TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ MÔN XỬ LÝ DỮ LIỆU LỚN**

**CÁC TẬP MỤC PHỔ BIẾN (FREQUENT ITEMSETS)**

*Người hướng dẫn*: **TS. NGUYỄN HẢI ĐĂNG**

*Người thực hiện*: **LÊ PHAN THẾ VĨ - 52200038**

**NGUYỄN QUANG VINH - 52200195**

Lớp **: 24BD848010101**

Khoá  **: 26**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

A red and blue logo

Description automatically generated

**BÁO CÁO CUỐI KỲ MÔN XỬ LÝ DỮ LIỆU LỚN**

**CÁC TẬP MỤC PHỔ BIẾN (FREQUENT ITEMSETS)**

*Người hướng dẫn*: **TS. NGUYỄN HẢI ĐĂNG**

*Người thực hiện*: **LÊ PHAN THẾ VĨ - 52200038**

**NGUYỄN QUANG VINH - 52200195**

Lớp **: 24BD848010101**

Khoá  **: 26**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

LỜI CẢM ƠN

Chúng em xin được gửi lời cảm ơn chân thành và sâu sắc đến với trường Đại Học Tôn Đức Thắng, Khoa Công Nghệ Thông Tin và đặt biệt là thầy Nguyễn Hải Đăng - giảng viên dạy môn Xử Lí Dữ Liệu Lớn - đã hỗ trợ tận tình để chúng em có thể hoàn thành báo cáo này.

Trong quá trình bước đầu tiếp cận với môn học đòi hỏi tính tư duy cao như môn Xử Lí Dữ Liệu Lớn khó tránh khỏi những khó khăn, nhưng nhờ sự hướng dẫn đầy nhiệt huyết và tận tâm của thầy đã giúp chúng em dần hiểu sâu hơn về môn học. Không chỉ cung cấp những kiến thức về lí thuyết, thầy còn đưa ra những ví dụ thực tiễn để giúp em hiểu và nắm vững những kiến thức của môn học để áp dụng chúng vào cuộc sống. Bên cạnh đó, thầy cũng là người truyền cảm hứng để chúng em tiếp tục phấn đấu trên con đường học tập. Mặc dù đã cố gắng hoàn thành thật tốt bài tập lớn song cũng khó có thể tránh khỏi những sai sót, chúng em kính mong nhận được ý kiến đóng góp của quý thầy cô để bài tiểu luận được hoàn thiện hơn.

Một lần nữa chúng em xin cảm ơn trường Đại Học Tôn Đức Thắng, Khoa Công Nghệ Thông Tin và thầy Nguyễn Hải Đăng đã tạo điều kiện cũng như giúp đỡ chúng em hoàn thành báo cáo này.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Chúng tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của TS. Nguyễn Hải Đăng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 08 tháng 02 năm 2025*

*Tác giả*

A black line with a crown

Description automatically generated

A close-up of a logo

Description automatically generated *Lê Phan Thế Vĩ*

*Nguyễn Quang Vinh*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Bài báo cáo này chứa các nội dung lý thuyết về xử lí dữ liệu, các thuật toán trong bài toán Frequent Itemsets, ngoài ra còn giới thiệu thêm về một nhánh nhỏ của Frequent Itemsets – High Utility Itemsets.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc189680565)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc189680566)

[TÓM TẮT iv](#_Toc189680567)

[MỤC LỤC 1](#_Toc189680568)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 6](#_Toc189680569)

[CHƯƠNG 1 – MÔ HÌNH MARKET-BASKET 7](#_Toc189680570)

[1.1 Định nghĩa về tập mục phổ biến 7](#_Toc189680571)

[1.1.1 Mô hình Market-Basket 7](#_Toc189680572)

[1.1.2 Định nghĩa về tập mục phổ biến 7](#_Toc189680573)

[1.2 Ứng dụng của tập mục phổ biến 8](#_Toc189680574)

[1.2.1 Phân tích giỏ hàng thị trường: 8](#_Toc189680575)

[1.2.2 Ví dụ “Diapers and Beer”: 9](#_Toc189680576)

[1.2.3 Các ứng dụng khác của khai thác tập phổ biến 9](#_Toc189680577)

[1.3 Luật kết hợp 10](#_Toc189680578)

[1.3.1 Định nghĩa 10](#_Toc189680579)

[1.3.2 Confidence 10](#_Toc189680580)

[1.3.3 Interest 10](#_Toc189680581)

[1.4 Tìm luật kết hợp có độ tin cậy cao 10](#_Toc189680582)

[CHƯƠNG 2 – MARKET-BASKET VÀ GIẢI THUẬT A – PRIORI 12](#_Toc189680583)

[2.1 Biểu diễn dữ liệu Market-Basket 12](#_Toc189680584)

[2.2 Dùng bộ nhớ chính để đếm tập mục 12](#_Toc189680585)

[2.2.1 Triangular-Matrix Method 13](#_Toc189680586)

[2.2.1.1 Tổng quan 13](#_Toc189680587)

[2.2.1.2 Ví dụ 13](#_Toc189680588)

[2.2.2 Triples Method 14](#_Toc189680589)

[2.2.2.1 Tổng quan 14](#_Toc189680590)

[2.2.2.2 Ví dụ 14](#_Toc189680591)

[2.2.3 Khi nào nên dùng 16](#_Toc189680592)

[2.3 Tính đơn điệu của tập mục 16](#_Toc189680593)

[2.4 Sự khó khăn trong việc đếm các cặp (Tyranny of counting pairs) 16](#_Toc189680594)

[2.5 Giải thuật A-Priori 17](#_Toc189680595)

[2.5.1 Mục tiêu 17](#_Toc189680596)

[2.5.2 Quá trình của A-Priori 17](#_Toc189680597)

[2.5.2.1 Lần quét đầu tiên (First Pass) 17](#_Toc189680598)

[2.5.2.2 Lần quét thứ hai (Second Pass) 17](#_Toc189680599)

[2.6 A-Priori cho tất cả tập mục phổ biến 18](#_Toc189680600)

[2.6.1 Ý tưởng chung 18](#_Toc189680601)

[2.6.2 Chi tiết thuật toán theo từng bước 19](#_Toc189680602)

[CHƯƠNG 3 – XỬ LÝ CÁC TẬP DỮ LIỆU LỚN TRONG BỘ NHỚ CHÍNH 21](#_Toc189680603)

[3.1 Giải thuật PCY (Parl, Chen và Yu) 21](#_Toc189680604)

[3.1.1 Vấn đề với A-Priori và bộ nhớ chín 21](#_Toc189680605)

[3.1.2 Giải pháp của thuật toán PCY 21](#_Toc189680606)

[3.1.3 Phân tích 22](#_Toc189680607)

[3.1.4 Lưu ý và hạn chế 23](#_Toc189680608)

[3.2 Giải thuật Multistage 23](#_Toc189680609)

[3.2.1 Lượt quét thứ nhất 23](#_Toc189680610)

[3.2.2 Lượt quét thứ hai 23](#_Toc189680611)

[3.2.3 Lượt quét thứ ba: 24](#_Toc189680612)

[3.3 Giải thuật Multihash 24](#_Toc189680613)

[3.3.1 Lần quét thứ nhất 24](#_Toc189680614)

[3.3.2 Lần quét thứ hai 24](#_Toc189680615)

[3.3.3 Lần quét thứ ba 25](#_Toc189680616)

[3.4 So sánh 3 giải thuật 25](#_Toc189680617)

[3.4.1 Giải thuật PCY: 25](#_Toc189680618)

[3.4.2 Giải thuật Multistage: 25](#_Toc189680619)

[3.4.3 Giải thuật Multihash: 26](#_Toc189680620)

[3.4.4 So sánh tổng quan: 26](#_Toc189680621)

[CHƯƠNG 4 – GIẢI THUẬT LIMIT-PASS 27](#_Toc189680622)

[4.1 Giải thuật Sample, Randomized 27](#_Toc189680623)

[4.1.1 Cách tiếp cận 27](#_Toc189680624)

[4.1.2 Kết luận 28](#_Toc189680625)

[4.2 Tránh lỗi trong giải thuật Sampling 28](#_Toc189680626)

[4.2.1 Tránh false positives 29](#_Toc189680627)

[4.2.2 Giảm thiểu false negatives 29](#_Toc189680628)

[4.3 Giải thuật của Savasere, Omiecinski, và Navathe 29](#_Toc189680629)

[4.4 Giải thuật SON và MapReduce 30](#_Toc189680630)

[4.4.1 Hàm Map đầu tiên 30](#_Toc189680631)

[4.4.2 Hàm Reduce đầu tiên 30](#_Toc189680632)

[4.4.3 Hàm Map thứ hai 30](#_Toc189680633)

[4.4.4 Hàm Reduce thứ hai 30](#_Toc189680634)

[4.5 Giải thuật Toivonen 31](#_Toc189680635)

[4.5.1 Chọn mẫu ngẫu nhiên 31](#_Toc189680636)

[4.5.1 Xây dựng biên âm 31](#_Toc189680637)

[4.5.3 Quét toàn bộ dữ liệu 31](#_Toc189680638)

[4.6 Tại sao giải thuật của Toivonen lại hiệu quả? 32](#_Toc189680639)

[CHƯƠNG 5 – ĐẾM CÁC TẬP MỤC PHỔ BIẾN TRONG MỘT LUỒNG 34](#_Toc189680640)

[5.1 Phương pháp lấy mẫu (sample) cho luồng 34](#_Toc189680641)

[5.1.1 Thu thập mẫu dữ liệu: 34](#_Toc189680642)

[5.1.2 Phân tích tập mẫu: 34](#_Toc189680643)

[5.1.3 Cập nhật kết quả: 34](#_Toc189680644)

[5.1.4 Ưu điểm của phương pháp 35](#_Toc189680645)

[5.2 Các tập mục phổ biến trong Decaying Window 35](#_Toc189680646)

[5.2.1 Nguyên lí hoạt động 35](#_Toc189680647)

[5.2.2 Cách phát hiện tập mục phổ biến 36](#_Toc189680648)

[5.2.3 Ví dụ 37](#_Toc189680649)

[5.3 Phương pháp Hybrid 37](#_Toc189680650)

[5.3.1 Tổng quan về phương pháp Hybrid Methods 37](#_Toc189680651)

[5.3.2 Cách tính điểm số ban dầu của tập mục 38](#_Toc189680652)

[5.3.3 Cách xử lí tập mục mới xuất hiện 38](#_Toc189680653)

[CHƯƠNG 6 – SỰ PHÁT TRIỂN CỦA TẬP MỤC PHỔ BIẾN – TẬP MỤC HỮU ÍCH VÀ GIỚI THIỆU CHUNG VỀ MỘT SỐ GIẢI THUẬT TÌM TẬP MỤC HỮU ÍCH HIỆN NAY 40](#_Toc189680654)

[6.1 Giới thiệu về High utility itemsets 40](#_Toc189680655)

[6.2 So sánh Frequent itemsets và High utility itemsets 41](#_Toc189680656)

[6.3 Một số giải thuật tìm top K high utility itemsets 42](#_Toc189680657)

[6.3.1 EHMIN 42](#_Toc189680658)

