**Câu hỏi ôn tập lý thuyết**

**2.1.1.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?*

CRISP-DM (Cross-Industry Standard Process for Data Mining) là một quy trình 6 bước chuẩn hóa để thực hiện dự án khai phá dữ liệu:

1. **Business Understanding** (Hiểu nghiệp vụ)
2. **Data Understanding** (Hiểu dữ liệu)
3. **Data Preparation** (Chuẩn bị dữ liệu)
4. **Modeling** (Xây dựng mô hình)
5. **Evaluation** (Đánh giá)
6. **Deployment** (Triển khai)

SEMMA là một quy trình khai phá dữ liệu 5 bước do viện SAS phát triển:

1. **Sample** (Lấy mẫu)
2. **Explore** (Khám phá)
3. **Modify** (Điều chỉnh/Biến đổi dữ liệu)
4. **Model** (Xây dựng mô hình)
5. **Assess** (Đánh giá)

*+ 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.*

Nó hoạt động giống như một sơ đồ "nếu-thì".

* Bắt đầu từ **nút gốc (root node)**, chứa toàn bộ dữ liệu.
* Thuật toán đặt một câu hỏi về một đặc trưng (ví dụ: "Tuổi > 30?"). Câu trả lời sẽ chia dữ liệu thành các **nhánh (branches)**.
* Quá trình này lặp lại ở các nút con, tiếp tục chia nhỏ dữ liệu.
* Khi thuật toán dừng lại (ví dụ, đạt độ sâu tối đa hoặc các nút đã "thuần khiết"), nút cuối cùng được gọi là **nút lá (leaf node)**. Nút lá chứa dự đoán cuối cùng (ví dụ: "Sống sót" hoặc "Không sống sót").

*+ 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?*

Đây là các thước đo toán học được sử dụng để quyết định xem câu hỏi (đặc trưng) nào là tốt nhất để phân tách dữ liệu tại một nút.

* **Entropy/Gini Index:** Cả hai đều đo lường "độ hỗn loạn" (impurity) của một nút. Một nút "thuần khiết" (ví dụ: 100% Sống sót) có Gini/Entropy bằng 0. Một nút hỗn loạn (50% Sống, 50% Chết) có Gini/Entropy cao nhất.
* **Information Gain (Độ lợi thông tin):** Đo lường mức độ *giảm* Entropy (hoặc Gini) sau khi tách bằng một đặc trưng. Cây quyết định luôn chọn đặc trưng nào mang lại **Information Gain cao nhất** (tức là làm cho các nút con "thuần khiết" nhất có thể).

*+ 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?*

**Random Forest** là một thuật toán *ensemble* (tập hợp), nó xây dựng *rất nhiều* cây quyết định riêng lẻ (một "khu rừng").

**Khác biệt:**

1. Mỗi cây trong rừng chỉ được huấn luyện trên một *mẫu ngẫu nhiên* (có lặp lại) của dữ liệu (gọi là *bagging*).
2. Tại mỗi nút, thay vì kiểm tra *tất cả* các đặc trưng, cây chỉ được phép kiểm tra một *tập con ngẫu nhiên* của các đặc trưng.

Một cây quyết định đơn lẻ rất dễ bị **overfitting** (học thuộc lòng dữ liệu huấn luyện). Random Forest kết hợp dự đoán của hàng trăm cây "yếu" và "khác biệt" nhau, lấy kết quả trung bình (hoặc đa số). Điều này giúp *giảm overfitting* đáng kể và mang lại độ chính xác tổng thể cao hơn.

