BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP.HCM

NGUYỄN VĂN ĐIỂN

KHAI THÁC LUẬT PHÂN LỚP KẾT HỢP TRÊN CƠ SỞ DỮ LIỆU BỊ SỬA ĐỔI

LUẬN VĂN THẠC SĨ

Chuyên ngành: Công nghệ thông tin

Mã số ngành: 60480201

TP. HỒ CHÍ MINH, tháng 11 năm 2016

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP.HCM

NGUYỄN VĂN ĐIỂN

KHAI THÁC LUẬT PHÂN LỚP KẾT HỢP TRÊN CƠ SỞ DỮ LIỆU BỊ SỬA ĐỔI

LUẬN VĂN THẠC SĨ

Chuyên ngành: Công nghệ thông tin

Mã số ngành: 60480201

CÁN BỘ HƯỚNG DẪN KHOA HỌC: TS. NGUYỄN THỊ THUÝ LOAN

TP. HỒ CHÍ MINH, tháng 12 năm 2016

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

Cán bộ hướng dẫn khoa học : TS. NGUYỄN THỊ THUÝ LOAN

(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Luận văn Thạc sĩ được bảo vệ tại Trường Đại học Công nghệ TP. HCM ngày 17 tháng 12 năm 2016

Thành phần Hội đồng đánh giá Luận văn Thạc sĩ gồm:

(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ Luận văn Thạc sĩ)

TT	Họ và tên	Chức danh Hội đồng
1	GS. TS. Phan Thị Tươi	Chủ tịch
2	TS. Cao Tùng Anh	Phản biện 1
3	TS. Phạm Thị Thiết	Phản biện 2
4	PGS. TS. Võ Đình Bảy	Ủy viên
5	TS. Văn Thiên Hoàng	Ủy viên, Thư ký

Xác nhận của Chủ tịch Hội đồng đánh giá Luận sau khi Luận văn đã được sửa chữa (nếu có).

Chủ tịch Hội đồng đánh giá LV

TRƯỜNG ĐH CÔNG NGHỆ TP. HCM
VIỆN ĐÀO TẠO SAU ĐẠI HỌC

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

TP. HCM, ngày 01 tháng 12 năm 2016

NHIỆM VỤ LUẬN VĂN THẠC SĨ

Họ tên học viên: NGUYỄN VĂN ĐIỀN..... Giới tính: Nam .....
Ngày, tháng, năm sinh: 22/06/1982 ..... Nơi sinh: Đồng Nai .....
Chuyên ngành: Công nghệ thông tin ..... MSHV: 1441860006.....

I- Tên đề tài:

Khai thác luật phân lớp kết hợp trên cơ sở dữ liệu bị sửa đổi.....

II- Nhiệm vụ và nội dung:

- ✓ Tìm hiểu thuận toán khai thác luật phân lớp trên cơ sở dữ liệu tĩnh.
- ✓ Tìm hiểu thuận toán khai thác luật phân lớp kết hợp trên cơ sở dữ liệu tăng trưởng.
- ✓ Tìm hiểu và xây dựng ví dụ cho thuận toán khai thác luật phân lớp kết hợp trên cơ sở dữ liệu bị sửa đổi.
- ✓ Xây dựng chương trình khai thác luật phân lớp kết hợp trên liệu bị sửa đổi.
- ✓ Viết báo cáo.

III- Ngày giao nhiệm vụ: 15/07/2015 .....

IV- Ngày hoàn thành nhiệm vụ: 1/12/2016.....

V- Cán bộ hướng dẫn: TS. NGUYỄN THỊ THUÝ LOAN

CÁN BỘ HƯỚNG DẪN	KHOA QUẢN LÝ CHUYÊN NGÀNH
(Họ tên và chữ ký)	(Họ tên và chữ ký)

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Luận văn này đã được cảm ơn và các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

Học viên thực hiện Luận văn

(Ký và ghi rõ *họ tên)*

NGUYỄN VĂN ĐIỂN

LỜI CÁM ƠN

Trước tiên tôi xin bày tỏ lòng biết ơn chân thành đến cô Nguyễn Thị Thuý Loan đã tận tình hỗ trợ, hướng dẫn và động viên tinh thần giúp chúng tôi hoàn thành luận văn này.

Cho chúng tôi bày tỏ lòng biết ơn chân thành đến Quý Thầy Cô đã hết lòng giảng dạy, truyền đạt những tri thức khoa học và kinh nghiệm quý báu cho chúng tôi trong suốt thời gian tham gia học tập theo chương trình thạc sỹ ở trường đại học Công nghệ Tp.HCM.

Sau cùng, cho tôi được chuyển lời cám ơn gia đình thân yêu của tôi đã luôn luôn bên cạnh tôi những lúc khó khăn nhất, là nguồn động viên và ủng hộ tinh thần rất lớn để tôi có thể hoàn thành luận văn.

NGUYỄN VĂN ĐIỂN

TÓM TẮT

Ngày nay, dữ liệu ngày càng phong phú, đa dạng và khổng lồ về nhiều lĩnh vực. Đặc biệt sự phát triển của công nghệ thông tin và việc ứng dụng công nghệ thông tin trong nhiều lĩnh vực đã làm cho kho dữ liệu ấy tăng lên nhanh chóng. Điều này dẫn đến một vấn đề là cần có những kỹ thuật và công cụ mới để tự động chuyển đổi lượng dữ liệu khổng lồ kia thành các tri thức có ích, phục vụ cho con người. Mặt khác, trong môi trường cạnh tranh thì người ta ngày càng cần có thông tin với tốc độ nhanh để giúp cho việc ra quyết định và ngày càng có nhiều câu hỏi mang tính chất định tính cần phải trả lời dựa trên khối lượng dữ liệu khổng lồ đã có. Hiện nay cũng đã có nhiều thuật toán về khai thác luật phân lớp kết hợp trên cơ sở dữ liệu tĩnh nhưng các thuật toán khai thác luật phân lớp trên cơ sở dữ liệu bị sửa đổi thì chưa có.

Để giải quyết các vấn đề như đã nêu ở trên, nội dung nghiên cứu của luận văn sẽ tập trung vào nghiên cứu các thuật toán khai thác luật kết hợp, khai thác luật phân lớp kết hợp trên cơ sở dữ liệu bị sửa đổi, viết chương trình thực nghiệm một thuật toán đã nghiên cứu.

ABSTRACT

Today, data is increasingly rich, and huge variety of fields. In particular the development of information technology and the application of information technology in various fields has made data warehouse was increasing rapidly. This leads to a problem is the need for new techniques and tools to automatically convert other huge amounts of data into useful knowledge, to serve man. On the other hand, in a competitive environment, people increasingly need information at a fast pace to help in decision-making and more questions of qualitative nature need to be answered based on the volume of data giant had. Currently also had many mining algorithms combined classification rules based on static data mining algorithms but subclass law on database is modified, then no.

To solve the problems as mentioned above, the research content of the thesis will focus on the study of algorithms combined mining law, mining law combined classification on the basis of revised data, Useful program an algorithm experimentally studied.

DANH MỤC CÁC TỪ VIẾT TẮT

VIẾT TẮT	VIẾT ĐẦY ĐỦ	Ý NGHĨA
CSDL	Cơ sở dữ liệu	
DHP	Direct Hashing and Pruning	Kỹ thuật băm và tỉa trực tiếp
FUP	Fast Update algorithm	Thuật toán cập nhật nhanh
S_U	Upper support threshold	Độ hỗ trợ ngưỡng trên
S_L	Lower support threshold	Độ hỗ trợ ngưỡng dưới
minSup	Minimum support	Độ hỗ trợ tối thiểu
minConf	Minimum Confidence	Độ tin cậy tối thiểu

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ

MỤC LỤC

Chương 1: MỞ ĐẦU

1.1. Đặt vấn đề

Ngày nay, con người đang sở hữu kho dữ liệu phong phú, đa dạng và khổng lồ về nhiều lĩnh vực. Đặc biệt sự phát triển của công nghệ thông tin và việc ứng dụng công nghệ thông tin trong nhiều lĩnh vực đã làm cho kho dữ liệu ấy tăng lên nhanh chóng. Việc bùng nổ này đã dẫn tới một vấn đề cấp bách cần được giải quyết là cần có những kỹ thuật và công cụ mới để tự động chuyển đổi lượng dữ liệu khổng lồ kia thành các tri thức có ích, phục vụ cho con người. Mặt khác, trong môi trường cạnh tranh thì người ta ngày càng cần có thông tin với tốc độ nhanh để giúp cho việc ra quyết định và ngày càng có nhiều câu hỏi mang tính chất định tính cần phải trả lời dựa trên khối lượng dữ liệu khổng lồ đã có. Quá trình tiến hành các công việc như vậy gọi là quá trình phát hiện tri thức trong cơ sở dữ liệu, trong đó kỹ thuật khai thác dữ liệu cho phép phát hiện tri thức tiềm ẩn ấy. Từ đó, các kỹ thuật khai thác dữ liệu đã trở thành một lĩnh vực thời sự của nền Công nghệ thông tin thế giới hiện nay nói chung và Việt Nam nói riêng. Rất nhiều tổ chức và công ty lớn trên thế giới đã áp dụng kỹ thuật khai thác dữ liệu vào các hoạt động sản xuất kinh doanh của mình và thu được những lợi ích to lớn.

Các kỹ thuật phát hiện tri thức và khai thác dữ liệu được thực hiện qua nhiều giai đoạn và sử dụng nhiều kỹ thuật: phân lớp, phân cụm, phân tích sự tương tự, tổng hợp, luật kết hợp,… áMột trong những nội dung cơ bản và phổ biến trong khai thác dữ liệu là phát hiện cc luật kết hợp. Phương pháp này nhằm tìm ra các tập thuộc tính thường xuất hiện đồng thời trong cơ sở dữ liệu và rút ra các luật về ảnh hưởng của một tập thuộc tính dẫn đến sự xuất hiện của một hoặc nhiều tập thuộc tính khác như thế nào? Do đó việc phát hiện ra các luật kết hợp là một bước rất quan trọng trong khai thác dữ liệu.

1.2. Tính cấp thiết của đề tài

Nắm bắt được nhu cầu đó, nhiều giải pháp đã được áp dụng nhằm rút trích tri thức, trong đó khai thác luật kết hợp là một trong những bài toán thu hút được nhiều sự quan tâm nghiên cứu của các nhà nghiên cứu trên thế giới. Cũng đã có nhiều giải pháp được đề xuất cho vấn đề khai thác luật như luật khai thác luật truyền thống, khai thác luật không dư thừa.

Tuy nhiên những phương pháp trên chỉ tập trung chủ yếu trong việc xây dựng thuật toán phân lớp dựa trên luật kết hợp hoặc xây dựng luật phân lớp, mà không thảo luận nhiều về vấn đề thời gian thực thi (khai thác) của các thuật toán. Hơn thế nữa, khai thác phân lớp dựa trên luật kết hợp (CARs) tiêu tốn rất nhiều thời gian bởi vì nó khai thác một tập đầy đủ các luật thỏa ngưỡng. Vì thế, cải thiện thời gian khai thác phân lớp dựa trên luật kết hợp là một trong những vấn đề chính cần được giải quyết. Ngoài ra, các nghiên cứu hiện nay tập trung chủ yếu về khai thác luật phân lớp kết hợp trên cơ sở dữ liệu tĩnh, mà ít quan tâm đến việc khai thác trên cơ sở dữ liệu bị sửa đổi.

Từ những vấn đề nêu trên, tôi chọn đề tài "Khai thác luật phân lớp kết hợp trên cơ sở dữ liệu bị sửa đổi" để làm luận văn tốt nghiệp.

1.3. Mục tiêu, đối tượng và phương pháp nghiên cứu

1.3.1. Mục tiêu của đề tài

- ✓ Tìm hiểu khái quát về khai thác dữ liệu trong đó đi sâu về luật kết hợp.
- ✓ Tìm hiểu luật phân lớp kết hợp CAR-Miner.
- ✓ Tìm hiểu luật phân lớp kết hợp trên cơ sở dữ liệu tăng trưởng.
- ✓ Nghiên cứu xây dựng thuật toán luật phân lớp kết hợp trên cơ sở dữ liệu bị sửa đổi.
- ✓ Cài đặt thuật toán khai thác luật phân lớp kết hợp trên cơ sở dữ liệu bị sửa đổi.

1.3.2. Đối tượng và phạm vi nghiên cứu

- ✓ Nghiên cứu bài toán khai thác luật phân lớp dựa vào luật kết hợp và thuật toán CAR-Miner.
- ✓ Khai thác luật phân lớp kết hợp trên cở sở dữ liệu bị sửa đổi.

1.3.3. Phương pháp luận và phương pháp nghiên cứu

- ✓ Tìm hiểu các tài liệu trong và ngoài nước về khai thác dữ liệu, luật kết hợp, luật phân lớp, luật phân lớp kết hợp.
- ✓ Tìm hiểu thuật toán CAR-Miner trong bài toán phân lớp dựa vào luật kết hợp.

Chương 2: TỔNG QUAN

2.1. Khai thác dữ liệu

2.1.1. Khái niệm

Khai thác dữ liệu (Data Mining) là một khái niệm ra đời vào những năm cuối của thập kỷ 1980. Nó là quá trình khám phá thông tin tìm ẩn trong các cơ sở dữ liệu và có thể xem như là một bước trong quá trình khám phá tri thức. Khai thác dữ liệu là giai đoạn quan trọng nhất trong tiến trình khai thác tri thức từ cơ sở dữ liệu, các tri thức này hỗ trợ trong việc ra quyết định trong khoa học và kinh doanh, ...

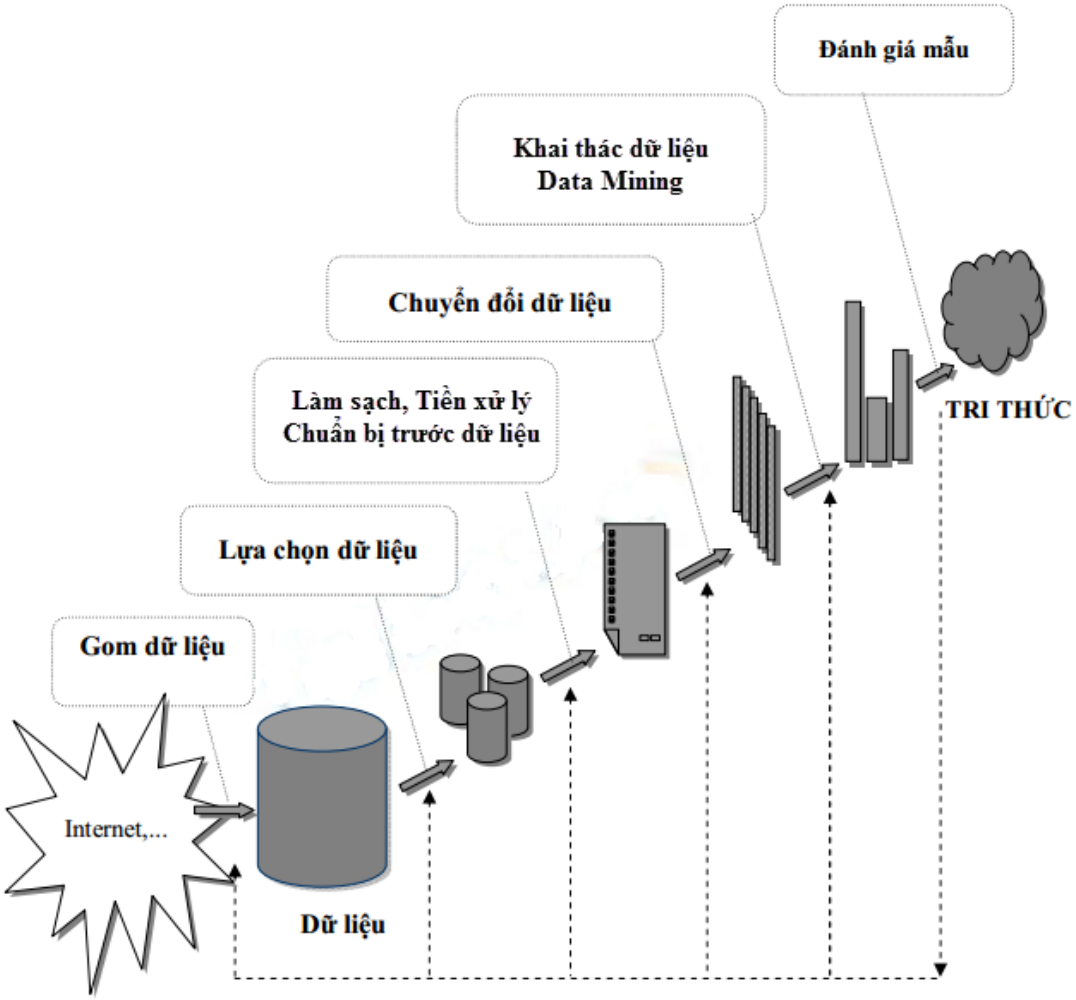
Khai thác dữ liệu là tiến trình khái quát các sự kiện rời rạc trong dữ liệu thành các tri thức mang tính khái quát, tính quy luật hỗ trợ tích cực cho các tiến trình ra quyết định. Khai thác dữ liệu là việc rút trích tri thức một cách tự động và hiệu quả từ một khối dữ liệu rất lớn. Tri thức đó thường ở dạng các mẫu tin có tính chất không tầm thường, không tường minh (ẩn), chưa được biết đến và có tiềm năng mang lại lợi ích.