[6.3.2 EMHUN 45](#_Toc189680659)

**DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT**

**CÁC KÝ HIỆU**

Không có

**CÁC CHỮ VIẾT TẮT**

FI (Frequent itemsets): Các tập mục phổ biến

HUI (High utility itemsets): Các tập mục hữu ích

EHMIN (Efficient High-utility pattern MIning with Negative unit profits): Khai thác mẫu có lợi ích cao hiệu quả với lợi nhuận đơn vị âm

EMHUN (An Efficient method for Mining High-utility itemsets from Unstable Negative profit databases): Phương pháp hiệu quả để khai thác tập mục có lợi ích cao từ cơ sở dữ liệu có lợi nhuận âm không ổn định

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 6. 1 Cấu trúc EHMIN-list 43](#_Toc189680212)

[Hình 6. 2 Tổng quan về quá trình hoạt động của EHMIN 44](#_Toc189680213)

**DANH MỤC BẢNG**

[Bảng 2. 1 Bảng tính Triangular-Matrix method 14](#_Toc189680234)

[Bảng 2. 2 Ví dụ về giỏ hàng và sản phẩm 14](#_Toc189680235)

[Bảng 2. 3 Số lần xuất hiện của các cặp item 15](#_Toc189680236)

[Bảng 2. 4 Kết quả bảng Triples method 15](#_Toc189680237)

[Bảng 2. 5 Bảng item và số lần xuất hiện 19](#_Toc189680238)

[Bảng 2. 6 Số cặp item và số lần xuất hiện 19](#_Toc189680239)

[Bảng 2. 7 Bộ ba item và số lần xuất hiện 20](#_Toc189680240)

[Bảng 6. 1 Bảng so sánh Frequent itemsets và High utility itemsets 41](#_Toc189680262)

CHƯƠNG 1 – MÔ HÌNH MARKET-BASKET

* 1. Định nghĩa về tập mục phổ biến
     1. Mô hình Market-Basket

Mô hình Market-Basket mô tả mối quan hệ "nhiều-nhiều" giữa hai loại đối tượng:

* Items (Mặt hàng): Các sản phẩm hoặc phần tử riêng lẻ (ví dụ: từ khóa, sản phẩm trong siêu thị).
* Baskets (Giỏ hàng/Giao dịch): Một tập hợp các mặt hàng xuất hiện cùng nhau. Mỗi giỏ hàng thường có số lượng mặt hàng nhỏ so với tổng số mặt hàng có sẵn.

Đặc điểm chính:

* Số lượng giỏ hàng rất lớn (ví dụ: hàng triệu giao dịch), thường vượt quá khả năng lưu trữ trong bộ nhớ chính.
* Dữ liệu được lưu trữ dưới dạng tệp, mỗi dòng biểu diễn một giỏ hàng.
  + 1. Định nghĩa về tập mục phổ biến

Support (Độ hỗ trợ): Số lượng giỏ hàng chứa tập hợp mặt hàng.

Ngưỡng Support (s): Giá trị tối thiểu để một tập hợp được coi là "phổ biến".

Tập phổ biến: Tập hợp I có support ≥ s.

Ví dụ:

Nếu *s*=3, một tập hợp xuất hiện trong 3 giỏ hàng trở lên được xem là phổ biến.

Giả sử ta có:

**Dữ liệu Giỏ Hàng** (Mỗi giỏ là một tập các từ khóa không trùng lặp):

Giỏ 1: {a, b, c}

Giỏ 2: {a, b}

Giỏ 3: {a, c}

Giỏ 4: {b, c}

Ngưỡng Support: s=2.

Ta tìm tập phổ biến như sau:

**Bước 1: Tập Đơn (Singleton)**

{a}: Xuất hiện ở Giỏ 1, Giỏ 2, Giỏ 3 → Support = 3 (phổ biến)

{b}: Xuất hiện ở Giỏ 1, Giỏ 2, Giỏ 4 → Support = 3 (phổ biến)

{c}: Xuất hiện ở Giỏ 1, Giỏ 3, Giỏ 4 → Support = 3 (phổ biến)

→ **Tập phổ biến đơn:** {a}, {b}, {c}

**Bước 2: Tập Đôi (Doubleton)**

Chỉ xét các cặp với các phần tử đều là phổ biến:

{a, b}: Xuất hiện ở Giỏ 1, Giỏ 2 → Support = 2 (phổ biến)

{a, c}: Xuất hiện ở Giỏ 1, Giỏ 3 → Support = 2 (phổ biến)

{b, c}: Xuất hiện ở Giỏ 1, Giỏ 4 → Support = 2 (phổ biến)

→ **Tập phổ biến đôi:** {a, b}, {a, c}, {b, c}

**Bước 3: Tập Ba (Triple)**

{a, b, c}: Xuất hiện chỉ ở Giỏ 1 → Support = 1 (không phổ biến)

Đây là một ví dụ ngắn gọn minh họa cách xác định tập phổ biến theo ngưỡng support s=2

* 1. Ứng dụng của tập mục phổ biến
     1. Phân tích giỏ hàng thị trường:

Ban đầu, mô hình giỏ hàng được áp dụng trong việc phân tích các giỏ hàng thật tại siêu thị và cửa hàng chuỗi. Mỗi giỏ hàng ghi lại các sản phẩm mà khách hàng mua cùng nhau. Bằng cách tìm các tập hợp sản phẩm xuất hiện đồng thời với tần suất cao (support), ta có thể nhận ra các mối liên hệ giữa các mặt hàng. Ví dụ, ngoài việc phát hiện ra rằng bánh mì và sữa (bread và milk) là những sản phẩm phổ biến riêng lẻ, phân tích còn giúp nhận diện các cặp sản phẩm như hot dogs và mustard – những cặp mà sự xuất hiện đồng thời cao hơn so với kỳ vọng nếu khách hàng mua độc lập. Điều này mở ra cơ hội cho các chiến lược marketing như quảng cáo khuyến mãi cho món này và điều chỉnh giá cho món kia nhằm tối đa hóa lợi nhuận.

* + 1. Ví dụ “Diapers and Beer”:

Một ứng dụng nổi tiếng của phương pháp này là phát hiện ra mối liên hệ bất ngờ giữa tã lót (diapers) và bia (beer). Mặc dù hai sản phẩm này về lý thuyết không liên quan, dữ liệu cho thấy khách hàng mua tã có xu hướng mua bia. Lý giải cho hiện tượng này là người mua tã thường có trẻ nhỏ ở nhà, và thay vì ra ngoài quán bar, họ thường mang bia về nhà. Từ đó, nhà bán lẻ có thể áp dụng các chiến lược marketing khéo léo.

* + 1. Các ứng dụng khác của khai thác tập phổ biến
* **Liên kết khái niệm trong văn bản:** Xét mỗi từ là một “mục” và mỗi tài liệu (như trang web, blog hay tweet) là một “giỏ hàng”. Khi tìm các tập từ khóa xuất hiện cùng nhau nhiều lần, ta có thể nhận diện các khái niệm liên quan. Tuy nhiên, cần loại bỏ những từ thông dụng (stop words) để không bị chi phối bởi các từ phổ biến không mang ý nghĩa đặc biệt.
* **Phát hiện đạo văn:** Ở đây, “mục” là tài liệu và “giỏ hàng” là các câu trong tài liệu. Nếu hai tài liệu chia sẻ nhiều câu chung, đó có thể là dấu hiệu của việc sao chép nội dung (đạo văn).
* **Phân tích biomarkers trong y tế:** Mỗi giỏ hàng là hồ sơ của một bệnh nhân, chứa thông tin về các biomarkers (như gen hay protein máu) cùng với tiền sử bệnh. Nếu một tập phổ biến bao gồm một bệnh kèm theo một hoặc nhiều biomarkers, điều này có thể gợi ý mối liên hệ giữa chúng và hỗ trợ việc xây dựng các phương pháp chẩn đoán mới.
  1. Luật kết hợp
     1. Định nghĩa

Luật kết hợp có dạng “I → j”, trong đó I là một tập hợp các mặt hàng và j là một mặt hàng đơn lẻ. Ý nghĩa của luật này là nếu trong một giỏ hàng có chứa toàn bộ các mặt hàng của tập I, thì khả năng cao giỏ hàng đó cũng sẽ chứa mặt hàng j.

* + 1. Confidence

Confidence của luật I → j được tính bằng tỷ lệ giữa số giỏ hàng chứa tập I ∪ {j} với số giỏ hàng chứa tập I. Nói cách khác, nó cho biết phần trăm giỏ hàng có chứa I mà cũng có mặt hàng j.

Ví dụ: Nếu {a, b} xuất hiện trong 5 giỏ hàng và trong số đó “c” chỉ có trong 3 giỏ hàng, thì confidence của luật {a, b} → c là 3/5.

* + 1. Interest

Để đánh giá xem mối liên hệ trong luật có thật sự đặc biệt hay không, ta tính “interest” bằng hiệu số giữa confidence của luật và tỷ lệ xuất hiện của mặt hàng j trên tổng số giỏ hàng.

* Nếu I không ảnh hưởng đến j, thì tỉ lệ xuất hiện của j trong các giỏ hàng có chứa I sẽ bằng với tỷ lệ xuất hiện của j trên toàn bộ dữ liệu, tức interest = 0.
* Luật có interest dương (hoặc âm) cao cho thấy sự hiện diện của I có tác động đáng kể đến khả năng xuất hiện của j.

*Ví dụ:* Trong trường hợp “diapers and beer”, luật {diapers} → beer có interest cao nếu tỷ lệ người mua tã lót mua thêm bia vượt xa tỷ lệ mua bia chung của toàn bộ khách hàng. Ngược lại, luật {coke} → pepsi có thể có interest âm, do người tiêu dùng đã uống Coke thì không việc gì phải mua thêm Pepsi, cho thấy việc mua Coke làm giảm khả năng mua Pepsi.

* 1. Tìm luật kết hợp có độ tin cậy cao

**Giả thiết ban đầu:**

Giả sử ta đã có tập hợp các tập phổ biến (frequent itemsets) có support ≥ ngưỡng s. Những tập này được thu thập từ các giỏ hàng (baskets) và có số lượng xuất hiện đủ lớn để được xem là đáng chú ý.

**Mục tiêu của luật kết hợp:**

Ta muốn tìm các luật có dạng I → j với hai đặc điểm chính:

* Support của I (hoặc I ∪ {j}) phải cao: Để luật áp dụng cho một phần đáng kể các giỏ hàng (trong thực tế, thường khoảng 1% trở lên đối với các cửa hàng).
* Độ tin cậy (confidence) phải cao: Thường đặt ngưỡng tối thiểu khoảng 50% để đảm bảo rằng khi có I xuất hiện, khả năng j cũng xuất hiện là đáng kể.

**Quy trình tìm luật kết hợp:**

* Với mỗi tập phổ biến J (gồm n mặt hàng), ta có thể tạo ra tối đa n luật dưới dạng J − {j} → j (với mỗi j thuộc J).
* Vì J là tập phổ biến, nên các tập con như J − {j} cũng có support cao. Do đó, độ tin cậy của luật được tính là tỉ lệ giữa support(J) và support(J − {j}).
* Chỉ những luật mà cả support và confidence đều vượt qua ngưỡng đã đặt mới được coi là hữu ích.

