TRƯỜNG ĐẠI HỌC SƯ PHẠM – ĐẠI HỌC ĐÀ NẴNG

KHOA TIN HỌC

ĐỒ ÁN CHUYÊN NGÀNH

ĐỀ TÀI: THUẬT TOÁN LUẬT KẾT HỢP

Sinh viên thực hiện:

Đinh Văn Lộc 21CNTT3

Người hướng dẫn: GV. Phạm Dương Thu Hằng

Đà Nẵng, 12/2024

LỜI CẢM ƠN

Đầu tiên, em xin chân thành cảm ơn các thầy các cô trong khoa Tin học đã trang bị những kiến thức cho em trong suốt quá trình học tập tại Trường Đại học Sư phạm – Đại học Đà Nẵng vừa qua. Chính nhờ công lao giảng dạy, chỉ bảo tận tình của các thầy các cô mà em mới có được những kiến thức chuyên ngành công nghệ thông tin để có thể thực hiện tiếp chặng đường học tập, vận dụng và sáng tạo ra những sản phẩm hữu ích góp phần phục vụ các lĩnh vực khác nhau.

Thứ hai, chúng em xin chân thành cảm ơn TS. Phạm Dương Thu Hằng người đã tận tình hướng dẫn, chỉ bảo, góp ý, hỗ trợ cho chúng em trong suốt quá trình thực hiện đề tài này. Đó là những góp ý hết sức quý báu không chỉ trong quá trình thực hiện đồ án này mà còn là hành trang tiếp bước cho em trong quá trình học tập và lập nghiệp sau này.

Mặc dù đã có nhiều cố gắng để hoàn thành đồ án nhưng trong phạm vi và khả năng cho phép, chắc chắn đồ án không tránh khỏi những thiếu sót. Em rất mong nhận được sự thông cảm, góp ý và tận tình chỉ bảo của quý thầy cô.

Chúng em xin chân thành cảm ơn!

Đà Nẵng, tháng 12 năm 2024

Tác giả

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Đà Nẵng, ngày ... tháng ... năm ...

Giảng viên hướng dẫn

NHẬN XÉT CỦA HỘI ĐỒNG PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Đà Nẵng, ngày ... tháng ... năm ...

Hội đồng phản biện

MỤC LỤC

DANH MỤC CÁC ẢNH

MỞ ĐẦU

1. Lý do chọn đề tài

Khai phá dữ liệu đã trở thành một lĩnh vực quan trọng trong phân tích dữ liệu, đặc biệt là trong việc khám phá các mẫu và mối quan hệ tiềm ẩn trong tập dữ liệu lớn. Thuật toán luật kết hợp (Association Rule Learning) là quy trình nhằm tìm ra những mẫu tương quan và quy tắc kết hợp giữa các mục trong tập dữ liệu. Trong số các thuật toán luật kết hợp, Apriori, FP-growth, và ECLAT là ba thuật toán phổ biến với những ưu điểm và nhược điểm khác nhau. Việc nghiên cứu và so sánh các thuật toán này sẽ cung cấp cái nhìn sâu sắc về cách thức hoạt động của chúng và khả năng áp dụng trong các tình huống khác nhau. Đề tài này không chỉ giúp hiểu rõ hơn về các thuật toán này mà còn tạo điều kiện để áp dụng chúng trong các bài toán thực tế, từ đó cải thiện khả năng phân tích và ra quyết định dựa trên dữ liệu.

2. Mục tiêu đề tài

Mục tiêu của đề tài là xây dựng và triển khai các thuật toán luật kết hợp trên tập dữ liệu thực tế, cụ thể là các thuật toán Apriori, FP-growth, và ECLAT. Đề tài sẽ tập trung vào việc giới thiệu và giải thích cơ chế hoạt động của ba thuật toán luật kết hợp. Thực hiện các thuật toán trên một tập dữ liệu mẫu và so sánh hiệu quả của chúng.

3. Đối tượng và phạm vi nghiên cứu

Đối tượng: các thuật toán luật kết hợp Apriori, FP-growth, và ECLAT; dữ liệu mẫu dùng để kiểm tra và đánh giá.

Phạm vi nghiên cứu của đồ án này tập trung vào việc cài đặt ba thuật toán trên dữ liệu mẫu trong ngôn ngữ lập trình Python. Việc sử dụng các thư viện hỗ trợ hạn chế, tập trung vào việc thực hiện thuật toán từ đầu để hiểu rõ cơ chế hoạt động của chúng.

4. Nhiệm vụ thực hiện của đề tài

Nghiên cứu lý thuyết về thuật toán luật kết hợp và các thuật toán Apriori, FPgrowth, ECLAT.

Chuẩn bị dữ liệu: Chọn và chuẩn bị tập dữ liệu cho các thuật toán phân loại, đảm bảo dữ liệu được xử lý và làm sạch phù hợp.

Cài đặt từng thuật toán trong ngôn ngữ lập trình Python, không sử dụng thư viện hỗ trợ sẵn có.

Ghi chép và báo cáo kết quả nghiên cứu, bao gồm mô tả quá trình thực hiện, kết quả đánh giá, và phân tích các điểm mạnh và điểm yếu của từng thuật toán.

5. Phương pháp thực hiện

Nghiên cứu tài liệu để thu thập kiến thức lý thuyết về các thuật toán.

Xác định và thu thập dữ liệu bao gồm thu thập dữ liệu và chọn lọc dữ liệu.

Tiền xử lí dữ liệu bao gồm làm sạch, chuyển đổi, chia dữ liệu.

Cài đặt các thuật toán.

Phân tích, đánh giá kết quả thông qua việc so sánh hiệu quả hoạt động của các thuật toán dựa trên thời gian thực thi và kết quả khai phá.

Ghi chép quá trình và viết báo cáo.

6. Kết quả của đề tài

Báo cáo của đề tài: báo cáo trình bày mục tiêu nghiên cứu, mô tả dữ liệu và phương pháp nghiên cứu sử dụng các thuật toán Apriori, FP-growth, ECLAT, đánh giá các thuật toán, đưa ra kết luận.

Sản phẩm: Cơ sở lý thuyết và các cơ sở thực hành, khả năng ứng dụng của lần lượt các thuật toán.

7. Bố cục của đề tài

Mở đầu

Chương 1. Cơ sở lý thuyết

1. Luật kết hợp

2. Thuật toán Apriori

3. Thuật toán ECLAT

4. Thuật toán FP-growth

Chương 2: Thiết kế và xây dựng

1. Quy trình khai phá dữ liệu

2. Xác định bài toán

3. Thu thập dữ liệu

4. Xử lí và làm sạch dữ liệu

5. Xây dựng mô hình thuật toán Apriori

6. Xây dựng mô hình thuật toán ECLAT

7. Xây dựng mô hình thuật toán FP-growth

Chương 3: Đánh giá thuật toán và kết quả

1. Đánh giá thuật toán

2. Kết quả

Kết luận

8. Kế hoạch thực hiện

Tuần	Nội dung thực hiện	Kết quả đạt được	Ghi chú
1.	Nghiên cứu lý thuyết về luật kết hợp và ba thuật toán	Thu thập kiến thức cơ sở lý thuyết cơ bản về các thuật toán	
2.	Chuẩn bị dữ liệu	Xác định được bài toán, dữ liệu thu thập	
3.	Tiền xử lý dữ liệu	Dữ liệu được chọn lọc, làm sạch và mã hóa	
4.	Nghiên cứu và cài đặt thuật toán Apriori	Hoàn thành cài đặt thuật toán Apriori	
5.	Nghiên cứu và cài đặt thuật toán ECLAT	Hoàn thành cài đặt thuật toán ECLAT	

6.	Hoàn thành cài đặt thuật toán FP-growth	Hoàn thành cài đặt thuật toán FP-growth	
7.	Thực hiện các thử nghiệm với dữ liệu thực tế	Phân tích kết quả thực nghiệm, so sánh giữa các thuật toán	
8.	Viết báo cáo	Các phần nội dung của báo cáo	
9.	Hoàn thiện báo cáo	Báo cáo hoàn chỉnh và sản phẩm cài đặt hoàn thiện	

CHƯƠNG 1: CƠ SỞ LÍ THUYẾT

1. Luật kết hợp

1.1. Khái niệm

Quy tắc kết hợp Quy tắc kết hợp là một biểu thức hàm ý có dạng $X \rightarrow Y$, trong đó X và Y là các tập mục rời rạc, tức là $X \cap Y = \emptyset$. Độ mạnh của quy tắc kết hợp có thể được đo lường theo độ hỗ trợ và độ tin cậy của nó. Độ hỗ trợ xác định tần suất một quy tắc được áp dụng cho một tập dữ liệu nhất định, trong khi độ tin cậy xác định tần suất các mục trong Y xuất hiện trong các giao dịch có chứa X.

1.2. Ưu điểm

Hữu ích cho việc khám phá các mối quan hệ thú vị ẩn trong các tập dữ liệu lớn. Các mối quan hệ chưa được khám phá có thể được biểu diễn dưới dạng các tập hợp các mục có trong nhiều giao dịch, được gọi là các tập mục phổ biến.

1.3. Nhược điểm

Một số nhược điểm chính của thuật toán quy tắc liên kết trong e-learning là: các thuật toán được sử dụng có quá nhiều tham số đối với người không chuyên về khai thác dữ liệu và các quy tắc thu được quá nhiều, hầu hết đều không thú vị và có khả năng hiểu thấp.

2. Thuật toán Apriori

Apriori là thuật toán đầu tiên để khai thác các mẫu phổ biến, được đưa ra bởi R. Agarwal và R. Srikant vào năm 1994. Thuật toán này hoạt động trên cơ sở dữ liệu theo kiểu layout ngang. Nó dựa trên các quy tắc kết hợp Boolean và sử dụng phương pháp tạo và kiểm tra.

3. Thuật toán ECLAT

Eclat là một thuật toán sử dụng cấu trúc cơ sở dữ liệu dạng dọc (vertical database layout) để khai thác các tập mục phổ biến. Nó dựa trên thuật toán tìm kiếm theo chiều sâu (depth-first search). Trong bước đầu tiên, dữ liệu được biểu diễn dưới dạng ma trận bit.