2.1.2. Quá trình khai thác dữ liệu

Bắt đầu của quá trình là kho dữ liệu thô và kết thúc với tri thức được chiết xuất ra. Về lý thuyết thì có vẻ rất đơn giản nhưng thực sự đây là một quá trình rất khó khăn gặp phải rất nhiều vướng mắc như: quản lý các tập dữ liệu, phải lặp đi lặp lại toàn bộ quá trình, ...

1. **Gom dữ liệu:** Tập hợp dữ liệu là bước đầu tiên trong quá trình khai thác dữ liệu. Đây là bước được khai thác trong một cơ sở dữ liệu, một kho dữ liệu và thậm chí các dữ liệu từ các nguồn ứng dụng Web.
2. **Lựa chọn dữ liệu:** hay còn gọi là trích lọc dữ liệu. Ở giai đoạn này dữ liệu được lựa chọn hoặc phân chia theo một số tiêu chuẩn nào đó, ví dụ chọn tất cả những người có tuổi đời từ 25 - 35 và có trình độ đại học.
3. **Làm sạch, tiền xử lý và chuẩn bị trước dữ liệu**: Giai đoan thứ ba này là giai đoạn hay bị sao lãng, nhưng thực tế nó là một bước rất quan trọng trong quá

trình khai thác dữ liệu. Một số lỗi thường mắc phải trong khi gom dữ liệu là tính không đủ chặt chẽ, logic. Vì vậy, dữ liệu thường chứa các giá trị vô nghĩa và không có khả năng kết nối dữ liệu. Ví dụ: tuổi = 273. Giai đoạn này sẽ tiến hành xử lý những dạng dữ liệu không chặt chẽ nói trên. Những dữ liệu dạng này được xem như thông tin dư thừa, không có giá trị. Bởi vậy, đây là một quá trình rất quan trọng vì dữ liệu này nếu không được "làm sạch - tiền xử lý - chuẩn bị trước" thì sẽ gây nên những kết quả sai lệch nghiêm trọng.

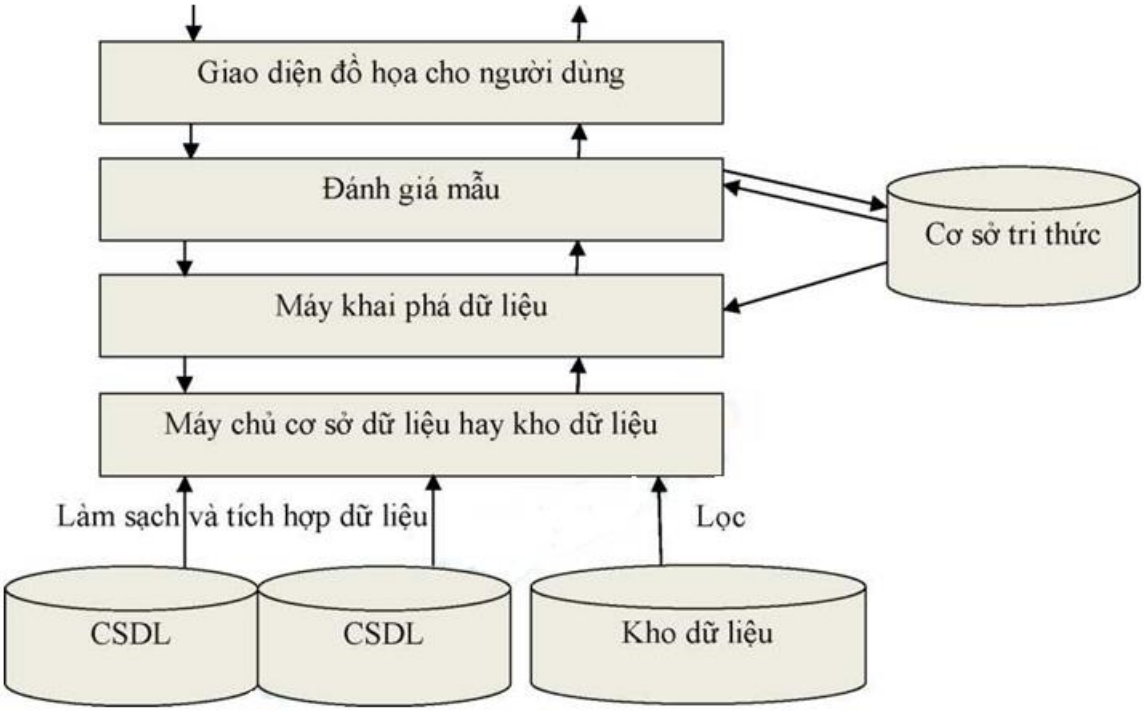


Hình 2.1: Quá trình khai thác dữ liệu [3]

4. **Chuyển đổi dữ liệu:** Tiếp theo là giai đoạn chuyển đổi dữ liệu, dữ liệu đưa ra có thể sử dụng và điều khiển được bởi việc tổ chức lại nó. Dữ liệu đã được chuyển đổi phù hợp với mục đích khai thác.

5. **Khai thác dữ liệu:** Là phát hiện và trích mẫu dữ liệu. Ở giai đoạn này nhiều thuật toán khác nhau đã được sử dụng để trích ra các mẫu từ dữ liệu. Thuật toán thường dùng là nguyên tắc phân loại, nguyên tắc kết hợp hoặc các mô hình dữ liệu tuần tự.
6. **Đánh giá kết quả mẫu:** Đây là giai đoạn cuối trong quá trình khai thác dữ liệu. Ở giai đoạn này, các mẫu dữ liệu được chiết xuất ra bởi phần mềm khai thác dữ liệu. Không phải bất cứ mẫu dữ liệu nào cũng đều hữu ích, đôi khi nó còn bị sai lệch. Vì vậy, cần phải ưu tiên những tiêu chuẩn đánh giá để chiết xuất ra các tri thức.

2.1.3. Kiến trúc của một hệ thống khai thác dữ liệu



Hình 2.2: Kiến trúc của một hệ thống khai thác dữ liệu [3]

Kiến trúc của một hệ thống khai thác dữ liệu: gồm 5 phần sau:

1. **Máy chủ cơ sở dữ liệu hay máy chủ kho dữ liệu:** Máy chủ này có trách nhiệm lấy dữ liệu thích hợp dựa trên những yêu cầu khai thác của người dùng.
2. **Cơ sở tri thức:** Đây là miền tri thức được dùng để tìm kiếm hay đánh giá độ quan trọng của các hình mẫu kết quả.

3. **Máy khai thác dữ liệu:** Một hệ thống khai thác dữ liệu cần phải có một tập các mô đun chức năng để thực hiện công việc, chẳng hạn như đặc trưng hóa, kết hợp, phân lớp, phân cụm, phân tích sự tiến hoá.
4. **Mô đun đánh giá mẫu:** Bộ phận này tương tác với các mô đun khai thác dữ liệu để tập trung vào việc duyệt tìm các mẫu đáng được quan tâm. Cũng có thể mô đun đánh giá mẫu được tích hợp vào mô đun khai thác tuỳ theo sự cài đặt của phương pháp khai thác được dùng.
5. **Giao diện đồ họa cho người dùng:** Thông qua giao diện này, người dùng tương tác với hệ thống bằng cách đặc tả một yêu cầu.

2.1.4. Một số phương pháp khai thác dữ liệu

✓ **Phương pháp quy nạp**

Một cơ sở dữ liệu là một kho thông tin nhưng các thông tin quan trọng hơn cũng có thể được suy diễn từ kho thông tin đó. Có hai kỹ thuật chính để thực hiện việc này là suy diễn và quy nạp.

Phương pháp suy diễn: Nhằm rút ra thông tin là kết qủa logic của các thông tin trong cơ sở dữ liệu. Phương pháp suy diễn dựa trên các sự kiện chính xác để suy ra các tri thức mới từ các thông tin cũ. Mẫu chiết xuất được bằng cách sử dụng phương pháp này thường là các luật suy diễn.

Phương pháp quy nạp: Phương pháp quy nạp suy ra các thông tin được sinh ra từ cơ sở dữ liệu. Có nghĩa là nó tự tìm kiếm, tạo mẫu và sinh ra tri thức chứ không phải bắt đầu với các tri thức đã biết trước. Các thông tin mà phương pháp này đem lại là các thông tin hay các tri thức cấp cao diễn tả về các đối tượng trong cơ sở dữ liệu. Phương pháp này liên quan đến việc tìm kiếm các mẫu trong cở sở dữ liệu (CSDL). Trong khai thác dữ liệu, quy nạp được sử dụng trong cây quyết định và tạo luật.

✓ **Cây quyết định và luật**

Cây quyết định: Là một mô tả tri thức dạng đơn giản nhằm phân các đối tượng dữ liệu thành một số lớp nhất định. Các nút của cây được gán nhãn là tên các thuộc tính, các cạnh được gán các giá trị có thể của các thuộc tính, các lá mô tả các

lớp khác nhau. Các đối tượng được phân lớp theo các đường đi trên cây, qua các cạnh tương ứng với các giá trị, thuộc tính của đối tượng tới lá.

Tạo luật: Các luật được tạo ra nhằm suy diễn một số mẫu dữ liệu có ý nghĩa về mặt thống kê. Các luật có dạng Nếu P thì Q, với P là mệnh đề đúng một phần trong CSDL, Q là mệnh đề dự đoán.

Cây quyết định và luật có ưu điểm là hình thức mô tả đơn giản, mô hình suy diễn khá dễ hiểu đối với người sử dụng. Tuy nhiên, giới hạn của nó là mô tả cây và luật chỉ có thể biểu diễn được một số dạng chức năng, vì vậy giới hạn về cả độ chính xác của mô hình.

✓ **Phát hiện các luật kết hợp**

Phương pháp này nhằm phát hiện ra các luật kết hợp giữa các thành phần dữ liệu trong cơ sở dữ liệu. Mẫu đầu ra của giải thuật khai thác dữ liệu là tập luật kết hợp tìm được. Ta có thể lấy một ví dụ đơn giản về luật kết hợp như sau: sự kết hợp giữa hai thành phần A và B có nghĩa là sự xuất hiện của A trong dòng kéo theo sự xuất hiện của B trong cùng dòng đó: $A \rightarrow B$.

Ví dụ chỉ có số ít người mua sách tiếng anh mà mua thêm đĩa CD. Số lượng các luật kết hợp trong một số cơ sở dữ liệu lớn gần như vô hạn. Do vậy thuật toán sẽ không thể phát hiện hết các luật và không phân biệt được luật nào là thông tin thực sự có giá trị và thú vị.

Vậy chúng ta đặt ra câu hỏi là luật kết hợp nào là thực sự có giá trị? Chẳng hạn ta có luật: Âm nhạc, ngoại ngữ, thể thao $\rightarrow$ CD, nghĩa là những người mua sách âm nhạc, ngoại ngữ, thể thao thì cũng mua đĩa CD. Lúc đó ta quan tâm đến số lượng trường hợp khách hàng thoả mãn luật này trong cơ sở dữ liệu hay độ hỗ trợ cho luật này. Độ hỗ trợ cho luật chính là phần trăm số dòng có cả sách âm nhạc, ngoại ngữ, thể thao và đĩa CD hay tất cả những người thích cả ba loại sách trên.

Tuy nhiên, giá trị hỗ trợ là không đủ. Có thể có trường hợp ta có một nhóm tương đối những người đọc cả ba loại sách trên nhưng lại có một nhóm với lượng lớn hơn những người thích sách thể thao, âm nhạc, ngoại ngữ mà không thích mua đĩa CD. Trong trường hợp này tính kết hợp rất yếu mặc dù độ hỗ trợ tương đối cao.

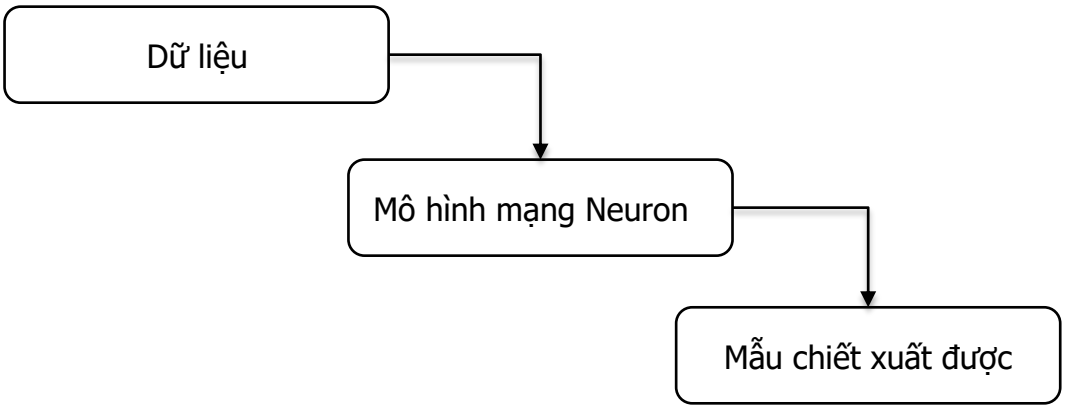
Như vậy chúng ta cần thêm một độ đo thứ hai đó là độ tin cậy (Confidence). Độ tin cậy là phần trăm các dòng có đĩa CD trong số các dòng có sách âm nhạc, thể thao, ngoại ngữ.

Nhiệm vụ của việc phát hiện các luật kết hợp là phải tìm tất cả các luật dạng $X \rightarrow B$ sao cho tần số của luật không nhỏ hơn độ hỗ trợ tối thiểu (minimum support - *minSup*) cho trước và độ tin cậy của luật không nhỏ hơn độ tin cậy tối thiểu (Min Confidence - *minConf*) cho trước. Từ một cơ sở dữ liệu ta có thể tìm được hàng nghìn và thậm chí hàng trăm nghìn các luật kết hợp.

✓ **Mạng Neuron**

Mạng Neuron là tiếp cận tính toán mới liên quan tới việc phát triển cấu trúc toán học và khả năng học. Các phương pháp là kết quả của việc nghiên cứu mô hình học của hệ thống thần kinh con người.

