ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - TIN HỌC


# BTTH TUẦN 4: KHAI THÁC DỮ LIỆU

**Sinh viên thực hiện: Nguyễn Công Hoài Nam**
**Mã số sinh viên: 21280099**

Ngày 28 tháng 4 năm 2024

### 1. Load thư viện và data

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

#Load data
df = pd.read_csv('data.csv', header = None)
df
```

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | Wine | Chips | Bread | Butter | Milk | Apple |
| 1 | Wine | NaN | Bread | Butter | Milk | NaN |
| 2 | NaN | NaN | Bread | Butter | Milk | NaN |
| 3 | NaN | Chips | NaN | NaN | NaN | Apple |
| 4 | Wine | Chips | Bread | Butter | Milk | Apple |
| 5 | Wine | Chips | NaN | NaN | Milk | NaN |
| 6 | Wine | Chips | Bread | Butter | NaN | Apple |
| 7 | Wine | Chips | NaN | NaN | Milk | NaN |
| 8 | Wine | NaN | Bread | NaN | NaN | Apple |
| 9 | Wine | NaN | Bread | Butter | Milk | NaN |
| 10 | NaN | Chips | Bread | Butter | NaN | Apple |
| 11 | Wine | NaN | NaN | Butter | Milk | Apple |
| 12 | Wine | Chips | Bread | Butter | Milk | NaN |
| 13 | Wine | NaN | Bread | NaN | Milk | Apple |
| 14 | Wine | NaN | Bread | Butter | Milk | Apple |
| 15 | Wine | Chips | Bread | Butter | Milk | Apple |
| 16 | NaN | Chips | Bread | Butter | Milk | Apple |
| 17 | NaN | Chips | NaN | Butter | Milk | Apple |
| 18 | Wine | Chips | Bread | Butter | Milk | Apple |
| 19 | Wine | NaN | Bread | Butter | Milk | Apple |
| 20 | Wine | Chips | Bread | NaN | Milk | Apple |
| 21 | NaN | Chips | NaN | NaN | NaN | NaN |

Chuyển df về dạng list

```
records = []
for i in range(0,df.shape[0]):
    records.append([str(df.values[i,j]) for j in range (0, df.shape[1])])
records
```

Trong list có các giá trị 'nan' đây không phải là hạng mục của giao dịch mà là giá trị 'NaN' khi chuyển sang dạng chuỗi

```
# Ta loại bỏ nó
records = [[item for item in i if item != 'nan'] for i in records]
records
```

Kết quả:

```
[['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk'],
 ['Bread', 'Butter', 'Milk'],
 ['Chips', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Milk'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Apple'],
 ['Wine', 'Chips', 'Milk'],
 ['Wine', 'Bread', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk'],
 ['Chips', 'Bread', 'Butter', 'Apple'],
 ['Wine', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk'],
 ['Wine', 'Bread', 'Milk', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Chips', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Bread', 'Butter', 'Milk', 'Apple'],
 ['Wine', 'Chips', 'Bread', 'Milk', 'Apple'],
 ['Chips']]
```

Từ bộ data trên ta tiến hành áp dụng thuật toán apriori

### 2. Hàm Apriori từ thư viện

*a) Apriori*

Để áp dụng được thư viện ta cần mã hoá nó thành dạng 'transaction'

```
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df1 = pd.DataFrame(te_ary, columns = te.columns_)
df1
```

Kết quả:

| | Apple | Bread | Butter | Chips | Milk | Wine |
|---|---|---|---|---|---|---|
| 0 | True | True | True | True | True | True |
| 1 | False | True | True | False | True | True |
| 2 | False | True | True | False | True | False |
| 3 | True | False | False | True | False | False |
| 4 | True | True | True | True | True | True |
| 5 | False | False | False | True | True | True |
| 6 | True | True | True | True | False | True |
| 7 | False | False | False | True | True | True |
| 8 | True | True | False | False | False | True |
| 9 | False | True | True | False | True | True |
| 10 | True | True | True | True | False | False |
| 11 | True | False | True | False | True | True |
| 12 | False | True | True | True | True | True |
| 13 | True | True | False | False | True | True |
| 14 | True | True | True | False | True | True |
| 15 | True | True | True | True | True | True |
| 16 | True | True | True | True | True | False |
| 17 | True | False | True | True | True | False |
| 18 | True | True | True | True | True | True |
| 19 | True | True | True | False | True | True |
| 20 | True | True | False | True | True | True |
| 21 | False | False | False | True | False | False |

Kết quả thuật toán Apriori với min_support = 0.6

