2025-04-25 16:09

Tags:

# Họ và Tên: Mai Phong Đăng

MSSV: 22280008

Lớp: 22KDL

# Data\_Mining\_Lab5\_report

## Tóm tắt code của thuật toán Vertical Apriori

### **Mục tiêu**

* Hàm vertical\_apriori() là một triển khai thuật toán Apriori theo hướng tiếp cận **vertical format** – tức là thay vì duyệt từng dòng giao dịch, ta theo dõi các **danh sách ID giao dịch (TID list)** tương ứng với từng item hoặc itemset. Việc xử lý và tính toán hỗ trợ thông qua phép giao tập của các TID giúp tăng hiệu quả, đặc biệt trên dữ liệu thưa (sparse).

### **Tóm tắt**

#### Bước 1: **Chuyển đổi dữ liệu đầu vào**

* Chuyển từ DataFrame nhị phân sang dạng từ điển {transaction\_id: set(items)}.

tid\_database = {}  
for tid, row in transactions.iterrows():  
 items\_in\_transaction = set(row.index[row == 1])  
 if items\_in\_transaction: # Chỉ thêm nếu không rỗng  
 tid\_database[tid] = items\_in\_transaction

#### Bước 2: **Tìm tập mục đơn phổ biến (1-itemsets)**

* Với mỗi item, ta tạo danh sách các giao dịch mà item đó xuất hiện:

item\_tids = {}  
for item in transactions.columns:  
 item\_tids[frozenset([item])] = {  
 tid for tid, items in tid\_database.items() if item in items  
}

* Sau đó, lọc ra các item có độ hỗ trợ thỏa điều kiện:

F1 = {itemset: tid\_list for itemset, tid\_list in item\_tids.items()  
 if len(tid\_list) >= abs\_min\_sup}

#### Bước 3: **Lặp để tìm các tập mục phổ biến cấp cao hơn (k-itemsets)**

* Ta sử dụng vòng lặp chính để kết hợp các tập phổ biến hiện tại nhằm tạo các ứng viên mới:

while Fk:  
 ...  
 for i in range(len(Fk\_items)):  
 for j in range(i+1, len(Fk\_items)):  
 itemset1 = set(Fk\_items[i])  
 itemset2 = set(Fk\_items[j])  
 if len(itemset1.union(itemset2)) == k:  
 candidate = frozenset(itemset1.union(itemset2))  
  
 # Kiểm tra downward closure - nếu bất kỳ tập con nào không phổ biến thì loại  
 for subset in combinations(candidate, k-1):  
 if frozenset(subset) not in Fk:  
 should\_prune = True  
 break  
   
 if not should\_prune:  
 # Giao TID list để tính hỗ trợ  
 tid\_list1 = Fk[frozenset(itemset1)]  
 tid\_list2 = Fk[frozenset(itemset2)]  
 Ck[candidate] = tid\_list1.intersection(tid\_list2)

* Cuối mỗi vòng, ta lọc ra các ứng viên đủ điều kiện hỗ trợ:

Fk = {itemset: tid\_list for itemset, tid\_list in Ck.items()  
 if len(tid\_list) >= abs\_min\_sup}

#### Bước 4: **Trả về kết quả dạng {itemset: support}**

* Sau khi thu được tất cả các tập phổ biến, ta tính lại độ hỗ trợ (support) tương đối:

result = {}  
for itemset, tid\_list in all\_frequent\_itemsets.items():  
 support = len(tid\_list) / n\_transactions  
 item\_str = ' & '.join(sorted(itemset)) if len(itemset) > 1 else next(iter(itemset))  
 result[item\_str] = support

## So Sánh kết quả với hàm có sẵn

* Sau khi cài đặt thuật toán vertical\_apriori(), ta cần kiểm tra độ chính xác bằng cách **so sánh kết quả** với thư viện uy tín – ở đây là **mlxtend**, vốn cài sẵn thuật toán Apriori truyền thống (horizontal format).

#### Bước 1: **Gọi lại thuật toán Vertical Apriori đã tự cài đặt**

frequent\_itemsets\_custom = vertical\_apriori(df\_bin, min\_support)

* Kết quả là một dictionary {itemset\_string: support\_value}
* Ta chuyển nó về dạng DataFrame để tiện xử lý:

result\_custom = pd.DataFrame(frequent\_itemsets\_custom.items(), columns=['Item', 'support'])  
result\_custom = result\_custom[result\_custom['Item'].str.count('&') >= min\_combination - 1]

* **Lọc theo độ dài itemset**, ví dụ: chỉ giữ những tập có 2 mục trở lên (A & B trở lên).

#### Bước 2: **Dùng thư viện mlxtend để tính frequent itemsets**

from mlxtend.frequent\_patterns import apriori  
  
frequent\_itemsets\_lib = apriori(df\_bin, min\_support=min\_support, use\_colnames=True)

* Output của apriori() là DataFrame chứa các cột:
  + itemsets: một tập hợp các item (frozenset)
  + support: tần suất xuất hiện tương ứng
* Ta xử lý lại để format giống với kết quả custom:

frequent\_itemsets\_lib['Item'] = frequent\_itemsets\_lib['itemsets'].apply(  
 lambda x: ' & '.join(sorted(x))  
)  
result\_lib = frequent\_itemsets\_lib[['Item', 'support']]  
result\_lib = result\_lib[result\_lib['Item'].str.count('&') >= min\_combination - 1]

#### Bước 3: **So sánh kết quả giữa hai phương pháp**

* Hiển thị bảng kết quả riêng biệt:

print("\n[Custom Vertical Apriori]")  
print(result\_custom.sort\_values(by='support', ascending=False))  
  
print("\n[Apriori - mlxtend]")  
print(result\_lib.sort\_values(by='support', ascending=False))

* Và tạo bảng giao nhau – tức là các tập mục phổ biến giống nhau giữa 2 cách:

common = pd.merge(result\_custom, result\_lib, on='Item', suffixes=('\_custom', '\_lib'))  
print("\n[Common Frequent Itemsets]")  
print(common)

#### Giải thích kết quả:

* Kết quả cho thấy thuật toán **Vertical Apriori tự cài đặt** và **Apriori từ mlxtend** đã **tìm ra cùng một itemset phổ biến** (Milk & Wine), với độ hỗ trợ hoàn toàn khớp.
* Điều này chứng tỏ thuật toán **Vertical Apriori tự cài đặt** đã hoạt động chính xác và cho kết quả tương đương với thuật toán chuẩn của thư viện mlxtend.