Mạng Neuron có thể đưa ra ý nghĩa từ các dữ liệu phức tạp hoặc không chính xác và có thể được sử dụng để chiết xuất các mẫu và phát hiện ra các xu hướng quá phức tạp mà con người cũng như các kỹ thuật máy tính khác không thể phát hiện được. Khi đề cập đến khai thác dữ liệu, người ta thường đề cập nhiều đến mạng Neuron. Tuy mạng Neuron có một số hạn chế gây khó khăn trong việc áp dụng và phát triển nhưng nó cũng có những ưu điểm đáng kể.



Hình 2.3: Thể hiện sơ đồ khai thác dữ liệu bằng mạng Neuron.

Một trong số những ưu điểm phải kể đến của mạng Neuron là khả năng tạo ra các mô hình dự đoán có độ chính xác cao, có thể áp dụng được cho rất nhiều loại

bài toán khác nhau, đáp ứng được nhiệm vụ đặt ra của khai thác dữ liệu như phân lớp, gom nhóm, mô hình hóa, dự báo các sự kiện phụ thuộc vào thời gian, v.v.

✓ **Giải thuật di truyền**

Giải thuật di truyền, nói theo nghĩa rộng là mô phỏng lại hệ thống tiến hóa trong tự nhiên, chính xác hơn đó là giải thuật chỉ ra tập các cá thể được hình thành, được ước lượng và biến đổi như thế nào? Ví dụ như xác định xem làm thế nào để lựa chọn các cá thể tạo giống và lựa chọn các cá thể nào sẽ bị loại bỏ. Giải thuật cũng mô phỏng lại yếu tố gen trong nhiễm sắc thể sinh học trên máy tính để có thể giải quyết nhiều bài toán thực tế khác nhau.

Giải thuật di truyền là một giải thuật tối ưu hóa. Nó được sử dụng rất rộng rãi trong việc tối ưu hóa các kỹ thuật khai thác dữ liệu trong đó có kỹ thuật mạng Neuron. Sự liên hệ của nó với các quá trình khai thác dữ liệu. Ví dụ như trong kỹ thuật cây quyết định, tạo luật. Như đã đề cập ở phần trước, các luật mô hình hóa dữ liệu chứa các tham số được xác định bởi các giải thuật phát hiện tri thức.

Giai đoạn tối ưu hóa là cần thiết để xác định xem các giá trị tham số nào tạo ra các luật tốt nhất. Vì vậy mà giải thuật di truyền đã được sử dụng trong các công cụ khai thác dữ liệu.

2.2. Khai thác luật kết hợp

2.2.1. Giới thiệu

Từ khi được giới thiệu từ năm 1993, bài toán khai thác luật kết hợp nhận được rất nhiều sự quan tâm của nhiều nhà khoa học. Ngày nay việc khai thác các luật như thế vẫn là một trong những phương pháp khai thác mẫu phổ biến nhất trong việc khám phá tri thức và khai thác dữ liệu (Knowledge Discovery and Data mining - KDD).

Mục đích chính của khai thác dữ liệu là các tri thức được kết xuất ra sẽ được sử dụng trong dự báo thông tin trợ giúp trong sản xuất kinh doanh và nghiên cứu khoa học.

Trong hoạt động sản xuất kinh doanh, ví dụ kinh doanh các mặt hàng tại siêu thị, các nhà quản lý rất thích có được các thông tin mang tính thống kê như: "90% phụ nữ có xem máy màu đỏ và đeo đồng hồ Thuỵ Sỹ thì dùng nước hoa hiệu Chanel" hoặc "70% khách hàng là công nhân khi mua TV thường mua loại TV 21 inches". Những thông tin như vậy rất hữu ích trong việc định hướng kinh doanh. Vậy vấn đề đặt ra là liệu có tìm được các luật như vậy bằng các công cụ khai thác dữ liệu hay không? Câu trả lời là hoàn toàn có thể. Đó chính là nhiệm vụ khai thác luật kết hợp.

2.2.2. Một số hướng tiếp cận trong khai thác luật kết hợp

Lĩnh vực khai thác luật kết hợp cho đến nay đã được nghiên cứu và phát triển theo nhiều hướng khác nhau. Có những đề xuất nhằm cải tiến tốc độ thuật toán, có những đề xuất nhằm tìm kiếm luật có ý nghĩa hơn... và có một số hướng chính sau đây:

- ✓ Luật kết hợp nhị phân (binary association rule hoặc Boolean association rule).
- ✓ Luật kết hợp tiếp cận theo hướng tập thô (mining association rules base on rough set).
- ✓ Luật kết nhiều mức (multi-level association rule).
- ✓ Luật kết hợp mờ (fuzzy association rule).
- ✓ Luật kết với thuộc tính được đánh trọng số (association rule with weighted items).
- ✓ Khai thác Luật kết hợp song song (parallel mining of association rules).

Bên cạnh những nghiên cứu về những biến thể của luật kết hợp, các nhà nghiên cứu còn chú trọng đề xuất những thuật toán nhằm tăng tốc quá trình tìm kiếm tập phổ biến từ cơ sở dữ liệu.

Ngoài ra, còn có một số hướng nghiên cứu khác về khai thác luật kết hợp như: khai thác luật kết hợp trực tuyến, khai thác luật kết hợp được kết nối trực tuyến đến các kho dữ liệu đa chiều (Multidimensional data, data warehouse) thông qua

công nghệ OLAP (Online Analysis Processing), MOLAP (multidimensional OLAP), ROLAP (Relational OLAP), ADO (Active X Data Object) for OLAP.v.v..

2.3. Khai thác luật phân lớp

2.3.1. Giới thiệu

Kho dữ liệu luôn chứa rất nhiều các thông tin hữu ích có thể dùng cho việc ra các quyết định liên quan đến điều hành, định hướng của một đơn vị, tổ chức. Phân lớp và dự đoán là hai dạng của quá trình phân tích dữ liệu được sử dụng để trích rút các mô hình biểu diễn các lớp dữ liệu quan trọng hoặc dự doán các dữ liệu phát sinh trong tương lai. Kỹ thuật phân tích này giúp cho chúng ta hiểu kỹ hơn về các kho dữ liệu lớn. Ví dụ chúng ta có thể xây dựng một mô hình phân lớp để xác định một giao dịch cho vay của ngân hàng là an toàn hay có rủi ro, hoặc xây dựng mô hình dự đoán để phán đoán khả năng chi tiêu của các khách hàng tiềm năng dựa trên các thông tin liên quan đến thu nhập của họ. Rất nhiều các phương pháp phân lớp và dự đoán được nghiên cứu trong các lĩnh vực máy học, nhận dạng mẫu và thông kê. Hầu hết các thuật toán đều có hạn chế về bộ nhớ với các giả định là kích thước dữ liệu đủ nhỏ. Kỹ thuật khai thác dữ liệu gần đây đã được phát triển để xây dựng các phương pháp phân lớp và dự đoán phù hợp hơn với nguồn dữ liệu có kích thước lớn.

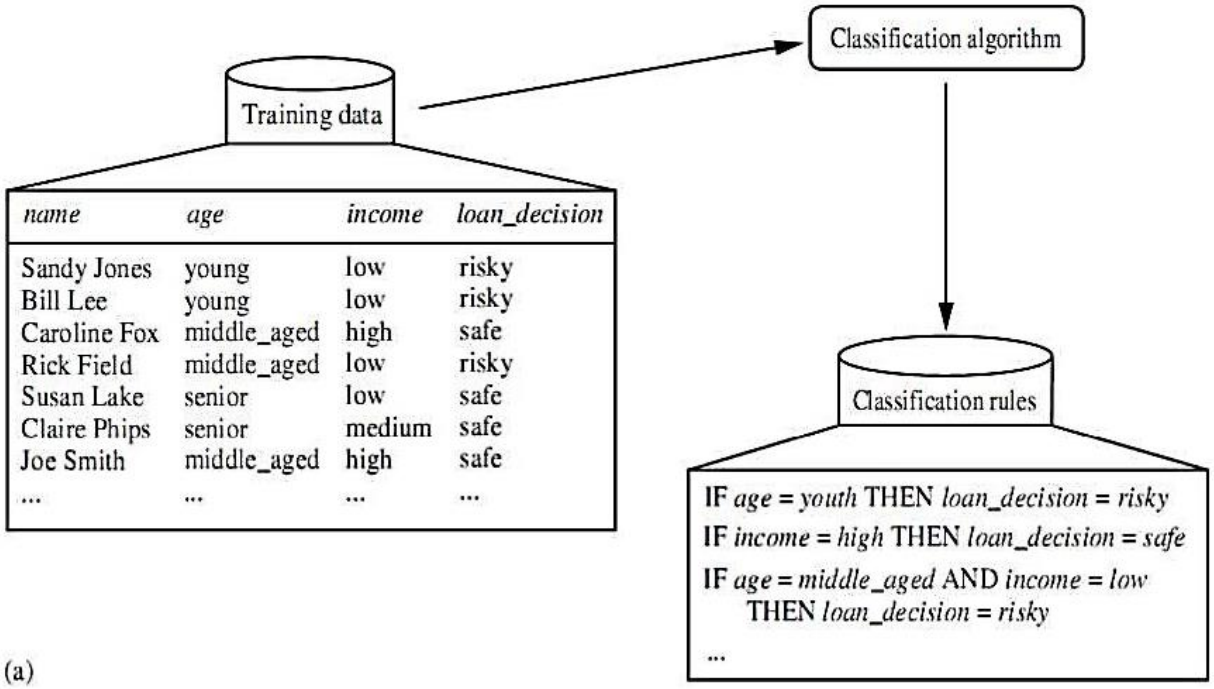
2.3.2. Quá trình phân lớp

Quá trình phân lớp thực hiện nhiệm vụ xây dựng mô hình các công cụ phân lớp giúp cho việc gán nhãn phân loại cho các dữ liệu. Ví dụ nhãn “An toàn” hoặc “Rủi ro” cho các yêu cầu vay vốn; “Có” hoặc “Không” cho các thông tin thị trường.... Các nhãn dùng phân loại được biểu diễn bằng các giá trị rời rạc trong đó việc sắp xếp trùng là không có ý nghĩa.

Phân lớp dữ liệu gồm hai quá trình:

- **Quá trình học:** Trong quá trình này một công cụ phân lớp sẽ được xây dựng để xem xét nguồn dữ liệu. Trong đó một thuật toán phân lớp được xây dựng bằng cách phân tích hoặc “học” từ tập dữ liệu huấn luyện được xây dựng sẵn bao gồm

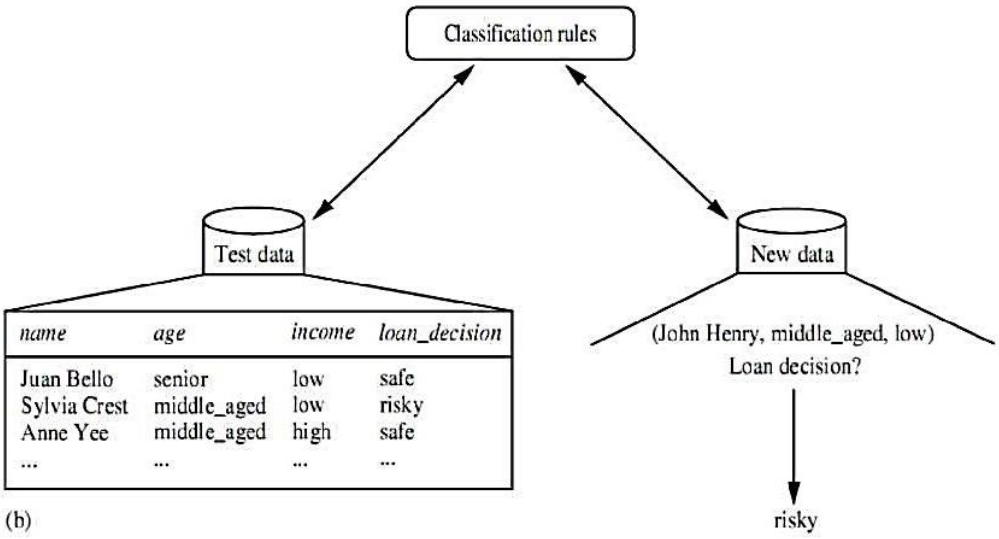
nhiều bộ dữ liệu. Một bộ dữ liệu X biểu diễn bằng một vector n chiều, X = (xl, x2,., xn) , đây là các giá trị cụ thể của một tập n thuộc tính của nguồn dữ liệu {Al, A2, ., An}. Mỗi bộ được giả sử rằng nó thuộc về một lớp được định nghĩa trước với các nhãn xác định.



Hình 2.4: Quá trình học [5]

Quá trình đầu tiên của phân lớp có thể được xem như việc xác định ánh xạ hoặc hàm y = f(X), hàm này có thể dự đoán nhãn y cho bộ X. Nghĩa là với mỗi lớp dữ liệu chúng ta cần học (xây dựng) một ánh xạ hoặc một hàm tương ứng.

❖ **Quá trình phân lớp:** Trong bước này, mô hình thu được sẽ được sử dụng để phân lớp. Để đảm bảo tính khách quan nên áp dụng mô hình này trên một tập kiểm thử hơn là làm trên tập dữ liệu huấn luyện ban dầu. Tính chính xác của mô hình phân lớp trên tập dữ liệu kiểm thử là số phần trăm các bộ dữ liệu kiểm tra được đánh nhãn đúng bằng cách so sánh chúng với các mẫu trong bộ dữ liệu huấn luyện.



Hình 2.5: Quá trình phân lớp [5]

Nếu như độ chính xác của mô hình dự đoán là chấp nhận được thì chúng ta có thể sử dụng nó cho các bộ dữ liệu với thông tin nhãn phân lớp chưa xác định.

2.3.3. Một số phương pháp phân lớp

- ✓ Phương pháp quy nạp cây quyết định.
- ✓ Phương pháp Bayesian.
- ✓ Phương pháp bằng mạng lan truyền ngược.
- ✓ Phương pháp dựa trên nguyên lý khai thác luật kết hợp.
- ✓ Một số phương pháp phân lớp khác.
  - Dựa trên khoảng cách (K-Nearest Neighbor).
  - Giải thuật di truyền (Genetic Algorithms).
  - Tiếp cân tập thô.
  - Tiếp cận tập mờ.
  - Suy luận dựa trên trường hợp (Case-based reasoning).

2.4. Khai thác luật phân lớp dựa vào luật kết hợp

Phân lớp đóng vai trò quan trọng trong việc hỗ trợ ra quyết định hệ thống. Nhiều phương pháp khai thác luật phân lớp được phát triển gồm C4.5 [11], ILA [15] và ILA-2 [14]. Gần đây, một phương pháp phân lớp từ khai thác dữ liệu, gọi là phân lớp dựa trên sự kết hợp (CBA) [1], được đề xuất cho việc khai thác phân lớp

dựa trên luật kết hợp (CARs). Phương pháp này có nhiều lợi ích hơn phương pháp heuristic và tham lam, trong đó các vấn đề trước có thể dễ dàng loại bỏ nhiễu, và độ chính xác cao hơn. Nó tạo ra một bộ luật hoàn chỉnh hơn C4.5 và ILA.

Trong khai thác luật kết hợp, thuộc tính đích (hoặc thuộc tính lớp) là không được xác định trước. Tuy nhiên, thuộc tính đích phải được xác định trước trong vấn đề phân lớp. Do đó, một số phương pháp nhằm nâng cao hiệu quả khai thác luật phân lớp dựa trên khai thác luật kết hợp được đề xuất như:

- ✓ Phân lớp dựa trên luật kết hợp dự đoán (CPAR) [19].
- ✓ Phân lớp dựa trên nhiều luật kết hợp [7].
- ✓ Phân lớp dựa trên sự kết hợp [6].
- ✓ Phân lớp dựa trên luật kết hợp đa lớp (MCAR) [12].
- ✓ Phân loại kết hợp dựa trên entropy tối đại [13].

