THỰC HÀNH: CÁC GIẢI THUẬT PHÂN LOẠI CƠ BẢN

A. MỤC TIÊU CHƯƠNG

Mục tiêu của phần thực hành các giải thuật phân loại dữ liệu (Data Classification Algorithms) được thiết kế để giúp sinh viên phát triển kỹ năng áp dụng các giải thuật phân loại vào việc giải quyết các bài toán thực tế trong khoa học dữ liệu. Cụ thể, phần thực hành nhằm đạt được các mục tiêu sau: Hiểu và triển khai các thuật toán phân loại: Cây quyết định và rừng cây (Decision Tree & Random Forest), Support Vector Machine (SVM) và Bayes ngây thơ (Naive Bayes) thông qua ngôn ngữ lập trình như Python và sử dụng các thư viện Scikit-learn, SciPy, …

Hướng dẫn thực hiện việc triển khai xây dựng một mô hình học máy cổ điển với các bước:
- Tiền xử lý dữ liệu: Hướng dẫn cách chuẩn bị dữ liệu, bao gồm xử lý giá trị thiếu, chuẩn hóa dữ liệu, và mã hóa các biến phân loại để phù hợp với yêu cầu của các thuật toán phân loại.
- Đánh giá và tối ưu hóa mô hình: sinh viên sẽ thực hành đánh giá hiệu suất mô hình thông qua các chỉ số như accuracy, precision, recall, F1-score và sử dụng các kỹ thuật như cross-validation, grid search để tối ưu hóa tham số.

- Phân tích và diễn giải kết quả: Phát triển khả năng diễn giải kết quả phân loại, nhận diện các vấn đề như quá khớp (overfitting) và đề xuất cải tiến.

B. KẾT CẤU THỰC HÀNH

Thực hành bao gồm 3 phần là
- Giải thuật cây quyết định và rừng cây
- Giải thuật Support Vector Machine (SVM)
- Giải thuật Bayes ngây thơ

C. NỘI DUNG THỰC HÀNH

2.1. GIẢI THUẬT 1: CÂY QUYẾT ĐỊNH VÀ RỪNG CÂY

2.1.1. Ôn tập lý thuyết

1) Quy trình khai phá dữ liệu CRISP – DM (Cross Industry Standard Process for Data Mining) là gì ? Quy trình khai phá dữ liệu SEMMA (Sample, Explore, Modify, Model, Assess) là gì?

- Quy trình khai phá dữ liệu là một chuỗi lặp (iterative) và tương tác (interactive) gồm các bước (giai đoạn) bắt đầu với dữ liệu thô (raw data) và kết thúc với tri thức (knowledge of interest) đáp ứng được sự quan tâm của người sử dụng.

- CRISP – DM (Cross Industry Standard Process for Data Mining) là mô hình toàn diện thiên về quản lý dự án và kinh doanh, quy trình lặp, có khả năng quay lui (backtracking) gồm 6 giai đoạn: Tìm hiểu nghiệp vụ (Business understanding), tìm hiểu dữ liệu (Data understanding), chuẩn bị dữ liệu (Data preparation), mô hình hóa (Modeling), đánh giá (Evaluation), triển khai (Deployment).

- SEMMA (Sample, Explore, Modify, Model, Assess) là mô hình kỹ thuật, tập trung vào các bước phân tích dữ liệu chuyên sâu, có 5 bước như sau: Lấy mẫu để mô hình hóa (Sample), khám phá và tìm hiểu về dữ liệu (Explore), làm sạch và chuẩn bị dữ liệu (Modify), áp dụng thuật toán máy học (Model), kiểm tra và đánh giá mô hình (Assess).

2) Cây quyết định hoạt động như thế nào? Hãy giải thích các thành phần chính (nút gốc, nút lá, nhánh) và cách cây đưa ra dự đoán.

- Cây quyết định hoạt động bằng cách chia tập dữ liệu lớn thành các tập con nhỏ hơn dựa trên việc kiểm tra các đặc trưng (features) của dữ liệu. Các thành phần chính trong cây:

+ Nút gốc (Root Node): Là nút đầu tiên của cây. Nó đại diện cho toàn bộ tập dữ liệu ban đầu và chứa câu hỏi kiểm tra quan trọng nhất (thuộc tính phân loại tốt nhất).

+ Nhánh (Branch): Là đường nối các nút. Mỗi nhánh đại diện cho một kết quả có thể xảy ra của câu hỏi kiểm tra tại nút trước đó.

+ Nút lá (Leaf Node): Là các nút cuối cùng của cây và không có nhánh nào đi ra. Nút lá đại diện cho kết quả hoặc dự đoán cuối cùng (nhãn lớp hoặc giá trị).

- Cây quyết định đưa ra dự đoán bằng cách áp dụng một chuỗi các quy tắc IF-THEN đã được học từ dữ liệu huấn luyện, cho đến khi mẫu dữ liệu rơi vào một danh mục cuối cùng (Nút Lá).

3) Các tiêu chí phân tách (splitting criteria) như Gini Index, Entropy, hay Information Gain được sử dụng trong cây quyết định là gì? Chúng khác nhau ra sao?

- Các tiêu chí phân tách (Gini Index, Entropy, và Information Gain) là các thước đo toán học được sử dụng để xác định đặc trưng (feature) và giá trị (value) tốt nhất để chia dữ liệu tại mỗi nút, nhằm tạo ra các tập con (child nodes) "tinh khiết" nhất có thể.

Tiêu chí	Vai trò & Mục tiêu	Nguyên tắc
Entropy	Đo lường mức độ hỗn loạn (độ không tinh khiết) trong một tập dữ liệu.	Tính dựa trên hàm logarit của phân phối xác suất lớp. Mô hình muốn Entropy giảm sau khi chia.
Information Gain (Độ lợi thông tin)	Tiêu chí quyết định. Đo lường sự giảm Entropy đạt được sau khi thực hiện một phép chia cụ thể.	Là sự chênh lệch giữa Entropy trước khi chia và Entropy sau khi chia. Tối đa hóa Information Gain.
Gini Index (Gini Impurity)	Đo xác suất phân loại sai nếu chọn một mẫu ngẫu nhiên. Chỉ số Gini = 0 là tinh khiết hoàn toàn.	Tính dựa trên bình phương của xác suất lớp. Tối thiểu hóa Gini Index.

4) Rừng cây (Random Forest) là gì? Nó khác gì so với một cây quyết định đơn lẻ? Tại sao Random Forest thường có hiệu suất tốt hơn cây quyết định trong các bài toán phân loại?

- Rừng cây (Random Forest) là một thuật toán học máy tập hợp (Ensemble Learning). Thay vì chỉ sử dụng một cây quyết định đơn lẻ, nó xây dựng và kết hợp kết quả của nhiều cây quyết định độc lập để đưa ra dự đoán cuối cùng.

- Bảng so sánh giữa cây quyết định đơn lẻ (Single Decision Tree) và rừng cây (Random Forest):

Đặc điểm	Cây quyết định đơn lẻ (Single Decision Tree)	Random Forest (Rừng Cây)
Cấu trúc	Một mô hình duy nhất.	Tập hợp của nhiều cây quyết định.
Tính đa dạng	Không có.	Rất cao, do dùng kỹ thuật Bagging và ngẫu nhiên hóa thuộc tính.
Độ chính xác	Thấp hơn, dễ bị Overfitting (học quá khớp) với dữ liệu huấn luyện.	Cao hơn, nhờ vào việc kết hợp nhiều quan điểm khác nhau.
Phương sai	Cao (dễ thay đổi mạnh khi dữ liệu huấn luyện thay đổi).	Thấp (ổn định và đáng tin cậy hơn).

- Random Forest thường có hiệu suất tốt hơn cây quyết định đơn lẻ vì nó giúp giảm phương sai (Variance) mà không tăng nhiều độ chệch (Bias) của mô hình:

+ Giảm overfitting: Cây quyết định đơn lẻ có xu hướng học quá chi tiết các nhiễu trong dữ liệu, dẫn đến overfitting. Random Forest khắc phục điều này bằng cách lấy trung bình dự đoán của nhiều cây, làm phẳng các quyết định quá chi tiết và giảm thiểu ảnh hưởng của các nhiễu.

