TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM

KHOA CÔNG NGHỆ THÔNG TIN

LAB 02

CÀI ĐẶT VÀ ỨNG DỤNG 2 THUẬT TOÁN: APRIORI VÀ TREE PROJECTION CHO VIỆC KHAI THÁC TẬP PHỔ BIẾN TRÊN CÁC TẬP DỮ LIỆU

Học phần: Khai thác dữ liệu và ứng dụng

Lớp: 19_21

Giáo viên hướng dẫn: Nguyễn Ngọc Đức

Sinh viên: Lê Kiệt

MSSV: 19120554

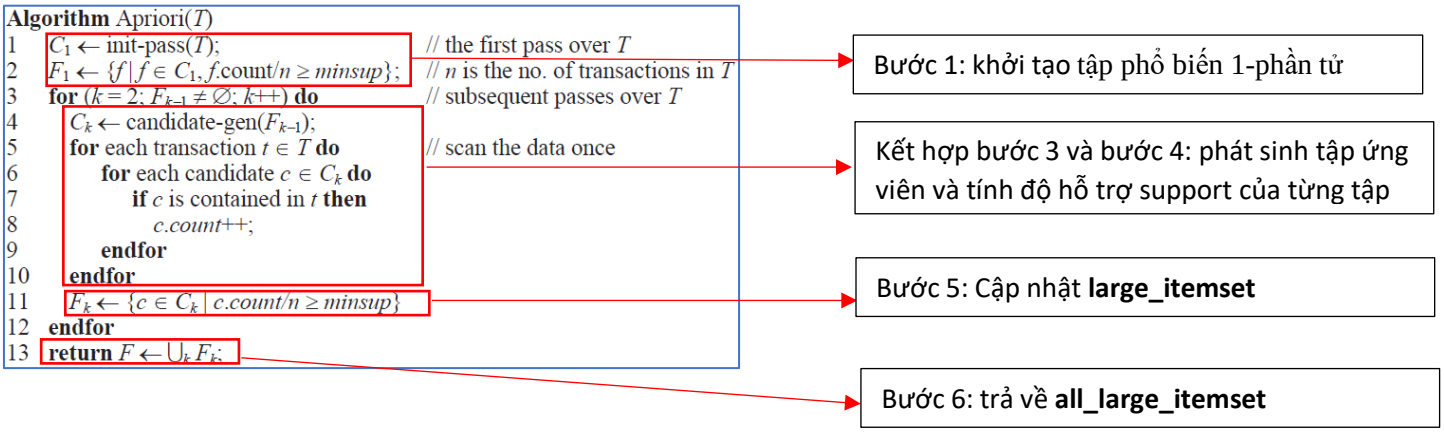
MỤC LỤC

I. THUẬT TOÁN APRIORI

- Sử dụng hàm apriori(database, min_sup) để thực hiện thuật toán. Hàm nhận 2 tham số là **database** và **min_sup** lần lượt là cơ sở dữ liệu giao tác và độ hỗ trợ tối thiểu; trả về **all_large_itemset** là 1 danh sách (Array) tất cả các tập phổ biến thỏa **min_sup**. Sau đây là các bước mà hàm này thực hiện
  - Bước 1: khởi tạo tập **large_itemset** là danh sách các tập phổ biến 1-phần tử **one_itemset** đã được sắp xếp theo thứ tự từ điển thỏa **min_sup**,
  - Bước 2: lưu **large_itemset** này vào biến kết quả **all_large_itemset**
  - Bước 3: phát sinh **candidate_list** là danh sách các tập ứng viên sinh ra từ tập phổ biến large_itemset nhờ hàm candidate_gen(large_itemset)
  - Bước 4: cập nhật support của các tập ứng viên trong
    với mỗi dòng/giao tác trong database, kiểm tra:
    với mỗi ứng viên trong tập ứng viên **candidate_list**, kiểm tra:
    if (ứng viên ⊆ giao tác):
    support(ứng viên) += 1
  - Bước 5: kết thúc bước 4, lọc lại các tập ứng viên thỏa **min_sup**. Cập nhật **large_itemset** là các tập ứng viên mới lọc này, nếu **large_itemset** là tập rỗng (tức không còn tập ứng viên nào thỏa **min_sup**) thì dừng thuật toán; ngược lại, quay lại bước 2
  - Bước 6: trả về **all_large_itemset**

