***Logo, icon

Description automatically generatedShape, rectangle

Description automatically generated***

*Hồ Chí Minh, 13 tháng 03 năm 2023*

**KHAI THÁC DỮ LIỆU VÀ ỨNG DỤNG – CS313.N22**

***Bài tập 3:*** *Sử dụng ngôn ngữ Python triển khai bài toán Khai Thác Tập Phổ Biến Và Luật Kết Hợp với các thuật toán cơ bản*

**Nhóm 3:**

Bùi Nguyễn Anh Trung (NT) 20520332

Hồ Thanh Tịnh 20520813

Nguyễn Trần Minh Anh 20520394

Lê Nguyễn Bảo Hân 20520174

Nguyễn Văn Đức Ngọc 20521666

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

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỌC MÁY TÍNH**

Nội dung

[**1. Các thuật toán khai thác tập phổ biến 3**](#_Toc131620600)

[**1.1. Thuật toán Apriori 3**](#_Toc131620601)

[**1.1.1. Nội dung thuật toán 3**](#_Toc131620602)

[**1.1.2. Các bước thực hiện 4**](#_Toc131620603)

[**1.1.3. Mã giả 4**](#_Toc131620604)

[**1.1.4. Ví dụ minh họa 6**](#_Toc131620605)

[**1.1.5. Áp dụng Apriori khai thác luật kết hợp bằng Python 9**](#_Toc131620606)

[**1.2. Thuật toán AprioriTID 11**](#_Toc131620607)

[**1.2.1. Nội dung thuật toán 11**](#_Toc131620608)

[**1.2.2. Các bước thực hiện 12**](#_Toc131620609)

[**1.2.3. Mã giả 12**](#_Toc131620610)

[**1.2.4. Ví dụ minh họa 13**](#_Toc131620611)

[**1.2.5. Cấu trúc lưu trữ 16**](#_Toc131620612)

[**1.2.6. Nhận xét 17**](#_Toc131620613)

[**1.3. Thuật toán FP-Growth 17**](#_Toc131620614)

[**1.3.1. Giới thiệu thuật toán 17**](#_Toc131620615)

[**1.3.2. Các bước thực hiện 18**](#_Toc131620616)

[**1.3.3. Mã giả 18**](#_Toc131620617)

[**1.3.4. Ví dụ minh họa 22**](#_Toc131620618)

[**1.4. Thuật toán EClaT 25**](#_Toc131620619)

[**1.4.1. Nội dung thuật toán 26**](#_Toc131620620)

[**1.4.2. Các bước thực hiện 27**](#_Toc131620621)

[**1.4.3. Mã giả 28**](#_Toc131620622)

[**1.4.4. Nhận xét 29**](#_Toc131620623)

[**1.4.5. Áp dụng EClaT khai thác luật kết hợp bằng Python 29**](#_Toc131620624)

[**2. Độ đo đánh giá 33**](#_Toc131620625)

[**2.1. Support 33**](#_Toc131620626)

[**2.2. Confidence 34**](#_Toc131620627)

[**2.3. Lift 34**](#_Toc131620628)

[**2.4. Conviction 34**](#_Toc131620629)

[**3. Tham khảo 35**](#_Toc131620630)

**Bảng phân công**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Thành**  **viên**  **Công**  **Việc** | **Bùi Nguyễn Anh Trung** | **Hồ Thanh Tịnh** | **Nguyễn Trần Minh Anh** | **Lê Nguyễn Bảo Hân** | **Nguyễn Văn Đức Ngọc** |
| Phân công, quản lý công việc chung | Checkmark with solid fill |  | Checkmark with solid fill |  |  |
| Tìm hiểu thuật toán Apriori |  | Checkmark with solid fill |  |  | Checkmark with solid fill |
| Tìm hiểu thuật toán AprioriTID |  |  |  | Checkmark with solid fill | Checkmark with solid fill |
| Tìm hiểu thuật toán FP-Growth | Checkmark with solid fill | Checkmark with solid fill |  |  |  |
| Tìm hiểu thuật toán EClaT |  |  | Checkmark with solid fill | Checkmark with solid fill |  |
| Tổng hợp nội dung và định dạng báo cáo | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| Chuẩn bị demo Thuyết trình | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| Làm slide | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| Mức độ hoàn thiện (%) | 100% | 100% | 100% | 100% | 100% |

# Các thuật toán khai thác tập phổ biến

Có 2 dạng khai thác tập phổ biến chính là theo chiều ngang và theo chiều dọc. Đa số các thuật toán khai thác tập phổ biến đều dựa trên cách khai thác theo chiều ngang: mỗi tập phổ biến sẽ có một mã định danh (TID) và tập phổ biến {TID: Itemset}.

Bảng 1. Khai thác theo chiều ngang

|  |  |
| --- | --- |
| **TID** | **List of item\_IDs** |
| T10 | Coke, Slice, Kurkure |
| T20 | Coke, Kurkure |
| T30 | Slice, Slice |

Ngược lại, cách khai thác theo chiều dọc sẽ triển khai dữ liệu theo một tập phổ biến và tập định danh của nó {Item: Tidset}

Bảng 2. Khai thác theo chiều dọc

|  |  |
| --- | --- |
| **Itemsets** | **TID\_set** |
| Coke | T10, T20 |
| Slice | T10, T30 |
| Kurkure | T10, T20 |
| Pizza | T30 |

## Thuật toán Apriori

### Nội dung thuật toán

***Thuật toán Apriori là gì?***

Thuật toán Apriori [1] được công bố bởi R. Agrawal và R. Srikant vào năm 1994 để tìm các tập phổ biến trong một bộ dữ liệu lớn. Tên của thuật toán là Apriori vì nó sử dụng kiến thức đã có từ trước (prior) về các thuộc tính, vật phẩm thường xuyên xuất hiện trong cơ sở dữ liệu. Để cải thiện hiệu quả của việc lọc các mục thường xuyên theo cấp độ, một thuộc tính quan trọng được sử dụng gọi là thuộc tính Apriori giúp giảm phạm vi tìm kiếm của thuật toán.

***Ứng dụng của thuật toán Apriori***

Thuật toán Apriori có thể nhìn vào quá khứ và đưa ra xác suất nếu một việc xảy ra kéo theo sự việc tiếp theo xảy ra. Điều này rất có ích cho các nhà kinh doanh. Ví dụ, một siêu thị muốn nghĩ cách sắp xếp các gian hàng một cách hợp lí nhất, họ có thể nhìn vào lịch sử mua hàng và sắp xếp các tập sản phẩm thường được mua cùng nhau vào một chỗ. Hoặc một trang web tin tức muốn giới thiệu cho người dùng các bài viết liên quan đến nhau nhất, cũng có thể áp dụng quy luật tương tự.

### Các bước thực hiện

***Ý tưởng:*** *Nếu không phải là tập phổ biến thì tập bao nó cũng không phổ biến.*

Phương pháp bao gồm các bước:

* **Bước 1:** Tìm tất cả các tập phổ biến 1- hạng mục.
* **Bước 2:** Tạo các tập ứng viên kích thước k-hạng mục Ck (k - candidate itemset) từ các tập phổ biến có kích thước L(k-1) hạng mục.
* **Bước 3:** Kiểm tra độ phổ biến của các ứng viên trên CSDL và loại các ứng viên không phổ biến.
* **Bước 4:** Dừng khi không tạo được tập phổ biến hay tập ứng viên

### Mã giả

**Ck:** Tập ứng viên có kích thước là k

**Lk:** Tập phổ biến có kích thước là k

**D:** Cơ sở dữ liệu gốc

**L1** = {Tập ứng viên có kích thước là 1}

**for** (k = 1; Lk != ∅; k++) **do begin** //Đến khi nào không tìm được tập phổ biến nữa thì dừng

Ck+1 = Tập ứng viên kích thước k+1 sinh ra từ tập phổ biến trước Lk;

(𝐶𝑘+1 = *apriori\_gen*(Lk))

**for** (𝑐 ∈ 𝐶K+1)

c.count = 0;

**for** (𝑡 ∈ 𝐷) //Quét các tập giao dịch trong D, với mỗi phần tử t trong D thì ta quét qua tất cả các phần tử c trong 𝐶K+1 nếu là con thì tăng biến đếm của c

{

**for** (𝑐 ∈ 𝐶K+1 and c ⊂ t )

c.count++;

}

Lk+1 = {𝑐 ∈ 𝐶 𝑘+1|𝑐. 𝑐𝑜𝑢𝑛𝑡 ≥ 𝑚𝑖𝑛𝑠𝑢𝑝𝑝} //Tìm được Lk dựa trên minsup cung cấp từ đầu và các c.count tính bước trên