Một số nghiên cứu cho thấy phân loại dựa trên luật kết hợp lớp chính xác hơn các phương pháp truyền thống như C4.5, ILA về cả lý thuyết và kết quả thực nghiệm. Veloso đề xuất phân lớp kết hợp lười [16,17].

Thuật toán di truyền cũng được áp dụng cho việc khai thác CARs, và một số phương pháp tiếp cận được đề xuất như:

- ✓ Phương pháp tiếp cận dựa trên GA để xây dựng bộ phân loại cho bộ dữ liệu số và áp dụng nó vào dữ liệu giao dịch chứng khoán [2].
- ✓ Phương pháp dựa trên GA mà không cần ngưỡng độ hỗ trợ tối thiểu hoặc độ tin cậy tối thiểu [10].

Ngoài ra, cũng có một số nghiên cứu cải tiến thuật toán khai thác luật phân lớp kết hợp được đề xuất như:

- ✓ Thuật toán khai thác hiệu quả luật phân lớp kết hợp CAR-Miner [8].
- ✓ Thuật toán cải tiến khai thác luật phân lớp kết hợp dùng sự khác nhau về bộ định danh các đối tượng chứa tập phổ biến [9].

2.5. Khai thác luật phân lớp kết hợp CAR-Miner

Trong phần này sẽ trình bày thuật toán CAR-Miner, một thuật toán được áp dụng để khai thác hiệu quả luật phân lớp kết hợp trên cơ sở dữ liệu tĩnh.

2.5.1. Một số khái niệm

Gọi D là tập các dữ liệu huấn luyện với n thuộc tính $A_1, A_2, \ldots, A_n$ và $|D|$ đối tượng (mẫu). Gọi $C = \{c1, c2, \ldots, ck\}$ là danh sách các nhãn lớp. Mỗi giá trị của thuộc tính Ai và thuộc tính lớp C được ký hiệu bởi các ký tự chữ viết thường a và c tương ứng.

Định nghĩa 2.1. Một itemset là một tập các cặp thuộc tính và giá trị, được ký hiệu bởi $\{(A_{i1}, a_{i1}), (A_{i2}, a_{i2}), \ldots, (A_{im}, a_{im})\}$.

Định nghĩa 2.2. Một luật phân lớp kết hợp r có dạng $\{(A_{i1}, a_{i1}),\ldots, (A_{im}, a_{im})\} \rightarrow c$, trong đó $\{(A_{i1}, a_{i1}),\ldots, (A_{im}, a_{im})\}$ là một itemset và $c \in C$ là một nhãn lớp.

Định nghĩa 2.3. Khả năng xảy ra của luật r trong tập dữ liệu D, kí hiệu $ActOcc(r)$, là số dòng trên D chứa vế trái của r.

Định nghĩa 2.4. Độ hỗ trợ của luật r, kí hiệu $Sup(r)$, là số dòng trên D chứa vế trái và vế phải của r.

Định nghĩa 2.5. Độ tin cậy của luật r là tỉ số giữa $Sup(r)$ chia cho $ActOcc(r)$, nghĩa là $\text{Conf}(r) = \frac{Sup(r)}{ActOcc(r)}$

Chẳng hạn, xét luật r: $\{(A, a1)\} \rightarrow y$ từ tập cơ sở dữ liệu Bảng 2.1, chúng ta có $ActOcc(r) = 3$ và $Sup(r) = 2$. Do có 3 đối tượng có giá trị $A = a1$, trong đó hai đối tượng có lớp là $y \Rightarrow$ Độ tin cậy của luật là $conf(r) = 2/3$.

2.5.2. Cấu trúc cây MECR

Cấu trúc cây MERC được chỉnh sửa từ cấu trúc cây ERC [18] với M là ký tự viết tắt của Modification. Trong cây MERC, mỗi nút trên cây chỉ chứa một itemset và bao gồm các thông tin sau:

- ✓ Itemset: (được nêu ở định nghĩa 2.1).
- ✓ Obidset: Là 1 mảng số nguyên chứa obidset identifiers của các dòng chứa itemset đó.
- ✓ (#c1, #c2, . . . , #ck): class count là 1 mảng số nguyên có kích thước bằng số lượng lớp trong dataset.

- ✓ pos: là một số nguyên lưu trữ vị trí của các lớp với số lần xuất hiện nhiều nhất, nghĩa là pos = arg max $\{\#c_i\}$ với i từ [1, k].

Ví dụ về tạo một cây MECR từ bảng CSDL huấn luyện mẫu:

Bảng 2.1: Một CSDL huấn luyện mẫu cho thuật toán Car-Miner

OID	A	B	C	Class
1	a1	b1	c1	y
2	a1	b2	c1	n
3	a2	b2	c1	n
4	a3	b3	c1	y
5	a3	b1	c2	n
6	a3	b3	c1	y
7	a1	b3	c2	y
8	a2	b2	c2	n

Xét một nút chứa itemset X = {(A, a3), (B, b3)} từ bảng 2.1. Do X chứa trong các OID là 4, 6 và tất cả đều thuộc lớp *y*. Vì vậy ta viết nút $\begin{matrix}\{(A,a3),(B,b3)\}\\ 46(\underline{2},0)\end{matrix}$ hay có thể viết đơn giản là $\begin{matrix}3 \text{ x } a3b3\\ 46(\underline{2},0)\end{matrix}$ được tạo ra trên cây. Trong đó:

- ✓ Itemset = { (A, a3), (B, b3) }
- ✓ Obidset = {4, 6}.
- ✓ Class count = {2, 0} (do có 2 lớp nên mảng có 2 giá trị, itemset X đều thuộc lớp *y* nên giá trị đầu của mảng là 2, còn giá trị sau là 0.
- ✓ pos = 1 (được gạch chân tại vị trí 1 của mảng class count của nút này) do số đếm thuộc về lớp *y* là lớn nhất (2 so với 0).

Trong thuật toán này, các tác giả đã dùng bit để lưu trữ đầy đủ tập các thuộc tính giúp tiết kiệm được bộ nhớ. Sử dụng cách biểu diễn bit như sau: thuộc tính A có giá trị là $2^0 = 1$, B có giá trị là $2^1 = 2$, C có giá trị là $2^2 = 4$ nên AB = $2^0 + 2^1 = 3$,

tương tự ABC = $2^0 + 2^1 + 2^2 = 7$). Ví dụ: AB sẽ được thay bằng 11 trong cách dùng bit đại diện dạng nhị phân và bằng 3 ở dạng thập phân.

2.5.3. Thuật toán khai thác hiệu quả cho CAR-Miner

Định lý 1: Cho trước hai nút $\begin{matrix} att_1 \text{ x } value_1 \\ Obidset_1(c_{11}, \ldots, c_{1k}) \end{matrix}$ và $\begin{matrix} att_2 \text{ x } value_2 \\ Obidset_2(c_{21}, \ldots, c_{2k}) \end{matrix}$,

nếu $att_1 = att_2$ và $value_1 \neq value_2$, thì $Obidset_1 \cap Obidset_2 = \emptyset$.

Chứng minh:

Khi $att_1 = att_2$ và $values_1 \neq values_2$, tồn tại một $val_1 \in values_1$ và $val_2 \in values_2$, val_1 và val_2 có thuộc tính giống nhau nhưng giá trị lại khác nhau. Như vậy, nếu một dòng với OID_i chứa val_1 nhưng không chứa val_2 thì $\forall OID \in Obidset_1 \Rightarrow OID \notin Obidset_2$. Do đó, $Obidset_1 \cap Obidset_2 = \emptyset$.

Theo vào định lý 1, tác giả chia một itemset thành att x values cho dễ sử dụng. Dựa vào định lý 1 → nếu có 2 itemset X và Y có cùng thuộc tính thì chúng sẽ không cần kết hợp với nhau thành itemset XY bởi vì $Sup(XY) = 0$. Ví dụ, xét hai nút

$\begin{matrix} 1 \text{ x } a_1 \\ 127(\underline{2},1) \end{matrix}$ và $\begin{matrix} 1 \text{ x } a_2 \\ 38(\underline{1},1) \end{matrix}$, trong đó $Obidset(\{(A, a1)\}) = 127$ và $Obidset(\{(A ,a2)\}) = 38$.

$\Rightarrow Obidset(\{(A, a1),(A, a2)\}) = Obidset(\{(A, a1)\}) \cap Obidset(\{(A, a2)\}) = \emptyset$.

Tương tự ta có $Obidset(\{(A, a1), (B, b1)\}) = 1$ và $Obidset(\{(A, a1), (B, b2)\}) = 2$.

$\Rightarrow$ Obidset($\{(A, a1), (B, b1)\}$) = $Obidset(\{(A, a1), (B, b2)\}) = \emptyset$ bởi vì cả hai itemset này có cùng thuộc tính AB nhưng khác giá trị.

Định lý 2: Cho trước hai nút $\begin{matrix} itemset_1 \\ Obidset_1(c_{11}, \ldots, c_{1k}) \end{matrix}$ và $\begin{matrix} itemset_2 \\ Obidset_2(c_{21}, \ldots, c_{2k}) \end{matrix}$,

nếu $itemset_1 \subset itemset_2$ và $|Obidset_1| = |Obidset_2|$ thì $\forall i \in [1, k]: c_{1i} = c_{2i}$.

Chứng minh:

Ta có: $itemset_1 \subset itemset_2$, có nghĩa là tất cả các dòng chứa trong $itemset_2$ cũng chứa trong $itemset_1$. Do đó $Obidset_2 \subseteq Obidset_1$. Hơn nữa, ta có $|Obidset_1| = |Obidset_2|$

$\Rightarrow \forall i \in [1, k]: c_{1i} = c_{2i}$.

Từ định lý 2, khi kết hai nút cha thành một nút con, có thể kiểm tra số phần tử của Obidset kết quả, nếu có kết quả bằng với một trong hai nút cha thì chúng ta không cần phải tính toán lại giá trị count và pos của nút này, vì thông tin của nút này sẽ giống với nút cha.

Bằng cách sử dụng hai định lý này sẽ giúp thuật toán CAR-Miner khai thác luật phân lớp kết hợp hiệu quả hơn. Với định lý 1, chúng ta không cần kết hai nút có cùng thuộc tính. Còn với định lý 2, chúng ta không cần phải tính toán lại thông tin cho một vài nút con.

2.5.4. Thuật toán CAR-Miner

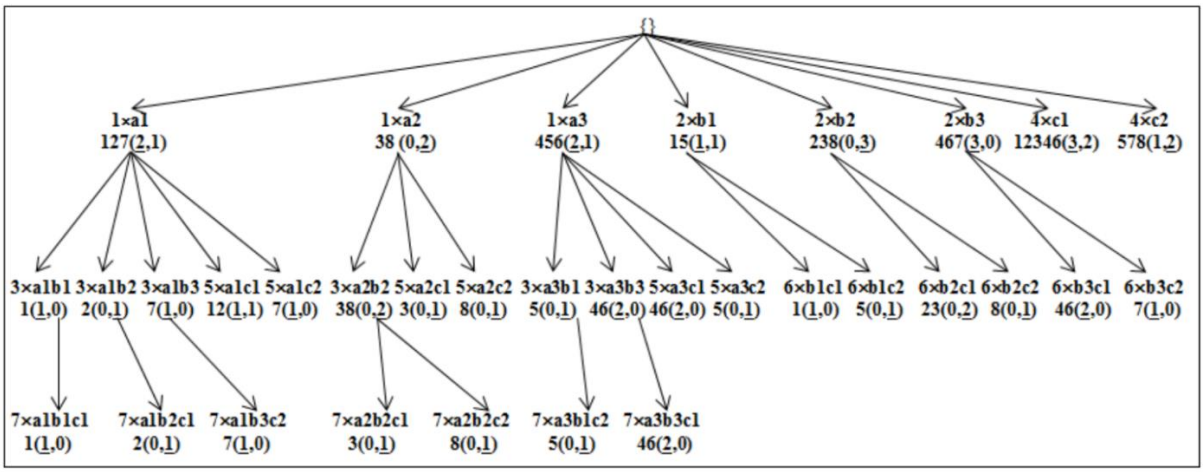
Đầu vào thuật toán CAR-Miner là nút gốc (*L*r) chứa các nút 1-itemset phổ biến. Thủ tục CAR-Miner được gọi để khai thác tất cả các luật phân lớp kết hợp với đầu vào là *L*r, *minSupCount* (là số đếm của *minSup*), *minConf*.

Thủ tục CAR-Miner xét mỗi nút l_i với các nút sau nó l_j trên *L*r sao cho j > i để tạo ra các nút con *l*. Thủ tục này cũng sử dụng định lý 1 và 2 để loại bỏ nhanh các nút con và tính nhanh các thông tin của các nút con.

Input: *A dataset D, minSupCount and minConf*
Output : *all CARs satisfy minSupCount and minConf*
Procedure :
CAR-Miner*(L_r, minSupCount, minConf)*
CARs = Ø
for all $l_j \in L_r$.children, with j > i do
if l_i.att ≠ l_j.att then *// Sử dụng định lý 1*
O.att = l_i.att ∪ l_j.att;
O.value = l_i.value ∪ l_j.value;
O.Obidset = l_i.*Obidset* ∩ l_j.*Obidset*;
if |*O*.*Obidset*| = |l_i.*Obidset*| then *// Sử dụng định lý 2*
O.count = l_i.count;
O.pos = l_i.pos;
else if |*O*.*Obidset*| = |l_j.*Obidset*| then *// Sử dụng định lý 2*
O.count = l_j.count;
O.pos = l_j.pos;
else
O.count = {count(x ∈ *O*.Obidset | class(x) = c_i, ∀i ∈ [1,k]};
O.pos = argmax {*l*.$count_i$};
$i \in [1,k]$
if *O*.count[*O*.pos] ≥ *minSupCount* then
$P_i = P_i \cup O$;
CAR-Miner(P_i, *minSupCount*, *minConf*)

ENUMERATE-CAR(*l*, *minConf*)
conf = *l*.count[*l*.pos] / |*l*.Obidset|;
if conf ≥ *minConf* then
CARs = CARs ∪ {*l*.itemset → c_{pos} (*l*.count[*l*.pos], conf)}

Hình 2.6: Thuật toán CAR-Miner



Hình 2.7: Cây MECR được xây dựng từ CSDL của bảng 2.1

Chương 3: PHƯƠNG PHÁP KHAI THÁC LUẬT PHÂN LỚP KẾT HỢP TRÊN DỮ LIỆU BỊ SỬA ĐỔI

3.1. Thuật toán khai thác luật kết hợp trên cơ sở dữ liệu bị sửa đổi [4]

3.1.1. Giới thiệu thuật toán

Trong quá khứ, các nhà nghiên cứu thường giả định cơ sở dữ liệu là dữ liệu tĩnh để đơn giản hóa việc khai thác dữ liệu. Vì vậy hầu hết các thuật toán cổ điển đều được xây dựng dựa trên cơ sở dữ liệu tĩnh.