+ Tính ổn định (Robustness): Bằng cách sử dụng nhiều cây được huấn luyện trên các tập dữ liệu hơi khác nhau, lỗi dự đoán của một cây sẽ được bù trừ bởi các cây khác, dẫn đến một dự đoán cuối cùng ổn định và chính xác hơn.

5) Những ưu điểm và hạn chế của cây quyết định và Random Forest là gì? Trong trường hợp nào thì cây quyết định có thể hoạt động kém hiệu quả?

- Những ưu điểm và hạn chế của cây quyết định và Random Forest:

Đặc điểm	Cây quyết định (Decision Tree)	Rừng cây (Random Forest)
Ưu điểm Chính	Dễ hiểu & Dễ giải thích (Interpretability). Cấu trúc là một chuỗi quy tắc IF-THEN rõ ràng.	Độ chính xác và độ ổn định cao. Hiệu suất vượt trội trên hầu hết các tập dữ liệu, ít bị nhiễu.
Hạn chế Chính	Dễ bị Overfitting (Học quá khớp) và không ổn định (Unstable). Rất nhạy cảm với sự thay đổi nhỏ của dữ liệu.	Khó giải thích (Less Interpretable), hoạt động như một "hộp đen" do kết hợp nhiều cây.
Hiệu suất	Tính toán nhanh khi huấn luyện và dự đoán.	Tốn thời gian và tài nguyên tính toán hơn để huấn luyện hàng trăm cây.
Xử lý dữ liệu	Yêu cầu ít tiền xử lý hơn.	Vẫn yêu cầu xử lý dữ liệu, nhưng có khả năng xử lý tốt hơn các vấn đề thiếu dữ liệu ngẫu nhiên.

- Các trường hợp hoạt động kém hiệu quả của cây quyết định:

+ Dễ bị Overfitting: Khi cây quá sâu, nó học các nhiễu thay vì các mẫu tổng quát, dẫn đến hiệu suất kém trên dữ liệu mới.

+ Mối quan hệ Tuyến tính: Khi mối quan hệ giữa các biến là tuyến tính đơn giản, Cây Quyết định phải tạo ra nhiều phân tách phức tạp để xấp xỉ đường thẳng đó, kém hiệu quả hơn mô hình hồi quy tuyến tính.

+ Dữ liệu không ổn định: Dữ liệu có nhiều biến động hoặc nhiễu ngẫu nhiên có thể làm thay đổi hoàn toàn cấu trúc cây, dẫn đến dự đoán không nhất quán.

6) Viết đoạn code mẫu bằng Python (sử dụng Scikit-learn) để xây dựng một mô hình cây quyết định không? Hãy mô tả các bước thực hiện

- Đoạn code mẫu sử dụng Python dựa trên tập dữ liệu Iris có sẵn để xây dựng mô hình:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load data and get sample
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
# Modeling
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=42)# criterion='gini' là tiêu chí phân tách mặc định
dt_model.fit(X_train, y_train)
# Prediction and accuracy
y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree Accuracy: {accuracy:.2f}")
```

- Mô tả các bước thực hiện:

+ Tải và chia dữ liệu: Tải dữ liệu mẫu và chia nó thành tập huấn luyện (Train) và kiểm tra (Test) để đảm bảo mô hình được đánh giá công bằng.

+ Khởi tạo và huấn luyện: Khởi tạo mô hình DecisionTreeClassifier, thiết lập các tham số (ví dụ: max_depth=3 để tránh overfitting), và sử dụng phương thức fit() trên dữ liệu huấn luyện.

+ Dự đoán và đánh giá: Sử dụng phương thức predict() trên tập kiểm tra (X_test) và đánh giá hiệu suất bằng thước đo độ chính xác (Accuracy).

7) Làm thế nào để triển khai một mô hình Random Forest trong Python? Bạn thường thiết lập các tham số nào (ví dụ: n_estimators, max_depth)?

- Để triển khai mô hình Random Forest trong Python ta sử dụng lớp RandomForestClassifier (cho phân loại) hoặc RandomForestRegressor (cho hồi quy) của Scikit-learn.

- Mã mẫu trong Python sử dụng dữ liệu dữ liệu X_train, y_train, X_test từ câu 6:

```
from sklearn.ensemble import RandomForestClassifier
# Modeling
rf_model = RandomForestClassifier(
    n_estimators=150,      # Số lượng cây
    max_depth=8,           # Chiều sâu tối đa của mỗi cây
    random_state=42
)
rf_model.fit(X_train, y_train)
# Prediction and accuracy
rf_pred = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")
```

- Các giá trị điển hình của các tham số thường thiết lập:
  + n_estimators: 100 đến 500 (thường là 100-200).
  + max_depth: 3 đến 15 (tùy theo độ phức tạp của dữ liệu).

8) Làm thế nào để đánh giá tầm quan trọng của các đặc trưng (feature importance) trong Random Forest bằng Python?

- Random Forest tự động đánh giá tầm quan trọng của các đặc trưng thông qua thuộc tính feature_importances_ sau khi huấn luyện.

- Mã mẫu trong Python sử dụng rf_model đã huấn luyện ở câu 7:

```
# Importance Features
importances = rf_model.feature_importances_
feature_names = iris.feature_names
```

- Cơ chế hoạt động của tầm quan trọng được tính bằng cách đo mức độ giảm độ không tinh khiết trung bình (Mean Decrease Impurity - MDI) mà mỗi đặc trưng đóng góp trong tất cả các cây. Đặc trưng nào giúp giảm sự hỗn loạn nhiều nhất sẽ có điểm số cao nhất.

9) Điều chỉnh siêu tham số (hyperparameter tuning) cho cây quyết định hoặc Random Forest chưa? Hãy mô tả cách bạn sử dụng GridSearchCV hoặc RandomizedSearchCV

- Để tìm bộ siêu tham số tốt nhất cho Random Forest, ta sử dụng GridSearchCV hoặc RandomizedSearchCV từ Scikit-learn.

- Mã mẫu trong Python sử dụng GridSearchCV:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
# Grid settings
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, None], # None nghĩa là cây phát triển hết mức
    'min_samples_split': [2, 5]
}
# Modeling
rf = RandomForestClassifier(random_state=42)
# GridSearchCV
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,                  # Sử dụng 5-fold cross-validation
    scoring='accuracy',
    n_jobs=-1              # Sử dụng tất cả các nhân CPU
)
# Finding the best params
grid_search.fit(X, y)
```

- Mô tả phương pháp: Cả hai phương pháp đều sử dụng kiểm định chéo (Cross-Validation - cv) để đánh giá hiệu suất của mỗi sự kết hợp tham số một cách đáng tin cậy.

Phương pháp	Cách hoạt động	Khi nào sử dụng
GridSearchCV	Thử nghiệm mọi sự kết hợp có thể của các tham số được định nghĩa trong lưới (param_grid).	Khi lưới tham số nhỏ và bạn muốn đảm bảo tìm ra sự kết hợp tối ưu nhất.
RandomizedSearchCV	Chọn ngẫu nhiên một số lượng giới hạn các sự kết hợp tham số từ lưới.	Khi lưới tham số rất lớn và bạn muốn tìm kiếm một kết quả tốt một cách nhanh chóng hơn, tiết kiệm thời gian tính toán.

