BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC KINH TẾ TP HỒ CHÍ MINH

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

# BÁO CÁO ĐỒ ÁN KHAI PHÁ DỮ LIỆU

# TÊN ĐỀ TÀI: NGHIÊN CỨU THUẬT TOÁN KHAI PHÁ TẬP PHỔ BIẾN ECLAT


**Danh Sách Nhóm:**

1. **NGUYỄN HỮU THANH - 31221021356**
2. **TRẦN MỸ THIÊN TÂN - 31221026343**
3. **HÀ VIỆT THẮNG - 31221024779**
4. **THỚI TRẦN NGỌC THẠCH - 31221025597**
5. **MAI THANH THẢO - 31221026921**

**Giảng Viên: TS. Nguyễn An Tế**


*Tp. Hồ Chí Minh, tháng 12 năm 2024*

# MỤC LỤC

# DANH MỤC HÌNH ẢNH

# DANH MỤC BẢNG BIỂU

# CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

## 1.1. Giới thiệu đề tài

Trong thời đại mà dữ liệu được thu thập và lưu trữ với khối lượng khổng lồ, việc khai thác các mẫu ẩn từ dữ liệu đang trở thành một nhu cầu thiết yếu trong nhiều lĩnh vực. Một trong những bài toán nổi bật trong khai phá dữ liệu là khai thác tập phổ biến (Frequent Itemset Mining), giúp khám phá các mối quan hệ và tương quan giữa các mục trong tập dữ liệu giao dịch hoặc quan hệ. Bài toán này đặc biệt có ý nghĩa trong việc hỗ trợ ra quyết định kinh doanh, như thiết kế danh mục sản phẩm và phân tích hành vi mua sắm của khách hàng. [5]

Ví dụ điển hình của khai thác tập phổ biến là phân tích giỏ hàng Market Basket Analysis. Quá trình này tìm kiếm các mối liên hệ giữa các mặt hàng mà khách hàng thường mua cùng nhau. Thông tin từ các mối liên kết này không chỉ cung cấp cái nhìn sâu sắc về thói quen mua sắm của khách hàng mà còn giúp các nhà bán lẻ tối ưu hóa chiến lược tiếp thị và bố trí không gian kệ hàng. Chẳng hạn, việc biết được khả năng khách hàng mua bánh mì khi mua sữa có thể giúp nhà bán lẻ đề xuất sản phẩm phù hợp và thúc đẩy doanh số bán hàng thông qua tiếp thị chọn lọc.[5]

Trong bối cảnh đó, thuật toán **ECLAT (Equivalence Class Transformation)** nổi lên như một phương pháp hiệu quả trong khai thác tập phổ biến. ECLAT sử dụng cấu trúc dữ liệu dọc để giảm thiểu chi phí tính toán, giúp nó trở thành lựa chọn ưu việt khi làm việc với các tập dữ liệu lớn. Bài toán khai thác tập phổ biến bằng ECLAT không chỉ có ý nghĩa lý thuyết mà còn mang lại giá trị thực tiễn lớn trong các ứng dụng kinh doanh và nghiên cứu.[5]

Nghiên cứu này nhóm sẽ tập trung vào việc trình bày các khái niệm cơ bản, quy trình áp dụng thuật toán ECLAT và ý nghĩa của bài toán khai thác tập phổ biến trong thực tiễn. Qua đó, nhóm hy vọng nghiên cứu này không chỉ cung cấp cái nhìn sâu sắc về thuật toán ECLAT mà còn mở ra hướng đi mới trong việc áp dụng khai thác tập phổ biến trong thực tiễn, giúp cho các doanh nghiệp tận dụng dữ liệu để phát triển hiệu quả hơn.

### 1.2. Mục tiêu nghiên cứu

Các mục tiêu của nghiên cứu này:

- **Tìm hiểu thuật toán ECLAT:** Nghiên cứu nguyên lý hoạt động, cách thức xử lý dữ liệu và khai thác tập phổ biến bằng thuật toán ECLAT.
- **Áp dụng ECLAT trên tập dữ liệu thực tế:** Sử dụng tập dữ liệu giao dịch thực tế để triển khai thuật toán ECLAT.
- **So sánh ECLAT với các thuật toán khác:** Phân tích, đánh giá ưu, nhược điểm và sự khác biệt của ECLAT so với các thuật toán phổ biến khác trong khai phá tập phổ biến (Apriori, FP-Growth) dựa trên các tiêu chí về hiệu năng, độ phức tạp và khả năng áp dụng trong thực tế.

### 1.3. Phương pháp nghiên cứu

- **Áp dụng ECLAT trên tập dữ liệu:** Sử dụng một tập dữ liệu giao dịch thực tế để triển khai thuật toán ECLAT.
- **Đánh giá hiệu quả** của thuật toán thông qua các chỉ số như thời gian xử lý, chi phí tài nguyên và độ chính xác của các tập phổ biến được khai thác.

### 1.4. Tài nguyên sử dụng

- Sử dụng ngôn ngữ lập trình **Python**
- Các thư viện hỗ trợ: **pandas, matplotlib, mlxtend**
- Bộ dữ liệu: **Market-Basket-Analysis-4.csv** (Nguồn - TS. Nguyễn An Tế)

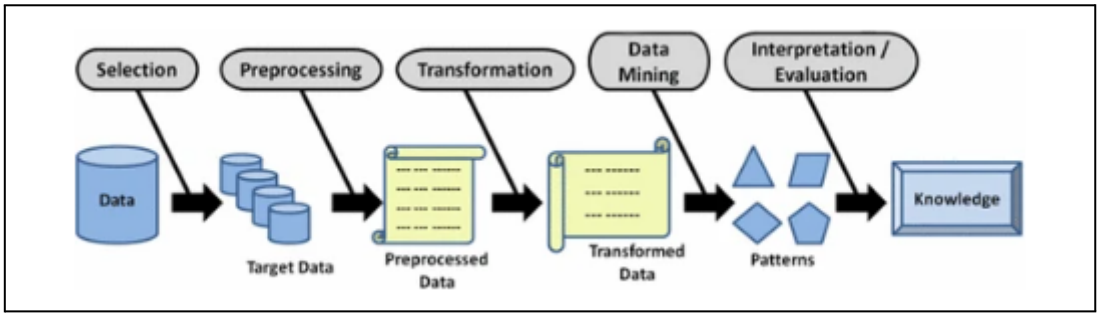
# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## 2.1. Một số khái niệm cơ bản

### 2.1.1. Khai phá dữ liệu (Data Mining)

Khai phá dữ liệu là quá trình phân loại các tập dữ liệu lớn để xác định các xu hướng chung và mối quan hệ giữa các biến để có thể giúp giải quyết các vấn đề thông qua phân tích dữ liệu. Các kỹ thuật và công cụ khai thác dữ liệu giúp doanh nghiệp dự đoán xu hướng trong tương lai và đưa ra quyết định kinh doanh sáng suốt hơn.[1]

Khai phá dữ liệu là một phần quan trọng của quá trình phân tích dữ liệu và là một trong những công việc cốt lõi trong khoa học dữ liệu, sử dụng các kỹ thuật phân tích nâng cao để tìm thông tin hữu ích trong các tập dữ liệu. Nói một cách chi tiết hơn, khai phá dữ liệu là một bước trong quy trình khám phá kiến thức trong cơ sở dữ liệu (KDD)(*), một phương pháp khoa học dữ liệu để thu thập, xử lý và phân tích dữ liệu. Khai thác dữ liệu và KDD đôi khi được gọi thay thế cho nhau, nhưng chúng thường được coi là những thứ riêng biệt.




*Hình 2.1: Quá trình khám phá dữ liệu trong hệ cơ sở dữ liệu*[2]

*(*) KDD: Viết tắt của "Knowledge Discovery in Databases", là quá trình đúc kết những kiến thức hữu ích dựa trên tập dữ liệu.*

Theo Gillis và cộng sự (2024), quá trình khai phá dữ liệu sẽ dựa vào việc triển khai việc thu thập, lưu trữ và xử lý dữ liệu một cách hiệu quả. Quá trình này ngoài ra còn có thể được sử dụng để mô tả một tập dữ liệu mục tiêu, dự đoán kết quả, phát hiện gian lận hoặc các vấn đề bảo mật, tìm hiểu thêm về cơ sở người dùng hoặc phát hiện

các nút thắt và sự phụ thuộc. Nó cũng có thể được thực hiện tự động hoặc bán tự động.