- Trong hàm trên có sử dụng thuật toán phát sinh các tập ứng viên là hàm candidate_gen(large_itemset). Hàm nhận 1 **large_itemset** là danh sách các tập phổ biến k-phần tử có thứ tự từ điển và trả về **candidate_list** là danh sách các tập ứng viên k+1 phần tử. Sau đây là các bước mà hàm này thực hiện
  - Bước 1: với mỗi cặp tập phổ biến k-phần tử (large_itemset[i], large_itemset[j]), $i \neq j$ sao cho k-1 phần tử đầu của từng tập phổ biến trong cặp phải giống nhau → tiến hành bước 2
  - Bước 2: dùng toán tử hợp (union) để hợp cặp này lại, kết quả sau khi hợp lưu vào **u**.
  - Bước 3: tỉa nhánh: xét tất cả tập con (k-1)-phần tử của **u**, có tập con nào là tập không phổ biến, tức không nằm trong **large_itemset** hay không. Nếu tồn tại 1 tập con như vậy thì không chấp nhận tập ứng viên **u**; ngược lại, nếu không tồn tại bất kỳ tập con nào không phổ biến thì thêm **u** vào **candidate_list** (Dựa vào quy tắc: tập con không phổ biến → tập cha chứa nó không phổ biến)
    - Lưu ý ở bước 3: khi xét tập con có thuộc vào tập **large_itemset** hay không. Về mặt lập trình trong Julia, cần sắp xếp lại tập con theo thứ tự bảng chữ cái (nếu là chữ) hoặc từ bé tới lớn (nếu là số). Ví dụ, ['C','T'] ∈ [['A','B'], ['C', 'T']] trong khi ['T','C'] ∉ [['A','B'], ['C', 'T']]
  - Bước 4: Sau khi xét xong tất cả các cặp large_itemset[i] và large_itemset[j] thỏa bước 1 thì **candidate_list** cũng đã chứa các tập ứng viên (k+1)-phần tử hợp lệ → trả về **candidate_list**

- 2 hàm trên dược cài đặt dựa vào mã giả sau (trích từ slide bài giảng trên lớp)



```
Function candidate-gen(F_{k-1})
1   C_k ← ∅;                                  // initialize the set of candidates
2   forall f_1, f_2 ∈ F_{k-1}                 // find all pairs of frequent itemsets
3     with f_1 = {i_1, … , i_{k-2}, i_{k-1}}  // that differ only in the last item
4     and f_2 = {i_1, … , i_{k-2}, i'_{k-1}}
5     and i_{k-1} < i'_{k-1} do               // according to the lexicographic order
6       c ← {i_1, …, i_{k-1}, i'_{k-1}};      // join the two itemsets f_1 and f_2
7       C_k ← C_k ∪ {c};                      // add the new itemset c to the candidates
8       for each (k–1)-subset s of c do
9         if (s ∉ F_{k-1}) then
10          delete c from C_k;                // delete c from the candidates
11      endfor
12  endfor
13  return C_k;                               // return the generated candidates
```

