**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỆ THỐNG THÔNG TIN**

---------



**ĐỒ ÁN MÔN: KHAI THÁC DỮ LIỆU**

**CÁC THUẬT TOÁN KHAI THÁC DỮ LIỆU**

**Giảng viên:**

Mai Xuân Hùng

**Sinh viên thực hiện:**

1. Trần Phi Thành – 22521365
2. Dương Văn Súa – 22521267
3. Nguyễn Mai Thanh Thảo – 22521370
4. Huỳnh Lê Minh Thành – 22521346

Thành phố Hồ Chí Minh, tháng 12 năm 2024

**LỜI CẢM ƠN**

Trong quá trình hoàn thành báo cáo đồ án về các thuật toán trong môn khai thác dữ liệu, em xin bày tỏ lòng biết ơn sâu sắc đến giảng viên Mai Xuân Hùng. Thầy đã tận tâm giảng dạy và hướng dẫn chúng em không chỉ trong lý thuyết mà còn trong thực hành, giúp chúng em từng bước nắm vững kiến thức và kỹ năng cần thiết. Những bài giảng sinh động và những ví dụ thực tế của thầy đã mở ra cho em cái nhìn sâu sắc về lĩnh vực khai thác dữ liệu, khơi dậy niềm đam mê nghiên cứu và khám phá trong sinh viên chúng em.

Bên cạnh đó, sự hỗ trợ tận tình của các cán bộ trong khoa cũng đã góp phần không nhỏ vào quá trình học tập của nhóm em. Sự sẵn sàng giúp đỡ và chia sẻ kinh nghiệm của các thầy cô đã giúp các bạn sinh viên như chúng em vượt qua những khó khăn trong quá trình thực hiện đồ án.

Em cũng xin cảm ơn Trường Đại Học Công Nghệ Thông Tin, Khoa Hệ Thống Thông Tin đã tạo ra một môi trường học tập chuyên nghiệp và đầy đủ trang thiết bị. Những điều kiện học tập tốt nhất đã giúp em phát triển kiến thức và kỹ năng cần thiết cho tương lai.

Cuối cùng, chúng em xin chân thành cảm ơn thầy đã góp phần vào hành trình học tập của chúng em. Những kinh nghiệm và bài học quý giá sẽ theo chúng em trong suốt quá trình học tập và làm việc sau này.

Một lần nữa, chúng em xin chân thành cảm ơn!

Nhóm sinh viên thực hiện

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

**MỤC LỤC**

[DANH MỤC HÌNH ẢNH 2](#_Toc184599798)

[NỘI DUNG THỰC HIỆN 3](#_Toc184599799)

[IMPLEMENTATION CONTENT 5](#_Toc184599800)

[CHƯƠNG 1: MỞ ĐẦU 7](#_Toc184599801)

[1. Tổng quan về Khai phá dữ liệu. 7](#_Toc184599802)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 8](#_Toc184599803)

[1. Tập phổ biến và luật kết hợp. 8](#_Toc184599804)

[2. Tập thô. 9](#_Toc184599805)

[3. Phân lớp. 12](#_Toc184599806)

[4. Gom cụm. 16](#_Toc184599807)

[CHƯƠNG 3: CÀI ĐẶT 22](#_Toc184599808)

[1. Tập phổ biến và luật kết hợp. 22](#_Toc184599809)

[2. Tập thô. 26](#_Toc184599810)

[3. Phân lớp. 32](#_Toc184599811)

[4. Gom cụm. 43](#_Toc184599812)

[TÀI LIỆU THAM KHẢO 53](#_Toc184599813)

# 

# DANH MỤC HÌNH ẢNH

[Hình 1.1: Công thức cập nhật trọng số cát nút trong vùng lân cận của noron chiến thắng 23](#_Toc184681618)

[Hình 2.1: Bộ dữ liệu tập phổ biến và luật kết hợp 24](#_Toc184681619)

[Hình 2.2: Giao diện và kết quả web bằng Django 27](#_Toc184681620)

[Hình 2.3: Giao diện của tập phổ biến và luật kết hợp 27](#_Toc184681621)

[Hình 2.4: Kết quả sau khi nhập minsup và minconf 28](#_Toc184681622)

[Hình 2.5: Bộ dữ liệu để tìm quan hệ bất khả phân biệt và xấp xỉ tập hợp 29](#_Toc184681623)

[Hình 2.6: Bộ dữ liệu để rút gọn 29](#_Toc184681624)

[Hình 2.7: Giao diện của tập thô 33](#_Toc184681625)

[Hình 2.8: Kết quả thực hiện tập thô 34](#_Toc184681626)

[Hình 2.9: Bộ dữ liệu của Cây quyết định và Naive Bayes 35](#_Toc184681627)

[Hình 2.10: Giao diện của Cây quyết định 41](#_Toc184681628)

[Hình 2.11: Kết quả thực hiện Cây quyết định 41](#_Toc184681629)

[Hình 2.12: Giao diện của Naive Bayes 46](#_Toc184681630)

[Hình 2.13: Kết quả thực hiện Navie Bayes 46](#_Toc184681631)

[Hình 2.14: Bộ dữ liệu của K-Means và Kohonen 47](#_Toc184681632)

[Hình 2.15: Giao diện của K-Means 50](#_Toc184681633)

[Hình 2.16: Kết quả thực hiện K-Means 50](#_Toc184681634)

[Hình 2.17: Giao diện của Kohonen 54](#_Toc184681635)

[Hình 2.18: Kết quả thực hiện Kohonen 54](#_Toc184681636)

# NỘI DUNG THỰC HIỆN

Trong đồ án này, chúng tôi đã thực hiện việc lập trình và áp dụng các thuật toán khai thác dữ liệu mà chúng em đã học, bao gồm: tập phổ biến và luật kết hợp, tập thô, phân lớp (cây quyết định và Naive Bayes), cũng như các phương pháp gom cụm như K-means và Kohonen. Dưới đây là chi tiết về từng phần:

**Tập Phổ Biến và Luật Kết Hợp:**

**Dữ liệu:** Lịch sử mua hàng tại một tiệm tạp hóa, bao gồm ba cột: mã khách hàng (Member\_number), ngày mua (Date), và món đồ mua (itemDescription.

**Thuật toán:** dùng Apriori để tìm ra các tập phổ biến và luật kết hợp giữa các tập

**Giao diện:** có 2 ô để nhập các giá trị minsup, minconf và một nút để chạy kết quả.

**Tập thô:**

**Dữ liệu:**

* Tương tự trong bài giảng môn học với 3 cột là số tuổi (Age), số buổi (Sesions) và thi đậu (Pass).
* Có trong bài giảng môn học gồm 6 cột là Tên người, Màu tóc, Chiều cao, Cân nặng, Dùng thuốc và Kết quả

**Thuật toán:** dùng các hàm như *ind*, *lower\_approximation*, *upper\_approximation*,… để tìm quan hệ bất khả phân biệt, tính xấp xỉ và rút gọn.

**Giao diện:** Có 2 phần tính xấp xỉ và tính rút gọn tập thể. Mỗi phần đều gồm 2 nút

là Xem dữ liệu và Xem kết quả.

**Cây quyết định:**

**Dữ liệu:** loại thuốc các bệnh nhân sẽ dùng (Drug), được phân loại bởi tuổi (Age), giới tính (Sex), huyết áp (BP), nồng độ cholesterol (Cholesterol), chỉ số natri và kali (Na\_to\_K).

**Thuật toán:** ID3 và dùng chỉ mục Gini để xây dựng cây quyết định, phân tích và dự đoán loại thuốc bệnh nhân sẽ dùng

**Giao diện:** gồm 2 nút là Xem dữ liệu và Xem kết quả.

**Naive Bayes:**

**Dữ liệu:** dùng bộ dữ liệu tương tự như thuật toán ID3

**Thuật toán:** thực hiện huấn luyện và dự đoán bằng mô hình Naive Bayes.

**Giao diện:** gồm nút xem dữ liệu và một ô nhập dữ liệu. bấm nút Chạy kết quả để phân lớp người đó xem người đó sử dụng loại thuốc nào.

**K-Means**

**Dữ liệu:** mức độ chi tiêu của khách hàng ở trung tâm mua sắm. Bộ dữ liệu chứa các thông tin được trung tâm tổng hợp lại gồm mã khách hàng (CustomerID), giới tính (Genre), tuổi (Age), thu nhập hàng năm (Annual\_Income\_(k$)) và mức độ chi tiêu (Spending\_Score)

**Thuật toán:** thực hiện các hàm như *initialize \_centroids, assign\_clusters, update\_centroids, kmeans,..* và dùng hai đặc trưng là thu nhập hàng năm cùng với mức độ chi tiêu để gom cụm.

**Giao diện:** một nút xem dữ liệu, một ô để nhập số lượng cụm cần để gom cụm và nút Chạy kết quả.

**Kohonen**

**Dữ liệu:** dùng bộ dữ liệu tương tự như K-Means

**Thuật toán:** thiết kế các hàm như *initialize\_som, euclidean\_distance, find\_bmu, update\_weights,..* để huấn luyện mạng tự tổ chức **SOM** (Self-Organizing Map) nhằm phân nhóm các điểm dữ liệu

**Giao diện:** thiết kế để nhập số lượng cột, hàng, tốc độ học, số lần lặp cũng như bán kính.

# IMPLEMENTATION CONTENT

In this project, we programmed and applied the data mining algorithms we have learned, including: frequent itemsets and association rules, rough sets, classification (decision trees and Naive Bayes), as well as clustering methods such as K-means and Kohonen. Below are the details of each part:

**Frequent Itemsets and Association Rules:**

**Data**: Transaction history from a grocery store, including three columns: customer ID (Member\_number), purchase date (Date), and purchased item (itemDescription).

**Algorithm:** Used Apriori to identify frequent itemsets and association rules among the itemsets.

**Interface:** Includes two input fields for minsup and minconf values, and a button to execute the results.

**Rough Sets:**

**Data:**

* Similar to the course lecture with three columns: age (Age), number of sessions (Sessions), and pass/fail status (Pass).
* Contains six columns from the course lecture: Name, Hair Color, Height, Weight, Medication Use, and Result.

**Algorithm**: Utilized functions such as ind, lower\_approximation, upper\_approximation, etc., to find indistinguishable relations, compute approximations, and perform reduction.

**Interface**: Comprises two sections for approximation and set reduction. Each section includes two buttons: View Data and View Results.

**Decision Trees:**

**Data**: Types of medication patients will use (Drug), classified by age (Age), gender (Sex), blood pressure (BP), cholesterol levels (Cholesterol), and sodium and potassium levels (Na\_to\_K).

**Algorithm:** Implemented ID3 and used the Gini index to construct the decision tree, analyze, and predict the medication type for patients.