### 2.1.2. Tập phổ biến (Frequent Itemset)

Cho một tập hạng mục X và cơ sở dữ liệu D. Tập X là phổ biến trong D nếu sup(X) >= minsup, với minsup là ngưỡng hỗ trợ tối thiểu do người dùng đặt trước.

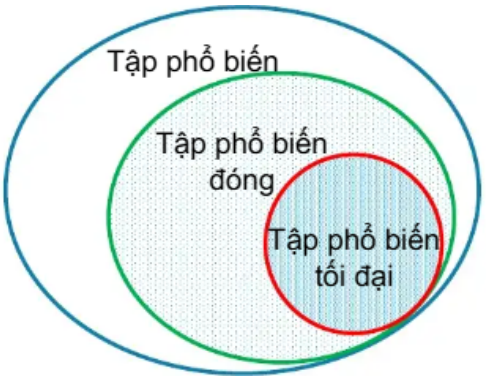
| TID | Items |
|---|---|
| 1 | A, C, T, W |
| 2 | C, D, W |
| 3 | A, C, T, W |
| 4 | A, C, D, W |
| 5 | A, C, D, T, W |
| 6 | C, D, T |

**Ví dụ:** minsup = 70%. Có sup(A) = 66.67% < minsup; sup(C) = 100% > minsup. Vậy A không phải là tập phổ biến, C là tập phổ biến.

- **Tập phổ biến đóng (closed frequent itemset):** Một tập mục X được gọi là đóng (closed) nếu không có tập cha nào của X có cùng độ hỗ trợ với nó, tức là không tồn tại một tập mục X' nào mà X' ⊃ X và t(X) = t(X') (với t(X) và t(X') tương ứng là tập các giao chứa tập mục X và X'). Ký hiệu tập phổ biến đóng là FCI.

- **Tập phổ biến lớn nhất (maximal frequent itemset):** Nếu X là phổ biến và không tập cha nào của X là phổ biến, ta nói rằng X là một tập phổ biến lớn nhất

(maximal frequent itemset). Ký hiệu tập tất cả các tập phổ biến lớn nhất là MFI. Dễ thấy MFI $\subseteq$ FCI $\subseteq$ FI.




### 2.1.3. Luật kết hợp (Association Rules)

Khái niệm Luật kết hợp được giới thiệu lần đầu tiên trong bài "Mining association rules between sets of items in large databases" của Agrawal và cộng sự (1993). Mục đích của Luật kết hợp là được dùng để tìm ra các mối tương quan, các mẫu thường gặp, các liên kết hoặc các cấu trúc ngẫu nhiên giữa tập hợp các mục trong hệ cơ sở dữ liệu giao dịch. Luật kết hợp được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau như mạng viễn thông, quản lý thị trường và rủi ro, kiểm soát hàng tồn kho và được ứng dụng nhiều nhất trong lĩnh vực kinh doanh trên cơ sở tìm hiểu liệu khách hàng nếu mua sản phẩm A, B có tiếp tục mua sản phẩm C nào đó hay không.[3]

Cho I={$I_1$, $I_2$, .., $I_m$} là tập hợp của m tính chất riêng biệt. Giả sử D là CSDL, với các bản ghi chứa một tập con T các tính chất (có thể coi như ), các bản ghi đều có chỉ số riêng. Một luật kết hợp là một mệnh đề kéo theo có dạng X->Y, trong đó X, Y $\subseteq$ I, thỏa mãn điều kiện X $\cap$ Y = Ø. Các tập hợp X và Y được gọi là các tập hợp tính chất (itemset). Tập X gọi là nguyên nhân, tập Y gọi là hệ quả.[4]

Có 2 độ đo quan trọng đối với luật kết hợp: Độ hỗ trợ (support) và độ tin cậy (confidence).

➢ ***Độ hỗ trợ:***

Độ hỗ trợ của một tập hợp X trong cơ sở dữ liệu D là tỉ số giữa các bản ghi T ⊆ D có chứa tập X và tổng số bản ghi trong D (hay là phần trăm của các bản ghi trong D có chứa tập hợp X), ký hiệu là support(X) hay supp(X) (support sẽ tự sinh ra khi cài thuật toán).

$$S_0 = \frac{|\{T \subset D : Y \supset X\}|}{|D|}$$

Ta có: 0 ≤ supp(X) ≤ 1 với mọi tập hợp X

Độ hỗ trợ của một luật kết hợp XY là tỷ lệ giữa số lượng các bản ghi chứa tập hợp X ∪ Y, so với tổng số các bản ghi trong D - Ký hiệu supp(X->Y).

$$\text{Supp}(X \rightarrow Y) = \frac{|\{T \subset D : T \supseteq X \cup Y\}|}{|D|}$$

➢ ***Độ tin cậy:***

Độ tin cậy của một luật kết hợp X->Y là tỷ lệ giữa số lượng các bản ghi trong D chứa X ∪ Y với số bản ghi trong D có chứa tập hợp X. Ký hiệu độ tin cậy của một luật là conf(r). Ta có 0 ≤ conf(r) ≤ 1

Nhận xét: Độ hỗ trợ và độ tin cậy có xác suất sau:

- supp(X->Y) = P(X ∪ Y)
- conf(X->Y) = P(X/Y) = supp(X ∪ Y)/supp(X)

Hoặc có thể nói, độ tin cậy của một luật kết hợp X->Y là tỷ lệ giữa số lượng các bản ghi của tập hợp chứa X ∪ Y, so với tổng số các bản ghi chứa X.

## 2.2. Bài toán khai thác tập phổ biến

### 2.2.1. Định nghĩa bài toán và ứng dụng thực tế

Bài toán khai thác tập phổ biến (frequent itemset) là một vấn đề cốt lõi trong lĩnh vực khai phá dữ liệu. Nhiệm vụ chính của bài toán là xác định tất cả các tập mẫu,

liên kết, tương quan hoặc cấu trúc nhân quả có tần suất xuất hiện cao trong tập hợp các hạng mục hoặc đối tượng từ cơ sở dữ liệu giao dịch, cơ sở dữ liệu quan hệ, và các kho thông tin dữ liệu khác.

Bạn đã bao giờ thắc mắc vì sao các sản phẩm trong siêu thị lại được sắp xếp cạnh nhau? Tại sao sản phẩm A được đặt ở dưới, còn sản phẩm B lại nằm trên cao? Hoặc tại sao những món đồ bạn thường cần lại luôn xuất hiện cạnh nhau trong siêu thị? Những điều trên chính là những ví dụ điển hình về ứng dụng của khai phá dữ liệu, mà cụ thể là khai thác các tập phổ biến.

Bài toán khai thác các tập phổ biến được áp dụng rộng rãi trong nhiều lĩnh vực, nổi bật nhất là **Basket Data Analysis – phân tích giỏ hàng** (dự đoán và gợi ý các món hàng thường được mua kèm với một sản phẩm đã chọn trước đó). Ngoài ra, bài toán này còn được ứng dụng trong các lĩnh vực như: tiếp thị chéo, thiết kế danh mục sản phẩm, phân tích thua lỗ, phân cụm, phân loại, hệ thống khuyến nghị, v.v. Đặc biệt, nó còn có tiềm năng trong việc hỗ trợ thiết kế các dịch vụ tiện ích cho nhà thông minh.

### 2.2.2. Thách thức trong khai phá tập phổ biến

Một trong những thách thức trong việc khai phá tập phổ biến đó chính là việc sẽ tạo ra một lượng lớn các luật kết hợp. Điều này xảy ra sẽ dẫn tới việc những luật được tạo ra sẽ thiếu tính liên quan, từ đó giúp việc xác định các mẫu (pattern) trở nên khó khăn hơn. Ngoài ra, sự hạn chế khi thực hiện việc phát hiện/xác định các mối quan hệ phức tạp giữa các items với nhau cũng đang là một thách thức rất lớn khi khai phá tập phổ biến, khi mà nó chỉ cân nhắc và xem xét sự đồng xuất hiện của các mục trong cùng một giao dịch.