2.1.2. Bài làm mẫu

Bài toán 1: Xây dựng cây quyết định và rừng cây với dữ liệu lấy từ https://www.kaggle.com/datasets/deceneu/default-of-credit-card-clients

Nhiệm vụ 1: Xây dựng cây quyết định bằng thư viện Scikit-Learn

1. Tải một số package mà chúng tôi sử dụng và package graphviz, để vẽ cây quyết định

```
import numpy as np #numerical computation
import pandas as pd #data wrangling
import matplotlib.pyplot as plt #plotting package
#Next line helps with rendering plots
%matplotlib inline
import matplotlib as mpl #add'l plotting functionality
mpl.rcParams['figure.dpi'] = 400 #high res figures
import graphviz #to visualize decision trees
```

2. Nạp dữ liệu vào bộ nhớ, phân tích và loại bỏ những features không liên quan đến bài toán cần giải quyết

```
df = pd.read_csv('data.csv') #Load the cleaned data
features_response = df.columns.tolist() #Get a list of column names
#Make a list of columns to remove that aren't features or the response
variable
items_to_remove = ['ID', 'SEX', 'PAY_2', 'PAY_3',\
                   'PAY_4', 'PAY_5', 'PAY_6',\
                   'EDUCATION_CAT', 'graduate school',\
                   'high school', 'none',\
                   'others', 'university']
features_response = [item for item in features_response if item not
in items_to_remove]
features_response
```

3. Chuẩn bị dữ liệu cho tập train và tập test

```
from sklearn.model_selection import train_test_split
```

```
from sklearn import tree
#Split the data into training and testing sets using the same random seed
X_train, X_test, y_train, y_test = \
    train_test_split(df[features_response[:-1]].values,
    df['default payment next month'].values,
    test_size=0.2, random_state=24)
```

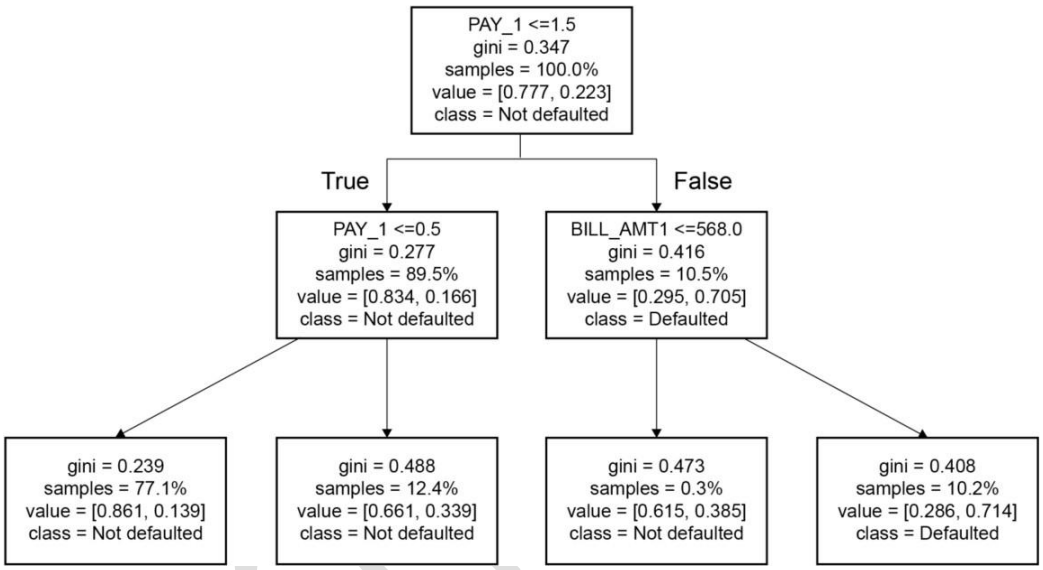
4. Xây dựng cây quyết định từ lớp DecisionTreeClassifier có trong thư viện Scikit-Learn

```
# the tree will grow to a depth of at most 2
dt = tree.DecisionTreeClassifier(max_depth=2)
dt.fit(X_train, y_train)
```

5. Hiển thị cây quyết định với package **graphviz**

```
dot_data = tree.export_graphviz(dt,
                out_file=None,
                filled=True,
                rounded=True,
                feature_names=\
                  features_response[:-1],
                  proportion=True,
                  class_names=['Not defaulted', 'Defaulted'])
graph = graphviz.Source(dot_data)
graph
```

Kết quả ta được cây quyết định như hình bên dưới



Hình 2.1 - Cây quyết định

Nhiệm vụ 2: Tìm tham số tối ưu cho cây quyết định bằng GridSearchCV và vẽ biểu đồ đánh giá mô hình với các tham số khác nhau

1. Thực hiện 1, 2 và 3 như ở nhiệm vụ 1 để tải thư viện, nạp dữ liệu và chuẩn bị dữ liệu
2. Tạo cây quyết định và xác định các giá trị tham số có thể chọn để tìm tham số tối ưu

```
from sklearn.model_selection import GridSearchCV
params = {'max_depth':[1, 2, 4, 6, 8, 10, 12]} #parameters
dt = tree.DecisionTreeClassifier() #tree modal
cv = GridSearchCV(dt, param_grid=params, scoring='roc_auc',
                n_jobs=None, refit=True, cv=4, verbose=1,
                error_score=np.nan,
                return_train_score=True) # cv is the best model.
cv.fit(X_train, y_train)
```

Sử dụng ChatGPT để tìm hiểu thêm các khái niệm bên dưới

scoring = 'roc_auc': the ROC AUC metric

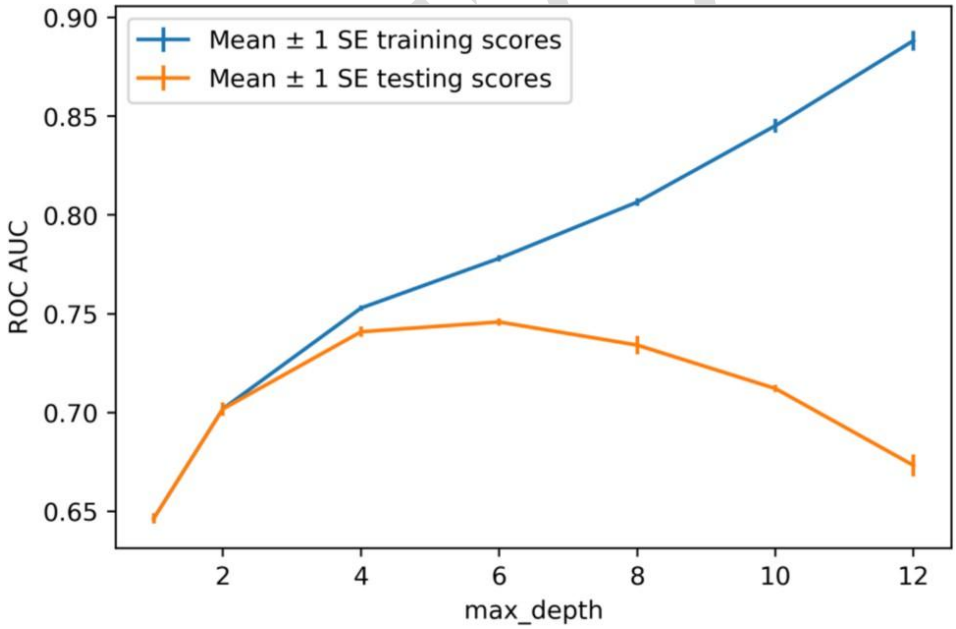
cv = 4: 4-fold cross-validation

return_train_score = true: để đánh giá bias và variance

3. Vẽ biểu đồ đánh giá mô hình với các tham số chiều sâu của cây khác nhau

```
cv_results_df = pd.DataFrame(cv.cv_results_)
#View the names of the remaining columns in the results DataFrame
cv_results_df.columns
ax = plt.axes()
ax.errorbar(cv_results_df['param_max_depth'],
            cv_results_df['mean_train_score'],
            yerr=cv_results_df['std_train_score']/np.sqrt(4),
            label='Mean $\pm$ 1 SE training scores')
ax.errorbar(cv_results_df['param_max_depth'],
            cv_results_df['mean_test_score'],
            yerr=cv_results_df['std_test_score']/np.sqrt(4),
            label='Mean $\pm$ 1 SE testing scores')
ax.legend()
plt.xlabel('max_depth')
plt.ylabel('ROC AUC')
```

Kết quả ta được biểu đồ so sánh kết quả đánh giá mô hình với các tham số khác nhau



Hình 2.2 - Biểu đồ đánh giá hiệu quả thực hiện cây quyết định với các chiều sâu khác nhau

Nhiệm vụ 3: Xây dựng rừng cây (random forest)