**return** ∪ k L k; //Thêm Lk vào L là tập chứa tất cả các tập phổ biến thỏa mãn

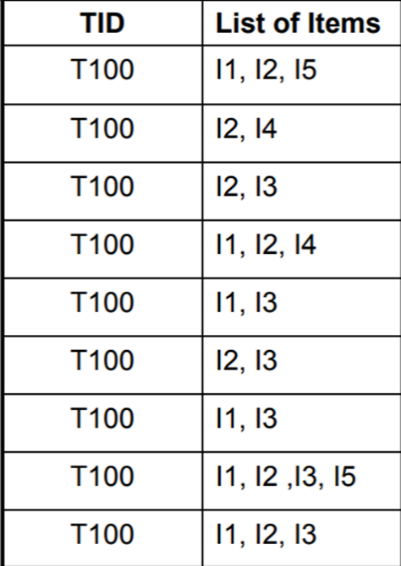
***Lưu ý:*** Thật ra để tìm được Lk+1 từ Ck+1 thì có 2 bước loại bỏ là ở bước

𝐶𝑘+1 = *apriori\_gen*(Lk): bước này ta tìm tập ứng viên 𝐶𝑘+1 từ Lk ,ta kết hợp các phần tử Lk lại để tìm ra các ứng viên có k +1 phần tử sau đó xem xét các tập con ứng cử viên gồm k + 1 phần tử mới được tạo ra này có trong tập phổ biến Lk hay không nếu tồn tại 1 con mà không thuộc Lk thì ta loại bỏ phần tử đó ra khỏi Ck+1

Lk+1 = {𝑐 ∈ 𝐶𝑘+1 | 𝑐. 𝑐𝑜𝑢𝑛𝑡 ≥ 𝑚𝑖𝑛𝑠𝑢𝑝𝑝}: bước này loại bỏ các phần tử c trong Ck+1 dựa trên *minsup* cung cấp ban đầu.

### Ví dụ minh họa

Cho tập giao dịch *D* với *minsup*=2/9 (support count=2) và *mincof*=70%



**Bước 1:** Tạo tập phổ biến gồm các phần tử có 1 item

**Table

Description automatically generated**

Ta tìm được C1 như hình bên dựa trên phép đếm số lần mà các item xuất hiện trong các phần tử của tập giao dịch D (quét qua cơ sở dữ liệu D).

Table

Description automatically generated

Tìm được L1 dựa trên C1 trên và thông qua việc duyệt qua các phần tử C1 thỏa mãn *minsup*=2 mà đề bài ban đầu yêu cầu.

**Bước 2:** Tạo tập phổ biến gồm các phần tử có 2 item

**Table

Description automatically generated**

Ta sinh ra C2 từ L1 và duyệt qua cở sở D đếm số lần xuất hiện của các phần tử trong C­2. Đối chiếu với *minsup*=2, loại bỏ các phần tử có *supcount*<2 từ C2 ta được tập L2 như bên dưới

Table

Description automatically generated

**Bước 3:** Tạo tập phổ biến gồm các phần tử có 3 item. Tương tự bước 1, 2 ta thu được C3=L3 như bên dưới

Table

Description automatically generated

**Bước 4:** Tạo tập phổ biến gồm các phần tử có 4 item

C4 lúc đầu được sinh ra từ L3 là {{I1, I2, I3, I5}} nhưng sau đó tập mục này bị lược bớt vì tập con của nó {{I2, I3, I5}} không là tập phổ biến có trong L3.

Do đó, C4 = φ và thuật toán kết thúc.

L=∪k Lk

**Bước 5:** Tạo tập phổ biến gồm các phần tử có 3 item

Ta có L = {{I1}, {I2}, {I3}, {I4}, {I5}, {I1,I2}, {I1,I3}, {I1,I5}, {I2,I3}, {I2,I4}, {I2,I5}, {I1,I2,I3}, {I1,I2,I5}}

* Giả sử I = {I1,I2,I5}.
* Tất cả các tập con khác rỗng của nó là {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2}, {I5}.
* minconf=70%

Các luật kết hợp sinh ra từ tập I được tính với độ tin cậy như dưới đây:

R1: I1 ^ I2  I5

* Confidence = suppc{I1,I2,I5}/suppc{I1,I2} = 2/4 = 50%

 R1 bị loại do <70%

R2: I1 ^ I5  I2

* Confidence = suppc{I1,I2,I5}/suppc{I1,I5} = 2/2 = 100%

 R2 được chọn

R3: I2 ^ I5  I1

* Confidence = suppc{I1,I2,I5}/suppc{I2,I5} = 2/2 = 100%

 R3 được chọn (>70%)

R4: I1  I2 ^ I5

* Confidence = suppc{I1,I2,I5}/suppc{I1} = 2/6 = 33%

 R4 bị loại

R5: I2  I1 ^ I5

* Confidence = suppc{I1,I2,I5}/suppc{I2} = 2/7 = 29%

 R5 bị loại

R6: I5  I1 ^ I2

* Confidence = suppc{I1,I2,I5}/ suppc{I5} = 2/2 = 100%

 R6 được chọn

Như vậy ta đã tìm ra được các luật kết hơp của I (I thuộc L)

Tương tự như trên ta cũng sẽ tìm được các luật kết hợp các phần tử của L và cuối cùng ta sẽ được tất các các luật kết hợp từ tập D với *minsupp* và *minconf* ta cung cấp ban đầu.

### Áp dụng Apriori khai thác luật kết hợp bằng Python

***Mô tả tập dữ liệu***

Các sản phẩm khác nhau được cung cấp 7500 giao dịch trong vòng một tuần tại một cửa hàng bán lẻ ở Pháp.

Ta dùng thư viện **apyori** để tính quy tắc kết hợp bằng Apriori.

***Khai báo các thư viện sẽ dùng:***

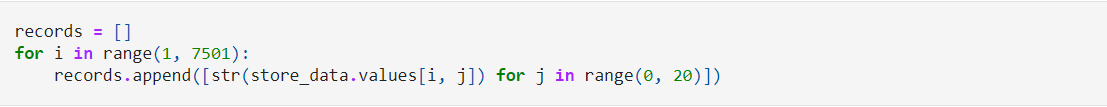
A picture containing graphical user interface

Description automatically generated

Đọc dữ liệu từ file csv, ta thấy có 20 cột chính là những item và có 7501 dòng dữ liệu:

A picture containing graphical user interface

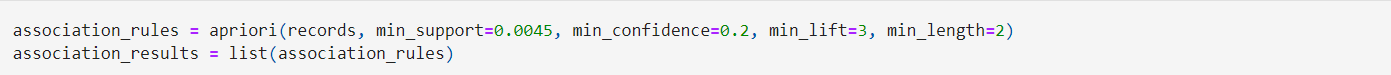
Description automatically generated***Tiền xử lí dữ liệu:*** chuyển dữ liệu ban đầu thành 1 list



***Sử dụng thư viện apriori:*** Dùng dữ liệu đã được xử lí trên và cung cấp các tham số cho hàm: *min\_support, min\_confidence, min\_lift, min length, min\_length*  để tìm ra luật kết hợp cho bộ dữ liệu đầu

* *min\_support (minsupp):* Độ phổ biến ban đầu ta đặt ra để tìm ra các tập phổ biến hay còn gọi là độ phổ biến tối thiểu
* *min\_confidence (min\_conf):* Độ phổ biến ban đầu ta đặt ra để tìm ra các luật kết hợp từ các tập phổ biến thông qua thuật toán apriori hay còn gọi là độ tin cậy tối thiểu
* *min length, min\_length:* giới hạn nhỏ nhất ,lớn nhất của độ lớn của tập phổ biến sinh ra luật kết hợp ta cần tìm
* *min\_lift:* Sau khi tìm ra các luật kết hợp ta sử dùng giá trị này để tìm ra các luật kết hợp có giá trị. Lift(a,b)=supp(a,b)/(supp(a)\*supp(b))

Ta tính được các luật kết hợp thông qua hàm apriori ở thư viện đã khai báo:



In ra các tập phổ biển cần tìm:

Graphical user interface, text

Description automatically generated

In ra các luật kết hợp cùng các giá trị *supp, conf, lift* của luật kết hợp đó:

Graphical user interface, text, application

Description automatically generated

## Thuật toán AprioriTID

### Nội dung thuật toán

Thuật toán AprioriTID [1]:

* Có cùng phương thức tạo tập ứng viên như Apriori (*apriori\_gen*)
* Sau lần duyệt đầu tiên, thực hiện kiểm tra độ phổ biến (đếm *support*) của các ứng viên trên tập thay vì cơ sở dữ liệu *D*
* Mỗi ứng viên của tập có dạng với là tập *k-hạng mục* tiềm năng có trong giao dịch t có mã là TID, hay có thể viết
* Giao dịch không chứa bất kì ứng viên nào từ tập *k-hạng mục* sẽ không được đưa vào . Do đó, số lượng ứng viên được đưa vào có thể nhỏ hơn số lượng giao dịch trong cơ sở dữ liệu, đặc biệt là khi *k* có giá trị lớn

### Các bước thực hiện

**Bước 1:** Quét qua các giao dịch để tìm tất cả item có độ *Support* lớn hơn *MinSupport* và đưa tập *Large 1-Itemsets* vào

**Bước 2:** Đưa toàn bộ các TID của giao dịch cùng các items vào dưới dạng

**Bước 3:** Xây dựng các cặp 2-items từ đưa vào tập ứng viên . Quét tất cả các giao dịch trong để tìm tất cả các tập *Large 2-Itemsets* từ đưa vào dưới dạng , đồng thời đưa các tập *Large 2-Itemsets* ứng viên vào .