**Vấn đề về số lượng luật:**

Nếu thu được quá nhiều luật kết hợp (ví dụ, hàng triệu luật đáp ứng yêu cầu), thì việc xử lý, phân tích và áp dụng thực tế là không khả thi. Vì mỗi luật cần được kiểm tra và triển khai, nên thường điều chỉnh ngưỡng support sao cho số lượng tập phổ biến (và do đó số luật kết hợp) ở mức chấp nhận được. Điều này không chỉ giúp việc ứng dụng trở nên thực tiễn mà còn có những tác động quan trọng đến hiệu quả của các thuật toán tìm kiếm tập phổ biến.

CHƯƠNG 2 – MARKET-BASKET VÀ GIẢI THUẬT A – PRIORI

2.1 Biểu diễn dữ liệu Market-Basket

Thông thường, dữ liệu giỏ hàng (Market-Basket Data) được lưu dưới dạng tệp, trong đó mỗi giỏ hàng chứa một tập hợp các mặt hàng được biểu diễn bằng số nguyên (các số nguyên này chính là các sản phẩm của cửa hàng đã được mã hóa/băm nhằm xử lý dữ liệu nhanh cũng như tiết kiệm bộ nhớ hơn, đồng thời cũng vì lý do bảo mật). Dữ liệu trên có thể được xử lý trên một máy duy nhất hoặc phân tán bằng MapReduce. Ví dụ ta có tệp bắt đầu như sau:{1,2,3}{4,5,6,7}..., ta sẽ hiểu rằng{} biểu thị một giỏ hàng và các mặt hàng được phân tách bằng dấu phẩy. Vậy, giỏ hàng đầu tiên chứa các mặt hàng 1, 2, 3, giỏ hàng thứ hai chứa 4, 5, 6, 7.

Vì kích thước tệp lớn không thể tải toàn bộ vào bộ nhớ, nên chi phí chính là thời gian đọc dữ liệu từ đĩa. Khi dữ liệu được nạp vào bộ nhớ, có thể tạo ra các tập hợp con (itemsets) có kích thước k. Đối với giỏ hàng nhỏ, việc tạo cặp mục (pairs) diễn ra nhanh chóng. Tuy nhiên, khi kích thước tập con tăng, thời gian xử lý tăng theo cấp số nhân () để tạo tất cả các tập con có kích thước k cho một giỏ hàng có n phần tử, đây chính là yếu tố tốn thời gian nhất.

2.2 Dùng bộ nhớ chính để đếm tập mục

Khi xử lý tập hợp mục phổ biến (frequent itemset), ta cần lưu nhiều bộ đếm khi quét dữ liệu. Nếu không đủ bộ nhớ để lưu trữ các bộ đếm này, việc cập nhật từng bộ đếm có thể yêu cầu tải dữ liệu từ đĩa, làm giảm hiệu suất nghiêm trọng.

Ví dụ: Nếu có n mặt hàng, cần lưu trữ n²/2 bộ đếm, tương đương với 2n² byte. Với bộ nhớ 2GB (2³¹ byte), thuật toán chỉ có thể xử lý tối đa 33,000 mặt hàng.

Để tiết kiệm bộ nhớ, các mặt hàng nên được ánh xạ thành số nguyên liên tiếp từ 1 đến n thay vì lưu dưới dạng chuỗi (vd: "bread"). Quá trình này yêu cầu sử dụng bảng băm (hash table) để chuyển đổi từ chuỗi sang số nguyên, giúp truy xuất dữ liệu nhanh hơn.

Có 2 phương pháp chính nhằm thực hiện yêu cầu trên

2.2.1 Triangular-Matrix Method

2.2.1.1 Tổng quan

Mục tiêu: Tiết kiệm bộ nhớ khi lưu đếm các cặp {i, j}.

Cách hoạt động:

* Chỉ lưu trữ các cặp có i < j thay vì sử dụng mảng 2D kích thước n².
* Sử dụng một mảng 1D để lưu các cặp theo thứ tự từ điển (lexicographic order), giúp tối ưu bộ nhớ hơn.

Công thức để xác định vị trí lưu của cặp {i, j}:

Ưu điểm: Tiết kiệm bộ nhớ hơn mảng 2D.

Nhược điểm: Vẫn cần lưu trữ toàn bộ các cặp, kể cả khi có nhiều cặp có giá trị 0.

* + - 1. Ví dụ

Giả sử ta có n = 5 mục, được đánh số từ 1 đến 5.

Các cặp có thể có (i < j):

{1,2}, {1,3}, {1,4}, {1,5}

{2,3}, {2,4}, {2,5}

{3,4}, {3,5}

{4,5}

Tổng cộng có (n × (n - 1)) / 2 = 10 cặp.

Tiếp theo áp dụng công thức, ta xác định vị trí các lưu trữ số lần xuất hiện của các cặp trong mảng 1 chiều như bảng dưới

Bảng 2. Bảng tính Triangular-Matrix method

|  |  |  |
| --- | --- | --- |
| Cặp {i, j} | Áp dụng công thức | Vị trí lưu trữ |
| {1, 2} | |  |  | | --- | --- | | (1 - 1) \* (5 - 1 / 2) + (2 - 1) = 1 |  | | A[1] |
| {1, 3} | |  |  |  | | --- | --- | --- | |  | (1 - 1) \* (5 - 1 / 2) + (3 - 1) = 2 |  | | A[2] |
| {1, 4} | |  |  |  | | --- | --- | --- | |  | (1 - 1) \* (5 - 1 / 2) + (4 - 1) = 3 |  | | A[3] |
| {1, 5} | |  |  |  | | --- | --- | --- | |  | (1 - 1) \* (5 - 1 / 2) + (5 - 1) = 4 |  | | A[4] |
| {2, 3} | |  |  |  | | --- | --- | --- | |  | (2 - 1) \* (5 - 2 / 2) + (3 - 2) = 5 |  | | A[5] |
| {2, 4} | |  |  |  | | --- | --- | --- | |  | (2 - 1) \* (5 - 2 / 2) + (4 - 2) = 6 |  | | A[6] |
| {2, 5} | |  |  |  | | --- | --- | --- | |  | (2 - 1) \* (5 - 2 / 2) + (5 - 2) = 7 |  | | A[7] |
| {3, 4} | |  |  |  | | --- | --- | --- | |  | (3 - 1) \* (5 - 3 / 2) + (4 - 3) = 8 |  | | A[8] |
| {3, 5} | |  |  |  | | --- | --- | --- | |  | (3 - 1) \* (5 - 3 / 2) + (5 - 3) = 9 |  | | A[9] |
| {4, 5} | |  | | --- | | (4 - 1) \* (5 - 4 / 2) + (5 - 4) = 10 | | A[10] |

Từ đây mảng lưu trữ chỉ cần kích thước 10 thay vì 5 × 5 = 25 phần tử. Khi cần đếm số lần xuất hiện của {2,4}, ta chỉ cần truy cập A[6] thay vì tìm trong mảng 2D.

* + 1. Triples Method

2.2.2.1 Tổng quan

Mục tiêu: Chỉ lưu trữ các cặp thực sự xuất hiện trong dữ liệu, tránh lưu các cặp có giá trị đếm bằng 0.

Cách hoạt động:

* Mỗi cặp {i, j} có một giá trị đếm c được lưu dưới dạng [i, j, c].
* Dùng hash table để tìm kiếm nhanh cặp {i, j} và cập nhật giá trị c.

Ưu điểm: Tiết kiệm bộ nhớ khi dữ liệu thưa thớt (nhiều cặp không xuất hiện).

Nhược điểm: Tốn nhiều bộ nhớ hơn nếu có quá nhiều cặp xuất hiện vì phải lưu cả i, j, c, thay vì chỉ một giá trị như phương pháp Triangular-Matrix.

* + - 1. Ví dụ

Giả sử ta có n = 5 mục, được đánh số từ 1 đến 5, và các giỏ hàng chứa dữ liệu như sau

Bảng 2. Ví dụ về giỏ hàng và sản phẩm

|  |  |
| --- | --- |
| Giỏ hàng | Sản phẩm |
| 1 | {1,2,3} |
| 2 | {2,4} |
| 3 | {1,3,5} |
| 4 | {2,3,4} |

Ta đếm số lần xuất hiện các cặp như sau

Bảng 2. Số lần xuất hiện của các cặp item

|  |  |
| --- | --- |
| Cặp {i, j} | Số lần xuất hiện (c) |
| {1, 2} | 1 |
| {1, 3} | 2 |
| {1, 5} | 1 |
| {2, 3} | 2 |
| {2, 4} | 2 |
| {3, 4} | 1 |
| {3, 5} | 1 |

Khi đó ta lưu trữ dữ liệu [i, j, c] như sau

Bảng 2. Kết quả bảng Triples method

|  |  |  |
| --- | --- | --- |
| i | j | c |
| 1 | 2 | 1 |
| 1 | 3 | 2 |
| 1 | 5 | 1 |
| 2 | 3 | 2 |
| 2 | 4 | 2 |
| 3 | 4 | 1 |
| 3 | 5 | 1 |

Khi cần tìm số lần xuất hiện của cặp {2,3}, ta chỉ cần truy vấn hash table và lấy giá trị c = 2. Nếu cặp {1,4} không tồn tại trong danh sách, ta biết ngay số lần xuất hiện của nó là 0, không cần lưu trữ.

* + 1. Khi nào nên dùng

Nếu số lượng cặp xuất hiện nhiều (≥ 1/3 số cặp có thể có), dùng Triangular-Matrix sẽ hiệu quả hơn.

Nếu số lượng cặp xuất hiện ít (< 1/3 số cặp có thể có), dùng Triples Method sẽ tiết kiệm bộ nhớ hơn.

* 1. Tính đơn điệu của tập mục

Nếu một tập hợp item 𝐼 là phổ biến (frequent), thì mọi tập con của 𝐼 cũng sẽ là phổ biến. Điều này xảy ra vì mọi giỏ hàng chứa tất cả các item trong 𝐼 sẽ chứa tất cả các item trong bất kỳ tập con nào của 𝐼, và vì vậy, tần suất của tập con 𝐽 ít nhất bằng tần suất của 𝐼.

Điều này có ý nghĩa quan trọng trong tối ưu hóa thuật toán, ví dụ như A-Priori, bằng cách chỉ cần liệt kê các itemset tối đa. Một itemset là tối đa nếu không có tập superset nào phổ biến.

Ví dụ: Với một ngưỡng hỗ trợ 𝑠 = 3, ta có các itemset phổ biến sau:

* Các singleton phổ biến: {cat}, {dog}, {a}, {and}, {training}
* Các cặp phổ biến: {dog, a}, {dog, and}, {dog, cat}, {cat, a}, {cat, and}
* Cặp tối đa: {training}, {dog, a}, {dog, and}, {cat, and}, {dog, cat, a}

Trong đó, {dog, cat, a} là itemset tối đa, vì không có superset nào phổ biến hơn nó.

