TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

**BÁO CÁO**

**THỰC HÀNH THUẬT TOÁN PHÂN LỚP**

Môn học: **Nhập môn máy học**

Giảng viên hướng dẫn: **Đỗ Như Tài**

Nhóm lớp học: **Nhóm 1 (thứ 7, tiết 6 đến tiết 10)**

Sinh viên thực hiện: **Lê Khánh Hoàng MSSV: 3122410125**

**TP.HCM, ngày 30 tháng 10 năm 2025**

MỤC LỤC

[GIẢI THUẬT 1: CÂY QUYẾT ĐỊNH VÀ RỪNG CÂY 3](#_Toc212729586)

[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, Access) là gì? 3](#_Toc212729587)

[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. 4](#_Toc212729588)

[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? 6](#_Toc212729589)

[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? 9](#_Toc212729590)

[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? 11](#_Toc212729591)

[Viết đoạn code mẫu bằng Python (sử dụng Scikit-learn) để xây dựng một mô hình cây quyết định ? Hãy mô tả các bước thực hiện 13](#_Toc212729592)

[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)? 15](#_Toc212729593)

[Làm thế nào để đánh giá tầm quan trọng của các đặc trưng (feature importance) trong Random Forest bằng Python? 17](#_Toc212729594)

[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 19](#_Toc212729595)

[GIẢI THUẬT 2: SUPPORT VECTOR MACHINE (SVM) 22](#_Toc212729596)

[Giải thuật Support Vector Machine hoạt động như thế nào? Hãy giải thích khái niệm về ranh giới phân tách (hyperplane) và lề (margin) 22](#_Toc212729597)

[Các vector hỗ trợ (support vectors) có vai trò gì trong SVM? Tại sao chúng quan trọng? 24](#_Toc212729598)

[Sự khác biệt giữa SVM với lề cứng (hard margin) và lề mềm (soft margin) là gì? Khi nào nên sử dụng lề mềm? 26](#_Toc212729599)

[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 27](#_Toc212729600)

[Tham số C trong SVM có ý nghĩa gì? Nó ảnh hưởng như thế nào đến hiệu suất của mô hình? 29](#_Toc212729601)

[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 30](#_Toc212729602)

[Hàm nào trong Scikit-learn để chuẩn hóa dữ liệu (scaling) trước khi áp dụng SVM? Tại sao bước này quan trọng? 33](#_Toc212729603)

[GIẢI THUẬT 3: BAYES NGÂY THƠ (NAÏVE BAYES) 34](#_Toc212729604)

[Giải thuật Naive Bayes hoạt động như thế nào? Hãy giải thích định lý Bayes và giả định "ngây thơ" trong thuật toán này? 34](#_Toc212729605)

[Các loại mô hình Naive Bayes (Gaussian, Multinomial, Bernoulli) khác nhau ra sao? Khi nào nên sử dụng từng loại? 36](#_Toc212729606)

[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? 37](#_Toc212729607)

[Ưu điểm và hạn chế của Naive Bayes so với các thuật toán phân loại khác như SVM hoặc Random Forest là gì? 39](#_Toc212729608)

[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 40](#_Toc212729609)

[Làm thế nào để xử lý dữ liệu phân loại (categorical data) trước khi áp dụng Multinomial Naive Bayes trong Python? 43](#_Toc212729610)

[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? 44](#_Toc212729611)

# GIẢI THUẬT 1: CÂY QUYẾT ĐỊNH VÀ RỪNG CÂY

## 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, Access) là gì?

**Quy trình CRISP-DM (Cross Industry Standard Process for Data Mining)**

CRISP-DM là một quy trình tiêu chuẩn ngành cho khai phá dữ liệu (data mining), được phát triển bởi một liên minh các công ty và tổ chức vào những năm 1990. Đây là mô hình phổ biến nhất trong lĩnh vực phân tích dữ liệu, nhấn mạnh vào cách tiếp cận lặp lại (iterative) và tích hợp chặt chẽ giữa kinh doanh và kỹ thuật. Quy trình được chia thành 6 giai đoạn chính, thường được biểu diễn dưới dạng vòng lặp để cho phép quay lại các bước trước nếu cần.

Dưới đây là các giai đoạn của CRISP-DM:

| **Giai đoạn** | **Mô tả** |
| --- | --- |
| **1. Hiểu biết về kinh doanh (Business Understanding)** | Xác định mục tiêu kinh doanh, đánh giá tình hình hiện tại, xác định nguồn lực cần thiết và lập kế hoạch dự án. Đây là nền tảng để đảm bảo dự án phù hợp với nhu cầu thực tế. |
| **2. Hiểu biết về dữ liệu (Data Understanding)** | Thu thập dữ liệu ban đầu, mô tả dữ liệu, khám phá dữ liệu (EDA - Exploratory Data Analysis), và xác định vấn đề chất lượng dữ liệu. |
| **3. Chuẩn bị dữ liệu (Data Preparation)** | Làm sạch, chọn lọc, biến đổi và tích hợp dữ liệu để tạo tập dữ liệu cuối cùng phù hợp cho mô hình hóa. Đây thường là giai đoạn tốn thời gian nhất. |
| **4. Mô hình hóa (Modeling)** | Chọn kỹ thuật mô hình hóa, xây dựng mô hình sơ bộ, đánh giá và điều chỉnh tham số. Các phương pháp như phân loại, hồi quy, phân cụm được áp dụng. |
| **5. Đánh giá (Evaluation)** | Đánh giá mô hình dựa trên mục tiêu kinh doanh, xác định những điểm cần cải thiện, và quyết định xem mô hình có sẵn sàng triển khai không. |
| **6. Triển khai (Deployment)** | Lập kế hoạch triển khai, giám sát và bảo trì, cũng như sản xuất báo cáo cuối cùng. Có thể bao gồm đào tạo người dùng và tích hợp vào hệ thống kinh doanh. |

CRISP-DM linh hoạt, không tuyến tính, và có thể lặp lại giữa các giai đoạn để tinh chỉnh kết quả.

**Quy trình SEMMA (Sample, Explore, Modify, Model, Assess)**

SEMMA là quy trình khai phá dữ liệu do công ty SAS Institute phát triển, tập trung vào các hoạt động xử lý dữ liệu và mô hình hóa. Không giống CRISP-DM (nhấn mạnh kinh doanh), SEMMA mang tính kỹ thuật hơn, thường được sử dụng trong môi trường phần mềm SAS. Quy trình gồm 5 giai đoạn chính, chủ yếu tập trung vào dữ liệu và mô hình, với cách tiếp cận tuyến tính hơn nhưng vẫn hỗ trợ lặp lại.

Dưới đây là các giai đoạn của SEMMA:

| **Giai đoạn** | **Mô tả** |
| --- | --- |
| **1. Lấy mẫu (Sample)** | Chọn một tập mẫu đại diện từ dữ liệu lớn để giảm kích thước và tăng tốc độ xử lý, đảm bảo mẫu phản ánh đúng dữ liệu gốc. |
| **2. Khám phá (Explore)** | Phân tích dữ liệu mẫu để hiểu phân phối, mối quan hệ, và phát hiện bất thường (sử dụng biểu đồ, thống kê mô tả). |
| **3. Sửa đổi (Modify)** | Biến đổi dữ liệu: làm sạch, tạo biến mới, giảm chiều, xử lý giá trị thiếu, và chuẩn hóa để chuẩn bị cho mô hình hóa. |
| **4. Mô hình hóa (Model)** | Xây dựng và huấn luyện các mô hình dự đoán (như hồi quy, cây quyết định, mạng nơ-ron) dựa trên dữ liệu đã sửa đổi. |
| **5. Đánh giá (Assess)** | Đánh giá hiệu suất mô hình bằng các chỉ số (accuracy, precision, recall), so sánh các mô hình, và quyết định triển khai. |

SEMMA thường được sử dụng trong các dự án tập trung vào phân tích dữ liệu lớn với công cụ SAS, và nó có thể được mở rộng bằng cách thêm giai đoạn kinh doanh trước hoặc sau.

## 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 là một mô hình học máy phổ biến, được sử dụng cho cả bài toán phân loại (classification) và hồi quy (regression). Nó mô phỏng quá trình ra quyết định của con người bằng cách chia dữ liệu thành các nhánh dựa trên các quy tắc đơn giản, dễ hiểu. Cây được xây dựng từ dữ liệu huấn luyện bằng cách chọn các đặc trưng (features) tốt nhất để phân chia dữ liệu tại mỗi bước, nhằm giảm thiểu độ không chắc chắn (impurity) như entropy hoặc Gini index.

Quá trình hoạt động tổng quát:

1. **Xây dựng cây**: Sử dụng thuật toán như ID3, C4.5, hoặc CART để chọn đặc trưng phân chia tốt nhất tại mỗi nút, lặp lại đến khi đạt điều kiện dừng (ví dụ: độ sâu tối đa hoặc dữ liệu thuần nhất).
2. **Dự đoán**: Với dữ liệu mới, bắt đầu từ nút gốc và đi theo các nhánh dựa trên giá trị đặc trưng, đến nút lá để nhận kết quả.

Dưới đây là giải thích chi tiết về **các thành phần chính** và **cách đưa ra dự đoán**.

**Các Thành Phần Chính**

Cây quyết định có cấu trúc dạng cây phân cấp, với các phần sau:

| **Thành phần** | **Mô tả** |
| --- | --- |
| **Nút gốc (Root Node)** | Nút đầu tiên và cao nhất của cây, đại diện cho toàn bộ tập dữ liệu huấn luyện. Nó chứa câu hỏi phân chia đầu tiên dựa trên đặc trưng quan trọng nhất (ví dụ: "Tuổi > 30?"). Từ đây, cây phân nhánh ra các nút con. Nút gốc quyết định hướng đi ban đầu cho mọi dự đoán. |
| **Nhánh (Branches)** | Các đường nối giữa các nút, đại diện cho kết quả của câu hỏi phân chia. Mỗi nhánh tương ứng với một giá trị hoặc khoảng giá trị của đặc trưng (ví dụ: nhánh "Có" cho tuổi > 30, nhánh "Không" cho tuổi ≤ 30). Nhánh thể hiện luồng quyết định dẫn đến nút con. |
| **Nút lá (Leaf Node)** | Các nút cuối cùng của cây (không có nhánh con), đại diện cho kết quả dự đoán cuối cùng. Trong phân loại, nút lá chứa lớp (class) phổ biến nhất (ví dụ: "Mua" hoặc "Không mua"). Trong hồi quy, nó chứa giá trị trung bình hoặc trung vị của dữ liệu tại nút đó. |

Ví dụ minh họa: Giả sử cây dự đoán khách hàng có mua sản phẩm không dựa trên tuổi và thu nhập.

* **Nút gốc**: "Thu nhập > 50 triệu?"
  + Nhánh "Có" → Nút con: "Tuổi > 25?" → Nút lá: "Mua" (nếu có).
  + Nhánh "Không" → Nút lá: "Không mua".

**Cách Cây Đưa Ra Dự Đoán**

Quá trình dự đoán diễn ra theo hướng từ trên xuống dưới (top-down), rất nhanh và dễ theo dõi:

1. **Bắt đầu từ nút gốc**: Lấy dữ liệu mới (một mẫu) và kiểm tra điều kiện tại nút gốc dựa trên giá trị đặc trưng tương ứng.
2. **Theo nhánh phù hợp**: Dựa trên câu trả lời (ví dụ: nếu thu nhập > 50 triệu, đi theo nhánh "Có").
3. **Lặp lại đến nút lá**: Tiếp tục phân nhánh qua các nút nội bộ (internal nodes) cho đến khi đạt nút lá.
4. **Trả kết quả**: Giá trị tại nút lá là dự đoán cuối cùng. Nếu là phân loại, chọn lớp đa số; nếu hồi quy, lấy giá trị số.

**Ưu điểm**: Dễ giải thích, không cần chuẩn hóa dữ liệu, xử lý cả dữ liệu số và phân loại.

**Nhược điểm**: Dễ overfitting (cây quá sâu), có thể không ổn định với dữ liệu nhiễu – thường dùng ensemble như Random Forest để khắc phục.

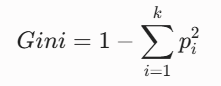
## Các tiêu chí phân tách (splitting criteria) như Gini Index, Entropy, hay Information Gain được sử dụng trong cây quyết định là gì? Chúng khác nhau ra sao?

Trong cây quyết định, tiêu chí phân tách (splitting criteria) là các hàm đo lường để đánh giá chất lượng của một điểm phân chia dữ liệu tại mỗi nút. Mục tiêu là chọn đặc trưng (feature) và ngưỡng phân chia sao cho giảm thiểu độ "không tinh khiết" (impurity) hoặc độ không chắc chắn (uncertainty) của dữ liệu con, giúp cây phân loại hoặc dự đoán chính xác hơn. Các tiêu chí phổ biến bao gồm Gini Index, Entropy, và Information Gain. Chúng được sử dụng trong các thuật toán như CART (cho Gini), ID3/C4.5 (cho Entropy và Information Gain).

Dưới đây là giải thích chi tiết từng tiêu chí, công thức cơ bản, và cách chúng được áp dụng.