*+ 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?*

|  |  |  |
| --- | --- | --- |
|  | **Cây quyết định (DT)** | **Random Forest (RF)** |
| *Ưu điểm* | Dễ hiểu, dễ giải thích (trực quan như flowchart), nhanh chóng khi dự đoán. | Độ chính xác rất cao, kháng overfitting tốt, ổn định. |
| *Nhược điểm* | Rất dễ bị **overfitting**, không ổn định (một thay đổi nhỏ trong dữ liệu có thể tạo ra một cây hoàn toàn khác). | Là một mô hình "hộp đen" (khó giải thích *tại sao* nó đưa ra dự đoán), tốn nhiều tài nguyên và thời gian hơn để huấn luyện. |

*+ 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*

from sklearn.tree import DecisionTreeClassifier

# 1. Khởi tạo mô hình (ví dụ: giới hạn độ sâu)

dt\_model = DecisionTreeClassifier(max\_depth=5, random\_state=42)

# 2. Huấn luyện (fit) mô hình

dt\_model.fit(X\_train, y\_train)

# 3. Dự đoán

y\_pred = dt\_model.predict(X\_test)

*+ 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)?*

from sklearn.ensemble import RandomForestClassifier

# 1. Khởi tạo mô hình

# n\_estimators: Số lượng cây quyết định trong rừng (ví dụ: 100)

# max\_depth: Độ sâu tối đa của MỖI cây

rf\_model = RandomForestClassifier(n\_estimators=100, max\_depth=5, random\_state=42)

# 2. Huấn luyện

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

# 3. Dự đoán

y\_pred\_rf = rf\_model.predict(X\_test)

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

import pandas as pd

# Lấy danh sách tên các đặc trưng (ví dụ: ['Age', 'Sex', 'Fare'])

feature\_names = X\_train.columns

# Lấy độ quan trọng

importances = rf\_model.feature\_importances\_

# Tạo DataFrame để xem

feat\_imp\_df = pd.DataFrame({'Feature': feature\_names, 'Importance': importances})

# Sắp xếp để xem đặc trưng quan trọng nhất

print(feat\_imp\_df.sort\_values(by='Importance', ascending=False))

*+ 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*

Đây là kỹ thuật tự động tìm kiếm bộ tham số tốt nhất.

1. Bạn định nghĩa một "lưới" (grid) các tham số muốn thử (ví dụ: {'n\_estimators': [50, 100, 200], 'max\_depth': [3, 5, 10]}).
2. GridSearchCV sẽ huấn luyện và đánh giá (thông qua *cross-validation* ) *tất cả các tổ hợp* (ví dụ: 3x3 = 9 mô hình) để tìm ra tổ hợp tốt nhất.
3. RandomizedSearchCV tương tự nhưng chỉ thử một số lượng tổ hợp *ngẫu nhiên* (nhanh hơn khi có nhiều tham số).

**2.2.1. 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)*

SVM tìm một **hyperplane** (siêu phẳng, hay ranh giới phân tách) để chia các điểm dữ liệu thuộc các lớp khác nhau. Mục tiêu của SVM là tìm ra hyperplane có **margin** (lề) *rộng nhất*. Margin là khoảng cách từ hyperplane đến các điểm dữ liệu gần nó nhất của cả hai lớp.

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

**Vector hỗ trợ** là các điểm dữ liệu nằm *ngay trên* đường biên của margin. Chúng là những điểm quan trọng nhất, vì chúng *một mình* xác định vị trí và độ rộng của margin. Nếu bạn di chuyển các điểm khác (không phải vector hỗ trợ), hyperplane sẽ không thay đổi.

*+ 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:** Yêu cầu tất cả các điểm phải được phân loại chính xác 100% và không có điểm nào nằm *trong* lề. Cách này chỉ hoạt động khi dữ liệu *phân tách tuyến tính hoàn hảo* và rất nhạy cảm với outliers.
* **Soft Margin:** Cho phép một số điểm dữ liệu nằm *bên trong* lề hoặc thậm chí bị *phân loại sai*. Đây là cách dùng trong thực tế, giúp mô hình linh hoạt hơn và ít bị ảnh hưởng bởi outliers.

*Vậy khi nào ta nên sử dụng lề mềm? Thực tế, chúng ta hầu như luôn sử dụng lề mềm, khi:*

1. **Dữ liệu có nhiễu (Outliers):** Đây là trường hợp phổ biến nhất. Dữ liệu thực tế không bao giờ hoàn hảo. Lề Mềm đủ thông minh để "phớt lờ" các điểm nhiễu này và tìm ra ranh giới phù hợp với phần lớn dữ liệu.
2. **Dữ liệu không phân tách tuyến tính:** Khi hai lớp dữ liệu bị xen kẽ, chồng chéo lên nhau. Lề Cứng sẽ không thể tìm được ranh giới, nhưng Lề Mềm sẽ tìm ra ranh giới "ít sai nhất".

