DM_W10_24042024

Phạm Ngọc Hải

April 24, 2024

1 Nội dung thực hành:

1. Download dữ liệu QtyT40I10D100K, Mushroom, Connect4.
   - https://archive.ics.uci.edu/dataset/73/mushroom
   - https://archive.ics.uci.edu/ml/datasets/QtyT40I10D100K
   - https://archive.ics.uci.edu/dataset/26/connect+4
2. Chạy thuật toán 'Apriori', 'FP-Growth' và 'Max Miner' với các giá trị MinSupp và MinConf tìm tập mục phổ biến và số lượng luật,thời gian chạy với các dữ liệu trên.
   - https://github.com/Mazeofthemind/MaxMiner
   - https://ongxuanhong.wordpress.com/2015/08/24/apriori-va-fp-growth-voi-tap-du-lieu-plants/
3. Đánh giá hiệu năng và so sánh các thuật toán.

2 Download data

[ ]:
```
# %pip install ucimlrepo
```

[ ]:
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
mushroom = fetch_ucirepo(id=73)

# data (as pandas dataframes)
X_mushroom = mushroom.data.features
y_mushroom = mushroom.data.targets

# metadata
print(mushroom.metadata)

# variable information
print(mushroom.variables)
```

```
{'uci_id': 73, 'name': 'Mushroom', 'repository_url':
'https://archive.ics.uci.edu/dataset/73/mushroom', 'data_url':
'https://archive.ics.uci.edu/static/public/73/data.csv', 'abstract': 'From
Audobon Society Field Guide; mushrooms described in terms of physical
characteristics; classification: poisonous or edible', 'area': 'Biology',
'tasks': ['Classification'], 'characteristics': ['Multivariate'],
```

'num_instances': 8124, 'num_features': 22, 'feature_types': ['Categorical'],
'demographics': [], 'target_col': ['poisonous'], 'index_col': None,
'has_missing_values': 'yes', 'missing_values_symbol': 'NaN',
'year_of_dataset_creation': 1981, 'last_updated': 'Thu Aug 10 2023',
'dataset_doi': '10.24432/C5959T', 'creators': [], 'intro_paper': None,
'additional_info': {'summary': "This data set includes descriptions of
hypothetical samples corresponding to 23 species of gilled mushrooms in the
Agaricus and Lepiota Family (pp. 500-525). Each species is identified as
definitely edible, definitely poisonous, or of unknown edibility and not
recommended. This latter class was combined with the poisonous one. The Guide
clearly states that there is no simple rule for determining the edibility of a
mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and
Ivy.", 'purpose': None, 'funded_by': None, 'instances_represent': None,
'recommended_data_splits': None, 'sensitive_data': None,
'preprocessing_description': None, 'variable_info': '   1. cap-shape:
bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s\r\n   2. cap-surface:
fibrous=f,grooves=g,scaly=y,smooth=s\r\n   3. cap-color:
brown=n,buff=b,cinnamon=c,gray=g,green=r,
pink=p,purple=u,red=e,white=w,yellow=y\r\n   4. bruises?:
bruises=t,no=f\r\n   5. odor:
almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s\r\n
6. gill-attachment: attached=a,descending=d,free=f,notched=n\r\n   7.
gill-spacing: close=c,crowded=w,distant=d\r\n   8. gill-size:
broad=b,narrow=n\r\n   9. gill-color:
black=k,brown=n,buff=b,chocolate=h,gray=g,
green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y\r\n   10. stalk-shape:
enlarging=e,tapering=t\r\n   11. stalk-root:
bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?\r\n   12.
stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s\r\n   13. stalk-
surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s\r\n   14. stalk-color-
above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,
pink=p,red=e,white=w,yellow=y\r\n   15. stalk-color-below-ring:
brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y\r\n
16. veil-type: partial=p,universal=u\r\n   17. veil-color:
brown=n,orange=o,white=w,yellow=y\r\n   18. ring-number:
none=n,one=o,two=t\r\n   19. ring-type:
cobwebby=c,evanescent=e,flaring=f,large=l,
none=n,pendant=p,sheathing=s,zone=z\r\n   20. spore-print-color:
black=k,brown=n,buff=b,chocolate=h,green=r,
orange=o,purple=u,white=w,yellow=y\r\n   21. population:
abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y\r\n   22.
habitat: grasses=g,leaves=l,meadows=m,paths=p,
urban=u,waste=w,woods=d', 'citation': None}}

	name	role	type	demographic	\
0	poisonous	Target	Categorical	None	
1	cap-shape	Feature	Categorical	None	
2	cap-surface	Feature	Categorical	None	
3	cap-color	Feature	Binary	None	

