DATA MINING

BÁO CÁO LAB02
Frequent Itemset Mining

Sinh viên thực hiện: Trần Hoàng Nam MSSV: 18120473

1 Ý tưởng, cài đặt

Hai thuật toán được cài đặt trên julia 1.4.1, cú pháp dòng lệnh để chạy thuật toán: `julia apriori.jl <file name> <min support>` và `julia tree_projection.jl <file name> <min support>`. Ví dụ `julia tree_projection.jl retail.txt 0.01`. Chương trình chỉ nhận dữ liệu có các item là số nguyên, và xuất ra danh sách các frequent itemset cùng số lượng.

Các chương trình sẽ đọc dữ liệu và lưu ở biến `transactions`, là một mảng các giao tác, mỗi giao tác là một tập các hạng mục.

Sau đó từ các giao tác, sinh ra các frequent 1-itemset. Bằng cách dùng một biến `item_dict` kiểu dictionary để đếm số lần xuất hiện của tất cả các item trong dữ liệu. Và lọc ra những item có số lần xuất hiện nhiều hơn hoặc bằng `minsupp_count = minsupp*length(transactions)`

Các bước cài đặt tiếp theo sẽ tách biệt cho hai thuật toán.

1.1 Apriori

Thuật toán apriori(được tham khảo trong tài liệu ở trang môn học) sẽ in kết quả theo từng frequent k-itemsets L_k. Mỗi L_k như vậy được quản lí bởi một tập(Set) các itemsets.

Như vậy ta đã có L_1, và sẽ thực hiện vòng lặp để in tất cả L_k, với k tăng dần cho đến khi tập rỗng thì dừng, vòng lặp được bắt đầu với $k = 2$, các bước như sau:

1. In ra L_{k-1} được thể hiện bởi biến `last`
2. Khởi tạo biến rỗng `current` để lưu L_k
3. Với tất cả các tổ hợp 2 itemset `itemset_i, itemset_j` có thể có ở trong `last`, ta thực hiện phép hợp(union) để cho ra itemset `itemset_ij`
   (a) Nếu `itemset_ij` có k phần tử, ta kiểm tra nó có phải là frequent itemset hay không. Bằng cách đếm tất cả các giao tác nhận `itemset_ij` là tập con, nếu số lượng này lớn hơn `minsupp_count` thì đó là frequent itemset, và `itemset_ij` được thêm vào `current`
   (b) Ngược lại, nếu `itemset_ij` có số lượng phần tử thì khác k thì ta bỏ qua
4. `last` $\leftarrow$ `current`, $k \leftarrow k + 1$

Nhận xét: sẽ có trường hợp hai cặp itemset khác nhau có kết quả hợp giống nhau, và nếu kết quả này là frequent itemset thì sẽ được thêm vào `current` 2 lần. Tuy nhiên `current` là một Set nên sẽ chỉ tính là 1. Ta cũng có thể kiểm tra sự tồn tại của một phần tử trong Set trước khi kiểm tra frequent itemset, sẽ tăng một chút về perfomance.

1.2 Tree Projection

Sau đây là ý tưởng và cài đặt một thuật toán tree projection để tạo một lexicographic tree theo hướng breath-first search, được tham khảo trên bài báo [1]

Cây được quản lí bởi biến `tree`, là một mảng trong đó phần tử thứ i là một mảng lưu các biến có cấu trúc `Node`(đại diện cho mỗi nút trên cây) ở tầng(level) thứ i, riêng ở level 0 thì được quản lí bởi biến `root`. Cấu trúc `Node` bao gồm các thành phần sau:

- `itemset`: Frequent itemset lưu tại nút, số lượng phần tử bằng với level của nút
- `extensions`: Các 1-extension của nút, tức là các item được thêm vào `itemset` để tạo thành các nút con mới.
- `projections`: "Hình chiếu" của `trasactions` lên nút, là phần giao(intersect) của tất cả giao tác nhận `itemset` làm tập con với `extensions`. Hay nói cách khác là tất cả các item của các giao tác mà đã được xác minh là có khả năng mở rộng cho nút.