**Interface:** Includes two buttons: View Data and View Results.

**Naive Bayes:**

**Data:** Used a dataset similar to the one for the ID3 algorithm.

**Algorithm:** Conducted training and prediction using the Naive Bayes model.

**Interface:** Includes a button to view data and an input field for data entry. Pressing the Run button classifies individuals to determine which medication they use.

**K-Means:**

**Data:** Customer spending levels at a shopping center. The dataset contains summarized information including customer ID (CustomerID), gender (Genre), age (Age), annual income (Annual\_Income\_(k$)), and spending score (Spending\_Score).

**Algorithm:** Implemented functions such as initialize\_centroids, assign\_clusters, update\_centroids, kmeans, etc., using annual income and spending score as features for clustering.

**Interface:** Includes a button to view data, an input field for the number of clusters needed for clustering, and a Run button.

**Kohonen:**

**Data:** Utilized a dataset similar to that used in K-Means.

**Algorithm:** Designed functions such as initialize\_som, euclidean\_distance, find\_bmu, update\_weights, etc., to train a Self-Organizing Map (SOM) for clustering data points.

**Interface:** Designed to input the number of columns, rows, learning rate, number of iterations, and radius.

# CHƯƠNG 1: MỞ ĐẦU

## Tổng quan về Khai phá dữ liệu.

* 1. **Giới Thiệu Khai Phá Dữ Liệu.**

Khai phá dữ liệu (Data Mining) là một lĩnh vực quan trọng, liên quan đến việc rút trích thông tin hữu ích từ các khối dữ liệu lớn, bao gồm các cơ sở dữ liệu khổng lồ và dữ liệu từ Internet. Quá trình này nhằm phân tích và giải thích dữ liệu một cách bán tự động, giúp phát hiện các mẫu và tri thức tiềm ẩn trong dữ liệu.

* 1. **Định Nghĩa Khai Phá Dữ Liệu.**

Khai phá dữ liệu được xem là một bước trong quá trình Khám Phá Tri Thức trong Cơ Sở Dữ Liệu (KDD). Quy trình KDD bao gồm ba bước chính: Tiền xử lý, Khai phá dữ liệu, và Hậu xử lý. Mục tiêu của khai phá dữ liệu là cung cấp tri thức hỗ trợ ra quyết định, dự báo và khái quát hóa dữ liệu.

* 1. **Ứng Dụng Tiềm Năng.**

Các ứng dụng của khai phá dữ liệu rất đa dạng, bao gồm phân tích dữ liệu để hỗ trợ ra quyết định trong quản lý thị trường, phân tích rủi ro, và quản lý sai hỏng. Ngoài ra, nó còn được áp dụng trong khai thác web, khai thác văn bản, và nhiều lĩnh vực khác.

* 1. **Tiến Trình Khai Phá Dữ Liệu.**

Tiến trình khai phá dữ liệu bắt đầu với việc nghiên cứu lĩnh vực và xác định vấn đề cần giải quyết. Dữ liệu sau đó được làm sạch và mã hóa, tiếp theo là chọn tác vụ khai thác dữ liệu và các thuật toán phù hợp. Cuối cùng, tri thức được biểu diễn và đánh giá để áp dụng vào thực tế.

* 1. **Từ Dữ Liệu Đến Quyết Định.**

Quá trình chuyển đổi từ dữ liệu đến quyết định liên quan đến việc biến đổi dữ liệu thành thông tin và tri thức. Dữ liệu là những trường thông tin riêng lẻ, trong khi thông tin là mối liên hệ giữa các thành phần của dữ liệu. Tri thức là những quy luật phổ biến được rút ra từ thông tin, giúp đưa ra quyết định.

* 1. **Kỹ Thuật Khai Phá Dữ Liệu.**

Các kỹ thuật khai phá dữ liệu bao gồm tập phổ biến và luật kết hợp, khai thác mẫu tuần tự, phân lớp dữ liệu, và gom cụm. Mỗi kỹ thuật có mục đích và ứng dụng riêng, từ việc tìm kiếm các thuộc tính phổ biến đến phân loại và nhóm các đối tượng dựa trên sự tương đồng.

* 1. **Kết luận.**

Khai phá dữ liệu là một quy trình khám phá tiềm năng trong các cơ sở dữ liệu lớn, hỗ trợ ra quyết định và dự đoán xu hướng. Các kỹ thuật khai thác dữ liệu giúp phát hiện các mẫu và tri thức có giá trị, từ đó nâng cao hiệu quả trong nhiều lĩnh vực như kinh doanh, y tế, và khoa học.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## Tập phổ biến và luật kết hợp.

* 1. **Giới thiệu luật kết hợp.**

Luật kết hợp là công cụ trong khai phá dữ liệu, cho phép xác định mối quan hệ giữa các mặt hàng trong giao dịch. Luật có dạng ***X ⇒ Y*,** trong đó ***X*** là tập mặt hàng (vế trái) và ***Y*** là tập mặt hàng (vế phải). Luật kết hợp được định nghĩa thông qua hai thành phần quan trọng: **Support (Độ phổ biến)** và **Confidence (Độ tin cậy)**.

Độ phổ biến được tính bằng công thức:

**SP(*S*) =**

Độ tin cậy của luật kết hợp được tính như sau:

**CF(*X⇒Y*) =**

* 1. **Ứng dụng của Luật kết hợp.**

Việc phân tích tập phổ biến giúp doanh nghiệp hiểu rõ xu hướng mua sắm của khách hàng. Từ đó, họ có thể xây dựng các chiến lược bố trí hàng hóa hợp lý, tối ưu hóa không gian bán hàng để thu hút khách hàng hơn. Ngoài ra, phân tích này còn hỗ trợ trong việc dự đoán lượng hàng nhập trong tương lai, giúp cải thiện quy trình quản lý tồn kho. Đối với các doanh nghiệp bán hàng trực tuyến, việc phân tích dữ liệu giỏ hàng rất quan trọng. Nó không chỉ giúp bố trí giao diện các mặt hàng một cách hợp lý mà còn cho phép họ loại bỏ hoặc thêm mới các mặt hàng, từ đó nâng cao trải nghiệm mua sắm của khách hàng và tăng doanh thu.

* 1. **Tập phổ biến.**

Là những tập có độ độ phổ biến lớn hơn hoặc bằng 1 ngưỡng cho trước là minsupp. Cách tìm tập phổ biến:

* **Bước1: Tìm tập phổ biến thỏa ngưỡng minsupp:** Xác định tỷ lệ tối thiểu mà các tập hợp mặt hàng phải đạt được để được xem là phổ biến. Thông thường, giá trị này được xác định dựa trên yêu cầu của bài toán hoặc mục tiêu phân tích.
* **Bước 2: Thành lập ma trận nhị phân:** Thiết lập ma trận nhị phân, trong đó mỗi hàng đại diện cho một giao dịch và mỗi cột đại diện cho một yếu tố. Giá trị trong ma trận sẽ là 1 nếu giao dịch đó chứa yếu tố tương ứng, và 0 nếu không.
* **Bước 3. Tìm tập phổ biến thỏa ngưỡng:** So sánh độ phổ biến của từng yếu tố với ngưỡng ***minsupp***. Chỉ giữ lại những yếu tố có độ phổ biến lớn hơn hoặc bằng ngưỡng này. Tiếp tục áp dụng quy trình tính toán độ phổ biến cho các tập hợp có kích thước lớn hơn cho đến khi không còn tập nào đạt yêu cầu.
  1. **Thuật toán Apriori.**

Thuật toán Apriori là một kỹ thuật mạnh mẽ trong khai phá dữ liệu, được sử dụng để tìm các tập phổ biến và luật kết hợp. Nó dựa trên nguyên tắc rằng một tập con của một tập phổ biến cũng phải là một tập phổ biến.

Các bước thực hiện:

* **Bước 1: Xác định ngưỡng phổ biến tối thiểu (minsupp**): Đặt giá trị cho ngưỡng hỗ trợ tối thiểu mà các tập hợp cần đạt được để được xem là phổ biến.
* **Bước 2: Tạo tập ứng cử viên đầu tiên (C1):** Khởi tạo tập đầu tiên, thường là các yếu tố đơn lẻ.
* **Bước 3: Tính toán độ phổ biến cho các yếu tố trong C1**: Đánh giá độ phổ biến của từng yếu tố và lọc ra những yếu tố đạt yêu cầu minsupp.
* **Bước 4: Tạo tập ứng cử viên tiếp theo (Ck):** Sử dụng tập phổ biến hiện có để tạo ra các tập mới bằng cách kết hợp các tập phổ biến lại với nhau.
* **Bước 5: Lọc các tập ứng cử viên**: Tính toán độ phổ biến cho các tập và giữ lại những tập có độ phổ biến lớn hơn hoặc bằng ngưỡng minsupp.
* **Bước 6: Lặp lại quy trình**: Tiếp tục lặp lại các bước trên cho đến khi không còn tập nào có thể được tạo ra.

## Tập thô.

* 1. **Giới thiệu tập thô.**

Tập thô được sử dụng để khắc phục hiện tượng dữ liệu bị nhiễu trong quá trình khai phá dữ liệu (KPDL). Phương pháp này giúp rút gọn dữ liệu bằng cách loại bỏ các phần dữ liệu thừa, từ đó cải thiện chất lượng phân tích. Ngoài ra, tập thô còn hỗ trợ việc tạo ra các luật quyết định, giúp xác định rõ ràng mối quan hệ giữa các thuộc tính và kết quả.

Bên cạnh đó, nó cũng cho phép nhận diện các phụ thuộc riêng phần và toàn phần của các thuộc tính, giúp tăng cường khả năng hiểu biết về cấu trúc dữ liệu và các yếu tố ảnh hưởng đến quyết định.

* 1. **Hệ thông tin, hệ quyết định.**

Hệ thông tin (Information System) được định nghĩa là một cặp (U, A), trong đó:

* U là tập không rỗng các đối tượng.
* A là tập hữu hạn các thuộc tính, sao cho với mọi thuộc tính a ∈ A , có một hàm ánh xạ  **,** trong đó được gọi là tập trị của thuộc tính *a*

Hàm ánh xạ này cho phép xác định giá trị của thuộc tính *a* cho từng đối tượng trong tập *U.*

Hệ quyết định (Decision System) được định nghĩa là một cặp , trong đó:

* U là tập không rỗng các đối tượng.
* A là tập hữu hạn các thuộc tính, và d là thuộc tính quyết định. Có thể có nhiều thuộc tính quyết định trong hệ thống này.
* Các phần tử của tập A được gọi là thuộc tính điều kiện, dùng để xác định và phân tích quyết định dựa trên các thuộc tính này.