Về mặt tính toán, thuật toán này có thể đòi hỏi chi phí lớn khi số lượng các items và số lượng giao dịch tăng lên sẽ kéo theo số lượng các itemset cũng sẽ gia tăng. Cuối cùng, việc xác định ngưỡng hỗ trợ tối thiểu (minimum support) và độ tin cậy tối thiểu (minimum confidence) trước khi thực hiện khai phá là điều bắt buộc, nhưng thường rất khó để thực và yêu cầu một hiểu biết sâu sắc về dữ liệu từ người khai thác.

## 2.3. Các thuật toán khai thác tập phổ biến

Dưới đây là một số thuật toán tiêu biểu được dùng trong khai thác tập phổ biến

### 2.3.1. Thuật toán APRIORI:

Apriori là một thuật toán được cải tiến từ thuật toán AIS, được giới thiệu cùng lúc với vấn đề khai thác tập phổ biến (frequent sets). Điểm cải tiến lớn nhất của Apriori là khai thác tính đơn điệu của độ hỗ trợ (support) của tập hợp, nghĩa là nếu một tập con không phổ biến thì tất cả các tập cha của nó cũng không phổ biến. Thuật toán này được thực hiện qua các bước sau: [6]

- Duyệt theo mức độ (Level-wise Search): Tìm các tập phổ biến kích thước nhỏ ( từ kích thước 1) trước, sau đó mở rộng lên kích thước lớn hơn (2, 3, ...)
- Tạo tập ứng viên (Candidate Generation): Sử dụng bước *join* để kết hợp các tập phổ biến hiện tại, sau đó dùng bước *prune* để loại bỏ các tập không khả thi.[6]
- Đếm độ hỗ trợ (Counting Support): Quét qua cơ sở dữ liệu để đếm số lần xuất hiện của các tập ứng viên trong từng giao dịch và lọc ra các tập phổ biến.

*Ưu điểm*: dễ hiểu và dễ áp dụng trong các bài toán thực tế.

*Nhược điểm*: khi thực hiện trên dữ liệu lớn, hiệu suất kém vì cần quét dữ liệu nhiều lần và tạo nhiều tập ứng viên.

### 2.3.2. Thuật toán ECLAT:

Eclat sử dụng cách tiếp cận dựa trên cơ sở dữ liệu dạng dọc (vertical layout), trong đó mỗi mục được liên kết với danh sách các giao dịch chứa nó. Tập phổ biến được tìm bằng cách giao nhau (intersection) giữa các danh sách này.[6]

- Duyệt sâu (Depth-First Search): Sử dụng chiến lược duyệt sâu thay vì duyệt theo mức như Apriori.
- Cơ sở dữ liệu điều kiện: Chỉ lọc các giao dịch liên quan đến tập hợp đang xét, giảm kích thước dữ liệu cần xử lý.
- Tối ưu bằng Diffset: Có thể lưu trữ sự khác biệt giữa các danh sách thay vì lưu toàn bộ danh sách, giúp giảm bộ nhớ. [6]

*Ưu điểm:* ít tốn bộ nhớ hơn và hiệu quả hơn với dữ liệu lớn.

*Nhược điểm:* vì không sử dụng bước *prune* nên có thể tạo nhiều tập ứng viên không cần thiết.

### 2.3.3. Thuật toán FP-Growth

FP-Growth là một cải tiến của Apriori, sử dụng cấu trúc cây FP (Frequent Pattern Tree) để lưu trữ dữ liệu. Thay vì tạo tập ứng viên, thuật toán nén dữ liệu vào cây FP và khai thác mẫu từ cây. [6]

- Xây dựng cây FP: Mỗi giao dịch được thêm vào cây theo thứ tự các mục phổ biến, giúp nén dữ liệu; mỗi phần tử có một danh sách liên kết với giao dịch chưa nó.
- Duyệt cây: Tìm các tập phổ biến bằng cách khai thác các tiền tố chung, thay vì quét toàn bộ cơ sở dữ liệu.

Ưu điểm: Giảm số lần quét dữ liệu (chỉ cần hai lần); hiệu quả với cơ sở dữ liệu dày đặc hoặc có nhiều giao dịch trùng lặp.

Nhược điểm: Tốn nhiều bộ nhớ; khi xây dựng và duyệt cây FP có thể phức tạp với cơ sở dữ liệu lớn.

## 2.4. Thuật toán ECLAT

Thuật toán ECLAT được viết tắt cho cụm "Equivalence Class Clustering and Bottom-up Lattice Traversal". Đây là một trong những phương pháp phổ biến của khai thác luật kết hợp và là phiên bản hiệu quả và có thể mở rộng hơn của thuật toán Apriori. Trong khi thuật toán Apriori hoạt động theo chiều ngang mô phỏng Breadth-First Search của đồ thị, thì thuật toán ECLAT hoạt động theo chiều dọc giống như Depth-First Search của đồ thị. Cách tiếp cận theo chiều dọc này của thuật toán ECLAT khiến nó hoạt động và trả kết quả nhanh hơn khi so sánh với thuật toán Apriori.

➢ ***Cách hoạt động của thuật toán***

Ý tưởng cơ bản của thuật toán là sử dụng các giao điểm Transaction Id Sets (tidsets) để tính toán giá trị hỗ trợ (support value) của một tập ứng viên (candidate) và tránh việc tạo ra các tập hợp con không tồn tại trong cây tiền tố (prefix tree). Trong lần

gọi hàm đầu tiên, tất cả các mục đơn lẻ được sử dụng cùng với các tidset của chúng. Sau đó, hàm được gọi đệ quy và trong mỗi lần gọi đệ quy, mỗi cặp item-tidset được xác minh và kết hợp với các cặp item-tidset khác. Quá trình này được tiếp tục cho đến khi không có cặp item-tidset ứng viên nào có thể được kết hợp được nữa.

➢ ***Ví dụ minh họa*** [7]

| Transaction Id | Bread | Butter | Milk | Coke | Jam |
|---|---|---|---|---|---|
| T1 | 1 | 1 | 0 | 0 | 1 |
| T2 | 0 | 1 | 0 | 1 | 0 |
| T3 | 0 | 1 | 1 | 0 | 0 |
| T4 | 1 | 1 | 0 | 1 | 0 |
| T5 | 1 | 0 | 1 | 0 | 0 |
| T6 | 0 | 1 | 1 | 0 | 0 |
| T7 | 1 | 0 | 1 | 0 | 0 |
| T8 | 1 | 1 | 1 | 0 | 1 |
| T9 | 1 | 1 | 1 | 0 | 0 |

Dữ liệu được đưa ra ở bảng trên là một ma trận boolean trong đó đối với mỗi ô (i, j), giá trị biểu thị liệu mục thứ j có được bao gồm trong giao dịch thứ i hay không. 1 có nghĩa là có trong khi 0 có nghĩa là không. Đầu tiên chúng ta sẽ đặt giá trị **min_support = 3** và sau đó chạy thuật toán.

**Bước 1: chuyển dữ liệu sang dạng dọc**

| Item | Tidset |
|---|---|
| Bread | {T1, T4, T5, T7, T8, T9} |
| Butter | {T1, T2, T3, T4, T6, T8, T9} |
| Milk | {T3, T5, T6, T7, T8, T9} |
| Coke | {T2, T4} |
| Jam | {T1, T8} |

**Bước 2: lọc ra các item ban đầu khi không thỏa mãn min_support = 3**

| Item | Tidset |
|---|---|
| Bread | {T1, T4, T5, T7, T8, T9} |
| Butter | {T1, T2, T3, T4, T6, T8, T9} |
| Milk | {T3, T5, T6, T7, T8, T9} |

Bây giờ chúng ta gọi hàm theo cách đệ quy cho đến khi không còn cặp mục-tidset nào có thể kết hợp được nữa.