Kết hợp bước 1 và bước 2

Bước 3: tỉa nhánh

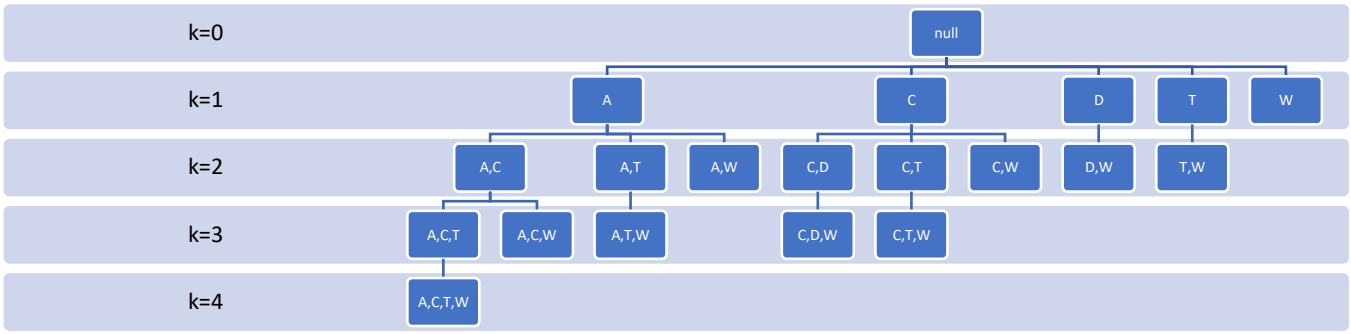
II. THUẬT TOÁN TREE PROJECTION

1. Ý tưởng

- Thuật toán Tree Projection được dựa theo thuật toán Apriori để tạo ra 1 cây được cấu trúc theo thứ tự từ điển (lexicographic tree) với mỗi node của cây là 1 tập phổ biến. Ta định nghĩa 1 cấp k trong cây (k nguyên dương, $k \geq 0$) là cấp mà ở đó các tập phổ biến (cũng chính là các node) có k phần tử. Như vậy node gốc (root node) ở cấp 0 sẽ có 0 phần tử hay chính là node rỗng; các node ở cấp 1 sẽ có 1 phần tử; các node ở cấp 2 sẽ có 1 phần tử; v.v.
    - Minh họa: 19 tập phổ biến trích từ slide "**FrequentItemsetMining.pdf**" trang 22 được cung cấp

```
[nothing]
├─ ["A"]
│  ├─ ["A", "C"]
│  │  ├─ ["A", "C", "T"]
│  │  │  └─ ["A", "C", "T", "W"]
│  │  └─ ["A", "C", "W"]
│  ├─ ["A", "T"]
│  │  └─ ["A", "T", "W"]
│  └─ ["A", "W"]
├─ ["C"]
│  ├─ ["C", "D"]
│  │  └─ ["C", "D", "W"]
│  ├─ ["C", "T"]
│  │  └─ ["C", "T", "W"]
│  └─ ["C", "W"]
├─ ["D"]
│  └─ ["D", "W"]
├─ ["T"]
│  └─ ["T", "W"]
└─ ["W"]
```

*Trích từ file **sample_tree.txt***



Cây được vẽ lại từ hình trên

- Các node là con của 1 node khi giữa node cha và mỗi node con có chung các phần tử của node cha cũng như node con phải nhiều hơn node cha 1 phần tử. Ví dụ như hình minh họa trên, node con của A là [A,W], [A,T], và [A,C] vì có chung 1 phần tử A và 3 node con này nhiều hơn 1 phần tử so với node cha A; node con của [A,W] là [A,T,W] và [A,C,W] vì có chung [A,W] và 2 node con nhiều hơn 1 phần tử so với node cha
- Tất cả các node đều có thể có nhiều con nhưng chỉ có 1 node cha, node nào đã có cha rồi thì không thêm cha khác vào nữa. Ví dụ node [A,W] đã có cha là A thì không cần thêm cha là W nữa

2. Định nghĩa cấu trúc dữ liệu phù hợp

- Để thực hiện vẽ cây với mỗi node là 1 tập phổ biến, ta cần định nghĩa 1 kiểu cấu trúc mới mang tên **Node**. 1 thể hiện Node (gọi là 1 node) sẽ có 4 thuộc tính để lưu trữ thông tin:
  - **itemset**: lưu trữ tập phổ biến
  - **children**: lưu trữ các tập phổ biến con của 1 node
  - **sup**: độ hỗ trợ (support) của tập phổ biến **itemset**
  - **parent**: lưu trữ đúng 1 node cha của tập phổ biến hiện tại
- Ngoài ra cấu trúc Node có constructor mặc định được định nghĩa như hình dưới

```
mutable struct Node
    itemset # ex: ['a', 'b']
    children::Vector{Any}
    sup
    parent

    # constructor
    Node(its, chi = [], su = 0, par = nothing) = new(its, chi, su, par)
end
```