Mỗi nút sẽ được trải qua hai giai đoạn, generate và examine. Ở giai đoạn generate, nút được tạo ra lần đầu tiên và xác định được thuộc tính `itemset`; ở giai đoạn examine, nút xác định được hai thông tin còn lại là `extensions, trasactions`. Ở bước khởi tạo, `root` được generate và examine thủ công, các nút ở tầng thứ nhất được generate thủ công.

Tương tự với apriori, tree projection sinh các frequent itemset với kích thước tăng dần bằng một vòng lặp ngoài. Bắt đầu với $k = 1$ và kết thúc khi không sinh thêm nút mới. Trong lần lặp thử k, ta gọi `parents, currents, children` là tập tất cả các nút ở tầng $k-1, k, k+1$; cùng với tất cả các thuộc tính tương ứng, ví dụ `parents.itemset, children.extensions`,... Lúc này, `parents` đã được generate và examine, `currents` đã được generate và chưa được examine, `children` chưa được generate. Mục đích của chúng ta là examine `currents` và generate `children`.

Bước generate `children` được thực hiện trước, sử dụng thông tin của `parents`. Cụ thể là với mỗi `parent` $\in$ `parents` sẽ sinh ra các `child` với các `child.itemset` được tạo ra từ việc thêm vào `parent.itemset` 2 item lấy từ `parent.extensions` mà thỏa điều kiện min support. Để làm điều này ta cần phải đếm số lần xuất hiện của tất cả tổ hợp 2 item trong `parent.projections`(các item này chắc chắn nằm trong `parent.extensions`). Ở bài báo tham khảo sử dụng một ma trận để đếm, để đơn giản em sử dụng một biến `d` kiểu dictionary để map một cặp item $\{$`item_i`, `item_j`$\}$(`item_i` $<$ `item_j`) với số lần xuất hiện của nó. Các `child` thỏa min support được thêm lần lượt vào `children`.

Bước examine `currents` được thực hiện tiếp theo. Với mỗi `child` $\in$ `children`, ta tìm nút cha của nó là một nút `current` $\in$ `currents`. Việc tìm kiếm này đơn giản vì `current.itemset` = `parent.itemset` $\cup$ `item_i`, sau đó ta thêm `item_j` vào `current.extensions`. Sau khi lặp xong tất cả `child` $\in$ `children`, ta xác định được `currents.extensions`. Sau đó `currents.projections` được xác định bằng cách lấy hợp giữa `parents.projections` và `currents.extensions`, và đặc biệt quan trọng là chỉ chọn những `projection` có chứa phần tử cuối cùng của `current.itemset`(tương ứng là `item_i`).

Algorithm 1: Mã giả tree projection

input : `transactions, minsupp_count`
output: In ra các frequent itemset

Function `project(`*`transactions, extensions, item`*`)`:
    Thực hiện phép hợp giữa tất cả phần tử của `transactions` chứa `item`(nếu có) với `extensions` rồi trả về mảng kết quả.
Lưu frequent 1_itemsets vào L
Khởi tạo `root, tree`
`root.itemset` $\leftarrow \{\}$
`root.extensions` $\leftarrow L$
`root.projections` $\leftarrow$ **project**(`transactions`, `root.extensions`, $None$)
`children` $\leftarrow \{$`Node` $(\{\text{item}\}, \{\}, \{\})$ **for** `item` $\in L\}$
$k \leftarrow 1$
while `children` $\neq \{\}$ **do**
    Thêm `children` vào `tree`
    `parents` = `tree`[k-1]
    `currents` = `tree`[k]
    `children` = $\{\}$
    for `parent` **in** `parents` **do**
        Đếm số lượng xuất hiện của tất cả tổ hợp 2 item trong `parent.projections`
        Lọc những cặp item thỏa `minsupp`
        for `item_ij` = $\{$`item_i`, `item_j`$\}$ nhận được **do**
            Tạo nút `child` mới với `child.itemset` = `parent.itemset` $\cup$ `item_ij`
            Thêm `child` vào `children`
            Tìm `current` $\in$ `currents` là cha của `child`
            Thêm `item_j` vào `current.extensions`
        end
        for `current` **in** `currents` **do**
            `item_i`$\leftarrow$ phần tử cuối cùng của `current.itemset`
            `current.projections` $\leftarrow$ **project**(`parent.projections, current.extensions, item_i`)
        end
    end
    $k \leftarrow k + 1$
end
In `tree`