**Bước 3: khởi động vòng lặp để duyệt theo chiều sâu, bắt đầu từ Bread**

| ItemSet | Tidset |
|---|---|
| {Bread, Butter} | {T1, T4, T8, T9} |
| {Bread, Milk} | {T5, T7, T8, T9} |

**Bước 4: gọi đệ quy để duyệt tiếp tục cho {Bread, Butter}**

| ItemSet | Tidset |
|---|---|
| {Bread, Butter, Milk} | {T8, T9} |

**Bước 5: trả về danh sách itemset phổ biến khi duyệt xong nhánh Bread**

| ItemSet | Tidset |
|---|---|
| {Bread, Butter} | {T1, T4, T8, T9} |
| {Bread,  Milk} | {T5, T7, T8, T9} |

**Bước 6: quay lại vòng lặp ban đầu duyệt tiếp Butter theo chiều sâu**

| ItemSet | Tidset |
|---|---|
| {Butter, Milk} | {T3, T6, T8, T9} |

**Bước 7: trả về danh sách itemset phổ biến sau khi thực hiện hết vòng lặp ban đầu**

| ItemSet | Tidset |
|---|---|
| {Bread, Butter} | {T1, T4, T8, T9} |
| {Bread,  Milk} | {T5, T7, T8, T9} |
| {Butter, Milk} | {T3, T6, T8, T9} |

➢ ***So sánh thuật toán ECLAT với các thuật toán tương tự (Apriori và FP-Growth)***

| | **Thời gian chạy** | **Cấu trúc dữ liệu** | **Phương pháp tìm kiếm** | **Tần suất tính toán (số lần tính toán độ hỗ trợ của các item)** |
|---|---|---|---|---|
| **ECLAT** | Xét với bộ dữ liệu lớn, có nhiều mục, giao dịch dài thì ECLAT có thời gian chạy nhanh hơn Apriori | Thuật toán ECLAT sử dụng định dạng dữ liệu dọc (vertical data format), trong đó mỗi mục là một tập hợp các giao dịch chứa nó | ECLAT sử dụng phương pháp tìm kiếm đệ quy | ECLAT tính tần suất xuất hiện của các tập luật kết hợp bằng cách đếm số lần xuất hiện của các sản phẩm trong các giao dịch |
| **Apriori** | Phù hợp với các tập dữ liệu nhỏ hoặc có kích thước trung bình nhưng khi áp dụng vào bộ dữ liệu lớn thì thời gian chạy chậm hơn ECLAT và FP-Growth | Apriori sử dụng cấu trúc dữ liệu dạng tập hợp (Horizontal Format). Như vậy, Apriori phải thực hiện nhiều lần lặp và tính toán tần suất xuất hiện nhiều hơn (Ứng với mỗi tập hợp mới) | Apriori sử dụng phương pháp tìm kiếm theo chiều ngang (Level-Wise Search) | Apriori tính tần suất xuất hiện bằng cách so sánh các tập luật kết hợp với các giao dịch |

| | | | | |
|---|---|---|---|---|
| **FP-Growth** | Thuật toán FP-Growth có thời gian chạy tối ưu và có hiệu suất cao đối với các tập dữ liệu lớn | FP-Growth sử dụng cấu trúc cây FP để lưu trữ và khai thác các tập phổ biến. Cây FP là một cấu trúc cây có gốc là null, mỗi cạnh chứa một mục, và mỗi nút biểu diễn một tập mục là nối các mục trên đường đi từ gốc đến nút đó | FP-Growth sử dụng phương pháp phát triển mẫu, tức là không sinh ra các tập ứng viên mà chỉ sử dụng cấu trúc cây FP để lưu trữ và khai thác các tập phổ biến. | FP-Growth có tần suất tính toán thấp nhất, vì nó không cần sinh ra các tập ứng viên mà chỉ sử dụng cấu trúc cây FP để lưu trữ và khai thác các tập phổ biến. |

**Hiệu suất:**

1. FP-Growth thường có hiệu suất tốt nhất trên các tập dữ liệu lớn và ít tốn thời gian so với Apriori.
2. ECLAT có thể hiệu quả tương đương hoặc vượt trội so với FP-Growth trong một số trường hợp.

**Bộ nhớ:**

1. ECLAT thường tốn ít bộ nhớ hơn FP-Growth vì không cần xây dựng cây.
2. FP-Growth yêu cầu ít bộ nhớ hơn Apriori vì không cần lưu trữ itemsets candidate.

# CHƯƠNG 3: KHÁM PHÁ BỘ DỮ LIỆU

## 3.1. Giới thiệu bộ dữ liệu

Bộ dữ liệu "Market Basket Analysis 4.csv" chứa thông tin về các giao dịch mua sắm tại một siêu thị, nhằm mục đích  tìm ra các luật kết hợp cho những mặt hàng từ đó hiểu được hành vi mua sắm và đưa ra những giải pháp hợp lý để tăng lợi nhuận bán hàng.

## 3.2. Khám phá dữ liệu

Bộ dữ liệu ban đầu, ta có:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | whole milk | pastry | salty snack | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | sausage | whole milk | semi-finished bread | yogurt | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | soda | pickled vegetables | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | canned beer | misc. beverages | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | sausage | hygiene articles | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 14958 | butter milk | whipped/sour cream | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 14959 | bottled water | herbs | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 14960 | fruit/vegetable juice | onions | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 14961 | bottled beer | other vegetables | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 14962 | soda | root vegetables | semi-finished bread | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

14963 rows × 11 columns

Bộ dữ liệu bao gồm 14,963 giao dịch. Trong đó số lượng sản phẩm nhiều nhất trong một giao dịch là 11 sản phẩm.

Nhận thấy khi sử dụng data frame dạng bảng sẽ khó xử lý theo từng giao dịch. Ngoài ra, các sản phẩm có dấu ‘/’. Đoạn mã sau dùng để tìm ra giá trị nói trên.

**Đoạn mã tìm tất cả các giá trị chứa dấu “/”**

```
values_with_slash = set()  # Dùng set để lưu giá trị không trùng lặp

for column in data.columns:  #
    matches = data[column][data[column].astype(str).str.contains("/", na=False)]
```

```
    values_with_slash.update(matches.tolist())

print('Danh sách tên các mặt hàng có dấu "/":')
print(values_with_slash)
```

```
Danh sách tên các mặt hàng có dấu "/":
{'cling film/bags', 'whipped/sour cream', 'rolls/buns', 'red/blush wine', 'fruit/vegetable juice',
'photo/film', 'packaged fruit/vegetables', 'nuts/prunes', 'flower soil/fertilizer'}
```

Kết quả cho về cho thấy rằng, dấu "/" được dùng để nhóm các sản phẩm chung vào cùng một loại hàng hóa. Đây không phải là dấu phân tách hai mặt hàng riêng biệt nên do đó không cần phải tách riêng ra.

**Đoạn mã tính tổng số lượng mặt hàng**

```
# Kết hợp tất cả các mặt hàng trong tập dữ liệu
all_items = data.stack().dropna().tolist()

# Loại bỏ các mặt hàng giống nhau
unique_items = set(all_items)

total_unique_items = len(unique_items)
print("Tổng số lượng mặt hàng duy nhất trong tất cả giao dịch:",
total_unique_items)
```

Đoạn mã này đầu tiên sẽ tiến hành loại bỏ các giá trị thiếu (thường được biểu thị là NaN) khỏi chuỗi. Sau đó, sẽ thực hiện việc chuyển đổi chuỗi kết quả nói trên thành một danh sách, danh sách này sẽ chứa các mặt hàng đã mua trong tất cả các giao dịch mà không chứa missing values nào. Tiếp theo, đoạn mã sẽ tiến hành chuyển đổi danh sach trên thành một set là "unique_items" và set này sẽ tự động loại bỏ các phần tử trùng lặp, chỉ để lại các mặt hàng duy nhất. Cuối cùng, nó sẽ đếm và in ra tổng số mặt hàng đã mua.

```
Tổng số lượng mặt hàng trong tất cả giao dịch: 167
```

Kết quả cho thấy, có 167 mặc hàng riêng biệt được mua trong tổng số 14,963 giao dịch.