**1. Gini Index (Chỉ số Gini)**

* **Mô tả**: Gini Index đo độ "tinh khiết" của một nút bằng cách tính xác suất chọn sai một mẫu ngẫu nhiên từ nút đó. Nó ưu tiên các phân chia làm cho dữ liệu con đồng nhất hơn (một lớp chiếm ưu thế).
* **Công thức** (cho nút với các lớp i = 1 đến k):



* : Tỷ lệ mẫu thuộc lớp i trong nút.
* Gini = 0 nếu nút thuần nhất (tất cả mẫu cùng lớp); Gini = 0.5 nếu phân bố đều (ví dụ: 50-50% hai lớp).

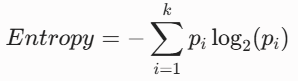
**Cách sử dụng**: Tại mỗi nút, tính Gini cho nút hiện tại, sau đó tính Gini trung bình có trọng số cho các nút con sau phân chia. Chọn phân chia giảm Gini nhiều nhất (Gini gain cao nhất).

**Ưu điểm**: Tính toán nhanh (không cần log), phù hợp với dữ liệu lớn.

**Nhược điểm**: Nhạy cảm với phân bố lớp không cân bằng.

**2. Entropy (Entropy)**

* **Mô tả**: Entropy đo độ "không chắc chắn" hoặc thông tin ngẫu nhiên trong nút, dựa trên lý thuyết thông tin (information theory). Nó cao khi các lớp phân bố đều, thấp khi một lớp chiếm ưu thế.
* **Công thức** (cho nút với các lớp i = 1 đến k):



* : Tỷ lệ mẫu thuộc lớp i.
* Entropy = 0 nếu thuần nhất; Entropy = 1 nếu hai lớp phân bố đều (50-50%).

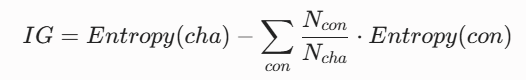
**Cách sử dụng**: Tương tự Gini, tính Entropy cho nút con sau phân chia và chọn phân chia giảm Entropy trung bình có trọng số nhiều nhất.

**Ưu điểm**: Dựa trên lý thuyết vững chắc, dễ mở rộng cho đa lớp.

**Nhược điểm**: Tính toán chậm hơn do hàm log, có thể bị ảnh hưởng bởi giá trị p\_i = 0 (thường dùng log(1 + p\_i) để tránh).

**3. Information Gain (Lợi ích Thông tin)**

* **Mô tả**: Không phải là impurity trực tiếp, mà là sự giảm Entropy sau khi phân chia. Nó đo "lượng thông tin" mà một đặc trưng cung cấp để phân biệt các lớp.
* **Công thức**:



* + Entropy(cha): Entropy của nút hiện tại.
  + : Trọng số của nút con.
  + IG = 0 nếu phân chia không cải thiện; IG cao nếu phân chia làm dữ liệu con tinh khiết hơn.
* **Cách sử dụng**: Chọn đặc trưng có IG cao nhất để phân chia (dùng trong ID3). Có biến thể như Gain Ratio để khắc phục bias với đặc trưng đa giá trị.
* **Ưu điểm**: Trực quan, ưu tiên đặc trưng giảm uncertainty mạnh.
* **Nhược điểm**: Có thể ưu tiên đặc trưng có nhiều giá trị (nhiều nhánh), dẫn đến overfitting – dùng Gain Ratio để điều chỉnh.

**Sự Khác Biệt Giữa Các Tiêu Chí**

Dưới đây là bảng so sánh để làm rõ sự khác nhau:

| **Tiêu chí** | **Mục đích chính** | **Phạm vi giá trị** | **Ưu tiên khi chọn phân chia** | **Thuật toán phổ biến** | **Đặc điểm nổi bật** |
| --- | --- | --- | --- | --- | --- |
| **Gini Index** | Đo impurity (không tinh khiết) | 0 (tinh khiết) đến ~0.5 (phân bố đều) | Giảm Gini (thấp hơn tốt hơn) | CART, Random Forest | Nhanh, không dùng log; ít nhạy cảm với đa lớp. |
| **Entropy** | Đo uncertainty (không chắc chắn) | 0 (tinh khiết) đến log2(k) (phân bố đều, k lớp) | Giảm Entropy (thấp hơn tốt hơn) | ID3, C4.5 (kết hợp IG) | Dựa trên thông tin; chậm hơn Gini nhưng chính xác hơn với đa lớp. |
| **Information Gain** | Đo sự cải thiện sau phân chia (dựa trên Entropy) | ≥ 0 (cao hơn tốt hơn) | Tăng IG (cao hơn tốt hơn) | ID3, C4.5 | Không phải impurity trực tiếp; dễ bias với đặc trưng đa giá trị. |

**Gini vs. Entropy**: Cả hai đều đo impurity nội tại của nút, nhưng Gini đơn giản hơn (dùng bình phương thay vì log), dẫn đến tương quan cao (~90% trường hợp chọn cùng đặc trưng). Gini thường dùng cho phân loại nhị phân, Entropy cho đa lớp.

**Information Gain vs. hai cái kia**: IG không đo impurity nút đơn lẻ mà đo "lợi ích" của toàn bộ phân chia, nên nó kết hợp Entropy để đánh giá toàn cục. Kết quả: IG thường dẫn đến cây nông hơn nhưng có thể bias.

## 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 gì?**

Random Forest (Rừng Cây Ngẫu Nhiên) là một thuật toán học máy thuộc nhóm **ensemble learning** (học tập kết hợp), được phát triển bởi Leo Breiman vào năm 2001. Nó hoạt động bằng cách xây dựng **nhiều cây quyết định (decision trees)** độc lập và kết hợp kết quả của chúng để đưa ra dự đoán cuối cùng. Mỗi cây được huấn luyện trên một tập dữ liệu con ngẫu nhiên (qua kỹ thuật **bagging - bootstrap aggregating**), và chỉ sử dụng một tập con các đặc trưng (features) ngẫu nhiên để phân tách.

* **Ứng dụng chính**: Phân loại (classification), hồi quy (regression), và đánh giá tầm quan trọng của đặc trưng.
* **Cách hoạt động tổng quát**:
  1. Tạo nhiều tập dữ liệu huấn luyện con bằng cách lấy mẫu ngẫu nhiên có thay thế (bootstrap) từ dữ liệu gốc.
  2. Xây dựng một cây quyết định cho mỗi tập con, với phân tách chỉ dựa trên một số đặc trưng ngẫu nhiên (thường là √số đặc trưng tổng).
  3. Với phân loại: Dự đoán bằng **đa số phiếu** (majority voting) từ tất cả cây. Với hồi quy: Lấy **trung bình** giá trị từ các cây.
* **Ưu điểm**: Độ chính xác cao, chống overfitting tốt, dễ song song hóa.

**Sự Khác Biệt Giữa Random Forest Và Cây Quyết Định Đơn Lẻ**

Cây quyết định đơn lẻ (Decision Tree) là nền tảng của Random Forest, nhưng Random Forest khắc phục các hạn chế của nó bằng cách sử dụng nhiều cây. Dưới đây là bảng so sánh chính:

| **Tiêu chí** | **Cây Quyết Định Đơn Lẻ (Decision Tree)** | **Random Forest (Rừng Cây)** |
| --- | --- | --- |
| **Cấu trúc** | Một cây duy nhất, phân cấp từ nút gốc đến nút lá. | Tập hợp hàng trăm/thousands cây độc lập, kết hợp kết quả. |
| **Cách xây dựng** | Sử dụng toàn bộ dữ liệu huấn luyện; phân tách dựa trên tiêu chí tốt nhất (Gini, Entropy) từ tất cả đặc trưng. | Sử dụng bagging (mẫu ngẫu nhiên có thay thế) và random feature selection (chỉ subset đặc trưng tại mỗi nút). |
| **Dự đoán** | Theo đường đi từ gốc đến lá dựa trên dữ liệu mới. | Kết hợp (voting cho phân loại, trung bình cho hồi quy) từ tất cả cây. |
| **Overfitting** | Dễ overfitting (cây quá sâu, học nhiễu). | Giảm overfitting nhờ đa dạng hóa (diversity) giữa các cây. |
| **Ổn định** | Nhạy cảm với thay đổi nhỏ trong dữ liệu (variance cao). | Ổn định hơn, variance thấp nhờ ensemble. |
| **Giải thích** | Dễ hiểu và trực quan (có thể vẽ cây). | Ít trực quan hơn, nhưng có thể đánh giá feature importance trung bình. |
| **Tốc độ** | Nhanh để huấn luyện và dự đoán. | Chậm hơn do nhiều cây, nhưng dễ song song hóa (parallel). |

Tóm lại, Decision Tree đơn giản và dễ triển khai, nhưng dễ bị overfitting và không ổn định. Random Forest "dân chủ hóa" quyết định bằng cách lấy ý kiến từ "rừng cây" để giảm lỗi.

**Tại Sao Random Forest Thường Có Hiệu Suất Tốt Hơn Trong Phân Loại?**

Random Forest vượt trội hơn cây quyết định đơn lẻ trong các bài toán phân loại nhờ **cân bằng bias-variance tradeoff** và các cơ chế chống nhiễu:

1. **Giảm Variance (Độ Biến Thiên)**:
   * Cây đơn lẻ có variance cao vì một thay đổi nhỏ trong dữ liệu có thể thay đổi toàn bộ cấu trúc cây. Random Forest trung bình hóa kết quả từ nhiều cây (uncorrelated nhờ bagging và random features), dẫn đến variance thấp hơn. Kết quả: Mô hình ổn định, ít bị ảnh hưởng bởi dữ liệu nhiễu hoặc outlier.
2. **Chống Overfitting**:
   * Bằng cách sử dụng bootstrap (mỗi cây chỉ thấy ~63% dữ liệu gốc), và random subset features, các cây trở nên đa dạng, tránh học thuộc lòng dữ liệu huấn luyện. Trong khi cây đơn lẻ có thể đạt accuracy 100% trên train nhưng kém trên test.
3. **Cải Thiện Accuracy**:
   * Trong phân loại, majority voting từ nhiều cây "lọc" lỗi của từng cây riêng lẻ. Nghiên cứu cho thấy Random Forest thường đạt accuracy cao hơn 5-10% so với cây đơn lẻ trên dataset thực tế (như Iris, Wine từ UCI ML Repository).
   * Nó xử lý tốt dữ liệu không cân bằng (imbalanced classes) nhờ out-of-bag (OOB) error estimation – một cách đánh giá mà không cần validation set riêng.
4. **Robust Với Đặc Trưng Không Liên Quan**:
   * Random feature selection giúp bỏ qua đặc trưng kém, tập trung vào những cái tốt, giảm bias từ đặc trưng nhiễu.

**Nhược điểm của Random Forest**: Tốn bộ nhớ và thời gian hơn (do nhiều cây), khó giải thích chi tiết từng quyết định. Trong thực tế, nó thường là baseline mạnh cho phân loại, và có thể kết hợp với boosting (như XGBoost) để tốt hơn nữa.

## 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?

**Ưu Điểm Và Hạn Chế Của Cây Quyết Định (Decision Tree)**

Cây quyết định là mô hình học máy cơ bản, dễ triển khai và giải thích. Dưới đây là bảng tóm tắt ưu nhược điểm chính:

| **Ưu Điểm** | **Hạn Chế** |
| --- | --- |
| **Dễ hiểu và giải thích**: Cấu trúc dạng cây giống quy trình ra quyết định của con người, dễ trực quan hóa và theo dõi lý do dự đoán. | **Dễ overfitting**: Cây có thể quá sâu, học thuộc lòng dữ liệu huấn luyện, dẫn đến hiệu suất kém trên dữ liệu mới. |
| **Không cần chuẩn hóa dữ liệu**: Xử lý tốt cả dữ liệu số (numerical) và phân loại (categorical) mà không yêu cầu scaling. | **Variance cao**: Nhạy cảm với thay đổi nhỏ trong dữ liệu, dẫn đến cây khác nhau đáng kể nếu dữ liệu thay đổi. |
| **Tốc độ nhanh**: Huấn luyện và dự đoán nhanh, đặc biệt với dữ liệu nhỏ đến trung bình. | **Bias với dữ liệu không tuyến tính**: Có thể tạo ra các ranh giới quyết định không mượt mà, kém hiệu quả với mối quan hệ phức tạp. |
| **Xử lý dữ liệu thiếu**: Có thể dễ dàng xử lý giá trị thiếu bằng cách phân nhánh riêng. | **Không ổn định**: Thay đổi thứ tự dữ liệu có thể thay đổi toàn bộ cấu trúc cây. |
| **Ít tham số cần điều chỉnh**: Chỉ cần kiểm soát độ sâu hoặc số nút lá. | **Kém với dữ liệu không cân bằng**: Có thể ưu tiên lớp đa số, bỏ qua lớp thiểu số. |