1. Thực hiện 1, 2 và 3 như ở nhiệm vụ 1 để tải thư viện, nạp dữ liệu và chuẩn bị dữ liệu

2. Tạo rừng cây với lớp RandomForestClassifier trong Scikit-Learn

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier\
  (n_estimators=10, criterion='gini', max_depth=3,
   min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
   max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0,
   bootstrap=True, oob_score=False, n_jobs=None,
   random_state=4, verbose=0, warm_start=False, class_weight=None)
```

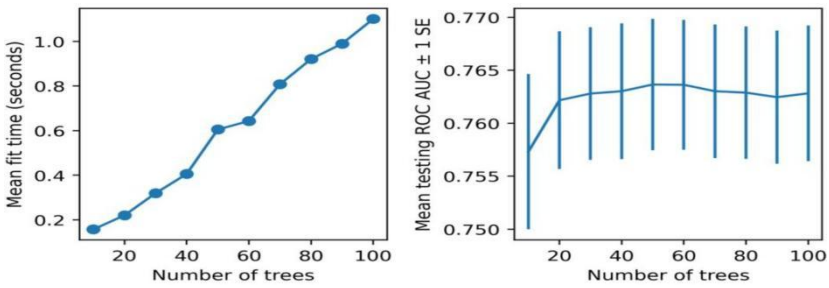
3. Tìm tham số tối ưu cho mô hình rừng cây và thực hiện train với tham số tối ưu đó

```
#a parameter grid for this exercise in order to search the numbers of
trees, ranging from 10 to 100 by 10s
rf_params_ex = {'n_estimators':list(range(10,110,10))}
cv_rf_ex = GridSearchCV(rf, param_grid=rf_params_ex,
                        scoring='roc_auc', n_jobs=None,
                        refit=True, cv=4, verbose=1,
                        error_score='np.nan',
                        return_train_score=True)
cv_rf_ex.fit(X_train, y_train)
```

4. Vẽ biểu đồ đánh giá mô hình rừng cây với các tham số số cây có trong rừng khác nhau

```
cv_rf_ex_results_df = pd.DataFrame(cv_rf_ex.cv_results_)
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(6, 3))
axs[0].plot(cv_rf_ex_results_df['param_n_estimators'],
            cv_rf_ex_results_df['mean_fit_time'],
            '-o')
axs[0].set_xlabel('Number of trees')
axs[0].set_ylabel('Mean fit time (seconds)')
axs[1].errorbar(cv_rf_ex_results_df['param_n_estimators'],
                cv_rf_ex_results_df['mean_test_score'],
                yerr=cv_rf_ex_results_df['std_test_score']/np.sqrt(4))
axs[1].set_xlabel('Number of trees')
axs[1].set_ylabel('Mean testing ROC AUC $\pm$ 1 SE ')
plt.tight_layout()
```

Kết quả thực hiện:

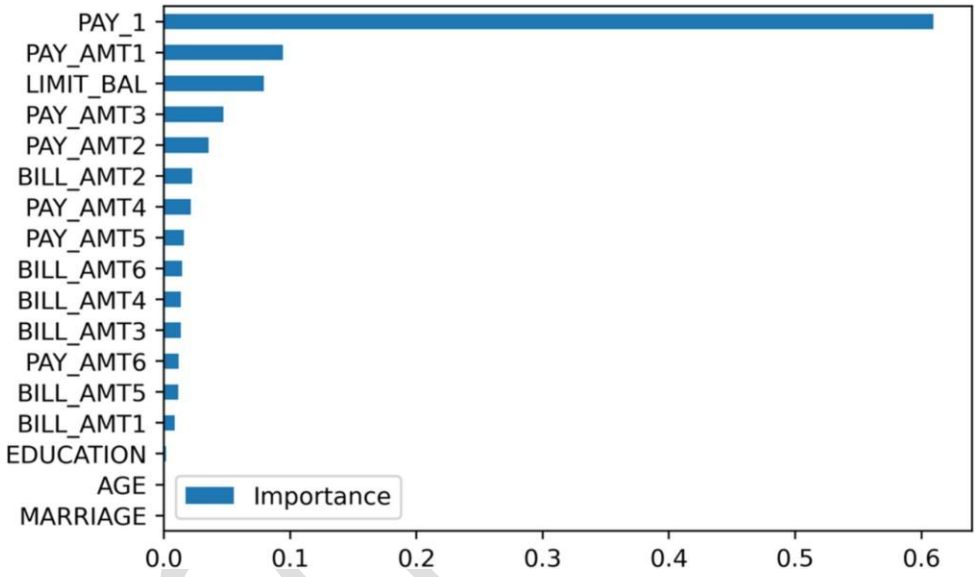


Hình 2.3 - Biểu đồ thể hiện mối quan hệ giữa số cây với Mean Fit Time và Mean Testing ROC AUC

5. Xem tham số tốt nhất của rừng cây, xem mức độ quan trọng của từng feature trong mô hình với tham số tốt nhất