**Bước 4:** Xây dựng các cặp *k* items từ đưa vào tập ứng viên . Quét tất cả các giao dịch trong để tìm tất cả các tập *Large k-Itemsets* từ và đưa vào dưới dạng , đồng thời đưa các tập *Large k-Itemsets* vào . Lặp lại *Bước 4* cho đến khi hết ứng viên mới. Phát sinh Luật.

### Mã giả

1. ; //tạo tập ứng viên

4. //xác định tập ứng viên có chứa trong giao dịch t.TID



### Ví dụ minh họa

Cho một tập giao dịch TID với các Items như sau (với *MinSupport=50%*, *MinConfidence=60%):*

|  |  |
| --- | --- |
| **TID** | **Items** |
| 100 | {1, 3, 4} |
| 200 | {2, 3, 5} |
| 300 | {1, 2, 3, 5} |
| 400 | {2, 5} |

Lấy toàn bộ đưa vào

|  |  |
| --- | --- |
| **TID** | **Tập 1-item** |
| 100 | { {1 }, {3}, {4} } |
| 200 | { {2}, {3}, {5} } |
| 300 | { {1}, {2}, {3}, {5} } |
| 400 | { {2}, {5} } |

Tính tập *Large1\_Itemsets*, ta có :

|  |  |
| --- | --- |
| **Tập 1-item** | **Support** |
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {5} | 3 |

Ở bước kết từ trên ta có tập gồm các cặp 2-item:

|  |
| --- |
| **Tập 2-item** |
| {1,2} |
| {1,3} |
| {1,5} |
| {2,3} |
| {2,5} |
| {3,5} |

Xác định ứng viên từ khi duyệt TID trong và đưa vào

|  |  |
| --- | --- |
| **TID** | **Tập 2-item** |
| 100 | { {1,3} } |
| 200 | { {2,3}, {2,5}, {3,5} } |
| 300 | { {1,2}, {1,3}, {1,5},  {2,3}, {2,5}, {3,5} } |
| 400 | { {2,5} } |

Tính tập *Large2\_Itemsets*, ta có :

|  |  |
| --- | --- |
| **Tập 2-item** | **Support** |
| {1,3} | 2 |
| {2,3} | 2 |
| {2,5} | 3 |
| {3,5} | 2 |

Ở bước kết từ ta có tập gồm các cặp 3-item:

|  |
| --- |
| **Tập 3-item** |
| {2,3,5} |

Xác định ứng viên từ khi duyệt TID trong và đưa vào

|  |  |
| --- | --- |
| **TID** | **Tập 3-item** |
| 200 | { {2,3,5} } |
| 300 | { {2,3,5} } |

Tính tập *Large3\_Itemsets*, ta có :

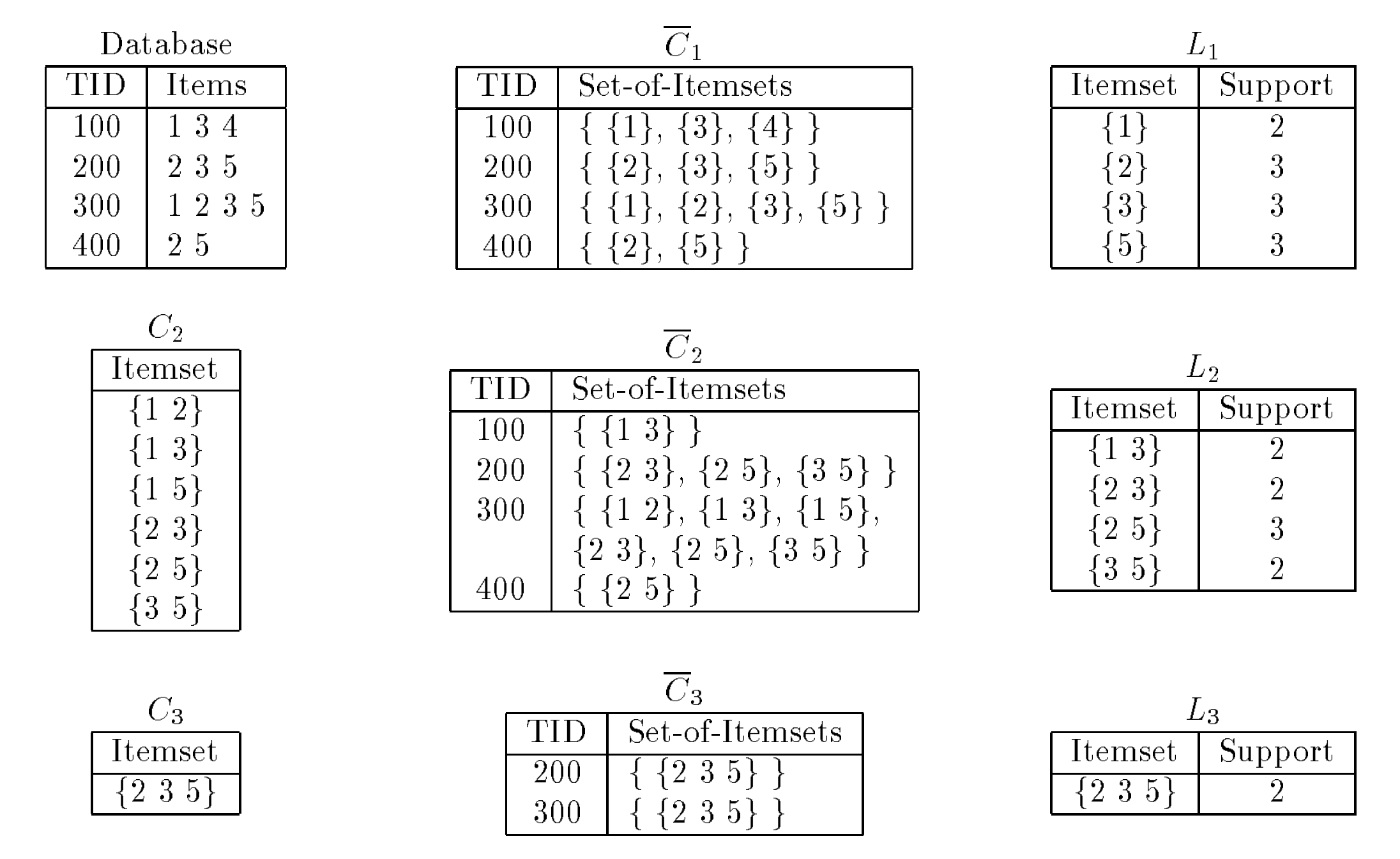
|  |  |
| --- | --- |
| **Tập 3-item** | **Support** |
| {2,3,5} | 2 |

Phát sinh luật:

* 2,3  5 có độ Confidence 2/2 = 100%
* 2,5  3 có độ Confidence 2/3 = 66,66%
* 3,5  2 có độ Confidence 2/2 = 100%

Ở bước kết từ ta có tập gồm các cặp 4-item là .

Thuật toán kết thúc.



Hình 1. Ví dụ minh họa thuật toán AprioriTID

### Cấu trúc lưu trữ

* Mỗi ứng viên trong *Itemsets* sẽ được gán cho một mã số **ID** duy nhất. Mỗi tập *Itemsets* được lưu trong một mảng. Mỗi ứng viên của bây giờ có dạng
* Hàm **Apriori\_Gen** phát sinh một tập các ứng viên *k-ItemSet* bằng cách kết hợp hai tập *Large(k-1)\_Itemsets*. Mỗi ứng viên thêm hai trường:

1. *Generators*
2. *Extensions*

* Trường *generators* của lưu các ID của hai tập *Large(k-1)\_Itemsets* kết với nhau để phát sinh
* Trường *extensions* của lưu các ID của các tập *Large(k+1)\_Itemsets* kết với nhau để phát sinh
* Khi một ứng viên *Itemsets* được phát sinh bằng cách kết hợp và thì các ID sẽ được lưu vào trường *generators* của , đồng thời ID của được lưu vào trường *extensions của*

### Nhận xét

* **Apriori**: Để xác định độ phổ biến của các tập ứng viên, Apriori luôn phải quét lại toàn bộ các giao dịch trong cơ sở dữ liệu. Điều này gây tốn rất nhiều thời gian khi giá trị của k-items tăng (số lần xét duyệt các giao dịch tăng).
* **AprioriTID**: Trong quá trình xét duyệt khởi tạo, kích thước thường rất lớn và hầu hết tương đương với CSDL gốc. Ngoài ra, thuật toán AprioriTID còn phải gánh chịu thêm chi phí phát sinh nếu vượt quá bộ nhớ trong mà phải sử dụng kèm bộ nhớ ngoài.