*+ 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)** chính là một "mánh khóe" toán học (gọi là *kernel trick*). Thay vì cố kẻ đường thẳng trên tờ giấy phẳng, bạn "uốn cong" tờ giấy (hoặc hất các chấm lên không trung) . Trong không gian 3D mới này, các chấm đỏ có thể nằm ở dưới thấp và các chấm xanh nằm ở trên cao. Bây giờ, việc chia tách chúng trở nên cực kỳ dễ dàng: bạn chỉ cần đặt một mặt phẳng (hyperplane) cắt ngang giữa chúng.

Các loại Kernel phổ biến:

**1. Linear (Tuyến tính):**

* **Giải thích:** Đây là kernel đơn giản nhất. Nó không "uốn cong" không gian mà chỉ cố gắng tìm một đường thẳng (trong 2D) hoặc một mặt phẳng (trong 3D) để phân chia dữ liệu.
* **Khi nào dùng:**
  + Khi dữ liệu của bạn vốn đã **phân tách tuyến tính** (bạn có thể nhìn thấy một đường thẳng rõ ràng chia tách chúng).
  + Khi bạn có **số lượng đặc trưng rất lớn** (ví dụ: hàng chục nghìn đặc trưng như trong phân loại văn bản). Kernel linear thường nhanh hơn RBF trong trường hợp này.

**2. Polynomial (Đa thức):**

* **Giải thích:** Kernel này tạo ra các ranh giới có dạng **cong, uốn lượn** (giống như đường cong y = x^2 hoặc x^3). Nó phức tạp hơn linear nhưng không linh hoạt bằng RBF.
* **Khi nào dùng:**
  + Khi bạn có lý do tin rằng mối quan hệ giữa các đặc trưng của bạn tuân theo một dạng đa thức.
  + Nó không còn được sử dụng phổ biến bằng RBF vì RBF thường cho kết quả tốt hơn.

**3. RBF (Radial Basis Function):**

* **Giải thích:** Đây là kernel **phổ biến và mạnh mẽ nhất**3. Nó tạo ra các ranh giới phi tuyến rất phức tạp, giống như các "hòn đảo" hoặc các đường viền linh hoạt. Nó hoạt động bằng cách xem xét "khoảng cách" của các điểm, gán tầm ảnh hưởng lớn hơn cho các điểm ở gần.
* **Khi nào dùng:**
  + **Hầu hết mọi lúc.** Đây là lựa chọn mặc định khi bạn không biết gì về cấu trúc dữ liệu của mình.
  + Khi dữ liệu của bạn bị **trộn lẫn phức tạp** và không thể chia tách bằng đường thẳng. (Như ví dụ bánh donut ở trên).

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

Tham số C (Cost) kiểm soát sự cân bằng giữa việc *tối đa hóa lề* (margin rộng) và *giảm thiểu lỗi phân loại* (ít điểm bị sai).

* **C thấp:** Ưu tiên lề rộng, chấp nhận nhiều điểm bị phân loại sai hơn (dễ bị *underfitting*).
* **C cao:** Ưu tiên phân loại đúng nhiều điểm nhất có thể, chấp nhận lề hẹp hơn (dễ bị *overfitting*).