Cheung và các đồng nghiệp của ông đã đề xuất một thuật toán khai thác cơ sở dữ liệu tăng dần gọi là FUP (Fast update algorithm), thuật toán dựa trên việc từng bước duy trì sinh luật kết hợp dựa trên các dữ liệu mới được chèn vào. Thuật toán FUP là một biến thể của thuật toán Apriori do Agrawal và Srikant đề xuất năm 1994, thuật toán này sử dụng kỹ thuật cắt tỉa cây từ thuật toán DHP (Direct Hashing and Pruning) do Park, Chen và Yu đề xuất năm 1997. Đầu tiên thuật toán đánh giá một tập là large-itemsets dựa trên các dữ liệu được thêm mới và so sánh chúng với tập large-itemsets được xây dựng từ cơ sở dữ liệu ban đầu. Theo kết quả so sánh, thuật toán FUP xác định xem có nên duyệt lại cơ sở dữ liệu ban đầu không, do đó tiết kiệm thời gian trong việc sinh luật kết hợp. Mặc dù thuật toán FUP thực sự có thể cải thiện hiệu suất khai thác cho cơ sở dữ liệu tăng dần nhưng ta vẫn phải quét từ cơ sở dữ liệu gốc khi cần thiết.

Có hai ngưỡng support được dùng là ngưỡng hỗ trợ trên S_U (upper support threshold) và ngưỡng hỗ trợ dưới S_L (lower support threshold) để giảm việc phải duyệt lại cơ sở dữ liệu ban đầu và tiết kiệm thời gian. Su thường sử dụng trong các thuật toán khai thác thông thường. Một itemset được coi là large-itemset nếu số lượng của tập phải lớn hơn ngưỡng S_U. Ngưỡng S_L được định nghĩa để một tập phải có số lượng thấp nhất là bằng ngưỡng S_L. Hai ngưỡng S_U và S_L được sử dụng như một bộ đệm trong thuật toán khai thác cơ sở dữ liệu tăng trưởng, chúng được dùng để làm giảm bớt sự dao động của tập itemsets từ lớn đến nhỏ và ngược lại.

Ngoài cơ sở dữ liệu tăng trưởng thì cơ sở dữ liệu bị sửa đổi cũng thường được thấy trong các ứng dụng thực tế. Cơ sở dữ liệu bị sửa đổi là cơ sở dữ liệu có các dữ liệu bị sửa đổi. Tương tự như thuật toán với cơ sở dữ liệu tăng trưởng, với cơ sở dữ liệu bị sửa đổi ta cũng dùng hai ngưỡng support là S_U và S_L, với số lượng tập itemsets bị sửa đổi lớn hơn một ngưỡng an toàn thì ta mới tiến hành duyệt lại cơ sở dữ liệu.

3.1.2. Cơ sở lý thuyết

Nếu số lượng các dòng bị sửa đổi quá nhỏ so với số lượng dòng ban đầu, thì một tập itemset là không phổ biến (không thuộc tập phổ biến và cũng không thuộc tập gần phổ biến) trong cơ sở dữ liệu ban đầu sẽ không thể là phổ biến trên toàn bộ cơ sở dữ liệu bị cập nhật sau khi có một vài dòng bị sửa đổi. Điều này được thể hiện trong định lý dưới đây.

Định lý 1: Cho S_L và S_U lần lượt là độ hỗ trợ ngưỡng dưới và ngưỡng trên, d và t lần lượt là số lượng dòng của cơ sở dữ liệu gốc và số lượng dòng bị sửa đổi. Nếu $t \leq (S_U - S_L)d$ thì một itemset được gọi là nhỏ (không bao giờ large và cũng không pre-large) trong cơ sở dữ liệu gốc và không large trên toàn bộ dữ liệu có cập nhật, nên các dòng bị sửa đổi không ảnh hưởng quá nhiều tới việc cập nhật lại cơ sở dữ liệu.

Chứng minh: Giả sử ta có $t \leq (S_U - S_L)d$:

$$t \leq (S_U - S_L)d$$

$$=> t \leq d\,S_U - dS_l$$

$$=> t + d\,S_L \leq dS_U$$

$$=> \frac{t+dS_L}{d} \leq S_U.$$

Nếu một itemset I là không phổ biến trong tập cơ sở dữ liệu gốc D, thì $S^D(I)$ sẽ nhỏ hơn Sl * d. Vì thế:

$$S^D(I) < dS_l$$

Số lượng tối đa của I từ itemset khác ID là t (số dòng bị sửa đổi T). Như vậy:

$$t \geq S^{ID}(I)$$

Toàn bộ tỷ lệ độ hỗ trợ của I trong cơ sở dữ liệu được cập nhật U là $S^U(I)/d$, có thể được tiếp tục mở rộng thành:

$$\frac{S^U(I)}{d} = \frac{S^D(I) + S^{ID}(I)}{d} < \frac{dS_l + t}{d} \leqslant S_u$$

Như vậy I không phổ biến so với toàn bộ cơ sở dữ liệu mới được cập nhật. Điều này được chứng minh.

Kết luận 1: Cho r là tỷ lệ của t (số dòng bị sửa đổi) so với d (toàn bộ số dòng). Nếu r ≤ (Su – Sl) thì một itemset là không phổ biến trong cơ sở dữ liệu gốc và là không phổ biến cho toàn bộ cơ sở dữ liệu được cập nhật.

3.1.3. Các bước thực hiện

Đầu vào thuật toán: Hai ngưỡng support là S_U và S_L, một tập phổ biến lớn large-itemsets từ cơ sở dữ liệu gốc của D mẫu, và một tập t mẫu được sửa đổi.

Đầu ra của thuật toán: Tập hợp luật kết hợp cho cơ sở dữ liệu bị sửa đổi.

Các bước thực hiện thuật toán:

Bước 1: Tính giá trị ngưỡng an toàn *f* cho các mẫu bị sửa đổi theo công thức:

$$f = (S_U - S_L) \times |D|$$

Bước 2: Tìm dòng khác ID từ các dòng bị sửa đổi.

Bước 3: Gán k = 1 với k là số lượng item trong itemsets đang được xử lý.

Bước 4: Rút ra các k-itemset khác biệt ID_k từ các dòng bị sửa đổi.

Bước 5: Phân loại tập itemsets trong ID_k thành: large, pre-large hoặc small trong cơ sở dữ liệu gốc.

Bước 6: Duyệt từng itemset I trong cả tập large k-itemsets L_k^D và trong tập k-itemset khác biệt ID_k và thực hiện:

- ✓ $S^U(I) = S^D(I) + S^{ID}(I)$
- ✓ Nếu $S^U(I) / d \geq S_U$ thì giữ *I* trong tập large itemsets và $S^D(I) = S^U(I)$. Ngược lại, nếu $S^U(I) / d \geq S_L$ thì xóa I trong large itemset và giữ nguyên trong pre-large itemsets, và $S^D(I) = S^U(I)$.
- ✓ Ngược lại, xóa I trong tập large itemsets.

Bước 7: Duyệt từng itemset I tồn tại trong cả pre-large k-itemsets P_k^D dữ liệu gốc và k-itemset khác biệt ID_k, thực hiện:

- ✓ $S^D(I) = S^D(I) + S^{ID}(I)$.
- ✓ Nếu $S^U(I)/d \geq S_u$, xóa I trong large itemset và giữ nguyên trong pre-large itemsets, và $S^D(I) = S^U(I)$.
- ✓ Ngược lại nếu $S^U(I)/d \geq S_l$, giữ I trong pre-large itemset và $S^D(I) = S^U(I)$.

Bước 8: Duyệt từng itemset I trong ID_k nhưng không nằm trong large itemset L_k^D dữ liệu gốc hay pre-large itemsets P_k^D, nếu I có một positive count difference, thì tiến hành thêm I vào tập rescan-set R (được dùng trong việc duyệt lại ở bước 11). Nếu không tiến hành bỏ qua I.

Bước 9: Gán k = k + 1.

Bước 10: Lặp lại bước 4 tới bước 9 cho đến khi không còn tập large hay pre-large itemsets nào được cập nhật nữa.

Bước 11: Nếu t + c ≤ f hay R null thì không làm gì cả, ngược lại tiến hành rescan lại cơ sở dữ liệu gốc tới khi xác định được tập itemsets trong rescan-set R là large hay pre-large.

Bước 12: Sửa đổi lại luật phân lớp kết hợp theo large itemsets.

Bước 13: Nếu t + c > f thì c = 0, nếu không c = t + c.

3.2. Thuật toán luật phân lớp kết hợp trên cơ sở dữ liệu tăng trưởng

Với cơ sở dữ liệu tăng dần ta cần sửa lại một chút thuật toán CAR-Miner ở phần 2.5 như sau:

Định lý 1: Cho trước hai nút l_1 và l_2 thuộc MECR-tree, giả sử l_1 là nút cha của l_2, nếu $Sup(l_1.itemset \rightarrow c_{l1}.pos) < minSup$ thì $Sup(l_2.itemset \rightarrow c_{l2}.pos) < minSup$.

Chứng minh: Nếu l_1 là nút cha của l_2, tức là $l_1.itemset \subset l_2.itemset$

=> tất cả *Obidsets* chứa $l_1.itemset$ thì cũng chứa $l_2.itemset$ hay $l_1.Obidset \supseteq l_2.Obidset$ => $\forall i$, $l_1.count_i \geq l_2.count_i$ hay $\max\{l_1.count_i\}_{i=1}^k \geq \max\{l_2.count_i\}_{i=1}^k$

=> $Sup(l_1.itemset \rightarrow c_{l1}.pos) \geq Sup(l_2.itemset \rightarrow c_{l2}.pos)$

Bởi vậy, nếu $Sup(l_1.itemset \rightarrow c_{l1}.pos) < minSup$ thì $Sup(l_2.itemset \rightarrow c_{l2}.pos) < minSup$.

Thuật toán Modified-CAR-Miner

```
Modified-CAR-Miner(Lr)
    CARs = Ø
    for all li ∈ Lr.children do
        GENERATE-CAR(li, SU, minConf)
        Pi = Ø
        for all lj ∈ Lr.children, with j > i do
            if li.att ≠ lj.att then
                O.att = li.att ∪ lj.att;
                O.values = li.values ∪ lj.values;
                O.Obidset = li.Obidset ∩ lj.Obidset;
                if |O.Obidset| = |li.Obidset| then
                    O.count = li.count;
                    O.pos = li.pos;
                else if |O.Obidset| = |lj.Obidset| then
                    O.count = lj.count;
                    O.pos = lj.pos;
                else
                    O.count = {count(x ∈ O.Obidset | class(x) = ci, ∀i ∈ [1,k]};
                    O.pos = argmax {l.counti};
                                i ∈ [1,k]
                    if O.count[O.pos] ≥ SL x |D| then
                        Pi = Pi ∪ O;
        Endfor
    Modified-CAR-Miner(Pi);

GENERATE-CAR(l, Su, minConf)
    if l.count[l.pos] ≥ SU x |D| then
        conf = l.count[l.pos] / |l.Obidset|;
        if conf ≥ minConf then
                CARs = CARs ∪ {l.itemset → cpos (l.count[l.pos], conf)};
```

Hình 3.1: Thuật toán Modified-CAR-Miner cho cơ sở dữ liệu tăng trưởng

Thuật toán Modified-CAR-Miner cho cơ sở dữ liệu tăng dần được thực hiện như sau:

Đầu vào: MECR-tree được tạo từ cơ sở dữ liệu gốc D với *L*r là nút root, dataset D' được thêm mới và hai ngưỡng *minSup* là S_U và S_L và *minConf*.

Đầu ra: Luật phân lớp kết hợp thoả mãn S_U và *minConf* của D + D'.

CAR-Incre()

if |D| = 0 then

Modified-CAR-Miner(D', S_L**)**

$f = \left\lfloor \frac{(S_U - S_L) \times |D'|}{1 - S_U} \right\rfloor$

else if |D'| > f then

Modified-CAR-Miner(D + D', S_L **)**

$f = \left\lfloor \frac{(S_U - S_L) \times |D| + |D'|}{1 - S_U} \right\rfloor$

else

Xoá Obidset của mỗi nút trên MECR-tree

Gọi **UPDATE-TREE**( ) để cập nhật MERC-tree

Gọi **GENERATE-RULE** để sinh luật từ MECR-tree

$f = f$ - |D'|

D = D + D'

UPDATE-TREE(L_r)

Cập nhật thông tin của các nút trên level đầu tiên của nút L_r như: Obidset, count, pos và đánh dấu chúng.

for all $_i \in L_r$.children do

if l_i is not marked then

TRAVERSE-TREE-TO-CHECK(l_i)

else if l_i.count[l_i.pos] < S_L x (|D| + |D'|) then

DELETE-TREE(l_i) *// theo định lý 1, xoá nút l_i và các nút con của nó*

else

for all $l_j \in L_r$.children, with j > i do

if l.att ≠ l_j.att and l_j is marked then

Gán O là nút con trực tiếp của 2 nút l_i và l_j;

if O has existed in the tree then

O.Obidset = l_i.Obidset ∩ l_j.*Obidset*;

if |*O.Obidset*| > 0 then

Cập nhật *O*.count dựa trên *O.Obidset*;

O.pos = argmax{*O*.count$_i$};

$i \in [1,k]$

if *O*.count[*O*.pos] ≥ S_L x (|D| + |D'|) then

Mark *O*;

UPDATE-TREE(l_i)

Hình 3.2: Mã giả hàm CAR-Incre và UPDATE-Tree.

```
TRAVERSE-TREE-TO-CHECK(l)
    if l.count[l.pos] < S_L x (|D| + |D'|) then
        DELETE-TREE(l);
    else
        for all l_i ∈ l.children do
            TRAVERSE-TREE-TO-CHECK(l_i);

DELETE-TREE(l)
    for all l_i ∈ l.children do
        DELETE-TREE(l_i);
    delete l;

GENERATE-RULES(L_r, S_U, minConf)
    for all l_i ∈ l.children do
        if l.count[l.pos] ≥ S_U x |D| then
            conf = l.count[l.pos] / Σ_{i=1}^{k} l.count[i];
            if conf ≥ minConf then
                CARs = CARs ∪ {l.itemset → c_pos (l.count[l.pos], conf)};
        GENERATE-RULES(l, S_U, minConf);
```

Hình 3.3: Mã giả hàm TRAVERSE-TREE-TO-CHECK, DELETE-TREE và GENERATE-RULES.

Ví dụ minh họa: Giả sử ta có cơ sở dữ liệu gốc với S_U = 25%, S_L = 12.5% và *minConf* = 50%.