```
# {'n_estimators': 50}
cv_rf_ex.best_params_
# the feature names and importances
feat_imp_df = pd.DataFrame({
    'Importance':cv_rf_ex.best_estimator_.feature_importances_},
    index=features_response[:-1])
feat_imp_df.sort_values('Importance', ascending=True).plot.barh()
```

Kết quả thực hiện



Hình 2.4 - Biểu đồ hiển thị mức độ quan trọng của từng feature

2.1.3. Bài tập thực hành 1

Xây dựng cây quyết định và rừng cây trên dữ liệu Titanic lấy từ https://www.kaggle.com/code/dmilla/introduction-to-decision-trees-titanic-dataset

- Phần bài làm nằm trong thư mục có tên là "Code" và tên là "DT-RF_titanic.ipynb".

2.1.4. Bài tập thực hành 2

Xây dựng cây quyết định và rừng cây trên dữ liệu bệnh tiểu đường. Dữ liệu lấy từ https://www.kaggle.com/code/tumpanjawat/diabetes-eda-random-forest-hp

- Phần bài làm nằm trong thư mục có tên là "Code" và tên là "DT-RF_diabetes_prediction.ipynb".

2.2. GIẢI THUẬT 2: SUPPORT VECTOR MACHINE (SVM)

2.2.1. Ôn tập lý thuyết

1) Giải thuật Support Vector Machine hoạt động như thế nào? Hãy giải thích khái niệm về ranh giới phân tách (hyperplane) và lề (margin)

- SVM là một thuật toán học máy có giám sát, chủ yếu dùng cho phân loại (Classification), hồi quy (Regression) và phát hiện ngoại lai. Nó hoạt động bằng cách tìm một siêu phẳng (Hyperplane) tối ưu để phân tách các lớp dữ liệu sao cho khoảng cách (Margin) giữa Hyperplane và các điểm dữ liệu gần nhất của mỗi lớp là lớn nhất. Mục tiêu là tối đa hóa Margin để mô hình tổng quát hóa tốt hơn trên dữ liệu mới, giảm nguy cơ overfitting.

+ Hyperplane (ranh giới phân tách): Đây là một mặt phẳng trong không gian đa chiều phân chia dữ liệu thành hai lớp. Trong không gian 2D, nó là đường thẳng; trong 3D, là mặt phẳng; và trong n chiều, là siêu phẳng (n-1 chiều). Công thức cơ bản: $w \cdot x + b = 0$, trong đó w là vector pháp tuyến, x là điểm dữ liệu, b là bias.

+ Margin (lề): Khoảng cách từ Hyperplane đến các điểm gần nhất (Support Vectors) của mỗi lớp. SVM tìm Hyperplane sao cho Margin lớn nhất, giúp mô hình robust hơn với nhiễu.

2) Các vector hỗ trợ (support vectors) có vai trò gì trong SVM? Tại sao chúng quan trọng?

- Support Vectors là các điểm dữ liệu nằm sát nhất với Hyperplane (ngay trên hoặc trong Margin). Chúng là những điểm quyết định vị trí của Hyperplane.

- Vai trò: Mô hình SVM chỉ phụ thuộc vào Support Vectors để xây dựng Hyperplane; các điểm khác không ảnh hưởng. Chúng định nghĩa Margin và giúp tối ưu hóa bài toán toán học (sử dụng Lagrange multipliers).

- Quan trọng: Nếu thay đổi hoặc xóa Support Vectors, Hyperplane có thể thay đổi hoàn toàn. Chúng giúp SVM tiết kiệm bộ nhớ (Sparse Model) và chống overfitting bằng cách tập trung vào các điểm "khó" nhất. Trong dữ liệu thực tế, chỉ một phần nhỏ dữ liệu trở thành Support Vectors.

3) Sự khác biệt giữa SVM với lề cứng (hard margin) và lề mềm (soft margin) là gì? Khi nào nên sử dụng lề mềm?

- Hard Margin SVM: Không cho phép bất kỳ điểm dữ liệu nào nằm trong Margin hoặc sai lớp. Chỉ áp dụng khi dữ liệu hoàn toàn tuyến tính phân tách được (Linearly Separable) mà không có nhiễu. Tham số C được coi như vô hạn.

- Soft Margin SVM: Cho phép một số điểm nằm trong Margin hoặc sai lớp (Slack Variables) để xử lý nhiễu và dữ liệu không hoàn hảo. Tham số C kiểm soát mức phạt cho các vi phạm.

- Khi nào dùng soft Margin: Hầu hết các bài toán thực tế (dữ liệu có nhiễu, chồng lấn lớp, hoặc không tuyến tính phân tách hoàn toàn). Hard Margin dễ dẫn đến overfitting hoặc không tìm được giải pháp nếu dữ liệu không lý tưởng.

4) Hàm nhân (kernel) trong SVM là gì? Hãy giải thích các loại kernel phổ biến (linear, polynomial, RBF) và khi nào nên sử dụng chúng

- Kernel (Kernel Trick) là kỹ thuật biến đổi dữ liệu từ không gian gốc sang không gian chiều cao hơn mà không cần tính toán trực tiếp tọa độ mới, giúp xử lý dữ liệu phi tuyến tính. Nó tính khoảng cách giữa các điểm qua hàm Kernel mà không cần không gian mới.

+ Linear Kernel: $K(x_i, x_j) = x_i^T x_j$. Dùng khi dữ liệu gần tuyến tính phân tách, nhanh và đơn giản.

+ Polynomial Kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$. Dùng cho dữ liệu có mối quan hệ phi tuyến bậc thấp (d=2–5), như hình ảnh đơn giản.

+ RBF (Radial Basis Function) Kernel: $K(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2)$. Phổ biến nhất, dùng cho dữ liệu phi tuyến phức tạp, không biết dạng dữ liệu trước.

- Khi nào dùng: Bắt đầu với RBF cho hầu hết trường hợp; linear nếu dữ liệu lớn và tuyến tính; polynomial nếu cần bậc cụ thể. Sử dụng GridSearchCV để chọn Kernel tốt nhất.

5) Tham số C trong SVM có ý nghĩa gì? Nó ảnh hưởng như thế nào đến hiệu suất của mô hình?

- Tham số C là hệ số điều chỉnh (Regularization Parameter) trong soft Margin SVM, kiểm soát sự đánh đổi giữa việc tối đa hóa Margin và giảm lỗi phân loại.

+ Ý nghĩa: C lớn → phạt nặng lỗi phân loại → cố gắng phân tách đúng hết → Margin nhỏ hơn, dễ overfitting. C nhỏ → cho phép nhiều lỗi hơn → Margin lớn, mô hình đơn giản hơn, dễ underfitting.scikit-learn.orgtowardsdatascience.com

+ Ảnh hưởng: C cao tăng variance (overfit trên train), giảm bias. C thấp tăng bias, giảm variance (tổng quát hóa tốt hơn). Thường dùng GridSearchCV để tìm C tối ưu (ví dụ: 0.1–100).

6) Viết đoạn code mẫu bằng Python (sử dụng Scikit-learn) để xây dựng một mô hình SVM cho bài toán phân loại không? Hãy mô tả các bước thực hiện

- Dưới đây là code mẫu viết bằng Python sử dụng bộ dữ liệu Iris từ scikit-learn:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, Classification _report
# Load data and split train/test & Scaling
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create and train SVM
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)
# Prediction and accuracy
y_pred = svm_model.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(Classification _report(y_test, y_pred))
```

- Mô tả các bước thực hiện:

+ Tải dữ liệu mẫu (Iris).

+ Chia dữ liệu thành train/test.

+ Chuẩn hóa để các đặc trưng có thang đo ngang nhau.

+ Tạo mô hình SVM với kernel phù hợp và huấn luyện.

+ Dự đoán trên test và đánh giá (accuracy, precision, recall,...).

7) Hàm nào trong Scikit-learn để chuẩn hóa dữ liệu (scaling) trước khi áp dụng SVM? Tại sao bước này quan trọng?

- Hàm chính: StandardScaler từ sklearn.preprocessing (chuẩn hóa về mean=0, std=1). Ngoài ra có MinMaxScaler (đưa về [0,1]).scikit-learn.org

- Tại sao quan trọng: SVM dựa trên khoảng cách Euclidean giữa các điểm. Nếu đặc trưng có thang đo khác nhau (ví dụ: tuổi 0–100, lương 0–1e6), đặc trưng lớn sẽ chi phối → Hyperplane lệch lạc. Scaling đảm bảo mọi đặc trưng góp phần ngang nhau, cải thiện hiệu suất và tốc độ hội tụ. Không scale có thể làm accuracy giảm mạnh (từ 80% xuống 60% trên một số tập).

2.2.2. Bài làm mẫu

Bài toán 1: Thực hiện các nhiệm vụ trong bài toán 1 để xây dựng mô hình với giải thuật SVM cho dữ liệu Iris-data lấy từ https://www.kaggle.com/code/xvivancos/tutorial-knn-in-the-iris-data-set

Nhiệm vụ 1: Xây dựng mô hình SVM để phân loại các loài hoa cẩm chướng

1. Tải dữ liệu về, nạp dữ liệu, xem thông tin các feature có trong tập dữ liệu và chuẩn bị dữ liệu cho xây dựng mô hình

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
# Download&Load dữ liệu iris từ datasets của scikit-learn
iris = datasets.load_iris()
# Hiển thị mô ta dữ liệu, chỉ có trong các bộ dữ liệu chuẩn và mở để học tập và nghiên cứu
print(iris.DESCR)
# Từ tập dữ liệu ban đầu, tách lấy ma trận biểu diễn các đặc trưng và nhãn.
data = iris.data
target = iris.target
# TODO: Chia dữ liệu và nhãn thành 2 tập dữ liệu huấn luyện và dữ liệu kiểm tra theo tỉ lệ 80:20




X_train, X_test, y_train, y_test = train_test_split(data, target, test_size
                                        = 0.2, random_state=101)
```

2. Tạo mô hình SVM với dữ liệu đã chuẩn bị

```
from sklearn import svm
# khởi tạo mô hình phân lớp
clf = svm.SVC()
# Sử dụng phương thức 'fit' để huấn luyện mô hình với dữ liệu huấn luyện và nhãn huấn luyện
# fit (X,Y) với X là tập các đối tượng, Y là tập nhãn tương ứng của đối tượng.
clf.fit(X_train, y_train)
```

3. Đánh giá độ chính xác của mô hình

```
# Tính độ chính xác trên tập huấn luyện và tập kiểm tra
train_acc = clf.score(X_train,y_train)
val_acc = clf.score(X_test,y_test)
print('Training accuracy: {}'.format(train_acc))
print('Validation accuracy: {}'.format(val_acc))
```

4. Tìm tham số kernel tối ưu cho mô hình SVM

```
# best_svm, best_val_acc và best_kernel lần lượt là các biến lưu mô hình tốt nhất,
# độ chính xác cao nhất trên tập kiểm tra và kernel tốt nhất
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
best_svm = None
best_val_acc = -1
best_kernel = None
# Huấn luyện các mô hình dựa trên dữ liệu huấn luyện và tham số kernel
# Tính toán độ chính xác trên tập huấn luyện và tập kiểm tra để tìm được mô hình tốt nhất
for i in range(4):
    clf = svm.SVC(kernel=kernels[i], probability=True)
    clf.fit(X_train, y_train)
    tmp_val_acc = clf.score(X_test, y_test)
    if (tmp_val_acc > best_val_acc):
        best_val_acc = tmp_val_acc
        best_svm = clf
        best_kernel = kernels[i]
# Hiển thị mô hình tốt nhất cùng với độ chính xác
print("Best validation accuracy : {} with kernel: {}".format(best_val_acc, best_kernel))
# Mô hình tốt nhất của bạn nên có độ chính xác xấp xỉ 86,67%
```

Ghi chú: Sinh viên có thể sử dụng GridSearchCV để tìm tham số tối ưu cho mô hình SVM trên.

Bài toán 2: Xây dựng mô hình dựa vào giải thuật SVM trên dữ liệu hình ảnh Handwritten-Digit-MNIST-SVM. Dữ liệu lấy từ https://www.kaggle.com/code/nishan192/mnist-digit-recognition-using-svm

Nhiệm vụ 1: Tìm hiểu về cách biểu diễn và hiển thị các ảnh từ tập dữ liệu là hình ảnh

1. Import thư viện và tải dữ liệu là tập các hình ảnh viết tay từ số 0 đến số 9

2. Khảo sát thông tin có trong digits