Nhận xét:

- Cách cài đặt cây bằng mảng khiến các kết nối giữa các nút trở nên lỏng lẻo, tuy nhiên việc duyệt nút trên một tầng là rất dễ dàng.
- Tính thứ tự của lexicographic tree cùng với việc sử dụng projected transactions làm giảm đáng kể độ phức tạp thời gian.
- Có thể tăng perfomance một chút từ việc bỏ qua các projection có kích thước nhỏ hơn 2, hoặc bỏ qua các nút có kích thước extensions nhỏ hơn 2. Vì chắc chắn sẽ không sinh ra nút con mới.

2 Đánh giá, so sánh

Thuật toán apriori

- Dễ hiểu, dễ cài đặt. Có thể cài đặt song song.
- Độ phức tạp thời gian, không gian cao
  - Phải duyệt toàn bộ dữ liệu nhiều lần
  - Lưu trữ nhiều các itemset trung gian
  - Độ phức tạp không gian, thời gian là $\mathcal{O}\left(2^{|D|}\right)$

Thuật toán tree projection

- Các projected transactions thường có kích thước rất nhỏ so với transactions gốc, kích thước giảm hơn nữa khi đi xuống tầng sâu hơn, chỉ giữ lại những item cần thiết. Làm tăng tốc hơn rất nhiều cho việc duyệt frequent itemset
- Linh động, có thể áp dụng các chiến lược khác như DFS hoặc phối hợp BFS và DFS, có thể được cài đặt song song.
- Tốn nhiều bộ nhớ khi phải lưu tập projected transations cho từng nút.

Các so sánh thực nghiệm giữa hai thuật toán cũng được đề cập chi tiết ở bài báo [1]

3 Khai thác dữ liệu

Frequent Itemset Mining có ứng dụng rất rộng rãi trong thực tế, một loại bài toán phổ biến là Market Basket Analysis. Mục đích là người bán hàng tìm hiểu những sản phẩm đi cùng với nhau, dựa vào những quan sát ghi lại trên người mua. Từ đó có chiến lược buôn bán hợp lí nhằm tối đa lợi nhuận.

Sau đây em demo với một ví dụ nhỏ, sử dụng dataset từ trang web kaggle, có dạng như hình dưới đây

	Apple	Bread	Butter	Cheese	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Sugar	Unicorn	Yogurt	chocolate
0	False	True	False	False	True	True	False	True	False	False	False	False	True	False	True	True
1	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False
2	True	False	True	False	False	True	False	True	False	True	False	False	False	False	True	True
3	False	False	True	True	False	True	False	False	False	True	True	True	False	False	False	False
4	True	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False
5	True	True	True	True	False	True	False	True	False	False	True	False	False	True	True	True
6	False	False	True	False	False	False	True	True	True	True	True	True	False	False	True	False
7	True	False	False	True	False	False	True	False	False	False	True	False	True	False	True	False
8	True	False	False	False	True	True	True	True	False	True	True	True	True	True	True	True
9	True	False	False	False	False	True	True	True	False	True	False	True	True	True	False	True

Hình 1: Phần đầu của dataset, các hàng biểu diễn các giao tác(tổng 999 giao tác), các cột biểu diễn các mặt hàng. Giá trị True/False thể hiện có giao dịch hay không.

Dữ liệu này có thể khai thác với apriori, em sử dụng môt script julia khác là `apriori2.jl`. Với cú pháp dòng lệnh tương tự 2 thuật toán trên, ví dụ: `julia apriori2.jl basket_analysis.csv 0.1`. Chỉ có một số thay đổi nhỏ so với script `apriori.jl`

- Đọc file với các file có định dạng như trên

- Cung cấp thêm điểm support thay vì chỉ liệt kê itemset, và sắp xếp giảm dần theo điểm support
- Lưu vào file thay vì in ra màn hình, giúp người bán hàng dễ dàng theo dõi hơn. Tên file mới có cùng tiền tố với tên file cũ và có bổ sung minsupp.