Bảng 3.1: Bảng cơ sở dữ liệu mẫu

OID	A	B	C	Class
1	a1	b1	c1	y
2	a1	b1	c2	n
3	a1	b2	c2	n
4	a1	b3	c3	y
5	a2	b3	c1	n
6	a1	b3	c3	y
7	a2	b1	c3	y
8	a2	b2	c2	n

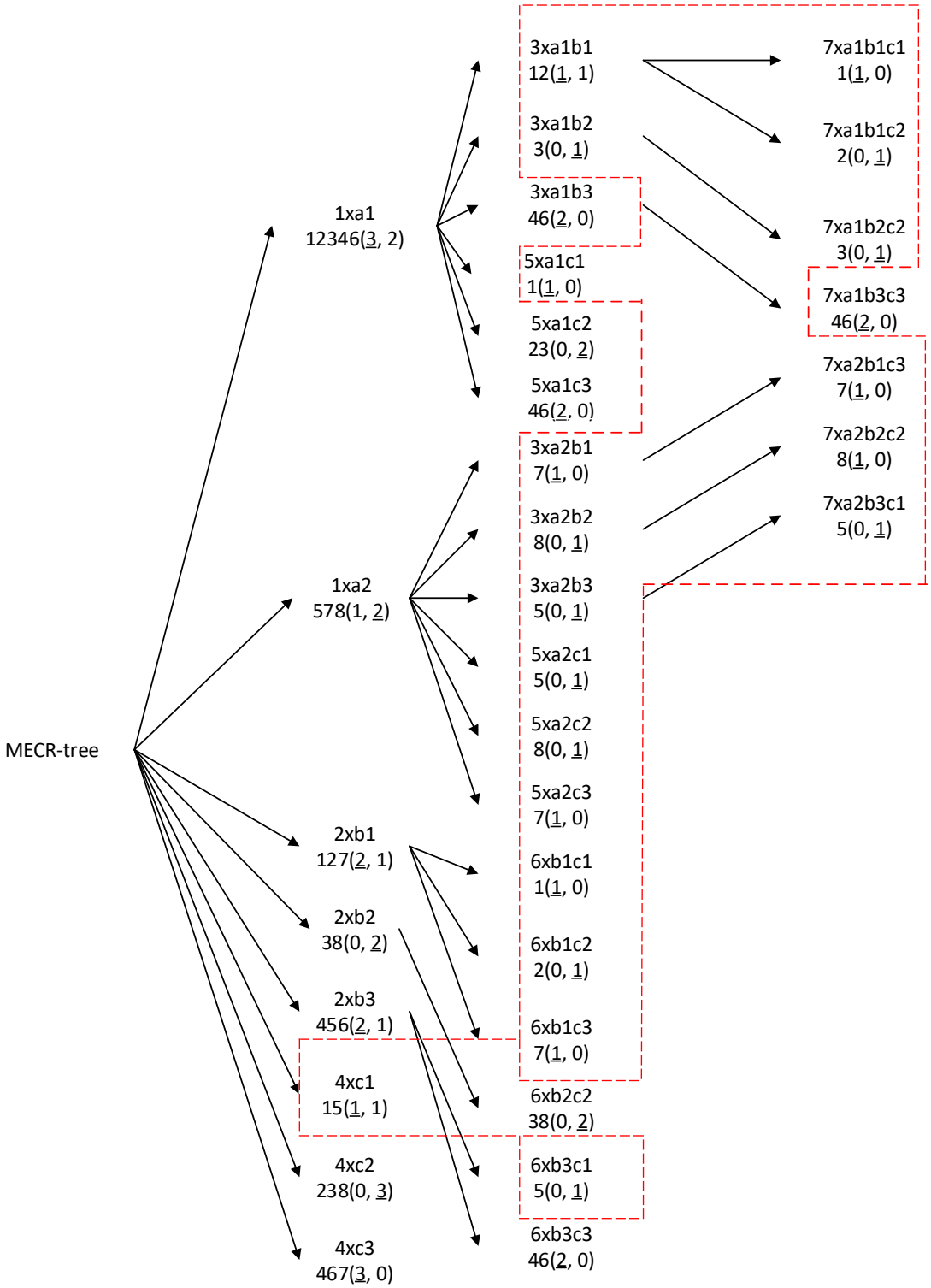
Ta thêm vào cơ sở dữ liệu mẫu một record có giá trị sau:

Bảng 3.2: Bảng cơ sở dữ liệu được thêm mới.

OID	A	B	C	Class
9	a1	b1	c2	y

Thực hiện thuật toán:

Bước 1: Xây dựng MECR-tree từ bảng cơ sở dữ liệu mẫu. Kết quả ta được MECR-tree với số nút 1-itemset là 8 như sau:



Hình 3.4: Cây MECR-tree được tạo từ cơ sở dữ liệu của bảng 3.1

Bảng 3.3: Bảng nút 1-itemset của MECR-tree

STT	Attribute	Obidset array	Class count array
1	a1	(1, 2, 3, 4, 6)	(3, 2)
2	a2	(5, 7, 8)	(1, 2)
3	b1	(1, 2, 7)	(2, 1)
4	b2	(3, 8)	(0, 2)
5	b3	(4, 5, 6)	(2, 1)
6	c2	(2, 3, 8)	(0, 3)
7	c3	(4, 6, 7)	(3,0)

Bước 2: Kiểm tra bảng cơ sở dữ liệu gốc có dữ liệu không. Nếu không có thì cập nhật bảng dữ liệu thêm mới vào bảng dữ liệu gốc rồi gọi hàm Modified-Car-Miner để sinh luật cho bảng này. Ngược lại thì thực hiện bước 3.

Bước 3: Tính giá trị ngưỡng an toàn của bảng cơ sở dữ liệu gốc bằng công thức:

$$f = \lfloor \text{case} * (S_U - S_L)/(1 - S_U) \rfloor$$

Trong đó:

- ✓ f: là giá trị ngưỡng an toàn của cơ sở dữ liệu gốc.
- ✓ case: tổng số record của cơ sở dữ liệu gốc. Ở đây là 8.
- ✓ $S_U = 25\% = 0.25$
- ✓ $S_L = 12.5\% = 0.125$

Vậy $f = \lfloor 8 * (0.25 - 0.125)/(1 - 0.25) \rfloor = 1$.

Bước 4: Kiểm tra số lượng dòng ở bảng cơ sở dữ liệu thêm mới nhỏ hơn giá trị ngưỡng an toàn không. Nếu nhỏ hơn thì tiến hành bước 6. Nếu không thì tiến hành bước 5.

Bước 5: Hợp bảng dữ liệu gốc và bảng dữ liệu thêm mới. Sinh MECR-tree mới cho bảng này rồi tiến hành gọi hàm Modified-Car-Miner.

Bước 6:

- ✓ Xoá toàn bộ Obidset của các nút 1-itemset trong MECR-tree ta được:

Bảng 3.4: Bảng nút 1-itemset của MECR-tree sau khi clear Obidset.

STT	Attribute	Obidset array	Class count array
1	a1	(Ø)	(3, 2)
2	a2	(Ø)	(1, 2)
3	b1	(Ø)	(2, 1)
4	b2	(Ø)	(0, 2)
5	b3	(Ø)	(2, 1)
6	c2	(Ø)	(0, 3)
7	c3	(Ø)	(3, 0)

- ✓ Cập nhật lại Obidset với bảng dữ liệu được thêm mới và đánh dấu các nút được thêm mới. Kết quả:

Bảng 3.5: Bảng nút 1-itemset của MECR-tree sau khi update Obidset, Class Count và Is Mark

STT	Attribute	Obidset array	Class count array	Is Mark
1	a1	(9)	(4, 2)	True
2	a2	(Ø)	(1, 2)	False
3	b1	(9)	(3, 1)	True
4	b2	(Ø)	(0, 2)	False
5	b3	(Ø)	(2, 1)	False
6	c2	(9)	(1, 3)	True
7	c3	(Ø)	(3, 0)	False

Bước 7: Duyệt qua các nút 1-itemset. Kiểm tra nút nào được bật cờ true thì giữ lại, còn nút nào không được bật cờ thì xóa toàn bộ nút đó cùng các nút con. Thực hiện sinh các nút con của các nút cha được bật cờ tại hàm Update-tree.

Bước 8: Gọi hàm sinh luật Generate-rules.

3.3. Thuật toán luật phân lớp kết hợp trên cơ sở dữ liệu bị xoá

3.3.1. Thuật toán CAR-Miner cho cơ sở dữ liệu bị xoá

CAR_Del()

Tính độ hỗ trợ an toàn

$$f = \left\lfloor \frac{(S_U - S_L) \times |D|}{1 - S_U} \right\rfloor$$

if |D'| <= f

Gọi hàm **UPDATE-TREE_LV1_DEL** để cập nhật lại các node 1-itemset của MECR-tree

Gọi hàm **UPDATE-TREE_LV_OTHER_DEL** để cập nhật lại các node từ 2-itemset trở đi của MECR-tree

Gọi hàm **GENERATE_RULES** để khai thác luật phân lớp kết hợp trên MECR-tree dùng S_U

else

Tạo MECR-tree mới với D + D'

Gọi hàm **Modified-CAR-Miner** để khai thác luật phân lớp kết hợp trên MECR-tree dùng S_L

D = D – D'

UPDATE-TREE_LV1_DEL(L_r)

Duyệt qua Dataset D'

Duyệt qua các node 1-itemset

for all $l_i \in L_r$.children do

Nếu li.values chứa item ϵ D' và li.att chứa thuộc tính tại vị trí item ϵ D' đang xét

Cập nhật thông tin của level đầu tiên của nút L_r như: Obidset, count, và pos.

l_i.Obidset xoá item

Cập nhật l_i.count

Cập nhật l_i.pos

Cập nhật l.mark = true

Hình 3.5: Mã giả hàm CAR_Del và UPDATE_TREE_LV1_DEL

```
UPDATE-TREE_LV_OTHER_DEL(Lr, SU, minConf)
for all  li ∈ Lr.children do
    for all lj ∈ Lr.children, with j > I do
        if (li.mark && lj.mark && li.att != lj.att)
            Duyệt node con của li.
            for all lk ∈ li.children do
            Nếu lk là node con được sinh từ li và lj thì
            Duyệt qua item ∈ D' cập nhật giá trị cho lk
                Cập nhật lk.Obidset
                Cập nhật lk.count
                Cập nhật lk.pos
                Cập nhật lk.mark = true
```

Hình 3.6: Mã giả hàm UPDATE-TREE_LV_OTHER_DEL

Dựa vào thuật toán khai thác luật phân lớp kết hợp cho dữ liệu tăng trưởng, ta áp dụng sửa đổi thuật toán CAR-Miner cho cơ sở dữ liệu bị xoá như sau:

Bước 1: Tìm kiếm các dòng bị xóa dữ liệu.

Bước 2: Kiểm tra số dòng bị sửa có lớn hơn giá trị ngưỡng an toàn f không. Nếu lớn hơn thì thực hiện bước 3. Nếu không thì thực hiện bước 5.

Bước 3: Xây dựng lại MECR-tree từ tập dữ liệu cũ và dữ liệu mới. Xong rồi thì thực hiện bước 4.

Bước 4: Gọi lại hàm Modified CAR-Miner để sinh luật. Hoàn thành thuật toán

Bước 5: Cập nhật lại toàn bộ 1-itemset của MECR-tree sử dụng hàm UPDATE_TREE_LV1_DEL.Xóa Obidset, cập nhật class count, pos tại những node có item thuộc D' bị xóa.

Bước 6: Cập nhật lại toàn bộ node con của các node 1-itemset sử dụng hàm UPDATE_TREE_LV_OTHER_DEL.Xóa Obidset, cập nhật class count, pos tại những node có item D' bị xóa.

Bước 7: Gọi hàm sinh luật Generate-rules.

3.3.2. Ví dụ minh họa về xóa dữ liệu

Với cơ sở dữ liệu tại bảng 2. Ta tiến hành xóa dòng 8

Bảng 3.6: Cơ sở dữ liệu bị xóa.

OID	Thuộc tính 1	Thuộc tính 2	Thuộc tính 3	Class
8	a2	b2	c2	n

Bước 1: Tính giá trị ngưỡng an toàn của bảng cơ sở dữ liệu gốc bằng công thức:

$$f = \lfloor \text{case} * (S_U - S_L)/(1 - S_U) \rfloor$$

Trong đó:

- ✓ f: là giá trị ngưỡng an toàn của cơ sở dữ liệu gốc.
- ✓ case: tổng số record của cơ sở dữ liệu gốc. Ở đây là 8.
- ✓ $S_U = 25\% = 0.25$
- ✓ $S_L = 12.5\% = 0.125$

Vậy $f = \lfloor 8 * (0.25 - 0.125)/(1 - 0.25) \rfloor = 1$.

Bước 2: Nếu bảng dữ liệu gốc có số lượng dòng bị xoá <= f thì tiến hành duyệt qua các nút 1-itemset. Xóa các itemset tại cơ sở dữ liệu bị xóa, cập nhật lại mảng class count, pos. Ngược lại, nếu số lượng dòng bị xoá lớn hơn f thì thực hiện khai thác lại luật trên cơ sở dữ liệu mới.

Từ hình 3.4 cây MECR-tree của bảng (dữ liệu gốc) ta tiến hành duyệt qua các nút 1-itemset để cập nhật. Các nút 1-itemset trong dữ liệu gốc bao gồm :

1xa1 12346(3, 2) | 1xa2 578(1, 2) | 2xb1 127(2, 1) | 2xb2 38(0, 2) | 2xb3 456(2, 1) | 4xc1 15(1, 1) | 4xc2 238(0, 3) | 4xc3 467(3, 0)

Ta lần lượt xóa Obidset và cập nhật lại count, pos của các itemset tại dòng OID thứ 8 ta được các nút 1-itemset sau:

1xa1 12346(3, 2) | 1xa2 57(1, 1) | 2xb1 127(2, 1) | 2xb2 3(0, 1) | 2xb3 456(2, 1) | 4xc1 15(1, 1) | 4xc2 23(0, 2) | 4xc3 467(3, 0)

Các nút được cập nhật sẽ bật cờ mark = true, ngược lại là false. Các nút không thỏa mãn count[pos] < 0.25 * 8 = 2 sẽ được đánh dấu isDel = true, ngược lại là false.

Kết quả ta được bảng 1-itemset dưới đây

Bảng 3.7: Bảng nút 1-itemset của MECR-tree sau khi cập nhật cơ sở dữ liệu bị xóa

STT	Attribute	Obidset array	Class count array	Mark	IsDel
1	a1	(1, 2, 3, 4, 6)	(3, 2)	False	False
2	a2	(5,7)	(1, 1)	**True**	**True**
3	b1	(1, 2, 7)	(2, 1)	False	False
4	b2	(3)	(0, 1)	**True**	**True**
5	b3	(4, 5, 6)	(2, 1)	False	False
6	c1	(1, 5)	(1, 1)	False	**True**
7	c2	(2, 3)	(0, 2)	**True**	False
8	c3	(4, 6, 7)	(3,0)	False	False

Bước 3: Tiếp theo ta cập nhật các nút từ 2-itemset trở đi.