**Đoạn mã phân tích tần suất xuất hiện của các mặt hàng trong dữ liệu và trực quan hóa chúng bằng biểu đồ**

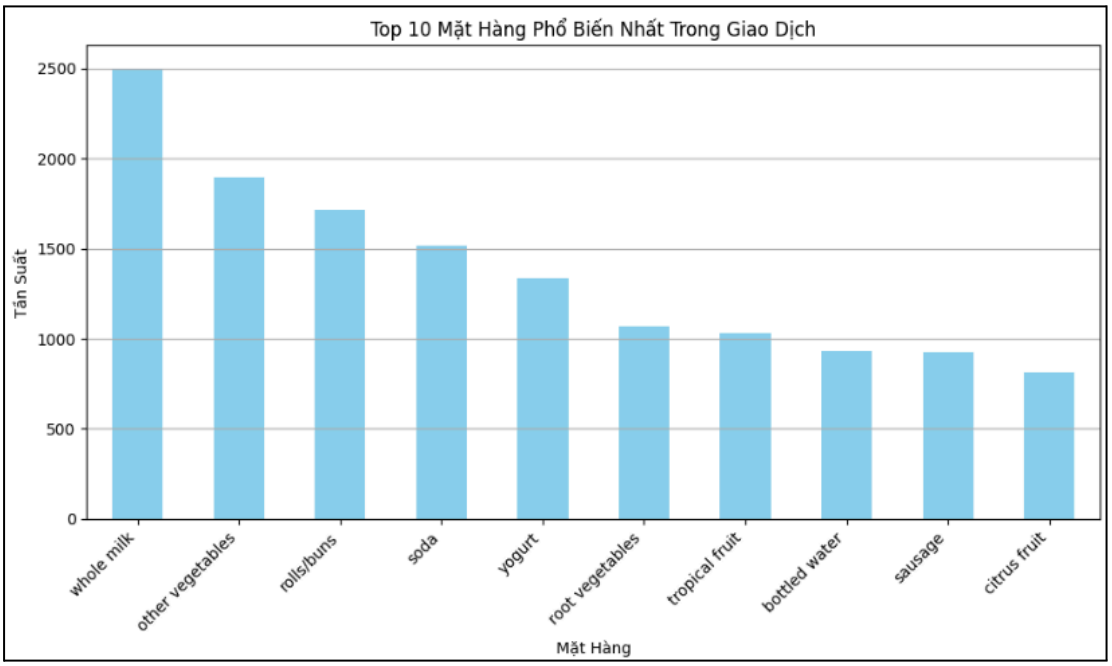
```
# Kết hợp tất cả các mặt hàng để tính tần suất
all_items = data.stack().dropna().tolist()
item_frequencies = pd.Series(all_items).value_counts()
```

Đoạn mã này phân tích dữ liệu giao dịch, tính toán tần suất xuất hiện của từng mặt hàng, giúp người dùng dễ dàng nhận biết được những mặt hàng nào được mua nhiều nhất và ít nhất.

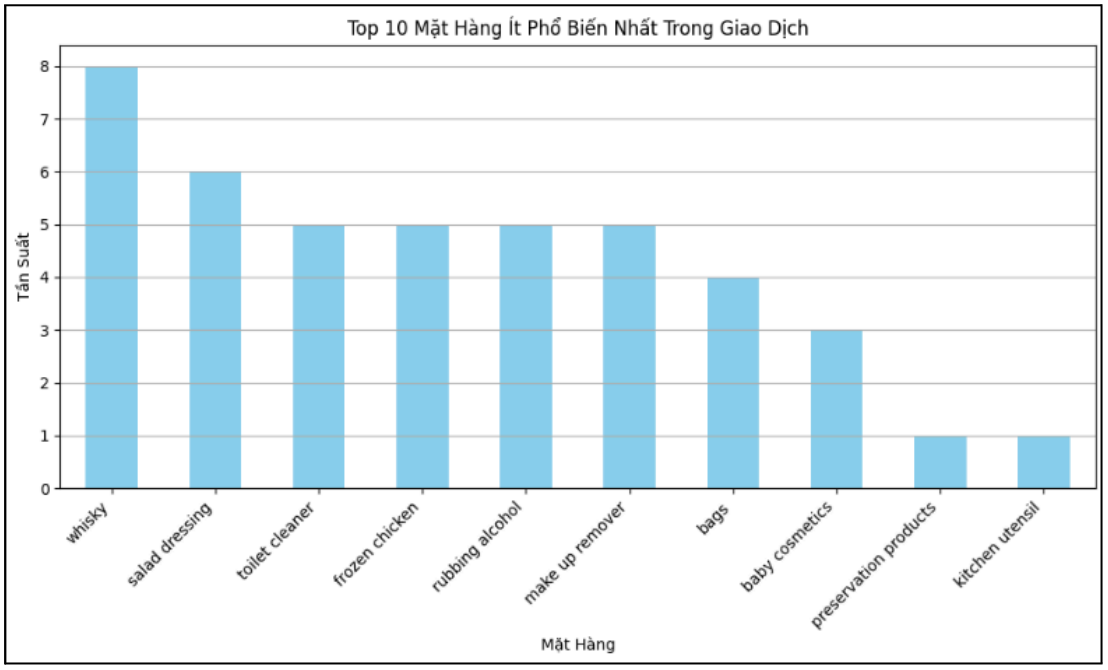
**Đoạn mã trực quan hóa dữ liệu các giao dịch**

```
# Vẽ biểu đồ tần suất của 10 mặt hàng phổ biến nhất
top_items = item_frequencies.head(10)

# Thiết lập biểu đồ
plt.figure(figsize=(10, 6))
top_items.plot(kind='bar', color='skyblue')
plt.title("Top 10 Mặt Hàng Phổ Biến Nhất Trong Giao Dịch")
plt.xlabel("Mặt Hàng")
plt.ylabel("Tần Suất")
plt.xticks(rotation=45, ha="right")
plt.grid(axis='y')
plt.tight_layout()
plt.show()

# Vẽ biểu đồ tần suất của 10 mặt hàng phổ biến nhất
bot_items = item_frequencies.tail(10)

# Thiết lập biểu đồ
plt.figure(figsize=(10, 6))
bot_items.plot(kind='bar', color='skyblue')
plt.title("Top 10 Mặt Hàng Ít Phổ Biến Nhất Trong Giao Dịch")
plt.xlabel("Mặt Hàng")
plt.ylabel("Tần Suất")
plt.xticks(rotation=45, ha="right")
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

Sau khi đã tính được tần suất xuất hiện của các mặt hàng, đoạn mã này vẽ hai biểu đồ cột để trực quan hóa top 10 mặt hàng phổ biến nhất và top 10 mặt hàng ít phổ biến nhất. Điều này sẽ giúp người dùng dễ dàng nhận biết được những mặt hàng nào được mua nhiều nhất và ít nhất.




*Hình 3.1: Biểu đồ Top 10 mặt hàng phổ biến nhất trong giao dịch*



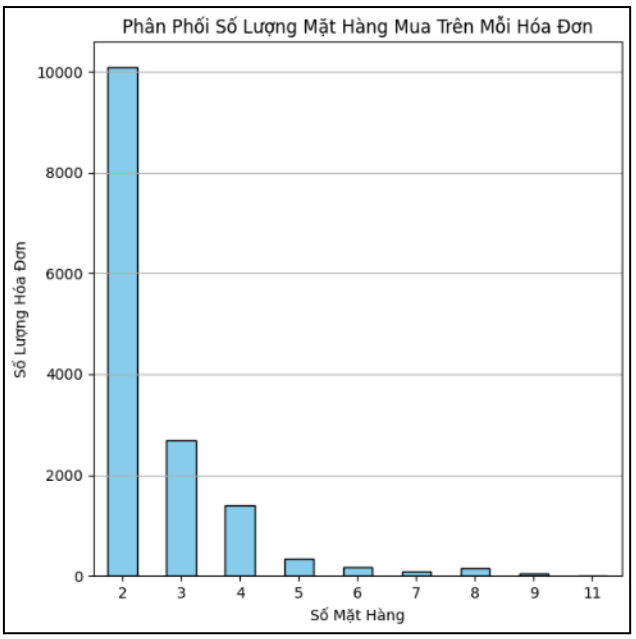
*Hình 3.2: Biểu đồ Top 10 mặt hàng ít phổ biến nhất trong giao dịch*

Từ 2 hình trên, nhận thấy các mặt hàng ít phổ biến có tần suất xuất hiện rất thấp (dưới 10 giao dịch). Hơn nữa, tần suất xuất hiện của các mặt hàng phổ biến lại rất cao, điều này sẽ ảnh hưởng đến kết quả của luật phổ biến khi dựa vào độ đo *confidence.*

**Đoạn mã trực quan hóa dữ liệu các mặt hàng**

```
# Đếm số mặt hàng không phải NaN trên mỗi hóa đơn
item_counts_per_invoice = data.notna().sum(axis=1)

# Đếm số lần xuất hiện của mỗi số lượng mặt hàng
item_count_distribution = item_counts_per_invoice.value_counts().sort_index()

# Vẽ biểu đồ cột
plt.figure(figsize=(6, 6))
item_count_distribution.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title("Phân Phối Số Lượng Mặt Hàng Mua Trên Mỗi Hóa Đơn")
plt.xlabel("Số Mặt Hàng")
plt.ylabel("Số Lượng Hóa Đơn")
plt.grid(axis='y')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

Đoạn code trên dùng để cung cấp thông tin chi tiết về số lượng mặt hàng trung bình mà khách hàng mua trong một giao dịch bằng cách tạo phân phối và trực quan hóa nó bằng biểu đồ cột.