* 1. Sự khó khăn trong việc đếm các cặp (Tyranny of counting pairs)

Trong thực tế, việc đếm các cặp (pairs) là yêu cầu bộ nhớ chính (main memory) lớn nhất vì số lượng các itemset cặp thường rất lớn. Số lượng các item (mặc dù có thể rất lớn) hiếm khi đủ lớn để không thể đếm tất cả các singleton trong bộ nhớ chính cùng một lúc. Đối với các tập hợp lớn hơn như triples (ba phần tử), quadruples (bốn phần tử), và các tập hợp tương tự, ta thường đặt ngưỡng hỗ trợ (support threshold) đủ cao để chỉ có một số ít tập hợp đủ điều kiện là phổ biến. Nên tất nhiên chúng cũng chiếm ít tài nguyên hơn. Khi ta đếm được các cặp, ta có thể biết được các triple, nhưng khi ta đếm được các triple, chưa chắc ta có đủ các cặp vì có thể cặp không nằm trong triple.

Vì thế, các thuật toán như A-Priori tập trung vào việc tránh phải đếm tất cả các triples hay các set lớn hơn, giúp giảm thiểu số lượng các bộ đếm cần phải xử lý. Ta chỉ cần tập trung vào việc đếm các cặp thôi.

* 1. Giải thuật A-Priori
     1. Mục tiêu

Thuật toán A-Priori được thiết kế để tìm kiếm các itemset phổ biến trong một tập hợp các giỏ hàng (baskets). Mục tiêu là xác định các item hoặc tập hợp các item có mặt trong một số lượng giỏ hàng đủ lớn (đạt ngưỡng hỗ trợ, support threshold).

* + 1. Quá trình của A-Priori

2.5.2.1 Lần quét đầu tiên (First Pass)

Trong lần quét đầu tiên, thuật toán tính toán tần suất xuất hiện của từng item (singleton). Những item có tần suất xuất hiện đủ cao sẽ được coi là phổ biến (frequent).

Ngưỡng hỗ trợ (s) sẽ được đặt đủ cao để chỉ có một số ít các itemset được coi là phổ biến. Ví dụ, nếu ngưỡng là 1%, tức là chỉ những item xuất hiện trong ít nhất 1% tổng số giỏ hàng mới được coi là phổ biến.

Sau lần quét đầu tiên, thuật toán tạo một bảng "frequent-items table", trong đó đánh dấu các item phổ biến với chỉ số mới từ 1 đến m, nơi m là số lượng item phổ biến. Những item không phổ biến sẽ có giá trị 0.

* + - 1. Lần quét thứ hai (Second Pass)

Trong lần quét thứ hai, thuật toán sẽ tập trung vào việc đếm tất cả các cặp item (pairs) từ các item phổ biến. Một cặp {i, j} chỉ có thể là phổ biến nếu cả hai item i và j đều là phổ biến.

Để tối ưu bộ nhớ, nếu sử dụng phương pháp triangular matrix, không cần phải lưu trữ tất cả các cặp mà chỉ lưu trữ các cặp của các item phổ biến, giúp tiết kiệm không gian đáng kể (2m² byte thay vì 2n² byte).

Nếu sử dụng phương pháp triples, chỉ cần đếm các cặp phổ biến mà xuất hiện ít nhất một lần trong giỏ hàng.

***\*Cách thực hiện lần quét thứ hai***

- Kiểm tra item phổ biến: Với mỗi giỏ hàng, thuật toán sẽ kiểm tra bảng "frequent-items table" để xác định những item nào trong giỏ hàng đó là phổ biến.

- Tạo các cặp item phổ biến: Dùng một vòng lặp đôi (double loop) để tạo tất cả các cặp item phổ biến từ giỏ hàng.

- Cập nhật tần suất: Với mỗi cặp item, thuật toán sẽ tăng một đơn vị vào tần suất của cặp đó trong cấu trúc dữ liệu lưu trữ tần suất (counts).

Sau khi hoàn thành lần quét thứ hai, thuật toán sẽ kiểm tra cấu trúc dữ liệu để xác định những cặp nào là phổ biến (frequent pairs).

* 1. A-Priori cho tất cả tập mục phổ biến

2.6.1 Ý tưởng chung

Bắt đầu với tập đơn (L1):

* Xác định tập hợp tất cả các item đơn lẻ (C1) từ dữ liệu giao dịch.
* Đếm số lần xuất hiện của từng item trong toàn bộ tập dữ liệu.
* Giữ lại các item có tần suất xuất hiện ≥ ngưỡng hỗ trợ s → tập hợp này gọi là L1.

Tạo tập ứng viên (Ck) và kiểm tra tính phổ biến (Lk):

* Sử dụng tập L(k-1) để tạo danh sách ứng viên Ck (tập hợp các item có thể phổ biến).
* Đếm số lần xuất hiện của từng tập hợp trong Ck trong toàn bộ tập dữ liệu.
* Các tập mục có số lần xuất hiện ≥ ngưỡng hỗ trợ s tạo thành tập Lk.
* Lặp lại quá trình này cho đến khi không còn tập phổ biến nào mới được tìm thấy.

2.6.2 Chi tiết thuật toán theo từng bước

Bước 1: Khởi tạo với C1 và L1

* C1: Danh sách tất cả các item đơn lẻ trong giao dịch.
* Đếm số lần xuất hiện của từng item và lọc ra những item có hỗ trợ ≥ s để tạo tập L1.

Ví dụ:

Bảng 2. Bảng item và số lần xuất hiện

|  |  |
| --- | --- |
| Item | Số lần xuất hiện |
| A | 5 |
| B | 3 |
| C | 6 |
| D | 2 |

Giả sử ngưỡng hỗ trợ s = 3, ta thu được L1 = {A, B, C}.

Bước 2: Xây dựng tập ứng viên C2 và tìm L2

* Tạo các cặp C2 bằng cách kết hợp các item trong L1.
* Đếm số lần xuất hiện của từng cặp trong toàn bộ giao dịch.
* Lọc ra những cặp có số lần xuất hiện ≥ s để tạo L2.

Ví dụ:

Bảng 2. Số cặp item và số lần xuất hiện

|  |  |
| --- | --- |
| Cặp item | Số lần xuất hiện |
| {A, B} | 3 |
| {A, C} | 4 |
| {B, C} | 2 |

Với ngưỡng hỗ trợ s = 3, chỉ có L2 = {(A, B), (A, C)}.

Bước 3: Xây dựng tập ứng viên C3 và tìm L3

* Kết hợp các cặp trong L2 để tạo danh sách ứng viên C3.
* Một bộ ba (triple) hợp lệ khi hai cặp con của nó thuộc L2.
* Đếm số lần xuất hiện của từng bộ ba trong toàn bộ giao dịch.
* Lọc ra các bộ ba có số lần xuất hiện ≥ s để tạo L3.

Ví dụ:

Bảng 2. Bộ ba item và số lần xuất hiện

|  |  |
| --- | --- |
| Bộ ba item | Số lần xuất hiện |
| {A, B, C} | 3 |

L3 = {(A, B, C)} vì nó đạt điều kiện hỗ trợ ≥ s.

Bước 4: Tiếp tục cho đến khi không thể tạo thêm tập phổ biến

* Tiếp tục tạo C4, C5,... theo cùng nguyên tắc.
* Nếu không tìm thấy tập phổ biến nào nữa (Lk = ∅), thuật toán dừng.

***\*Ứng dụng tính chất Monotonicity***

Nếu không có tập Lk, thì tất cả tập có kích thước lớn hơn cũng sẽ không phổ biến. Nhờ đó, thuật toán không cần kiểm tra tất cả tổ hợp có thể có, giúp giảm số lượng tính toán đáng kể.

CHƯƠNG 3 – XỬ LÝ CÁC TẬP DỮ LIỆU LỚN TRONG BỘ NHỚ CHÍNH

3.1 Giải thuật PCY (Parl, Chen và Yu)

3.1.1 Vấn đề với A-Priori và bộ nhớ chín

Thuật toán A-Priori thực hiện hai lượt quét (passes) qua tập dữ liệu để đếm các mặt hàng đơn (singletons) và sau đó là các cặp ứng viên (C2).

Vấn đề: bước đếm các cặp ứng viên (C2) đòi hỏi bộ nhớ lớn vì phải lưu trữ tất cả các cặp có khả năng xảy ra. Khi tập dữ liệu rất lớn, nếu bộ nhớ chính không đủ, hệ thống sẽ phải “thrash” – tức là dữ liệu liên tục được chuyển giữa bộ nhớ chính và ổ đĩa, làm giảm hiệu năng xử lý.

3.1.2 Giải pháp của thuật toán PCY

**Tận Dụng Bộ Nhớ Dư Thừa ở Lượt Quét Đầu (Pass 1):**

* Trong lượt quét đầu tiên, bên cạnh việc đếm số lần xuất hiện của từng mặt hàng, ta có thể tận dụng không gian bộ nhớ dư thừa để đếm số cặp mặt hàng.
* Thay vì lưu trữ trực tiếp các cặp, PCY sử dụng một mảng số nguyên (hoạt động như bảng băm) để lưu trữ các bucket count.
* Mỗi cặp mặt hàng được hash vào một bucket cụ thể, và chỉ số của bucket tăng lên 1 mỗi khi một cặp rơi vào đó.

**Xác Định Các Bucket “Frequent”:**

* Sau khi quét xong, mỗi bucket có một giá trị đếm, thể hiện tổng số cặp đã được hash vào đó.
* Một bucket được xem là frequent nếu giá trị đếm của nó ≥ ngưỡng support s.
* **Quan trọng:** nếu một bucket là infrequent (đếm < s), thì bất kỳ cặp nào hash vào bucket đó cũng không thể là cặp phổ biến, dù các mặt hàng riêng lẻ có phổ biến hay không.

**Tóm Tắt Bảng Băm Thành Bitmap:**

* Trước lượt quét thứ hai, bảng băm được chuyển thành bitmap: mỗi bucket chỉ lưu một bit (1 nếu frequent, 0 nếu không).
* Việc này giúp tiết kiệm không gian bộ nhớ, vì thay vì lưu số nguyên (thường 4 byte), chỉ lưu một bit.

**Lượt Quét Thứ Hai (Pass 2) – Đếm Các Cặp Ứng Viên:**

* Ở lượt quét thứ hai, ta chỉ xem xét các cặp ứng viên thỏa mãn hai điều kiện:
  + Cả hai mặt hàng đều là frequent.
  + Cặp đó hash vào một bucket đã được xác định là frequent.

3.1.3 Phân tích

**Ví dụ về số liệu:**

* Giả sử dữ liệu có 1 tỷ giỏ hàng, mỗi giỏ chứa 10 mặt hàng. Tổng số cặp có thể là khoảng 109 ≈ 4.51010 cặp.
* Nếu bộ nhớ có thể lưu trữ khoảng 250 triệu buckets (mỗi bucket 4 byte), trung bình mỗi bucket nhận khoảng 180 đếm.