- ✓ Nút 2-itemset bao gồm :

3xa1b1	3xa1b2	3xa1b3	5xa1c1	5xa1c2	5xa1c3	3xa2b1	3xa2b2	3xa2b3
12(1, 1)	3(0, 1)	46(2, 0)	1(1, 0)	23(0, 2)	46(2, 0)	7(1, 0)	8(0, 1)	5(0, 1)

5xa2c1	5xa2c2	5xa2c3	6xb1c1	6xb1c2	6xb1c3	6xb2c2	6xb3c1	6xb3c3
5(0, 1)	8(0, 1)	7(1, 0)	1(1, 0)	2(0, 1)	7(1, 0)	38(0, 2)	5(0, 1)	46(2, 0)

Ta duyệt các nút 1-itemset với 2 vòng lặp. Vòng lặp i từ 0 đến hết và vòng lặp j từ i+1 đến hết. Ta tiến hành kiểm tra nếu nút li và l_j đều có mark = true và l_i.att # l_j.att thì tiến hành duyệt nút con của l_i. Nếu nút con nào của li được sinh từ l_i và l_j thì tiến hành cập nhật Obidset, class count, pos. Khi đó các nút 2-itemset được cập nhật lại như sau :

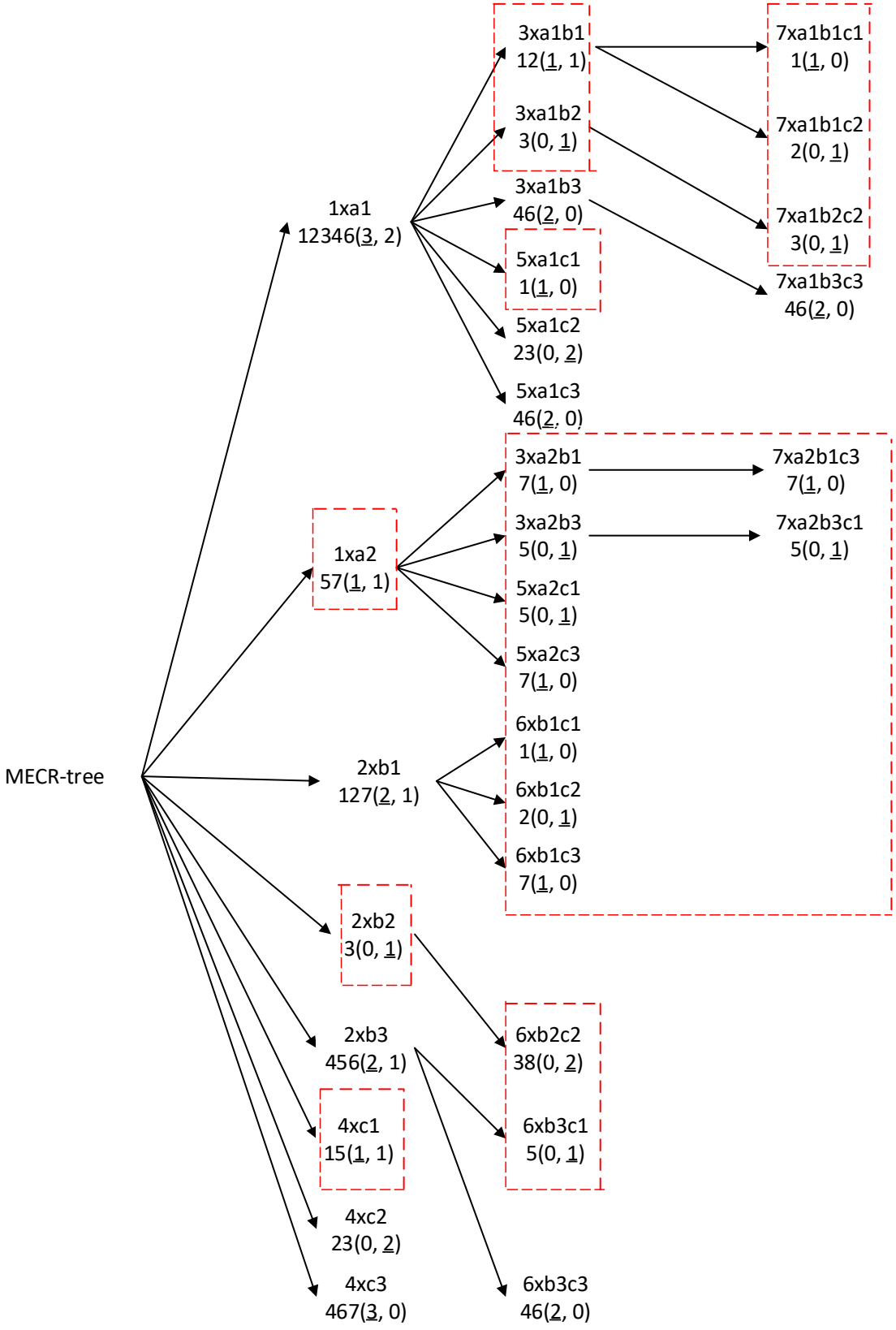
3xa1b1	3xa1b2	3xa1b3	5xa1c1	5xa1c2	5xa1c3	3xa2b1	3xa2b2	3xa2b3
12(1, 1)	3(0, 1)	46(2, 0)	1(1, 0)	23(0, 2)	46(2, 0)	7(1, 0)	Ø(0, 0)	5(0, 1)

5xa2c1	5xa2c2	5xa2c3	6xb1c1	6xb1c2	6xb1c3	6xb2c2	6xb3c1	6xb3c3
5(0, 1)	Ø(0, 0)	7(1, 0)	1(1, 0)	2(0, 1)	7(1, 0)	3(0, 1)	5(0, 1)	46(2, 0)

Các nút được cập nhật sẽ bật cờ mark = true, ngược lại là false. Các nút không thỏa mãn count[pos] < 0.25 * 8 = 2 sẽ được đánh dấu isDel = true, ngược lại là false.

- ✓ Tương tự với các nút 3-itemset.

Bước 4: Gọi hàm GENERATE_RULES để sinh luật.



Hình 3.6: Cây MERC-tree được cập nhật sau khi bị xoá

3.4. Thuật toán khai thác luật phân lớp kết hợp cho dữ liệu bị sửa đổi

3.4.1. Thuật toán CAR-Miner cho cơ sở dữ liệu bị sửa đổi

Dựa vào thuật toán ở mục 3.2 - thuật toán khai thác luật phân lớp kết hợp cho dữ liệu tăng trưởng, ta áp dụng sửa đổi thuật toán CAR-Miner như sau:

Bước 1: Tìm kiếm các dòng bị xóa dữ liệu.

Bước 2: Kiểm tra số dòng bị sửa có lớn hơn giá trị ngưỡng an toàn f không. Nếu lớn hơn thì thực hiện bước 3. Nếu không thì thực hiện bước 5.

Bước 3: Xây dựng lại MECR-tree từ tập dữ liệu cũ và dữ liệu mới. Xong rồi thì thực hiện bước 4.

Bước 4: Gọi lại hàm Modified-Car-Miner để sinh luật. Hoàn thành thuật toán.

Bước 5: Cập nhật lại toàn bộ 1-itemset của MECR-tree sử dụng hàm UPDATE_TREE_LV1_DEL. Xóa Obidset, cập nhật class count, pos tại những nút có item thuộc D bị xóa.

Bước 6: Cập nhật lại toàn bộ node con của các node 1-itemset sử dụng hàm UPDATE_TREE_LV_OTHER_DEL. Xóa Obidset, cập nhật class count, pos tại những node có item D bị xóa.

Bước 7: Gọi hàm CAR_Incre() với MECR-tree đã xóa dòng được cập nhật và dataset D’ là dòng được sửa mới.

CAR_ Modified ()

Tính độ hỗ trợ an toàn

$$f = \left\lfloor \frac{(S_U - S_L) \times |D|}{1 - S_U} \right\rfloor$$

if |D'| <= f

Gọi hàm **UPDATE-TREE_LV1_DEL** để xóa các item ở các node 1-itemset của MECR-tree

Gọi hàm **UPDATE-TREE_LV_OTHER_DEL** để xóa các item ở các node từ 2-itemset trở đi của MECR-tree

Gọi **CLEAR_OBIDSET()** xoá Obidset của mỗi nút trên MECR-tree

Gọi **UPDATE-TREE_LV1**( ), **UPDATE_TREE_LV_OTHER(), UPDATE_TREE()** để cập nhật MERC-tree

Gọi **GENERATE-RULE** để sinh luật từ MECR-tree

Gọi hàm **GENERATE_RULES** để khai thác luật phân lớp kết hợp trên MECR-tree dùng S_U

else

Tạo MECR-tree mới với D + D'

Gọi hàm **Modified-CAR-Miner** để khai thác luật phân lớp kết hợp trên MECR-tree dùng S_L

D = D – D'

Hình 3.7: Mã giả hàm CAR_Modified

3.4.2. Ví dụ minh họa về sửa dữ liệu

Giả sử ta có cơ sở dữ liệu gốc tại bảng 2 với S_U = 50%, S_L = 25% và *minConf* = 50%.

Ta tiến hành sửa dòng 8 như sau:

Bảng 3.8: Cơ sở dữ liệu bị sửa

OID	Thuộc tính 1	Thuộc tính 2	Thuộc tính 3	Class
8	a2	b3	c2	n

Bước 1: Tính giá trị ngưỡng an toàn của bảng cơ sở dữ liệu gốc bằng công thức:

$$f = \lfloor \text{case} * (S_U - S_L)/(1 - S_U) \rfloor$$

Trong đó:

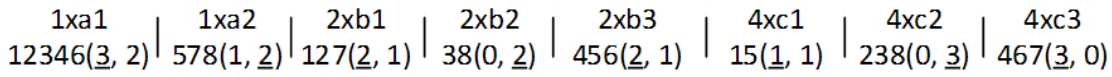
- ✓ *f*: là giá trị ngưỡng an toàn của cơ sở dữ liệu gốc.

- ✓ case: tổng số record của cơ sở dữ liệu gốc. Ở đây là 8.
- ✓ S_U = 25% = 0.25
- ✓ S_L = 12.5% = 0.125

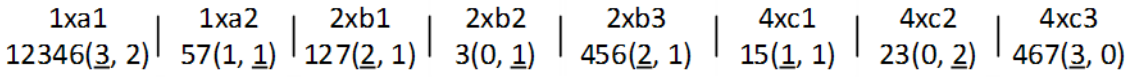
Vậy $f = \lfloor 8 * (0.25 - 0.125)/(1 - 0.25) \rfloor = 1$.

Bước 2: Nếu bảng dữ liệu bị sửa có số dòng <= f thì tiến hành duyệt qua các nút 1-itemset. Cập nhật các obidset tại cơ sở dữ liệu bị sửa, cập nhật lại mảng class count. Nếu nhỏ hơn thì khai thác luật trên cơ sở dữ liệu mới.

Từ hình 3.4 cây MECR-tree của bảng (dữ liệu gốc) ta tiến hành duyệt qua các nút 1-itemset để cập nhật. Các nút 1-itemset trong dữ liệu gốc bao gồm :



Ta lần lượt xóa Obidset và cập nhật lại count, pos của các itemset tại dòng OID thứ 8 ta được các nút 1-itemset sau:



Các nút được cập nhật sẽ bật cờ mark = true, ngược lại là false. Các nút không thỏa mãn count[pos] < 0.25 * 8 = 2 sẽ được đánh dấu isDel = true, ngược lại là false

Kết quả ta được bảng 1-itemset dưới đây

Bảng 3.9: Bảng nút 1-itemset của MECR-tree sau khi cập nhật cơ sở dữ liệu bị xóa

STT	Attribute	Obidset array	Class count array	Mark	IsDel
1	a1	(1, 2, 3, 4, 6)	(3, 2)	False	False
2	a2	(5,7)	(1, 1)	**True**	**True**
3	b1	(1, 2, 7)	(2, 1)	False	False
4	b2	(3)	(0, 1)	**True**	**True**
5	b3	(4, 5, 6)	(2, 1)	False	False
6	c1	(1, 5)	(1, 1)	False	**True**
7	c2	(2, 3)	(0, 2)	**True**	False
8	c3	(4, 6, 7)	(3,0)	False	False

Bước 3: Tiếp theo ta cập nhật các nút từ 2-itemset trở đi.

- ✓ **Nút 2-itemset bao gồm**

3xa1b1	3xa1b2	3xa1b3	5xa1c1	5xa1c2	5xa1c3	3xa2b1	3xa2b2	3xa2b3
12(1, 1)	3(0, 1)	46(2, 0)	1(1, 0)	23(0, 2)	46(2, 0)	7(1, 0)	8(0, 1)	5(0, 1)

5xa2c1	5xa2c2	5xa2c3	6xb1c1	6xb1c2	6xb1c3	6xb2c2	6xb3c1	6xb3c3
5(0, 1)	8(0, 1)	7(1, 0)	1(1, 0)	2(0, 1)	7(1, 0)	38(0, 2)	5(0, 1)	46(2, 0)

Ta duyệt các nút 1-itemset với 2 vòng lặp. Vòng lặp i từ 0 đến hết và vòng lặp j từ i+1 đến hết. Ta tiến hành kiểm tra nếu nút li và lj đều có mark = true và li.att # lj.att thì tiến hành duyệt nút con của li. Nếu nút con nào của li được sinh từ li và lj thì tiến hành cập nhật Obidset, class count, pos. Khi đó các nút 2-itemset được cập nhật lại như sau :

3xa1b1	3xa1b2	3xa1b3	5xa1c1	5xa1c2	5xa1c3	3xa2b1	3xa2b2	3xa2b3
12(1, 1)	3(0, 1)	46(2, 0)	1(1, 0)	23(0, 2)	46(2, 0)	7(1, 0)	Ø(0, 0)	5(0, 1)

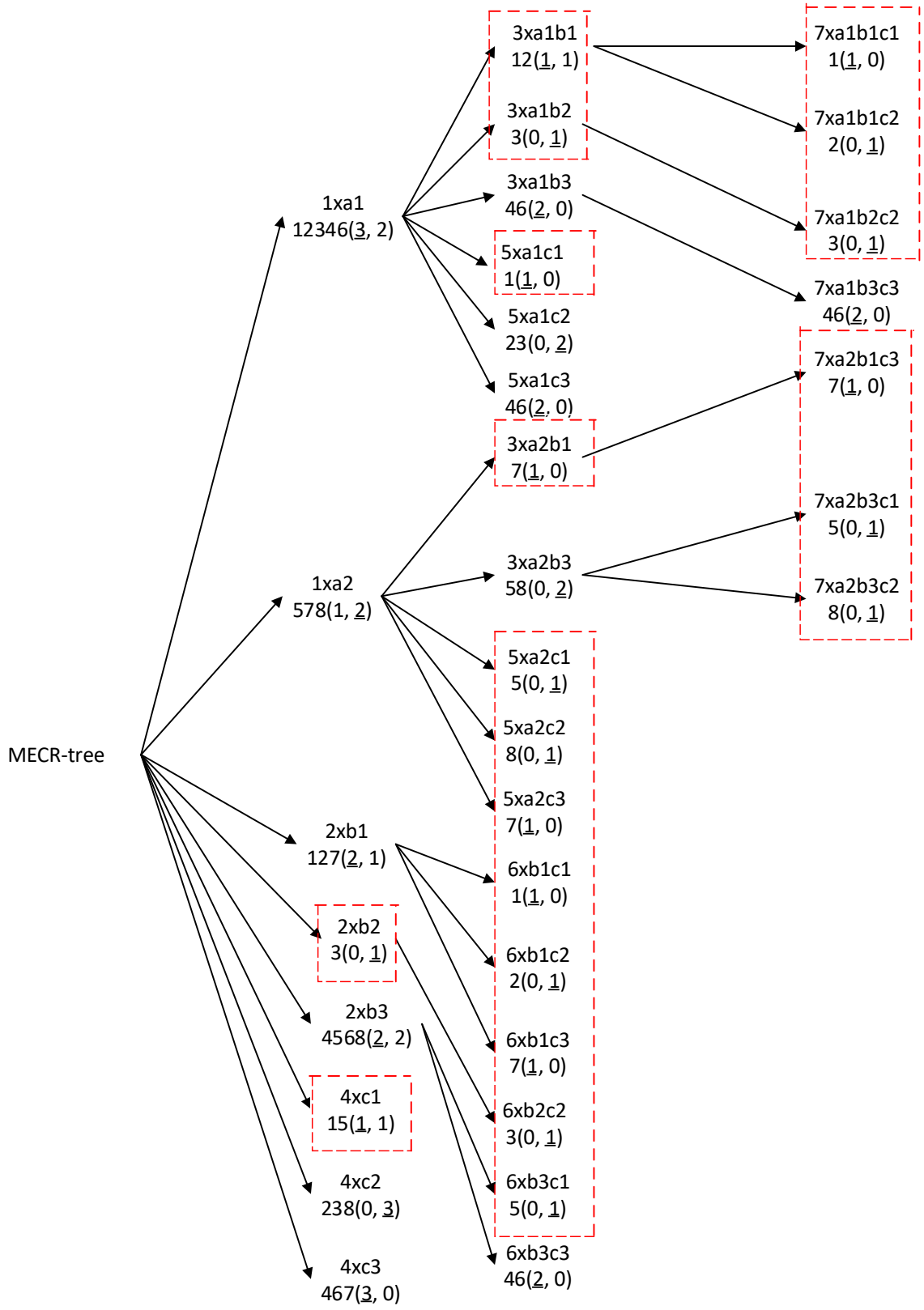
5xa2c1	5xa2c2	5xa2c3	6xb1c1	6xb1c2	6xb1c3	6xb2c2	6xb3c1	6xb3c3
5(0, 1)	Ø(0, 0)	7(1, 0)	1(1, 0)	2(0, 1)	7(1, 0)	3(0, 1)	5(0, 1)	46(2, 0)

Các nút được cập nhật sẽ bật cờ mark = true, ngược lại là false. Các nút không thỏa mãn count[pos] < 0.25 * 8 = 2 sẽ được đánh dấu isDel = true, ngược lại là false.