4. Thuật toán FP-growth

Frequent Pattern Growth, hay còn gọi là FP-Growth, là thuật toán dựa trên cây để khai thác các mẫu phổ biến trong cơ sở dữ liệu. Ý tưởng này được đề xuất bởi (Han et al., 2000). Thuật toán này áp dụng cho cơ sở dữ liệu kiểu dự báo (projected database) và sử dụng phương pháp chia để trị (divide and conquer).

5. Python

5.1 Định nghĩa

Python là một ngôn ngữ lập trình được sử dụng rộng rãi trong các ứng dụng web, phát triển phần mềm, khoa học dữ liệu và máy học (ML). Các nhà phát triển sử dụng Python vì nó hiệu quả, dễ học và có thể chạy trên nhiều nền tảng khác nhau**.**

5.2 Ưu điểm khi sử dụng Python trong khai phá dữ liệu

Thư viện phong phú cho phép thực hiện các công việc khai phá dữ liệu từ việc xử lý, phân tích đến xây dựng mô hình dễ dàng và hiệu quả.

Python được biết đến với cú pháp đơn giản và dễ đọc, giúp người mới học và người có kinh nghiệm dễ dàng tiếp cận. Điều này giúp giảm thời gian học và triển khai các dự án khai phá dữ liệu.

CHƯƠNG 2: THIẾT KẾ VÀ XÂY DỰNG

1. Quy trình khai phá dữ liệu

1.1. Xác định bài toán

Xác định mục tiêu cụ thể muốn đạt được thông qua việc khai phá dữ liệu. Điều này bao gồm việc hiểu rõ vấn đề cần giải quyết, câu hỏi cần trả lời hoặc mục tiêu kinh doanh cụ thể.

1.2. Thu thập dữ liệu

Thu thập dữ liệu từ các nguồn khác nhau như cơ sở dữ liệu, tệp tin, web, sensor, và các nguồn dữ liệu khác phù hợp với mục tiêu.

1.3. Xử lí và làm sạch dữ liệu

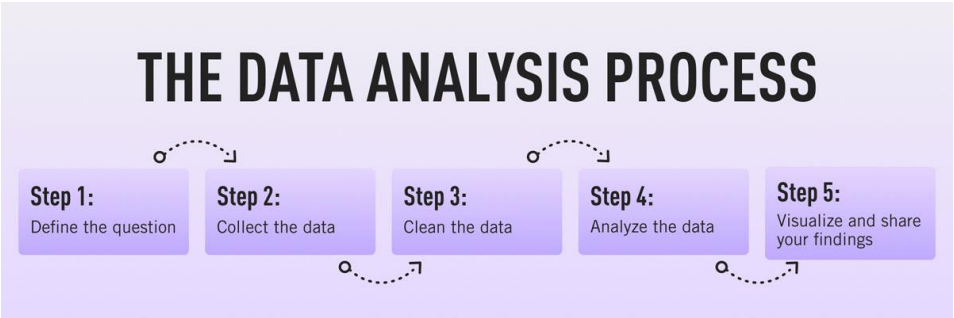
Bước này bao gồm xử lý và làm sạch dữ liệu để chuẩn hóa, loại bỏ giá trị thiếu, loại bỏ nhiễu, điều chỉnh định dạng và chuẩn hóa dữ liệu để chuẩn bị cho quá trình khai phá.

1.4. Phân tích dữ liệu

Áp dụng các phương pháp và thuật toán khai phá dữ liệu để tìm ra mẫu, quy luật, thông tin tiềm ẩn và mối quan hệ trong dữ liệu. Các phương pháp có thể bao gồm phân tích thống kê, học máy, kỹ thuật khai phá dữ liệu, và các phương pháp khác.

1.5. Đánh giá và lựa chọn mô hình

Đánh giá kết quả từ việc phân tích dữ liệu, lựa chọn mô hình tốt nhất dựa trên mục tiêu của bạn



Hình 1. Quy trình phân tích dữ liệu

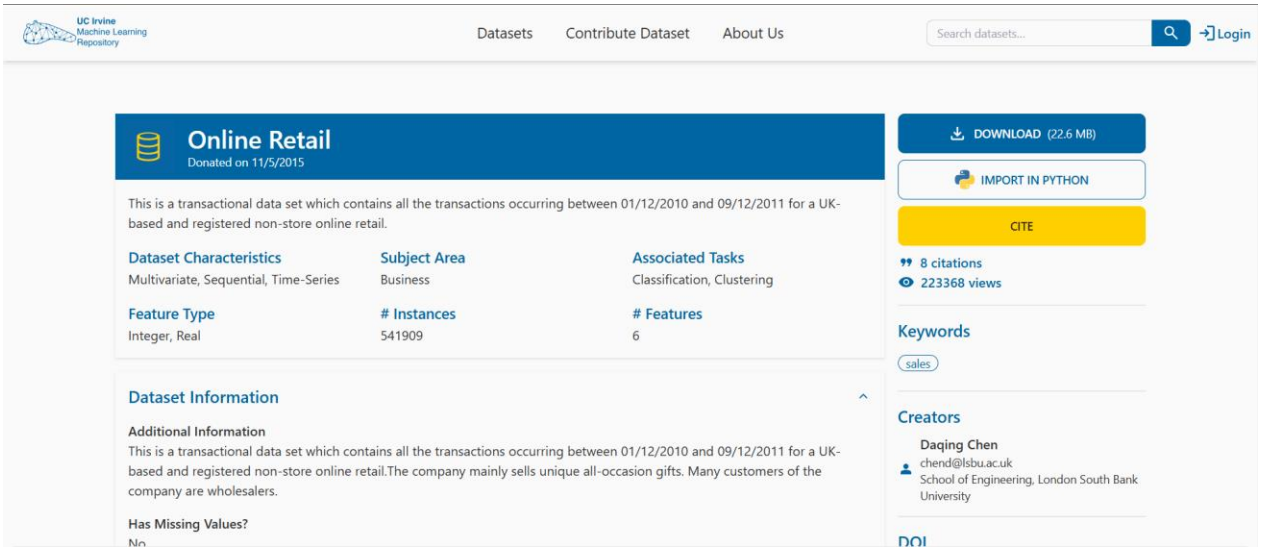
2. Xác định bài toán

Dữ liệu bản lẻ trực tuyến là một tập dữ liệu giao dịch chứa tất cả các giao dịch diễn ra trong khoảng thời gian từ 01/12/2010 đến 09/12/2011 cho một công ty bán lẻ trực tuyến cửa hàng có trụ sở tại Vương quốc Anh và bản lẻ không qua cửa hàng đã được đăng ký. Công ty chủ yếu bán những món quà độc đáo cho mọi dịp. Nhiều khách hàng của công ty là những người buôn bán.

3. Thu thập dữ liệu

3.1. Nguồn dữ liệu

Dữ liệu được thu thập từ trang web https://archive.ics.uci.edu/dataset/352/online+retail, nơi cung cấp dữ liệu về các giao dịch.



Hình 2. Website UC Irvine Machine Learning Repository

3.2. Phạm vi và loại dữ liệu thu thập

3.2.1. Phạm vi

Dữ liệu được thu thập có 541909 mục và 8 thuộc tính chứa thông tin về các giao dịch bán lẻ .

3.2.2. Loại dữ liệu

Invoice: Số hóa đơn. Số hiệu. Một số nguyên gồm 6 chữ số được gán duy nhất cho mỗi giao dịch. Nếu mã này bắt đầu bằng chữ cái 'c', nó biểu thị một lệnh hủy.

StockCode: Mã sản phẩm (mặt hàng). Mã danh nghĩa. Một số nguyên gồm 5 chữ số được gán duy nhất cho mỗi sản phẩm riêng biệt.

Description: Tên sản phẩm (mặt hàng). Tên danh nghĩa.

Quantity: Số lượng của từng sản phẩm (mặt hàng) cho mỗi giao dịch. Dạng số.

InvoiceDate: Ngày và giờ của hóa đơn. Số. Ngày và giờ khi giao dịch được tạo.

Price: Giá cho mỗi đơn vị sản phẩm.

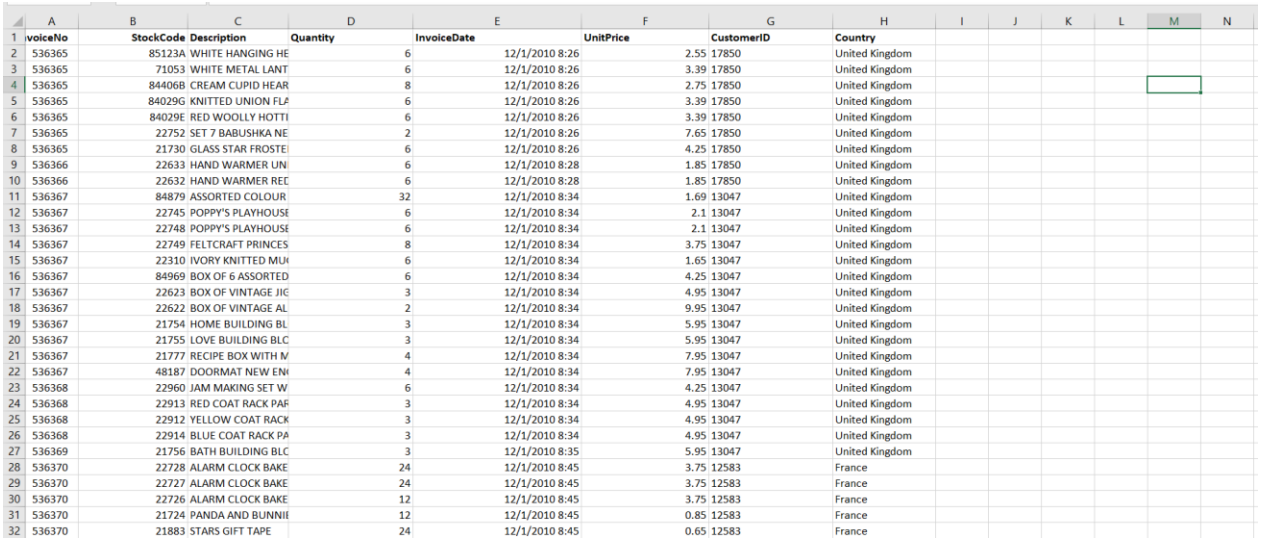
Customer ID: số nguyên gồm 5 chữ số được gán duy nhất cho mỗi khách hàng.