**Ưu Điểm Và Hạn Chế Của Random Forest**

Random Forest là phiên bản ensemble của cây quyết định, sử dụng nhiều cây để cải thiện độ chính xác. Bảng tóm tắt:

| **Ưu Điểm** | **Hạn Chế** |
| --- | --- |
| **Độ chính xác cao**: Kết hợp nhiều cây giúp giảm lỗi, thường vượt trội hơn cây đơn lẻ 5-10% trong phân loại. | **Tốn tài nguyên**: Yêu cầu bộ nhớ và CPU lớn do xây dựng hàng trăm cây, không phù hợp với thiết bị yếu. |
| **Chống overfitting tốt**: Bagging và random feature selection tạo đa dạng, giảm variance. | **Khó giải thích**: "Hộp đen" hơn cây đơn, khó theo dõi quyết định chi tiết của từng cây. |
| **Ổn định và robust**: Ít nhạy cảm với nhiễu, outlier hoặc dữ liệu thiếu. | **Thời gian huấn luyện dài**: Có thể mất vài phút đến giờ tùy kích thước dữ liệu và số cây. |
| **Đánh giá feature importance**: Tự động xếp hạng tầm quan trọng của đặc trưng dựa trên trung bình từ các cây. | **Dự đoán chậm hơn**: Phải tổng hợp kết quả từ tất cả cây, dù có thể song song hóa. |
| **Xử lý dữ liệu lớn và không cân bằng**: Hiệu quả với dataset lớn, có OOB error để đánh giá mà không cần validation set riêng. | **Không phù hợp với dữ liệu streaming**: Khó cập nhật mô hình theo thời gian thực mà không huấn luyện lại toàn bộ. |

**Trường Hợp Nào Thì Cây Quyết Định Có Thể Hoạt Động Kém Hiệu Quả?**

Cây quyết định đơn lẻ thường hoạt động kém trong các tình huống sau, nơi các hạn chế như overfitting và variance cao trở nên rõ rệt:

1. **Dữ liệu có nhiễu cao hoặc outlier**: Cây có thể tạo các nhánh riêng cho outlier, dẫn đến overfitting và accuracy thấp trên test set. Ví dụ: Trong dữ liệu y tế với nhiễu từ đo lường sai, cây đơn lẻ sẽ kém hơn Random Forest.
2. **Dữ liệu lớn và phức tạp**: Với hàng triệu mẫu, cây có thể trở nên quá sâu, tốn thời gian huấn luyện và dễ bias. Random Forest hoặc các ensemble khác sẽ ổn định hơn.
3. **Lớp không cân bằng (imbalanced classes)**: Nếu một lớp chiếm 90%, cây có thể ưu tiên lớp đa số, bỏ qua lớp thiểu số. Cần kỹ thuật như undersampling hoặc dùng ensemble để khắc phục.
4. **Mối quan hệ đặc trưng phức tạp**: Khi dữ liệu có tương tác phi tuyến tính cao (non-linear interactions) giữa các đặc trưng, cây đơn có thể không nắm bắt đầy đủ, dẫn đến underfitting ở một số trường hợp.
5. **Yêu cầu độ chính xác cao và ổn định**: Trong sản xuất (production), cây đơn dễ bị ảnh hưởng bởi thay đổi dữ liệu theo thời gian, trong khi Random Forest duy trì hiệu suất tốt hơn.

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

**Các Bước Thực Hiện Xây Dựng Mô Hình Cây Quyết Định Với Scikit-learn**

Dưới đây là hướng dẫn chi tiết các bước để xây dựng một mô hình cây quyết định (Decision Tree) đơn giản cho bài toán phân loại, sử dụng dataset Iris (một bộ dữ liệu cổ điển với 3 lớp hoa Iris dựa trên 4 đặc trưng). Tôi sử dụng thư viện Scikit-learn để minh họa. Quy trình này có thể áp dụng cho các dataset khác tương tự.

1. **Import các thư viện cần thiết**:
   * sklearn.datasets: Để tải dataset mẫu.
   * sklearn.model\_selection: Để chia dữ liệu thành tập huấn luyện và kiểm tra.
   * sklearn.tree: Để tạo mô hình DecisionTreeClassifier.
   * sklearn.metrics: Để đánh giá hiệu suất mô hình (ví dụ: accuracy).
2. **Tải và chuẩn bị dữ liệu**:
   * Tải dataset Iris (150 mẫu, 4 đặc trưng: chiều dài/rộng lá đài và cánh hoa).
   * Chia dữ liệu thành tập huấn luyện (80%) và kiểm tra (20%) bằng train\_test\_split để tránh overfitting.
3. **Tạo và huấn luyện mô hình**:
   * Khởi tạo mô hình DecisionTreeClassifier với các tham số mặc định (có thể điều chỉnh như max\_depth để kiểm soát độ sâu cây).
   * Huấn luyện mô hình bằng phương thức fit() trên tập huấn luyện.
4. **Dự đoán và đánh giá mô hình**:
   * Sử dụng predict() để dự đoán trên tập kiểm tra.
   * Tính toán độ chính xác bằng accuracy\_score và in kết quả.
5. **(Tùy chọn) Hiển thị cấu trúc cây**:
   * Sử dụng export\_text từ sklearn.tree để in cây dưới dạng văn bản (dễ đọc).

Quy trình này là cơ bản và lặp lại được. Trong thực tế, bạn có thể thêm bước khám phá dữ liệu (EDA), xử lý dữ liệu thiếu, hoặc điều chỉnh siêu tham số bằng GridSearchCV.

**Đoạn Code Mẫu (Python)**

Dưới đây là code hoàn chỉnh. Bạn có thể copy-paste và chạy trực tiếp trong môi trường Python có Scikit-learn (phiên bản >= 1.0).

# Bước 1: Import các thư viện cần thiết

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, export\_text

from sklearn.metrics import accuracy\_score

# Bước 2: Tải và chuẩn bị dữ liệu

iris = load\_iris() # Tải dataset Iris

X = iris.data # Đặc trưng (4 cột: sepal length, sepal width, petal length, petal width)

y = iris.target # Nhãn (0: setosa, 1: versicolor, 2: virginica)

# Chia dữ liệu: 80% train, 20% test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Bước 3: Tạo và huấn luyện mô hình

clf = DecisionTreeClassifier(random\_state=42) # Mô hình cây quyết định (phân loại)

clf.fit(X\_train, y\_train) # Huấn luyện trên tập train

# Bước 4: Dự đoán và đánh giá

y\_pred = clf.predict(X\_test) # Dự đoán trên tập test

accuracy = accuracy\_score(y\_test, y\_pred) # Tính độ chính xác

print(f"Độ chính xác trên tập kiểm tra: {accuracy:.2f}")

# Bước 5: Hiển thị cấu trúc cây (tùy chọn)

tree\_rules = export\_text(clf, feature\_names=iris.feature\_names)

print("\nCấu trúc cây quyết định (quy tắc phân chia):")

print(tree\_rules)

**Kết Quả Chạy Mẫu**

Khi chạy code này, bạn sẽ nhận được output tương tự:

* Độ chính xác: 1.00 (100%, vì Iris là dataset đơn giản).
* Cấu trúc cây: Một danh sách các quy tắc phân chia dựa trên đặc trưng (ví dụ: petal length <= 2.45 thì lớp 0, v.v.).

## 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)?

**Các Bước Triển Khai Mô Hình Random Forest Trong Python Với Scikit-learn**

Random Forest là một mô hình ensemble dựa trên nhiều cây quyết định, thường được triển khai qua lớp RandomForestClassifier (cho phân loại) hoặc RandomForestRegressor (cho hồi quy) trong thư viện Scikit-learn. Quy trình triển khai tương tự như cây quyết định đơn lẻ, nhưng bạn cần điều chỉnh các tham số để cân bằng giữa độ chính xác và tốc độ. Dưới đây là hướng dẫn chi tiết các bước, sử dụng dataset Iris làm ví dụ (phân loại 3 lớp hoa Iris dựa trên 4 đặc trưng).

1. **Import các thư viện cần thiết**:
   * sklearn.datasets: Tải dataset mẫu.
   * sklearn.model\_selection: Chia dữ liệu train/test.
   * sklearn.ensemble: Tạo mô hình RandomForestClassifier.
   * sklearn.metrics: Đánh giá (accuracy\_score).
2. **Tải và chuẩn bị dữ liệu**:
   * Tải dataset và chia thành tập huấn luyện (80%) và kiểm tra (20%) bằng train\_test\_split. Đặt random\_state để kết quả lặp lại được.
3. **Tạo và huấn luyện mô hình**:
   * Khởi tạo RandomForestClassifier với các tham số (sẽ giải thích chi tiết bên dưới).
   * Huấn luyện bằng fit() trên tập train.
4. **Dự đoán và đánh giá**:
   * Dự đoán bằng predict() trên tập test.
   * Tính độ chính xác bằng accuracy\_score. Có thể thêm confusion matrix hoặc feature importance.
5. **(Tùy chọn) Đánh giá thêm**:
   * Sử dụng feature\_importances\_ để xem tầm quan trọng của đặc trưng.
   * Điều chỉnh siêu tham số bằng GridSearchCV nếu cần tối ưu.

**Các Tham Số Thường Thiết Lập Trong Random Forest**

Random Forest có nhiều tham số để kiểm soát số lượng cây, độ phức tạp và ngẫu nhiên. Dưới đây là các tham số phổ biến nhất, với giá trị mặc định và lý do thiết lập (dựa trên kinh nghiệm thực tế, thường điều chỉnh qua cross-validation):

| **Tham Số** | **Mô Tả** | **Giá Trị Mặc Định** | **Giá Trị Thường Dùng** | **Lý Do Thiết Lập** |
| --- | --- | --- | --- | --- |
| **n\_estimators** | Số lượng cây quyết định trong rừng. Tăng giúp giảm variance nhưng tăng thời gian huấn luyện. | 100 | 100-500 (bắt đầu với 100) | Cân bằng tốc độ và độ chính xác; >500 cho dataset lớn. |
| **max\_depth** | Độ sâu tối đa của mỗi cây. Giới hạn để chống overfitting. | None (không giới hạn) | 5-20 (hoặc None cho dataset nhỏ) | Ngăn cây quá sâu; dùng None nếu dùng pruning. |
| **max\_features** | Số đặc trưng ngẫu nhiên xem xét tại mỗi nút phân chia. Giảm tương quan giữa cây. | 'sqrt' (√số đặc trưng) | 'sqrt', 'log2', hoặc 0.8 | 'sqrt' cho phân loại (tốt với nhiều đặc trưng); None dùng tất cả. |
| **min\_samples\_split** | Số mẫu tối thiểu để phân chia nút. Tăng để chống overfitting. | 2 | 2-10 | Tránh nhánh quá nhỏ; 5-10 cho dataset lớn. |
| **min\_samples\_leaf** | Số mẫu tối thiểu tại nút lá. Tăng để mượt mà hơn. | 1 | 1-5 | Giảm overfitting; 2 cho phân loại cân bằng. |
| **random\_state** | Seed cho ngẫu nhiên, đảm bảo kết quả lặp lại. | None | 42 (hoặc bất kỳ số nào) | Luôn đặt để debug và so sánh. |
| **bootstrap** | Có dùng bootstrap sampling không. | True | True | True để đa dạng hóa; False cho toàn bộ dữ liệu. |
| **oob\_score** | Sử dụng out-of-bag samples để đánh giá (không cần validation set). | False | True | True để ước lượng nhanh accuracy mà không chia dữ liệu. |

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

Đánh Giá Tầm Quan Trọng Của Các Đặc Trưng (Feature Importance) Trong Random Forest Bằng Python

Trong Random Forest, **tầm quan trọng của đặc trưng (feature importance)** là một chỉ số đo lường mức độ đóng góp của từng đặc trưng vào việc cải thiện độ chính xác của mô hình. Nó được tính dựa trên **sự giảm Gini impurity** (hoặc entropy, tùy tiêu chí) trung bình mà đặc trưng đó mang lại qua tất cả các nút phân chia trong các cây quyết định. Giá trị cao hơn nghĩa là đặc trưng đó quan trọng hơn (ví dụ: giúp phân loại tốt hơn). Đây là ưu điểm lớn của Random Forest, giúp xác định đặc trưng "hữu ích" để giảm chiều dữ liệu hoặc giải thích mô hình.

**Các Bước Thực Hiện**

1. **Import thư viện**: Sử dụng sklearn.ensemble.RandomForestClassifier (hoặc Regressor) và dataset mẫu nếu cần.
2. **Chuẩn bị dữ liệu**: Tải/chia dữ liệu huấn luyện.
3. **Huấn luyện mô hình**: Fit Random Forest trên dữ liệu.
4. **Truy cập và phân tích**: Sử dụng thuộc tính feature\_importances\_ để lấy mảng giá trị (tổng = 1.0). Sắp xếp và hiển thị.
5. **(Tùy chọn) Visualize**: Vẽ biểu đồ cột bằng Matplotlib để dễ quan sát.

**Code Mẫu (Sử Dụng Dataset Iris)**

Dưới đây là code đơn giản để tính và hiển thị feature importance. Bạn có thể chạy trực tiếp trong Python (cần Scikit-learn và Matplotlib).

