu (Prepare Data)

- **Giải thích:**
  1. stratify=y: Rất quan trọng khi chia train/test, đảm bảo tỷ lệ các lớp trong tập train và test tương tự như tập gốc.
  2. StandardScaler: *Bắt buộc* với k-NN, vì k-NN dựa trên khoảng cách. Nếu không chuẩn hóa, các đặc trưng có thang đo lớn sẽ "át" các đặc trưng có thang đo nhỏ.
- **Code:**

```
Python
# Phân chia dữ liệu, giữ nguyên tỷ lệ các lớp với 'stratify=y_bc'
X_train_bc, X_test_bc, y_train_bc, y_test_bc = train_test_split(
    X_bc, y_bc, test_size=0.2, random_state=42, stratify=y_bc
```

```
)

# Chuẩn hóa
scaler_bc = StandardScaler()
X_train_bc_scaled = scaler_bc.fit_transform(X_train_bc)
X_test_bc_scaled = scaler_bc.transform(X_test_bc)
```

Bước 3 & 4: Huấn luyện và Đánh giá (Lần 1 - Dữ liệu gốc)

- **Giải thích:** Huấn luyện mô hình k-NN (k=5). Đánh giá bằng classification_report [10] để xem chi tiết F1-score của lớp 0 (thiểu số).
- **Code:**

```
Python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

model_knn_1 = KNeighborsClassifier(n_neighbors=5)
model_knn_1.fit(X_train_bc_scaled, y_train_bc)

y_pred_knn_1 = model_knn_1.predict(X_test_bc_scaled)

print("--- Kết quả (Lần 1 - Dữ liệu gốc) ---")
print(f"Accuracy: {accuracy_score(y_test_bc,
y_pred_knn_1):.4f}")
print("Confusion Matrix:\n", confusion_matrix(y_test_bc,
y_pred_knn_1))
print("Classification Report:\n",
classification_report(y_test_bc, y_pred_knn_1,
target_names=data.target_names))
```

- **Phân tích:** Dù Accuracy cao (ví dụ: 95%), hãy nhìn vào **Recall** và **F1-score** của lớp malignant (ác tính). Đây là lớp quan trọng (chúng ta không muốn bỏ sót ca ác tính).

Bước 5: Cải thiện (Improve) - Xử lý mất cân bằng với SMOTE

- **Giải thích:** SMOTE (Synthetic Minority Over-sampling Technique) sẽ tạo ra các mẫu "tổng hợp" mới cho lớp thiểu số (lớp 0) *chỉ trong tập huấn luyện* để cân bằng dữ liệu.
- **Code:**

```
Python
from imblearn.over_sampling import SMOTE
```

```
print("\n--- Áp dụng SMOTE để cân bằng tập huấn luyện ---")

smote = SMOTE(random_state=42)

# Chỉ fit_resample trên tập TRAIN
X_train_resampled, y_train_resampled =
smote.fit_resample(X_train_bc_scaled, y_train_bc)

print("Phân bố các lớp sau khi SMOTE (tập train):")
print(y_train_resampled.value_counts())

# --- Huấn luyện và Đánh giá (Lần 2 - Dữ liệu SMOTE) ---
model_knn_2 = KNeighborsClassifier(n_neighbors=5)

# Huấn luyện trên dữ liệu đã resample
model_knn_2.fit(X_train_resampled, y_train_resampled)

# Đánh giá trên tập TEST gốc (không resample)
y_pred_knn_2 = model_knn_2.predict(X_test_bc_scaled)

print("\n--- Kết quả (Lần 2 - Dữ liệu đã SMOTE) ---")
print(f"Accuracy: {accuracy_score(y_test_bc,
y_pred_knn_2):.4f}")
print("Confusion Matrix:\n", confusion_matrix(y_test_bc,
y_pred_knn_2))
print("Classification Report:\n",
classification_report(y_test_bc, y_pred_knn_2,
target_names=data.target_names))
```

- **Phân tích:** So sánh 2 Báo cáo Phân loại. Kết quả lần 2 (với SMOTE) có thể có Accuracy tổng thể giảm nhẹ, nhưng **Recall** và **F1-score** của lớp malignant (lớp 0) thường sẽ tăng lên, cho thấy mô hình phát hiện ca ác tính tốt hơn.

Phần 2: Bài tập tự làm và nộp bài

Sinh viên áp dụng các kỹ thuật nâng cao: Pipeline, cross_val_score, và GridSearchCV.

Yêu cầu 1: Hồi quy (Diabetes) & Kiểm định chéo (Cross-Validation)

- **Dữ liệu:** sklearn.datasets.load_diabetes()
- **Nhiệm vụ (Tasks):**
  1. Tải dữ liệu Diabetes.
  2. Tạo một Pipeline (sklearn.pipeline.Pipeline) bao gồm 2 bước:
     - StandardScaler()

    - LinearRegression()
3. Sử dụng hàm cross_val_score từ sklearn.model_selection để đánh giá Pipeline của bạn.
4. Thiết lập cv=5 (5-fold cross-validation).
5. Thiết lập scoring='r2' để tính R2-score và scoring='neg_mean_squared_error' để tính MSE. (Lưu ý: cross_val_score trả về MSE âm, cần lấy giá trị tuyệt đối hoặc nhân với -1).
6. In ra R2-score trung bình và MSE trung bình của 5 lần chạy.

Yêu cầu 2: Phân loại (Wine) & Tinh chỉnh tham số với GridSearchCV

- **Dữ liệu:** sklearn.datasets.load_wine() (Phân loại 3 lớp rượu)
- **Nhiệm vụ (Tasks):**
    1. Tải dữ liệu Wine. Phân chia dữ liệu (Train/Test, tỷ lệ 80/20, random_state=42).
    2. Tạo một Pipeline (sklearn.pipeline.Pipeline) bao gồm 2 bước:
        - StandardScaler()
        - KNeighborsClassifier()
    3. Tạo một param_grid (dạng dictionary) để GridSearchCV tìm tham số k (tức là n_neighbors) tốt nhất cho KNeighborsClassifier.
        - Gợi ý param_grid: {'kneighborsclassifier__n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15]}
        - (Lưu ý: Tên tham số trong Pipeline phải có dạng tên_bước__tên_tham_số).
    4. Sử dụng GridSearchCV từ sklearn.model_selection để bọc Pipeline và param_grid. Thiết lập cv=5 và scoring='accuracy'.
    5. fit GridSearchCV trên tập *huấn luyện*.
    6. In ra best_params_ (tham số tốt nhất) và best_score_ (accuracy trung bình tốt nhất).
    7. Sử dụng mô hình tốt nhất (grid_search.best_estimator_) để dự đoán trên tập *kiểm thử* và in ra classification_report.