*+ 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*

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

# 1. (RẤT QUAN TRỌNG) Chuẩn hóa dữ liệu

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# 2. Khởi tạo mô hình (ví dụ dùng RBF)

svm\_model = SVC(kernel='rbf', C=1.0, random\_state=42)

# 3. Huấn luyện

svm\_model.fit(X\_train\_scaled, y\_train)

# 4. Dự đoán

y\_pred = svm\_model.predict(X\_test\_scaled)

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

* **Hàm:** StandardScaler (phổ biến nhất, đưa về mean=0, std=1) hoặc MinMaxScaler (đưa về khoảng [0, 1]).
* **Tại sao quan trọng:** SVM hoạt động dựa trên *khoảng cách* giữa các điểm để tìm margin. Nếu một đặc trưng có thang đo lớn (ví dụ: Lương: 10,000,000 - 100,000,000) và đặc trưng khác có thang đo nhỏ (ví dụ: Tuổi: 20-60), thì đặc trưng "Lương" sẽ *hoàn toàn chi phối* phép tính khoảng cách, khiến đặc trưng "Tuổi" bị phớt lờ. Chuẩn hóa để đảm bảo tất cả các đặc trưng đều có tầm ảnh hưởng như nhau.

**2.3.1 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?*

* **Định lý Bayes:** Thuật toán tính toán *xác suất* của một lớp (ví dụ: "Thư rác") *khi biết* các đặc trưng (ví dụ: các từ trong email). Nó tính **P(Lớp | Đặc trưng)** cho mọi lớp và chọn lớp có xác suất cao nhất.
* **Giả định "Ngây thơ" (Naïve):** Đây là giả định quan trọng nhất (và cũng là điểm yếu nhất). Thuật toán "ngây thơ" tin rằng tất cả các đặc trưng là *độc lập* với nhau. Ví dụ: nó giả định từ "mua" và từ "ngay" xuất hiện trong email một cách độc lập, mặc dù thực tế chúng thường đi kèm với nhau.

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

Chúng được chọn dựa trên loại dữ liệu của đặc trưng (X):

* **GaussianNB:** Dùng cho các đặc trưng *liên tục* (ví dụ: Tuổi, Lương). Nó giả định rằng dữ liệu của đặc trưng đó tuân theo phân phối Gaussian (hình chuông).
* **MultinomialNB:** Dùng cho các đặc trưng *đếm* (ví dụ: số lần một từ xuất hiện trong văn bản, như trong bài mẫu email spam ).
* **BernoulliNB:** Dùng cho các đặc trưng *nhị phân* (0/1 hoặc True/False) (ví dụ: từ đó *có* xuất hiện trong văn bản hay *không*?).

*+ 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?*

* Nó được gọi là "ngây thơ" vì giả định rằng các đặc trưng là độc lập (ví dụ: màu sắc của nấm độc lập với mùi của nấm) gần như luôn *sai* trong thực tế.
* **Ảnh hưởng:** Mặc dù giả định này sai, Naive Bayes vẫn hoạt động *tốt đáng ngạc nhiên* trong nhiều bài toán (đặc biệt là phân loại văn bản). Nó rất nhanh và hiệu quả.

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

* **Ưu điểm:** Rất nhanh (cả huấn luyện và dự đoán), yêu cầu ít dữ liệu huấn luyện, hoạt động tốt với dữ liệu đa chiều (nhiều đặc trưng, như văn bản).
* **Hạn chế:** Giả định độc lập "ngây thơ" là phi thực tế. Nếu một đặc trưng trong tập test chưa bao giờ xuất hiện trong tập train, nó có thể gây ra xác suất bằng 0.

*+ 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*

from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler

# 1. (Tùy chọn nhưng nên làm) Chuẩn hóa dữ liệu

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# 2. Khởi tạo mô hình

nb\_model = GaussianNB()

# 3. Huấn luyện

nb\_model.fit(X\_train\_scaled, y\_train)

# 4. Dự đoán

y\_pred = nb\_model.predict(X\_test\_scaled)

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

Bạn phải chuyển dữ liệu phân loại thành *dữ liệu đếm số*. Cách phổ biến nhất (như trong bài mẫu spam) là sử dụng CountVectorizer. CountVectorizer sẽ tạo ra một ma trận, trong đó mỗi hàng là một tài liệu (email) và mỗi cột là một từ, giá trị trong ô là *số lần* từ đó xuất hiện.

*+ 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?*

**Giai đoạn 1: Biến văn bản thành đặc trưng (Vector hóa)**

Mô hình Naive Bayes không thể "đọc" văn bản. Nó chỉ hiểu được các con số. Vì vậy, bước đầu tiên là phải chuyển đổi mỗi văn bản (mỗi email) thành một vector số. Phương pháp phổ biến nhất là **"Túi từ" (Bag-of-Words)**.