```
4                    bruises  Feature  Categorical        None
5                       odor  Feature  Categorical        None
6            gill-attachment  Feature  Categorical        None
7               gill-spacing  Feature  Categorical        None
8                  gill-size  Feature  Categorical        None
9                 gill-color  Feature  Categorical        None
10               stalk-shape  Feature  Categorical        None
11                stalk-root  Feature  Categorical        None
12  stalk-surface-above-ring  Feature  Categorical        None
13  stalk-surface-below-ring  Feature  Categorical        None
14    stalk-color-above-ring  Feature  Categorical        None
15    stalk-color-below-ring  Feature  Categorical        None
16                 veil-type  Feature       Binary        None
17                veil-color  Feature  Categorical        None
18               ring-number  Feature  Categorical        None
19                 ring-type  Feature  Categorical        None
20         spore-print-color  Feature  Categorical        None
21                population  Feature  Categorical        None
22                   habitat  Feature  Categorical        None

                                          description units missing_values
0                                                    None  None             no
1   bell=b,conical=c,convex=x,flat=f, knobbed=k,su…  None            no
2               fibrous=f,grooves=g,scaly=y,smooth=s  None             no
3   brown=n,buff=b,cinnamon=c,gray=g,green=r, pink…  None            no
4                                  bruises=t,no=f  None             no
5   almond=a,anise=l,creosote=c,fishy=y,foul=f, mu…  None            no
6            attached=a,descending=d,free=f,notched=n  None             no
7                        close=c,crowded=w,distant=d  None             no
8                                broad=b,narrow=n  None             no
9   black=k,brown=n,buff=b,chocolate=h,gray=g, gre…  None            no
10                           enlarging=e,tapering=t  None             no
11  bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,…  None           yes
12                fibrous=f,scaly=y,silky=k,smooth=s  None             no
13                fibrous=f,scaly=y,silky=k,smooth=s  None             no
14  brown=n,buff=b,cinnamon=c,gray=g,orange=o, pin…  None            no
15  brown=n,buff=b,cinnamon=c,gray=g,orange=o, pin…  None            no
16                             partial=p,universal=u  None             no
17                brown=n,orange=o,white=w,yellow=y  None             no
18                                none=n,one=o,two=t  None             no
19  cobwebby=c,evanescent=e,flaring=f,large=l, non…  None            no
20  black=k,brown=n,buff=b,chocolate=h,green=r, or…  None            no
21  abundant=a,clustered=c,numerous=n, scattered=s…  None            no
22  grasses=g,leaves=l,meadows=m,paths=p, urban=u,…  None            no
```

[ ]:
```
from ucimlrepo import fetch_ucirepo
```

```python
# fetch dataset
connect_4 = fetch_ucirepo(id=26)