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

import matplotlib.pyplot as plt

import numpy as np

# Bước 1-2: Tải và chuẩn bị dữ liệu

iris = 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)

# Bước 3: Huấn luyện mô hình

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

# Bước 4: Truy cập feature importance

feature\_importance = rf.feature\_importances\_

feature\_names = iris.feature\_names

# Sắp xếp theo tầm quan trọng giảm dần

importances = list(zip(feature\_names, feature\_importance))

importances.sort(key=lambda x: x[1], reverse=True)

print("Tầm quan trọng của các đặc trưng:")

for name, importance in importances:

print(f"{name}: {importance:.4f}")

# Bước 5: Visualize (tùy chọn)

indices = np.argsort(feature\_importance)[::-1] # Sắp xếp chỉ số

plt.figure(figsize=(8, 5))

plt.title("Feature Importance in Random Forest")

plt.bar(range(X.shape[1]), feature\_importance[indices])

plt.xticks(range(X.shape[1]), [feature\_names[i] for i in indices], rotation=45)

plt.ylabel("Importance")

plt.tight\_layout()

plt.show()

**Kết Quả Mẫu (Chạy Trên Dataset Iris)**

Khi chạy code trên, kết quả hiển thị sẽ tương tự như sau (dựa trên seed ngẫu nhiên cố định):

* **petal length (cm)**: 0.4400
* **petal width (cm)**: 0.4215
* **sepal length (cm)**: 0.1081
* **sepal width (cm)**: 0.0304

Biểu đồ cột sẽ cho thấy hai đặc trưng về cánh hoa (petal) quan trọng nhất, phù hợp với đặc tính sinh học của Iris.

## 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

**Điều Chỉnh Siêu Tham Số (Hyperparameter Tuning) Cho Cây Quyết Định Hoặc Random Forest**

Đây là một bước quan trọng để cải thiện hiệu suất mô hình, vì siêu tham số (như độ sâu cây, số lượng cây) không được học từ dữ liệu mà phải được chọn thủ công. Tuning giúp tránh overfitting/underfitting và tăng độ chính xác.

Dưới đây là mô tả chi tiết về cách sử dụng **GridSearchCV** và **RandomizedSearchCV** từ thư viện Scikit-learn. Cả hai đều dựa trên **cross-validation (CV)** để đánh giá mô hình trên tập dữ liệu (thường chia thành k folds, ví dụ k=5 hoặc 10) và chọn bộ tham số tốt nhất dựa trên metric như accuracy (cho phân loại) hoặc MSE (cho hồi quy). Quy trình chung:

1. Định nghĩa không gian tham số (parameter grid/distribution).
2. Khởi tạo estimator (mô hình DecisionTreeClassifier hoặc RandomForestClassifier).
3. Chạy search trên tập huấn luyện.
4. Lấy bộ tham số tốt nhất và refit mô hình.
5. Đánh giá trên tập kiểm tra.

**1. GridSearchCV: Tìm Kiếm Toàn Bộ Lưới Tham Số**

* **Mô tả**: Phương pháp kiệt sức (exhaustive search), thử **tất cả** các tổ hợp tham số trong lưới định nghĩa trước. Nó phù hợp với không gian tham số nhỏ (vài tham số, mỗi cái 3-5 giá trị) vì thời gian tính toán là O(số tổ hợp × số folds × thời gian huấn luyện).
* **Ưu điểm**: Đảm bảo tìm bộ tham số tối ưu toàn cục.
* **Nhược điểm**: Chậm với không gian lớn (ví dụ: 3 tham số × 5 giá trị = 125 tổ hợp, nhân với 5 folds = 625 lần huấn luyện).
* **Cách sử dụng**:
  + Import: from sklearn.model\_selection import GridSearchCV.
  + Định nghĩa lưới: param\_grid = {'max\_depth': [3, 5, 7], 'min\_samples\_split': [2, 5, 10]} (cho Decision Tree) hoặc thêm 'n\_estimators': [50, 100] cho Random Forest.
  + Khởi tạo: grid\_search = GridSearchCV(estimator=RandomForestClassifier(random\_state=42), param\_grid=param\_grid, cv=5, scoring='accuracy').
  + Fit: grid\_search.fit(X\_train, y\_train).
  + Kết quả: best\_params = grid\_search.best\_params\_ (ví dụ: {'max\_depth': 5, 'n\_estimators': 100}), best\_score = grid\_search.best\_score\_ (accuracy trung bình từ CV), và grid\_search.predict(X\_test) cho dự đoán cuối.
* **Ví dụ tham số phổ biến**:
  + Decision Tree: {'max\_depth': [3, 5, 10, None], 'min\_samples\_split': [2, 5, 10], 'min\_samples\_leaf': [1, 2, 4]}.
  + Random Forest: {'n\_estimators': [50, 100, 200], 'max\_depth': [5, 10, None], 'min\_samples\_split': [2, 5]}.

**2. RandomizedSearchCV: Tìm Kiếm Ngẫu Nhiên**

* **Mô tả**: Thử **một số tổ hợp ngẫu nhiên** từ phân phối tham số (thay vì lưới cố định), với số lần thử cố định (n\_iter). Nó nhanh hơn GridSearchCV cho không gian lớn, bằng cách lấy mẫu ngẫu nhiên (sampling) từ các phân phối (như uniform hoặc randint).
* **Ưu điểm**: Hiệu quả với nhiều tham số hoặc giá trị liên tục (ví dụ: max\_depth từ 1-20); thường tìm kết quả gần tối ưu với ít lần thử hơn.
* **Nhược điểm**: Không đảm bảo tối ưu toàn cục, phụ thuộc vào seed ngẫu nhiên.
* **Cách sử dụng**:
  + Import: from sklearn.model\_selection import RandomizedSearchCV và from scipy.stats import randint, uniform.
  + Định nghĩa phân phối: param\_dist = {'max\_depth': randint(3, 11), 'min\_samples\_split': randint(2, 11), 'n\_estimators': randint(50, 201)} (sử dụng phân phối để tạo giá trị ngẫu nhiên).
  + Khởi tạo: random\_search = RandomizedSearchCV(estimator=RandomForestClassifier(random\_state=42), param\_distributions=param\_dist, n\_iter=20, cv=5, scoring='accuracy', random\_state=42).
  + Fit: random\_search.fit(X\_train, y\_train).
  + Kết quả: Tương tự GridSearchCV, với best\_params\_ và best\_score\_. Tham số n\_iter=20 nghĩa là thử 20 tổ hợp ngẫu nhiên.
* **Ví dụ tham số phổ biến**:
  + Decision Tree: {'max\_depth': randint(3, 15), 'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random']}.
  + Random Forest: {'n\_estimators': randint(50, 300), 'max\_features': uniform(0.1, 0.9), 'max\_depth': randint(5, 20)}.

**So Sánh GridSearchCV Và RandomizedSearchCV**

| **Tiêu chí** | **GridSearchCV** | **RandomizedSearchCV** |
| --- | --- | --- |
| **Phương pháp** | Thử tất cả tổ hợp | Thử ngẫu nhiên (n\_iter lần) |
| **Thời gian** | Chậm (tăng theo số tổ hợp) | Nhanh (cố định theo n\_iter) |
| **Phù hợp** | Không gian nhỏ (2-3 tham số) | Không gian lớn (nhiều tham số) |
| **Kết quả** | Tối ưu chính xác | Gần tối ưu, hiệu quả |
| **Sử dụng** | Khi tài nguyên dồi dào | Khi cần nhanh, thử nghiệm ban đầu |

# GIẢI THUẬT 2: SUPPORT VECTOR MACHINE (SVM)

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

**Giải Thuật Support Vector Machine (SVM) Hoạt Động Như Thế Nào?**

Support Vector Machine (SVM) là một thuật toán học máy có giám sát (supervised learning), chủ yếu dùng cho bài toán phân loại (classification) và hồi quy (regression), nhưng phổ biến nhất là phân loại. SVM được phát triển bởi Vapnik và các cộng sự vào những năm 1990, dựa trên ý tưởng tìm ra một **ranh giới phân tách tối ưu** giữa các lớp dữ liệu. Nó đặc biệt hiệu quả với dữ liệu chiều cao (high-dimensional) và ít nhạy cảm với overfitting so với một số mô hình khác.

**Nguyên Lý Hoạt Động Tổng Quát**

SVM hoạt động bằng cách **tối ưu hóa một hàm mục tiêu** để tìm ranh giới phân tách (hyperplane) sao cho khoảng cách từ ranh giới đến các điểm dữ liệu gần nhất (gọi là support vectors) là lớn nhất. Quá trình bao gồm:

1. **Huấn luyện (Training)**: Với dữ liệu có nhãn (labeled data), SVM sử dụng phương pháp học lồi (convex optimization) để giải bài toán quadratic programming, tìm hyperplane tốt nhất.
2. **Dự đoán (Prediction)**: Với dữ liệu mới, SVM xác định lớp bằng cách kiểm tra vị trí của điểm so với hyperplane (phía nào của ranh giới).
3. **Xử lý dữ liệu không tuyến tính**: Nếu dữ liệu không thể phân tách tuyến tính, SVM sử dụng **kernel trick** (như RBF, polynomial) để ánh xạ dữ liệu lên không gian chiều cao hơn mà không cần tính toán trực tiếp.

SVM có hai biến thể chính:

* **Hard Margin SVM**: Giả định dữ liệu hoàn toàn phân tách được (linearly separable), không cho phép lỗi.
* **Soft Margin SVM**: Cho phép một số lỗi (dùng tham số C để kiểm soát độ nghiêm ngặt), phù hợp với dữ liệu thực tế có nhiễu.

**Khái Niệm Về Ranh Giới Phân Tách (Hyperplane)**

* **Định nghĩa**: Hyperplane là một **siêu mặt phẳng** (hyperplane) trong không gian n-chiều, đại diện cho ranh giới quyết định giữa các lớp dữ liệu. Trong không gian 2D, nó là một đường thẳng; 3D là mặt phẳng; cao hơn là siêu mặt phẳng.
* **Công thức toán học**: Một hyperplane được biểu diễn bởi phương trình , trong đó:
  + : Vector pháp tuyến (normal vector), quyết định hướng của hyperplane.
  + : Vector đặc trưng của điểm dữ liệu.
  + : Bias (thiên vị), quyết định vị trí dịch chuyển của hyperplane.
* **Vai trò**: Hyperplane phân chia không gian thành hai nửa (hoặc nhiều hơn với multi-class SVM qua one-vs-one hoặc one-vs-all). Điểm nào có thuộc lớp dương, ngược lại thuộc lớp âm.
* **Ví dụ minh họa**: Giả sử phân loại email spam/not spam dựa trên 2 đặc trưng (từ khóa, độ dài). Hyperplane là đường thẳng phân tách hai nhóm điểm.

**Khái Niệm Về Lề (Margin)**

* **Định nghĩa**: Margin là **khoảng cách** từ hyperplane đến các điểm dữ liệu gần nhất (support vectors) ở mỗi bên. SVM tìm hyperplane có **margin lớn nhất** (maximum margin hyperplane) để tăng khả năng tổng quát hóa (generalization), giảm rủi ro phân loại sai dữ liệu mới.
* **Công thức toán học**: Khoảng cách từ một điểm đến hyperplane là . Margin tổng = (khoảng cách giữa hai support hyperplanes song song với hyperplane chính).
  + Để tối ưu, SVM giảm thiểu (tương đương tăng margin) dưới ràng buộc phân loại đúng.
* **Phân biệt**:
  + **Functional Margin**: Khoảng cách có dấu (signed distance), dùng để kiểm tra phân loại đúng/sai.
  + **Geometric Margin**: Khoảng cách tuyệt đối, là giá trị SVM tối ưu hóa.
* **Vai trò**: Margin lớn giúp mô hình "chắc chắn" hơn (confident), chống overfitting. Nếu margin nhỏ, hyperplane dễ bị ảnh hưởng bởi nhiễu.

**Minh Họa Sự Khác Biệt Giữa Hyperplane Và Margin**

Dưới đây là bảng so sánh ngắn gọn:

| **Khái Niệm** | **Mô Tả** | **Vai Trò Trong SVM** |
| --- | --- | --- |
| **Hyperplane** | Siêu mặt phẳng phân tách các lớp dữ liệu (ranh giới quyết định). | Xác định lớp cho dữ liệu mới. |
| **Margin** | Khoảng cách từ hyperplane đến support vectors (các điểm gần nhất). | Tối ưu hóa để tăng độ chính xác tổng quát. |

**Ví dụ trực quan**: Trong không gian 2D với hai lớp điểm (đỏ và xanh), hyperplane là đường thẳng giữa chúng. Hai đường song song sát các điểm gần nhất định nghĩa margin. SVM chọn đường thẳng giữa có khoảng cách rộng nhất đến các điểm biên.

**Ưu Nhược Điểm Ngắn Gọn**

* **Ưu**: Hiệu quả với dữ liệu nhỏ/trung bình, chiều cao; margin giúp chống overfitting.
* **Nhược**: Chậm với dữ liệu lớn; khó giải thích (hộp đen); nhạy cảm với kernel và tham số C.