```
#thông tin toàn bộ dữ liệu đã tải về
digits
#xem thông tin của một hình dưới dạng ma trận 8 x 8
digits['data'][0].reshape(8,8)
```

```
#xem thông tin của một hình dưới dạng mảng
digits['data'][0]
#xem thông tin 9 nhãn đầu tiên
digits['target'][0:9]
```

3. Vẽ ra hình dựa vào dữ liệu dạng ma trận 8 x 8

```
# Each Digit is represented in digits.images as a matrix of 8x8 = 64 pixels. Each of the 64 values represent
# a greyscale. The Greyscale are then plotted in the right scale by the imshow method.
fig, ax = plt.subplots(8,8, figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(digits.images[i], cmap='binary')
    axi.set(xticks=[], yticks=[])
```

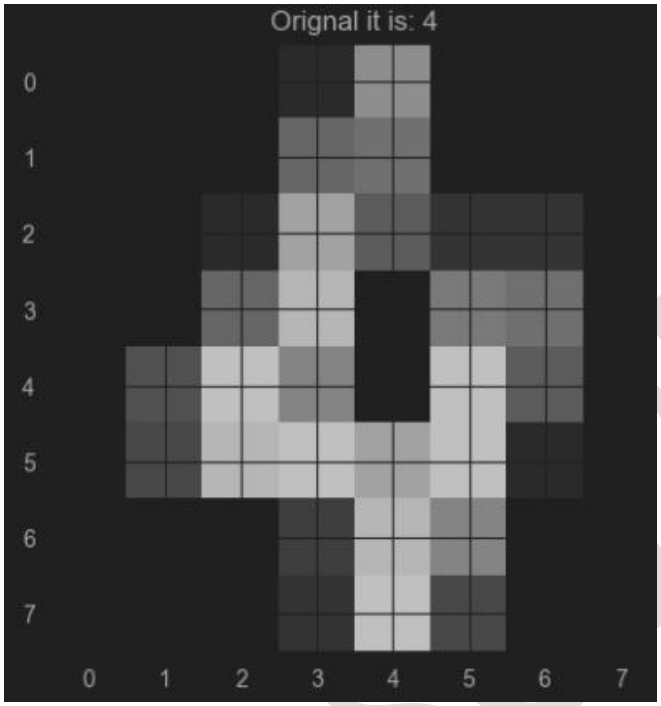
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.datasets import load_digits
digits = load_digits(n_class=10)
```

Kết quả thực hiện

4. Vẽ một ảnh từ ma trận 8 x 8

```
# Hàm vẽ 1 ảnh có kích thước 8 x 8 (ảnh lấy từ ma images)
def view_digit(index):
    plt.imshow(digits.images[index] , cmap = plt.cm.gray_r)
    plt.title('Orignal it is: '- str(digits.target[index]))
    plt.show()
# vẽ ảnh ở vị trí thứ 4
view_digit(4)
```

Kết quả thực hiện



Nhiệm vụ 2: Xây dựng mô hình SVM để nhận diện chữ viết tay từ 0 – 9

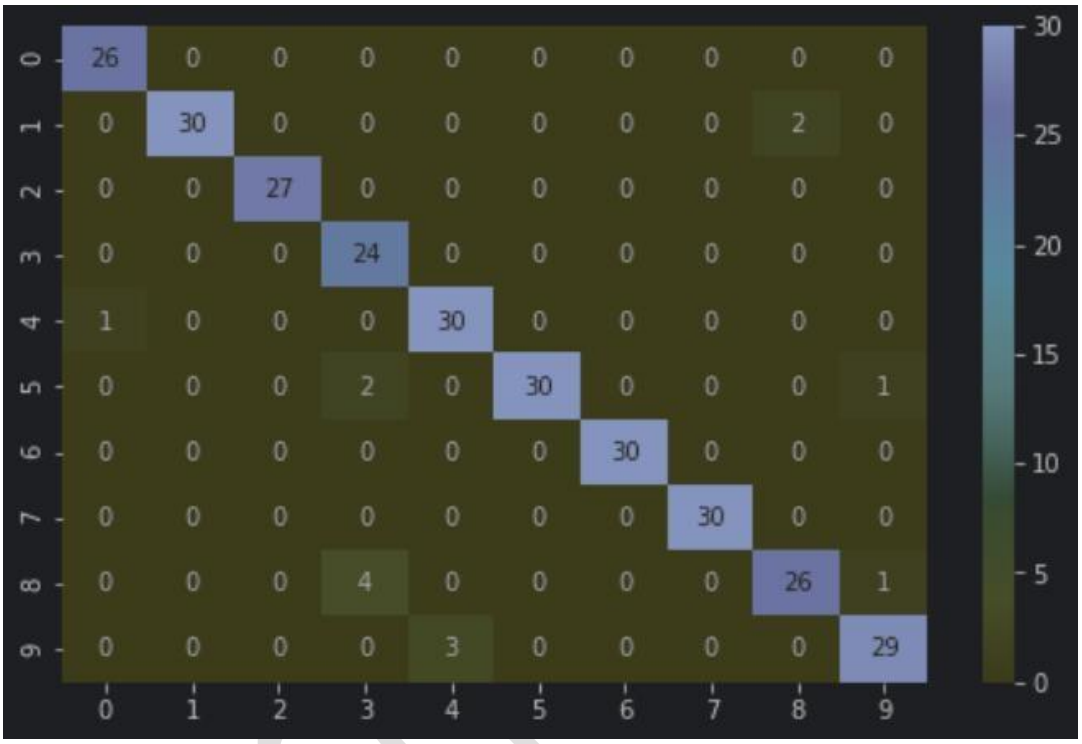
1. Chuẩn bị dữ liệu và xây dựng mô hình SVM

```
# Thực hiện import các thư viện cần thiết để xây dựng mô hình SVM
# Thực hiện bước 1 của nhiệm vụ 1
from sklearn import svm
main_data = digits['data']
targets = digits['target']
svc = svm.SVC(gamma=0.001 , C = 100)
# GAMMA is a parameter for non linear hyperplanes.
# The higher the gamma value it tries to exactly fit the training data set
# C is the penalty parameter of the error term.
#  It controls the trade off between smooth decision boundary and
classifying the training points correctly.
svc.fit(main_data[:1500] , targets[:1500])
predictions = svc.predict(main_data[1501:])
# list(zip(predictions , targets[1501:]))
```

2. Đánh giá hiệu quả của mô hình với Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(predictions, targets[1501:])
conf_matrix = pd.DataFrame(data = cm)
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu");
```

Kết quả thực hiện



Hình 2.5 - Confusion matrix trình bày dưới dạng head map

3. In kết quả dạng văn bản