# data (as pandas dataframes)
X_connect_4 = connect_4.data.features
y_connect_4 = connect_4.data.targets

# metadata
print(connect_4.metadata)

# variable information
print(connect_4.variables)
```

```
{'uci_id': 26, 'name': 'Connect-4', 'repository_url':
'https://archive.ics.uci.edu/dataset/26/connect+4', 'data_url':
'https://archive.ics.uci.edu/static/public/26/data.csv', 'abstract': 'Contains
connect-4 positions', 'area': 'Games', 'tasks': ['Classification'],
'characteristics': ['Multivariate', 'Spatial'], 'num_instances': 67557,
'num_features': 42, 'feature_types': ['Categorical'], 'demographics': [],
'target_col': ['class'], 'index_col': None, 'has_missing_values': 'no',
'missing_values_symbol': None, 'year_of_dataset_creation': 1995, 'last_updated':
'Sat Mar 09 2024', 'dataset_doi': '10.24432/C59P43', 'creators': ['John Tromp'],
'intro_paper': None, 'additional_info': {'summary': 'This database contains all
legal 8-ply positions in the game of connect-4 in which neither player has won
yet, and in which the next move is not forced.\r\n\r\nx is the first player; o
the second.\r\n\r\nThe outcome class is the game theoretical value for the first
player.', 'purpose': None, 'funded_by': None, 'instances_represent': None,
'recommended_data_splits': None, 'sensitive_data': None,
'preprocessing_description': None, 'variable_info': 'Attribute Information:
(x=player x has taken, o=player o has taken, b=blank)\r\n\r\nThe board is
numbered like:\r\n6 . . . . . .\r\n5 . . . . . .\r\n4 . . . . . .\r\n3 . .
. . . . .\r\n2 . . . . . .\r\n1 . . . . . .\r\n a b c d e f g\r\n\r\n    1.
a1: {x,o,b}\r\n    2. a2: {x,o,b}\r\n    3. a3: {x,o,b}\r\n    4. a4:
{x,o,b}\r\n    5. a5: {x,o,b}\r\n    6. a6: {x,o,b}\r\n    7. b1: {x,o,b}\r\n
8. b2: {x,o,b}\r\n    9. b3: {x,o,b}\r\n   10. b4: {x,o,b}\r\n   11. b5:
{x,o,b}\r\n   12. b6: {x,o,b}\r\n   13. c1: {x,o,b}\r\n   14. c2: {x,o,b}\r\n
15. c3: {x,o,b}\r\n   16. c4: {x,o,b}\r\n   17. c5: {x,o,b}\r\n   18. c6:
{x,o,b}\r\n   19. d1: {x,o,b}\r\n   20. d2: {x,o,b}\r\n   21. d3: {x,o,b}\r\n
22. d4: {x,o,b}\r\n   23. d5: {x,o,b}\r\n   24. d6: {x,o,b}\r\n   25. e1:
{x,o,b}\r\n   26. e2: {x,o,b}\r\n   27. e3: {x,o,b}\r\n   28. e4: {x,o,b}\r\n
29. e5: {x,o,b}\r\n   30. e6: {x,o,b}\r\n   31. f1: {x,o,b}\r\n   32. f2:
{x,o,b}\r\n   33. f3: {x,o,b}\r\n   34. f4: {x,o,b}\r\n   35. f5: {x,o,b}\r\n
36. f6: {x,o,b}\r\n   37. g1: {x,o,b}\r\n   38. g2: {x,o,b}\r\n   39. g3:
{x,o,b}\r\n   40. g4: {x,o,b}\r\n   41. g5: {x,o,b}\r\n   42. g6: {x,o,b}\r\n
43. Class: {win,loss,draw}', 'citation': None}}
     name     role         type demographic description units missing_values
0      a1  Feature  Categorical        None        None  None             no
1      a2  Feature  Categorical        None        None  None             no
```

2	a3	Feature	Categorical	None	None	None	no
3	a4	Feature	Categorical	None	None	None	no
4	a5	Feature	Categorical	None	None	None	no
5	a6	Feature	Categorical	None	None	None	no
6	b1	Feature	Categorical	None	None	None	no