- ✓ **Tương tự với các nút 3-itemset**

Bước 4 : Tiếp theo ta sử dụng lại hàm CAR_Incre() với MECR-tree đã được cập nhật ở bước 2 và bước 3.

Bước 5: Gọi hàm GENERATE_RULES để sinh luật.



Hình 3.8: Cập nhật cây MECR sau khi dữ liệu bị sửa

Chương 4: KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1. Cơ sở dữ liệu thực nghiệm và môi trường xây dựng

4.1.1. Cơ sở dữ liệu thực nghiệm

Các kết quả thực nghiệm được thực thi trên các CSDL được lấy từ UCI Machine Learning Repository (http://archive. ics. uci. edu/ml/). Bảng dưới đây sẽ mô tả đặc điểm của các CSDL thực nghiệm.

Bảng 4.1: Đặc điểm của các CSDL thực nghiệm

CSDL (gốc)	Số thuộc tính	Số lớp	Số giá trị phân biệt	Mẫu
Breast	10	2	737	699
Lymph	18	4	63	148
Iris	5	3	21	150

Các CSDL có đặc điểm khác nhau: Breast, chứa nhiều thuộc tính và giá trị phân biệt nhưng ít mẫu. Đặc biệt, Lymph có số mẫu khá ít. Ta sẽ lần lượt chạy thuật toán với số mẫu dữ liệu bị sửa đổi có tỉ lệ lần lượt là 20%, 40%, 60%, 80% và 100% mẫu bị sửa đổi.

4.1.2. Môi trường xây dựng thuật toán

- ✓ Môi trường xây dựng: Windows 10, 64 bit.
- ✓ Môi trường phát triển: Visual studio 2013.
- ✓ Ngôn ngữ phát triển: C#.
- ✓ Cấu hình máy tính thực nghiệm: Intel Core i3, RAM 4GB

4.2. Kết quả thực nghiệm

4.2.1. Kết quả thực nghiệm trên cơ sở dữ liệu Breast

Chọn S_U = 25%, S_L = 12.5%, *minConf* = 50.

Ta cho sửa 1/5 mẫu dữ liệu gốc từ cơ sở dữ liệu ban đầu của dữ liệu Breast (699 mẫu). Sau đó, ta lần lượt chạy thuật toán với số lượng mẫu bị sửa đổi.

- ✓ Số mẫu bị sửa đổi: $\lceil 1/5 \times 699 \rceil = 140$.
- ✓ 20% mẫu bị sửa đổi = 20% * 140 mẫu = 28 mẫu.
- ✓ 40% mẫu bị sửa đổi = 40% * 140 mẫu = 56 mẫu.
- ✓ 60% mẫu bị sửa đổi = 60% * 140 mẫu = 84 mẫu.
- ✓ 80% mẫu bị sửa đổi = 80% * 140 mẫu = 112 mẫu.
- ✓ 100% mẫu bị sửa đổi = 100% * 140 mẫu = 140 mẫu.

Thu được kết quả như sau:

Bảng 4.2: Bảng kết quả thực nghiệm trên cơ sở dữ liệu Breast

Số dòng dữ liệu bị sửa đổi	**CAR-Miner (s)**	**Car- Modified (s)**
28	1.520	0.982
56	1.382	1.238
84	1.413	1.185
112	1.525	1.397
140	1.249	1.254

4.2.2. Kết quả thực nghiệm trên cơ sở dữ liệu Lymph

Chọn S_U = 25%, S_L = 12.5%, *minConf* = 50.

Ta cho sửa 1/5 mẫu dữ liệu gốc từ cơ sở dữ liệu ban đầu của dữ liệu Lymph (148 mẫu). Sau đó, ta lần lượt chạy thuật toán với số lượng mẫu bị sửa đổi.

- ✓ Số mẫu bị sửa đổi: $\lceil 1/5 \times 148 \rceil = 30$.
- ✓ 20% mẫu bị sửa đổi = 20% * 30 mẫu = 6 mẫu.
- ✓ 40% mẫu bị sửa đổi = 40% * 30 mẫu = 12 mẫu.
- ✓ 60% mẫu bị sửa đổi = 60% * 30 mẫu = 18 mẫu.
- ✓ 80% mẫu bị sửa đổi = 80% * 30 mẫu = 24 mẫu.
- ✓ 100% mẫu bị sửa đổi = 100% * 30 mẫu = 30 mẫu.

Thu được kết quả như sau:

Bảng 4.3: Bảng kết quả thực nghiệm trên cơ sở dữ liệu Lymph

Số dòng dữ liệu bị sửa đổi	CAR-Miner (s)	Car- Modified (s)
6	11.587	11.214
12	12.625	11.965
18	11.745	11.115
24	12.687	12.056
30	12.785	12.792

4.2.3. Kết quả thực nghiệm trên cơ sở dữ liệu Iris

Chọn S_U = 25%, S_L = 12.5%, *minConf* = 50.

Ta cho sửa 1/5 mẫu dữ liệu gốc từ cơ sở dữ liệu ban đầu của dữ liệu Iris (150 mẫu). Sau đó, ta lần lượt chạy thuật toán với số lượng mẫu bị sửa đổi.

- ✓ Số mẫu bị sửa đổi: $\lceil 1/5 \times 150 \rceil = 30$
- ✓ 20% mẫu bị sửa đổi = 20% * 30 mẫu = 6 mẫu.
- ✓ 40% mẫu bị sửa đổi = 40% * 30 mẫu = 12 mẫu.
- ✓ 60% mẫu bị sửa đổi = 60% * 30 mẫu = 18 mẫu.
- ✓ 80% mẫu bị sửa đổi = 80% * 30 mẫu = 24 mẫu.
- ✓ 100% mẫu bị sửa đổi = 100% * 30 mẫu = 30 mẫu.

Thu được kết quả như sau:

Bảng 4.4: Bảng kết quả thực nghiệm trên cơ sở dữ liệu Iris

Số dòng dữ liệu bị sửa đổi	CAR-Miner (s)	Car- Modified (s)
6	0.575	0.566
12	0.785	0.701
18	0.654	0.633
24	0.698	0.674
30	0.712	0.706

4.2.4. Khảo sát kết quả thực nghiệm

Dựa vào kết quả thực nghiệm trên 3 CSDL Breast, Iris và Lymph, ta nhận thấy thời gian thực thi của CAR-Modified chạy nhanh hơn thời gian thực thi của CAR-Miner vì CAR-Miner phải duyệt lại toàn bộ cơ sở dữ liệu khi bị thay đổi. Như kết quả thực nghiệm phía trên khi ta sửa đổi 28 dòng dữ liệu của cơ sở dữ liệu Breast, nếu dùng thuật toán Car-Miner thì ta phải mất 1.520 giây để khai thác luật, trong khi đó nếu ta dùng thuật toán Car-Modified thì chỉ mất 0.982 giây để khai thác luật. Vì thuật toán Car-Miner ta phải khai thác luật lại từ đầu, còn thuật toán Car-Modified thì chỉ phải khai thác lại luật dựa trên những thuộc tính bị ảnh hưởng khi sửa đổi dữ liệu. Tương tự, ta cũng sửa 6 dòng trên CSDL Lymph, nếu dùng thuật toán Car-Miner thì mất 11.587 giây, còn nếu dùng thuật toán Car-Modified thì ta chỉ mất 11.214 giây để khai thác luật. Tương tự như những kết quả thực nghiệm khác trên 3 CSDL Breast, Lymph và Iris được thống kê trong các phần trên, ta thấy thuật toán Car-Miner cho hiệu quả tốt hơn khi phải khai thác lại luật phân lớp kết hợp trên CSDL bị sửa đổi.

Chương 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Luận văn này nghiên cứu bài toán khai thác luật kết hợp, luật phân lớp, luật phân lớp kết hợp trên CSDL tĩnh. Để từ đó nghiên xuất và đề xuất bài toán khai thác luật phân lớp kết hợp trên CSDL bị sửa đổi (thêm mới dữ liệu, sửa thông tin dữ liệu, xóa dữ liệu) sao cho khi sửa đổi CSDL việc khai thác luật không phải duyệt lại toàn bộ CSDL gốc nữa, tiết kiệm được thời gian và tài nguyên của máy tính.

Dựa vào kết quả thực nghiệm cho thấy thời gian thực thi của thuật toán CAR-Modified nhanh hơn so với hai quá trình thực hiện tuần tự của các thuật toán CAR-Incre, CAR-Del hay thực hiện trên CSDL tĩnh như CAR-Miner. Ngày nay các CSDL liên tục tăng trưởng và thay đổi không ngừng, do đó thuật toán CAR-Miner không còn phù hợp để áp dụng vào thực tế nữa, nên thuật toán CAR-Modified đã giúp cho việc khai thác luật phân lớp kết hợp giảm được chi phí duyệt lại toàn bộ CSDL từ đầu và hoàn toàn phù hợp với CSDL thực tế hiện nay.

5.2. Hướng phát triển

Luận văn hiện đang thực hiện thuật toán trên các file dữ liệu tĩnh trên một máy, nên thời gian tới tôi sẽ tìm kiếm giải pháp cài đặt thuật toán trên các CSDL thật và CSDL đã được phân tán. Có thể thấy đây là một bài toán thực tế cần giải quyết. Một trong những phương pháp có thể thực hiện được là cải tiến các thuật toán khai thác song song tập phổ biến cho bài toán khai thác luật phân lớp kết hợp. Cách làm này có ưu điểm là tận dụng được thế mạnh xử lý và nguồn tài nguyên của nhiều máy tính. Tuy nhiên, khi số lượng luật thu được lớn thì phương pháp này tốn nhiều thời gian truyền/nhận kết quả và tổng hợp kết quả. Bên cạnh đó, việc tính toán song song cũng cần một hệ thống máy chuyên nghiệp trong khi nhu cầu là khai thác tức thời tận dụng năng lực xử lý của các máy hiện hành đang dùng. Với các máy tính kết nối qua LAN/WAN, việc truyền/nhận dữ liệu tương đối chậm do

thường truyền theo cơ chế file. Với cách phân tích như trên, việc đề nghị một mô hình xử lý có thể tận dụng được năng lực của các bên tham gia, giảm chi phí truyền thông là cần thiết. Mô hình đề xuất không gửi nhận tất cả các thông tin của CSDL cho bên cần khai thác mà chỉ nhận thông tin của các item có khả năng phổ biến (nghĩa là phổ biến ở ít nhất một vị trí). Chính điều này làm giảm thiểu số item cần gửi nhận giữa các bên tham gia. Việc xử lý cũng không phải tập trung hoàn toàn vào bên cần khai thác mà xử lý phân tán trên các bên tham gia, bên khai thác chỉ tổng hợp kết quả và tiến hành khai thác. Nhược điểm chính của phương pháp này là không tận dụng được năng lực tính toán của các bên tham gia trong giai đoạn khai thác và tốn thời gian chờ giữa các bên cho việc gửi/nhận thông tin của các bên. Chính vì vậy, khi độ hỗ trợ tối thiểu nhỏ dần dẫn đến số lượng item thỏa mãn độ hỗ trợ tối thiểu lớn thì giải pháp đề xuất có thể sẽ không hiệu quả.

Một trong những khó khăn của việc xử lý song song trong giai đoạn khai thác luật là việc tổng hợp kết quả, chính vì vậy trong thời gian tới, tôi sẽ cố gắng nghiên cứu đề xuất giải pháp cho vấn đề này. Bên cạnh đó, việc tìm kiếm giải pháp để giảm chi phí truyền thông như sử dụng cấu trúc dữ liệu bit khi truyền dữ liệu cũng sẽ được quan tâm.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] W.C. Chen, C.C. Hsu, J.N. Hsu (2012). “Adjusting and generalizing CBA algorithm to handling class imbalance”. Expert Systems with Applications, vol.39, no.5, pp. 5907-5919.

[2] Y.W.C. Chien and Y.L. Chen (2010): "Mining associative classification rules with stock trading data – A GA-based method”. Knowledge-Based Systems vol. 23, no 6, pp. 605-615.

[3] H.T.T. Giang (2010). “Nghiên cứu các luật kết hợp song song trong khai thác dữ liệu”.

[4] T.P. Hong, C.Y. Wang (2010). “An efficient and effective association-rule maintenance algorithm for record modification”. Expert Systems with Applications, vol.37, no., pp. 618–626.

[5] Khoa Công nghệ thông tin. “Khai phá dữ liệu”. Trường Đại học Hàng Hải Việt Nam.

[6] B. Liu, W. Hsu, Y. Ma (1998). “Integrating classification and association rule mining”. In Proc. of the 4th International Conference on Knowledge Discovery and Data Mining, New York, USA, pp. 80-86.

[7] W. Li, J. Han, J. Pei (2001). “CMAR: Accurate and efficient classification based on multiple class-association rules”. In Proc. of the 1st IEEE International Conference on Data Mining, San Jose, California, USA, pp. 369-376.

[8] L.T.T. Nguyen, B. Vo, T.P. Hong, H.C. Thanh (2013). “CAR-Miner: An efficient algorithm for mining class-association rules”. Expert Systems with Applications, vol.40, no.6, pp. 2305-2311.

[9] L.T.T. Nguyen, N.T. Nguyen (2015). “An improved algorithm for mining class association rules using the difference of Obidsets”. Expert Systems with Applications. Vol. 42, Iss. 9, pp. 4361-4369 (SCIE, 2013 IF 1.965).

[10] H.R. Qodmanan, M. Nasiri, B. Minaei-Bidgoli (2011). "Multi objective association rule mining with genetic algorithm without specifying minimum

support and minimum confidence ". Expert Systems with Applications, vol. 38, no 1, January 2011, pp. 288-298.

[11] J. R. Quinlan (1992). “C4.5: program for machine learning”. Morgan Kaufmann.

[12] Thabtah, P. Cowling, Y. Peng (2005). “MCAR: Multi-class classification based on association rule”. In Proc. of The 3rd ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia, pp. 33-39.

[13] R. Thonangi, V. Pudi (2005). “ACME: An associative classifier based on maximum entropy principle”. In Proc. of the 16th International Conference Algorithmic Learning Theory, LNAI 3734, Singapore, pp. 122-134.

[14] M.R. Tolun, H. Sever, M. Uludag, S.M. Abu-Soud (1999). “ILA-2: An inductive learning algorithm for knowledge discovery”. Cybernetics and Systems, vol.30, no.7, pp. 609-628.

[15] M.R. Tolun, S.M. Abu-Soud (1998). “ILA: An inductive learning algorithm for production rule discovery”. Expert Systems with Applications vol.14, no.3, pp. 361-370.

[16] A. Veloso, W. Meira Jr., M. Goncalves, H.M. Almeida, M.J. Zaki (2011). “Calibrated lazy associative classification”. Information Sciences, vol.181, no.13, pp. 2656-2670.

[17] A. Veloso, W. Meira Jr., M.J. Zaki (2006). “Lazy associative classification”. In Proc. of the 2006 IEEE International Conference on Data Mining (ICDM’06), Hong Kong, China, 645-654.

[18] B. Vo, B. Le (2008). A novel classification algorithm based on association rule mining. In Proc. of the 2008 Pacific Rim Knowledge Acquisition Workshop (Held with PRICAI’08), LNAI 5465, vol. 5465, pp. 61-75.

[19] X. Yin, J. Han (2003). “CPAR: Classification based on predictive association rules”. In Proc. of SIAM International Conference on Data Mining (SDM’03), San Francisco, CA, USA, pp. 331-335.