## Thuật toán FP-Growth

### Giới thiệu thuật toán

* Thuật toán FP-Growth được giới thiệu lần đầu tiên vào năm 1996 bởi Jiawei Han, Jian Pei và Yiwen Yin với tên gọi "Mining Frequent Patterns without Candidate Generation". Tuy nhiên, thuật toán này chưa được phổ biến cho đến khi phiên bản của nó được tối ưu hóa lại vào năm 2000. Tại đó, Jiawei Han, Jian Pei và Yongjian Fu đã đề xuất một phiên bản cải tiến của thuật toán mang tên "FP-Growth", giúp tăng tốc độ xử lý và giảm độ phức tạp tính toán so với phiên bản ban đầu.
* Kể từ đó, FP-Growth đã trở thành một trong những thuật toán phổ biến nhất trong khai phá dữ liệu, đặc biệt là trong việc tìm kiếm các mẫu kết hợp phổ biến. Hơn nữa, nó cũng được sử dụng rộng rãi trong các ứng dụng thực tế như phân tích dữ liệu, quản lý kho sản phẩm, đề xuất sản phẩm, phân loại tin nhắn thư rác và nhiều lĩnh vực khác.

### Các bước thực hiện

**Bước 1 :** Đếm tần suất xuất hiện của các phần tử trong tập dữ liệu và xác định tập các phần tử phổ biến.

**Bước 2 :** Xây dựng FP-Tree từ tập dữ liệu và tập các phần tử phổ biến.

**Bước 3 :** Duyệt qua FP-Tree để tìm các itemset phổ biến (được biểu diễn trên FP-Tree dưới dạng các đường đi từ gốc đến lá).

**Bước 4 :** Dùng các itemset phổ biến để tạo các luật kết hợp.

### Mã giả

**Input:** tập dữ liệu D, ngưỡng hỗ trợ tối thiểu minsup

**Output:** tập phổ biến

**Bước 1:** Đếm tần số và lọc các mục dưới minsup

*freq = {}*

*for transaction in D:*

*for item in transaction:*

*freq[item] = freq.get(item, 0) + 1*

*freq = {item: count for item, count in freq.items() if count >= minsup}*

*freq\_items = set(freq.keys())*

**Bước 2:** Xây dựng cây FP-Tree

*root = Node(None)*

*for transaction in D:*

*transaction = [item for item in transaction if item in freq\_items]*

*transaction = sorted(transaction, key=lambda item: freq[item], reverse=True)*

*current\_node = root*

*for item in transaction:*

*if item in current\_node.children:*

*current\_node.children[item].count += 1*

*else:*

*new\_node = Node(item)*

*new\_node.parent = current\_node*

*current\_node.children[item] = new\_node*

*if freq[item] > freq[current\_node.item]:*

*current\_node.item = item*

*if item not in freq\_items:*

*freq\_items.add(item)*

*freq[item] = 0*

*new\_node.link = freq[item]*

*freq[item] += 1*

*current\_node = current\_node.children[item]*

**Bước 3:** Khai thác tập mục phổ biến đệ quy

*freq\_itemsets = []*

*prefix = []*

*mine\_fptree(root, freq, minsup, prefix, freq\_itemsets)*

***def mine\_fptree(node, freq, minsup, prefix, freq\_itemsets):***

*# Tạo cơ sở mẫu điều kiện*

*cond\_items = {}*

*while node:*

*prefix\_path = get\_prefix\_path(node)*

*if len(prefix\_path) > 1:*

*cond\_items[frozenset(prefix\_path[1:])] = node.count*

*node = node.link*

*# Tạo Conditional FP-Tree*

*cond\_tree, cond\_freq = construct\_fptree(cond\_items, minsup)*

*if cond\_tree:*

*if prefix:*

*freq\_set = prefix.copy()*

*else:*

*freq\_set = set()*

*freq\_set.add(node.item)*

*freq\_itemsets.append(freq\_set)*

*mine\_fptree(cond\_tree, cond\_freq, minsup, freq\_set, freq\_itemsets)*

***def get\_prefix\_path(node):***

*path = []*

*while node and node.item is not None:*

*path.append(node.item)*

*node = node.parent*

*return path[::-1]*

*def construct\_fptree(items, minsup):*

*freq = {}*

*root = Node(None)*

*for itemset, count in items.items():*

*transaction = []*

*for item in itemset:*

*if item in freq:*

*freq[item] += count*

*else:*

*freq[item] = count*

*if freq[item] >= minsup:*

*transaction.append(item)*

*if transaction:*

*add\_transaction(root, transaction, freq)*

*return root, freq*

*def add\_transaction(node, transaction, freq):*

*if not transaction:*

*return*

*item = transaction[0]*

*if item in node.children:*

*child = node.children[item]*

*else:*

*child = Node(item)*

*child.parent = node*

*node.children[item] = child*

*if freq[item] > freq[node.item]:*

*node.item = item*

*child.increment()*

*add\_transaction(child, transaction[1:], freq)*

### Ví dụ minh họa

|  |  |
| --- | --- |
| **TID** | **Item** |
| T100 | A, B, C, D, F, H, K |
| T200 | C, D, E, K, G, E |
| T300 | A, B, F, G, H |
| T400 | B, C, E, F, G |
| T500 | A, C, D, E, F |

Cho minsupp=40%, minconf=70%.

Hãy tìm tập phổ biến với thuật toán FP Growth.

Bài làm

* Tập phổ biến 1 phần tử và tần suất xuất hiện của nó

|  |  |
| --- | --- |
| **Item** | **Supp** |
| A | 3 |
| B | 3 |
| C | 4 |
| D | 3 |
| E | 3 |
| F | 4 |
| G | 3 |
| H | 2 |
| K | 2 |

* Tập phổ biến 1 phần tử thỏa minsupp =2 và được sắp xếp theo thứ tự giảm dần.

|  |  |
| --- | --- |
| **Item** | **Supp** |
| C | 4 |
| F | 4 |
| A | 3 |
| B | 3 |
| D | 3 |
| E | 3 |
| G | 3 |
| H | 2 |
| K | 2 |

* Điền vào cột Order Item Frequent

|  |  |  |
| --- | --- | --- |
| **TID** | **Item** | **Order Item Frequent** |
| T100 | A, B, C, D, F, H, K | C, F, A, B, D, H, K |
| T200 | C, D, E, K, G, E | C, D, E, G, K |
| T300 | A, B, F, G, H | F, A, B, G, H |
| T400 | B, C, E, F, G | C, F, B, E, G |
| T500 | A, C, D, E, F | C, F, A, D, E |

* Vẽ cây FP
* Xây dựng cơ sở điều kiện cho mỗi mục phổ biến

|  |  |
| --- | --- |
| **Item** | **Conditional Pattern Base** |
| K | {C, F, A, B, D, H}: 1; {C, D, E, G}: 1 |
| H | {C, F, A, B, D}: 1, {F, A, B, G}: 1 |
| G | {C, F, B, E}: 1; {C, D, E}: 1; {F, A, B}: 1 |
| E | {C, F, B}: 1; {C, F, A, D}: 1; {C, D}: 1 |
| D | {C, F, A}: 1, {C, F, A, B}: 1, {C}: 1 |
| B | {C, F}: 1; {C, F, A}: 1; {F, A}: 1 |
| A | {C, F}: 2, {F}: 1 |
| F | {C}: 3 |
| C | {} |

* Xây dựng cây FP điều kiện

|  |  |  |
| --- | --- | --- |
| **Item** | **Conditional Pattern Base** | **Condition FP-Tree (supp 2)** |
| K | {C, F, A, B, D, H}: 1; {C, D, E, G}: 1 | {C:2, D:2}|K |
| H | {C, F, A, B, D}: 1, {F, A, B, G}: 1 | {F:2, A:2, B:2}|H |
| G | {C, F, B, E}: 1; {C, D, E}: 1; {F, A, B}: 1 | {C:1, F:1, B:1, E:1}, {C:1, E:1}, {F:1, B:1}|G |
| E | {C, F, B}: 1; {C, F, A, D}: 1; {C, D}: 1 | {C: 2, F:2, D: 1}, {C:1, D:1}|E |
| D | {C, F, A}: 1, {C, F, A, B}: 1, {C}: 1 | {C:3, F:2, A:2}|D |
| B | {C, F}: 1; {C, F, A}: 1; {F, A}: 1 | {C:2, F:2, A:1},{F:1, A:1}|B |
| A | {C, F}: 2, {F}: 1 | {C, F}:2, {F}:1|A |
| F | {C}: 3 | {C}:3|F |
| C | {} | {} |