*Hình 3.3: Biểu đồ Phân phối số lượng mặt hàng mua trên mỗi hóa đơn*

Phần lớn các hóa đơn chỉ chứa 2 mặt hàng, chiếm số lượng lớn nhất (khoảng hơn 10.000 hóa đơn). Điều này thể hiện xu hướng khách hàng thường mua số lượng ít mặt hàng trong mỗi lần mua sắm.

Số lượng hóa đơn giảm rõ rệt khi số mặt hàng trên hóa đơn tăng. Từ 3 đến 4 mặt hàng, số hóa đơn vẫn khá đáng kể nhưng giảm mạnh so với 2 mặt hàng. Khi số mặt hàng vượt quá 5, số lượng hóa đơn trở nên rất ít.

### 3.3. Chuyển đổi dữ liệu

Một bước quan trọng trong việc xử lý dữ liệu để áp dụng thuật toán Eclat là chuyển đổi dữ liệu từ dạng ngang (horizontal) sang dạng dọc (vertical). Trong dữ liệu dạng ngang, mỗi giao dịch được lưu trữ dưới dạng danh sách các mặt hàng đi cùng nhau.  Ngược lại, trong dữ liệu dạng dọc, thông tin được tổ chức theo từng mặt hàng, kèm theo danh sách các giao dịch chứa mặt hàng đó. Việc chuyển đổi sang dữ liệu dạng dọc giúp việc tính toán tập phổ biến nhanh chóng hơn khi áp dụng phép giao trên các danh sách giao dịch. [6]

**Đoạn mã để đọc file CSV và chuyển thành danh sách giao dịch**

```
# Đọc file CSV và chuyển thành danh sách giao dịch
def read_transactions_from_csv(file_path):
    transactions = []
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            transactions.append([item.strip() for item in row if item.strip()])
    return transactions
```

Hàm này sẽ nhận một file CSV, đọc nó theo từng dòng, làm sạch từng phần tử trong dòng và tạo ra một danh sách lồng nhau (nested list), trong đó mỗi danh sách con đại diện cho một giao dịch chứa các mặt hàng đã mua.

Dưới đây là danh sách lồng nhau *(lưu ý: danh sách này đã được chuyển đổi để có thể biểu diễn dưới dạng bảng)*:

| | Transaction |
|---|---|
| 0 | whole milk, pastry, salty snack |
| 1 | sausage, whole milk, semi-finished bread, yogurt |
| 2 | soda, pickled vegetables |
| 3 | canned beer, misc. beverages |
| 4 | sausage, hygiene articles |
| … | … |
| 14958 | butter milk, whipped/sour cream |
| 14959 | bottled water, herbs |
| 14960 | fruit/vegetable juice, onions |
| 14961 | bottled beer, other vegetables |
| 14962 | soda, root vegetables, semi-finished bread |

*Bảng 3.4: Danh sách giao dịch hiển thị ở dạng bảng*

**Đoạn mã chuyển đổi giao dịch từ dạng ngang sang dạng dọc**

```
# Chuyển đổi giao dịch từ dạng ngang sang dạng dọc
def convert_to_vertical_format(transactions):
    vertical_db = {}
    for transaction_id, transaction in enumerate(transactions):
        for item in transaction:
            if item not in vertical_db:
                vertical_db[item] = []
            vertical_db[item].append(transaction_id)
    return vertical_db
```

Từ danh sách lồng nhau được chuyển đổi từ file csv ở trên. Nhóm tiếp tục tiến hành chuyển đổi thành dữ liệu dạng dọc được lưu trữ dưới cấu trúc dữ liệu từ điển *(dictionary)*. Trong đó, các cặp khóa (key) - giá trị (value) tương ứng là tên sản phẩm và danh sách giao dịch có chứa sản phẩm đó.

Dưới đây là dữ liệu dạng dọc đã được chuyển đổi *(lưu ý: dữ liệu này đã được chuyển đổi một lần nữa để có thể biểu diễn dưới dạng bảng)*:

| | Item | Transactions |
|---|---|---|
| 0 | pastry | 0, 21, 37, 43, 117, 121, 164, 202, 277, 292, 3... |
| 1 | salty snack | 0, 63, 101, 144, 212, 219, 326, 330, 387, 392,... |
| 2 | whole milk | 0, 1, 5, 6, 10, 20, 21, 24, 29, 31, 45, 48, 51... |
| 3 | yogurt | 1, 33, 35, 36, 45, 47, 60, 70, 79, 84, 93, 100... |
| 4 | semi-finished bread | 1, 264, 379, 404, 478, 737, 752, 780, 807, 950... |
| … | … | … |
| 162 | frozen chicken | 3085, 6883, 8513, 10776, 13827 |
| 163 | salad dressing | 3974, 4305, 5882, 6906, 7716, 8799 |
| 164 | specialty vegetables | 3990, 4044, 7743, 7805, 8828, 10775, 10782, 11... |

| | | |
|---|---|---|
| 165 | toilet cleaner | 4990, 5524, 8408, 10991, 13781 |
| 166 | rubbing alcohol | 5349, 6740, 7443, 13063, 14833 |

*Bảng 3.5: Cơ sở dữ liệu theo chiều dọc*

# CHƯƠNG 4: THUẬT TOÁN ECLAT VÀ LUẬT KẾT HỢP

## 4.1. Thuật toán ECLAT

**Lọc các mục bổ phiến ban đầu**

```
def eclat_algorithm(vertical_db, total_transactions, min_sup):
    # Lọc ra items và tid_lists có frequency >= min_sup * total_transactions
    items, tid_lists = zip(*[(key, set(tid_list)) for key, tid_list in vertical_db.items() if
len(tid_list) >= min_sup * total_transactions])
```

Đầu tiên, hàm **eclat_algorithm** sẽ lọc ra danh sách sản phẩm **(items)** và danh sách giao dịch tương ứng **(tid_lists)** có tần suất xuất hiện (frequency) thỏa mãn giá trị min_sup được đưa vào.

**Hàm đệ quy**

```
    # Tạo hàm để thực hiện đệ quy
    def find_frequent_itemsets(prefix, items, tid_lists, min_sup, frequent_itemsets):
        if not items:
            return frequent_itemsets

        for i in range(len(items)):
            item = items[i]
            tid_list = tid_lists[i]
            support = len(tid_list) / total_transactions

            # Kiểm tra điều kiện min_sup
            if support >= min_sup:
                frequent_itemsets.append((prefix + [item], support))

                new_prefix = prefix + [item]
                new_items = items[i + 1:]
                new_tid_lists = [
                    tid_list & other_tid_list
                    for other_tid_list in tid_lists[i + 1:]
                ]

                find_frequent_itemsets(new_prefix, new_items, new_tid_lists, min_sup,
frequent_itemsets)

        return frequent_itemsets

    frequent_itemsets = find_frequent_itemsets([], items, tid_lists, min_sup, [])
```

```
    return frequent_itemsets
```

Tiếp theo, hàm **find_frequent_itemsets** được sử dụng để thực hiện tìm kiếm đệ quy các tập phổ biến. Hàm này bắt đầu với một tiền tố (prefix) rỗng và duyệt qua tất cả các mục trong **items**. Đối với mỗi mục, hàm tính toán sẽ tính độ hỗ trợ (support) bằng cách chia số lượng giao dịch chứa nó cho tổng số giao dịch.