Hệ quyết định giúp tổ chức thông tin và hỗ trợ việc ra quyết định dựa trên các thuộc tính điều kiện và thuộc tính quyết định.

* 1. **Quan hệ bất khả phân biệt.**

Cho IS = (U, A) là hệ thông tin, với tập có quan hệ tương đương tương ứng:

được gọi là quan hệ bất khả phân biệt theo B (B-indiscernibility relation)

Nếu thì các đối tượng *x* và *x’* là không thể phân biệt nhau qua tập thuộc tính B. Các lớp tương đương của quan hệ bất khả phân theo B được ký hiệu là **.**

Quan hệ tương đương (bất khả phân biệt) dẫn đến một phân hoạch tập phổ quát. Có thể dùng các phân hoạch để tạo các tập con mới của tập phổ quát. Các tập con thường được quan tâm có cùng giá trị thuộc tính điều kiện.

* 1. **Xấp xỉ tập hợp.**

Mục đích của xấp xỉ tập hợp là chỉ ra được khách hàng nào có thuộc tính quyết định có giá trị dương, chỉ ra được khách hàng nào có thuộc tính quyết định không có giá trị dương và những khách hàng nào thuộc vào vùng biên giữa các trường hợp chắc chắn.

Gọi T = (U, A) và và . Chúng ta có thể xấp xỉ X dùng các thông tin chứa trong B bằng cách tạo các xấp xỉ B-dưới và B-trên của X, ký hiệu lần lượt là *BX và X* với:

Các đối tượng trong chắc chắn được phân lớp như là các thành viên của tập X. Các đối tượng trong chỉ có thể phân lớp là các đối tượng dương tính. Vùng B-biên của X, chứa các đối tượng không thể phân lớp chắc chắn vào X theo B. Vùng B-ngoài của X, , Chứa các đối tượng chắn chắn được phân lớp không thuộc về X. Một tập được gọi là thô (rough) nếu vùng biên của nó khác rỗng, ngược lại tập là rõ (crisp).

Độ chính xác của xấp xỉ tập hợp được tính như sau:

* Với |X| là lực lượng của
* Rõ ràng
* Nếu X là rõ so với B.
* Nếu X là thô so với B.
  1. **Rút gọn.**

Chỉ giữ lại các thuộc tính bảo toàn quan hệ bất khả phân biệt và hệ quả là bảo toàn xấp xỉ tập hợp. Thường có nhiều tập con như thế và tập con nhỏ nhất được gọi là rút gọn (reducts). Các bước rút gọn tập thô như sau:

* **Bước 1: Xác định ma trận phân biệt:** Cho IS= (U, A) là 1 hệ thông tin ma trận phân biệt của S là 1 ma trận n×n (n là số đối tượng), với được tính bởi công thức:
* **Bước 2: Xác định hàm phân biệt và rút gọn hàm:** Hàm phân biệt là một hàm logic, được xác định từ ma trận phân biệt .Hàm phân biệt được định nghĩa như sau:

rong đó,

Tập các đơn thức của xác định tập các rút gọn của IS

* 1. **Phụ thuộc thuộc tính.**

Tập thuộc tính D phụ thuộc hoàn toàn vào tập thuộc tính C, ký hiệu là C => D nếu tất cả các thuộc tính của D đều được xác định duy nhất bởi giá trị của các thuộc tính trong C. Công thức tính là:

* Nếu k = 1 thì D phụ thuộc hoàn toàn vào C.
* Nếu k < 1 thì D phụ thuộc một phần (theo mức độ k) vào C

## Phân lớp.

Phân lớp (Classification) là một kỹ thuật trong khai phá dữ liệu và học máy, được sử dụng để dự đoán các nhãn phân lớp cho các bộ dữ liệu hoặc mẫu mới dựa trên các mẫu dữ liệu đã biết trước. Mục đích là đự đoán nhãn phân lớp cho các mẫu dữ liệu mới chưa được gắn nhãn, từ đó hỗ trợ quá trình ra quyết định hoặc phân tích dữ liệu. Phân lớp thường được ứng dụng trong nhiều lĩnh vực như nhận diện khuôn mặt, phân loại email (spam hoặc không spam), và chẩn đoán bệnh từ dữ liệu y tế.

Đầu vào của bài toán là một tập hợp các mẫu dữ liệu huấn luyện, trong đó mỗi mẫu đã được gắn nhãn phân lớp cụ thể. Tập dữ liệu này đóng vai trò làm cơ sở để xây dựng mô hình. Đầu ra một mô hình (bộ phân lớp) được sinh ra từ quá trình huấn luyện, có khả năng gán nhãn phân lớp chính xác cho các dữ liệu mới dựa trên thông tin học được từ tập huấn luyện. Các bước trong phân lớp dữ liệu:

* **Bước 1: Xây dựng mô hình:** Mỗi bộ/mẫu dữ liệu được phân vào một lớp được xác định trước. Lớp của một bộ/mẫu dữ liệu được xác định bởi thuộc tính gán nhãn lớp. Tập các bộ/mẫu dữ liệu huấn luyện tập huấn luyện- được dùng để xây dựng mô hình.Mô hình được biểu diễn bởi các luật phân lớp, các cây quyết định hoặc các công thức toán học
* **Bước 2: Sử dụng mô hình:** Phân lớp cho những đối tượng mới hoặc chưa được phân lớp.
* **Bươc 3: Đánh giá độ chính xác của mô hình:**
* Lớp biết trước của một mẫu/bộ dữ liệu đem kiểm tra được so sánh với kết quả thu được từ mô hình.
* Tỉ lệ chính xác = phần trăm các mẫu/bộ dữ liệu được phân lớp đúng bởi mô hình trong số các lần kiểm tra
  1. **Cây quyết định.**
     1. **Định nghĩa.**

Cây quyết định là một kiểu mô hình dự báo. Kỹ thuật học máy dùng trong cây quyết định được gọi là học bằng cây quyết định, hay chỉ gọi với cái tên ngắn gọn là cây quyết định. Phương tiện có tính mô tả dành cho việc tính toán các xác suất có điều kiện. Sự kết hợp của các kỹ thuật toán học và tính toán nhằm hỗ trợ việc mô tả, phân loại và tổng quát hóa một tập dữ liệu cho trước. Cây quyết định là một cấu trúc phân cấp của các nút và các nhánh. Có 3 loại nút trên cây:

* Nút gốc
* Nút nội bộ: mang tên thuộc tính của CSDL
* Nút lá: mang tên lớp

Cây quyết định được sử dụng trong phân lớp bằng cách duyệt từ nút gốc của cây cho đến khi đụng đến nút lá, từ đó rút ra lớp của đối tượng cần xét.

* + 1. **Xây dựng cây quyết định.**

Cây quyết định được thiết lập theo cấu trúc từ trên xuống dưới. Quá trình xây dựng bắt đầu bằng việc rời rạc hóa các thuộc tính dạng phi số để thuận tiện cho việc phân loại. Tất cả các mẫu huấn luyện ban đầu sẽ được đặt tại nút gốc của cây.

Trong quá trình xây dựng, một thuộc tính được chọn để phân chia dữ liệu thành các nhánh. Việc lựa chọn thuộc tính này dựa trên các độ đo thống kê hoặc heuristic, nhằm đảm bảo sự phân chia hiệu quả nhất. Sau đó, cây quyết định tiếp tục được xây dựng lặp lại cho từng nhánh con đến khi đạt điều kiện dừng, chẳng hạn như khi các nhánh chỉ còn một lớp duy nhất hoặc không còn thuộc tính nào để phân chia.

Quá trình này giúp tạo ra một cây quyết định hoàn chỉnh, sẵn sàng cho việc phân loại các mẫu dữ liệu mới. Quá trình xây dựng cây quyết định sẽ kết thúc khi thỏa mãn một trong các điều kiện dừng sau. Thứ nhất, nếu tất cả các mẫu dữ liệu tại một nút thuộc về cùng một lớp, nút đó sẽ trở thành nút lá đại diện cho lớp đó. Thứ hai, khi không còn thuộc tính nào có thể sử dụng để tiếp tục phân chia dữ liệu, quá trình xây dựng cây sẽ dừng lại. Cuối cùng, nếu tại một nút không còn mẫu dữ liệu nào để xử lý, việc phân chia tại nút đó cũng sẽ dừng. Những điều kiện này giúp đảm bảo cây quyết định được xây dựng một cách tối ưu, tránh mở rộng không cần thiết và giữ cho mô hình đơn giản, dễ hiểu.

* + 1. **Thuật toán ID3.**

Giải thuật ID3 (gọi tắt là ID3) Được phát triển đồng thời bởi Quinlan trong AI và Breiman, Friedman, Olsen và Stone trong thống kê.

Độ đo để lựa chọn thuộc tính: Thuộc tính được chọn là thuộc tính có lợi nhất cho quá trình phân lớp (tạo ra cây nhỏ nhất). Có 2 độ đo thường dùng: Độ lợi thông tin (Information gain), Chỉ số Gini (Gini index).

***Độ lợi thông tin (Information gain):***

* S: số lượng tập huấn luyện
* : số các mẫu của S nằm trong lớp với i = {1, …, m}
* Thông tin cần biết để phân lớp một mẫu

Cho **P** và **N** là hai lớp và **S** là một tập dữ liệu có p phần tử lớp **P** và n phần tử lớp **N.** Khối lượng thông tin cần thiết để quyết định một mẫu tùy ý có thuộc về lớp **P** hay **N** hay không là

Giả sử thuộc tính A có các giá trị {a1, a2,..., an}, việc sử dụng thuộc tính A sẽ phân chia tập huấn luyện thành n tập con {S1​, S2​,..., Sn​}. Trong đó, là số mẫu của lớp thuộc tập con với điều kiện . Entropy của thuộc tính A, ký hiệu E(A) , được tính bằng công thức:

Độ lợi thông tin dựa trên phân nhánh bằng thuộc tính A:

Tại mỗi cấp, chúng ta chọn thuộc tính có độ lợi lớn nhất để phân nhánh cây hiện tại.

***Gini index***

Giả sử:

* **D**: Tập huấn luyện.
* tập các mẫu của D thuộc lớp với i = {1, ..., m}, **m** số lớp
* **,**: lực lượng của tập và D
* xác suất để một mẫu bất kỳ của D thuộc lớp
* **Gini(D):** Gini index của tập D:
* *Gini index dựa trên phân chia theo 1 thuộc tính (A)*
* Thuộc tính A có các giá trị: {}
* Dùng A để phân chia tập train D thành tập con {}
* Gini index của phân chia D theo thuộc tính A:
  1. **Naive Bayes.**

Phân lớp bằng Bayes là một phương pháp dự đoán xác suất mà một mẫu mới thuộc về một lớp cụ thể. Nền tảng của phương pháp này dựa vào định lý Bayes, trong đó X và Y là các biến bất kỳ, và mục tiêu là dự đoán Y từ X. Để thực hiện phân lớp, ta cần lượng giá các tham số của P(X∣Y) và P(Y) trực tiếp từ tập dữ liệu huấn luyện.