Country: tên quốc gia nơi mỗi khách hàng cư trú

4. Xử lí và làm sạch dữ liệu

4.1. Dữ liệu tiền xử lí

Trước khi xử lý dữ liệu, các thông tin ta cần chú ý là số hóa đơn và tên sản phẩm. Dữ liệu có cùng số hóa đơn bị tách thành các dòng riêng biệt, vì vậy ta cần nhóm lại thành một danh sách.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	voiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country						
2	536365	85123A	WHITE HANGING HE	6	12/1/2010 8:26	2.55	17850	United Kingdom						
3	536365	71053	WHITE METAL LANT	6	12/1/2010 8:26	3.39	17850	United Kingdom						
4	536365	84406B	CREAM CUPID HEAR	8	12/1/2010 8:26	2.75	17850	United Kingdom						
5	536365	84029G	KNITTED UNION FLA	6	12/1/2010 8:26	3.39	17850	United Kingdom						
6	536365	84029E	RED WOOLLY HOTTI	6	12/1/2010 8:26	3.39	17850	United Kingdom						
7	536365	22752	SET 7 BABUSHKA NE	2	12/1/2010 8:26	7.65	17850	United Kingdom						
8	536365	21730	GLASS STAR FROSTE	6	12/1/2010 8:26	4.25	17850	United Kingdom						
9	536366	22633	HAND WARMER UN	6	12/1/2010 8:28	1.85	17850	United Kingdom						
10	536366	22632	HAND WARMER RED	6	12/1/2010 8:28	1.85	17850	United Kingdom						
11	536367	84879	ASSORTED COLOUR	32	12/1/2010 8:34	1.69	13047	United Kingdom						
12	536367	22745	POPPY'S PLAYHOUSE	6	12/1/2010 8:34	2.1	13047	United Kingdom						
13	536367	22748	POPPY'S PLAYHOUSE	6	12/1/2010 8:34	2.1	13047	United Kingdom						
14	536367	22749	FELTCRAFT PRINCES	8	12/1/2010 8:34	3.75	13047	United Kingdom						
15	536367	22310	IVORY KNITTED MU	6	12/1/2010 8:34	1.65	13047	United Kingdom						
16	536367	84969	BOX OF 6 ASSORTED	6	12/1/2010 8:34	4.25	13047	United Kingdom						
17	536367	22623	BOX OF VINTAGE JIG	3	12/1/2010 8:34	4.95	13047	United Kingdom						
18	536367	22622	BOX OF VINTAGE AL	2	12/1/2010 8:34	9.95	13047	United Kingdom						
19	536367	21754	HOME BUILDING BL	3	12/1/2010 8:34	5.95	13047	United Kingdom						
20	536367	21755	LOVE BUILDING BLO	3	12/1/2010 8:34	5.95	13047	United Kingdom						
21	536367	21777	RECIPE BOX WITH M	4	12/1/2010 8:34	7.95	13047	United Kingdom						
22	536367	48187	DOORMAT NEW EN	4	12/1/2010 8:34	7.95	13047	United Kingdom						
23	536368	22960	JAM MAKING SET W	6	12/1/2010 8:34	4.25	13047	United Kingdom						
24	536368	22913	RED COAT RACK PAR	3	12/1/2010 8:34	4.95	13047	United Kingdom						
25	536368	22912	YELLOW COAT RACK	3	12/1/2010 8:34	4.95	13047	United Kingdom						
26	536368	22914	BLUE COAT RACK PA	3	12/1/2010 8:34	4.95	13047	United Kingdom						
27	536369	21756	BATH BUILDING BLO	3	12/1/2010 8:35	5.95	13047	United Kingdom						
28	536370	22728	ALARM CLOCK BAKE	24	12/1/2010 8:45	3.75	12583	France						
29	536370	22727	ALARM CLOCK BAKE	24	12/1/2010 8:45	3.75	12583	France						
30	536370	22726	ALARM CLOCK BAKE	12	12/1/2010 8:45	3.75	12583	France						
31	536370	21724	PANDA AND BUNNIE	12	12/1/2010 8:45	0.85	12583	France						
32	536370	21883	STARS GIFT TAPE	24	12/1/2010 8:45	0.65	12583	France						

Hình 3. Kiểm tra dữ liệu tiền xử lí

4.2. Làm sạch dữ liệu

4.2.1. Chọn thông tin liên quan và có ý nghĩa

Sử dụng kỹ thuật list slicing trên tập dữ liệu để tạo một dataframe chỉ bao gồm 2 cột InvoiceNo và Description. Việc này giúp ta dễ dàng nhóm các sản phẩm có cùng số hóa đơn lại với nhau.

```
data = df[['InvoiceNo', 'Description']]
```

```
data.head()
```

	InvoiceNo	Description
0	536365	WHITE HANGING HEART T-LIGHT HOLDER
1	536365	WHITE METAL LANTERN
2	536365	CREAM CUPID HEARTS COAT HANGER
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE
4	536365	RED WOOLLY HOTTIE WHITE HEART.

Hình 4. Tập dữ liệu chỉ có 2 cột InvoiceNo và Description

4.2.2. Loại bỏ các hóa đơn bị hủy và các giá trị N/A

Sử dụng các hàm như isna().any() để kiểm tra xem có giá trị N/A nào trong hai cột hay không và isna().sum() để đếm tổng số giá trị N/A. Sau đó, dùng hàm dropna(axis=0) để loại bỏ các dòng chứa giá trị N/A.

```
[105]: data.isna().any()

[105]: InvoiceNo      False
       Description     True
       dtype: bool

[107]: data.isna().sum()

[107]: InvoiceNo         0
       Description    1455
       dtype: int64

[109]: data = data.dropna(axis=0)
```

Hình 5. Loại bỏ các dòng

Loại bỏ các dấu phẩy xuất hiện trong phần tên sản phẩm để tránh nhầm lẫn khi đọc file. Nếu tên sản phẩm chứa dấu phẩy, chương trình đọc file có thể hiểu nhầm đó là ranh giới giữa các cột dữ liệu, dẫn đến sai lệch trong quá trình xử lý.

536380	22961	JAM MAKING SET PRINTED
536381	22139	RETROSPOT TEA SET CERAMIC 11 PC
536381	84854	GIRLY PINK TOOL SET
536381	22411	JUMBO SHOPPER VINTAGE RED PAISLEY
536381	82567	AIRLINE LOUNGE,METAL SIGN
536381	21672	WHITE SPOT RED CERAMIC DRAWER KNOB
536381	22774	RED DRAWER KNOB ACRYLIC EDWARDIAN
536381	22771	CLEAR DRAWER KNOB ACRYLIC EDWARDIAN
536381	71270	PHOTO CLIP LINE

Hình 6. Có một số tên sản phẩm chứa dấu phẩy

4.2.3. Định dạng và tổ chức dữ liệu

Để chuẩn bị dữ liệu cho các bước tiếp theo, ta thực hiện nhóm các sản phẩm dựa trên số hóa đơn. Đầu tiên, ta lọc các giá trị trong cột InvoiceNo để đảm bảo chỉ giữ lại các hóa đơn hợp lệ. Sau đó, dữ liệu được nhóm lại theo InvoiceNo, và các sản phẩm trong từng nhóm được đưa vào danh sách dưới dạng đã sắp xếp.

```
[117]: data = data[pd.to_numeric(data['InvoiceNo'], errors='coerce').notnull()]
       data['InvoiceNo'] = data['InvoiceNo'].astype('int32')

[119]: groupinvoice = data.groupby('InvoiceNo')

[121]: transactions = []
       for name,group in groupinvoice:
           transactions.append(sorted(list(group['Description'].map(str))))
       transactions

[121]: [['CREAM CUPID HEARTS COAT HANGER',
         'GLASS STAR FROSTED T-LIGHT HOLDER',
         'KNITTED UNION FLAG HOT WATER BOTTLE',
         'RED WOOLLY HOTTIE WHITE HEART.',
         'SET 7 BABUSHKA NESTING BOXES',
         'WHITE HANGING HEART T-LIGHT HOLDER',
         'WHITE METAL LANTERN'],
        ['HAND WARMER RED POLKA DOT', 'HAND WARMER UNION JACK'],
        ['ASSORTED COLOUR BIRD ORNAMENT',
         'BOX OF 6 ASSORTED COLOUR TEASPOONS',
         'BOX OF VINTAGE ALPHABET BLOCKS',
         'BOX OF VINTAGE JIGSAW BLOCKS ',
         'DOORMAT NEW ENGLAND',
         'FELTCRAFT PRINCESS CHARLOTTE DOLL',
         'HOME BUILDING BLOCK WORD',
         'IVORY KNITTED MUG COSY ',
         'LOVE BUILDING BLOCK WORD',
         "POPPY'S PLAYHOUSE BEDROOM "
```

Hình 7. Định dạng dữ liệu

Sau khi nhóm các sản phẩm theo số hóa đơn, ta lưu dữ liệu vào file để chuẩn bị cho bước áp dụng thuật toán. Mỗi nhóm sản phẩm được ghi vào file dưới dạng một dòng, với các sản phẩm được phân tách bằng dấu phẩy.

```
with open('online_retail.txt', 'w') as f:
    for sublist in transactions:
        f.write(','.join(sublist) + '\n')
```

Hình 8. Lưu trữ dữ liệu đã định dạng

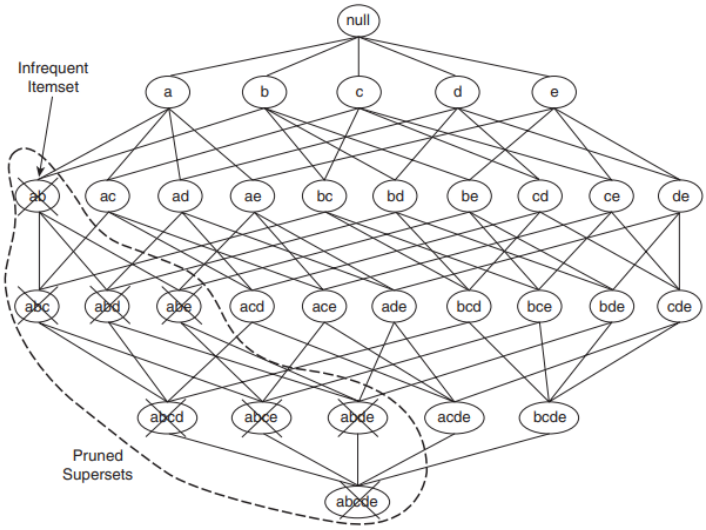
Để đảm bảo các sản phẩm được sắp xếp theo thứ tự từ điển (lexicographical order), ta tạo một danh sách chứa tất cả các tên sản phẩm duy nhất. Danh sách này được lưu vào file dưới dạng từng dòng tương ứng với mỗi sản phẩm.