Sau khi chạy script với cú pháp trên thì ta được 169 itemset với tối đa 3 item cho mỗi itemset. Các kết hợp tốt nhất của từng k-itemset được thể hiện trên hình sau

```
0.42142 (chocolate)
0.42042 (Butter)
0.42042 (Yogurt)
0.41041 (Ice cream)
0.40941 (Sugar)
0.40841 (Kidney Beans)
0.40741 (Corn)
0.40541 (Milk)
0.4044  (Cheese)
0.4034  (Onion)
0.4014  (Nutmeg)
0.3984  (Dill)
0.38939 (Unicorn)
0.38438 (Bread)
0.38438 (Eggs)
0.38338 (Apple)
0.21121 (Milk, chocolate)
0.20721 (Butter, Ice cream)
0.2022  (Kidney Beans, Butter)
0.2022  (Butter, chocolate)
0.2022  (chocolate, Ice cream)
0.2002  (Kidney Beans, Cheese)
0.1992  (Kidney Beans, Milk)
0.1992  (chocolate, Dill)
0.1982  (Butter, Nutmeg)
```

```
0.11411 (Milk, chocolate, Dill)
0.11011 (Kidney Beans, Butter, Ice cream)
0.10911 (Butter, chocolate, Ice cream)
0.10611 (Sugar, Butter, Ice cream)
0.10511 (Kidney Beans, Milk, Corn)
0.1041  (Cheese, Kidney Beans, Ice cream)
0.1041  (Kidney Beans, Milk, Butter)
0.1041  (Butter, Nutmeg, Onion)
0.1041  (Milk, chocolate, Yogurt)
0.1031  (chocolate, Onion, Dill)
0.1031  (Butter, Ice cream, Onion)
0.1031  (chocolate, Ice cream, Dill)
0.1021  (Cheese, Dill, Onion)
0.1021  (Butter, Corn, Ice cream)
0.1021  (Milk, chocolate, Corn)
0.1021  (Milk, Butter, chocolate)
0.1021  (Butter, Nutmeg, Ice cream)
0.1021  (Kidney Beans, Cheese, chocolate)
0.1011  (chocolate, Ice cream, Onion)
0.1011  (Cheese, Sugar, Kidney Beans)
0.1011  (Kidney Beans, Butter, Corn)
0.1011  (chocolate, Nutmeg, Ice cream)
0.1011  (Unicorn, chocolate, Dill)
0.1011  (Kidney Beans, Butter, Nutmeg)
0.1011  (Kidney Beans, Cheese, Nutmeg)
```

Từ đó rút ra được một số nhận xét

- Các mặt hàng được bán khá đều nhau, trong đó nhiều nhất là Chocolate, Butter, Yogurt, Ice cream,...
- Kết hợp 2 tốt nhất là (Milk, Chocolate); và nó sẽ là kết hợp 3 tốt nhất nếu thêm Dill
- Nên kết hợp (Butter, Chocolate) hơn (Butter, Kidney Beans), vì dù chúng có supp bằng nhau nhưng Chocolate lại phổ biến hơn Kidney Beans. Tuy nhiên khi thêm Ice Scream thì cần xem xét kĩ lưỡng hơn vì (Butter, Kidney Beans, Ice cream) phổ biến hơn (Butter, Chocolate, Ice cream)

Có thể thay đổi min supp để phù hợp với mục đích của chúng ta . Tuy nhiên, để khai thác được nhiều hơn, ta cần phải khai thác các luật kết hợp. Nếu có giá cả của các mặt hàng thì có thể sử dụng các mô hình như quy hoạch tuyến tính để khai thác các kết hợp mang nhiều lợi ích kinh tế nhất.

Tài liệu

[1] R. C. Agarwal, C. C. Aggarwal, and V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of parallel and Distributed Computing*, 61(3):350–371, 2001.