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

**Vai Trò Của Các Vector Hỗ Trợ (Support Vectors) Trong SVM**

Trong thuật toán **Support Vector Machine (SVM)**, **vector hỗ trợ (support vectors)** là các điểm dữ liệu đặc biệt nằm trên hoặc gần **ranh giới phân tách (hyperplane)**, cụ thể là các điểm gần nhất với hyperplane từ mỗi lớp dữ liệu. Chúng đóng vai trò cốt lõi trong việc xây dựng và sử dụng mô hình SVM. Dưới đây là giải thích chi tiết về vai trò và tầm quan trọng của chúng.

**Vai Trò Chính Của Support Vectors**

1. **Xác Định Vị Trí Hyperplane**:
   * Hyperplane (ranh giới phân tách) được tính toán sao cho khoảng cách đến support vectors (margin) là lớn nhất. Các support vectors là những điểm "giới hạn" – nếu di chuyển chúng, hyperplane sẽ thay đổi.
   * Trong toán học, support vectors thỏa mãn điều kiện ràng buộc của bài toán tối ưu: (với là nhãn lớp, là điểm dữ liệu). Các điểm khác có giá trị lớn hơn 1, nên không ảnh hưởng đến hyperplane.
2. **Xây Dựng Margin**:
   * Support vectors định nghĩa hai đường biên song song với hyperplane (support hyperplanes), tạo nên margin. Chúng là các điểm nằm trên hai đường biên này, giúp SVM đạt được "maximum margin" – khoảng cách rộng nhất giữa các lớp.
3. **Dự Đoán Cho Dữ Liệu Mới**:
   * Khi dự đoán, SVM chỉ cần sử dụng support vectors để tính toán vị trí của điểm mới so với hyperplane. Công thức dự đoán là , trong đó là hàm kernel, và chỉ support vectors () có hệ số .
4. **Trong SVM Với Kernel (Non-Linear)**:
   * Support vectors giúp "kernel trick" hoạt động hiệu quả bằng cách chỉ cần tính toán trên các điểm này trong không gian ánh xạ cao chiều, thay vì toàn bộ dữ liệu.

**Tại Sao Support Vectors Quan Trọng?**

Support vectors không chỉ là "công cụ" mà còn là yếu tố làm nên sức mạnh của SVM. Dưới đây là các lý do chính:

| **Lý Do Quan Trọng** | **Giải Thích** |
| --- | --- |
| **Tối Ưu Hóa Tổng Quát Hóa (Generalization)** | Bằng cách tập trung vào margin lớn nhất (do support vectors định nghĩa), SVM giảm overfitting và tăng khả năng dự đoán chính xác trên dữ liệu chưa thấy, đặc biệt với dữ liệu chiều cao. |
| **Giảm Độ Phức Tạp Mô Hình (Sparsity)** | SVM chỉ lưu trữ support vectors (thường chỉ 1-10% dữ liệu), không cần toàn bộ dataset. Điều này làm mô hình nhẹ hơn, tiết kiệm bộ nhớ và thời gian dự đoán (O(số support vectors) thay vì O(n)). |
| **Robust Với Nhiễu** | Các điểm xa margin (không phải support vectors) bị bỏ qua, nên nhiễu hoặc outlier ở vùng xa không ảnh hưởng đến hyperplane, tăng độ bền vững. |
| **Dễ Giải Thích Và Mở Rộng** | Support vectors giúp hiểu mô hình (ví dụ: "mô hình dựa trên những điểm dữ liệu cụ thể này"). Chúng cũng hỗ trợ các biến thể như SVM cho multi-class hoặc one-class detection. |
| **Hiệu Quả Trong Soft Margin SVM** | Với tham số C (kiểm soát lỗi), support vectors bao gồm cả các điểm "vi phạm" margin, giúp cân bằng giữa độ chính xác và tổng quát hóa. |

Ví dụ minh họa: Giả sử phân loại hình ảnh mèo/chó với 1000 ảnh. Chỉ khoảng 50 ảnh (support vectors) gần ranh giới "mèo-chó" sẽ quyết định hyperplane; các ảnh rõ ràng (xa ranh giới) không cần lưu trữ.

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

Support Vector Machine (SVM) là thuật toán phân loại dựa trên việc tìm hyperplane phân tách các lớp dữ liệu với margin (lề) lớn nhất. Sự khác biệt chính giữa hard margin SVM và soft margin SVM nằm ở cách xử lý dữ liệu không phân tách tuyến tính hoàn hảo (non-linearly separable), tức là khi có nhiễu, outlier hoặc các điểm dữ liệu "xâm phạm" margin. Dưới đây là bảng so sánh chi tiết:

| **Tiêu chí** | **Hard Margin SVM** | **Soft Margin SVM** |
| --- | --- | --- |
| **Giả định dữ liệu** | Dữ liệu phải hoàn toàn phân tách tuyến tính (tất cả điểm thuộc đúng lớp, không vi phạm margin). | Dữ liệu có thể không phân tách hoàn hảo; cho phép một số điểm vi phạm margin. |
| **Ràng buộc** | cho tất cả điểm . Không cho phép lỗi. | Giới thiệu biến slack : . Cho phép lỗi có kiểm soát. |
| **Hàm mục tiêu** | Tối thiểu hóa (tăng margin lớn nhất). | Tối thiểu hóa , trong đó C kiểm soát mức phạt lỗi (trade-off giữa margin và lỗi). |
| **Margin** | Margin lớn nhất, nhưng chỉ khả thi nếu dữ liệu separable. | Margin có thể lớn hơn nhờ cho phép vi phạm, nhưng linh hoạt hơn. |
| **Xử lý nhiễu/outlier** | Không xử lý được; nếu có nhiễu, bài toán không có nghiệm (infeasible). | Xử lý tốt bằng cách "bỏ qua" một số điểm nhiễu qua ξ\_i lớn. |
| **Số lượng support vectors** | Thường ít hơn, vì yêu cầu nghiêm ngặt. | Có thể nhiều hơn, bao gồm cả support vectors vi phạm (boundary và error vectors). |
| **Ứng dụng** | Chỉ dùng cho dữ liệu lý tưởng, separable (hiếm gặp thực tế). | Phổ biến hơn, dùng cho dữ liệu thực tế có nhiễu. |

**Hard Margin**: Tập trung hoàn toàn vào việc tối ưu margin mà không cho phép bất kỳ sai lầm nào. Nếu dữ liệu không separable, thuật toán thất bại (không hội tụ).

**Soft Margin**: Giới thiệu tham số **C** (regularization parameter):

* C lớn (ví dụ: >1): Phạt lỗi mạnh → gần giống hard margin (ít vi phạm).
* C nhỏ (ví dụ: <1): Cho phép nhiều lỗi hơn → margin lớn, tổng quát hóa tốt hơn.

**Khi Nào Nên Sử Dụng Lề Mềm (Soft Margin)?**

Soft margin SVM là lựa chọn mặc định trong hầu hết các trường hợp thực tế vì dữ liệu hiếm khi hoàn hảo. Nên sử dụng khi:

1. **Dữ liệu không phân tách tuyến tính hoàn hảo**: Có overlap giữa các lớp, nhiễu (noise), hoặc outlier (ví dụ: dữ liệu y tế với lỗi đo lường, hình ảnh có biến đổi nhỏ).
2. **Cần cân bằng giữa độ chính xác và tổng quát hóa**: Hard margin dễ overfitting (học thuộc lòng dữ liệu huấn luyện), trong khi soft margin giúp mô hình dự đoán tốt hơn trên dữ liệu mới bằng cách cho phép một số lỗi nhỏ.
3. **Dataset lớn hoặc phức tạp**: Với kernel (như RBF), soft margin giúp tránh overfit bằng cách điều chỉnh C qua cross-validation.
4. **Ứng dụng thực tế**: Như phân loại văn bản (có từ đồng nghĩa gây nhiễu), nhận diện khuôn mặt (ánh sáng thay đổi), hoặc dự báo tài chính (dữ liệu biến động).

## 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

**Hàm Nhân (Kernel) Trong SVM Là Gì?**

Trong **Support Vector Machine (SVM)**, **hàm nhân (kernel function)** là một kỹ thuật toán học quan trọng để xử lý dữ liệu **không tuyến tính** (non-linearly separable). Thay vì cố gắng phân tách dữ liệu trực tiếp trong không gian gốc (có thể không khả thi), kernel "ánh xạ" (map) dữ liệu lên một **không gian đặc trưng cao chiều hơn** (feature space), nơi dữ liệu trở nên tuyến tính phân tách được. Điều này được thực hiện thông qua **kernel trick** – một cách tính toán thông minh chỉ sử dụng tích vô hướng (dot product) giữa các điểm dữ liệu, mà không cần tính toán trực tiếp không gian cao chiều (tiết kiệm tài nguyên).

* **Công thức tổng quát**: Kernel , trong đó là hàm ánh xạ lên không gian cao chiều.
* **Vai trò**: Kernel quyết định hình dạng của hyperplane phân tách, giúp SVM linh hoạt với dữ liệu phức tạp. Nếu dữ liệu tuyến tính, có thể dùng kernel tuyến tính (không cần ánh xạ).

Dưới đây là giải thích các loại kernel phổ biến: **linear**, **polynomial**, và **RBF** (Radial Basis Function). Tôi sử dụng bảng để so sánh rõ ràng.

**Các Loại Kernel Phổ Biến**

| **Loại Kernel** | **Công Thức** | **Đặc Điểm** | **Khi Nào Nên Sử Dụng** |
| --- | --- | --- | --- |
| **Linear** |  | - Ánh xạ tuyến tính, không thay đổi không gian (tương đương SVM tuyến tính). - Đơn giản, nhanh chóng. - Không có tham số cần tuning (trừ C). | - Dữ liệu **tuyến tính phân tách** (linearly separable). - Dataset lớn, chiều cao (high-dimensional, như text classification). - Khi ưu tiên tốc độ và ít tài nguyên (ví dụ: phân loại email spam cơ bản). |
| **Polynomial** | (d: bậc đa thức, c: hằng số) | - Tạo tương tác đa thức (polynomial features) lên không gian cao chiều. - Linh hoạt với d (bậc cao → phức tạp hơn). - Có thể overfit nếu d lớn. | - Dữ liệu có **mối quan hệ đa thức** (ví dụ: hình ảnh với đường cong, dữ liệu vật lý). - Khi cần mô hình phức tạp hơn linear nhưng không quá phi tuyến tính. - Tuning d=2-3 cho dataset trung bình. |
| **RBF (Gaussian)** | (σ: độ rộng Gaussian, hoặc γ = 1/(2σ²)) | - Ánh xạ lên không gian vô hạn chiều, tạo ranh giới "vòm" (curved). - Rất linh hoạt, phổ biến nhất. - Nhạy cảm với γ (nhỏ → overfit, lớn → underfit). | - Dữ liệu **phi tuyến tính phức tạp** (non-linear, không theo đa thức, như phân loại hình ảnh, sinh học). - Khi không biết hình dạng dữ liệu (mặc định tốt). - Dataset nhỏ/trung bình; tuning γ qua cross-validation. |

**Lời Khuyên Chọn Kernel**

* **Bắt đầu với Linear**: Nếu dữ liệu có ít đặc trưng và có vẻ tuyến tính (kiểm tra bằng scatter plot).
* **Chuyển sang RBF**: Nếu linear kém, vì RBF thường cho kết quả tốt nhất trên dữ liệu thực tế (nhưng tốn thời gian hơn).
* **Polynomial**: Chỉ dùng nếu biết dữ liệu có tính chất đa thức (hiếm hơn RBF).
* **Tuning**: Luôn dùng cross-validation (như GridSearchCV) để chọn kernel và tham số (C, γ, d). Trong Scikit-learn, kernel là tham số của SVC.

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

Tham Số C Trong SVM Có Ý Nghĩa Gì?

Trong **Support Vector Machine (SVM)** với **lề mềm (soft margin)**, **tham số C** (hay còn gọi là **regularization parameter**) là một hệ số điều chỉnh quan trọng, kiểm soát sự cân bằng giữa hai mục tiêu chính của mô hình:

* **Tối đa hóa margin**: Tăng khoảng cách giữa hyperplane và support vectors để cải thiện khả năng tổng quát hóa (generalization).
* **Giảm thiểu lỗi phân loại**: Phạt các điểm dữ liệu vi phạm margin (slack variables ).

**Công thức toán học**: Hàm mục tiêu của soft margin SVM là:

* : Phần regularization để tăng margin (giảm ).
* : Phần phạt lỗi, với quyết định mức độ nghiêm ngặt của phạt.
* **Ý nghĩa cụ thể**:
  + C lớn (ví dụ: C = 100): Phạt lỗi mạnh → mô hình ưu tiên phân loại đúng tất cả điểm huấn luyện → gần giống hard margin SVM.
  + C nhỏ (ví dụ: C = 0.1): Phạt lỗi yếu → cho phép nhiều vi phạm hơn → ưu tiên margin lớn, bỏ qua nhiễu/outlier.

C không ảnh hưởng đến hard margin SVM (vì không có slack variables), nhưng là tham số bắt buộc trong soft margin.