Hình 9. Tạo danh sách thứ tự sắp xếp theo từ điển

5. Xây dựng mô hình thuật toán Apriori

5.1. Ý tưởng thuật toán

Phần này mô tả cách thức biện pháp tham số hỗ trợ có thể được sử dụng để giảm số lượng các tập ứng viên được khám phá trong quá trình tạo tập phổ biến. Việc sử dụng hỗ trợ để cắt tỉa các tập ứng viên được áp dụng bởi nguyên tắc sau: nếu một tập là phổ biến thì tất cả các tập con của nó cũng phải phổ biến.



Hình 10. Một minh họa về cắt tỉa dựa trên tham số hỗ trợ. Nếu {a,b} không phổ biến, thì tất cả các tập chứa {a,b} cũng không phổ biến

5.2. Nguyên lý hoạt động của giải thuật Apriori

Mã giả cho phần tạo tập mục phổ biến của thuật toán Apriori được trình bày trong phần Thuật toán 5.3. Giả sử C_k biểu thị tập hợp các tập ứng cử viên k-itemsets và F_k biểu thị tập hợp các ứng viên k-itemsets phổ biến:

- Ban đầu, thuật toán thực hiện một lần duyệt qua tập dữ liệu để xác định mức hỗ trợ của từng mục. Sau khi hoàn tất bước này, tập hợp tất cả các tập mục phổ biến có k = 1, ta sẽ có được F_1 (bước 1 và 2).
- Tiếp theo, thuật toán sẽ lặp lại việc tạo ra các tập mục k ứng cử viên mới và cắt tỉa các ứng cử viên không cần thiết được đảm bảo là không phổ biến dựa trên các tập mục phổ biến (k −1) được tìm thấy trong lần lặp trước (bước 5 và 6). Việc tạo và cắt tỉa ứng viên được triển khai bằng cách sử dụng các hàm candidate-gen và candidate-prune.
- Để đếm số lượng hỗ trợ của các ứng cử viên, thuật toán cần thực hiện một lần duyệt bổ sung trên tập dữ liệu (các bước 7–12). Hàm subset được sử dụng để xác định tất cả các tập mục ứng cử viên trong C_k có trong mỗi giao dịch t.

- Sau khi đếm số lượng hỗ trợ của chúng, thuật toán loại bỏ tất cả các tập ứng cử viên có số lượng hỗ trợ nhỏ hơn N × minsup (bước 13).
- Thuật toán kết thúc khi không có tập phổ biến mới nào được tạo ra, tức là $F_k = \emptyset$ (bước 14).

Phần sinh các tập phổ biến của thuật toán Apriori có hai đặc điểm quan trọng. Đầu tiên, đây là thuật toán theo từng cấp; tức là nó duyệt qua mạng tập mục theo từng cấp, từ tập phổ biến có kích thước bằng 1 đến kích thước tối đa của tập phổ biến. Thứ hai, nó sử dụng chiến lược tạo và kiểm tra để tìm các tập mục phổ biến. Ở mỗi lần lặp (mức), các tập ứng viên mới được tạo ra từ các tập mục phổ biến được tìm thấy trong lần lặp trước đó. Sau đó, độ hỗ trợ cho mỗi mục ứng cử viên được đếm và kiểm tra theo ngưỡng minsup. Tổng số lần lặp cần thiết của thuật toán là k_{max}+ 1, trong đó k_{max} là kích thước tối đa của các tập mục phổ biến.

5.3. Mã giả giải thuật

Algorithm 5.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: C_k = candidate-gen(F_{k-1}). {Generate candidate itemsets.}
6: C_k = candidate-prune(C_k, F_{k-1}). {Prune candidate itemsets.}
7: **for** each transaction $t \in T$ **do**
8: C_t = subset(C_k, t). {Identify all candidates that belong to t.}
9: **for** each candidate itemset $c \in C_t$ **do**
10: $\sigma(c) = \sigma(c) + 1$. {Increment support count.}
11: **end for**
12: **end for**
13: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets.}
14: **until** $F_k = \emptyset$
15: Result = $\bigcup F_k$.

Hình 11. Mã giả sinh tập phổ biến của thuật toán Apriori

5.4. Triển khai mã nguồn thuật toán

Mã dưới định nghĩa đường dẫn tới dữ liệu và các tham số quan trọng cho thuật toán Apriori. path_to_data lưu đường dẫn tới file dữ liệu giao dịch (online_retail.txt),

trong khi min_support và min_confidence xác định ngưỡng hỗ trợ tối thiểu (5%) và độ tin cậy tối thiểu (60%) cho các quy tắc kết hợp. path_to_order chứa đường dẫn tới file order.txt, danh sách các sản phẩm được sắp xếp theo thứ tự từ điển, hỗ trợ trong việc tạo ra các tập hợp mục phổ biến khi áp dụng thuật toán.

```
[3]: path_to_data = "C:/Users/Admin/DA_Projects/online_retail.txt"
     min_support = 0.05
     min_confidence = 0.6
     path_to_order = "C:/Users/Admin/DA_Projects/order.txt"
```

Hình 12. Cài đặt tham số và đường dẫn dữ liệu

Khởi tạo các cấu trúc dữ liệu cần thiết cho thuật toán Apriori. C chứa các tập con ứng viên với kích thước tương ứng, trong khi L sẽ lưu trữ các itemset phổ biến (chưa được cập nhật). itemset_size được khởi tạo là 1, chỉ các itemset kích thước 1, và discarded lưu các itemset bị loại bỏ. Cuối cùng, C được cập nhật với các itemset kích thước 1 từ danh sách order đã được sắp xếp.

Initialization

```
[41]: C = {}
      L = {}
      itemset_size = 1
      discarded = {itemset_size: []}
      C.update({itemset_size: [[f] for f in order]})

[43]: C

[43]: {1: [['3 TRADITIONAL COOKIE CUTTERS  SET'],
           ['3 TRADITIONAl BISCUIT CUTTERS  SET'],
           ['ALARM CLOCK BAKELIKE CHOCOLATE'],
           ['ALARM CLOCK BAKELIKE GREEN'],
```

Hình 13. Khởi tạo tập phổ biến

Tạo ra các itemset phổ biến L1 bằng cách sử dụng hàm get_frequent, nhập vào các candidate 1-itemsets từ C[itemset_size], giao dịch mẫu, và ngưỡng hỗ trợ tối thiểu (min_support). Hàm trả về các itemsets phổ biến (f), số lần xuất hiện (sup), và các itemsets bị loại bỏ (new_discarded). Các giá trị này được cập nhật vào các từ

điển: discarded, L, và supp_count_L để chuẩn bị cho các bước tiếp theo của thuật toán Apriori.

Create L1

```
supp_count_L = {}
f, sup, new_discarded = get_frequent(C[itemset_size], sampled_transactions, min_support, discarded)
discarded.update({itemset_size : new_discarded})
L.update({itemset_size : f})
supp_count_L.update({itemset_size : sup})
```

```
print_table(L[1], supp_count_L[1])
```

```
Itemset | Frequency
['ASSORTED COLOUR BIRD ORNAMENT']  :  13
['CHOCOLATE HOT WATER BOTTLE']  :  11
['CREAM CUPID HEARTS COAT HANGER']  :  15
['GLASS STAR FROSTED T-LIGHT HOLDER']  :  15
['HAND WARMER BIRD DESIGN']  :  20
['HAND WARMER OWL DESIGN']  :  17
['HAND WARMER RED POLKA DOT']  :  15
['HAND WARMER SCOTTY DOG DESIGN']  :  20
['HAND WARMER UNION JACK']  :  29
```

Hình 14. Tạo L1 và cập nhập từng tập con ứng cử viên

Hàm get_frequent dùng để tìm các itemsets phổ biến dựa trên ngưỡng hỗ trợ tối thiểu (min_support). Đầu vào của hàm là các itemsets ứng cử viên, giao dịch (transactions), ngưỡng hỗ trợ tối thiểu và các itemsets đã bị loại bỏ trong các vòng lặp trước đó.

Hàm lặp qua từng itemset trong danh sách itemsets. Nếu itemset đã bị loại bỏ ở vòng trước (kiểm tra qua prev_discarded), nó sẽ bị bỏ qua. Nếu không, hàm đếm số lần xuất hiện của itemset trong các giao dịch bằng hàm count_occurences. Nếu tần suất xuất hiện của itemset vượt qua ngưỡng hỗ trợ, itemset đó sẽ được thêm vào danh sách L (itemsets phổ biến) và supp_count (số lần xuất hiện của chúng). Nếu không, itemset đó sẽ được thêm vào danh sách new_discarded để loại bỏ sau này.