* Kết quả các mẫu thường xuyên

|  |  |  |
| --- | --- | --- |
| **Item** | **Condition FP-Tree (supp 2)** | **Frequent Pattern** |
| K | {C:2, D:2}|K | {K}: 2, {C, K}:2, {K, D}: 2, {C, K, D}: 2 |
| H | {F:2, A:2, B:2}|H | {H}:2, {H,F}: 2, {H, A}:2, {H, B}:2, {H, F, A}: 2, {H, F, B}: 2, {H, A, B}:2, {H, F, A, B}:2 |
| G | {C:1, F:1, B:1, E:1}, {C:1, E:1}, {F:1, B1}|G | {G}:3, {G, C}:2, {G, F}: 2, {G, B}: 2, {G, E}: 2, {C, E, G}: 2, {B, G, F}: 2 |
| E | {C: 2, F:2, D: 1}, {C:1, D:1}|E | {E}:3, {C, E}: 3, {F, E}: 2, {D, E}: 2, {D, E, C}:2, {F, C, E}: 2 |
| D | {C:3, F:2, A:2}|D | {D}: 3, {D, C}: 3, {D, F}:2, {D, A}:2, {D, C, F}: 2, {D, C, A}:2, {D, F, A}:2, {D, C, F, A}:2 |
| B | {C:2, F:2, A:1}, {F:1, A:1}|B | {B}: 3, {B, C}:2, {B, F}: 3, {B, A}:2, {F, A, B}:2, {F, C, B}:2 |
| A | {C, F}:2, {F}:1|A | {A}: 3, {C, A}:2, {A, F}:3, {F, A, C}:2 |
| F | {C}:3|F | {F}:3, {C, F}: 3 |
| C | {} | {} |

* Tập phổ biến 2 phần tử

L2 ={{C,K}:2, {K, D}:2, {H, F}:2, {H,A}:2, {H,B}:2, {G,C}:2, {G,F}:2, {G,B}:2 ,{G,E}:2, {C,E}:3, {F, E}:2, {D, E}:2, {D, C}:3, {D,F}:2, {D, A}:2,{B,C}:2, {B,F}:3, {B,A}:2, {C,A}:2, {A, F}: 3, {C,F}:3}

* Tập phổ biến 3 phần tử

L3 = {{C, K, D}: 2, {H, F, A}:2, {H, F,B}:2, {H, A, B}:2, {C, E, G}:2, {B, G, F}:2, {D, E, C}:2 ,{F, C, E}:2, {D, C, F}: 2, {D, C, A}:2, {D, F, A}:2, {F,A ,B}:2, {F, C, B}: 2, {F, A, C}:2}

* Tập phổ biến 4 phần tử

L4 = {{H, F, A, B}:2, {D, C, F, A}:2}

## Thuật toán EClaT

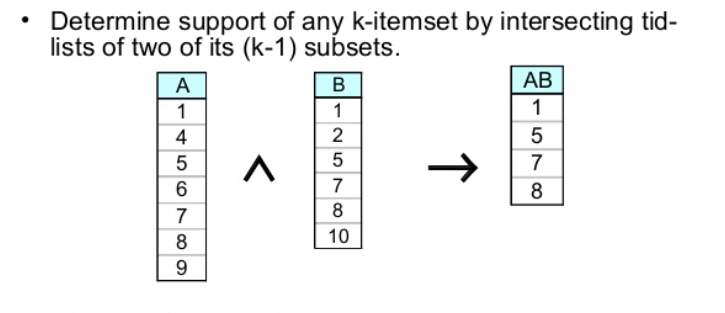
Thuật toán EClaT [2] sử dụng cấu trúc IT-tree (Tidset Itemset-tree) để lưu tidset của các tập danh mục trên mỗi nút và đưa ra khái niệm lớp tương đương để kết nối các tập danh mục trong cùng một lớp tương đương để tạo ra tập danh mục mới. Thuật toán EClaT chỉ cần một lần quét cơ sở dữ liệu là một tiếp cận hiện đại, tiết kiệm thời gian xử lý và có thể áp dụng khai thác tập phổ biến trên nhiều loại cơ sở dữ liệu một cách hiệu quả.

### Nội dung thuật toán

EClaT là bài toán điển hình của cách khai thác dữ liệu theo chiều dọc và chỉ cần duyệt qua CSDL một lần duy nhất để hình thành các tập phổ biến. Sau đó, dựa trên các tập phổ biến để tìm kiếm các tập phổ biến đóng.

Có thể hình dung EClaT giống như thuật toán tìm kiếm theo chiều sâu trên cây (Depth-First Search), trong khi Apriori là thuật toán tìm kiếm theo chiều rộng (Breadth-First Search). Tương tự như thuật toán tìm kiếm theo chiều sâu, EClaT sẽ bắt đầu từ các lá trên cây (nốt con), tới các nốt cha để hình thành tập phổ biến.

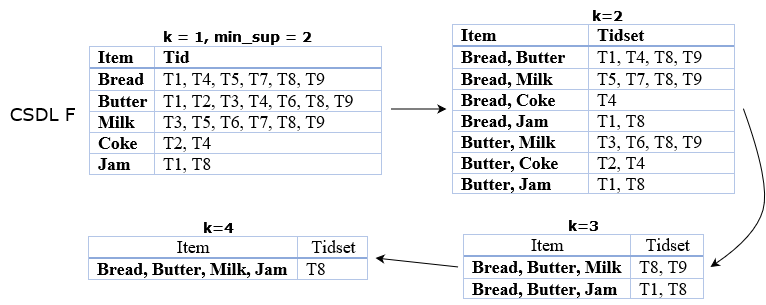
Đồng thời, thuật toán EClaT cũng không cần sinh ra các tập hợp con (subset) hay kiểm tra tập hợp con tập phổ biến như Apriori. EClaT chỉ cần giao giữa hai Tidset, từ đó tạo nên tập phổ biến. Ý tưởng chính của thuật toán dựa theo tính chất này: ***tập cha của các tập phổ biến cũng là tập phổ biến****.*[3][4]



Hình 2. Giao giữa các Tidset trong thuật toán EClaT

Dựa theo hình trên, giao giữa 2 Tidset của A và B, ta được một Tidset mới của AB với *support = 2*. Với trường hợp *minsup* là 2, không cần phải duyệt qua CSDL mà chỉ cần một lần giao là đã có thể tạo ra tập phổ biến.

### Các bước thực hiện



Hình 3. Quá trình triển khai thuật toán EClaT

Dựa trên cơ sở dữ liệu F đã được khai thác theo chiều dọc như hình trên, ta bắt đầu triển khai thuật toán EClaT:

* **Bước 1:** Giao Tidset của **{Bread}** lần lượt với các Tidset còn lại, sau đó từ Tidset của **{Butter}** lần lượt với các Tidset còn lại,… cho đến khi không còn 2 tập Tidset nào chưa được giao nữa.

Ta có các dãy các Tidset với *k = 2*.

* **Bước 2:** Tiếp tục giao từ Tidset của Itemset (*k = 2*) đã có từ bước 1: Giao từ Itemset **{Bread, Butter}** với các Itemset còn lại như đã làm ở bước 1

Ta có các Itemset mới với *k = 3.*

* **Bước 3:** Từ các Itemset có được từ bước 2, *thực hiện giao các Tidset cho đến khi chỉ còn lại một Tidset duy nhất*.

Ta có Tidset cuối cùng là **{Butter, Bread, Milk, Jam}** với *sup = 1* (Lưu ý: Tidset cuối cùng không nhất thiết phải là tập phổ biến đóng).

* **Bước 4:** Từ các tập phổ biến trên, ta bắt đầu xét các tập phổ biến với .

Ta có các gợi ý mua hàng như sau:

Bảng 3. Kết quả gợi ý mua hàng dựa trên EClaT

|  |  |
| --- | --- |
| **Items Bought** | **Recommended Products** |
| Bread | *Butter* |
| Bread | *Milk* |
| Bread | *Jam* |
| Butter | *Milk* |
| Butter | *Coke* |
| Butter | *Jam* |
| Bread and Butter | *Milk* |
| Bread and Butter | *Jam* |

Từ các bước thực hiện trên, ta thấy rằng thuật toán EClaT cơ bản chỉ hoạt động dựa trên phép giao giữa các Tidset. Ta chỉ cần thực hiện phép giao đó cho tới khi chỉ còn lại một Tidset cuối cùng.

***Tổng quát các bước thực hiện:***

1. Quét toàn bộ CSDL để xác định tập giao dịch (tidset) của các mục. Chọn các mục có *support* thỏa ngưỡng *minsup* (1-itemset).
2. Chèn *1-itemset* vào mức 1 của IT-tree
3. Mỗi nút ở mức *k* - 1 kết hợp với các nút có cùng nút cha với nó tạo ra các nút ở mức *k* nếu *support* của các nút này thỏa ngưỡng *minsup*.
4. Lặp lại bước 3 cho đến khi không thể tạo thêm nút mới trên IT-tree.
5. Duyệt cây IT-tree để lấy ra các frequent items (Tất cả các nút trên IT-tree là frequent items).