**Ảnh Hưởng Của C Đến Hiệu Suất Mô Hình**

Tham số C ảnh hưởng trực tiếp đến **trade-off giữa bias và variance**, dẫn đến overfitting/underfitting. Dưới đây là bảng tóm tắt tác động:

| **Giá Trị C** | **Ảnh Hưởng Đến Mô Hình** | **Ưu Điểm** | **Nhược Điểm** | **Hiệu Suất Tổng Thể** |
| --- | --- | --- | --- | --- |
| **C lớn** (ví dụ: >10) | - Giảm lỗi huấn luyện (high training accuracy). - Margin nhỏ hơn (ít vi phạm). - Nhiều support vectors hơn. | - Độ chính xác cao trên dữ liệu huấn luyện. - Ít nhầm lẫn với dữ liệu sạch. | - Dễ **overfitting** (variance cao): Kém trên dữ liệu test mới, nhạy cảm với nhiễu. - Mô hình phức tạp, ít tổng quát hóa. | Tốt cho dữ liệu sạch, nhỏ; kém với nhiễu. |
| **C nhỏ** (ví dụ: <1) | - Tăng margin (ít vi phạm hơn). - Lỗi huấn luyện cao hơn nhưng ổn định. - Ít support vectors hơn. | - **Tổng quát hóa tốt** (low variance): Robust với nhiễu/outlier. - Mô hình đơn giản, ít phức tạp. | - Có thể **underfitting** (bias cao): Độ chính xác thấp trên cả train và test. - Bỏ qua một số mẫu quan trọng. | Tốt cho dữ liệu lớn, có nhiễu; kém nếu dữ liệu separable. |
| **C trung bình** (ví dụ: 1) | - Cân bằng giữa margin và lỗi. - Hiệu suất ổn định. | - Linh hoạt, thường là điểm khởi đầu tốt. | - Cần tuning để tối ưu. | Tối ưu cho hầu hết trường hợp thực tế. |

**Ví dụ minh họa**:

* Với dataset Iris (dữ liệu sạch), C=1 cho accuracy ~95-100%. Nếu C=1000, accuracy train=100% nhưng test có thể giảm do overfit.
* Với dữ liệu có nhiễu (như MNIST với outlier), C=0.1 giúp tránh overfit, tăng accuracy test lên 5-10%.

## 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

**Các Bước Thực Hiện Xây Dựng Mô Hình SVM Cho Bài Toán Phân Loại Với Scikit-learn**

Support Vector Machine (SVM) trong Scikit-learn được triển khai qua lớp SVC (Support Vector Classification) cho phân loại. Dưới đây là hướng dẫn chi tiết các bước để xây dựng một mô hình SVM đơn giản cho bài toán phân loại, sử dụng dataset Iris (một bộ dữ liệu cổ điển với 3 lớp hoa Iris dựa trên 4 đặc trưng: chiều dài/rộng lá đài và cánh hoa). Quy trình này có thể áp dụng cho các dataset khác tương tự.

1. **Import các thư viện cần thiết**:
   * sklearn.datasets: Để tải dataset mẫu.
   * sklearn.model\_selection: Để chia dữ liệu thành tập huấn luyện và kiểm tra.
   * sklearn.svm: Để tạo mô hình SVC.
   * sklearn.metrics: Để đánh giá hiệu suất (ví dụ: accuracy\_score).
2. **Tải và chuẩn bị dữ liệu**:
   * Tải dataset Iris (150 mẫu).
   * Chia dữ liệu thành tập huấn luyện (80%) và kiểm tra (20%) bằng train\_test\_split để đánh giá tổng quát hóa.
3. **Tạo và huấn luyện mô hình**:
   * Khởi tạo mô hình SVC với các tham số cơ bản (ví dụ: kernel='linear', C=1.0 để bắt đầu).
   * Huấn luyện bằng phương thức fit() trên tập huấn luyện.
4. **Dự đoán và đánh giá mô hình**:
   * Sử dụng predict() để dự đoán trên tập kiểm tra.
   * Tính độ chính xác bằng accuracy\_score và in kết quả.
5. **(Tùy chọn) Tuning tham số**: Sử dụng GridSearchCV để tối ưu C hoặc kernel nếu cần, nhưng ở đây giữ đơn giản.

Quy trình này nhấn mạnh soft margin SVM (mặc định trong Scikit-learn). Trong thực tế, bạn có thể thêm kernel='rbf' cho dữ liệu phi tuyến tính hoặc tuning C qua cross-validation.

**Đoạn Code Mẫu (Python)**

Dưới đây là code hoàn chỉnh. Bạn có thể copy-paste và chạy trực tiếp trong môi trường Python có Scikit-learn (phiên bản >= 1.0).

# Bước 1: Import các thư viện cần thiết

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Bước 2: Tải và chuẩn bị dữ liệu

iris = load\_iris() # Tải dataset Iris

X = iris.data # Đặc trưng (4 cột: sepal length, sepal width, petal length, petal width)

y = iris.target # Nhãn (0: setosa, 1: versicolor, 2: virginica)

# Chia dữ liệu: 80% train, 20% test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Bước 3: Tạo và huấn luyện mô hình

svm = SVC(kernel='linear', C=1.0, random\_state=42) # Mô hình SVM phân loại (linear kernel)

svm.fit(X\_train, y\_train) # Huấn luyện trên tập train

# Bước 4: Dự đoán và đánh giá

y\_pred = svm.predict(X\_test) # Dự đoán trên tập test

accuracy = accuracy\_score(y\_test, y\_pred) # Tính độ chính xác

print(f"Độ chính xác trên tập kiểm tra: {accuracy:.2f}")

# Bước 5: Hiển thị số lượng support vectors (tùy chọn)

print(f"Số lượng support vectors: {svm.n\_support\_}")

**Kết Quả Chạy Mẫu**

Khi chạy code này, bạn sẽ nhận được output tương tự:

* Độ chính xác: 1.00 (100%, vì Iris dễ phân loại với SVM linear).
* Số lượng support vectors: [10 13 8] (số lượng theo từng lớp).

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

**Hàm Chuẩn Hóa Dữ Liệu (Scaling) Trong Scikit-learn Trước Khi Áp Dụng SVM**

Trong thư viện Scikit-learn, hàm phổ biến nhất để chuẩn hóa (scaling) dữ liệu trước khi áp dụng SVM là **StandardScaler** từ module sklearn.preprocessing. Đây là một transformer (biến đổi) chuẩn hóa các đặc trưng về phân phối có **trung bình = 0** và **độ lệch chuẩn = 1** (z-score normalization). Nó phù hợp nhất cho SVM vì thuật toán này dựa trên khoảng cách Euclidean và nhạy cảm với scale của dữ liệu.

**Cách Sử Dụng Ngắn Gọn**

* Import: from sklearn.preprocessing import StandardScaler.
* Tạo và áp dụng:

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train) # Fit trên train, transform train

X\_test\_scaled = scaler.transform(X\_test) # Chỉ transform test (không fit lại)

* Sau đó, truyền X\_train\_scaled vào mô hình SVM (ví dụ: SVC().fit(X\_train\_scaled, y\_train)).

Một lựa chọn khác là MinMaxScaler, chuẩn hóa về khoảng [0, 1] (hoặc khoảng tùy chỉnh), nhưng StandardScaler thường được ưu tiên cho SVM vì nó bảo toàn phân phối Gaussian (nếu có).

**Tại Sao Bước Chuẩn Hóa Này Quan Trọng Với SVM?**

SVM hoạt động dựa trên nguyên lý **tối ưu hóa margin** và **khoảng cách giữa các điểm dữ liệu** (Euclidean distance) để tìm hyperplane phân tách. Nếu không scaling, các đặc trưng có scale khác nhau (ví dụ: một đặc trưng từ 0-1, một từ 0-1000) sẽ làm méo mó mô hình:

1. **Bias hướng đặc trưng có scale lớn**: Đặc trưng với giá trị lớn (như thu nhập hàng năm) sẽ thống trị khoảng cách, khiến SVM ưu tiên chúng quá mức, dẫn đến hyperplane lệch lạc và accuracy thấp.
2. **Overfitting hoặc underfitting**: Không scaling làm margin không phản ánh đúng "tầm quan trọng" tương đối của đặc trưng, tăng variance (overfit) hoặc bias (underfit).
3. **Hiệu quả kernel**: Với kernel như RBF (phụ thuộc vào khoảng cách), scaling giúp kernel hoạt động đúng, tránh overfit (γ quá nhỏ) hoặc underfit (γ quá lớn).

Kết quả thực tế: Scaling có thể tăng accuracy SVM lên 10-20% trên dataset thực tế (như Iris hoặc MNIST). Luôn fit scaler chỉ trên train set để tránh data leakage.

# GIẢI THUẬT 3: BAYES NGÂY THƠ (NAÏVE BAYES)

## 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 Hoạt Động Như Thế Nào?**

**Naive Bayes** là một thuật toán học máy có giám sát (supervised learning), thuộc họ **phân loại xác suất** (probabilistic classifier), được sử dụng chủ yếu cho bài toán phân loại văn bản, spam detection, hoặc phân loại cảm xúc. Nó dựa trên **định lý Bayes** để tính xác suất lớp của một mẫu dữ liệu mới dựa trên các đặc trưng. Tên "Naive" (ngây thơ) xuất phát từ giả định đơn giản hóa mạnh mẽ về sự độc lập giữa các đặc trưng. Thuật toán nhanh, hiệu quả với dữ liệu lớn, và thường đạt hiệu suất tốt dù giả định đơn giản.

Quá trình hoạt động tổng quát:

1. **Huấn luyện (Training)**: Tính xác suất tiên nghiệm (prior probability) của từng lớp và xác suất có điều kiện (likelihood) của từng đặc trưng cho mỗi lớp, dựa trên dữ liệu huấn luyện.
2. **Dự đoán (Prediction)**: Với dữ liệu mới, tính xác suất hậu nghiệm (posterior probability) cho từng lớp bằng cách nhân các xác suất có điều kiện (dưới giả định độc lập), rồi chọn lớp có xác suất cao nhất.

Naive Bayes có các biến thể như **Gaussian Naive Bayes** (cho dữ liệu liên tục), **Multinomial Naive Bayes** (cho dữ liệu đếm, như văn bản), và **Bernoulli Naive Bayes** (cho dữ liệu nhị phân).

**Định Lý Bayes**

**Định lý Bayes** là nền tảng toán học của thuật toán, mô tả cách cập nhật xác suất dựa trên bằng chứng mới. Định lý này được đặt tên theo Thomas Bayes (thế kỷ 18), và được sử dụng để tính **xác suất hậu nghiệm** – xác suất một mẫu thuộc lớp sau khi quan sát bằng chứng .

**Công thức cơ bản**:

* : **Xác suất hậu nghiệm** (posterior) – xác suất lớp khi biết (kết quả cần tìm).
* : **Xác suất có điều kiện** (likelihood) – xác suất quan sát nếu lớp là .
* : **Xác suất tiên nghiệm** (prior) – xác suất lớp trước khi quan sát (thường tính từ tỷ lệ lớp trong dữ liệu huấn luyện).
* : **Xác suất bằng chứng** (evidence) – xác suất quan sát (hằng số chuẩn hóa, thường không tính trực tiếp vì so sánh giữa các lớp).

**Ví dụ minh họa**: Giả sử phân loại email là spam hay không dựa trên từ "miễn phí". Nếu , , , thì – email có khả năng cao là spam.

Định lý Bayes giúp Naive Bayes "cập nhật niềm tin" từ prior sang posterior dựa trên dữ liệu mới.

**Giả Định "Ngây Thơ" (Naive Assumption) Trong Naive Bayes**

Giả định cốt lõi của Naive Bayes là **các đặc trưng độc lập có điều kiện** (conditional independence): Các đặc trưng của một mẫu dữ liệu độc lập với nhau khi biết lớp . Điều này đơn giản hóa tính toán, nhưng thường không đúng hoàn toàn trong thực tế (các đặc trưng có thể tương quan).

**Công thức với giả định**:

* Thay vì tính xác suất liên hợp phức tạp , chỉ cần nhân các xác suất riêng lẻ.
* **Ưu điểm**: Giảm độ phức tạp tính toán từ O(2^n) xuống O(n), làm thuật toán nhanh và dễ triển khai.
* **Nhược điểm**: Nếu đặc trưng tương quan mạnh (ví dụ: "mèo" và "đuôi" trong phân loại động vật), mô hình có thể kém chính xác. Tuy nhiên, thực tế Naive Bayes vẫn hiệu quả nhờ "over-simplification" đôi khi bù đắp lỗi (theo Occam's razor).

**Bảng so sánh với Bayes thực tế**:

| **Tiêu chí** | **Naive Bayes (với giả định ngây thơ)** | **Bayes Thực Tế (không giả định)** |
| --- | --- | --- |
| **Tính toán P(X|C)** | Nhân độc lập: $$ \prod P(X\_i | C) $$ |
| **Độ phức tạp** | Thấp (O(n × số lớp)) | Cao (cần dữ liệu lớn để ước lượng) |
| **Hiệu suất** | Nhanh, tốt với dữ liệu lớn/văn bản; chịu nhiễu | Chính xác hơn nhưng chậm, khó triển khai |
| **Ứng dụng** | Phân loại spam, sentiment analysis | Mô hình Bayesian phức tạp (như Bayesian networks) |

Tóm lại, Naive Bayes hoạt động bằng cách áp dụng định lý Bayes với giả định độc lập để tính posterior nhanh chóng, chọn lớp có xác suất cao nhất. Dù "ngây thơ", nó là baseline mạnh mẽ trong nhiều bài toán thực tế.