**Tác động của ngưỡng support:**

* Nếu ngưỡng support s thấp (ví dụ, khoảng 180), hầu hết các bucket có thể đạt yêu cầu frequent, khiến PCY không giảm được số lượng cặp ứng viên.
* Ngược lại, nếu s cao (ví dụ, 1000), đa số bucket sẽ là infrequent. Khi đó, chỉ một phần nhỏ các cặp (những hash vào bucket frequent) cần được đếm lại trong lượt quét thứ hai, giúp tiết kiệm bộ nhớ và tránh hiện tượng thrashing.

3.1.4 Lưu ý và hạn chế

**Bố trí dữ liệu không thuận lợi cho việc “nén”:**

* Trong A-Priori, các mặt hàng frequent có thể được đánh số lại từ 1 đến m, cho phép sử dụng ma trận tam giác để lưu trữ các cặp.
* Tuy nhiên, với PCY, các cặp ứng viên được loại bỏ dựa trên kết quả hash nên vị trí của chúng trong ma trận không đồng nhất, buộc phải sử dụng phương pháp “triples” để đếm, có thể làm tăng yêu cầu bộ nhớ.

**Hiệu quả của PCY phụ thuộc vào dữ liệu:**

* Nếu không đủ số lượng các bucket infrequent (tức hầu hết các bucket đều frequent), thì PCY không đem lại lợi ích so với A-Priori.
* Để PCY có hiệu quả, cần giảm được ít nhất khoảng 2/3 số cặp của các mặt hàng frequent.

3.2 Giải thuật Multistage

3.2.1 Lượt quét thứ nhất

* Đếm số lần xuất hiện của từng mặt hàng (item) để xác định các mặt hàng phổ biến.
* Sử dụng một bảng băm đầu tiên để đếm số lượng cặp mặt hàng bằng cách băm các cặp vào các bucket và tăng bộ đếm tương ứng.
* Sau lượt quét này, xác định các bucket phổ biến (các bucket có số đếm ≥ ngưỡng hỗ trợ) và tạo một bitmap từ bảng băm này, trong đó mỗi bit biểu thị một bucket (1 nếu bucket phổ biến, 0 nếu không).

3.2.2 Lượt quét thứ hai

* Thay vì đếm các cặp ứng viên như trong giải thuật PCY, giải thuật Multistage sử dụng bộ nhớ chính để tạo một bảng băm thứ hai với một hàm băm khác.
* Duyệt qua tập dữ liệu, băm các cặp mặt hàng thỏa mãn hai điều kiện:
  + Cả hai mặt hàng đều phổ biến.
  + Cặp này đã băm vào một bucket phổ biến trong bảng băm đầu tiên (dựa trên bitmap từ lượt quét thứ nhất).
* Tăng bộ đếm cho các bucket tương ứng trong bảng băm thứ hai.
* Sau lượt quét này, tạo một bitmap từ bảng băm thứ hai tương tự như ở bước trước.

3.2.3 Lượt quét thứ ba:

* Duyệt qua tập dữ liệu lần nữa để đếm số lần xuất hiện của các cặp ứng viên thỏa mãn ba điều kiện:
  + Cả hai mặt hàng đều phổ biến.
  + Cặp này băm vào một bucket phổ biến trong bảng băm thứ nhất.
  + Cặp này băm vào một bucket phổ biến trong bảng băm thứ hai.

Bằng cách sử dụng nhiều bảng băm và các hàm băm khác nhau, giải thuật Multistage giảm đáng kể số lượng cặp ứng viên cần xem xét, từ đó tiết kiệm bộ nhớ và tăng hiệu quả xử lý so với giải thuật PCY truyền thống.

3.3 Giải thuật Multihash

Giải thuật Multihash là một biến thể của giải thuật PCY, nhằm cải thiện hiệu quả trong việc tìm kiếm các cặp mục phổ biến bằng cách sử dụng nhiều bảng băm trong một lần quét dữ liệu. Điều này giúp giảm số lượng cặp ứng viên cần xem xét trong các bước tiếp theo.

3.3.1 Lần quét thứ nhất

* Ánh xạ tên các mục thành các số nguyên.
* Đếm số lần xuất hiện của từng mục để xác định các mục phổ biến.
* Sử dụng nhiều hàm băm (ví dụ: hai hàm băm khác nhau) để băm các cặp mục vào các bảng băm riêng biệt, mỗi bảng có số lượng ô nhỏ hơn so với bảng băm duy nhất trong PCY.

3.3.2 Lần quét thứ hai

Chuyển đổi mỗi bảng băm thành một bitmap, trong đó mỗi bit biểu thị liệu ô tương ứng có phải là ô phổ biến hay không (tức là số đếm của ô đó đạt ngưỡng hỗ trợ).

3.3.3 Lần quét thứ ba

* Xác định các cặp ứng viên cần đếm bằng cách kiểm tra:
  + Cả hai mục trong cặp đều là mục phổ biến.
  + Cặp đó được băm vào các ô phổ biến trong tất cả các bảng băm (dựa trên các bitmap).
* Đếm số lần xuất hiện của các cặp ứng viên thỏa mãn các điều kiện trên để xác định các cặp mục phổ biến cuối cùng.

3.4 So sánh 3 giải thuật

Giải thuật PCY (Park, Chen và Yu), Multistage và Multihash đều nhằm mục đích tối ưu hóa quá trình tìm kiếm các tập mục phổ biến trong các tập dữ liệu lớn, nhưng chúng có những khác biệt quan trọng về cách tiếp cận và hiệu quả.

3.4.1 Giải thuật PCY:

**Phương pháp**: Trong lần duyệt đầu tiên, PCY sử dụng một hàm băm để ánh xạ các cặp mục vào các bucket trong bảng băm và đếm số lần xuất hiện của chúng. Sau đó, các bucket có số đếm vượt ngưỡng hỗ trợ được đánh dấu là "phổ biến".

**Ưu điểm**: Giảm số lượng cặp mục cần xem xét trong các lần duyệt tiếp theo bằng cách loại bỏ sớm các cặp không phổ biến.

**Hạn chế**: Hiệu quả phụ thuộc vào kích thước của bảng băm và hàm băm được sử dụng; nếu nhiều cặp mục được ánh xạ vào cùng một bucket (hiện tượng va chạm), có thể dẫn đến việc giữ lại các cặp không phổ biến.

3.4.2 Giải thuật Multistage:

**Phương pháp**: Mở rộng từ PCY bằng cách sử dụng nhiều bảng băm liên tiếp qua các lần duyệt. Sau mỗi lần duyệt, chỉ các cặp mục ánh xạ vào các bucket "phổ biến" mới được xem xét trong lần duyệt tiếp theo với một hàm băm khác.

**Ưu điểm**: Giảm thêm số lượng cặp mục cần xem xét bằng cách áp dụng nhiều hàm băm và bảng băm, giúp loại bỏ nhiều cặp không phổ biến hơn.

**Hạn chế**: Yêu cầu nhiều lần duyệt dữ liệu hơn, dẫn đến tăng thời gian xử lý và tài nguyên tính toán.

3.4.3 Giải thuật Multihash:

**Phương pháp**: Thay vì sử dụng nhiều bảng băm qua các lần duyệt như Multistage, Multihash sử dụng nhiều hàm băm và bảng băm đồng thời trong một lần duyệt duy nhất. Mỗi cặp mục được ánh xạ vào nhiều bảng băm khác nhau, và chỉ các cặp ánh xạ vào các bucket "phổ biến" trong tất cả các bảng băm mới được xem xét trong các bước tiếp theo.

**Ưu điểm**: Giảm số lượng cặp mục cần xem xét trong các lần duyệt tiếp theo mà không cần tăng số lần duyệt dữ liệu, tiết kiệm thời gian so với Multistage.

**Hạn chế**: Mỗi bảng băm có ít bucket hơn (do chia sẻ bộ nhớ), có thể dẫn đến tăng khả năng va chạm và giảm hiệu quả nếu không được cấu hình phù hợp.

3.4.4 So sánh tổng quan:

**Số lần duyệt dữ liệu**: PCY và Multihash yêu cầu ít lần duyệt hơn (thường là hai), trong khi Multistage yêu cầu nhiều lần duyệt hơn để đạt được mức độ lọc tương tự.

**Sử dụng bộ nhớ**: Multihash chia sẻ bộ nhớ giữa các bảng băm trong một lần duyệt, trong khi Multistage sử dụng bộ nhớ cho các bảng băm khác nhau qua các lần duyệt.

**Hiệu quả lọc**: Multistage có thể đạt được mức độ lọc cao hơn do sử dụng nhiều bảng băm liên tiếp, nhưng đánh đổi bằng việc tăng số lần duyệt dữ liệu. Multihash cố gắng đạt được hiệu quả tương tự trong ít lần duyệt hơn bằng cách sử dụng nhiều bảng băm đồng thời.

CHƯƠNG 4 – GIẢI THUẬT LIMIT-PASS

Thuật toán tìm kiếm itemset phổ biến (frequent itemsets) thường yêu cầu nhiều lượt quét (passes) qua toàn bộ dữ liệu. Mỗi lượt quét tương ứng với việc tính toán tần suất cho các itemset có kích thước khác nhau. Tuy nhiên, nếu bộ nhớ chính (main memory) không đủ lớn để chứa toàn bộ dữ liệu và không gian cần thiết để đếm các itemset phổ biến, thì việc phải thực hiện nhiều lượt quét sẽ trở thành một vấn đề lớn. Trong trường hợp đó, có thể cần thực hiện k lượt quét để tính toán chính xác tất cả các itemset phổ biến.

Tuy nhiên, trong nhiều ứng dụng, không phải lúc nào cũng cần phải phát hiện tất cả các itemset phổ biến. Ví dụ, nếu ta đang tìm kiếm những sản phẩm phổ biến được mua cùng nhau trong siêu thị, thì không cần phải tìm tất cả các itemset phổ biến. Thực tế, chỉ cần tìm ra phần lớn các itemset phổ biến là đủ.

Các thuật toán giới hạn lượt quét (Limited-Pass Algorithms) giải quyết vấn đề này bằng cách sử dụng ít lượt quét hơn.

4.1 Giải thuật Sample, Randomized

Ý tưởng của giải thuật này là thay vì sử dụng toàn bộ tệp dữ liệu giỏ hàng, chúng ta có thể chọn một mẫu ngẫu nhiên của các giỏ hàng và giả định rằng mẫu này đại diện cho toàn bộ dữ liệu. Tuy nhiên, khi làm vậy, chúng ta cần điều chỉnh ngưỡng hỗ trợ (support threshold) sao cho phù hợp với số lượng giỏ hàng trong mẫu.

4.1.1 Cách tiếp cận

Chọn mẫu ngẫu nhiên: Giả sử ngưỡng hỗ trợ cho toàn bộ tập dữ liệu là s. Nếu chọn một mẫu ngẫu nhiên của 1% giỏ hàng, thì ta cần tìm các itemset xuất hiện ít nhất s/100 lần trong mẫu.

Quy trình chọn mẫu:

* Đọc toàn bộ tập dữ liệu và đối với mỗi giỏ hàng, chọn giỏ hàng đó vào mẫu với một xác suất cố định p. Nếu có m giỏ hàng trong toàn bộ tập dữ liệu, thì cuối cùng ta sẽ có một mẫu có kích thước xấp xỉ pm giỏ hàng.
* Nếu giỏ hàng đã xuất hiện theo thứ tự ngẫu nhiên trong tệp, ta có thể không cần đọc toàn bộ tệp. Chỉ cần chọn pm giỏ hàng đầu tiên từ tệp là đủ. Hoặc nếu tệp nằm trong hệ thống tệp phân tán, ta có thể chọn ngẫu nhiên một số phân đoạn để làm mẫu.