Bài toán phân lớp có thể được hình thức hóa bằng xác suất a-posteriori, được biểu diễn bởi công thức:

Ý tưởng chính là gán cho mẫu nhãn phân lớp là sao cho là lớn nhất, từ đó đưa ra quyết định phân loại chính xác.

Định lý Bayes:

Cụ thể:

Phương pháp này có nhiều ưu điểm, bao gồm việc dễ dàng cài đặt, thời gian thi hành tương tự như cây quyết định, và khả năng đạt kết quả tốt trong phần lớn các trường hợp. Tuy nhiên, nó cũng tồn tại nhược điểm, đó là giả thiếc về tính độc lập điều kiện của các thuộc tính, điều này có thể làm giảm độ chính xác của mô hình.

***Thuật toán Bayes Làm trơn Laplace:***

Thuật toán Bayes áp dụng công thức Laplace để tránh trường hợp

Cụ thể, công thức được sử dụng là:

Trong đó, *m* là số phần lớp và *r* là số giá trị rời rạc của thuộc tính.

## Gom cụm.

* 1. **Tổng quan.**

Gom cụm là một quá trình quan trọng trong việc nhóm các đối tượng dữ liệu lại với nhau dựa trên những đặc điểm tương đồng. Gom cụm có nghĩa là tập hợp các đối tượng lại thành các nhóm, trong đó các đối tượng trong cùng một cụm có sự tương tự với nhau, trong khi lại không tương tự với các đối tượng thuộc các cụm khác. Mục tiêu chính của việc gom cụm là tạo ra các nhóm có sự tương đồng cao, từ đó giúp cho việc phân tích và khai thác thông tin trở nên dễ dàng và hiệu quả hơn.

Gom cụm có nhiều ứng dụng tiêu biểu trong phân tích dữ liệu. Đầu tiên, nó là một công cụ độc lập giúp xem xét cách phân bố dữ liệu, từ đó cung cấp cái nhìn sâu sắc về cấu trúc và tổ chức của dữ liệu. Thứ hai, kết quả của quá trình gom cụm có thể được sử dụng như một bước tiền xử lý cho các thuật toán phân tích dữ liệu hoặc học máy khác, góp phần nâng cao độ chính xác và hiệu quả trong các ứng dụng tiếp theo.

Ứng dụng của gom cụm rất đa dạng, bao gồm việc khám phá các nhóm khách hàng phân biệt trong cơ sở dữ liệu mua hàng, nhận dạng các vùng dữ liệu sử dụng hiệu quả, và phân tích các nhóm công ty có chính sách bảo hiểm phù hợp. Ngoài ra, gom cụm còn hỗ trợ trong việc hoạch định thành phố bằng cách nhận diện các nhóm nhà theo loại hình, giá trị và vị trí địa lý. Những ứng dụng này không chỉ nâng cao hiệu quả phân tích mà còn cung cấp cái nhìn sâu sắc về cấu trúc và tổ chức dữ liệu.

* 1. **Các yêu cầu của các cụm.**

Bao gồm khả năng thay đổi quy mô (scalability), cho phép hệ thống mở rộng hoặc thu hẹp kích thước tùy theo nhu cầu. Hệ thống cũng cần có khả năng làm việc với các loại thuộc tính khác nhau và khám phá các cụm có hình dạng bất kỳ, không nhạy cảm với tự tụ các bản ghi nhập vào. Thêm vào đó, hệ thống cần hỗ trợ dữ liệu đa chiều và hợp tác với các ràng buộc của người dùng để đáp ứng nhu cầu sử dụng. Cuối cùng, cần có khả năng diễn dịch và khái dụng thông tin một cách hiệu quả.

* 1. **Các biến thang đo theo khoảng.**

Các độ đo liên tục của các thang đo tuyến tính, thô bao gồm các ví dụ như trọng lượng, chiều cao và tuổi. Đơn vị đo có thể ảnh hưởng đến phân tích cụm, vì vậy để tránh sự phụ thuộc vào đơn vị đo, cần chuẩn hóa dữ liệu.

Chuẩn hóa các độ đo bao gồm việc tính sai biệt tuyệt đối trung bình, được xác định bằng công thức:

Với

và tính độ đo chuẩn (z-score) bằng công thức:

Khoảng cách Minkowski là một khái niệm quan trọng trong toán học và vật lý, được sử dụng để đo khoảng cách giữa các điểm trong không gian. Định nghĩa khoảng cách giữa hai điểm *I* và *j* với các tọa độ ( và ( được biểu diễn bằng công thức:

Với i = ( và j = ( là các đối tượng dữ liệu p-chiều và q là số nguyên dương.

Khoảng cách Minkowski có thể được điều chỉnh theo giá trị của *q*. Nếu ***q=1***, khoảng cách được gọi là khoảng cách Manhattan (hay còn gọi là khoảng cách city block), được tính bằng tổng giá trị tuyệt đối của sự chênh lệch giữa các tọa độ:

Ngược lại, nếu ***q=2***, khoảng cách trở thành khoảng cách Euclidean, được tính bằng căn bậc hai của tổng bình phương sự chênh lệch giữa các tọa độ:

Hai loại khoảng cách này thường được sử dụng trong các bài toán phân tích dữ liệu và học máy để đo lường sự tương đồng giữa các đối tượng trong không gian.

* 1. **Thuật toán gom cụm K-Means.**
     1. **Định nghĩa.**

Thuật toán gom cụm k-means là một phương pháp phổ biến trong phân tích dữ liệu, được sử dụng để phân hoạch dữ liệu thành k cụm. Đầu tiên, k là số cụm cần phân hoạch được xác định, và k điểm làm trọng tâm ban đầu được chọn ngẫu nhiên trong không gian dữ liệu.

* + 1. **Quy trình thực hiện thuật toán.**
* **Bước 1:** Chọn ngẫu nhiên k điểm làm trọng tâm ban đầu của k cụm.
* **Bước 2:** Gán (hoặc gán lại) từng điểm vào cụm có trọng tâm gần điểm đang xét nhất.
* **Bước 3:** Thuật toán kiểm tra xem có điểm nào được gán lại hay không; nếu không có sự thay đổi nào trong việc gán cụm, quá trình sẽ dừng lại. Ngược lại, nếu có sự thay đổi, trọng tâm của các cụm sẽ được cập nhật bằng cách tính toán lại vị trí dựa trên các điểm đã được gán cho từng cụm.
* **Bước 4:** Quá trình này sẽ tiếp tục lặp lại cho đến khi không còn sự thay đổi trong việc phân cụm, đảm bảo rằng thuật toán đã hội tụ.

Lưu ý rằng thuật toán k-means nhạy cảm với vị trí khởi tạo của các trọng tâm, vì vậy có thể cần chạy nhiều lần với các khởi tạo khác nhau để đạt được kết quả tốt nhất. Kết quả cuối cùng có thể khác nhau tùy thuộc vào cách chọn k và các trọng tâm ban đầu.

* + 1. **Tổng kết.**

Phân tích gom cụm là một kỹ thuật mạnh mẽ được sử dụng để nhóm các đối tượng dựa trên sự tương tự giữa chúng. Quá trình này có thể áp dụng trong nhiều lĩnh vực khác nhau, từ phân tích thị trường đến nhận dạng mẫu.

Đặc biệt, phân tích gom cụm có khả năng mở rộng để xử lý các dữ liệu có phạm vi ứng dụng lớn. Để thực hiện điều này, cần có cách tính độ tương tự cho nhiều loại dữ liệu khác nhau, vì mỗi loại dữ liệu có thể yêu cầu các phương pháp khác nhau để đánh giá sự tương đồng. Việc lựa chọn độ tương tự cũng phụ thuộc vào loại dữ liệu cụ thể được sử dụng và mục tiêu của phân tích.

* 1. **Thuật toán gom cụm Kohonen.**
     1. **Giới thiệu về Kohonen.**

Học có giám sát là kỹ thuật học sử dụng các bài toán phân loại (Classification). Để thực hiện được bài toán này, trước tiên cần phải có hai điều kiện. Điều kiện thứ nhất: phải biết trước số nhãn lớp cần phân loại, tức là phải biết được các lớp mà dữ liệu có thể rơi vào. Giá trị số trong công thức đầu vào có thể là xóa, cam, táo, ổi, dứa (đây chính là các nhãn lớp). Điều kiện thứ hai: phải có tập dữ liệu trung tính có chứa thông tin về các lớp. Tập dữ liệu này cần đủ lớn để mô hình có thể học và phân loại chính xác. Kỹ thuật này thường được áp dụng cho các loại dữ liệu có cấu trúc rõ ràng và có thể phân loại dễ dàng, cũng như trong nhiều bài toán thực tiễn khác nhau.

Học không giám sát là kỹ thuật học sử dụng cho các bài toán phân cụm, gom cụm (Clustering). Để thực hiện được bài toán này, cần phải có tập đặc trưng của mỗi loại dữ liệu. Tập đặc trưng này có độ phong phú và đa dạng để mô hình có thể nhận diện các nhóm khác nhau trong dữ liệu. Điểm khác giữa học không giám sát và học có giám sát là ở chỗ học không giám sát không yêu cầu thông tin nhãn cho các lớp, mà thay vào đó, nó tìm kiếm cấu trúc và mẫu trong dữ liệu một cách tự động. Phương pháp này thường được áp dụng trong phân tích dữ liệu lớn và khám phá dữ liệu để phát hiện các nhóm tiềm năng mà không cần biết trước về chúng.

Thuật toán gom cụm bằng mạng Kohonen là một kỹ thuật học không giám sát. Mạng Kohonen được phát triển bởi Teuvo Kohonen vào năm 1980. Kỹ thuật này được sử dụng trong việc gom cụm các đối tượng, cho phép phân loại và tổ chức dữ liệu một cách hiệu quả. Một trong những ưu điểm nổi bật của mạng Kohonen là không cần phải chỉ định trước số cụm, điều này giúp linh hoạt hơn trong việc tìm kiếm cấu trúc dữ liệu mà không bị ràng buộc bởi các giả định ban đầu.