Cuối cùng, hàm trả về ba giá trị: L (itemsets phổ biến), supp_count (số lần xuất hiện), và new_discarded (itemsets bị loại bỏ).

```python
def get_frequent(itemsets, transactions, min_support, prev_discarded):
    L = []
    supp_count = []
    new_discarded = []

    k = len(prev_discarded.keys())

    for s in range(len(itemsets)):
        discarded_before = False
        if k > 0:
            for it in prev_discarded[k]:
                if set(it).issubset(set(itemsets[s])):
                    discarded_before = True
                    break

        if not discarded_before:
            count = count_occurences(itemsets[s], transactions)
            if count/len(transactions) >= min_support:
                L.append(itemsets[s])
                supp_count.append(count)
            else:
                new_discarded.append(itemsets[s])

    return L, supp_count, new_discarded
```

Hình 15. Hàm get_frequent()

Hàm count_occurences đếm số lần một itemset xuất hiện trong các giao dịch. Nó lặp qua mỗi giao dịch và kiểm tra xem itemset có phải là tập con của giao dịch đó không. Nếu có, hàm tăng biến đếm và cuối cùng trả về tổng số lần xuất hiện của itemset.

```python
def count_occurences(itemset, transactions):
    count = 0
    for i in range(len(transactions)):
        if set(itemset).issubset(set(transactions[i])):
            count += 1
    return count
```

Hình 16. Hàm count_occurences()

Vòng lặp này tạo và cập nhật các tập phổ biến từ L1 đến Lk cho đến khi không còn itemsets phổ biến. Ở mỗi vòng, các itemsets ứng cử viên được tạo ra từ các itemsets phổ biến của vòng trước, sau đó được kiểm tra và lọc dựa trên ngưỡng hỗ trợ. Nếu có itemsets phổ biến, chúng sẽ được lưu vào L và số lần xuất hiện vào supp_count_L. Vòng lặp dừng khi không còn itemsets phổ biến nào ở cấp độ k.

```
k = itemset_size + 1
convergence = False
while not convergence:
    C.update({ k : join_set_itemsets(L[k-1], order)})
    f, sup, new_discarded = get_frequent(C[k], sampled_transactions, min_support, discarded)
    discarded.update({k : new_discarded})
    L.update({k : f})
    supp_count_L.update({k : sup})
    if len(L[k]) == 0:
        convergence = True
    else:
        print("Table L{}: \n".format(k))
        print_table(L[k], supp_count_L[k])
    k += 1
```

Hình 17. Vòng lặp tạo Lk và cập nhật các itemsets

Hai hàm dưới thực hiện việc kết hợp các itemsets trong quá trình tìm kiếm tập phổ biến. Hàm join_two_itemsets kết hợp hai itemsets lại với nhau, kiểm tra xem các itemset có thể hợp nhất được không bằng cách so sánh các phần tử của chúng. Nếu hợp nhất được, nó sẽ trả về một itemset mới, nếu không sẽ trả về một danh sách rỗng. Hàm join_set_itemsets sử dụng hàm trên để kết hợp tất cả các itemsets trong một tập hợp các itemsets (set_of_items), tạo ra các ứng cử viên mới. Quá trình kết hợp được thực hiện theo thứ tự trong order để đảm bảo tính hợp lý của kết quả.

```
def join_two_itemsets(it1, it2, order):
    it1.sort(key=lambda x: order.index(x))
    it1.sort(key=lambda x: order.index(x))

    for i in range(len(it1) - 1):
        if it1[i] != it2[i]:
            return []

    return it1 + [it2[-1]]


def join_set_itemsets(set_of_items, order):
    C = []
    for i in range(len(set_of_items)):
        for j in range(i + 1, len(set_of_items)):
            it_out = join_two_itemsets(set_of_items[i], set_of_items[j], order)
            if len(it_out) > 0:
                C.append(it_out)
    return C
```

Hình 18. Hàm kết hợp itemsets

Hàm combinations tạo ra tất cả các tổ hợp của một iterable với độ dài r. Nó sử dụng chỉ số để sinh ra từng tổ hợp một cách hiệu quả. Hàm powerset tạo ra tất cả các tập con của một tập hợp s, ngoại trừ tập con rỗng và tập hợp đầy đủ. Nó sử dụng hàm combinations để tạo các tập con có độ dài từ 1 đến len(s) - 1, rồi trả về danh sách tất cả các tổ hợp này.

```python
def combinations(iterable, r):
    # combinations('ABCD', 2) --> AB AC AD BC BD CD
    # combinations(range(4), 3) --> 012 013 023 123
    pool = tuple(iterable)
    n = len(pool)
    if r > n:
        return
    indices = list(range(r))
    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != i + n - r: #123 124 125 [1,2,3,4,5]
                break
        else:
            return
        indices[i] += 1
        for j in range(i+1, r):
            indices[j] = indices[j-1] + 1
        yield tuple(pool[i] for i in indices)

def powerset(s):
    rules = []
    for r in range(1, len(s)):
       rule = combinations(s,r)
       rules += rule

    return list(rules)
```

Hình 19. Hàm kết hợp và tạo powerset

Tạo các quy tắc kết hợp từ các tập phổ biến đã được tìm thấy. Đầu tiên, vòng lặp qua các tập phổ biến từ L1 đến Lk. Với mỗi itemset, hàm powerset được sử dụng để tạo tất cả các tập con của itemset đó. Sau đó, các quy tắc kết hợp được tạo ra từ các tập con này, với các giá trị hỗ trợ, độ tin cậy (confidence) và độ lift được tính toán. Quy tắc kết hợp chỉ được giữ lại nếu độ tin cậy đạt ngưỡng tối thiểu (min_confidence). Các quy tắc này được thêm vào chuỗi assoc_rules_str.

```python
assoc_rules_str = ""
num_trans = len(filtered_transactions)
for i in range(1, len(L)):
    for j in range(len(L[i])):
        s = powerset(L[i][j])
        for z in s:
            S = set(z)
            X = set(L[i][j])
            X_S = set(X-S)
            sup_x = supp_count_L[i][j]
            sup_x_s = count_occurences(X_S, sampled_transactions)
            conf = sup_x / count_occurences(S, sampled_transactions)
            lift = conf / (sup_x_s / num_trans)
            if conf >= min_confidence:
                assoc_rules_str += write_rules(X, X_S, S, conf, sup_x, lift, num_trans)
```

Hình 20. Mã tạo các quy luật kết hợp

6. Xây dựng mô hình thuật toán ECLAT

6.1. Ý tưởng thuật toán

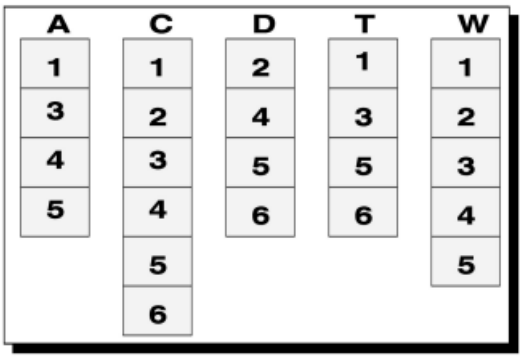
Việc triển khai thuật toán Eclat biểu diễn tập các giao dịch dưới dạng ma trận bit và thực hiện giao các hàng để xác định độ hỗ trợ của các tập mục. Thuật toán này sử dụng phương pháp duyệt sâu (depth-first traversal) của một cây tiền tố (prefix tree).

6.2. Nguyên lý hoạt động của giải thuật ECLAT

Một cách thuận tiện để biểu diễn các giao dịch trong thuật toán Eclat là sử dụng ma trận bit, trong đó mỗi hàng tương ứng với một mục (item) và mỗi cột tương ứng với một giao dịch (hoặc ngược lại). Một bit trong ma trận sẽ được đặt (set) nếu mục tương ứng với hàng xuất hiện trong giao dịch tương ứng với cột, ngược lại bit sẽ bị xóa (cleared).

Về cơ bản, có hai cách để biểu diễn một ma trận bit như vậy:

- Sử dụng một ma trận bit thực sự, trong đó mỗi bit bộ nhớ đại diện cho một mục và một giao dịch.
- Sử dụng một danh sách các cột cho mỗi hàng, trong đó chỉ lưu trữ các cột mà bit được đặt. (Cách biểu diễn này tương đương với việc sử dụng một danh sách các mã giao dịch cho mỗi mục.)



Hình 21. TID - list

Triển khai phương pháp tìm bottom-up: các tập phổ biến (frequent itemsets) được tạo ra bằng cách giao nhau giữa các danh sách tid (tid-lists) của tất cả các cặp phần tử (atoms) khác biệt và kiểm tra số lượng phần tử (cardinality) trong danh sách tid kết quả. Sau đó gọi đệ quy được thực hiện với những tập mục (itemsets) được phát

hiện là phổ biến ở cấp độ hiện tại. Quá trình này được lặp lại cho đến khi tất cả các tập phổ biến được liệt kê đầy đủ.

6.3. Mã giả giải thuật

Bottom-Up(S):
for all atoms $A_i \in S$ **do**
  $T_i = \emptyset$;
  for all atoms $A_j \in S$, with $j > i$ **do**
    $R = A_i \cup A_j$;
    $\mathcal{L}(R) = \mathcal{L}(A_i) \cap \mathcal{L}(A_j)$;
    if $\sigma(R) \geq min\_sup$ **then**
      $T_i = T_i \cup \{R\}$; $\mathcal{F}_{|R|} = \mathcal{F}_{|R|} \cup \{R\}$;
  end
end
for all $T_i \neq \emptyset$ **do** Bottom-Up(T_i);

Hình 22. Mã giả cho tìm bottom-up

6.4. Triển khai mã nguồn thuật toán

Đoạn mã trên chuyển đổi tập dữ liệu ban đầu thành dạng biểu diễn sử dụng danh sách mã giao dịch (tid-list) cho mỗi mục, một cách tương đương với ma trận bit trong thuật toán ECLAT, giúp xác định các giao dịch chứa mục đó và lọc các mục phổ biến dựa trên ngưỡng hỗ trợ tối thiểu (minsup).

```
def preprocess_dataset(dataset, minsup):
    C1 = defaultdict(set)
    for tid, transaction in enumerate(dataset):
        for item in transaction:
            C1[frozenset([item])].add(tid)
    return sorted(
        [(itemset, tidset) for itemset, tidset in C1.items() if len(tidset) >= minsup],
        key=lambda x: len(x[1])
    )
```

Hình 23. Tiền xử lý dữ liệu cho thuật toán ECLAT: Biểu diễn danh sách TID

Đoạn code này triển khai thuật toán ECLAT để tìm các tập mục phổ biến trong dữ liệu giao dịch. Mục đích chính của code là duyệt qua các tập mục trong danh sách C, kiểm tra xem chúng có đạt mức hỗ trợ tối thiểu (minsup) không, sau đó in ra các tập mục phổ biến cùng với hỗ trợ của chúng. Tiếp theo, code thực hiện tìm kiếm các cặp mục kết hợp (gọi là R) từ các tập mục hiện tại, tính toán giao nhau của chúng và kiểm tra điều kiện hỗ trợ tối thiểu. Nếu cặp mục này là phổ biến, chúng sẽ được thêm vào danh sách T và thuật toán tiếp tục đệ quy với danh sách mới.