Nếu độ hỗ trợ này lớn hơn hoặc bằng **min_sup**, tập phổ biến mới được tạo ra. Sau đó, thực hiện chuẩn bị dữ liệu cho lần gọi đệ quy tiếp theo bao gồm **new_prefix** lưu trữ itemset phổ biến hiện tại đang được xét, **new_items** lưu trữ danh sách các items tiếp theo tính từ item đang được xét để tránh tình trạng sự kết hợp trùng lặp, **new_tid_lists** lưu trữ các danh sách giao dịch, các danh sách này là kết quả của phép giao giữa giao dịch của item đang xét theo từng giao dịch của các item tiếp theo.

Sau đó, hàm được gọi đệ quy với **new_prefix** , **new_items** và **new_tid_lists** . Chu trình này tiếp tục cho đến khi không còn mục nào trong danh sách **items**. Cuối cùng, hàm sẽ trả về kết quả là danh sách tất cả các tập phổ biến được tìm thấy.

## 4.2. Luật kết hợp

**Khởi tạo**

```
def association_rules(frequent_itemsets, min_confidence):
    rules = []

    # Tạo thư viện để lưu frequency theo các itemset phổ biến
    freq_dict = {frozenset(itemset): frequency for itemset, frequency in
frequent_itemsets}
```

Ban đầu, hàm sẽ bắt đầu bằng việc tạo ra một danh sách rỗng **rules** để lưu trữ các luật kết hợp tìm được và một từ điển **freq_dict** để lưu trữ tần suất xuất hiện của các tập phổ biến. Từ điển được tạo ra nhằm mục đích có thể truy cập **frequency** của một **itemset** nhanh chóng với độ phức tạp thấp hơn khi duyệt bằng vòng lặp.

**Vòng lặp duyệt qua các tập phổ biến**

```
    for itemset, frequency in frequent_itemsets:
        if len(itemset) >= 2:
            for i in range(1, len(itemset)):
                for subset in combinations(itemset, i):
                    antecedent = set(subset)
                    consequent = set(itemset) - antecedent

                    if not consequent:
                        continue

                    freq_antecedent = freq_dict.get(frozenset(antecedent), 0)
                    freq_consequent = freq_dict.get(frozenset(consequent), 0)

                    # Tính confidence
                    confidence = (frequency / freq_antecedent)  if freq_antecedent > 0 else
0

                    # Tính lift
                    lift = (frequency / (freq_antecedent * freq_consequent)) if
freq_antecedent >= 0 and freq_consequent >= 0 else 0

                    if confidence >= min_confidence:
                        rules.append((antecedent, consequent, frequency, confidence, lift))

    return rules
```

Hàm sau đó sẽ duyệt qua từng tập phổ biến (itemset) trong danh sách **frequent_itemsets**. Đối với mỗi tập phổ biến phải có ít nhất 2 mục để có thể tại luật kết hợp. Ở vòng lặp tiếp theo **hàm combinations** sẽ tạo ra tất cả các tập con (subset) có thể có của nó. Mỗi tập con được coi là tiền đề (antecedent) của một luật kết hợp, phần còn lại của tập phổ biến được coi là kết quả (consequent).

Tiếp theo từ thư viện ban  đầu được tạo ra, truy cập **frequency**  của antecedent và consequent.

Hàm tính toán độ tin cậy (confidence) và độ lift của luật kết hợp dựa trên tần suất xuất hiện của tiền đề (antecedent), kết quả (consequent) và tập phổ biến ban đầu. Nếu độ tin cậy của luật kết hợp lớn hoặc bằng ngưỡng tối thiểu **min_confidence**, luật đó sẽ được thêm vào danh sách **rules**. Cuối cùng, hàm trả về danh sách **rules** chưa tất cả các luật kết hợp đạt yêu cầu cùng với giá trị confidence và lift của từng luật.

# CHƯƠNG 5: KẾT QUẢ VÀ ĐÁNH GIÁ

## 5.1. Kết quả thực nghiệm

Sau nhiều lần thực nghiệm trên tập dữ liệu '**Market Basket Analysis 4.csv**', nhóm đã tiến hành thực nghiệm nhiều lần dựa trên Min Support và Min Confidence khác nhau. Các kết quả cho ra không quá khả quan trong khi Min Support và Min Confidence rất thấp.

### 5.1.2. Thử nghiệm với Min Support: 0.01 và Min Confidence: 0.1

Trong nhiều lần thực nghiệm, nhóm nhận thấy với Min Support = 0.01 là giá trị lớn nhất để có thể ra số lượng tập phổ biến lớn hơn 5.

| | Frequent Itemsets | Support |
|---|---|---|
| 0 | whole milk, yogurt | 0.011161 |
| 1 | whole milk, soda | 0.011629 |
| 2 | whole milk, rolls/buns | 0.013968 |
| 3 | whole milk, other vegetables | 0.014837 |
| 4 | rolls/buns, other vegetables | 0.010559 |

*Bảng 5.1: Bảng kết quả Frequent Itemsets*

Sau đó, nhóm tiếp tục tạo luật kết hợp dựa trên các tập phổ biến đó. Kết quả cho ra cũng không quá khả quan khi độ **Confidence** rất thấp dưới 0.2 và **Lift** dưới 1.

Điều này cho thấy các luật kết hợp cho ra từ Min Support = 0.01 và Min Confidence = 0.1 điều mang ý nghĩa tiêu cực.

| | Antecedents | Consequents | Support | Confidence | Lift |
|---|---|---|---|---|---|
| 0 | yogurt | whole milk | 0.011161 | 0.129961 | 0.822940 |
| 1 | soda | whole milk | 0.011629 | 0.119752 | 0.758296 |
| 2 | rolls/buns | whole milk | 0.013968 | 0.126974 | 0.804028 |

| | | | | | |
|---|---|---|---|---|---|
| 3 | other vegetables | whole milk | 0.014837 | 0.121511 | 0.769430 |

*Bảng 5.2: Bảng kết quả Association Rules*

### 5.1.2. Thử nghiệm với Min Support: 0.005 và Min Confidence: 0.1

Lần này nhóm thực hiện tìm các tập phổ biến với Min Support rất thấp, gần như không có ý nghĩa. Nhưng số lượng tập phổ biến lại cho kết quả khả quan hơn lên đến 1289 tập.

| | Frequent Itemsets | Support |
|---|---|---|
| 0 | pastry, salty snack | 0.000668 |
| 1 | pastry, whole milk | 0.006483 |
| 2 | UHT-milk, brown bread | 0.000535 |
| 3 | UHT-milk, citrus fruit | 0.000802 |
| 4 | yogurt, white wine | 0.000535 |
| … | … | … |
| 1289 | beef, berries | 0.000735 |

*Bảng 5.3: Bảng kết quả Frequent Itemsets*

Tiếp theo, nhóm tiến hành tạo luật kết hợp dựa trên các tập phổ biến đó. Kết quả cho ra giờ đây đã khả quan hơn khi có các tập có độ **Confidence** lớn hơn 0.3 và **Lift** lớn hơn 1. Tuy nhiên, như đã nói trước đó, các luật này được sinh ra trong điều kiện Min Support = 0.005, điều này khiến các luật kết hợp này không có quá nhiều ý nghĩa vì sự xuất hiện rất ít của nó.

| | Antecedents | Consequents | Support | Confidence | Lift |
|---|---|---|---|---|---|
| 0 | sausage, pork | whole milk | 0.000601 | 0.391304 | 2.477819 |
| 1 | brandy | whole milk | 0.000869 | 0.342105 | 2.166281 |
| 2 | softener | whole milk | 0.000802 | 0.292683 | 1.853328 |
| 3 | rolls/buns, white bread | whole milk | 0.000601 | 0.281250 | 1.780933 |

| | | | | | |
|---|---|---|---|---|---|
| 4 | sausage, shopping bags | other vegetables | 0.000535 | 0.000535 | 2.259291 |
| … | … | … | … | … | … |

*Bảng 5.4: Bảng kết quả Association Rules*

### 5.2. So sánh với các thuật toán tìm tập phổ biến khác

Nhóm tiến hành so sánh các thuật toán tìm tập phổ biến bằng cách so sánh kết quả hiệu suất của Apriori, FP-Growth và ECLAT về thời gian thực thi và bộ nhớ sử dụng. Trong đó 2 thuật toán Apriori và FP-Growth nhóm đã sử dụng thư viện **mlxtend**.