```
frequent_itemsets = apriori(df1, min_support = 0.6, use_colnames = True)
frequent_itemsets
```

| | support | itemsets |
|---|---|---|
| 0 | 0.681818 | (Apple) |
| 1 | 0.727273 | (Bread) |
| 2 | 0.681818 | (Butter) |
| 3 | 0.636364 | (Chips) |
| 4 | 0.772727 | (Milk) |
| 5 | 0.727273 | (Wine) |
| 6 | 0.636364 | (Milk, Wine) |

*b) Association Rules*

Luật kết hợp với metric = "support"

```
rules = association_rules(frequent_itemsets, metric = "support", support_only = True,
                          min_threshold=0.1)
rules = rules[['antecedents','consequents','support','confidence','lift']]
print(rules)
```

```
  antecedents consequents   support  confidence  lift
0      (Milk)      (Wine)  0.636364         NaN   NaN
1      (Wine)      (Milk)  0.636364         NaN   NaN
```

Với metric = "confidence"

```
rules = association_rules(frequent_itemsets, metric = "confidence",
                          min_threshold=0.1)
rules = rules[['antecedents','consequents','support','confidence','lift']]
print(rules)
```

```
  antecedents consequents   support  confidence      lift
0      (Milk)      (Wine)  0.636364    0.823529  1.132353
1      (Wine)      (Milk)  0.636364    0.875000  1.132353
```

### 3. Hàm Apriori từ đầu (không dùng thư viện)

Ta có cơ sở lý thuyết như sau:
- $C_k$: k-item candidate itemset (tập ứng viên có k-hạng mục)

- $F_k$: k-item frequent itemset (tập hạng mục thường xuyên được lọc ra bởi $C_k$ với 'suppport >= minimum_support')

### 4. Apriori

Vì tập $C_1$ (tập ứng viên một hạng mục) là các item duy nhất của itemsets nên ta tạo nó trước

```
def create_candidate_1_itemsets(X):
    """
    Duyệt lần lượt các giao dịch, lấy ra các hạng mục duy nhất thêm vào list c1
    """

    c1 = []
    for transaction in X:
        for t in transaction:
            # frozenset là tập không thay đổi (immutable set) tức nó không cho phép chỉnh sửa, thay đ
    ổi hay giá trị lặp
            t = frozenset([t])
            if t not in c1:
                c1.append(t)
    return c1
```

Hàm tính $F_k$

```
def create_frequent_itemsets(X, ck, min_support):
    """
    Lọc từ candidate itemset (ck) ra frequent itemset (fk)
    với min_support
    """
    # Lặp từng giao dịch và đếm số lần xuất hiện của mỗi hạng mục
    item_count = {}
    for transaction in X:
        for item in ck:
            if item.issubset(transaction):
                if item not in item_count:
                    item_count[item] = 1
                else:
                    item_count[item] += 1

    n_row = len(X)
    freq_item = []
    item_support = {}

    # Nếu support của mỗi hạng mục lớn hơn hoặc bằng
    # min_support thì thêm nó vào freq_item (fk)
    # Lưu lại support của các hạng mục (để so sánh)

    for item in item_count:
        support = item_count[item] / n_row
        if support >= min_support:
            freq_item.append(item)

        item_support[item] = support

    return freq_item, item_support
```

Hàm tính $C_k$ ($C_1$ tính riêng)

```
# Hàm tính tổ hợp từ tập hợp các phần tử cho trước
from itertools import combinations

# Hàm tạo k-item candidate itemsets
def create_candidate_k_itemsets(freq_item, k):
    ck = []

    # Với k = 1, tạo 2-item candidate itemsets (c2)
    if k == 1:
        for f1, f2 in combinations(freq_item, 2):
            item = f1 | f2 # union of two sets
            ck.append(item)
    # Với k != 1
    else:
        for f1, f2 in combinations(freq_item, 2):

            # Nếu k-itemset có k - 1 phần tử chung
            # sẽ lấy phần hợp của chúng vào k+1-itemset
            intersection = f1 & f2
            if len(intersection) == k - 1:
                item = f1 | f2
```

```
                if item not in ck:
                    ck.append(item)
    return ck
```

Từ các hàm con tạo hàm ‘apriori_from_scratch‘ tổng hợp với hai tham số
- ‘X‘ là tập hạng mục (transactions)
- ‘min_support‘ là suppport tối thiểu

