1. **Tổng quan**

Khai phá tần số của “itemset” là một nhiệm vụ cơ bản trong khai phá dữ liệu. Có rất nhiều các ứng dụng sử dụng việc khai phá như trên. Trong những năm gần đây, một số cấu trúc dữ liệu được biểu diễn dựa vào cấu trúc “node” trên cây tiền tố. Các cấu trúc dữ liệu đó chứa tất cả các thông tin về tần số của “itemset”.

NegNodeSet là một trong những cấu trúc dữ liệu khá phổ biến. Cũng giống như các kiểu cấu trúc dữ liệu khác, NegNodeset cũng bao gồm một tập các node trong cây tiền tố dựa vào các tập “bitmap” được biểu diễn. Dựa vào cấu trúc của NegNodeSet, negFIN được phát triển để khai phá ra các tần số của “itemset”. Thuận lợi của việc sử dụng thuật toán negFIN đó chính là nó được xác nhận vởi 3 lí do:

* Một là NegNodesets của các itemsets được trích xuất bằng cách sử dụng các toán tử bitwise
* Hai là độ phức tạp của NegNodesets và các toán tử đếm được giảm đến O(n) với n là số lượng NegNodesets
* Ba là nó sử dụng truy vấn cây để tạo ra tần số của các “itemsets” và sử dụng một phương thức khác để loại bỏ bớt đi không gian tìm kiếm trên cây. Việc mở rộng biểu diễn dựa vào một số các tiêu chuẩn của các tập “itemset” có trong data và negFIN là thuật toán nhanh nhất so với các thuật toán tương tự trước đây, Tuy nhiên, thuật toán này chạy với tốc độ cùng bằng với dFIN trên cùng tập dữ liệu

1. **Giới thiệu**

Việc khai phá tần số của “itemsets” giúp ta có thể khám phá ra được một số quy tắc liên kết, nhóm và phân loại dữ liệu. Mục tiêu ban đầu đó chính là phân tích và chuyển dữ liệu với tần số đã khám phá ra nhằm giảm đi số lượng các phép biến đổi dữ liệu

* 1. **Một số định nghĩa**

Giả sử là tập các item trong tập chuyển đổi dữ liệu, một phép biển đổi T là tập các items với một TID phân biệt và một cơ sở dữ liệu là tập các phép biến đổi. Với mỗi P sao cho được gọi là một “itemset”. Khi đó P cũng được gọi là k-itemset với . Một phép biến đổi T chứa một itemset P nếu và chỉ nếu . Giá trị support của P(kí hiệu là support(P)) được xác định là tỉ lệ các phép biến đổi trong DB chứa P. Ngưỡng giá trị min-support sẽ được người dùng đưa vào. P còn được gọi là tần ố của itemset nếu và chỉ nếu . Cho cơ sở dữ liệu DB và ngưỡng giá trị min-support, việ tìm ra tần số các itemset được định nghĩa như việc khai phá tất cả tần số của các itemsets với giá trị support của chúng. Số lượng của các itemsets được kiểm tra để tìm ra tần số của itemset là với . Vì vậy, việc tìm ra tần số của itemset chính là NP

* 1. **Đông lực và đóng góp**

Khai phá tần số của itemset là một chủ đề rất nóng trong lĩnh vực khai phá dữ liệu trong hai thập kỉ cuối này. Trong những năm gần đây, 4 kiểu cấu trúc dữ liệu dựa vào tập các “nodes” trong cây tiền tố được biểu diễn để tăng cường thuận tiện trong việc tìm ra tần số của các itemsets. Đó là Node-list, N-list, Nodeset và DifNodeset. Tất cả các cấu trúc dữ liệu sử dụng cây tiền tố với các nodes mã hóa và các liên kết giữa các tập nodes với mỗi itemset. Những node có trong Node-list và N-list được mã hóa bởi thứ tự sắp xếp tăng bậc của các nodes. Hai thuật toán: PPV và PrePost được đề xuất để tìm ra tần số các itemset dựa vào hai cấu trúc dữ liệu. Hai thuật toán trên biểu diễn các node cha. Tuy nhiên, chúng có một hạn chế: sử dụng quá nhiều bộ nhớ. Để khắc phục được vấn đề này, Nodeset được đề xuất đưa ra. Không giống như N-list và Node-list, các nodes trong một Nodeset được mã hóa chỉ bởi duyệt tiền tố(hoặc duyệt hậu tố) thứ tự xếp hạng của các node. Nodeset của mỗi k-itemset (k≥ 3) được trích ra bởi phép giao của Nodesets của 2 tập (k-1) itemset. Thuật toán FIN được đề xuất dựa vào cấu trúc này. Nhược điểm của Nodeset đó chính là nhiều Nodeset trở nên rất lớn trong cơ sở dữ liệu. Để khắc phục hạn chế này, một cấu trúc dữ liệu khác đã được sử dụng: DiffNodeset. Không như Nodeset, mỗi k-itemset (3≤k) trong DiffNodeset được trích ra bởi 2 (k-1)itemsets khác nhau. Và theo kết quả thực nghiệm, DiffNodeset nhỏ hơn so với Nodeset. Chính vì thế, thuật toán dFIN dựa vào cấu trúc này được đề xuất sử dụng chạy nhanh hơn so với các thuật toán trước của nó.