7	b2	Feature	Categorical	None	None	None	no
8	b3	Feature	Categorical	None	None	None	no
9	b4	Feature	Categorical	None	None	None	no
10	b5	Feature	Categorical	None	None	None	no
11	b6	Feature	Categorical	None	None	None	no
12	c1	Feature	Categorical	None	None	None	no
13	c2	Feature	Categorical	None	None	None	no
14	c3	Feature	Categorical	None	None	None	no
15	c4	Feature	Categorical	None	None	None	no
16	c5	Feature	Categorical	None	None	None	no
17	c6	Feature	Categorical	None	None	None	no
18	d1	Feature	Categorical	None	None	None	no
19	d2	Feature	Categorical	None	None	None	no
20	d3	Feature	Categorical	None	None	None	no
21	d4	Feature	Categorical	None	None	None	no
22	d5	Feature	Categorical	None	None	None	no
23	d6	Feature	Categorical	None	None	None	no
24	e1	Feature	Categorical	None	None	None	no
25	e2	Feature	Categorical	None	None	None	no
26	e3	Feature	Categorical	None	None	None	no
27	e4	Feature	Categorical	None	None	None	no
28	e5	Feature	Categorical	None	None	None	no
29	e6	Feature	Categorical	None	None	None	no
30	f1	Feature	Categorical	None	None	None	no
31	f2	Feature	Categorical	None	None	None	no
32	f3	Feature	Categorical	None	None	None	no
33	f4	Feature	Categorical	None	None	None	no
34	f5	Feature	Categorical	None	None	None	no
35	f6	Feature	Categorical	None	None	None	no
36	g1	Feature	Categorical	None	None	None	no
37	g2	Feature	Categorical	None	None	None	no
38	g3	Feature	Categorical	None	None	None	no
39	g4	Feature	Categorical	None	None	None	no
40	g5	Feature	Categorical	None	None	None	no
41	g6	Feature	Categorical	None	None	None	no
42	class	Target	Categorical	None	None	None	no

Bộ data: qtyt40i10d100k.zip bị lỗi (do bên họ nén bị lỗi)

3 Chạy thuật toán 'Apriori và FP-Growth' và 'Max Miner' với các giá trị MinSupp và MinConf tìm tập mục phổ biến và số lượng luật, thời gian chạy với các dữ liệu trên

3.1 Tổng quan về các giải thuật

Dưới đây là một tổng quan về **ba thuật toán khai thác luật kết hợp phổ biến** là Apriori, FP-Growth và Max-Miner (đều là 3 phương pháp học không giám sát), bao gồm: Mục đích chung, input, output, ý tưởng hướng tiếp cận và giải thuật của từng thuật toán

3.1.1 Mục đích chung:

Khai thác luật kết hợp (Association Rule Mining) từ tập dữ liệu: 1. **Khai thác mối quan hệ tương quan:** - Tìm kiếm các mối quan hệ tương quan giữa các mặt hàng trong tập dữ liệu. - Phát hiện các mẫu tương quan giữa các mặt hàng để có thể đưa ra các quyết định kinh doanh thông minh.

2. **Tạo ra các luật kết hợp:**
   - Tạo ra các luật kết hợp dạng "Nếu A xuất hiện, thì B cũng thường xuất hiện", hoặc "Nếu A và B xuất hiện cùng nhau, thì C thường xuất hiện".
   - Đưa ra các khuyến nghị sản phẩm, tăng hiệu suất bán hàng, hoặc cải thiện dịch vụ dựa trên các mối quan hệ tìm thấy trong tập dữ liệu.
3. **Tăng hiệu suất kinh doanh:**
   - Phân tích mẫu dữ liệu để hiểu hành vi của khách hàng.
   - Tối ưu hóa việc đặt hàng, quản lý kho, và chiến lược giá dựa trên các luật kết hợp.