1. **Xây dựng từ điển (Vocabulary):** Đầu tiên, mô hình sẽ quét toàn bộ tập dữ liệu huấn luyện (TẤT CẢ các email) và tạo ra một danh sách (từ điển) chứa MỌI từ duy nhất xuất hiện (ví dụ: "bạn", "tôi", "mua", "ngay", "khuyến mãi", "viagra"...).
2. **Đếm từ (Vector hóa):** Với mỗi email, mô hình sẽ tạo ra một vector (một hàng số) dựa trên từ điển đó. Vector này biểu thị số lần mỗi từ trong từ điển xuất hiện trong email đó.

Ví dụ:

* *Từ điển:* ["bạn", "khuyến mãi", "mua", "viagra"]
* *Email A:* "bạn ơi, mua ngay" -> Vector: [1, 0, 1, 0]
* *Email B:* "mua viagra khuyến mãi" -> Vector: [0, 1, 1, 1]

Lúc này, văn bản đã trở thành các đặc trưng (features) mà máy tính có thể hiểu được. Mô hình được sử dụng cho loại đặc trưng "đếm" này chính là **Multinomial Naive Bayes**2222.

**Giai đoạn 2: Áp dụng Định lý Bayes "Ngây thơ"**

Bây giờ chúng ta có các vector số (đặc trưng) và nhãn (Spam/Không Spam).

**1. Mục tiêu (Định lý Bayes)**

Mục tiêu của chúng ta là tính xác suất xem một email MỚI (với các từ bên trong nó) thuộc lớp nào là cao hơn. Chúng ta muốn so sánh:

* P(Spam | Email mới)
* P(Không Spam | Email mới)

Theo Định lý Bayes, công thức này được "lật ngược" lại (vì tính toán dễ hơn):

P(Lớp | Email) P(Email | Lớp) x P(Lớp)

**2. Giả định "Ngây thơ"**

Phần khó nhất là tính P(Email | Lớp), ví dụ: P("mua viagra khuyến mãi" | Spam).

Đây là lúc sự "ngây thơ" phát huy tác dụng. Thuật toán *giả định rằng sự xuất hiện của các từ là hoàn toàn độc lập với nhau* (đây là lý do nó "ngây thơ")4.

Thay vì tính xác suất phức tạp của cả câu, nó chia nhỏ ra:

P("mua viagra khuyến mãi" | Spam) = P("mua" | Spam) x P("viagra" | Spam) x P("khuyến mãi" | Spam)

Điều này làm bài toán trở nên cực kỳ đơn giản.

**3. Quá trình huấn luyện (Học xác suất)**

Trong quá trình huấn luyện, mô hình sẽ học 2 loại xác suất từ dữ liệu:

1. **Xác suất tiên nghiệm P(Lớp):** Đây là xác suất chung của một lớp.

P(Spam) =

1. **Xác suất hiện hữu P(Từ | Lớp):** Đây là xác suất một từ cụ thể xuất hiện, *biết rằng* nó nằm trong một lớp nhất định.

P("viagra" | Spam) =

Mô hình sẽ tính toán và lưu lại các xác suất này cho MỌI từ trong từ điển, cho MỌI lớp.

**4. Quá trình dự đoán (Tính điểm)**

Khi một email mới (ví dụ: "mua ngay viagra") đến, mô hình sẽ thực hiện hai phép tính:

1. Điểm Spam:

Score(Spam) = P(Spam) x P("mua" | Spam) x P("ngay" | Spam) x P("viagra" | Spam)

1. Điểm Không Spam:

Score(Không Spam) = P(Không Spam) x P("mua" | Không Spam) x P("ngay" | Không Spam) x P("viagra" | Không Spam)

Cuối cùng, mô hình so sánh hai "Điểm" này. Điểm nào cao hơn thì email sẽ được gán nhãn đó. Đó là lý do Naive Bayes cực kỳ nhanh và hiệu quả cho phân loại văn bản.