```
def apriori_from_scratch(X, min_support):

    # tạo trước 1-item candidate itemsets (c1)
    c1 = create_candidate_1_itemsets(X)
    # tạo f1 (frequent itemsets) từ c1
    freq_item, item_support_dict = create_frequent_itemsets(X, c1, min_support)
    # thêm f1 vào freq_items (tập hạng mục thường xuyên)
    freq_items = [freq_item]

    # lặp từ k = 1
    k = 1
    while True:
        # lấy ra fk từ tập freq_items (ở đây là list nên idx bắt đầu từ 0)
        freq_item = freq_items[k-1]
        # tính ck+1 từ fk
        ck = create_candidate_k_itemsets(freq_item, k)
        # tính fk+1 từ ck+1 và support dict
        freq_item, item_support = create_frequent_itemsets(X, ck, min_support)
        # nếu fK+1 trống => dừng
        # tạo đk dừng ở đây để tránh thêm list rỗng vào freq_items
        if len(freq_item) == 0:
            break
        # thêm fk+1 vào freq_items
        freq_items.append(freq_item)
        # cập nhật support dict tổng
        item_support_dict.update(item_support)
        # tăng k lên một tiếp tục lặp
        k += 1

    return freq_items, item_support_dict
```

So sánh kết quả Chạy ‘apriori_from_scratch‘ với ‘min_support = 0.6‘ (giống với hàm thư viện)

```
freq_items, item_support_dict = apriori_from_scratch(records,0.6)
freq_items
```

```
[[frozenset({'Wine'}),
  frozenset({'Chips'}),
  frozenset({'Bread'}),
  frozenset({'Butter'}),
  frozenset({'Milk'}),
  frozenset({'Apple'})],
 [frozenset({'Milk', 'Wine'})]]
```

Ta có thể thấy kết quả trùng với thư viện
Nhằm kiểm chứng tính đúng đắn, ta in ra các giá trị support sắp xếp tăng dần

```
dict(sorted(item_support_dict.items(), key=lambda item: item[1], reverse=True))
```

```
{frozenset({'Milk'}): 0.7727272727272727,
 frozenset({'Wine'}): 0.7272727272727273,
 frozenset({'Bread'}): 0.7272727272727273,
 frozenset({'Butter'}): 0.6818181818181818,
 frozenset({'Apple'}): 0.6818181818181818,
 frozenset({'Chips'}): 0.6363636363636364,
 frozenset({'Milk', 'Wine'}): 0.6363636363636364,
 frozenset({'Bread', 'Wine'}): 0.5909090909090909,
 frozenset({'Bread', 'Butter'}): 0.5909090909090909,
 frozenset({'Bread', 'Milk'}): 0.5909090909090909,
 frozenset({'Butter', 'Milk'}): 0.5909090909090909,
 frozenset({'Apple', 'Bread'}): 0.5454545454545454,
 frozenset({'Butter', 'Wine'}): 0.5,
 frozenset({'Apple', 'Wine'}): 0.5,
 frozenset({'Apple', 'Butter'}): 0.5,
 frozenset({'Apple', 'Milk'}): 0.5,
 frozenset({'Chips', 'Milk'}): 0.45454545454545453,
 frozenset({'Apple', 'Chips'}): 0.45454545454545453,
 frozenset({'Chips', 'Wine'}): 0.4090909090909091,
 frozenset({'Bread', 'Chips'}): 0.4090909090909091,
 frozenset({'Butter', 'Chips'}): 0.4090909090909091}
```

Chương trình em viết chỉ lấy những set có support >= 0.6
Để dễ nhìn hơn em có viết hàm chuyển nó sang df

```
def tidy(freq_items,item_support_dict):
    supports = []
    itemsets = []
    for itemset in freq_items:
        for set in itemset:
            supports.append(item_support_dict[set])
            itemsets.append(set)
    return pd.DataFrame({'Itemset': itemsets, 'Support': supports})