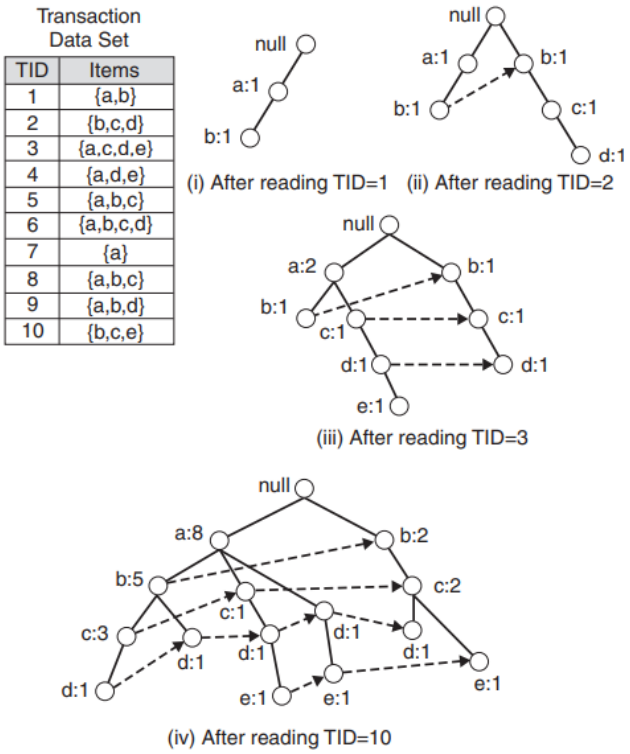
```python
def eclat(C, minsup):
    for i, (X, tX) in enumerate(C):
        if len(tX) >= minsup:
            print(f"Frequent itemset: {set(X)}, support: {len(tX)}")
        T = []
        for j in range(i + 1, len(C)):
            Y, tY = C[j]
            R = X | Y
            tR = tX & tY
            if len(tR) >= minsup:
                T.append((R, tR))
        if T:
            eclat(T, minsup)
```

Hình 24. Tìm các tập mục phổ biến theo phương pháp bottom-up

7. Xây dựng mô hình thuật toán FP-Growth

7.1. Ý tưởng thuật toán

Phần này trình bày một thuật toán thay thế gọi là FP-growth có cách tiếp cận hoàn toàn khác để khám phá các tập mục phổ biến. Thuật toán này không tuân theo mô hình tạo và kiểm tra của Apriori. Thay vào đó, nó mã hóa tập dữ liệu bằng một cấu trúc dữ liệu nhỏ gọn gọi là FP-tree và trích xuất các tập mục phổ biến trực tiếp từ cấu trúc này. Chi tiết về cách tiếp cận này được trình bày tiếp theo.



Hình 25. Xây dựng FP-tree

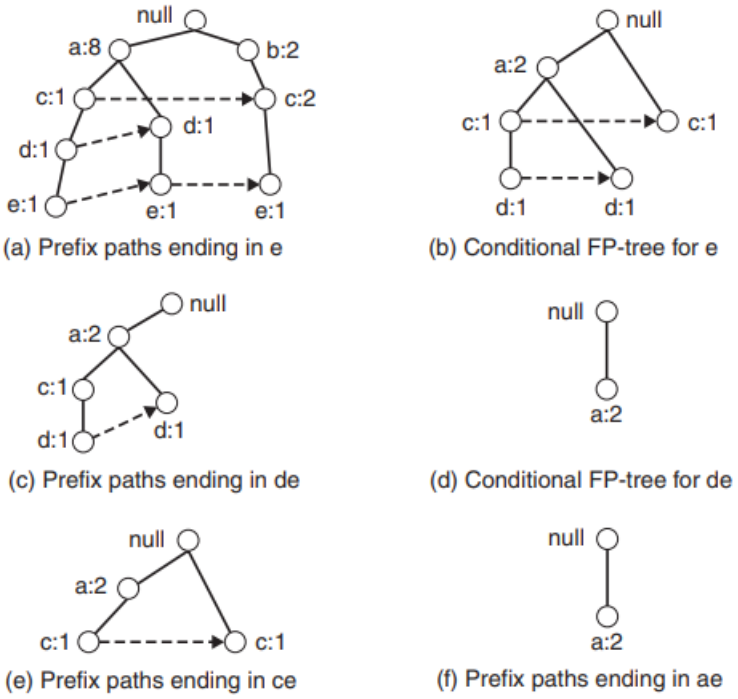
7.2. Nguyên lý hoạt động của giải thuật FP-Growth

FP-growth là một thuật toán tạo ra các tập phổ biến từ một cây FP (FP-tree) bằng cách duyệt qua cây theo hướng từ dưới lên trên. Với ví dụ cây được hiển thị trong Hình 20, thuật toán tìm kiếm các tập phổ biến kết thúc bằng e trước, sau đó đến d, c, b và cuối cùng là a. FP-growth tìm tất cả các tập phổ biến kết thúc với một hậu tố cụ thể bằng cách sử dụng chiến lược chia để trị, phân chia vấn đề thành các bài toán con nhỏ hơn.

Để có ví dụ cụ thể hơn về cách giải quyết các bài toán con, hãy xem xét nhiệm vụ tìm các tập phổ biến kết thúc bằng e.

- Bước đầu tiên là thu thập tất cả các đường đi chứa nút e. Những đường đi ban đầu này được gọi là các đường đi tiền tố và được minh họa trong Hình 21(a).
- Từ các đường đi tiền tố trong Hình 21(a), số lần xuất hiện của e được tính bằng cách cộng các số lần xuất hiện liên quan đến nút e. Giả sử rằng số lần

xuất hiện tối thiểu là 2, {e} được coi là một tập phổ biến vì số lần xuất hiện của nó là 3.



Hình 26. Ví dụ về việc áp dụng thuật toán FP-growth để tìm các tập phổ biến kết thúc bằng e

- Vì {e} là một tập phổ biến, thuật toán phải giải quyết các bài toán con để tìm các tập phổ biến kết thúc bằng de, ce, be và ae. Trước khi giải quyết các bài toán con này, thuật toán phải chuyển các đường đi tiền tố thành một cây FP điều kiện, cây này có cấu trúc tương tự như cây FP, nhưng được sử dụng để tìm các tập phổ biến kết thúc bằng một hậu tố cụ thể. Cây FP điều kiện được tạo ra theo cách sau:
    - Đầu tiên, các số đếm hỗ trợ dọc theo các đường đi tiền tố phải được cập nhật vì một số số đếm bao gồm các giao dịch không chứa món hàng e. Ví dụ, đường đi bên phải trên cùng trong Hình 21(a), null → b:2 → c:2 → e:1, bao gồm một giao dịch {b, c} không chứa món hàng e. Vì vậy, các số đếm dọc theo đường đi tiền tố phải được điều chỉnh thành 1 để phản ánh số lượng giao dịch thực tế chứa {b, c, e}.
    - Đường dẫn tiền tố được rút gọn bằng cách loại bỏ các nút cho e. Các nút này có thể được loại bỏ vì các số lượng hỗ trợ dọc theo các đường

dẫn tiền tố đã được cập nhật để chỉ phản ánh các giao dịch chứa e, và các bài toán con tìm tập phổ biến kết thúc bằng de, ce, be và ae không còn cần thông tin về nút e nữa.

- Sau khi cập nhật số lượng hỗ trợ dọc theo các đường dẫn tiền tố, một số mặt hàng có thể không còn phổ biến. Ví dụ, nút b chỉ xuất hiện một lần và có số lượng hỗ trợ bằng 1, điều này có nghĩa là chỉ có một giao dịch chứa cả b và e. Mặt hàng b có thể được bỏ qua trong các phân tích tiếp theo vì tất cả các tập hợp kết thúc bằng be chắc chắn không phổ biến.

- FP-growth sử dụng cây FP điều kiện cho e để giải quyết các bài toán con tìm các tập phổ biến kết thúc bằng de, ce và ae. Để tìm các tập phổ biến kết thúc bằng de, các đường đi tiền tố cho d được thu thập từ cây FP điều kiện cho e (Hình 21(c)). Bằng cách cộng dồn các tần suất liên quan đến nút d, ta có được độ hỗ trợ của {d, e}. Vì độ hỗ trợ bằng 2, {d, e} được công nhận là một tập phổ biến. Tiếp theo, thuật toán xây dựng cây FP điều kiện cho de bằng cách sử dụng phương pháp đã được mô tả trong bước 3. Sau khi cập nhật độ hỗ trợ và loại bỏ các mục infrequent như c, cây FP điều kiện cho de được hiển thị trong Hình 21(d). Vì cây FP điều kiện chỉ chứa một mục duy nhất là a, với độ hỗ trợ bằng minsup, thuật toán trích xuất tập phổ biến {a, d, e} và tiếp tục giải quyết bài toán con tiếp theo, đó là tạo ra các tập phổ biến kết thúc bằng ce. Sau khi xử lý các đường đi tiền tố cho c, {c, e} được tìm thấy là phổ biến. Tuy nhiên, cây FP điều kiện cho ce sẽ không có mục phổ biến nào và do đó sẽ bị loại bỏ. Thuật toán tiếp tục giải quyết bài toán con tiếp theo và tìm thấy {a, e} là tập phổ biến duy nhất còn lại.

7.3. Mã nguồn thuật toán FP-Growth

Lớp TreeNode biểu diễn các nút trong cây FP, lưu trữ một mục, số lượng hỗ trợ của mục đó, tham chiếu đến các nút cha và nút con của mục đó. Thuộc tính liên kết kết nối các nút với cùng một mục để tối ưu hóa việc duyệt. Cấu trúc này giúp FP-growth khai thác hiệu quả các tập mục phổ biến.

```python
# Tree Node Class (no core library used)
class TreeNode:
    def __init__(self, item, count, parent):
        self.item = item       # Item name
        self.count = count     # Support count
        self.parent = parent   # Parent node
        self.children = {}     # Child nodes
        self.link = None       # Next node in header table
```

Hình 27. Hàm TreeNode()

Hàm count_item_frequencies lặp lại qua từng giao dịch và các mục của nó, đếm số lần mỗi mục xuất hiện trong tất cả các giao dịch. Nó lưu trữ các tần suất này trong một từ điển (item_counts), trong đó khóa là các mục và giá trị là số lượng tương ứng của chúng. Điều này giúp xác định các mục phổ biến cho thuật toán FP-growth.

```python
# Function to manually count item frequencies
def count_item_frequencies(transactions):
    item_counts = {}
    for transaction in transactions:
        for item in transaction:
            if item in item_counts:
                item_counts[item] += 1
            else:
                item_counts[item] = 1
    return item_counts
```

Hình 28. Hàm count_item_frequencies()

Hàm build_fp_tree tạo một cây FP từ các giao dịch đã cho bằng các bước sau:

Đếm tần suất mục: Đầu tiên, hàm này đếm tần suất của từng mục trong tất cả các giao dịch bằng hàm count_item_frequencies. Sau đó, hàm này xóa các mục có tần suất thấp hơn ngưỡng min_support.