Chạy thuật toán trên mẫu:

* Sau khi chọn mẫu, chúng ta sẽ sử dụng phần bộ nhớ chính để lưu trữ các giỏ hàng này. Phần bộ nhớ còn lại sẽ được dùng để thực thi các thuật toán đã thảo luận, chẳng hạn như A-Priori, PCY, Multistage, hoặc Multihash.
* Thuật toán sẽ chạy qua các vòng lặp trên mẫu trong bộ nhớ chính cho mỗi kích thước itemset, cho đến khi không tìm thấy itemset nào có tần suất đủ lớn. Vì mẫu nằm trong bộ nhớ chính, không cần truy cập đĩa trong suốt quá trình xử lý mẫu.
* Các itemset phổ biến được tìm thấy có thể được ghi vào đĩa, và đây là các thao tác I/O duy nhất liên quan đến đĩa.

Nếu thuật toán không thể chạy do thiếu bộ nhớ sau khi lưu mẫu, một lựa chọn là đọc mẫu từ đĩa cho mỗi lượt quét. Vì mẫu nhỏ hơn nhiều so với toàn bộ dữ liệu, chúng ta vẫn tránh được phần lớn các thao tác I/O đĩa mà các thuật toán trước đây sẽ phải thực hiện.

4.1.2 Kết luận

Thuật toán này giúp giảm đáng kể việc truy xuất dữ liệu từ đĩa bằng cách chỉ sử dụng một mẫu ngẫu nhiên nhỏ hơn nhiều so với toàn bộ dữ liệu. Mặc dù phương pháp này không đảm bảo khám phá tất cả các itemset phổ biến, nhưng nó có thể cho phép khám phá hầu hết các itemset phổ biến với ít lượt quét và giảm thiểu các thao tác I/O đĩa.

4.2 Tránh lỗi trong giải thuật Sampling

Thuật toán lấy mẫu đơn giản có thể gặp phải hai loại lỗi: false positives (itemset phổ biến trong mẫu nhưng không trong toàn bộ dữ liệu) và false negatives (itemset phổ biến trong toàn bộ dữ liệu nhưng không trong mẫu). Tuy nhiên, nếu mẫu đủ lớn, các lỗi này ít xảy ra, trừ khi tần suất của itemset gần ngưỡng hỗ trợ.

4.2.1 Tránh false positives

Để loại bỏ false positives, có thể quét toàn bộ dữ liệu và đếm tất cả các itemset đã được xác định là phổ biến trong mẫu. Chỉ giữ lại những itemset vừa phổ biến trong mẫu, vừa phổ biến trong toàn bộ dữ liệu.

4.2.2 Giảm thiểu false negatives

Để giảm false negatives, có thể giảm ngưỡng hỗ trợ cho mẫu, ví dụ từ ps thành 0.9ps. Việc giảm ngưỡng này giúp phát hiện hầu hết itemset phổ biến trong toàn bộ dữ liệu nhưng có thể yêu cầu bộ nhớ lớn hơn.

4.3 Giải thuật của Savasere, Omiecinski, và Navathe

Nhằm tránh cả false negatives và false positives, giải thuật Savasere, Omiecinski, và Navathe (SON) ra đời. Giải thuật này cần hai lần quét dữ liệu đầy đủ.

Cách thực hiện:

* Chia dữ liệu thành các chunk: Dữ liệu được chia thành các phần nhỏ (chunks), có thể là các phần của hệ thống file phân tán hoặc chỉ là một đoạn của file. Mỗi chunk được xử lý như một mẫu.
* Chạy thuật toán lấy mẫu: Áp dụng thuật toán lấy mẫu cho mỗi chunk, với ngưỡng hỗ trợ cho mỗi chunk là ps (nếu mỗi chunk chiếm tỷ lệ p của toàn bộ file, và s là ngưỡng hỗ trợ).
* Lưu itemset phổ biến: Các itemset phổ biến được tìm thấy trong mỗi chunk được lưu trữ trên đĩa.
* Tạo tập itemset ứng cử viên: Sau khi xử lý tất cả các chunk, lấy hợp của tất cả các itemset phổ biến trong ít nhất một chunk để tạo thành tập itemset ứng cử viên. Nếu một itemset không phổ biến trong bất kỳ chunk nào, tổng hỗ trợ của nó sẽ nhỏ hơn s, do đó nó không thể là itemset phổ biến trong toàn bộ dữ liệu.
* Quét lần thứ hai: Quét dữ liệu lần hai để đếm tất cả các itemset ứng cử viên và chọn những itemset có hỗ trợ lớn hơn hoặc bằng s làm itemset phổ biến cuối cùng.

4.4 Giải thuật SON và MapReduce

Thuật toán SON rất phù hợp với môi trường tính toán song song, vì mỗi chunk dữ liệu có thể được xử lý song song. Các itemset phổ biến từ mỗi chunk sẽ được kết hợp để tạo thành các ứng cử viên. Quá trình này có thể được thực hiện bằng MapReduce với các bước như sau:

4.4.1 Hàm Map đầu tiên

Nhận một phần dữ liệu (subset của các giỏ hàng) và tìm các itemset phổ biến trong phần dữ liệu này bằng thuật toán lấy mẫu.

Giảm ngưỡng hỗ trợ từ s xuống ps (nếu mỗi nhiệm vụ Map nhận một phần dữ liệu có tỷ lệ p so với toàn bộ file).

Kết quả của hàm Map là một tập hợp các cặp key-value (F, 1), trong đó F là itemset phổ biến trong mẫu và giá trị luôn là 1.

4.4.2 Hàm Reduce đầu tiên

Mỗi nhiệm vụ Reduce nhận một tập hợp các khóa (itemsets). Giá trị không quan trọng, nhiệm vụ Reduce chỉ trả về các itemset xuất hiện ít nhất một lần.

Kết quả của hàm Reduce là tập hợp các itemset ứng cử viên.

4.4.3 Hàm Map thứ hai

Các nhiệm vụ Map của hàm này nhận tất cả các itemset ứng cử viên từ hàm Reduce đầu tiên và một phần của dữ liệu gốc.

Mỗi nhiệm vụ Map đếm số lần xuất hiện của các itemset ứng cử viên trong phần dữ liệu mà nó nhận được.

Kết quả là các cặp key-value (C, v), trong đó C là một itemset ứng cử viên và v là số lần xuất hiện của itemset đó trong phần dữ liệu.

4.4.4 Hàm Reduce thứ hai

Các nhiệm vụ Reduce nhận các itemset và tính tổng các giá trị liên quan đến chúng.

Kết quả là tổng hỗ trợ của mỗi itemset được giao cho nhiệm vụ Reduce. Các itemset có tổng hỗ trợ lớn hơn hoặc bằng s sẽ được coi là phổ biến trong toàn bộ dữ liệu và được xuất ra, trong khi các itemset có hỗ trợ thấp hơn s sẽ bị loại bỏ.

4.5 Giải thuật Toivonen

Thuật toán Toivonen thực hiện một lần quét qua một mẫu nhỏ và một lần quét toàn bộ dữ liệu (vẫn sử dụng ngẫu nhiên với điều kiện đủ bộ nhớ chính). Thuật toán này không có false negatives hoặc false positives, nhưng có một xác suất nhỏ mà nó sẽ không đưa ra được kết quả nào. Trong trường hợp đó, thuật toán sẽ phải lặp lại cho đến khi tìm được kết quả chính xác. Tuy nhiên, số lần cần lặp lại trước khi thuật toán trả về kết quả chính xác là một hằng số nhỏ.

4.5.1 Chọn mẫu ngẫu nhiên

Chọn một mẫu nhỏ từ bộ dữ liệu và tìm các itemset phổ biến trong mẫu đó với ngưỡng hỗ trợ phải được giảm xuống so với giá trị tỷ lệ. Cụ thể, nếu ngưỡng hỗ trợ cho toàn bộ bộ dữ liệu là s, và kích thước mẫu là tỷ lệ p, thì khi tìm các itemset phổ biến trong mẫu, ta sử dụng ngưỡng là 0.9ps hoặc 0.8ps.

4.5.1 Xây dựng biên âm

Biên âm là tập hợp các itemset không phổ biến trong mẫu, nhưng tất cả các tập con ngay lập tức của chúng (các tập con được tạo ra bằng cách xóa một phần tử) lại là phổ biến trong mẫu. Ví dụ: nếu {A, B} không phổ biến, nhưng các tập con của nó như {A}, {B} lại phổ biến, thì chúng thuộc biên âm.

4.5.3 Quét toàn bộ dữ liệu

Thực hiện một lần quét toàn bộ dữ liệu để đếm số lần xuất hiện của các itemset đã phổ biến trong mẫu hoặc thuộc biên âm. Có hai kết quả có thể xảy ra:

* Không có itemset nào trong biên âm là phổ biến trong toàn bộ dữ liệu: Khi đó, các itemset phổ biến chính xác là các itemset đã được tìm thấy trong mẫu.
* Có ít nhất một itemset trong biên âm là phổ biến trong toàn bộ dữ liệu: Khi đó, không thể chắc chắn rằng không có các itemset lớn hơn, không nằm trong biên giới âm hoặc mẫu, cũng phổ biến trong toàn bộ dữ liệu. Vì vậy, phải lặp lại thuật toán với một mẫu ngẫu nhiên khác.

4.6 Tại sao giải thuật của Toivonen lại hiệu quả?

Thuật toán Toivonen không bao giờ tạo ra false positive, vì nó chỉ báo cáo các tập mục là phổ biến nếu chúng đã được đếm và xác nhận là phổ biến trong toàn bộ bộ dữ liệu. Để chứng minh rằng thuật toán không tạo ra false negative thì khi không có thành phần nào trong negative border là phổ biến trong toàn bộ bộ dữ liệu, thì không thể có một tập mục nào thỏa mãn điều kiện: Phổ biến trong toàn bộ bộ dữ liệu, nhưng không có trong negative border hoặc không có trong tập mục phổ biến từ mẫu.

***\*Chứng minh theo mâu thuẫn***

Giả sử ngược lại, có một tập S là phổ biến trong toàn bộ bộ dữ liệu nhưng không có trong negative border và không phổ biến trong mẫu. Nếu giả thuyết này đúng, thì trong vòng lặp của thuật toán Toivonen, S sẽ không được đưa vào tập mục phổ biến.

Vì tính chất đơn điệu của các tập mục, nếu S là phổ biến trong toàn bộ bộ dữ liệu, thì tất cả các tập con của S cũng phải phổ biến trong toàn bộ bộ dữ liệu. Giả sử T là một tập con của S có kích thước nhỏ nhất trong tất cả các tập con của S không phổ biến trong mẫu.

***\*Tại sao T phải ở negative border***

T không phổ biến trong mẫu: Điều này rõ ràng vì chúng ta đã chọn T là một tập con không phổ biến trong mẫu.