Dù cho những thuận lơi mà DiffNodeset mang lại, chúng ta thấy rằng việc tính toán sự khác nhau giữa 2 DiffNodeset mất khá nhiều thời gian trên một số cơ sở dữ liệu. Để khắc phục tình trạng này, NegNodeset được đề xuất sử dụng dựa vào cây tiền tố cũng như những cấu trúc dữ liệu khác. Tuy nhiên, NegNodeset sử dụng mô hình mã hóa nodes mới dựa vào bitmap để biểu diễn các tập, Giả sử xét tập U với n thành phần, chúng ta sẽ biểu diễn các tập con của U bởi bitmap có size n. Mỗi thành phần của U được đánh dấu là một trong các bits trong bitmap. Nếu có một phần tử của tập con S thuộc U được đánh dấu là 1 thì các bits còn lại là 0.

Dựa vào cấu trúc của NegNodeset, negFIN, một thuật toán tìm tần số của itemset khá nhanh được đề xuất sử dụng. Điểm mạnh của thuật toán này là: Một là NegNodeset mới được tạo ra bởi các toán tử bitwise, chính vì vậy nó sẽ chạy rất nhanh. Hai là việc tạo ra một NegNodeset mới và các phép đếm có độ phức tạp là O(n) thay vì O(m+n) trong các thuật toán trước đó, Ba là nó sử dụng tập cây đếm để tạo ra tần số của itemsets và sử dụng một số phương thức giảm bớt không gian tìm kiếm của cây.

* 1. **Biểu diễn negFIN**

Bằng việc thực hiện một số thực nghiệm nghiên cứu để biểu diễn thuật toán negFIn, cần phải so sánh thuật toán negFIN và dFIn. Kết quả thực nghiệm đã chỉ ra rằng negFIN biểu diễn tốt hơn và chạy nhanh hơn với mọi tập dữ liệu. Ngoài ra, với một vài tập dữ liệu khác nó chạy nhanh hơn rất nhiều so với FP-growth, dFIN.

1. **Các vấn đề liên quan**

Các thuật toán để tìm ta tần số của các itemsets được chia thành 2 loại chính: Một là các thuật toán sử dụng phương thức “candidate generation” và hai là các thuật toán sử dụng phương thức “pattern growth”.

Trong phương thức sử dụng candidate generation, các thành phần itemset được tạo ra đầu tiên, và sau đó, tần số của chúng được xác định từ các candidate itemsets. Phương thức này không sử dụng các thành phần đơn điệu được gọi là Apriori để giảm không gian tìm kiếm. Nếu một itemsets không có tần số thì super-itemset của nó cũng vậy.

Không giống như candidate generation, phương thức “pattern growth” không tạo ra các thành phần itemsets và tránh quét nhiều database cùng một lúc bằng cách chứa thông tin về tần số của các itemset trong một cấu trúc dữ liệu đặc biệt. Điều cơ bản của thuật toán này đó chính là các danh mục trong thuật toán FP-growth. Nó chứa các thông tin về tần số các itemset trong một cây dữ liệu có tên là FP-tree. Tương tự như FP-growth và các thuật toán khác, việc sử dụng các phương thức mẫu để tìm ra tần số của itemsets. Mặc dù vậy chúng có những nhược điểm nhất định: Một là các dư liệu sẽ rỗng và thưa và hai là cấu trúc dữ liệu sử dụng bởi các mẫu trong thuật toán này khá phức tạp.