Xây dựng cây và bảng tiêu đề: Gốc của cây FP được tạo thành TreeNode không có mục nào. Một header_table trống cũng được khởi tạo để lưu trữ nút.

Chèn giao dịch: Đối với mỗi giao dịch, các mục được lọc chỉ còn lại bao gồm những mục đáp ứng ngưỡng hỗ trợ, sau đó được sắp xếp theo tần suất của chúng. Các mục đã sắp xếp được chèn vào cây. Đối với mỗi mục trong danh sách đã sắp xếp, một nút được tạo hoặc cập nhật trong cây. Nếu mục đã tồn tại trong cây, số lượng

của mục đó sẽ tăng lên; nếu không, một nút mới sẽ được thêm vào. header_table được cập nhật để duy trì liên kết đến lần xuất hiện đầu tiên của từng mục trong cây.

Hàm này trả về cây FP đã xây dựng và header_table để xử lý thêm trong thuật toán FP-growth.

```python
def build_fp_tree(transactions, min_support):
    # Step 1: Count item frequencies
    item_counts = count_item_frequencies(transactions)
    # Remove items below min_support
    for item in list(item_counts.keys()):
        if item_counts[item] < min_support:
            del item_counts[item]
    if len(item_counts) == 0:
        return None, None
    # Step 2: Build tree root and header table
    root = TreeNode(None, 0, None)
    header_table = {}
    # Step 3: Insert sorted transactions into the FP-tree
    for transaction in transactions:
        # Filter and sort items
        filtered_items = [item for item in transaction if item in item_counts]
        sort_items(filtered_items, item_counts)  # Sort based on frequency
        # Insert transaction into the tree
        current_node = root
        for item in filtered_items:
            if item in current_node.children:
                current_node.children[item].count += 1
            else:
                new_node = TreeNode(item, 1, current_node)
                current_node.children[item] = new_node
                # Update header table
                if item not in header_table:
                    header_table[item] = new_node
                else:
                    # Link new node to existing nodes in header table
                    temp = header_table[item]
                    while temp.link is not None:
                        temp = temp.link
                    temp.link = new_node
            current_node = current_node.children[item]
    return root, header_table
```

Hình 29. Hàm build_fp_tree()

Hàm find_conditional_patterns tìm cơ sở mẫu điều kiện cho một mục từ header_table. Nó lấy các đường đi từ các nút liên kết trong bảng tiêu đề, truy ngược từ mỗi nút đến gốc cây, xây dựng các mẫu điều kiện cho mục đó. Mỗi đường đi được lặp lại theo số lần xuất hiện của nút, sau đó trả về tất cả các mẫu điều kiện đã tìm

được. Hàm này hỗ trợ việc tạo các cây FP điều kiện để phát hiện các tập mục phổ biến trong thuật toán FP-growth.

```python
# Function to find conditional patterns for an item
def find_conditional_patterns(header_table, item):
    node = header_table[item]
    patterns = []
    while node:
        path = []
        parent = node.parent
        while parent and parent.item is not None:
            path.append(parent.item)
            parent = parent.parent
        if path:
            for _ in range(node.count):
                patterns.append(path[::-1])  # Reverse path
        node = node.link
    return patterns
```

Hình 30. Hàm find_conditional_patterns()

Hàm fp_growth thực hiện quá trình khai thác đệ quy trong thuật toán FP-growth. Đầu tiên, nó duyệt qua từng mục trong header_table để tạo các tập mục phổ biến mới, tính toán tổng số hỗ trợ cho mỗi mục. Sau đó, hàm tìm các mẫu điều kiện (conditional patterns) cho mỗi mục và nếu có, nó xây dựng cây FP điều kiện. Nếu cây FP điều kiện tồn tại, hàm gọi đệ quy để tiếp tục khai thác các tập mục phổ biến từ cây con này. Quá trình này tiếp tục cho đến khi không còn mẫu điều kiện nào, và các tập mục phổ biến đã tìm được được thêm vào danh sách kết quả.

```python
# đệ quy FP-Growth mining
def fp_growth(header_table, min_support, prefix, frequent_itemsets):
    for item in header_table:
        # Create new frequent itemset
        new_prefix = prefix + [item]
        total_support = 0
        node = header_table[item]
        while node:
            total_support += node.count
            node = node.link
        frequent_itemsets.append((new_prefix, total_support))

        # Find conditional patterns
        conditional_patterns = find_conditional_patterns(header_table, item)
        if not conditional_patterns:
            continue

        # Build conditional FP-tree
        conditional_root, conditional_header = build_fp_tree(conditional_patterns, min_support)
        if conditional_root:
            fp_growth(conditional_header, min_support, new_prefix, frequent_itemsets)
```

Hình 31. Hàm đệ quy fp_growth()

Hàm run_fp_growth là hàm chính để thực thi thuật toán FP-growth. Nó đầu tiên xây dựng cây FP từ các giao dịch đầu vào và loại bỏ các mục có hỗ trợ thấp hơn min_support. Nếu không có cây FP hợp lệ (nghĩa là không có mục nào phổ biến), hàm sẽ thông báo và trả về danh sách rỗng. Ngược lại, hàm sẽ khởi tạo danh sách các tập mục phổ biến và gọi hàm đệ quy fp_growth để tìm kiếm các tập mục phổ biến từ cây FP. Cuối cùng, nó trả về danh sách các tập mục phổ biến đã tìm được.

```python
# FP-Growth Entry Function
def run_fp_growth(transactions, min_support):
    root, header_table = build_fp_tree(transactions, min_support)
    if not root:
        print("No frequent itemsets found.")
        return []
    frequent_itemsets = []
    fp_growth(header_table, min_support, [], frequent_itemsets)
    return frequent_itemsets
```

Hình 32. Hàm run_fp_growth()

Mã này chạy thuật toán FP-Growth với mức hỗ trợ tối thiểu là 3 và in các tập mục phổ biến được tìm thấy trong các giao dịch. Nó gọi hàm run_fp_growth, xử lý các giao dịch đã lọc, sau đó in từng tập mục phổ biến và số lượng hỗ trợ của nó.

```python
min_support = 3
print("Running FP-Growth with min_support =", min_support)
frequent_itemsets = run_fp_growth(filtered_transactions, min_support)

print("\nFrequent Itemsets:")
for itemset, support in frequent_itemsets:
    print(f"{itemset}: {support}")
```

Hình 33. Cài đặt min_support và in tập phổ biến

CHƯƠNG 3: ĐÁNH GIÁ THUẬT TOÁN VÀ KẾT QUẢ

1. Đánh giá thuật toán

1.1. Thuật toán Apriori

Apriori có thể không hiệu quả lắm khi tập dữ liệu lớn vì nó tạo ra nhiều tập mục ứng viên ở mỗi lần lặp, đặc biệt là đối với các tập mục bậc cao hơn. Điều này có thể dẫn đến chi phí tính toán cao về cả bộ nhớ và thời gian.

Apriori tập trung vào việc lọc các tập mục dựa trên hỗ trợ và sử dụng độ tin cậy như một biện pháp bổ sung khi tạo các quy tắc liên kết. Thuật toán có thể bỏ qua các quy tắc có độ tin cậy thấp hơn nhưng vẫn có thể thú vị.

Điểm mạnh chính của thuật toán nằm ở tính đơn giản, nhưng như đã đề cập, nó có thể gặp phải vấn đề về khả năng mở rộng đối với các tập dữ liệu lớn do phương pháp tạo ứng viên của nó. Tuy nhiên, nó phù hợp với các tập dữ liệu nhỏ hơn hoặc các tình huống cần các quy tắc có độ tin cậy cao.

1.2. Thuật toán ECLAT

Eclat có hiệu suất rất tốt, đặc biệt là trong các bộ dữ liệu lớn. Nó chỉ cần một số lần quét cơ sở dữ liệu rất ít (thường chỉ 1 hoặc 2 lần) so với các thuật toán khác như Apriori, vốn phải quét cơ sở dữ liệu nhiều lần để sinh ra các tập mục phổ biến.

Tuy nhiên, trong các tập dữ liệu quá lớn hoặc có số lượng mục rất lớn, Eclat có thể gặp khó khăn về bộ nhớ vì nó sử dụng ma trận bit hoặc danh sách giao dịch, điều này có thể chiếm dụng một lượng bộ nhớ lớn.

1.3. Thuật toán FP-Growth

FP-growth thường hiệu quả hơn Apriori đối với các tập dữ liệu lớn vì nó tránh tạo ra các tập mục ứng viên. Thay vào đó, nó xây dựng một cây nhỏ gọn (FP-tree) và khai thác trực tiếp.

Ưu điểm chính của FP-growth là khả năng khai thác các tập mục phổ biến mà không cần tạo ra tất cả các ứng viên có thể, giúp giảm cả độ phức tạp về thời gian và dung lượng bộ nhớ khi so sánh với Apriori.

FP-growth hiệu quả và có khả năng mở rộng hơn Apriori, đặc biệt là đối với các tập dữ liệu lớn hơn, do sử dụng cây FP nhỏ gọn và khai thác trực tiếp mà không

tạo ra các tập mục ứng viên. Tuy nhiên, việc triển khai nó phức tạp hơn một chút so với Apriori.