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

**Các Loại Mô Hình Naive Bayes: Sự Khác Biệt Và Ứng Dụng**

Có ba biến thể phổ biến: Gaussian Naive Bayes, Multinomial Naive Bayes, và Bernoulli Naive Bayes. Sự khác biệt chính nằm ở phân phối xác suất mà chúng giả định cho dữ liệu đầu vào.

Dưới đây là bảng so sánh sự khác biệt giữa ba loại:

| **Loại mô hình** | **Phân phối giả định** | **Loại dữ liệu phù hợp** | **Công thức chính (xác suất có điều kiện)** |
| --- | --- | --- | --- |
| **Gaussian Naive Bayes** | Phân phối Gaussian (bình thường) | Dữ liệu liên tục (continuous), số thực (ví dụ: chiều cao, nhiệt độ) | (μ: trung bình, σ²: phương sai) |
| **Multinomial Naive Bayes** | Phân phối đa thức (multinomial) | Dữ liệu đếm (discrete, không âm), như tần suất (ví dụ: số lần xuất hiện từ trong văn bản) | (N\_{yi}: số lần xuất hiện, α: tham số Laplace smoothing, n: số đặc trưng) |
| **Bernoulli Naive Bayes** | Phân phối Bernoulli (nhị phân) | Dữ liệu nhị phân (binary), như có/không có (ví dụ: từ có xuất hiện trong văn bản hay không) | (x\_i = 1 nếu có, 0 nếu không) |

**Sự khác biệt chi tiết**

* **Gaussian**: Tập trung vào dữ liệu có thể đo lường liên tục và giả định chúng phân bố quanh một giá trị trung bình (như đường cong chuông). Nó ước lượng tham số bằng trung bình và phương sai của từng lớp.
* **Multinomial**: Dành cho dữ liệu "đếm" (count-based), nơi giá trị đặc trưng là số nguyên không âm (≥0). Nó xử lý tốt sự thưa thớt (sparsity) trong dữ liệu lớn, thường dùng với kỹ thuật bag-of-words.
* **Bernoulli**: Chỉ quan tâm đến sự hiện diện (1) hoặc vắng mặt (0) của đặc trưng, bỏ qua tần suất. Nó đơn giản hơn Multinomial và phù hợp với dữ liệu nhị phân thuần túy.

**Khi nào nên sử dụng từng loại?**

* **Gaussian Naive Bayes**: Sử dụng khi dữ liệu đầu vào là **liên tục và số thực**, không bị giới hạn ở các giá trị nguyên. Ví dụ: Phân loại bệnh dựa trên chỉ số máu (như huyết áp, cholesterol). Không phù hợp với dữ liệu đếm hoặc nhị phân vì có thể dẫn đến xác suất âm hoặc không hợp lý.
* **Multinomial Naive Bayes**: Lý tưởng cho **dữ liệu đếm hoặc tần suất**, đặc biệt trong xử lý ngôn ngữ tự nhiên (NLP). Ví dụ: Phân loại email spam dựa trên số lần xuất hiện từ khóa (bag-of-words model). Tránh dùng cho dữ liệu liên tục vì nó không xử lý tốt giá trị âm hoặc thập phân.
* **Bernoulli Naive Bayes**: Chọn khi dữ liệu là **nhị phân hoặc có thể chuyển thành nhị phân** (ví dụ: vector đặc trưng chỉ ghi nhận sự hiện diện của từ, không quan tâm số lần lặp). Ví dụ: Phân loại tài liệu dựa trên từ khóa có/không có. Nó hiệu quả hơn Multinomial nếu dữ liệu thưa thớt và bạn không cần thông tin tần suất.

Tóm lại, lựa chọn phụ thuộc vào **loại dữ liệu**: liên tục → Gaussian; đếm → Multinomial; nhị phân → Bernoulli. Trong thực tế, bạn có thể thử nghiệm cả ba (với thư viện như scikit-learn) để xem mô hình nào cho độ chính xác cao nhất trên dữ liệu cụ thể. Nếu dữ liệu hỗn hợp, có thể cần tiền xử lý (normalization) hoặc mô hình lai.

## 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?

**Tại sao Naive Bayes được gọi là "ngây thơ"?**

Naive Bayes được gọi là "ngây thơ" (naive) vì nó dựa trên một giả định đơn giản hóa mạnh mẽ: **các đặc trưng (features) trong dữ liệu đầu vào là độc lập với nhau có điều kiện (conditionally independent) khi biết lớp nhãn (class label)**. Giả định này thường không phản ánh đúng thực tế, vì trong hầu hết các bộ dữ liệu, các đặc trưng có thể phụ thuộc lẫn nhau (ví dụ: trong phân loại văn bản, sự xuất hiện của từ "bác sĩ" có thể liên quan đến từ "bệnh viện"). Sự "ngây thơ" nằm ở việc mô hình bỏ qua các mối quan hệ phức tạp này để đơn giản hóa tính toán, dẫn đến việc ước lượng xác suất dễ dàng hơn nhưng có thể kém chính xác hơn so với các mô hình phức tạp.

Giả định này xuất phát từ **định lý Bayes**, nơi xác suất hậu nghiệm được tính như sau:

Ở đây, được tính riêng lẻ cho từng đặc trưng , bỏ qua sự phụ thuộc giữa các . Điều này làm cho mô hình "ngây thơ" nhưng hiệu quả về mặt tính toán.

**Giả định về tính độc lập ảnh hưởng như thế nào đến hiệu suất của mô hình?**

Giả định độc lập là yếu tố cốt lõi quyết định ưu và nhược điểm của Naive Bayes. Dưới đây là phân tích chi tiết về ảnh hưởng đến hiệu suất:

**1. Ưu điểm (lý do mô hình vẫn hiệu quả dù "ngây thơ"):**

* **Tốc độ tính toán cao và dễ triển khai**: Vì không cần ước lượng ma trận hiệp phương sai (covariance matrix) giữa các đặc trưng, mô hình chỉ cần tính xác suất biên (marginal probabilities) riêng lẻ. Điều này làm cho Naive Bayes rất nhanh, đặc biệt với dữ liệu lớn (big data), và yêu cầu ít tài nguyên tính toán hơn so với các mô hình như Logistic Regression hoặc SVM.
* **Tránh overfitting**: Giả định độc lập giảm số lượng tham số cần học (chỉ trung bình/phương sai hoặc tần suất cho từng đặc trưng riêng lẻ), giúp mô hình tổng quát hóa tốt hơn trên dữ liệu mới, ngay cả khi dữ liệu huấn luyện nhỏ.
* **Hiệu suất tốt trong thực tế**: Ngạc nhiên thay, mô hình thường đạt độ chính xác cao ngay cả khi giả định không đúng hoàn toàn. Lý do là:
  + Trong nhiều bài toán (như phân loại văn bản hoặc spam detection), sự phụ thuộc giữa đặc trưng có thể "bù trừ" lẫn nhau, dẫn đến xác suất tổng thể vẫn hợp lý.
  + Nó hoạt động như một "mô hình cơ sở" (baseline) mạnh mẽ, thường vượt trội hơn các phương pháp phức tạp hơn trong các tình huống dữ liệu thưa thớt (sparse data).

**2. Nhược điểm (hậu quả tiêu cực):**

* **Mất thông tin về sự phụ thuộc**: Nếu các đặc trưng thực sự phụ thuộc mạnh (ví dụ: trong dữ liệu y tế, huyết áp và nhịp tim liên quan chặt chẽ), mô hình có thể ước lượng xác suất sai lệch, dẫn đến độ chính xác thấp hơn. Ví dụ: Trong phân loại hình ảnh, màu sắc và hình dạng của đối tượng thường phụ thuộc, nên Naive Bayes có thể kém hiệu quả.
* **Xử lý kém với dữ liệu không cân bằng hoặc outlier**: Giả định độc lập có thể phóng đại tác động của các đặc trưng nhiễu, làm mô hình nhạy cảm hơn với dữ liệu không đại diện.
* **Giới hạn trong các bài toán phức tạp**: Không phù hợp cho dữ liệu có tương tác cao giữa đặc trưng, nơi các mô hình như Random Forest hoặc Neural Networks sẽ tốt hơn.

**Tóm tắt ảnh hưởng qua ví dụ:**

* **Bài toán phù hợp (giả định gần đúng)**: Phân loại email spam dựa trên tần suất từ khóa – các từ thường độc lập tương đối → Hiệu suất cao (F1-score ~90-95%).
* **Bài toán không phù hợp (phụ thuộc mạnh)**: Dự đoán giá nhà dựa trên diện tích và vị trí – hai yếu tố phụ thuộc → Hiệu suất thấp hơn (có thể cần mô hình khác để cải thiện).

Tổng thể, giả định "ngây thơ" làm Naive Bayes trở thành lựa chọn lý tưởng cho các bài toán đơn giản, dữ liệu lớn và cần tốc độ cao, nhưng bạn nên kiểm tra độ chính xác trên dữ liệu cụ thể (qua cross-validation) và xem xét các biến thể (như Gaussian hoặc Multinomial) để tối ưu. Nếu phụ thuộc đặc trưng quá mạnh, có thể dùng kỹ thuật tiền xử lý như feature selection để giảm tác động.

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

**So sánh Naive Bayes với SVM và Random Forest**

Naive Bayes là một thuật toán phân loại xác suất đơn giản, trong khi SVM (Support Vector Machine) tập trung vào việc tìm biên giới phân cách tối ưu và Random Forest là mô hình ensemble dựa trên cây quyết định. Dưới đây là bảng so sánh ưu điểm và hạn chế của Naive Bayes so với hai thuật toán kia, tập trung vào các khía cạnh chính như tốc độ, độ chính xác, khả năng xử lý dữ liệu và tính dễ sử dụng.

| **Tiêu chí** | **Naive Bayes** | **SVM** | **Random Forest** |
| --- | --- | --- | --- |
| **Ưu điểm so với SVM** | - Tốc độ huấn luyện và dự đoán cực nhanh (O(n) thời gian), phù hợp dữ liệu lớn. - Yêu cầu ít bộ nhớ hơn (chỉ tính xác suất biên). - Tốt cho dữ liệu thưa thớt (sparse, như văn bản). | - Độ chính xác cao hơn trong dữ liệu chiều cao (high-dimensional) và không tuyến tính (với kernel). - Robust với outlier nhờ margin maximization. - Ít bị overfitting hơn nếu dùng kernel phù hợp. | - Xử lý tốt tương tác giữa đặc trưng mà không cần giả định độc lập. - Cung cấp feature importance dễ hiểu. - Ít overfitting nhờ bagging. |
| **Hạn chế so với SVM** | - Giả định độc lập đặc trưng có thể dẫn đến độ chính xác thấp nếu đặc trưng phụ thuộc mạnh. - Không xử lý tốt dữ liệu không tuyến tính mà không có biến đổi. - Xác suất đầu ra có thể không calibrated tốt. | - Chậm huấn luyện với dữ liệu lớn (O(n²) hoặc hơn với kernel). - Khó chọn kernel và tham số (C, gamma). - Không hiệu quả với dữ liệu thưa thớt lớn. | - Không cung cấp xác suất xác suất trực tiếp (cần thêm bước). - Có thể kém hơn Naive Bayes ở dữ liệu văn bản thuần túy do phức tạp hơn. |
| **Ưu điểm so với Random Forest** | - Huấn luyện nhanh hơn nhiều (phù hợp real-time). - Dễ triển khai và giải thích (dựa trên xác suất Bayes). - Hiệu quả cao với dữ liệu nhỏ và categorical (như Multinomial cho văn bản). | - Tốt hơn ở dữ liệu nhỏ, nơi Random Forest có thể overfitting nếu không tune. - Phân loại nhị phân chính xác hơn nhờ hyperplane. | - Độ chính xác cao hơn tổng thể nhờ ensemble (thường vượt Naive Bayes 5-10% ở dữ liệu phức tạp). - Xử lý missing values và mixed data tốt hơn. - Robust với noise và imbalance. |
| **Hạn chế so với Random Forest** | - Không nắm bắt tương tác đặc trưng (do giả định naive). - Dễ bị ảnh hưởng bởi dữ liệu imbalance (zero probability). - Độ chính xác thấp hơn ở dữ liệu có phụ thuộc mạnh. | - Chậm hơn Random Forest ở dữ liệu lớn (Random Forest scale tốt hơn). - Khó giải thích (black-box với kernel). - Không cung cấp feature importance trực tiếp. | - Huấn luyện chậm (O(n log n) với nhiều cây). - Yêu cầu nhiều bộ nhớ và tài nguyên hơn. - Ít hiệu quả ở dữ liệu rất lớn hoặc real-time. |

**Tóm tắt**

* **Chọn Naive Bayes khi**: Cần tốc độ cao, dữ liệu lớn/sparse (như NLP, spam detection), và chấp nhận giả định đơn giản. Nó thường là baseline mạnh mẽ và dễ dùng.
* **Chọn SVM khi**: Dữ liệu chiều cao, cần độ chính xác cao ở không gian tuyến tính/phi tuyến, nhưng dữ liệu không quá lớn.
* **Chọn Random Forest khi**: Muốn độ chính xác cao, xử lý tương tác đặc trưng, và có tài nguyên tính toán (như dự đoán tài chính hoặc y tế).