Trong quá trình chạy so sánh, nhóm cài đặt **Min Support** (độ hỗ trợ tối thiểu) giảm từ 0.01 đến 0.0005, giúp phân tích sự thay đổi trong hiệu suất khi thay đổi giá trị min support.

| **Min Support** | **Eclat** | **Apriori** | **FP-Growth** | **Số Itemset phổ biến** |
|---|---|---|---|---|
| **0.01** | 0.051 | 1.645 | 1.062 | 5 |
| **0.005** | 0.059 | 1.704 | 2.015 | 37 |
| **0.0025** | 0.066 | 1.818 | 5.117 | 143 |
| **0.0005** | 0.102 | 3.146 | 39.121 | 1289 |

*Bảng 5.5: Bảng đo chi phí thời gian (đơn vị đo: giây)*

**Eclat** có thời gian thực thi thấp hơn rất nhiều so với cả Apriori và FP-Growth ở tất cả các mức độ hỗ trợ. Điều này cho thấy Eclat là thuật toán hiệu quả nhất trong ba thuật toán về thời gian.

**Apriori** có thời gian thực thi tăng dần khi độ hỗ trợ giảm. Đây là đặc điểm chung của Apriori, vì nó phải quét qua tất cả các itemset trong từng vòng lặp.

**FP-Growth** có thời gian thực thi cao hơn Eclat, nhưng tốt hơn Apriori, trong trường hợp độ hỗ trợ cao tương ứng với việc tạo ra cây ít phức tạp hơn và phải duyệt cây ít hơn vì chỉ cần lấy ít itemset hơn. Tuy nhiên, khi độ hỗ trợ xuống thấp, FP-Growth lại có thời gian thực thi cao hơn rất nhiều so với hai thuật toán còn lại vì phải tạo cây phức tạp hơn và duyệt cây nhiều lần hơn để phục vụ cho việc lấy nhiều itemset hơn.

| ***Min Support*** | ***Eclat*** | ***Apriori*** | ***FP-Growth*** | **Số Itemset phổ biến** |
|---|---|---|---|---|
| ***0.01*** | *5.896* | *8.284* | *9.618* | *5* |
| ***0.005*** | *6.024* | *8.375* | *10.189* | *37* |
| ***0.0025*** | *6.218* | *8.459* | *10.681* | *143* |
| ***0.0005*** | *6.466* | *9.391* | *11.950* | *1289* |

*Bảng 5.6: Bảng đo chi phí bộ nhớ (đơn vị đo: MB)*

Khi Min Support giảm (từ 0.01 xuống 0.0005), chi phí bộ nhớ của cả ba thuật toán đều tăng. Điều này là do số lượng các tập phổ biến tăng lên đáng kể, dẫn đến yêu cầu lưu trữ lớn hơn.

**FP-Growth** luôn yêu cầu bộ nhớ cao nhất so với Eclat và Apriori ở mọi mức Min Support. Điều này có thể do cấu trúc cây FP-tree cần lưu trữ nhiều thông tin hơn.

**Eclat** sử dụng ít bộ nhớ nhất, cho thấy thuật toán này có ưu thế trong việc tối ưu hóa bộ nhớ bằng cách lưu trữ dữ liệu theo dạng dọc.

# CHƯƠNG 6: KẾT LUẬN VÀ ĐỊNH HƯỚNG

## 6.1. Tóm tắt kết quả đạt được

Sau khi tiến hành thử nghiệm tập dữ liệu '**Market Basket Analysis 4.csv**', nhóm rút ra được các kết luận như sau:

Tập dữ liệu này không phải là một tập dữ liệu lý tưởng để khai thác luật kết hợp. Dữ liệu này có tính phân tán cao, thiếu các mối quan hệ ý nghĩa giữa các mục, dẫn đến các kết quả không khả quan dù điều chỉnh các tham số Min Support và Min Confidence.

Dựa trên kết quả so sánh ba thuật toán tìm tập phổ biến, **Eclat** nổi bật như thuật toán hiệu quả nhất về thời gian thực thi. Eclat có thời gian thực thi thấp hơn đáng kể so với cả **Apriori** và **FP-Growth** ở mọi mức độ **Min Support**. Điều này nhờ vào việc sử dụng chiến lược duyệt theo chiều sâu và lưu trữ dữ liệu theo dạng dọc (TID-lists), giúp giảm thiểu số lượng phép toán và bộ nhớ cần thiết. **Apriori** có thời gian thực thi chậm nhất, đặc biệt khi **Min Support** giảm, vì thuật toán phải quét qua tất cả các itemset trong từng vòng lặp, dẫn đến tăng độ phức tạp và thời gian xử lý. **FP-Growth**, mặc dù có thời gian thực thi tốt hơn **Apriori**, lại tỏ ra kém hiệu quả khi **Min Support** giảm xuống rất thấp, vì thuật toán cần tạo và duyệt qua cây FP-tree phức tạp hơn khi số lượng tập phổ biến tăng lên.

Về bộ nhớ, **FP-Growth** yêu cầu bộ nhớ cao nhất, chủ yếu vì phải lưu trữ cây FP-tree và các thông tin liên quan đến các itemset. **Apriori** sử dụng bộ nhớ ở mức trung bình do cần duy trì các bảng đếm trong mỗi vòng lặp. Trong khi đó, **Eclat** có ưu thế rõ rệt trong việc tối ưu hóa bộ nhớ, nhờ vào việc lưu trữ dữ liệu theo chiều dọc, giúp giảm thiểu lượng bộ nhớ sử dụng đáng kể.

## 6.2. Định hướng nghiên cứu trong tương lai

Để tăng hiệu suất xử lý của thuật toán ECLAT, ta có thể biểu diễn các Tidset dưới dạng bit vector để thực hiện các phép giao bằng toán tử AND một cách nhanh hơn. Ngoài ra việc tối ưu hóa dữ liệu đầu vào cũng đóng vai trò rất quan trọng như lọc

dữ liệu bằng cách loại bỏ các item xuất hiện dưới ngưỡng **Min Support** trước khi bắt đầu thực thi thuật toán. Để giảm kích thước của Tidset, ta có thể áp dụng kỹ thuật nén dữ liệu như Run - Length Encoding. Ngoài ra, chuyển việc lưu trữ danh sách transaction ID thành Diffsets cũng là một cách tối ưu hóa bộ nhớ, giảm thiểu kích thước khi dữ liệu giao dịch có sự trùng lặp lớn

Đối với lĩnh vực thương mại điện tử, luật kết hợp có thể áp dụng để đề xuất các sản phẩm liên quan mà khách hàng thường mua cùng như bánh mì thường được mua chung với sữa. Việc xác định những sản phẩm mua cùng nhau giúp việc dự đoán nhu cầu sản phẩm theo thời gian được chính xác hơn.

Ngoài lĩnh vực thương mại điện tử thì việc áp dụng luật kết hợp trong y tế cũng góp phần quan trọng trong việc tối ưu các phác đồ điều trị khi xác định được các thuốc thường được kê cùng nhau, việc chẩn đoán bệnh sẽ trở nên dễ dàng hơn khi sử dụng các tập phổ biến để tìm triệu chứng bệnh thường đi kèm với nhau.

# TÀI LIỆU THAM KHẢO

[1] Gillis, A. S., Stedman, C., & Hughes, A. (2024, February 13). data mining. Search Business Analytics.

[2] Chee, C., Jaafar, J., Aziz, I. A., Hasan, M. H., & Yeoh, W. (2018). Algorithms for frequent itemset mining: a literature review. Artificial Intelligence Review, 52(4), 2603–2621.

[3] LAZAAR, N., LIRMM- UM, & COCONUT Team. (n.d.). Frequent Itemset Mining.

[4] Nguyễn, T. B. (2012). Khai phá Luật kết hợp trong cơ sở dữ liệu đa phương tiện (By Trường Đại học Công nghệ & V. Đặng)

[5] Han, J., Kamber, M., & Pei, J. ( 2012). Data mining: Concepts and techniques (3rd ed.). Morgan Kaufmann.

[6] Maimon, O., & Rokach, L. (Eds.). (2005). Data mining and knowledge discovery handbook. Springer.

[7] GeeksforGeeks. (2024b, August 2). ML | ECLAT Algorithm. GeeksforGeeks.