tidy(freq_items,item_support_dict)
```

| | Itemset | Support |
|---|---|---|
| 0 | (Wine) | 0.727273 |
| 1 | (Chips) | 0.636364 |
| 2 | (Bread) | 0.727273 |
| 3 | (Butter) | 0.681818 |
| 4 | (Milk) | 0.772727 |
| 5 | (Apple) | 0.681818 |
| 6 | (Wine, Milk) | 0.636364 |

*a) Association Rules*

Từ 'freq_items' và 'item_dict_support' trả về từ hàm 'apriori' trên ta tạo hàm tính metric chuẩn bị cho hàm tạo luật kết hợp

Hàm compute_metric này nhận vào các tham số như sau:

‘freq_items‘: Các tập hạng mục thường xuyên.

‘item_support_dict‘: Từ điển lưu trữ hỗ trợ của từng tập hạng mục.

‘freq_set‘: Tập hạng mục hiện tại cần xem xét để tạo ra các luật.

‘subsets‘: Các phần tử con của freq_set.

‘min_threshold‘: Ngưỡng tối thiểu cho metric được chỉ định.

‘metric‘: Loại metric để đánh giá luật kết hợp (Confidence, Support, Lift).

```
def compute_metric(freq_items, item_support_dict, freq_set, subsets, min_threshold, metric):

    # Trả lỗi nếu metric không hợp lệ
    if metric not in ("Confidence", "Support", 'Lift'):
        raise ValueError("Invalid metric value")

    rules = []
    right_hand_side = []

    for rhs in subsets:
        # Tạo phần trái của quy tắc
        lhs = freq_set - rhs

        # Kiểm tra xem phần tử hợp nhất của quy tắc có đủ phổ biến không
        if lhs in item_support_dict and rhs in item_support_dict:

            # Tính các metric
            supp = item_support_dict[freq_set]
            conf = item_support_dict[freq_set] / item_support_dict[lhs]
            lift = conf / item_support_dict[rhs]
            # Chọn metric để so sánh với min_threshold
            if metric == "confidence":
                metric_value = conf
            elif metric == "lift":
                metric_value = lift
            else:
                metric_value = supp

            # Thêm quy tắc vào danh sách nếu metric vượt qua ngưỡng
            if metric_value >= min_threshold:
                rules_info = lhs, rhs, supp, conf, lift
                rules.append(rules_info)
                right_hand_side.append(rhs)

    return rules, right_hand_side
```

Hàm ‘association_rules_from_scratch‘ tạo ra luật kết hợp từ đầu (không dùng thư viện)
Các tham số lần lượt là ‘(freq_items, item_supprt_dict)‘ từ hàm ‘apriori‘, ‘min_threshold‘ là ngưỡng giữ lại luật kết hợp tuỳ thuộc vào ‘metric‘ cụ thể

```
def association_rules_from_scratch(freq_items, item_support_dict, min_threshold, metric= "confidence"
    ):

    association_rules = []

    # Lặp trong tập hạng mục thường xuyên (freq_items)
    # Lặp từ phần tử thứ 2, vì phần từ đầu là c1 chứa các
    # hạng mục đơn không thể tạo luật kết hợp

    for idx, freq_item in enumerate(freq_items[1:(len(freq_items))]):
        for freq_set in freq_item:

            # Tạo luật cho các hạng mục đơn trước
            # Tạo tập con không lặp
            subsets = [frozenset([item]) for item in freq_set]
            # Tính rules (đã qua sàng lọc theo ngưỡng và metric) và tập hạng mục phía phải (
    consequents)
            rules, right_hand_side = compute_metric(freq_items, item_support_dict,
                                                    freq_set, subsets, min_threshold, metric)
            association_rules.extend(rules)

            # tạo luật từ 3-itemset  trở đi
            if idx != 0:
                k = 0
                while len(right_hand_side[0]) < len(freq_set) - 1:
                    ck = create_candidate_k_itemsets(right_hand_side, k = k)
```

```
                    rules, right_hand_side = compute_metric(freq_items, item_support_dict,
                                                            freq_set, ck, min_threshold, metric)
                    association_rules.extend(rules)
                    k += 1

    return association_rules
```

Kiểm tra kết quả

```
association_rules = association_rules_from_scratch(freq_items, item_support_dict, min_threshold =
    0.1, metric="confidence")
```

```
def tidy(list):
    return pd.DataFrame(list, columns=['Antecedents ', 'Consequents', 'Support', 'Confidence', 'Lift'
    ])

tidy(association_rules)
```

| | Antecedents | Consequents | Support | Confidence | Lift |
|---|---|---|---|---|---|
| 0 | (Milk) | (Wine) | 0.636364 | 0.823529 | 1.132353 |
| 1 | (Wine) | (Milk) | 0.636364 | 0.875000 | 1.132353 |

Giống với hàm thư viện