Tóm lại: - Mục đích chung của ba thuật toán Apriori, FP-Growth và Max-Miner là khai thác mối quan hệ tương quan giữa các mặt hàng trong tập dữ liệu để tạo ra các luật kết hợp hữu ích giúp tăng hiệu suất kinh doanh và đưa ra các quyết định thông minh dựa trên dữ liệu.

3.1.2 Apriori Algorithm:

Input:

- Tập dữ liệu gồm danh sách các giao dịch (transaction), trong đó mỗi giao dịch là một tập hợp các mặt hàng (itemset).
- Một ngưỡng độ tin cậy (confidence threshold).

Output:

- Các luật kết hợp có độ tin cậy cao, biểu diễn các mối quan hệ tương quan giữa các mặt hàng.

Ý tưởng hướng tiếp cận:

- Apriori sử dụng phương pháp "tăng cường" (incremental) để tìm kiếm tất cả các luật kết hợp có độ tin cậy thỏa mãn.

Giải thuật:

1. Bước 1: Xây dựng tập hợp các mặt hàng (itemsets) có 1 phần tử.

2. Bước 2: Tạo các tập hợp mặt hàng có kích thước lớn hơn bằng cách kết hợp các tập hợp mặt hàng nhỏ hơn.
3. Bước 3: Loại bỏ các tập hợp mặt hàng không đạt được ngưỡng độ tin cậy.
4. Lặp lại bước 2 và 3 cho đến khi không thể tạo thêm các tập hợp mặt hàng mới.

3.1.3 FP-Growth Algorithm:

Input:

- Tập dữ liệu gồm danh sách các giao dịch (transaction), trong đó mỗi giao dịch là một tập hợp các mặt hàng (itemset).

Output:

- Các luật kết hợp có độ tin cậy cao, biểu diễn các mối quan hệ tương quan giữa các mặt hàng.

Ý tưởng hướng tiếp cận:

- Sử dụng cấu trúc dữ liệu cây FP-Tree để biểu diễn tập dữ liệu và tìm kiếm các tập hợp mặt hàng.

Giải thuật:

1. Bước 1: Xây dựng cây FP-Tree từ tập dữ liệu.
2. Bước 2: Tạo bảng tham chiếu tiện ích (header table) từ cây FP-Tree.
3. Bước 3: Duyệt cây FP-Tree để trích xuất các tập hợp mặt hàng thỏa mãn độ tin cậy.

3.1.4 Max-Miner Algorithm:

Input:

- Tập dữ liệu gồm danh sách các giao dịch (transaction), trong đó mỗi giao dịch là một tập hợp các mặt hàng (itemset).

Output:

- Các tập hợp mặt hàng con lớn nhất.

Ý tưởng hướng tiếp cận:

- Tìm kiếm tất cả các tập hợp mặt hàng con lớn nhất trong tập dữ liệu.

Giải thuật:

1. Bước 1: Tìm tất cả các tập hợp mặt hàng con lớn nhất từ tập dữ liệu.
2. Bước 2: Loại bỏ các tập hợp mặt hàng con không lớn nhất.

3.1.5 So sánh ba thuật toán:

Hiệu suất:

- Apriori: Có thể chậm khi tập dữ liệu lớn vì phải duyệt qua nhiều lần.

- FP-Growth: Hiệu suất cao hơn Apriori do sử dụng cấu trúc cây FP-Tree.
- Max-Miner: Hiệu suất cao hơn Apriori và tốn ít tài nguyên hơn vì tìm kiếm các tập hợp mặt hàng con lớn nhất.

Tính toán:

- Apriori: Sử dụng phương pháp tăng cường.
- FP-Growth: Sử dụng cây FP-Tree để biểu diễn tập dữ liệu.
- Max-Miner: Tìm kiếm tất cả các tập hợp mặt hàng con lớn nhất.

Output:

- Apriori và FP-Growth: Các luật kết hợp có độ tin cậy cao.
- Max-Miner: Các tập hợp mặt hàng con lớn nhất.

Tóm lại:

- Apriori: Đơn giản, dễ hiểu, nhưng tốn nhiều tài nguyên tính toán.
- FP-Growth: Hiệu suất cao hơn Apriori, ít tốn tài nguyên hơn.
- Max-Miner: Hiệu suất cao, tốn ít tài nguyên, nhưng chỉ tìm kiếm các tập hợp mặt hàng con lớn nhất.

3.2 Triển khai giải thuật

3.2.1 Xử lý dữ liệu cho phù hợp

Chuyển sang dạng one-hot encoding

[ ]:
```
import pandas as pd