* + 1. **Ứng dụng mạng Kohonen để gom cụm dữ liệu.**

Kiến trúc cụm phẳng là một cấu trúc trong đó một tập đối tượng O được chia thành các cụm phẳng. Các đối tượng trong cụm có mức độ tương đồng cao, trong khi các đối tượng thuộc các cụm khác nhau có mức độ tương đồng thấp. Kết quả là, cụm phẳng sẽ tạo ra một phân hoạch tập đối tượng. Gọi C1, C2, ..., Ck là một tập kiến trúc cụm phẳng, các cụm này có tính chất sau:

Ý tưởng chính của mạng Kohonen là tạo ra một mạng hai chiều (map) với số dòng và số cột được khởi tạo trước. Trọng số của các phần tử (neuron) trong mạng được khởi tạo ngẫu nhiên. Trong quá trình học, các neuron sẽ được điều chỉnh để trở nên gần gũi hơn với dữ liệu đầu vào, nhằm đạt được sự phân bố hiệu quả hơn. Cụ thể, neuron chính sẽ thay đổi trọng số để phản ánh dữ liệu gần nhất, trong khi các neuron lân cận cũng được điều chỉnh nhưng với mức độ ít hơn. Quá trình này lặp đi lặp lại cho đến khi mạng đạt được sự ổn định trong việc phân loại các đầu vào.

* + 1. **Thuật toán gom cụm bằng Kohonen.**
* **Bước 1:** Khởi tạo ngẫu nhiên các trọng số trên lớp ra Kohonen và gán Nc(t) là bán kính của vùng láng giềng. Khởi gán biến chu kỳ t=1
* **Bước 2:** Đưa vào một mẫu học v(t) và chuẩn hóa vector nhập v(t) Tính khoảng cách Euclide từ vector nhập v(t) đến tất cả các vector trọng của tất cả các nơron trên lớp ra Kohonen và chọn nơron có khoảng cách Euclide dE nhỏ nhất từ vector học v(t) đến trọng ứng với nút đó. Nơron ở nút này được gọi là nơron chiến thắng.

Trong đó i,j là các chỉ số hợp lệ được xác lập theo kích thước của lớp ra Kohonen.

* **Bước 3:** Cập nhật các trọng số của các nút nằm trong vùng lân cận của nút chứa noron chiến thắng ( theo công thức:

**A close-up of a white background

Description automatically generated**

Hình 1.: Công thức cập nhật trọng số cát nút trong vùng lân cận của noron chiến thắng

* **Bước 4:** Cập nhật , đưa mẫu nhập kế tiếp vào mạng Kohonen và quay về bước 2 cho đến khi đạt được điều kiện hội tụ hay vượt qua số lần lặp quy định.

# CHƯƠNG 3: CÀI ĐẶT

## Tập phổ biến và luật kết hợp.

Bài toán tập phổ biến và luật kết hợp là một vấn đề cốt lõi trong lĩnh vực khai thác dữ liệu. Mục tiêu của bài toán là tìm kiếm các mối quan hệ tiềm ẩn giữa các mục trong một tập dữ liệu lớn. Các luật kết hợp được biểu diễn dưới dạng "Nếu X thì Y", trong đó X và Y là các tập hợp các mục. Độ phổ biến của một luật được đo bằng độ hỗ trợ (support) và độ tin cậy (confidence), thể hiện tần suất xuất hiện của luật trong dữ liệu và xác suất Y xảy ra khi X đã xảy ra.

* 1. **Dữ liệu.**

Trong đồ án này, nhóm chúng tôi dùng bộ dữ liệu về lịch sử mua hàng ở một tiệm tạp hóa, bộ dữ liệu gồm 3 cột là mã khách hàng (Member\_number), ngày mua (Date) và món đồ họ mua (itemDescription). Dữ liệu có cấu trúc cơ bản như hình sau:

A table with numbers and numbers

Description automatically generated

Hình 2.: Bộ dữ liệu tập phổ biến và luật kết hợp

* 1. **Thuật toán.**

Về thuật toán, chúng tôi dùng thuật toán Apriori để tìm ra các tập phổ biến và luật kết hợp giữa các tập. Thuật toán gồm 2 hàm là *apriori* và hàm *generate\_rules* như sau:

def apriori (df, min\_support):

itemset\_support = {}

customers = df['Member\_number'].unique()

for customer in customers:

items = df[df['Member\_number'] == customer]['itemDescription'].unique()

for r in range(1, len(items) + 1):

for itemset in combinations(items, r):

itemset = tuple(sorted(itemset))

if itemset in itemset\_support:

itemset\_support[itemset] += 1

else:

itemset\_support[itemset] = 1

num\_customers = len(customers)

frequent\_itemsets = {itemset: support / num\_customers for itemset, support in itemset\_support.items() if support / num\_customers >= min\_support}

return frequent\_itemsets

Trong hàm trên, dictionary *itemset\_support* dùng để lưu số lần xuất hiện của tập hợp món đồ, sau đó lấy ra danh sách các khách hàng *customers.* Dùng vòng lặp để lấy các món đồ mà khách hàng đã mua và tạo tất cả các tập hợp món đồ có thể có. Cuối cùng là tính độ hỗ trợ của các tập hợp món đồ trên bằng cách chia số lần xuất hiện của tập hợp đó cho tổng số khách hàng. Chỉ giữ lại các tập hợp có độ hỗ trợ lớn hơn hoặc bằng *min\_support*.

def generate\_rules(frequent\_itemsets, min\_confidence=0.1):

rules = []

for itemset in frequent\_itemsets:

if len(itemset) > 1:

for i in range(1, len(itemset)):

for antecedent in combinations(itemset, i):

consequent = tuple(sorted(set(itemset) - set(antecedent)))

antecedent\_support = frequent\_itemsets[antecedent] if antecedent in frequent\_itemsets else 0

if antecedent\_support > 0:

confidence = frequent\_itemsets[itemset] / antecedent\_support

if confidence >= min\_confidence:

rules.append((antecedent, consequent, confidence))

return rules

Trong hàm trên, danh sách *rules* dùng để chứa danh sách các luật kết hợp. Sau đó dùng vòng lặp để duyệt qua từng tập phổ biến, với mỗi món đồ, tạo ra các luật kết hợp bằng cách chia tập hợp thành hai phần *antecedent* và *consequent*. Độ tin cậy của luật được tính bằng cách chia độ hỗ trợ của tập phổ biến cho độ hỗ trợ của *antecedent*. Chỉ giữ lại các luật có độ tin cậy lớn hơn hoặc bằng *min\_confidence*.

* 1. **Giao diện và kết quả.**

Về giao diện, chúng tôi thực hiện một giao diện web bằng Django như sau:

A screenshot of a computer

Description automatically generated

Hình 2.: Giao diện và kết quả web bằng Django

A screenshot of a computer

Description automatically generated Về giao diện của tập phổ biến và luật kết hợp, có 2 ô để nhập các giá trị minsup, minconf và một nút để chạy kết quả, hiển thị kết quả từ thuật toán ra như sau:

Hình 2.3: Giao diện của tập phổ biến và luật kết hợp

Hình 2.3 Giao diện của tập phổ biến và luật kết hợp

Hình 2.: Giao diện của tập phổ biến và luật kết hợp

Sau khi nhập minsup và minconf (ở đây chúng tôi nhập minsup=0.15 và minconf=0.4) và bấm chạy kết quả thì kết quả hiển thị như sau:

A screenshot of a computer

Description automatically generated

Hình 2.: Kết quả sau khi nhập minsup và minconf

Khi nhập số liệu vào 2 ô trên, dữ liệu sẽ truyền về các hàm ở dưới local và chạy các thuật toán bên trên, sau đó trả về kết quả tương ứng trên giao diện web. Ở đây ta có các sản phẩm trong tập phổ biến với minsup từ 0.15 trở lên, có nghĩa là với bộ dữ liệu hơn 26000 dòng thì các sản phẩm trên đã xuất hiện từ 3900 lần trở lên. Và với minconf là 0.45 thì những người mua ‘other vegetables’ sẽ có xu hướng mua ‘whole milk’.

## Tập thô.

Tập thô là một tập con của các thuộc tính trong một tập dữ liệu, sao cho tập con này giữ nguyên khả năng phân biệt các đối tượng trong tập dữ liệu như tập thuộc tính ban đầu. Nói cách khác, một tập thô là một tập thuộc tính tối thiểu mà vẫn đảm bảo khả năng phân loại của hệ thống. Các khái niệm liên quan đến tập thô chúng tôi dùng trong đồ án này gồm quan hệ bất khả phân biệt, xấp xỉ tập hợp và rút gọn.

* 1. **Dữ liệu.**

Trong đồ án này, nhóm chúng tôi dùng bộ dữ liệu tương tự trong bài giảng môn học với 3 cột là số tuổi (Age), số buổi (Sesions) và thi đậu (Pass) như sau để tìm quan hệ bất khả phân biệt và xấp xỉ tập hợp:

A screenshot of a number

Description automatically generated

Hình .5: Bộ dữ liệu để tìm quan hệ bất khả phân biệt và xấp xỉ tập hợp

Về phần rút gọn, chúng tôi dùng bộ dữ liệu có trong bài giảng môn học gồm 6 cột là Tên người, Màu tóc, Chiều cao, Cân nặng, Dùng thuốc và Kết quả như sau:

A white background with black text

Description automatically generated

Hình .6: Bộ dữ liệu để rút gọn

* 1. **Thuật toán.**

Về thuật toán, chúng tôi dùng các hàm để tìm quan hệ bất khả phân biệt, tính xấp xỉ và rút gọn như sau:

**# Hàm tính quan hệ bất khả phân biệt:**

def ind (data, attributes):

equivalence\_classes = {}

for idx, row in data.iterrows():

key = tuple(row[attr] for attr in attributes) # Lấy giá trị của các thuộc tính

if key not in equivalence\_classes:

equivalence\_classes[key] = []

equivalence\_classes[key].append(idx)

return equivalence\_classes

eq\_classes = ind(data, ['Age', 'Sessions'])

Trong hàm trên, ta tạo dictionary *equivalance\_classes* để lưu các lớp tương đương chứa các index của hàng có giá trị giống nhau. Vòng lặp duyệt qua từng hàng của bộ dữ liệu, tạo một khóa *key* chứa giá trị của thuộc tính, sau đó kiểm tra xem nếu key đã tồn tại trong lớp tương đương hay chưa, nếu chưa thì tạo một danh sách rỗng cho *key* đó, ngược lại thì thêm chỉ số của hàng hiện tại vào danh sách tương ứng với key trong lớp tương đương.