3. Các bước thực hiện

- Thuật toán Tree Projection có các bước về bản chất giống với thuật toán Apriori đã nêu và điều chỉnh để thích hợp với kiểu dữ liệu struct Node; điểm khác biệt giữa cài đặt 2 thuật toán được mô tả bằng chữ xanh. Ta có hàm apiori_tree(database, min_sup) nhận 2 tham số là **database** và **min_sup** lần lượt là cơ sở dữ liệu giao tác và độ hỗ trợ tối thiểu; trả về **tree** là là cây với các Node lồng nhau theo quan hệ cha-con như ở mục 1 - Ý tưởng. Các bước thực hiện
  - Bước 1:
    - Khởi tạo tập phổ biến 1-phần tử **one_itemset** thỏa min_sup. Sort **one_itemset** theo thứ tự từ điển
    - Khởi tạo biến kết quả **tree** = Node(itemset = [nothing], children = one_itemset). Ban đầu **tree** được khởi tạo chỉ có root node = null và con của root node là tập one_itemset thỏa min_sup
    - Khởi tạo biến con trỏ chạy dùng để duyệt qua các cấp trong cây **itr** = **tree**.children
  - Bước 2: phát sinh danh sách các tập ứng viên **candidate_nodes** nhờ hàm candidate_nodes_gen(itr)
  - Bước 3: cập nhật support của các node ứng viên trong **candidate_nodes**

    với mỗi **transaction** trong database, kiểm tra:

    với mỗi **candidate** trong tập **candidate_nodes**, kiểm tra:

if (candidate.itemset ⊆ transaction):
candidate.sup += 1

- Bước 4: kết thúc bước 3, lọc lại các node ứng viên thỏa **min_sup** và lưu vào biến tạm là **large_itemset**. Ngược lại, đối với các node ứng viên không thỏa **min_sup** thì "xóa" các node này khỏi cây. Việc "xóa" node thực chất là việc cắt đứt quan hệ ***con-cha*** của node đó với node cha (node.parent = []) và cắt đứt quan hệ ***cha-con*** của node cha và node đó (node_cha.children = node_cha.children\{node_con})
- Bước 5: cập nhật con trỏ **itr** = **large_itemset**. Nếu itr rỗng thì thuật toán kết thúc; ngược lại quay lại bước 2

- Trong hàm trên có sử dụng thuật toán phát sinh các tập ứng viên là hàm candidate_nodes_gen (large_itemset_nodes). Hàm nhận 1 **large_itemset_nodes** là danh sách các node phổ biến k-phần tử và trả về **candidate_nodes** là danh sách các node ứng viên (k+1)-phần tử. Hàm này cũng tương tự hàm candidate_gen của thuật toán Apriori; điểm khác biệt chính nằm ở bước 3 khi thêm 1 node ứng viên hợp lệ sẽ có 2 thao tác chính liên quan tới cây. Sau đây là các bước mà hàm này thực hiện
  - Bước 1: với mỗi cặp tập phổ biến k-phần tử (large_itemset[i], large_itemset[j]), i ≠ j sao cho k-1 phần tử đầu của từng tập phổ biến trong cặp phải giống nhau → tiến hành bước 2
  - Bước 2: dùng toán tử hợp (union) để hợp cặp này lại, kết quả sau khi hợp lưu vào **u**
  - Bước 3: tỉa nhánh: xét tất cả tập con (k-1)-phần tử của **u**, có tập con nào là tập không phổ biến, tức không nằm trong **large_itemset_nodes** hay không. Nếu tồn tại 1 tập con như vậy thì không chấp nhận node ứng viên **u**. Ngược lại, chấp nhận **u** và thực hiện 2 thao tác liên quan tới cây:
    - Tạo quan hệ ***cha-con***: Bổ sung node **u** vào tập node children của node **large_itemset_nodes[i]** (ưu tiên lưu vào **large_itemset_nodes[i]** hơn vào **large_itemset_nodes[j]** để thỏa mãn thứ tự từ điển)
    - Tạo quan hệ ***con-cha***, tức **node_u.parent** = **large_itemset_nodes[i]**. Vì 1 node chỉ được phép có 1 cha nên không cập nhật **node_u.parent** = **large_itemset_nodes[j]**
    - Bổ sung node u vào **candidate_nodes**
  - Bước 4: Sau khi xét xong tất cả các cặp large_itemset[i] và large_itemset[j] thỏa bước 1 thì **candidate_nodes** cũng đã chứa các node ứng viên (k+1)-phần tử hợp lệ → trả về **candidate_nodes**