Trong thực tế, hiệu suất phụ thuộc vào dữ liệu cụ thể – hãy dùng cross-validation để thử nghiệm. Naive Bayes thường nhanh và tiết kiệm, nhưng SVM/Random Forest vượt trội ở độ chính xác cho bài toán phức tạp.

## 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

Các bước thực hiện xây dựng mô hình Gaussian Naive Bayes bằng Scikit-learn

Dưới đây là hướng dẫn chi tiết các bước để xây dựng một mô hình phân loại Naive Bayes (sử dụng Gaussian Naive Bayes, phù hợp với dữ liệu liên tục) bằng Python và thư viện Scikit-learn. Tôi sẽ sử dụng bộ dữ liệu Iris (một bộ dữ liệu cổ điển về phân loại hoa) để minh họa. Các bước được thực hiện theo quy trình tiêu chuẩn: chuẩn bị dữ liệu, huấn luyện mô hình, đánh giá và dự đoán.

1. **Cài đặt và import thư viện**: Đảm bảo bạn đã cài đặt Scikit-learn (qua pip install scikit-learn). Import các module cần thiết như GaussianNB từ sklearn.naive\_bayes, train\_test\_split để chia dữ liệu, load\_iris để tải dữ liệu mẫu, và accuracy\_score để đánh giá.
2. **Chuẩn bị dữ liệu**: Tải bộ dữ liệu (hoặc sử dụng dữ liệu của bạn dưới dạng DataFrame hoặc array). Với Iris, dữ liệu có 4 đặc trưng liên tục (chiều dài/dài cánh/dài/ rộng đài hoa) và 3 lớp nhãn.
3. **Chia dữ liệu**: Phân chia dữ liệu thành tập huấn luyện (train) và tập kiểm tra (test), thường theo tỷ lệ 80/20, để đánh giá mô hình trên dữ liệu chưa thấy.
4. **Khởi tạo và huấn luyện mô hình**: Tạo instance của GaussianNB(), sau đó gọi phương thức fit(X\_train, y\_train) để huấn luyện.
5. **Dự đoán và đánh giá**: Sử dụng predict(X\_test) để dự đoán lớp trên tập test. Đánh giá bằng các chỉ số như accuracy (độ chính xác) hoặc confusion matrix.
6. **Dự đoán trên dữ liệu mới** (tùy chọn): Sử dụng mô hình đã huấn luyện để dự đoán trên dữ liệu mới.

**Đoạn code mẫu**

Dưới đây là code hoàn chỉnh, có thể chạy trực tiếp trong Jupyter Notebook hoặc script Python. Code này đạt độ chính xác khoảng 95-98% trên bộ dữ liệu Iris.

# Bước 1: Import thư viện

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score, classification\_report

import numpy as np

# Bước 2: Chuẩn bị dữ liệu (sử dụng bộ dữ liệu Iris mẫu)

iris = load\_iris()

X = iris.data # Đặc trưng (4 cột: sepal length, sepal width, petal length, petal width)

y = iris.target # Nhãn (0: setosa, 1: versicolor, 2: virginica)

print("Kích thước dữ liệu:", X.shape) # Output: (150, 4)

# Bước 3: Chia dữ liệu thành train/test (80/20)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Bước 4: Khởi tạo và huấn luyện mô hình

model = GaussianNB()

model.fit(X\_train, y\_train) # Huấn luyện trên tập train

# Bước 5: Dự đoán và đánh giá

y\_pred = model.predict(X\_test) # Dự đoán trên tập test

accuracy = accuracy\_score(y\_test, y\_pred)

print("Độ chính xác trên tập test:", accuracy) # Thường ~0.9667

# In báo cáo chi tiết

print("\nBáo cáo phân loại:")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

# Bước 6: Dự đoán trên dữ liệu mới (ví dụ: một mẫu mới)

X\_new = np.array([[5.1, 3.5, 1.4, 0.2]]) # Mẫu giả định (gần với setosa)

prediction = model.predict(X\_new)

print("\nDự đoán cho mẫu mới:", iris.target\_names[prediction[0]]) # Output: setosa

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

Xử lý dữ liệu phân loại (Categorical Data) trước khi áp dụng Multinomial Naive Bayes

Multinomial Naive Bayes (MultinomialNB) trong Scikit-learn được thiết kế chủ yếu cho dữ liệu **đếm (count-based)** hoặc **không âm (non-negative integers)**, như tần suất từ trong văn bản (bag-of-words). Nó **không hỗ trợ trực tiếp dữ liệu phân loại (categorical)** dạng chuỗi (ví dụ: "nam"/"nữ", "đỏ"/"xanh"), vì các giá trị này không phải là số đếm. Do đó, bạn cần **chuyển đổi categorical data thành dạng số phù hợp** trước khi huấn luyện mô hình.

**Lý do và nguyên tắc xử lý**

* **Vấn đề**: MultinomialNB giả định đặc trưng là xác suất đa thức (multinomial distribution), nên cần dữ liệu ≥0 (thường là integers). Categorical data có thể gây lỗi (ValueError) nếu không encode.
* **Phương pháp chính**: Sử dụng **One-Hot Encoding (OHE)** để chuyển categorical thành binary vectors (0/1), đại diện cho sự hiện diện của từng danh mục. Điều này phù hợp vì OHE tạo ra "đếm" nhị phân, giống như Bernoulli nhưng dùng cho Multinomial.
  + Ưu điểm: Tránh thứ tự giả tạo (ordinal encoding không phù hợp nếu categorical không có thứ tự).
  + Nhược điểm: Tăng chiều dữ liệu nếu nhiều danh mục (curse of dimensionality) → Có thể dùng feature selection sau.
* **Đối với target variable (nhãn)**: Nếu categorical, dùng **LabelEncoder** để chuyển thành integers (0,1,2,...), vì MultinomialNB chấp nhận nhãn số.
* **Lưu ý**: Nếu dữ liệu có missing values, xử lý trước (ví dụ: fillna). Với dữ liệu lớn, dùng sparse matrix để tiết kiệm bộ nhớ.

**Các bước thực hiện**

1. **Chuẩn bị dữ liệu**: Tải hoặc tạo DataFrame với pandas. Xác định features categorical và target.
2. **Xử lý missing values** (nếu có): Điền giá trị mặc định hoặc loại bỏ.
3. **Encode categorical features**: Sử dụng OneHotEncoder từ sklearn.preprocessing.
4. **Encode target** (nếu cần): Sử dụng LabelEncoder.
5. **Chia dữ liệu**: Train/test split.
6. **Huấn luyện mô hình**: Fit MultinomialNB trên dữ liệu đã encode.
7. **Đánh giá**: Dự đoán và tính accuracy.

## 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?

**Triển khai Naive Bayes cho Phân loại Văn bản (Text Classification)**

Naive Bayes là lựa chọn phổ biến cho phân loại văn bản nhờ tốc độ cao, khả năng xử lý dữ liệu thưa thớt (sparse data) và hiệu suất tốt với giả định độc lập đặc trưng (mỗi từ là một đặc trưng độc lập). Trong text classification, chúng ta thường dùng **Multinomial Naive Bayes** vì nó phù hợp với dữ liệu đếm (count) từ ngữ, như mô hình bag-of-words (BoW). Ví dụ ứng dụng: Phân loại email spam, phân loại cảm xúc (positive/negative), hoặc phân loại tin tức.

Quy trình triển khai bao gồm:

1. **Tiền xử lý văn bản**: Chuyển chữ thường, loại bỏ dấu câu/stop words, stemming/lemmatization để giảm chiều dữ liệu.
2. **Vector hóa**: Chuyển văn bản thành ma trận số (vector) đại diện tần suất từ (CountVectorizer) hoặc trọng số (TfidfVectorizer).
3. **Huấn luyện mô hình**: Sử dụng MultinomialNB trên dữ liệu đã vector hóa.
4. **Đánh giá**: Sử dụng accuracy, precision, recall, F1-score qua cross-validation hoặc train-test split.
5. **Dự đoán**: Áp dụng trên dữ liệu mới.

Dưới đây là bảng tóm tắt các bước:

| **Bước** | **Mô tả** | **Công cụ trong Scikit-learn** |
| --- | --- | --- |
| **Tiền xử lý** | Làm sạch text: lowercase, remove punctuation/stopwords. | NLTK hoặc Scikit-learn's EnglishStopwords. |
| **Vector hóa** | Chuyển text → vector đếm/tfidf (giảm từ vựng lớn bằng max\_features). | CountVectorizer hoặc TfidfVectorizer. |
| **Huấn luyện** | Fit MultinomialNB trên X (vector), y (nhãn). | MultinomialNB(). |
| **Đánh giá** | Predict và tính metrics trên test set. | accuracy\_score, classification\_report. |

**Đoạn code mẫu**

Dưới đây là code hoàn chỉnh bằng Python với Scikit-learn, sử dụng bộ dữ liệu **20 Newsgroups** (phân loại tin tức thành 20 chủ đề). Code này tải dữ liệu mẫu, tiền xử lý đơn giản, vector hóa bằng TfidfVectorizer (tốt hơn Count cho text classification vì giảm trọng số từ phổ biến), huấn luyện và đánh giá. Độ chính xác thường đạt ~80-85%.

# Bước 1: Import thư viện

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.pipeline import Pipeline

import numpy as np

# Bước 2: Tải dữ liệu (chỉ dùng 4 categories để đơn giản; có thể dùng tất cả 20)

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med'] # 4 chủ đề mẫu

newsgroups\_train = fetch\_20newsgroups(subset='train', categories=categories, remove=('headers', 'footers', 'quotes')) # Tiền xử lý cơ bản

newsgroups\_test = fetch\_20newsgroups(subset='test', categories=categories, remove=('headers', 'footers', 'quotes'))

X\_train = newsgroups\_train.data # Văn bản huấn luyện

y\_train = newsgroups\_train.target

X\_test = newsgroups\_test.data # Văn bản test

y\_test = newsgroups\_test.target

print("Kích thước train/test:", len(X\_train), len(X\_test))

# Bước 3: Vector hóa và huấn luyện (sử dụng Pipeline để tự động hóa)

# TfidfVectorizer: Chuyển text → tfidf vector (max\_features=5000 để giảm chiều)

pipeline = Pipeline([

('tfidf', TfidfVectorizer(max\_features=5000, stop\_words='english')), # Tiền xử lý: remove stop words, lowercase mặc định

('nb', MultinomialNB(alpha=1.0)) # alpha: Laplace smoothing

])

# Fit trên train

pipeline.fit(X\_train, y\_train)

# Bước 4: Dự đoán và đánh giá

y\_pred = pipeline.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Độ chính xác trên tập test:", accuracy) # Thường ~0.85

print("\nBáo cáo phân loại:")

print(classification\_report(y\_test, y\_pred, target\_names=newsgroups\_train.target\_names))

# Bước 5: Dự đoán dữ liệu mới (ví dụ)

X\_new = ["The graphics are great and the colors are vibrant"] # Văn bản mới về comp.graphics

prediction = pipeline.predict(X\_new)

print("\nDự đoán cho văn bản mới:", newsgroups\_train.target\_names[prediction[0]]) # Output: comp.graphics

**Giải thích code**

* **fetch\_20newsgroups**: Tải dữ liệu mẫu (khoảng 1000-2000 mẫu). Tham số remove loại bỏ headers/footers để làm sạch.
* **TfidfVectorizer**: Chuyển văn bản thành ma trận tfidf (term frequency-inverse document frequency), tự động lowercase và remove stop words. max\_features=5000 giới hạn từ vựng để tránh overfitting.
* **Pipeline**: Kết hợp vector hóa và mô hình thành một quy trình, dễ dàng áp dụng transform/predict.
* **MultinomialNB(alpha=1.0)**: Alpha là tham số smoothing để tránh xác suất zero nếu từ không xuất hiện trong huấn luyện.
* **Kết quả**: Với dữ liệu đầy đủ (20 categories), accuracy ~82%. Bạn có thể cải thiện bằng cách thêm stemming (sử dụng NLTK: from nltk.stem import PorterStemmer và tùy chỉnh tokenizer trong Vectorizer).

**Lưu ý khi triển khai thực tế**

* **Tiền xử lý nâng cao**: Sử dụng NLTK hoặc spaCy cho stemming/lemmatization nếu dữ liệu tiếng Việt (thay 'english' bằng custom stop words).
* **Xử lý imbalance**: Nếu lớp không cân bằng, dùng class\_prior=True trong NB hoặc SMOTE.
* **Tối ưu**: Thử CountVectorizer nếu dữ liệu ngắn; Tfidf cho dài. Sử dụng GridSearchCV để tune alpha và max\_features.
* **Hạn chế**: Naive Bayes có thể kém nếu từ ngữ có phụ thuộc ngữ cảnh mạnh (ví dụ: sarcasm) – lúc đó dùng LSTM hoặc BERT.