**# Hàm tính xấp xỉ dưới:**

def lower\_approximation(eq\_classes, target\_set):

lower = []

for eq\_class in eq\_classes.values():

if set(eq\_class).issubset(target\_set): # Toàn bộ lớp tương đương nằm trong tập mục tiêu

lower.extend(eq\_class)

return set(lower)

**# Hàm tính xấp xỉ trên:**

def upper\_approximation(eq\_classes, target\_set):

upper = []

for eq\_class in eq\_classes.values():

if set(eq\_class).intersection(target\_set): # Có giao với tập mục tiêu

upper.extend(eq\_class)

return set(upper)

Hai hàm trên dùng để tính xấp xỉ trên và xấp xỉ dưới dựa trên lớp tương đương. Đầu tiên tạo một danh sách để lưu trữ kết quả, sau đó dùng vòng lặp duyệt qua từng phần trong lớp tương đương. Nếu toàn bộ lớp tương đương nằm trong tập mục tiêu *target\_set* (các dòng chứa giá trị Pass là Yes) thì thêm tất cả đối tượng trong lớp tương đương vào danh sách chứa kết quả của xấp xỉ dưới. Nếu lớp tương đương có giao với *target\_set* thì thêm tất cả các đối tượng trong lớp tương đương vào danh sách chứa kết quả của xấp xỉ trên.

Sau khi có xấp xỉ trên và xấp xỉ dưới, có thể tính được độ chính xác và khảo sát sự phụ thuộc của biến phụ thuộc vào biến độc lập như sau:

**# Tính xấp xỉ trên và xấp xỉ dưới**

lower = lower\_approximation(eq\_classes, target\_set)

upper = upper\_approximation(eq\_classes, target\_set)

**# Tính độ chính xác bằng cách chia lực lượng của xấp xỉ trên cho lực lượng của xấp xỉ dưới**

accuracy = len(lower) / len(upper)

**# Tính xấp xỉ dưới cho 'Yes' và 'No'**

lower\_yes = lower\_approximation(eq\_classes, set(data[data['Pass'] == 'Yes'].index))

lower\_no = lower\_approximation(eq\_classes, set(data[data['Pass'] == 'No'].index))

**# Khảo sát sự phụ thuộc của Pass vào Age và Sessions bằng cách lấy tổng lực lượng của xấp xỉ dưới của Yes và No chia cho lực lượng của bộ dữ liệu**

dependency = (len(lower\_yes) + len(lower\_no)) / len(data)

**# Hàm tính reduct:**

def calculate\_reduct(request):

df = pd.read\_csv('reduct.csv')

df\_encoded = pd.get\_dummies(df.drop('Kết quả', axis=1), drop\_first=True)

y = df['Kết quả'].apply(lambda x: 1 if x == 'Bị rám' else 0)

model = DecisionTreeClassifier()

rfe = RFE(model, n\_features\_to\_select=2)

fit = rfe.fit(df\_encoded, y)

selected\_features = df\_encoded.columns[fit.support\_]

num\_features\_list = [2, 3]

results = []

for num\_features in num\_features\_list:

rfe = RFE(model, n\_features\_to\_select=num\_features)

fit = rfe.fit(df\_encoded, y)

results.append({'num\_features': num\_features,

'selected\_features': df\_encoded.columns[fit.support\_].tolist(),

'mean\_score': scores.mean()})

res\_list = []

for result in results:

res\_list.append(result['selected\_features'])

Hàm trên sử dụng *pd.get\_dummies* để chuyển đổi các biến phân loại thành các biến số (one-hot encoding), bỏ cột 'Kết quả' và loại bỏ cột đầu tiên của mỗi biến phân loại để tránh đa cộng tuyến. Sau đó chuyển đổi cột 'Kết quả' thành biến nhị phân, với giá trị 1 nếu 'Kết quả' là 'Bị rám' và 0 nếu không bị rám. Chúng tôi dùng mô hình *Decision Tree* của thư viện *sklearn* để sử dụng trong loại bỏ đặc trưng bằng phương pháp đệ quy *RFE,* sau đó lưu kết quả vào một danh sách.

* 1. **Giao diện và kết quả.**

Về giao diện của tập thô, chúng tôi thiết kế thành 2 phần tính xấp xỉ và tính rút gọn tập thể như sau:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Hình .7: Giao diện của tập thô

Khi nhấn vào xem dữ liệu sẽ hiển thị ra dữ liệu tương ứng bên trên, và khi chọn xem kết quả sẽ hiển thị các kết quả do thuật toán chạy bên dưới local như sau:

A screenshot of a computer

Description automatically generated

A close-up of a text

Description automatically generated

Hình .8: Kết quả thực hiện tập thô

Sau khi bấm chạy kết quả của tính xấp xỉ, các hàm sẽ tính toán IND bằng cách tìm các dòng có giá trị Age và Sessions giống nhau và lưu thành một tập IND. Như hình trên thì kết quả Age từ 16-30 và Sessions là 50 là một phần tử của IND ở index 0, và ở index 2 và 3 thì có Age là 31-45 và Sessions từ 1-25 giống nhau nên cũng là một phần tử của IND. Sau đó sẽ tính xấp xỉ trên bằng cách xem xét nếu các index có kết quả Pass là Yes giao với IND khác rỗng thì sẽ thêm vào danh sách kết quả của xấp xỉ trên, và xấp xỉ dưới sẽ xem xét nếu index đó là tập con của IND. Sau đó tính độ chính xác bằng cách lấy lực lượng của xấp xỉ dưới chia cho lực lượng của xấp xỉ trên. Cuối cùng tính khảo sát sự phụ thuộc bằng cách lấy tổng của lực lược xấp xỉ dưới các thuộc tính Pass là No và lực lượng xấp xỉ dưới các thuộc tính Pass là Yes sau đó chia cho số lượng phần tử của dữ liệu.

## Phân lớp.

Phân lớp là một trong những bài toán quan trọng trong lĩnh vực khai phá dữ liệu. Bài toán này tập trung vào việc xây dựng các mô hình hoặc thuật toán để phân loại các đối tượng vào các nhóm hoặc lớp khác nhau dựa trên các đặc trưng của chúng. Trong đồ án này, nhóm chúng tôi thực hiện hai thuật toán phân lớp phổ biến là Cây quyết định (Decision Tree) và thuật toán Naive Bayes.

* 1. **Cây quyết định.**

Cây quyết định (Decision Tree) là một phương pháp phân lớp phổ biến trong khai phá dữ liệu. Đây là một mô hình dựa trên cấu trúc cây, trong đó mỗi nút đại diện cho một thuộc tính của dữ liệu, mỗi nhánh đại diện cho một giá trị của thuộc tính đó, và mỗi lá đại diện cho một nhãn hoặc kết quả phân loại.

* + 1. **Dữ liệu.**

Trong thuật toán này, chúng tôi dùng bộ dữ liệu về loại thuốc các bệnh nhân sẽ dùng (Drug), được phân loại bởi tuổi (Age), giới tính (Sex), huyết áp (BP), nồng độ cholesterol (Cholesterol), chỉ số natri và kali (Na\_to\_K). Dữ liệu có mẫu như sau:

A table with numbers and letters

Description automatically generated

Hình 2.9: Bộ dữ liệu của Cây quyết định và Naive Bayes

* + 1. **Thuật toán.**

Về thuật toán, chúng tôi dùng thuật toán ID3 và dùng chỉ mục Gini để xây dựng cây quyết định, phân tích và dự đoán loại thuốc bệnh nhân sẽ dùng.

**# Hàm tính chỉ mục Gini**

def gini\_index(freq):

prob = freq / float(freq.sum())

return 1.0 - np.sum(prob \*\* 2)

def calculate\_gini(target, ids):

if len(ids) == 0:

return 0

freq = np.array(target.iloc[ids].value\_counts())

return gini\_index(freq)

**# Hàm đặt nhãn cho 1 node:**

def set\_label(target, ids):

return target.iloc[ids].mode()[0]

**# Hàm chia node:**

def split\_node(data, target, ids, min\_samples\_split, min\_gain):

best\_gain = 0

best\_splits = []

best\_attribute = None

order = None

sub\_data = data.iloc[ids, :]

for i, att in enumerate(data.columns):

values = data.iloc[ids, i].unique().tolist()

if len(values) == 1:

continue

splits = []

for val in values:

sub\_ids = sub\_data.index[sub\_data[att] == val].tolist()

splits.append(sub\_ids)

if min(map(len, splits)) < min\_samples\_split:

continue

HxS = 0

for split in splits:

HxS += len(split) \* calculate\_gini(target, split) / len(ids)

gain = calculate\_gini(target, ids) - HxS

if gain < min\_gain:

continue

if gain > best\_gain:

best\_gain = gain

best\_splits = splits

best\_attribute = att

order = values

return best\_attribute, order, best\_splits

**# Hàm dựng cây quyết định:**

def build\_tree(data, target, ids, depth, max\_depth, min\_samples\_split, min\_gain):

gini = calculate\_gini(target, ids)

if depth >= max\_depth or gini <= min\_gain:

return {'label': set\_label(target, ids)}

split\_attribute, order, splits = split\_node(data, target, ids, min\_samples\_split, min\_gain)

if not splits:

return {'label': set\_label(target, ids)}

children = []

for split in splits:

child = build\_tree(data, target, split, depth + 1, max\_depth, min\_samples\_split, min\_gain)

children.append(child)

return {'split\_attribute': split\_attribute, 'order': order, 'children': children}

**# Hàm dự đoán nhãn cho dữ liệu:**

def predict(tree, new\_data):

labels = []

for \_, x in new\_data.iterrows():

node = tree

while 'children' in node:

idx = node['order'].index(x[node['split\_attribute']])

node = node['children'][idx]

labels.append(node['label'])

return labels

**# Hàm vẽ cây quyết định:**

def plot\_tree(node, depth=0, pos=(0, 0), dx=1.5, dy=1.5, ax=None):

if ax is None:

ax = plt.gca()

x, y = pos

node\_text = f"{node['split\_attribute']}" if 'label' not in node else f"{node['label']}"

if 'label' in node:

color = "lightgrey"

text\_color = "red" if node['label'] == "No" else "black"

else:

color = "limegreen"

text\_color = "white"

bbox\_props = dict(boxstyle="round,pad=0.3", edgecolor="black", facecolor=color)

ax.text(x, y, node\_text, ha='center', va='center', color=text\_color, bbox=bbox\_props, fontsize=12, fontweight='bold')

if 'children' in node:

num\_children = len(node['children'])

for i, child in enumerate(node['children']):

child\_x = x + (i - (num\_children - 1) / 2) \* dx / (2 \*\* depth)

child\_y = y - dy

ax.plot([x, child\_x], [y - 0.1, child\_y + 0.1], 'k-', lw=1.5)

edge\_label = node['order'][i]

ax.text((x + child\_x) / 2, (y + child\_y) / 2, edge\_label,

ha='center', va='center', fontsize=10, color="black",

bbox=dict(boxstyle="round,pad=0.2",edgecolor="none", facecolor="lightgrey"))

plot\_tree(child, depth + 1, (child\_x, child\_y), dx, dy, ax)

Đoạn code trên xây dựng cây quyết định sử dụng chỉ mục Gini. Đầu tiên, chúng tôi định nghĩa các hàm để tính toán chỉ mục Gini dựa trên tần suất xuất hiện của các nhãn trong bộ dữ liệu, với giá trị từ 0 đến 1. Tiếp theo, tìm thuộc tính tốt nhất để chia node hiện tại thành các node con, thực hiện bằng cách duyệt qua tất cả các thuộc tính và giá trị, tính toán chỉ mục Gini cho từng cách chia và chọn cách chia có chỉ mục Gini giảm nhiều nhất. Sau đó, xây dựng cây quyết định bằng cách đệ quy, chia node hiện tại thành các node con dựa trên thuộc tính tốt nhất cho đến khi đạt độ sâu tối đa hoặc chỉ mục Gini nhỏ hơn ngưỡng cho phép. Khi không thể chia tiếp, node sẽ được gán nhãn bằng cách chọn nhãn xuất hiện nhiều nhất trong node đó. Sau khi cây quyết định được xây dựng bằng cách duyệt qua cây từ gốc đến lá, dựa trên các thuộc tính của dữ liệu mới. Cuối cùng, vẽ cây quyết định bằng cách hiển thị các thuộc tính và nhãn của các node.