```
from sklearn.metrics import classification_report
print(classification_report(predictions, targets[1501:]))
```

Kết quả thực hiện

	precision	recall	f1-score	support
0	0.96	1.00	0.98	26
1	1.00	0.94	0.97	32
2	1.00	1.00	1.00	27
3	0.80	1.00	0.89	24
4	0.91	0.97	0.94	31
5	1.00	0.91	0.95	33
6	1.00	1.00	1.00	30
7	1.00	1.00	1.00	30
8	0.93	0.84	0.88	31
9	0.94	0.91	0.92	32

Hình 2.6 - Thông tin đánh giá về hiệu quả của mô hình trên từng lớp

2.2.3. Bài tập thực hành 1

Xây dựng mô hình từ giải thuật SVM trên dữ liệu bệnh tiểu đường. Dữ liệu lấy từ https://www.kaggle.com/code/tumpanjawat/diabetes-eda-random-forest-hp

- Phần bài làm nằm trong thư mục có tên là "Code" và tên là "SVM_diabetes_prediction.ipynb".

2.2.4. Bài tập thực hành 2

Xây dựng mô hình từ giải thuật SVM trên dữ liệu các con thú trong rừng. Dữ liệu lấy từ https://www.kaggle.com/code/kareemellithy/animal-condition-predict-svm-knn

- Phần bài làm nằm trong thư mục có tên là "Code" và tên là "SVM_animal_condition.ipynb".

2.3. GIẢI THUẬT 3: BAYES NGÂY THƠ (NAÏVE BAYES)

2.3.1. Ôn tập lý thuyết

1) Giải thuật Naive Bayes hoạt động như thế nào? Hãy giải thích định lý Bayes và giả định "ngây thơ" trong thuật toán này?

- Giải thuật Naive Bayes là một nhóm các thuật toán phân loại dựa trên Định lý Bayes với một giả định độc lập "ngây thơ" giữa các đặc trưng (features). Mục tiêu của thuật toán là dự đoán xác suất một mẫu dữ liệu thuộc về một lớp nhất định $P(y|x_1, x_2, \ldots, x_n)$ và chọn lớp có xác suất cao nhất làm kết quả dự đoán.

- Định lý Bayes (Bayes Theorem):

+ Định lý Bayes mô tả mối quan hệ giữa xác suất có điều kiện và xác suất biên của các sự kiện ngẫu nhiên. Trong bối cảnh phân loại, nó được biểu diễn như sau:

$$P(y|X) = [P(X|y) \times P(y)]/P(X)$$

+ Trong đó:

- $P(y|X)$: Xác suất hậu nghiệm (Posterior probability) - xác suất để một mẫu dữ liệu X thuộc về lớp y. Đây là điều mà mô hình muốn tính toán.
- $P(X|y)$: Xác suất khả năng (Likelihood) - xác suất quan sát được mẫu dữ liệu X khi biết nó thuộc về lớp y.
- $P(y)$: Xác suất tiên nghiệm (Prior probability) - xác suất ban đầu (trước khi quan sát dữ liệu) để một mẫu bất kỳ thuộc về lớp y.
- $P(X)$: Xác suất bằng chứng (Evidence) - xác suất biên của việc quan sát mẫu dữ liệu X. P(X) là hằng số cho mọi lớp trong quá trình dự đoán, nên ta thường bỏ qua nó khi so sánh các lớp.

- Giả định "Ngây thơ" (The "Naive" Assumption):

+ Giả định "ngây thơ" chính là lý do thuật toán có tên là Naive Bayes. Giả định này tuyên bố rằng tất cả các đặc trưng (features) x_i là độc lập có điều kiện với nhau khi biết lớp y.

+ Điều này cho phép ta đơn giản hóa công thức tính xác suất khả năng $P(X|y)$:

$$P(X|y) = P(x_1, x_2, \ldots, x_n|y) \approx P(x_1|y) \times P(x_2|y) \times \ldots \times P(x_n|y)$$

+ Quy tắc phân loại: Ta chọn lớp có xác suất hậu nghiệm cao nhất

2) Các loại mô hình Naive Bayes (Gaussian, Multinomial, Bernoulli) khác nhau ra sao? Khi nào nên sử dụng từng loại?

- Ba loại mô hình Naive Bayes chính là Gaussian, Multinomial, và Bernoulli, khác nhau chủ yếu ở cách ước tính xác suất khả năng $P(x_i|y)$, phụ thuộc vào bản chất của dữ liệu đặc trưng (x_i):

+ Gaussian Naive Bayes (Gaussian NB): Loại mô hình này được sử dụng khi các đặc trưng (x_i)là liên tục (continuous), ví dụ như chiều cao, cân nặng, hoặc điểm số. Gaussian NB giả định rằng các giá trị của mỗi đặc trưng tuân theo Phân bố Chuẩn (Gaussian/Normal Distribution) khi được nhóm theo lớp (y) Do đó, nó ước tính xác suất bằng cách sử dụng trung bình (μ_y) và phương sai $(\sigma_y 2)$ của đặc trưng đó trong tập huấn luyện của mỗi lớp. Bạn nên sử dụng Gaussian NB khi làm việc với dữ liệu giá trị thực (real-valued data).

+ Multinomial Naive Bayes (Multinomial NB): Mô hình này thích hợp cho các đặc trưng biểu thị số đếm rời rạc (discrete counts) hoặc tần suất, thường là các số nguyên không âm. Multinomial NB tính toán xác suất dựa trên tần suất xuất hiện của đặc trưng (x_i) trong các mẫu thuộc lớp y. Đây là mô hình Naive Bayes phổ biến nhất được sử dụng trong Phân loại Văn bản (Text Classification) với

mô hình Bag-of-Words, trong đó các đặc trưng là số lần một từ (word count) xuất hiện hoặc giá trị TF-IDF.

+ Bernoulli Naive Bayes (Bernoulli NB): Bernoulli NB được thiết kế đặc biệt cho các đặc trưng nhị phân (binary), chỉ có hai giá trị là 0 hoặc 1. Nó tính toán xác suất một đặc trưng có mặt (1) hay không có mặt (0) trong một lớp y. Mô hình này hữu ích khi bạn chỉ quan tâm đến sự hiện diện (presence) của một đặc trưng thay vì tần suất của nó (như trong Multinomial NB). Bernoulli NB cũng có thể được áp dụng cho phân loại văn bản bằng cách sử dụng một vector đặc trưng nhị phân, trong đó 1 biểu thị từ có xuất hiện, và 0 biểu thị từ không xuất hiện trong tài liệu.

- Tóm lại, việc lựa chọn mô hình Naive Bayes phụ thuộc hoàn toàn vào định dạng dữ liệu đầu vào của bạn.

3) Tại sao Naive Bayes được gọi là "ngây thơ"? Giả định về tính độc lập của các đặc trưng ảnh hưởng như thế nào đến hiệu suất của mô hình?

- Naive Bayes gọi là "ngây thơ" vì nó giả định rất mạnh mẽ và thường sai trong thực tế: tất cả các đặc trưng độc lập có điều kiện với nhau khi biết lớp y. Ví dụ, trong email spam, từ "tiền" và "giải thưởng" thường xuất hiện cùng nhau, không độc lập.

- Ảnh hưởng đến hiệu suất:

+ Hiệu suất phân loại: Mặc dù giả định độc lập hiếm khi đúng, Naive Bayes vẫn thường cho kết quả phân loại tốt, đặc biệt trong phân loại văn bản, vì ta chỉ cần so sánh thứ tự xác suất hậu nghiệm giữa các lớp.

+ Độ chính xác xác suất: Ước tính xác suất hậu nghiệm thường không chính xác do bỏ qua mối quan hệ giữa các đặc trưng.

+ Tính đơn giản và tốc độ: Giúp tính toán và huấn luyện nhanh chóng, hiệu quả với dữ liệu lớn và nhiều đặc trưng.

4) Ưu điểm và hạn chế của Naive Bayes so với các thuật toán phân loại khác như SVM hoặc Random Forest là gì?

Yếu Tố	Naive Bayes	SVM	Random Forest
Tốc độ huấn luyện	Rất nhanh, tính toán thống kê và tần suất đơn giản	Chậm hơn, tối ưu hóa phức tạp	Chậm hơn Naive Bayes
Khả năng mở rộng	Rất tốt với dữ liệu lớn và đặc trưng cao	Kém hơn khi số lượng mẫu tăng	Tốt nhưng không bằng Naive Bayes
Hiệu suất phân loại	Tốt, đặc biệt phân loại văn bản	Rất tốt, đặc biệt với dữ liệu phi tuyến	Rất tốt, ít bị overfitting, xử lý phi tuyến
Giả định	Giả định độc lập giữa các đặc trưng	Ít giả định, tìm siêu phẳng tối ưu	Không giả định về phân phối hay độc lập đặc trưng
Xử lý đặc trưng	Yêu cầu đặc trưng độc lập	Nhạy cảm với chuẩn hóa và kernel	Tự động xử lý đặc trưng liên tục và phân loại

- Tóm lại: Naive Bayes nổi bật về tốc độ, đơn giản và khả năng mở rộng, thích hợp cho phân loại văn bản và hệ thống cần xử lý nhanh. SVM và Random Forest thường cho độ chính xác cao hơn trong các bài toán phức tạp.

5) Viết đoạn code mẫu bằng Python (sử dụng Scikit-learn) để xây dựng một mô hình Naive Bayes (ví dụ: Gaussian Naive Bayes) không? Hãy mô tả các bước thực hiện

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

- Bước 1: Chuẩn bị dữ liệu (dữ liệu liên tục)