Mỗi tập con trực tiếp của T đều phổ biến trong mẫu: Nếu có một tập con trực tiếp của T không phổ biến trong mẫu, thì sẽ có một tập con nhỏ hơn của S không phổ biến trong mẫu, điều này mâu thuẫn với việc T là tập con nhỏ nhất của S không phổ biến trong mẫu.

Do đó, T phải đồng thời thuộc negative border (vì không phổ biến trong mẫu và các tập con trực tiếp của nó đều phổ biến trong mẫu) và phổ biến trong toàn bộ bộ dữ liệu (vì S là phổ biến trong toàn bộ bộ dữ liệu và T là tập con của S).

CHƯƠNG 5 – ĐẾM CÁC TẬP MỤC PHỔ BIẾN TRONG MỘT LUỒNG

* 1. Phương pháp lấy mẫu (sample) cho luồng

Phương pháp lấy mẫu (Sampling Methods for Streams) là một kỹ thuật giúp ước lượng tập mục phổ biến trong các luồng dữ liệu lớn mà không cần xử lý toàn bộ dữ liệu đến liên tục. Thay vì lưu trữ và phân tích toàn bộ dữ liệu, phương pháp này sử dụng một tập con đại diện để rút trích thông tin quan trọng.

5.1.1 Thu thập mẫu dữ liệu:

* Một số lượng giỏ hàng (baskets) nhất định được thu thập từ luồng dữ liệu và lưu trữ thành một tệp (file).
* Dữ liệu mới xuất hiện sau đó có thể bị bỏ qua hoặc tiếp tục được lưu trữ vào một tệp khác để xử lý sau.

5.1.2 Phân tích tập mẫu:

* Sử dụng thuật toán tìm tập mục phổ biến trên tập mẫu để xác định các tập hợp mục phổ biến.
* Kết quả thu được là một ước lượng về các tập mục phổ biến xuất hiện trong luồng dữ liệu.

5.1.3 Cập nhật kết quả:

* Một số cách để duy trì và cập nhật tập mục phổ biến:
* Chạy lại thuật toán phổ biến: Khi một tệp mới được thu thập, thuật toán tìm tập mục phổ biến sẽ được chạy lại trên tập dữ liệu mới, trong khi một tệp khác tiếp tục được thu thập.
* Duy trì tập mục phổ biến và loại bỏ những mục ít xuất hiện:
  + Mỗi tập mục phổ biến được theo dõi tần suất xuất hiện theo thời gian.
  + Nếu một tập mục có tần suất giảm đáng kể dưới ngưỡng hỗ trợ, nó sẽ bị loại bỏ.
  + Ngược lại, tập mục mới có thể được thêm vào danh sách nếu nó đạt tần suất đủ cao.

5.1.4 Ưu điểm của phương pháp

* Giảm thiểu bộ nhớ sử dụng: Không cần lưu toàn bộ luồng dữ liệu.
* Giúp phân tích nhanh: Chỉ tập trung vào một mẫu dữ liệu đại diện thay vì toàn bộ dữ liệu.
* Dễ dàng cập nhật tập mục phổ biến: Cho phép loại bỏ và thêm mới tập mục dựa trên dữ liệu gần đây.
  1. Các tập mục phổ biến trong Decaying Window

Phương pháp **Frequent Itemsets in Decaying Windows** được sử dụng để phát hiện tập mục phổ biến trong luồng dữ liệu liên tục, trong đó các mục xuất hiện gần đây sẽ có trọng số cao hơn các mục xuất hiện từ lâu. Điều này giúp phản ánh tốt hơn các xu hướng mới thay vì xem xét toàn bộ lịch sử dữ liệu như phương pháp truyền thống.

* + 1. Nguyên lí hoạt động

Cốt lõi của phương pháp này là sử dụng mô hình cửa sổ trượt giảm dần (decaying window) để tính toán mức độ phổ biến của các mục.

Mỗi phần tử trong luồng dữ liệu sẽ có một trọng số giảm dần theo thời gian để phản ánh mức độ quan trọng của nó.

Trọng số của một phần tử i xuất hiện cách phần tử mới nhất t đơn vị thời gian được tính theo công thức:

với c là một hằng số nhỏ (c > 0).

Công thức này đảm bảo rằng:

Các phần tử mới nhất có trọng số cao hơn.

Các phần tử cũ có trọng số giảm theo thời gian.

* + 1. Cách phát hiện tập mục phổ biến

**Định nghĩa điểm số (score) của một mục**

* Với mỗi mục X, điểm số của nó là tổng trọng số của tất cả lần xuất hiện của nó trong luồng dữ liệu.
* Nếu điểm số của X vượt quá một ngưỡng nhất định (ví dụ: 1/2), nó được coi là frequent item.

**Áp dụng vào dữ liệu dạng giỏ hàng (baskets of items)**

Khi luồng dữ liệu chứa các giỏ hàng (mỗi giỏ chứa nhiều mục), phương pháp cần điều chỉnh để xử lý tập mục có nhiều phần tử:

* Cập nhật trọng số của từng mục trong giỏ hàng
  + Khi một giỏ hàng mới xuất hiện, tất cả các mục trong giỏ đều nhận trọng số 1.
  + Tất cả các mục đã theo dõi trước đó sẽ bị giảm trọng số theo hệ số (1 - c).
  + Nếu một mục chưa có điểm số, nó sẽ được khởi tạo với trọng số 1.
* Mở rộng từ mục đơn lẻ sang tập mục nhiều phần tử (itemsets)
  + Nếu chỉ theo dõi từng mục đơn lẻ thì không đủ để tìm ra tập mục phổ biến có nhiều phần tử.
  + Một vấn đề xảy ra: Nếu khởi tạo điểm số cho mọi tập hợp con của một giỏ hàng, số lượng tập mục sẽ trở nên quá lớn.
  + Ví dụ: Một giỏ hàng có 20 mục có thể sinh ra 1 triệu tập hợp con, khiến việc theo dõi trở nên không khả thi.
* Giải pháp: Mượn ý tưởng từ thuật toán Apriori
  + Chỉ theo dõi tập mục I nếu tất cả các tập con trực tiếp của I đã có điểm số.
  + Quy tắc này giúp giảm số lượng tập hợp cần theo dõi, đồng thời đảm bảo các tập mục quan trọng vẫn được ghi nhận.
    1. Ví dụ

Giả sử có một tập mục lớn I xuất hiện trong luồng dữ liệu theo chu kỳ 1/2c giỏ hàng một lần:

* Lần đầu tiên I xuất hiện: chỉ các mục đơn lẻ (singleton) của I được ghi nhận và có điểm số.
* Lần thứ hai I xuất hiện: các tập con có hai phần tử (doubleton) của I bắt đầu được theo dõi, vì tất cả các mục đơn lẻ của I đều đã có điểm số.
* Lần thứ k I xuất hiện: các tập con có k-1 phần tử đã được ghi nhận từ trước, vì vậy tập hợp đầy đủ I sẽ được theo dõi.
* Cuối cùng, tập mục đầy đủ I được ghi nhận là phổ biến.

Nhờ cách tiếp cận này, hệ thống chỉ theo dõi tập mục nào thực sự có khả năng xuất hiện phổ biến, thay vì phải lưu tất cả các tập hợp con.

* 1. Phương pháp Hybrid
     1. Tổng quan về phương pháp Hybrid Methods

Phương pháp này chia quá trình phân tích thành hai giai đoạn chính:

Bước lấy mẫu:

* Lấy một tập con từ luồng dữ liệu (sample) chứa b giỏ hàng (baskets).
* Chạy một thuật toán khai phá tập mục phổ biến truyền thống (ví dụ: Apriori) trên tập mẫu này.
* Chọn ra các tập mục có tần suất vượt qua một ngưỡng nhất định.

Bước cập nhật theo cửa sổ trượt giảm dần:

* Các tập mục được chọn từ bước 1 sẽ được theo dõi trong luồng dữ liệu theo phương pháp cửa sổ trượt giảm dần (decaying window).
* Mỗi lần một giỏ hàng mới xuất hiện, các tập mục này sẽ có điểm số giảm theo hệ số (1 - c).
* Nếu một tập mục xuất hiện trong giỏ hàng mới, điểm số của nó sẽ được tăng lên.
* Nếu điểm số của một tập mục giảm xuống dưới mức ngưỡng tối thiểu s, nó sẽ bị loại bỏ.
  + 1. Cách tính điểm số ban dầu của tập mục

Giả sử:

b là số lượng giỏ hàng trong mẫu ban đầu.

c là hằng số giảm trọng số trong cửa sổ trượt.

s là ngưỡng điểm số tối thiểu để một tập mục được duy trì.

Khi chạy thuật toán phát hiện tập mục phổ biến trên mẫu, ngưỡng hỗ trợ cho thuật toán này được tính như sau:

Ngưỡng hỗ trợ

Nếu một tập mục I có tần suất xuất hiện là t trong mẫu, điểm số khởi tạo của nó sẽ là:

Điểm số ban đầu của

**Ví dụ:**

Chọn mẫu có 108 giỏ hàng.

Đặt hằng số giảm trọng số c = 10-6.

Ngưỡng điểm số tối thiểu s=10.

Ngưỡng hỗ trợ để một tập mục được coi là phổ biến trong mẫu:

Nếu một tập mục I xuất hiện 2000 lần trong mẫu, điểm số khởi tạo của nó là:

Khi luồng dữ liệu tiếp tục, mỗi lần một giỏ hàng mới đến, điểm số của I bị giảm theo hệ số 0.999999. Nếu I xuất hiện trong giỏ hàng mới, điểm số được cộng thêm 1. Nếu điểm số giảm xuống dưới 10, tập mục này sẽ bị loại bỏ.

* + 1. Cách xử lí tập mục mới xuất hiện

Một hạn chế lớn của phương pháp này là không có cách hiệu quả để khởi tạo điểm số cho các tập mục mới.

* Nếu một tập mục mới I xuất hiện trong một giỏ hàng đơn lẻ, điểm số của nó chỉ có thể tăng lên 1, không thể nhảy ngay lên ngưỡng tối thiểu 10.
* Cách duy nhất để thêm tập mục mới là chạy lại thuật toán phát hiện tập mục phổ biến trên một mẫu mới và bổ sung các tập mục đạt ngưỡng vào danh sách theo dõi.

CHƯƠNG 6 – SỰ PHÁT TRIỂN CỦA TẬP MỤC PHỔ BIẾN – TẬP MỤC HỮU ÍCH VÀ GIỚI THIỆU CHUNG VỀ MỘT SỐ GIẢI THUẬT TÌM TẬP MỤC HỮU ÍCH HIỆN NAY

Tập phổ biến là một trong những nội dung có tính ứng dụng cao nên từ khi khái niệm này được sinh ra cho đến hiện tại, giải thuật đã phân nhánh và có những cải thiện đáng kể. Một trong những sự phân nhánh của tập mục phổ biến là tập mục hữu ích (High utility itemsets).