Về ID3, chúng tôi chỉ thay đổi ở phần tính chỉ mục Gini thành tính chỉ số Entropy và độ lợi thông tin Information Gain như sau:

def entropy(freq):

prob = freq / float(freq.sum())

return -np.sum(prob \* np.log2(prob + 1e-9))

def calculate\_entropy(target, ids):

if len(ids) == 0:

return 0

freq = np.array(target.iloc[ids].value\_counts())

return entropy(freq)

* + 1. **Giao diện và kết quả.**

Về giao diện, chúng tôi cũng thực hiện tương tự như bên trên với 2 nút xem dữ liệu và xem kết quả của thuật toán chạy dưới local như sau:

A screenshot of a computer

Description automatically generated

Hình 2.10: Giao diện của Cây quyết định

Sau khi nhấn vào xem kết quả, giao diện hiển thị ra một cây quyết định như sau:

A diagram of a tree

Description automatically generated

Hình 2.11: Kết quả thực hiện Cây quyết định

Cây quyết định trong hình trên dùng thuật toán ID3 ra quyết định để lựa chọn loại thuốc (drug) phù hợp cho một bệnh nhân dựa trên các thuộc tính huyết áp (BP), giới tính (Sex) và cholesterol. Ở đây, thuộc tính BP được chọn làm nút gốc. Mỗi nhánh từ một nút đại diện cho một giá trị có thể có của thuộc tính tại nút đó. Từ nút BP, có 3 nhánh tương ứng với 3 giá trị: "Cao", "Thấp" và "Bình thường". Các nút lá là các nút cuối cùng của cây, không có nhánh con, các nút lá đại diện cho các loại thuốc khác nhau.

Để sử dụng cây quyết định để dự đoán loại thuốc cho một bệnh nhân mới, ta sẽ bắt đầu từ nút gốc và đi theo các nhánh tương ứng với giá trị của các thuộc tính của bệnh nhân đó. Khi đến một nút lá, ta sẽ nhận được kết quả dự đoán là loại thuốc tương ứng với nút lá đó.

Ví dụ nếu một bệnh nhân có huyết áp cao, giới tính nữ và cholesterol bình thường, ta sẽ đi theo các nhánh sau:

* Từ nút gốc BP ta chọn nhánh "Cao".
* Tiếp theo, từ nút Sex ta chọn nhánh "Nữ".
* Cuối cùng, từ nút Cholesterol, ta chọn nhánh "Bình thường".
* Kết quả là ta sẽ đến một nút lá có nhãn "drugY", nghĩa là bệnh nhân này nên dùng thuốc "drugY".
  1. **Naive Bayes.**

**Thuật toán Naive Bayes** là một phương pháp phân loại xác suất dựa trên định lý Bayes. Thuật toán này được gọi là "Naive" (ngây thơ) vì nó giả định rằng các thuộc tính của một đối tượng là độc lập với nhau, mặc dù trong thực tế, giả định này thường không hoàn toàn đúng. Naive Bayes tính toán xác suất hậu nghiệm của một lớp cho trước, dựa trên các giá trị của các thuộc tính. Thuật toán này sẽ chọn lớp có xác suất hậu nghiệm lớn nhất để phân loại đối tượng. Về bộ dữ liệu, chúng tôi dùng bộ dữ liệu tương tự như thuật toán ID3 bên trên.

* + 1. **Thuật toán.**

Thuật toán được triển khai như sau:

**# Hàm tách các đặc trưng của bộ dữ liệu:**

def split\_features\_labels(data):

features = [row[:-1] for row in data]

labels = [row[-1] for row in data]

return features, labels

**# Hàm tính xác suất cho mỗi nhãn:**

def calculate\_prior\_probs(labels):

total\_samples = len(labels)

class\_counts = defaultdict(int)

for label in labels:

class\_counts[label] += 1

prior\_probs = {label: count / total\_samples for label, count in class\_counts.items()}

return prior\_probs, set(labels)

**#Hàm tính xác suất có điều kiện cho mỗi đặc trưng:**

def calculate\_conditional\_probs(features, labels, classes):

conditional\_probs = defaultdict(lambda: defaultdict(lambda: defaultdict(float)))

for feature\_index in range(len(features[0])):

feature\_values = [row[feature\_index] for row in features]

for label in classes:

label\_feature\_values = [feature\_values[i] for i in range(len(feature\_values)) if labels[i] == label]

value\_counts = defaultdict(int)

for value in label\_feature\_values:

value\_counts[value] += 1

total\_label\_features = len(label\_feature\_values)

for value, count in value\_counts.items():

conditional\_probs[feature\_index][label][value] = count / total\_label\_features

return conditional\_probs

**# Hàm huấn luyện mô hình Bayes:**

def fit\_naive\_bayes(features, labels):

prior\_probs, classes = calculate\_prior\_probs(labels)

conditional\_probs = calculate\_conditional\_probs(features, labels, classes)

return prior\_probs, conditional\_probs, classes

**# Hàm dữ đoán nhãn cho các mẫu dữ liệu đã huấn luyện:**

def predict\_naive\_bayes(features, prior\_probs, conditional\_probs, classes):

predictions = []

for x in features:

class\_probs = {}

for label in classes:

class\_probs[label] = math.log(prior\_probs[label])

for feature\_index, feature\_value in enumerate(x):

if feature\_value in conditional\_probs[feature\_index][label]:

class\_probs[label] += math.log(conditional\_probs[feature\_index][label][feature\_value])

else:

class\_probs[label] += math.log(1e-6)

best\_label = max(class\_probs, key=class\_probs.get)

predictions.append(best\_label)

return predictions

Đoạn code thực hiện việc huấn luyện và dự đoán bằng mô hình Naive Bayes. Hàm *split\_features\_labels* tách dữ liệu thành các đặc trưng và nhãn bằng cách tạo danh sách *features* chứa tất cả các cột trừ cột cuối cùng của mỗi hàng và danh sách *labels* chứa cột cuối cùng của mỗi hàng. Hàm *calculate\_prior\_probs* tính toán xác suất tiên nghiệm cho mỗi nhãn bằng cách đếm số lượng mẫu cho mỗi nhãn và chia cho tổng số mẫu, sử dụng *defaultdict* để lưu trữ số lượng mẫu cho mỗi nhãn và trả về dictionary *prior\_probs* chứa xác suất tiên nghiệm và tập hợp classes chứa các nhãn duy nhất. Hàm *calculate\_conditional\_probs* tính toán xác suất có điều kiện cho mỗi đặc trưng dựa trên nhãn bằng cách đếm số lượng giá trị của đặc trưng đó cho mỗi nhãn và chia cho tổng số giá trị của đặc trưng đó cho mỗi nhãn, sử dụng *defaultdict* để lưu trữ xác suất có điều kiện. Hàm *fit\_naive\_bayes* kết hợp hai hàm trên để huấn luyện mô hình Naive Bayes, trả về xác suất tiên nghiệm, xác suất có điều kiện và các nhãn duy nhất. Để dự đoán nhãn cho các mẫu mới, hàm *predict\_naive\_bayes* tính toán xác suất hậu nghiệm cho mỗi nhãn bằng cách sử dụng xác suất tiên nghiệm và xác suất có điều kiện, sau đó chọn nhãn có xác suất hậu nghiệm cao nhất làm nhãn dự đoán.

* + 1. **Giao diện và kết quả.**

Về giao diện, chúng tôi thiết kế giao diện có nút xem dữ liệu và các ô để lựa chọn dữ liệu để phân lớp người đó xem người đó sử dụng loại thuốc nào như sau:

A screenshot of a computer

Description automatically generated

Hình 2.12: Giao diện của Naive Bayes

Sau khi nhập các dữ liệu vào và nhấn chạy kết quả, giao diện sẽ hiển thị ra loại thuốc người đó sẽ sử dụng cùng với độ chính xác của mô hình qua 2 kết quả có và không có làm trơn Laplace như sau:

A close up of a sign

Description automatically generatedA close up of a word

Description automatically generated

Hình 2.13: Kết quả thực hiện Navie Bayes

## Gom cụm.

Gom cụm là một kỹ thuật không giám sát trong khai phá dữ liệu, nhằm phân chia một tập hợp dữ liệu thành các nhóm (cụm) sao cho các đối tượng trong cùng một cụm có sự tương đồng cao về các thuộc tính, trong khi các đối tượng thuộc các cụm khác nhau có sự khác biệt rõ rệt. Mục tiêu của gom cụm là tìm ra các cấu trúc ẩn trong dữ liệu mà không cần có thông tin về nhãn lớp trước. Trong đồ án này, chúng tôi thực hiện hai thuật toán gom cụm phổ biến là K-Means và Kohonen.

* 1. **K-Means.**

Thuật toán K-means là một thuật toán phân cụm dựa trên nguyên tắc phân chia một tập dữ liệu thành K cụm không chồng lấp nhau. Mỗi cụm được đại diện bởi một tâm cụm (centroid). Mục tiêu của K-means là tìm ra vị trí của các tâm cụm sao cho tổng bình phương khoảng cách từ mỗi điểm dữ liệu đến tâm cụm gần nhất là nhỏ nhất.