2. Kết quả

2.1. Thuật toán Apriori

Nhiều quy tắc thể hiện giá trị độ tin cậy cao, đặc biệt là những quy tắc có độ tin cậy 1.000, chẳng hạn như:

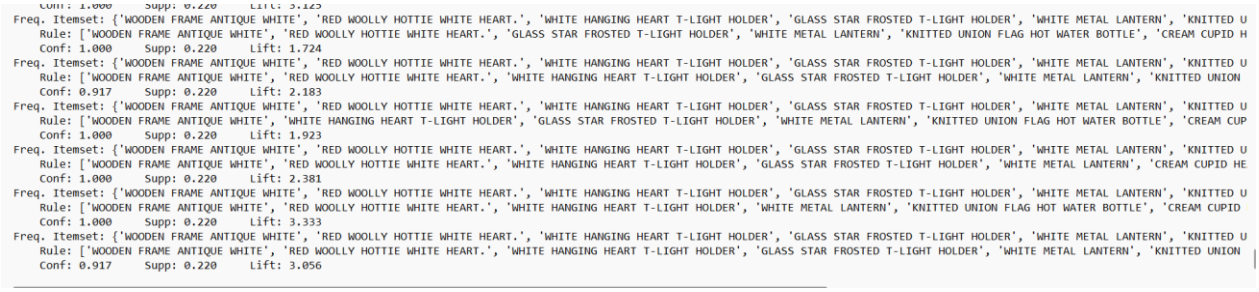
- 'WOODEN FRAME ANTIQUE WHITE', 'WHITE HANGING HEART T-LIGHT HOLDER', 'GLASS STAR FROSTED T-LIGHT HOLDER', 'WHITE METAL LANTERN', 'KNITTED UNION FLAG HOT WATER BOTTLE', 'CREAM CUPID HEARTS COAT HANGER', 'SET 7 BABUSHKA NESTING BOXES' → 'RED WOOLLY HOTTIE WHITE HEART.'
- 'WOODEN FRAME ANTIQUE WHITE', 'RED WOOLLY HOTTIE WHITE HEART.', 'WHITE HANGING HEART T-LIGHT HOLDER', 'GLASS STAR FROSTED T-LIGHT HOLDER', 'WHITE METAL LANTERN', 'CREAM CUPID HEARTS COAT HANGER', 'SET 7 BABUSHKA NESTING BOXES' → 'KNITTED UNION FLAG HOT WATER BOTTLE'
- 'WOODEN FRAME ANTIQUE WHITE', 'RED WOOLLY HOTTIE WHITE HEART.', 'WHITE HANGING HEART T-LIGHT HOLDER', 'GLASS STAR FROSTED T-LIGHT HOLDER', 'WHITE METAL LANTERN', 'KNITTED UNION FLAG HOT WATER BOTTLE', 'CREAM CUPID HEARTS COAT HANGER', 'SET 7 BABUSHKA NESTING BOXES' → 'GLASS STAR FROSTED T-LIGHT HOLDER'

Những quy tắc có độ tin cậy cao này cho thấy rằng nếu khách hàng mua những mặt hàng ở phía bên trái của quy tắc, rất có thể họ cũng sẽ mua những mặt hàng ở phía bên phải.

Phân tích Lift:

- Giá trị Lift cung cấp thông tin chi tiết về sức mạnh của mối liên kết giữa các mục. Ví dụ, trong quy tắc:

- ‘WOODEN FRAME ANTIQUE WHITE', 'RED WOOLLY HOTTIE WHITE HEART.', 'WHITE HANGING HEART T-LIGHT HOLDER', 'GLASS STAR FROSTED T-LIGHT HOLDER', 'WHITE METAL LANTERN', 'KNITTED UNION FLAG HOT WATER BOTTLE', 'CREAM CUPID HEARTS COAT HANGER' → 'SET 7 BABUSHKA NESTING BOXES', giá trị Lift là 2,183. Điều này cho thấy khả năng xuất hiện đồng thời của các mục này cao gấp đôi so với dự kiến nếu các mục này độc lập.
- Giá trị nâng cao (ví dụ: 3,333 đối với 'WOODEN FRAME ANTIQUE WHITE', 'RED WOOLLY HOTTIE WHITE HEART.', 'WHITE HANGING HEART T-LIGHT HOLDER', etc. → 'GLASS STAR FROSTED T-LIGHT HOLDER') cho thấy mối liên hệ chặt chẽ, không ngẫu nhiên giữa các mục này.



Hình 34. Các luật kết hợp được sinh bởi thuật toán Apriori

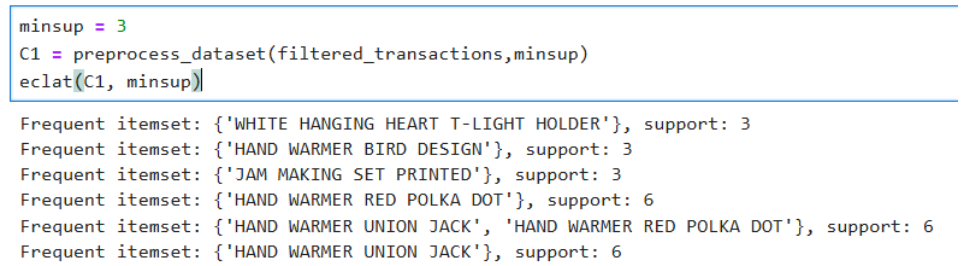
2.2. Thuật toán ECLAT

Các mặt hàng thường mua:

- Các tập mục thường gặp nhất bao gồm nhiều sản phẩm khác nhau như "HAND WARMER UNION JACK", "HAND WARMER RED POLKA DOT" và "HAND WARMER BIRD DESIGN". Các mục này xuất hiện trong nhiều giao dịch với mức hỗ trợ tương đối cao (6 giao dịch cho hai tập mục "HAND WARMER").
- Điều này cho thấy các sản phẩm như máy sưởi tay rất phổ biến trong tập dữ liệu, có khả năng chỉ ra nhu cầu cao đối với các mặt hàng theo mùa hoặc mới lạ.

Liên kết giữa các mặt hàng:

- Bộ sản phẩm {'HAND WARMER UNION JACK', 'HAND WARMER RED POLKA DOT'} cho thấy hai sản phẩm này thường được mua cùng nhau. Điều này có thể ngụ ý sở thích về các mặt hàng theo chủ đề hoặc phù hợp giữa các khách hàng. Những liên tưởng này có thể giúp tạo ra các chương trình khuyến mãi có mục tiêu hoặc các chiến lược bán chéo, chẳng hạn như cung cấp chiết khấu khi mua cả hai mặt hàng.



Hình 35. Các tập phổ biến được sinh ra bởi thuật toán ECLAT

2.3. Thuật toán FP-Growth

Các mặt hàng thường mua:

- 'HAND WARMER UNION JACK' (7 lần xuất hiện) và 'HAND WARMER RED POLKA DOT' (6 lần xuất hiện) là những mặt hàng được mua nhiều nhất, cho thấy khách hàng rất ưa chuộng những mặt hàng này.
- 'JUMBO BAG RED RETROSPOT' xuất hiện 4 lần, cho thấy đây cũng là một mặt hàng phổ biến trong tập dữ liệu.

Kết hợp sản phẩm:

- Sự xuất hiện đồng thời thường xuyên của 'HAND WARMER UNION JACK' và 'HAND WARMER RED POLKA DOT' (6 lần xuất hiện) cho thấy rằng khách hàng mua một trong những mặt hàng này có khả năng cũng sẽ mua mặt hàng kia. Điều này có thể hữu ích để tạo ra các ưu đãi khuyến mại, chẳng hạn như đóng gói hai sản phẩm này lại với nhau.

Cơ hội bán chéo:

- Các sản phẩm như 'JUMBO BAG RED RETROSPOT' và 'HAND WARMER UNION JACK' có thể được bán chéo cùng nhau, đặc biệt nếu khách hàng mua một trong những mặt hàng này có khả năng sẽ mua mặt hàng còn lại.

```
Running FP-Growth with min_support = 3

Frequent Itemsets:
['WHITE HANGING HEART T-LIGHT HOLDER']: 3
['HAND WARMER RED POLKA DOT']: 6
['HAND WARMER UNION JACK']: 7
['HAND WARMER UNION JACK', 'HAND WARMER RED POLKA DOT']: 6
['GREY HEART HOT WATER BOTTLE']: 3
['JAM MAKING SET PRINTED']: 3
['JAM MAKING SET WITH JARS']: 3
['JUMBO BAG RED RETROSPOT']: 4
```

Hình 36. Các tập phổ biến được sinh bởi thuật toán FP-Growth

KẾT LUẬN

Dựa trên kết quả từ các thuật toán Apriori, FP-Growth và ECLAT, chúng ta có thể rút ra một số kết luận về hiệu quả của từng phương pháp trong việc khai thác các mẫu phổ biến. Thuật toán Apriori, mặc dù hiệu quả trong việc tìm kiếm các mẫu kết hợp, nhưng gặp phải hạn chế lớn khi phải quét cơ sở dữ liệu nhiều lần và tạo ra số lượng lớn các tập con ứng viên, dẫn đến chi phí tính toán cao và khả năng mở rộng kém khi làm việc với cơ sở dữ liệu lớn. Trong khi đó, FP-Growth vượt trội về hiệu suất và khả năng mở rộng nhờ sử dụng cây mẫu phổ biến điều kiện, giúp giảm số lần quét cơ sở dữ liệu và loại bỏ bước tạo tập con ứng viên, làm tăng tốc quá trình tìm kiếm và giảm sự phức tạp tính toán.

ECLAT cũng cho thấy sự hiệu quả trong việc khai thác các mẫu phổ biến nhờ việc sử dụng ma trận bit và phép giao nhau đơn giản, giúp giảm số lần quét cơ sở dữ liệu. Thuật toán này có thể xử lý dữ liệu lớn tốt hơn Apriori nhờ vào cách lưu trữ tối ưu hơn và không cần đến việc tạo tập con ứng viên như Apriori. ECLAT hoạt động hiệu quả khi xử lý các tập dữ liệu thưa, với khả năng tìm kiếm các mối quan hệ kết hợp mạnh mẽ và dễ dàng mở rộng.

Tóm lại, mặc dù Apriori có thể phát hiện các mẫu phổ biến và quy tắc kết hợp đáng chú ý, nhưng hiệu suất của nó bị ảnh hưởng khi làm việc với dữ liệu lớn. FP-Growth và ECLAT tỏ ra vượt trội hơn trong việc xử lý dữ liệu lớn, với ECLAT đặc biệt hiệu quả trong việc khai thác các mẫu phổ biến mà không gặp phải vấn đề chi phí tính toán của Apriori. Do đó, FP-Growth và ECLAT là sự lựa chọn tối ưu hơn khi làm việc với các cơ sở dữ liệu quy mô lớn.

TÀI LIỆU THAM KHẢO

[1] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Anuj Karpatne "**introduction to Data Mining**", https://www-users.cse.umn.edu/~kumar/dmbook/index.php, 2019

[2] Kanwal Garg "**Comparing the Performance of Frequent Pattern Mining Algorithms**", https://www.researchgate.net/publication/272864559_Comparing_the_Performance_of_Frequent_Pattern_Mining_Algorithms, 2013

[3] Mohammed J. Zaki, "**Scalable Algorithms for Association Mining**", https://www.philippe-fournier-viger.com/spmf/zaki2000.pdf, 2000