### Mã giả

**INPUT:** CSDL *D*, ngưỡng support , tiền tố của item

**OUTPUT:** Danh sách Itemsets với tiền tố được chỉ định

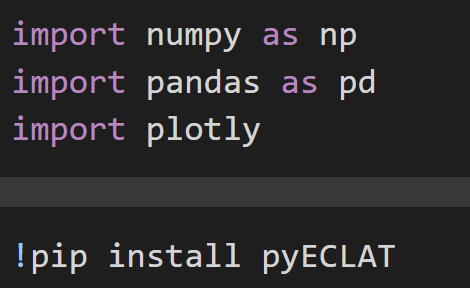
1. **Start**
2. **For all**
3. **If**
4. // initially empty
5. **If** **then start**

### Nhận xét

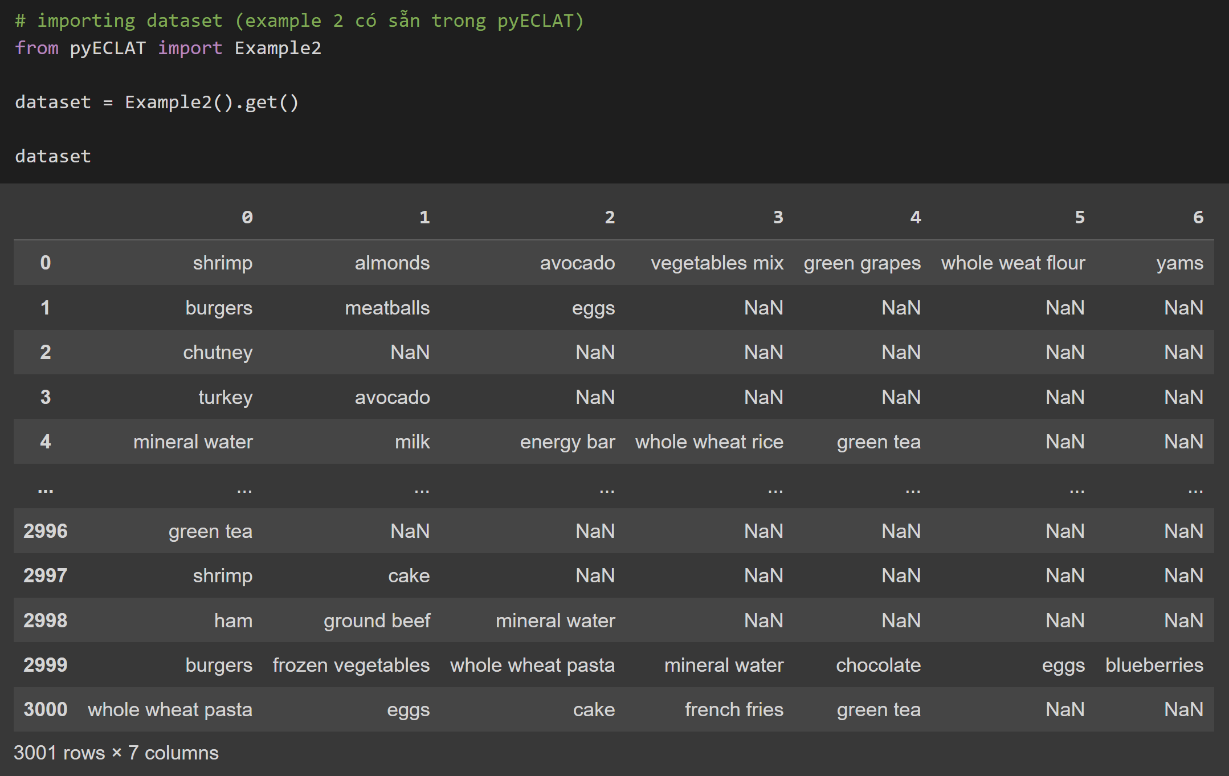
Thuật toán EClaT với cấu trúc IT-tree là hướng tiếp cận tốt được biết đến hiện nay với chỉ một lần quét CSDL, tiêu tốn ít bộ nhớ hơn Apriori. Tuy nhiên, phương pháp này có nhược điểm lớn là tốn bộ nhớ sử dụng để lưu tidset của các tập mục, do mỗi giao dịch chứa tập mục cần một ô nhớ. Đồng thời, bộ nhớ tạm cần thiết trong quá trình tính toán trung gian cũng rất lớn. Các hạn chế này làm cho thời gian tính toán của thuật toán EClaT chưa được tối ưu.

### Áp dụng EClaT khai thác luật kết hợp bằng Python

Khai báo thư viện, sử dụng thư viện **pyECLAT** [5]:

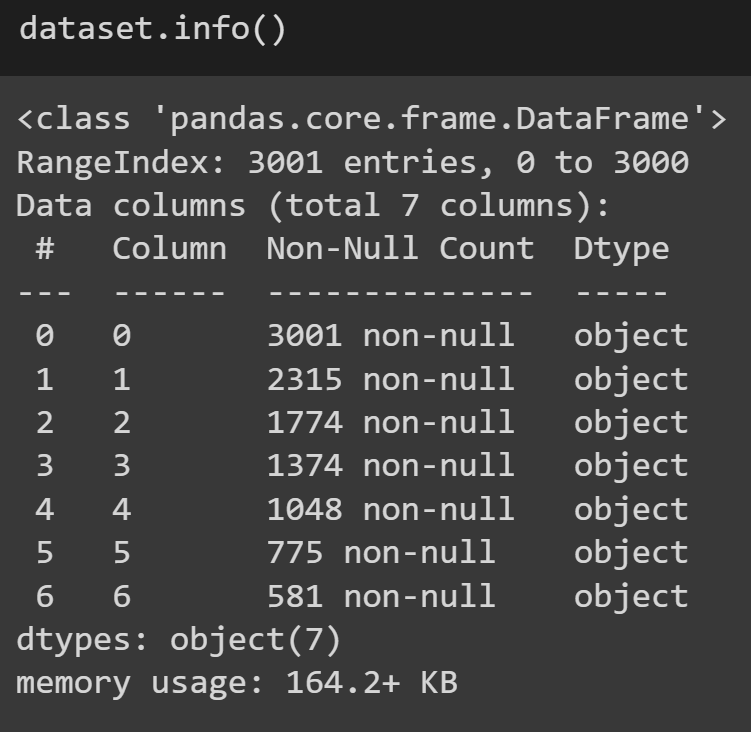


Sử dụng dataset được tích hợp sẵn trong **pyECLAT**:

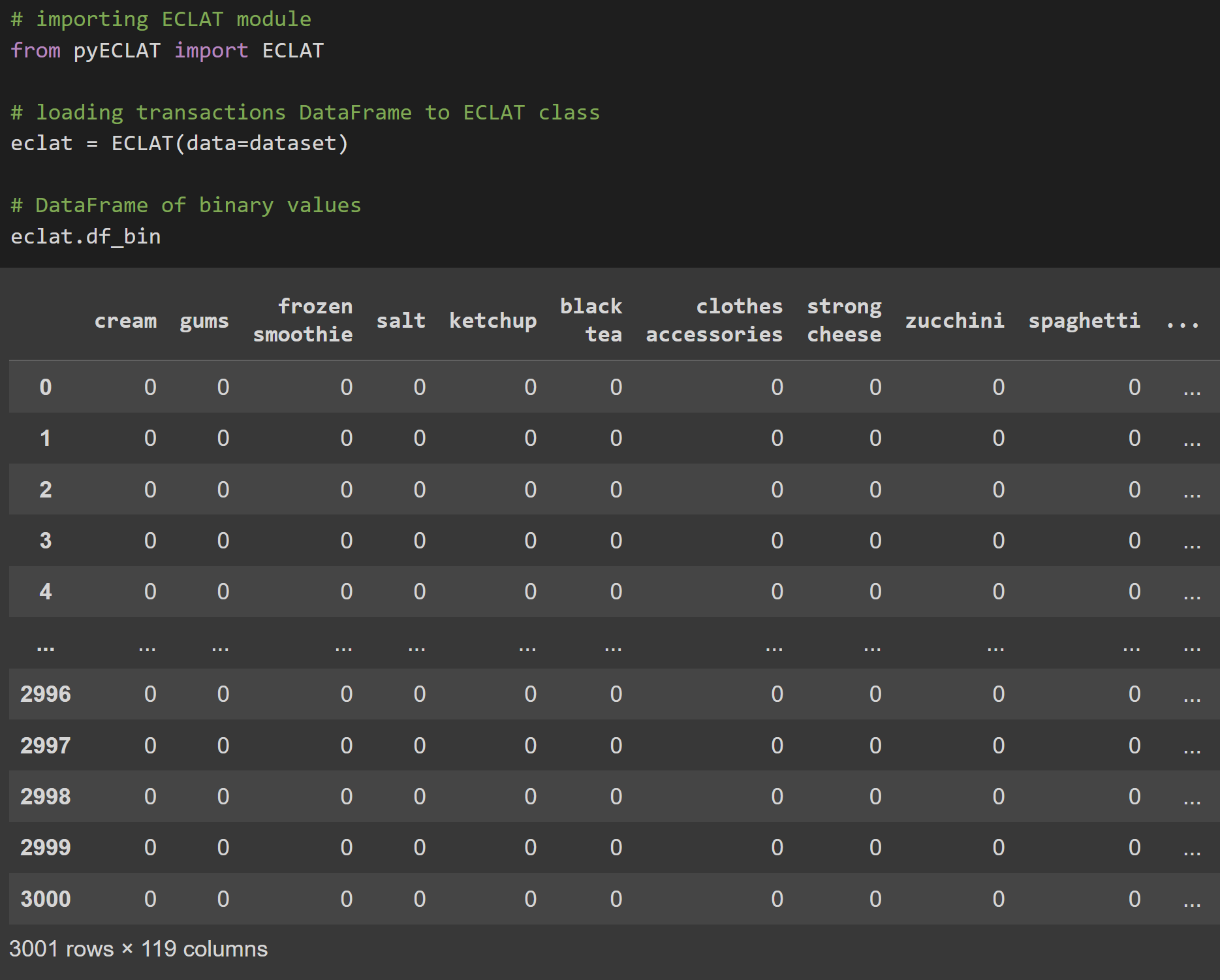


Mỗi hàng thể hiện giao dịch mua hàng của khách hàng tại siêu thị. Ví dụ: Ở hàng 1, khách hàng chỉ mua burger, thịt viên và trứng.

Chi tiết về bộ dữ liệu:



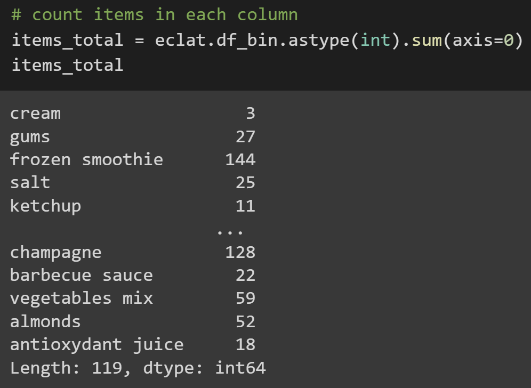
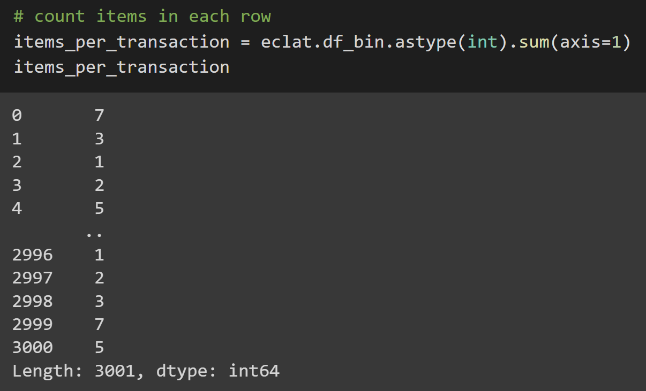
Để trực quan hóa các tập phổ biến, load dữ liệu vào lớp EClaT và tạo DataFrame nhị phân:



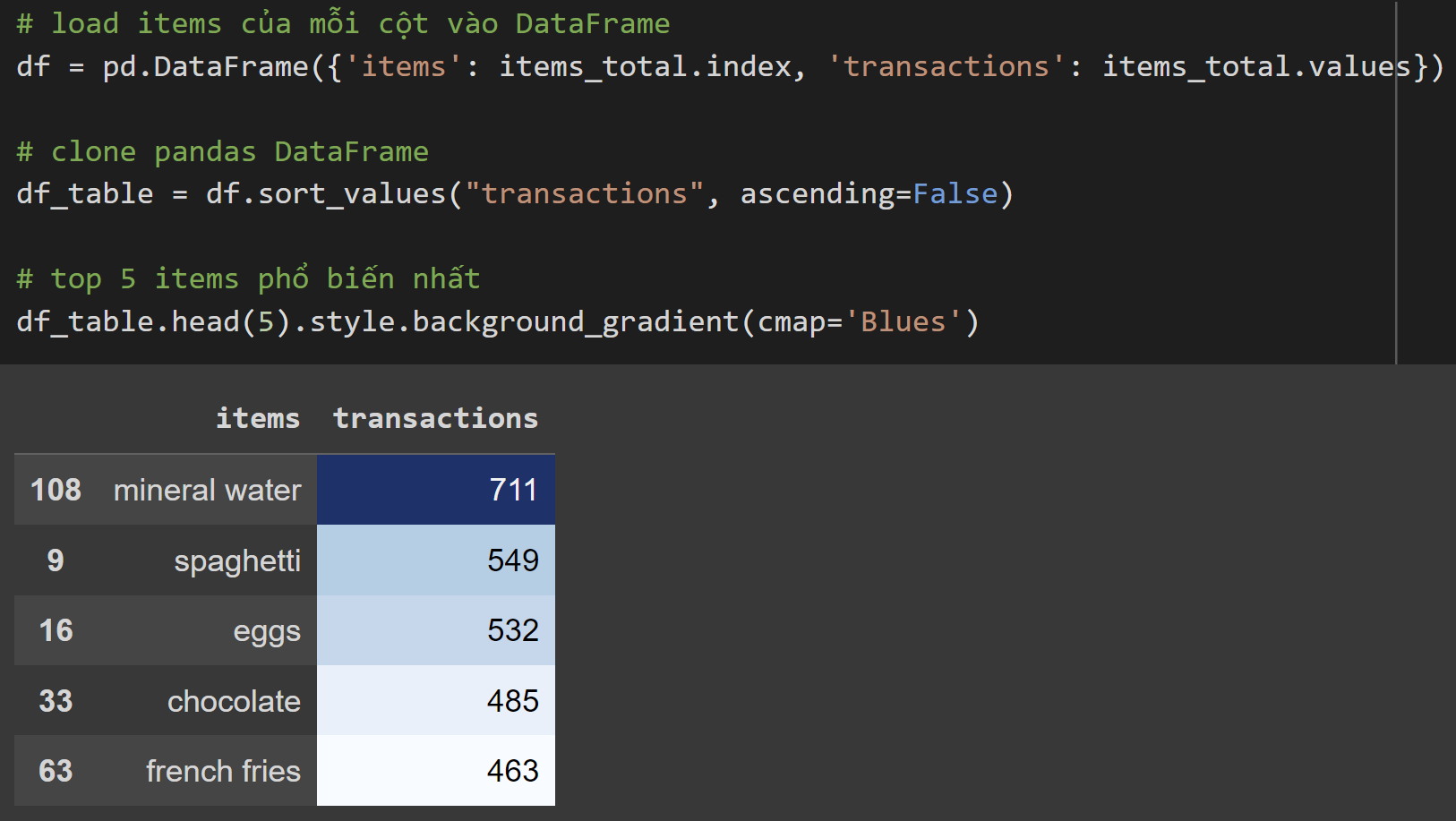
Mỗi hàng đại diện cho một giao dịch. Các cột là những sản phẩm có thể xuất hiện trong giao dịch. Mỗi ô chứa một trong hai giá trị:

* 0 - sản phẩm không có trong giao dịch
* 1 - sản phẩm có trong giao dịch

Sau đó, thực hiện đếm các items theo mỗi cột và hàng trong DataFrame:

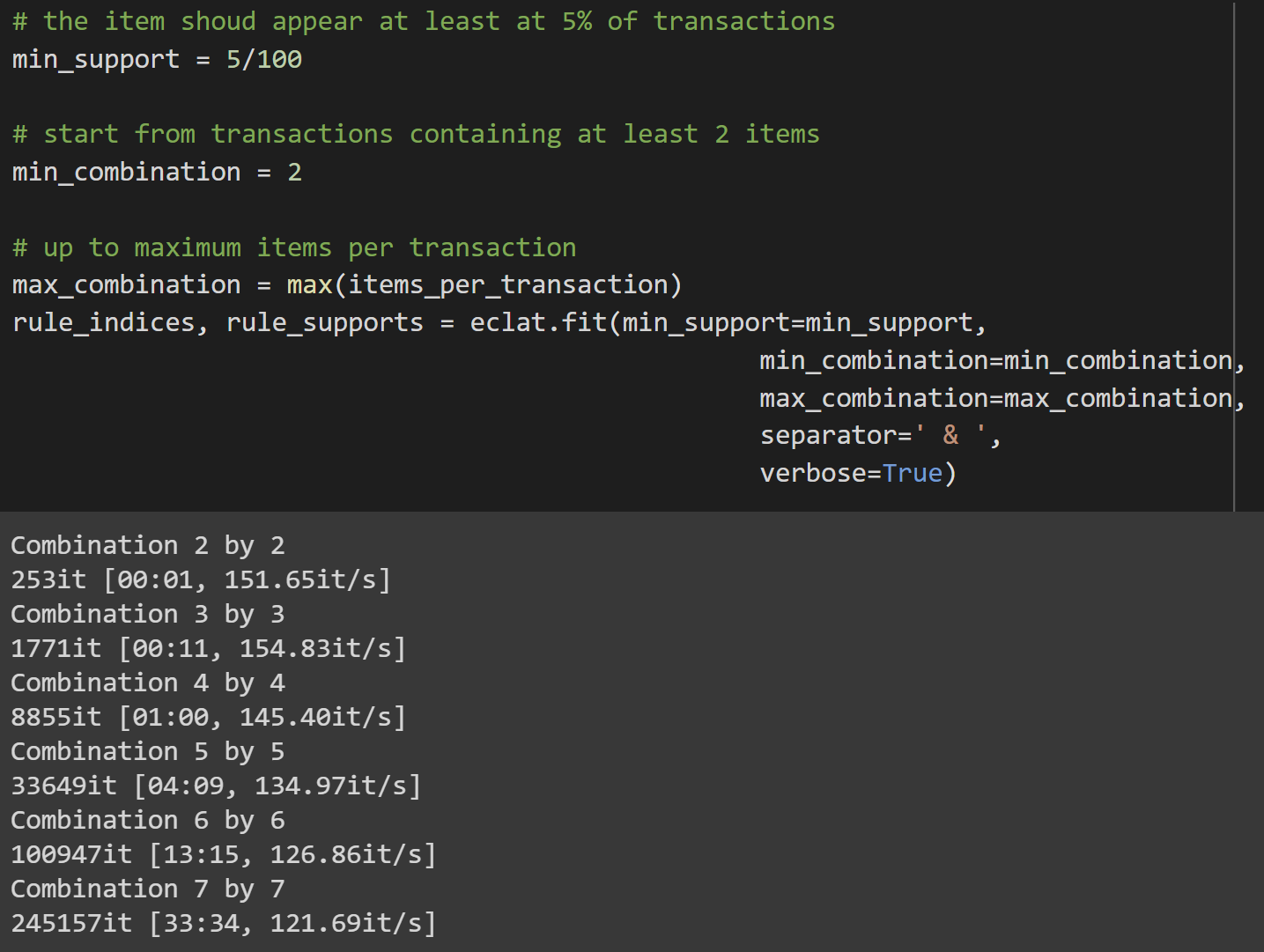
Minh họa phân phối của các items:



Để sinh luật kết hợp, ta cần định nghĩa:

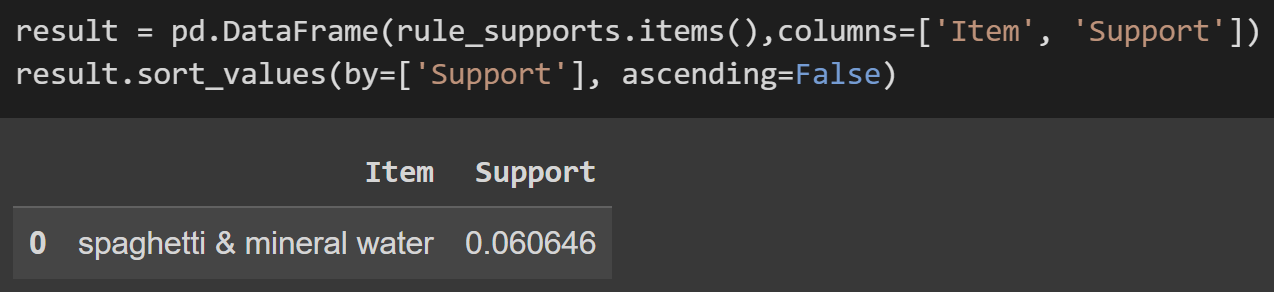
* *Minimum Support:* độ phổ biến tối thiểu
* *Minumum Combinations:* số lượng items tối thiểu trong giao dịch
* *Maximum Combinations:* số lượng items tối đa trong giao dịch

Lưu ý: Maximum Combinations càng cao thì thời gian tính toán càng lâu.



Phương thức fit() của lớp ECLAT trả về:

* Chỉ số của luật kết hợp
* Giá trị support của luật kết hợp

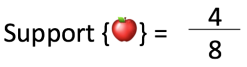


# Độ đo đánh giá

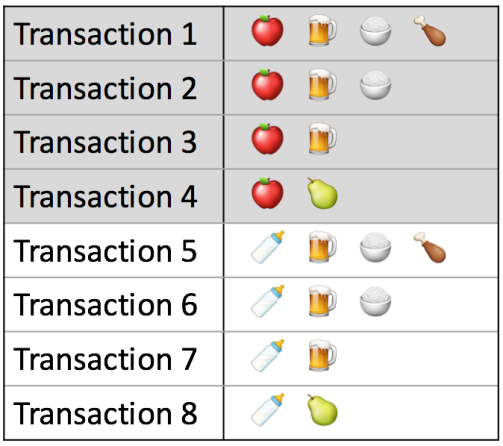
## Support

*Support* cho biết mức độ phổ biến của một danh mục, được đo bằng tỉ lệ giao dịch mà một danh mục xuất hiện. [6]

Trong bảng 4 bên dưới, support của {táo} là 4 trên 8, hay 50%. Các danh mục cũng có thể chứa nhiều mục. Chẳng hạn, support của {táo, bia, gạo} là 2 trên 8, hay 25%.



Bảng 4. Ví dụ các danh mục

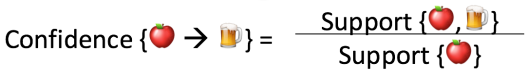


Nếu doanh số của một mặt hàng vượt quá một tỷ lệ nhất định và có xu hướng tác động đáng kể đến lợi nhuận, có thể cân nhắc sử dụng tỷ lệ đó làm ngưỡng. Xác định các danh mục có giá trị support trên ngưỡng này là danh mục quan trọng.

## Confidence

*Confidence* cho biết khả năng mặt hàng Y được mua khi mặt hàng X được mua, biểu thị bằng {X  Y}, được đo bằng tỷ lệ giao dịch của mặt hàng X mà trong đó mặt hàng Y cũng xuất hiện.

Trong Bảng 4, độ tin cậy của {táo  bia} là 3 trên 4, hay 75%.

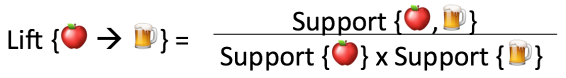


Nhược điểm của Confidence là có thể trình bày sai tầm quan trọng của một mối liên hệ. Điều này là do nó chỉ giải thích mức độ phổ biến của táo chứ khôngphải bia. Nếu bia nói chung cũng rất phổ biến thì nhiều khả năng giao dịch có táo cũng sẽ chứa bia, do đó làm tăng độ tin cậy.

## Lift

*Lift* cho biết khả năng mua mặt hàng Y khi mặt hàng X được mua (Y phụ thuộc vào X), đồng thời kiểm soát mức độ phổ biến của mặt hàng Y.

Trong Bảng X, mức tăng của {táo  bia} là 1, nghĩa là không có mối liên hệ nào giữa các mặt hàng. Giá trị nâng Lift hơn 1 có nghĩa là mặt hàng Y có khả năng được mua nếu mặt hàng X được mua, trong khi giá trị nhỏ hơn 1 có nghĩa là mặt hàng Y khó có thể được mua nếu mặt hàng X được mua.



## Conviction

*Conviction* cho biết độ tin tưởng vào *sự phụ thuộc* giữa mặt hàng Y khi mặt hàng X được mua.

Nhận xét mức độ tin cậy cũng tương tự như *Lift*, khi *Conviction* lớn hơn 1, nghĩa là vẫn có khả năng tin tưởng Y được mua khi X được mua.



# Tham khảo

|  |  |
| --- | --- |
| [1] | R. a. S. R. Agrawal, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int. Conf. Very Large Data Bases VLDB,* 2000. |
| [2] | M. Zaki, "Scalable algorithms for association mining. IEEE Trans Knowl Data Eng," *Knowledge and Data Engineering, IEEE Transactions on,* pp. 372 - 390, 2000. |
| [3] | "ML | ECLAT Algorithm," [Online]. Available: https://www.geeksforgeeks.org/ml-eclat-algorithm/. |
| [4] | "Association Rule Mining using ECLAT Algorithm," [Online]. Available: https://medium.com/machine-learning-and-artificial-intelligence/3-4-association-rule-mining-using-eclat-algorithm-b6e50aab2147. |
| [5] | "pyECLAT 1.0.2," [Online]. Available: https://pypi.org/project/pyECLAT/. |
| [6] | "Association Rules and the Apriori Algorithm: A Tutorial," [Online]. Available: https://www.kdnuggets.com/2016/04/association-rules-apriori-algorithm-tutorial.html. |