* + 1. **Dữ liệu.**

Về dữ liệu, chúng tôi dùng bộ dữ liệu về mức độ chi tiêu của khách hàng ở trung tâm mua sắm. Bộ dữ liệu chứa các thông tin được trung tâm tổng hợp lại gồm mã khách hàng (CustomerID), giới tính (Genre), tuổi (Age), thu nhập hàng năm (Annual\_Income\_(k$)) và mức độ chi tiêu (Spending\_Score). Bộ dữ liệu có dạng như sau:

A table with numbers and text

Description automatically generated

Hình 2.14: Bộ dữ liệu của K-Means và Kohonen

* + 1. **Thuật toán.**

Về thuật toán, chúng tôi thực hiện các hàm và dùng hai đặc trưng là thu nhập hàng năm cùng với mức độ chi tiêu để gom cụm như sau:

**# Hàm khởi tạo các tâm cụm:**

def initialize\_centroids(X, k):

np.random.seed(42)

centroids = X[np.random.choice(X.shape[0], k, replace=False)]

return centroids

**# Hàm gán nhãn cho các điểm dữ liệu:**

def assign\_clusters(X, centroids):

distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)

return np.argmin(distances, axis=1)

**# Hàm cập nhật các tâm cụm:**

def update\_centroids(X, labels, k):

new\_centroids = np.array([X[labels == i].mean(axis=0) for i in range(k)])

return new\_centroids

**# Hàm thực hiện K-Means:**

def kmeans(X, k, max\_iters=100):

centroids = initialize\_centroids(X, k)

for \_ in range(max\_iters):

labels = assign\_clusters(X, centroids)

new\_centroids = update\_centroids(X, labels, k)

if np.all(centroids == new\_centroids):

break

centroids = new\_centroids

return centroids, labels

Đoạn code trên thực hiện phân cụm bằng K-Means, trong đó, hàm *initialize \_centroids*khởi tạo các tâm cụm bằng cách chọn ngẫu nhiên k điểm từ dữ liệu đầu vào X. Hàm này sử dụng *np.random.seed(42)* để đảm bảo tính ngẫu nhiên có thể tái lập và *np.random.choice* để chọn ngẫu nhiên k điểm mà không lặp lại. Tiếp theo, hàm *assign\_clusters* gán nhãn cho các điểm dữ liệu bằng cách tính khoảng cách Euclid từ mỗi điểm dữ liệu đến các tâm cụm và chọn tâm cụm gần nhất. Hàm này sử dụng *np.linalg.norm* để tính khoảng cách và *np.argmin* để tìm chỉ số của tâm cụm gần nhất. Sau đó, hàm *update\_centroids* cập nhật các tâm cụm bằng cách tính trung bình của các điểm dữ liệu thuộc cùng một cụm. Hàm này sử dụng *np.array* để tạo mảng mới chứa các tâm cụm được cập nhật. Cuối cùng, hàm *kmeans* thực hiện thuật toán K-means bằng cách lặp lại quá trình gán nhãn và cập nhật tâm cụm cho đến khi các tâm cụm không thay đổi hoặc đạt đến số lần lặp tối đa *max\_iters*. Trong mỗi vòng lặp, hàm *assign\_clusters* được gọi để gán nhãn cho các điểm dữ liệu và hàm *update\_centroids* được gọi để cập nhật các tâm cụm. Nếu các tâm cụm không thay đổi sau một vòng lặp, vòng lặp sẽ dừng lại.

* + 1. **Giao diện và kết quả.**

Về giao diện, chúng tôi thiết kế một nút xem dữ liệu và một ô để nhập số lượng cụm cần để gom cụm như sau:

A screenshot of a computer

Description automatically generated

Hình 2.15: Giao diện của K-Means

Sau khi nhập số lượng cụm và chọn chạy kết quả, giao diện sẽ hiển thị một hình vẽ phân các cụm gồm các điểm dữ liệu và tâm cụm tương ứng như sau:

A screenshot of a computer screen

Description automatically generated

Hình 2.16: Kết quả thực hiện K-Means

Đầu tiên, khởi tạo các tâm cụm một cách ngẫu nhiên từ tập dữ liệu. Sau đó tính toán khoảng cách giữa từng điểm dữ liệu và các tâm cụm, gán mỗi điểm vào cụm gần nhất, cập nhật vị trí của các tâm cụm dựa trên trung bình của các điểm trong mỗi cụm. Quy trình này lặp lại cho đến khi các tâm cụm không thay đổi hoặc đạt số lần lặp tối đa.

Kết quả của quá trình phân cụm được hiển thị trong hình ảnh, trong đó các điểm khách hàng được phân loại bằng các màu khác nhau tương ứng với các cụm. Các tâm cụm được đánh dấu bằng các dấu X.

* 1. **Kohonen.**

Mạng Kohonen hay còn gọi là bản đồ tự tổ chức (SOM), là một loại mạng thần kinh nhân tạo không giám sát, được đề xuất bởi Teuvo Kohonen. SOM có khả năng tự tổ chức các dữ liệu đầu vào vào một không gian có chiều số thấp hơn (thường là 2 chiều) để tạo ra một bản đồ đặc trưng. Trong bản đồ này, các neuron (nút thần kinh) đại diện cho các cụm dữ liệu, và các neuron gần nhau đại diện cho các cụm có mối quan hệ gần gũi về mặt đặc trưng. Về dữ liệu, chúng tôi dùng bộ dữ liệu tương tự như K-Means.

* + 1. **Thuật toán.**

Về thuật toán, chúng tôi thiết kế các hàm như sau:

def initialize\_som(m, n, input\_dim):

        np.random.seed(42)

        return np.random.rand(m, n, input\_dim)

    def euclidean\_distance(x, y):

        return np.linalg.norm(x - y)

    def find\_bmu(som, x):

        distances = np.linalg.norm(som - x, axis=2)

        return np.unravel\_index(np.argmin(distances), distances.shape)

    def update\_weights(som, x, bmu\_idx, learning\_rate, radius):

        for i in range(som.shape[0]):

            for j in range(som.shape[1]):

                if euclidean\_distance(np.array([i, j]), bmu\_idx) <= radius:

                    som[i, j] += learning\_rate \* (x - som[i, j])

    def train\_som(X, som, num\_iters, learning\_rate, radius):

        for iter in range(num\_iters):

            for x in X:

                bmu\_idx = find\_bmu(som, x)

                update\_weights(som, x, bmu\_idx, learning\_rate, radius)

            learning\_rate \*= 0.9

            radius \*= 0.9

        return som

    som = initialize\_som(m, n, features.shape[1])

    som = train\_som(features, som, num\_iters, learning\_rate, radius)

    labels = np.array([find\_bmu(som, x) for x in features])

    data['Cluster'] = [str(label) for label in labels]

    unique\_labels = np.unique(labels, axis=0)

    colors = plt.cm.get\_cmap('viridis', len(unique\_labels))

    for i, label in enumerate(unique\_labels):

        mask = np.all(labels == label, axis=1)

        plt.scatter(features[mask, 0], features[mask, 1], color=colors(i), label=f'Cluster {i}')

Đoạn code trên thực hiện việc huấn luyện một mạng tự tổ chức SOM để phân nhóm các điểm dữ liệu. Đầu tiên, khởi tạo mạng SOM với kích thước m\*n và số chiều của dữ liệu đầu vào thông qua hàm *initialize\_som*, trong đó các trọng số của các nút trong mạng được khởi tạo ngẫu nhiên. Tiếp theo, hàm *euclidean\_distance* được định nghĩa để tính khoảng cách Euclid giữa hai điểm, xác định nút nào trong mạng là "best matching unit" (BMU) cho mỗi điểm dữ liệu. Hàm *find\_bmu* tìm kiếm nút BMU cho một điểm dữ liệu bằng cách tính toán khoảng cách từ điểm đó đến tất cả các nút trong mạng và xác định nút có khoảng cách nhỏ nhất. Sau khi tìm được BMU, hàm *update\_weights* thực hiện việc cập nhật trọng số của các nút trong mạng. Nó kiểm tra tất cả các nút trong SOM và nếu khoảng cách giữa nút đó và BMU nằm trong bán kính đã cho, nó sẽ điều chỉnh trọng số của nút đó để nó gần hơn với điểm dữ liệu. Quá trình này được lặp lại cho tất cả các điểm trong tập dữ liệu qua nhiều vòng lặp, được kiểm soát bởi biến *num\_iters*. Trong mỗi vòng lặp, tốc độ học và bán kính cũng được giảm dần để giúp mạng học một cách hiệu quả hơn theo thời gian. Sau khi huấn luyện hoàn tất, tạo ra nhãn cho từng điểm dữ liệu bằng cách tìm BMU tương ứng cho mỗi điểm và lưu vào biến *labels*. Cuối cùng, các điểm dữ liệu được phân nhóm và trực quan hóa bằng cách sử dụng *matplotlib*.

* + 1. **Giao diện và kết quả.**

Về giao diện, chúng tôi thiết kế để nhập số lượng cột, hàng, tốc độ học, số lần lặp cũng như bán kính như sau:

A screenshot of a computer

Description automatically generated

Hình 2.17: Giao diện của Kohonen

Sau khi nhập các thông tin vào và bấm chạy kết quả, giao diện sẽ hiển thị như sau:

A screenshot of a computer screen

Description automatically generated

Hình 2.18: Kết quả thực hiện Kohonen

Đầu tiên, hàm *initialize\_som* được gọi để khởi tạo mạng SOM với kích thước 5x5, trong đó mỗi nút được gán trọng số ngẫu nhiên từ phân phối đồng đều, với số chiều của dữ liệu đầu vào được xác định từ *features.shape[1]*. Tiếp theo, hàm *train\_som* tiến hành huấn luyện mạng bằng cách lặp qua 100 lần và cập nhật trọng số của mạng dựa trên các điểm dữ liệu trong *features*, sử dụng tốc độ học 0.1 và bán kính 1.0 để điều chỉnh mức độ thay đổi của các trọng số. Sau khi quá trình huấn luyện hoàn tất, mã sẽ tìm kiếm nút BMU cho mỗi điểm dữ liệu và gán nhãn cho chúng. Kết quả là một mảng nhãn, trong đó mỗi nhãn đại diện cho cụm mà điểm dữ liệu đó thuộc về.

# TÀI LIỆU THAM KHẢO

[1] Bộ dữ liệu thực hiện phân loại: Kaggle - <https://www.kaggle.com/datasets/jeevanrh/drug200csv/data>

[2] Bộ dữ liệu thực hiện tập phổ biến là luật kết hợp: Kaggle - <https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset>

[3] Bộ dữ liệu thực hiện gom cụm: Kaggle - <https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python>

[4] Unsupervised K-Means Clustering Algorithm - [Kristina P. Sinaga](https://ieeexplore.ieee.org/author/37086947847), [Miin-Shen Yang](https://ieeexplore.ieee.org/author/37334896100): <https://ieeexplore.ieee.org/abstract/document/9072123>