```
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3,
2]])
y = np.array([0, 0, 0, 1, 1, 1])  # Nhãn lớp
```

- Bước 2: Chia dữ liệu thành tập huấn luyện và kiểm tra

```
X_train, X_test, y_train, y_test
= train_test_split(X, y, test_size=0.33, random_state=42)
```

- Bước 3: Khởi tạo mô hình Gaussian Naive Bayes và huấn luyện mô hình

```
# Khởi tạo mô hình Gaussian Naive Bayes
gnb = GaussianNB()
# Huấn luyện mô hình
gnb.fit(X_train, y_train)
```

- Bước 4: Dự đoán trên tập kiểm tra, đánh giá hiệu suất và dự đoán mẫu mới

```
# Dự đoán trên tập kiểm tra
y_pred = gnb.predict(X_test)
# Đánh giá hiệu suất
accuracy = accuracy_score(y_test, y_pred)
print(f"Dữ liệu huấn luyện:\n{X_train}
\nNhãn huấn luyện:\n{y_train}")
print("-" * 30)
print(f"Kết quả dự đoán: {y_pred}")
print(f"Nhãn thực tế:    {y_test}")
print(f"Độ chính xác (Accuracy): {accuracy:.2f}")
# Dự đoán mẫu mới
new_sample = np.array([[-0.5, 1.5]])
prediction = gnb.predict(new_sample)
print(f"\nDự đoán cho mẫu mới {new_sample[0]}: Lớp {prediction[0]}")
```

6) Làm thế nào để xử lý dữ liệu phân loại (categorical data) trước khi áp dụng Multinomial Naive Bayes trong Python?

- Multinomial Naive Bayes yêu cầu đặc trưng đầu vào là số đếm hoặc giá trị số không âm, không thể

xử lý trực tiếp đặc trưng dạng chuỗi.

- Cách xử lý:

+ One-Hot Encoding: Chuyển mỗi danh mục thành một cột nhị phân (0 hoặc 1) thể hiện sự có mặt của danh mục đó.

+ Label Encoding: Gán số nguyên cho từng danh mục (không phù hợp với MNB vì tạo ra thứ tự giả).

- Ví dụ sử dụng pandas:

```
import pandas as pd
from sklearn.naive_bayes import MultinomialNB
data = {'Kích thước': [1, 2, 3, 1, 2],
        'Màu': ['Đỏ', 'Xanh', 'Đỏ', 'Vàng', 'Xanh'],
        'Nhãn': [0, 1, 0, 1, 1]}
df = pd.DataFrame(data)
# One-Hot Encoding cho cột 'Màu'
df_encoded = pd.get_dummies(df, columns=['Màu'], drop_first=True)
X = df_encoded.drop('Nhãn', axis=1)
y = df_encoded['Nhãn']
mnb = MultinomialNB()
mnb.fit(X, y)
```

7) Naive Bayes thường được sử dụng trong phân loại văn bản (text classification). Bạn có thể giải thích cách triển khai Naive Bayes cho bài toán này không?

- Multinomial Naive Bayes (MNB) thường dùng cho phân loại văn bản dựa trên mô hình Bag-of-Words.

- Các bước triển khai:

+ Tiền xử lý văn bản:

- Chuyển chữ hoa thành chữ thường.
- Loại bỏ dấu câu, số, từ dừng (stop words).
- Stemming hoặc Lemmatization (tuỳ chọn).

+ Trích xuất đặc trưng:

- Sử dụng CountVectorizer hoặc TfidfVectorizer để chuyển văn bản thành ma trận số đếm hoặc TF-IDF.

+ Huấn luyện mô hình:

- Dùng MultinomialNB huấn luyện trên ma trận đặc trưng và nhãn.

+ Dự đoán:

- Chuyển văn bản mới thành vector đặc trưng tương ứng.
- Dự đoán lớp bằng mô hình đã huấn luyện.

- Lưu ý: Áp dụng kỹ thuật làm mịn Laplace (thông số alpha trong MultinomialNB) để tránh xác suất bằng 0 khi gặp từ mới.

2.3.2. Bài làm mẫu

Bài toán 1: Xây dựng mô hình dữ liệu bằng giải thuật Bayes ngây thơ trên tập dữ liệu lấy tại https://www.kaggle.com/code/zabihullah18/email-spam-detection

Nhiệm vụ 1: Phân loại sử dụng Naïve Bays

1. Import thư viện và nạp dữ liệu vào notebook

```
#Importing the Necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix,
                                          classification_report
data = pd.read_csv('spam.csv', encoding='latin-1')
#display the first 5 rows
data.head()
```

2. Xử lý dữ liệu trước khi xây dựng mô hình từ dữ liệu

```
# Drop the columns with NaN values
data = data.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],
```

```
axis=1)
# Rename columns for clarity:
data.columns = ['label', 'text']
# Separate features (X) and target labels (y)
X = data.drop('label', axis=1)
y = data['label']


# Split the data into training and testing sets (80% training, 20%
testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

3. Xây dựng vector hóa nội dung HAM | SPAM của tập train và tập test

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()


# Fit and transform the training data (X_train)
X_train_vectorized = vectorizer.fit_transform(X_train['text'])
# Transform the test data (X_test)
X_test_vectorized = vectorizer.transform(X_test['text'])
```

4. Xây dựng mô hình Naïve Bayes

```
from sklearn.naive_bayes import MultinomialNB

classifier = MultinomialNB()

classifier.fit(X_train_vectorized, y_train)
```

5. Đánh giá hiệu quả của mô hình

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
# Make predictions on the test data

y_pred = classifier.predict(X_test_vectorized)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

classification_rep = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("Confusion Matrix:")

print(conf_matrix)

print("Classification Report:")
```

```
print(classification_rep)
```

Kết quả thực hiện

```
Accuracy: 0.98
Confusion Matrix:
[[963   2]
 [ 16 134]]
Classification Report:
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99       965
        spam       0.99      0.89      0.94       150

    accuracy                           0.98      1115
   macro avg       0.98      0.95      0.96      1115
weighted avg       0.98      0.98      0.98      1115
```

Hình 2.7 - Báo cáo đánh giá về hiệu quả của mô hình

2.3.3. Bài tập thực hành 1

Xây dựng mô hình Naïve ngây thơ trên tập dữ liệu hành vi của khách hàng lấy tại https://www.kaggle.com/code/arezalo/customer-behaviour-prediction-naive-bayes

- Phần bài làm nằm trong thư mục có tên là "Code" và tên là "Naive_customer_behavior.ipynb".

2.3.4. Bài tập thực hành 2

Xây dựng mô hình Naïve ngây thơ trên tập dữ liệu mushroom. Dữ liệu lấy tại https://www.kaggle.com/datasets/uciml/mushroom-classification/data

- Phần bài làm nằm trong thư mục có tên là "Code" và tên là "Naive_mushroom_classification.ipynb".

D. TÓM TẮT THỰC HÀNH

Với 3 phần là phân loại với giải thuật cây quyết định và rừng cây, giải thuật support vector machine (SVM) và giải thuật Bayes ngây thơ được trình bày trong chương đã phần nào giúp sinh viên có thể tiến hình triển phân loại dữ liệu trên một số tập dữ liệu cơ bản.

Với việc thực hành giải quyết các bài tập trên lớp và các bài tập ở nhà giúp sinh viên hiểu rõ hơn các khái niệm lý thuyết đã học và áp dụng tốt vào việc giải quyết các **bài toán phân loại** trong thực tế.