Trong những năm gần đây, có 4 loại cấu trúc dữ liệu dựa vào cây tiền tố được đề xuất sử dụng để chứa thông tin về tần số của các itemset: Node-list, N-list, Nodeset và DiffNodeset. Cả Node-list và N-list đều dựa vào cấu trúc của cây PPC, một loại cấu trúc dữ liệu mã hóa các node bằng phép duyệt tiền tố và hậu tố. Node-list và N-list của một itemset là một tập các nodes trong cây PPC. N-list có 2 thuận lợi hơn so với Node-list: thứ nhất là số nhiều của N-list của một itemset nhỏ hơn rất nhiều so với số lượng lớn của Node-list và thứ hai là N-list sử dụng “single path property” để trực tiếp tìm ra tần số của các itemsets mà không cần tạo ra những thành phần khác trong một số trường hợp. Hai thuật toán PPV và PrePost được đề xuất dựa vạo Node-list và N-list. Trong những năm gần đây đã sử dụng các kĩ thuật giảm bớt/cắt tỉa nhằm tăng sư biểu diễn cho PrePost. Mặc dù có rất nhiều thuận lợi khi sử dụng Node-list và N-list, nhưng chúng phải sử dụng một không gian bộ nhớ rất lớn, vì vậy, cần phải chứa phép duyệt tiền tố và hậu tố của các nodes. Để khắc phục tình trạng này, Nodeset được đề xuất sử dụng. Thuật toán này chỉ cần lưu một trong các phép duyệt tiền tố hoặc hậu tố của các nodes.

Năm 2014, thuật toán FIN được đề xuất sử dụng dựa trên cấu trúc của Nodesets. Tuy nhiên, Nodeset cũng có một số hạn chế: một số lượng các lớn các itemset trở nên rất lớn trong một số cơ sở dữ liệu, chính vì thế, DiffNodeset tiếp tục được đề xuất sử dụng.

Thêm vào đó, PUN – list, một thuật toán được công bố 2018 cũng dựa vào cấu trúc dữ liệu trên được đề xuất để khai phá “high utility itemset”, một mục tiêu khác trong việc khai phá dữ liệu. Với mỗi item có giá trị hữu ích và có thể được biến đổi nhiều hơn một lần. Sự tiện lợi của một tập itemset đó chính là sự tiện dụng của nó không nhỏ hơn ngưỡng support. Hơn thế nữa, để lưu một thông tin về tần số của itemsets, PUN-list cũng chứa thông tin về các các tiện ích. Thuât toán MIP được đề xuất cho việc khai thác ra tính tiện lợi của các itemset dựa vào PUN-list.

Dưới đây là một số ví dụ

Ví dụ 1: Xét một tập biến đổi trên cơ sở dữ liệu được cho trong Table 1 dưới đây, ngưỡng min-support = 0.4. Với Table 1, cột thứ 1 là ID(TID), cột thứ hai là các item trong mỗi phép biến đổi và cột thứ ba chính là tần số xuất hiện của mỗi item trong mỗi phép biến đổi đã được sắp xếp theo giá trị với là một item

|  |  |  |
| --- | --- | --- |
| **TID** | **items** | **Ordered frequent items** |
| 1 | e, b, g, d | b, d, e |
| 2 | c, e, b, a | a, b, c, e |
| 3 | c, b, a, i | a, b, c |
| 4 | a, d, h | a, d |
| 5 | a, d, c, b, f | a, b, c, d |

**Định nghĩa 1:** (tập các tần số của items), nếu và chỉ nếu

**Định nghĩa 2:** Cho là các vector không dựa trên sự sắp xếp tần số của các items, với các items được sắp xếp không giảm bởi giá trị – là một item

được kí hiệu với và . Một k-itemset P được kí hiệu hoặc với và

**Định nghĩa 3:** . được xác định bởi với có không có trong item

**Định nghĩa 4: (index(item i))** Với bất kì item , index(i) được xác định bằng vị trí của item i trong

**Định nghĩa 5:** (BMC(itemset ) - bitmap code của mỗi itemset) Mỗi itemset có thể biểu diễn bởi một bitmap code BMC( của độ lớn nf được tính như sau: item trong vector được đánh dấu đến bit trong bitmap này. Nếu mỗi item i là một thành phần của , nó sẽ mang giá trị 1, ngược lại mang giá trị 0.

**Định nghĩa 6:** (BMC-tree) một cây BMC là cây thõa mãn:

* Nó là node root rỗng và có một số cây con item tiền tố như thể con của nó.
* Mỗi node trong mỗi cây con item tiền tố lưu giữ một item i . Nếu cha của node này biểu diễn item j thì . Và đường đi đến node này biểu diễn bởi *node-path*
* Mỗi node có 4 thuộc tính: *item-name, count, bitmap-code, children-list*. *Item-name* lưu giữ item i . *count* lưu số lượng phép biến đổi bao gồm itemset *node-path*. *bitmap-code* lưu *BMC(node-path)* và *children-list* lưu tất cả các con của node này

**Định nghĩa 7: (phần chính và phân không quan tâm của BMC(node-path)** Node N lưu giữ một item , với , và được đánh dấu đến . Các bit được xác định bởi phần chính của và các bits được xác định bởi các phần không quan tâm (*don’t-care section of BMC(node-path)*)