6.1 Giới thiệu về High utility itemsets

Nếu frequent itemsets là những tập mục mà tần suất xuất hiện trong cơ sở dữ liệu vượt qua một ngưỡng hỗ trợ (support threshold). Nói cách khác, nếu một tập mục xuất hiện đủ số lần trong cơ sở dữ liệu (tính theo tần suất), nó được coi là phổ biến thì high utility itemsets là những tập mục không chỉ có tần suất xuất hiện cao mà còn có giá trị sử dụng cao trong việc tạo ra lợi nhuận hoặc giá trị trong các ứng dụng thực tế (ví dụ, trong bán hàng, quản lý kho, ...). Giá trị sử dụng có thể dựa trên yếu tố như lợi nhuận hoặc giá trị của sản phẩm liên quan đến tập mục đó.

Ví dụ: Trong bán lẻ, tập mục {A, B} có thể có tần suất xuất hiện cao (được coi là phổ biến), nhưng nếu lợi nhuận từ việc bán các sản phẩm A và B lại thấp, nó có thể không phải là một high utility itemset. Ngược lại, một tập mục ít xuất hiện nhưng có lợi nhuận cao có thể được coi là một high utility itemset.

Trong thực tế khi áp dụng vào một cửa hàng, giả sử cửa hàng bán các mặt hàng như: sữa, bánh mì, trái cây, kẹo, nước giải khát. Nhà quản lý cửa hàng muốn tìm hiểu các mẫu giao dịch phổ biến trong cửa hàng, tức là các sản phẩm phổ biến được mua cùng nhau để tổ chức lại cửa hàng, đặt các sản phẩm như sữa và bánh mì gần nhau, hoặc cung cấp các khuyến mãi như mua bánh mì và nước giải khát để khuyến khích khách hàng mua nhiều sản phẩm hơn, thì sẽ tìm frequent itemsets. Còn khi ngoài việc tìm các mặt hàng phổ biến được mua cùng nhau, cửa hàng còn quan tâm đến lợi nhuận mà các mặt hàng mang lại, cửa hàng sẽ quan tâm đến các mặt hàng không chỉ được mua nhiều mà còn mang lại lợi nhuận cao. Ví dụ, {sữa, trái cây, nước giải khát} có lợi nhuận cao hơn so với các mặt hàng khác, mặc dù không phải là mẫu mục phổ biến nhất. Cửa hàng có thể áp dụng chiến lược bán sữa, trái cây, và nước giải khát như một combo với mức giá ưu đãi để tăng doanh thu và lợi nhuận.

6.2 So sánh Frequent itemsets và High utility itemsets

Bảng 6. Bảng so sánh Frequent itemsets và High utility itemsets

|  |  |  |
| --- | --- | --- |
| Đặc điểm | Frequent itemsets | High utility itemsets |
| Định nghĩa | Là các tập mục xuất hiện phổ biến trong cơ sở dữ liệu. | Là các tập mục có giá trị sử dụng (utility) cao, thường là lợi nhuận, doanh thu hoặc giá trị thực tế khác. Có thể không phổ biến. |
| Tiêu chí xác định | Dựa trên hỗ trợ (support) – tần suất xuất hiện của các mục trong cơ sở dữ liệu. | Dựa trên giá trị sử dụng (utility) – giá trị thực tế mà tập mục mang lại, chẳng hạn như lợi nhuận hoặc chi phí. |
| Phương pháp khai thác | Thường sử dụng các thuật toán như Apriori hoặc FP-growth. | Sau khi tìm được tập mục phổ biến, giá trị sử dụng sẽ được tính toán để xác định các tập mục có utility cao. |
| Mục tiêu | Tìm các tập mục xuất hiện nhiều lần trong dữ liệu. | Tìm các tập mục không chỉ xuất hiện nhiều mà còn có giá trị thực tế cao trong ngữ cảnh cụ thể. |
| Áp dụng trong trường hợp nào? | Khai thác các mẫu dữ liệu chung, như trong các giao dịch mua sắm. | Áp dụng khi cần tìm các tập mục mang lại giá trị cao, như trong phân tích lợi nhuận của các sản phẩm. |
| Ví dụ | {A, B}, {A, C}, {B, C} (tập mục xuất hiện nhiều lần trong các giao dịch). | {A, C} có thể mang lại lợi nhuận cao hơn {A, B}, mặc dù {A, B} xuất hiện nhiều hơn. |
| Lợi ích | Giúp phát hiện các mối quan hệ giữa các mục trong cơ sở dữ liệu. | Giúp tối ưu hóa giá trị lợi nhuận từ các mối quan hệ sản phẩm trong các giao dịch. |

6.3 Một số giải thuật tìm top K high utility itemsets

Nói về High utility itemsets, nhiều nghiên cứu chỉ tập trung vào positive utility item, chẳng hạn như khi ta bán một món hàng và nó mang lại lợi nhuận, nó sẽ có positive utility, tuy vậy, negative utility cũng có những lợi ích riêng về kinh tế. Ví dụ khi ta có thông tin về tả lót và bia là những thứ được mua phổ biến, tuy vậy vào dịp giảm giá của cửa hàng, ta giảm giá tả lót và tăng giá bia, điều này làm tả lót có negative utility, nhưng dễ dàng nhận thấy số lượng bia bán được có phần gia tăng nên lợi ích mang lại của tả lót là phần nào đó làm tăng khả năng bán bia. Vì lí do trên mà ở phần sau em sẽ trình bày các giải thuật tìm kiếm High utility itemsets cho cả positive và negative utility item.

6.3.1 EHMIN

EHMIN (Efficient High-utility pattern Mining with Negative unit profits) là một thuật toán khai thác mẫu hữu ích cao được thực hiện bởi Heonho Kim, Taewoong Ryua, Chanhee Leea, Hyeonmo Kim, Eunchul Yoon, Bay Vo, Jerry Chun-Wei Lin, Unil Yun, đặc biệt hướng đến các tập mục lợi nhuận đơn vị âm. Mục tiêu của EHMIN là cải thiện hiệu suất khai thác dữ liệu khi có các giá trị lợi nhuận âm, điều mà các phương pháp trước đó không xử lý tốt. Ý tưởng chính của EHMIN dựa trên ba cốt lõi.

***\*Xử lý lợi nhuận đơn vị âm trong khai thác mẫu hữu ích cao***

Trong các thuật toán HUPM trước đây như EFIM hoặc HUI-Miner, lợi nhuận đơn vị (unit profit) của một mặt hàng thường được giả định là dương.

Khi gặp dữ liệu có lợi nhuận âm, việc áp dụng trực tiếp các chiến lược cắt tỉa (pruning) như Transaction Weighted Utility (TWU) có thể làm mất các mẫu hữu ích cao.

EHMIN điều chỉnh cách đánh giá và cập nhật lợi nhuận, đảm bảo rằng các mục tiêu khai thác vẫn chính xác ngay cả khi có lợi nhuận âm.

***\*Cải tiến cấu trúc dữ liệu điều kiện để giảm không gian tìm kiếm***

EHMIN sử dụng một cơ chế xây dựng cây điều kiện tối ưu hơn so với các phương pháp truyền thống.

Bằng cách tập trung vào các giao dịch có ảnh hưởng trực tiếp đến lợi nhuận tổng thể của một mẫu, EHMIN giảm bớt số lượng giao dịch cần xét, từ đó cải thiện tốc độ khai thác.

A diagram of a diagram

AI-generated content may be incorrect.

Hình 6. Cấu trúc EHMIN-list

***\*Chiến lược cắt tỉa hiệu quả để tối ưu thời gian chạy***

EHMIN cải tiến các phương pháp cắt tỉa bằng cách tính toán lại giới hạn trên của lợi nhuận (upper bound) sao cho phù hợp với dữ liệu có lợi nhuận âm.

Cách tiếp cận này giúp loại bỏ sớm các tập hợp mục không hữu ích mà không làm mất các mẫu quan trọng.

A diagram of data processing

AI-generated content may be incorrect.

Hình 6. Tổng quan về quá trình hoạt động của EHMIN

Qua các thử nghiệm trên nhiều tập dữ liệu thực và giả lập, EHMIN cho thấy:

* Tốc độ khai thác nhanh hơn so với các thuật toán trước đây khi làm việc với dữ liệu có lợi nhuận âm.
* Tiết kiệm bộ nhớ hơn nhờ vào cấu trúc dữ liệu tối ưu.
* Độ chính xác cao hơn so với các thuật toán truyền thống không được thiết kế để xử lý lợi nhuận âm.

Trên đây là những giới thiệu cơ bản về EHMIN, chi tiết về giải thuật này có thể đọc tại <https://www.sciencedirect.com/science/article/abs/pii/S0957417422013689>

6.3.2 EMHUN

Thuật toán EHMIN tuy có những cải thiện về tốc độ, hiệu năng so với các thuật toán trước, nhưng do vẫn áp dụng quét 2 lần dataset, điều này làm thời gian thực hiện việc tìm kiếm trở nên rất tốn thời gian, thuật toán EMHUN ra đời nhằm cải thiện những vấn đề trên.

Ý tưởng chính của thuật toán EMHUN kế thừa nhiều từ EHIN (2018) tuy nhiên có những cải thiện như sau:

***\*Phân thành ba loại mặt hàng thay vì hai loại***

Trong khi EHIN và EHMIN chia mặt hàng thành hai danh sách: Mặt hàng có lợi nhuận dương trong toàn bộ cơ sở dữ liệu và mặt hàng có lợi nhuận âm trong toàn bộ cơ sở dữ liệu. thì EMHUN bổ sung thêm một danh sách mới cho mặt hàng lai, tức là những mặt hàng có thể có lợi nhuận dương trong một số giao dịch nhưng lại có lợi nhuận âm trong những giao dịch khác.

***\*Thay đổi cách sắp xếp mặt hàng***

EHIN sắp xếp tất cả các mặt hàng có lợi nhuận dương trước, sau đó mới đến mặt hàng có lợi nhuận âm.

EMHUN sắp xếp lại theo một quy tắc khác để phản ánh chính xác hơn tác động của lợi nhuận biến đổi.

Từ những thay đổi trên giúp thuật toán xác định chính xác hơn giá trị lợi nhuận thực tế của các tập phổ biến, tránh bỏ sót tập có ích hoặc khai thác sai tập không có ích đồng thời giảm số lượng giao dịch cần quét, từ đó tăng tốc độ xử lý, làm cho bài toán tối ưu hơn.

Trên đây là những giới thiệu cơ bản về EMHUN, chi tiết về giải thuật này có thể đọc tại <https://www.sciencedirect.com/science/article/abs/pii/S0957417423019917>.

**TÀI LIỆU THAM KHẢO**

**Tiếng Anh**

1. [Jure Leskovec](http://cs.stanford.edu/~jure/), [Anand Rajaraman](https://twitter.com/anand_raj), [Jeff Ullman](http://infolab.stanford.edu/~ullman/). Mining of Massive Datasets (3rd ed.). Chapter 6, pp 213-251
2. H. Toivonen, “Sampling large databases for association rules,” Intl. Conf. on Very Large Databases, pp. 134–145, 1996.
3. PCY Algorithm in Big Data Analytics [Online]. Available: <https://shorturl.at/c9WyU>.
4. The SON Algorithm and Map – Reduce [Online]. Available: <https://shorturl.at/sDj4T>.