# Chuyển đổi dữ liệu sang dạng one-hot encoding
X_mushroom_encoded = pd.get_dummies(X_mushroom)
```

3.2.2 Cách 1: Sử dụng thư viện

[ ]:
```
# %pip install mlxtend
```

Apriori

Xác định qua cách sử dụng thư viện Tìm các luật kết hợp

[ ]:
```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# Sử dụng giải thuật Apriori để tìm các luật kết hợp
frequent_itemsets = apriori(X_mushroom_encoded, min_support=0.5,␣
 ↪use_colnames=True)

# Tìm các luật kết hợp
```

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# In ra các luật kết hợp
print(len(rules), rules[:10])
```

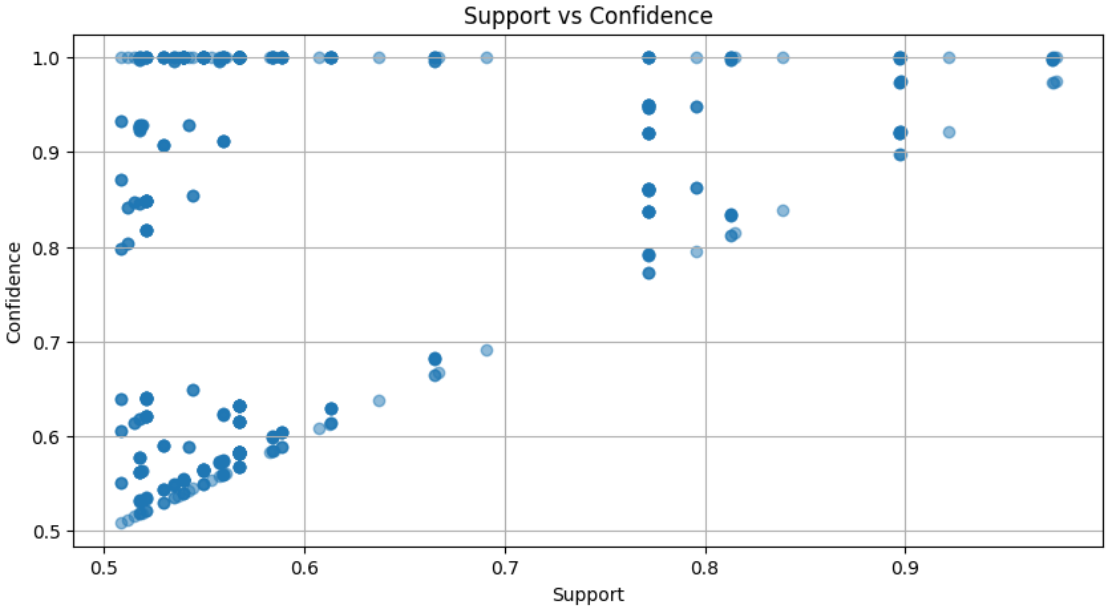
```
708                   antecedents                 consequents  antecedent
support  \
0                   (bruises_f)               (veil-type_p)            0.584441
1                 (veil-type_p)                 (bruises_f)            1.000000
2               (ring-number_o)                 (bruises_f)            0.921713
3                   (bruises_f)             (ring-number_o)            0.584441
4           (gill-attachment_f)             (stalk-shape_t)            0.974151
5               (stalk-shape_t)          (gill-attachment_f)            0.567208
6           (gill-attachment_f)  (stalk-color-above-ring_w)            0.974151
7   (stalk-color-above-ring_w)          (gill-attachment_f)            0.549483
8           (gill-attachment_f)  (stalk-color-below-ring_w)            0.974151
9   (stalk-color-below-ring_w)          (gill-attachment_f)            0.539636

   consequent support   support  confidence      lift  leverage  conviction  \
0            1.000000  0.584441    1.000000  1.000000  0.000000         inf
1            0.584441  0.584441    0.584441  1.000000  0.000000    1.000000
2            0.584441  0.542590    0.588675  1.007245  0.003903    1.010294
3            0.921713  0.542590    0.928391  1.007245  0.003903    1.093249
4            0.567208  0.567208    0.582259  1.026535  0.014662    1.036030
5            0.974151  0.567208    1.000000  1.026535  0.014662         inf
6            0.549483  0.549483    0.564064  1.026535  0.014204    1.033447
7            0.974151  0.549483    1.000000  1.026535  0.014204         inf
8            0.539636  0.539636    0.553955  1.026535  0.013949    1.032103
9            0.974151  0.539636    1.000000  1.026535  0.013949         inf

   zhangs_metric
0       0.000000
1       0.000000
2       0.091874
3       0.017308
4       1.000000
5       0.059727
6       1.000000
7       0.057377
8       1.000000
9       0.056150
```