III. ỨNG DỤNG 2 THUẬT TOÁN TRÊN CÁC TẬP DATASET

- Mỗi dataset có 2 file output: **[tên_dataset]_tree.txt** (output dùng thuật toán tree projection) và **[tên_dataset]_apriori.txt** (output dùng thuật toán apriori). Ví dụ: dataset foodmart.txt có 2 file output là: **foodmart_tree.txt** và **foodmart_apriori.txt**

1. Dataset foodmart.txt

- Với min_sup = 0.002421 = 0.2421% ⇔ sup.count ≥ 10 → cho ra 980 tập phổ biến, nhưng toàn bộ 980 tập này là tập one_itemset (check lại)
- Với min_sup = 0.000965 = 0.0965% ⇔ sup.count ≥ 4 → cho ra 1557 tập phổ biến, trong đó có 2 tập 2-itemset, còn lại là 1-itemset. min_sup này cũng cho thấy sự khác biệt về thời gian chạy của 2 thuật toán: thuật toán Apriori tốn ~21 phút để chạy trong khi thuật toán Tree Projection tốn ~ 17 phút để chạy

2. Dataset chess.txt

- Với min_sup = 90% → thuật toán tree projection tạo được sâu nhất tới cấp 7 (level=7); cả 2 thuật toán đều cho ra 622 tập phổ biến

3. Dataset mushrooms.txt

- Với min_sup = 40% → thuật toán tree projection tạo được sâu nhất tới cấp 6 (level=6); cả 2 thuật toán đều cho ra 505 tập phổ biến

4. Dataset retail.txt

- Với min_sup = 9% = 0.09 → thuật toán tree projection tạo được sâu nhất tới cấp 2 (level=2); cả 2 thuật toán đều cho ra 12 tập phổ biến
- Với min_sup = 5% → thuật toán tree projection tạo được sâu nhất tới cấp 3 (level=3); cả 2 thuật toán đều cho ra 16 tập phổ biến

IV. SO SÁNH 2 THUẬT TOÁN

	Apriori	Tree Projection
Điểm mạnh	- Dễ hiểu và dễ cài đặt - Dữ liệu đầu vào không cần gán nhãn - Tìm toàn bộ các luật từ dataset ban đầu - Dễ biến thể theo từng trường hợp cụ thể	- Đảm bảo không tạo node trùng trong quá trình phát sinh ứng viên → Tiết kiệm 1 vài bước tính toán so với Apriori nên nhanh hơn Apriori - Thừa kế các điểm mạnh của Apriori
Điểm yếu	- Tốn nhiều thời gian vì 1 trong các lý do sau: o Ngưỡng min_sup nhỏ → có thể rất nhiều tập phổ biến phát sinh thỏa min_sup o Số lượng tập itemset nạp vào là lớn o Duyệt lặp đi lặp lại toàn bộ cơ sở dữ liệu nhiều lần và kiểm tra một tập lớn các ứng viên bằng phương pháp so khớp mẫu o Số lượng tập hạng mục và luật được phát sinh là khổng lồ → khó khăn cho việc phân tích thông tin	- Cần định nghĩa 1 cấu trúc dữ liệu mới để lưu trữ các node - Phức tạp hơn Apriori do cần quan tâm tới quan hệ cha-con giữa các node trong cây với mỗi thao tác thêm/xóa node

V. ỨNG DỤNG TRONG KHAI THÁC DỮ LIỆU THỰC TẾ

- Nguồn dữ liệu: https://www.kaggle.com/irfanasrullah/groceries/version/2
- File groceries.csv là file miêu tả từng giỏ hàng của người mua tại 1 cửa hàng. Từng giỏ hàng tương ứng với từng dòng trong file. Ta sẽ tiến hành rút các tập phổ biến với min_sup = 1%
- Output: được thể hiện trong 2 file groceries_apriori.txt và groceries_tree.txt

VI. THAM KHẢO

[1] https://www.academia.edu/12469140/Tree_Projection_Based_Frequent_Itemset_Mining_on_Multicore_CPUs_and_GPUs

[2] https://www.engati.com/glossary/apriori-algorithm?utm_content=apriori-algorithm

[3] https://www-users.cse.umn.edu/~kumar001/dmbook/ch5_association_analysis.pdf