Trực quan hóa các luật thu được

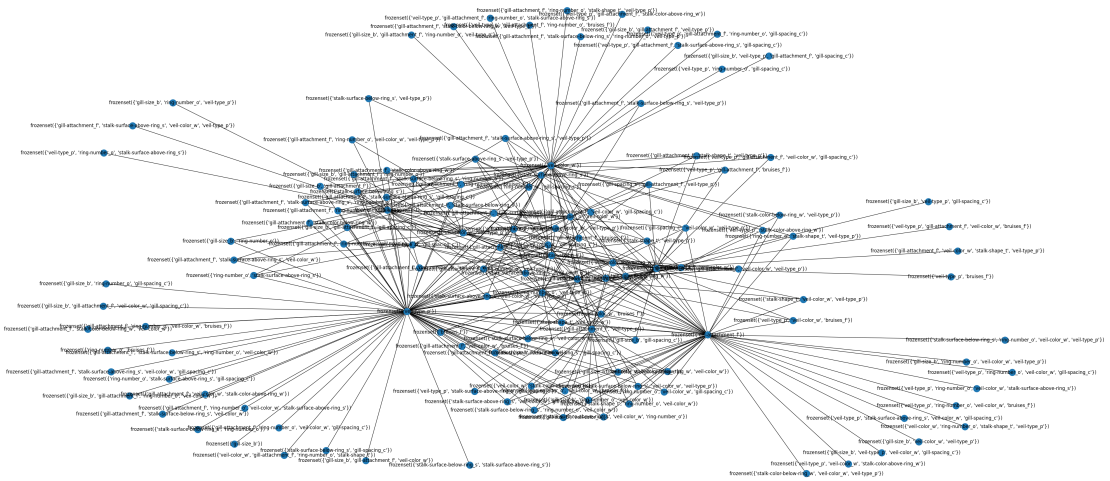
[ ]:
```
import matplotlib.pyplot as plt

# Trực quan hóa luật kết hợp bằng Scatter plot
plt.figure(figsize=(10,5))
```

```
plt.scatter(rules['support'], rules['confidence'], alpha=0.5)
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Support vs Confidence')
plt.grid()
plt.show()
```

```
/home/harito/venv/py/lib/python3.11/site-packages/pyparsing.py:108:
DeprecationWarning: module 'sre_constants' is deprecated
  import sre_constants
```



[ ]:
```
import networkx as nx

# Tạo đồ thị từ các luật kết hợp
G = nx.from_pandas_edgelist(rules, 'antecedents', 'consequents')

# Vẽ đồ thị
plt.figure(figsize=(40,20))
nx.draw(G, with_labels=True)
plt.show()
```



Sử dụng các luật tìm được để dự đoán cho record mới

```
def predict_with_association_rules(new_data_encoded, rules):
    predictions = {}

    for index, row in new_data_encoded.iterrows():
        predicted_labels = set()
        for _, rule in rules.iterrows():
            if rule['antecedents'].issubset(row.index) and rule['consequents'].issubset(row.inc
                predicted_labels.update(rule['consequents'])

        predictions[index] = predicted_labels

    return predictions

# Dự đoán sử dụng luật kết hợp
predictions = predict_with_association_rules(new_data_encoded, rules)

# Hiển thị dự đoán
for index, labels in predictions.items():
    print(f"Index: {index}, Predicted labels: {labels}")
```

Thử nghiệm thuật toán với các gái trị min_support, min_confidence khác nhau

[ ]:
```
import time

# Chức năng tính thời gian chạy
def run_apriori(min_support, min_confidence):
    start_time = time.time()
```

```
    # Sử dụng giải thuật Apriori để tìm các tập mục phổ biến
    frequent_itemsets = apriori(X_mushroom_encoded, min_support=min_support,␣
 ↪use_colnames=True)

    # Sử dụng các tập mục phổ biến để tạo ra các luật kết hợp
    rules = association_rules(frequent_itemsets, metric="confidence",␣
 ↪min_threshold=min_confidence)

    end_time = time.time()
    run_time = end_time - start_time

    return len(frequent_itemsets), len(rules), run_time

# Giá trị min_support và min_confidence
min_support_values = [0.5, 0.6, 0.7]
min_confidence_values = [0.5, 0.6, 0.7]

# Lặp qua các giá trị min_support và min_confidence và tính số lượng tập mục␣
 ↪phổ biến, số lượng luật và thời gian chạy
for min_support in min_support_values:
    for min_confidence in min_confidence_values:
        num_itemsets, num_rules, run_time = run_apriori(min_support,␣
 ↪min_confidence)
        print(f"MinSupp = {min_support}, MinConf = {min_confidence}:")
        print(f"Number of frequent itemsets: {num_itemsets}")
        print(f"Number of association rules: {num_rules}")
        print(f"Runtime: {run_time} seconds")
        print()
```

```
MinSupp = 0.5, MinConf = 0.5:
Number of frequent itemsets: 151
Number of association rules: 1146
Runtime: 0.04031634330749512 seconds

MinSupp = 0.5, MinConf = 0.6:
Number of frequent itemsets: 151
Number of association rules: 877
Runtime: 0.04703092575073242 seconds

MinSupp = 0.5, MinConf = 0.7:
Number of frequent itemsets: 151
Number of association rules: 666
Runtime: 0.027643918991088867 seconds

MinSupp = 0.6, MinConf = 0.5:
Number of frequent itemsets: 51
Number of association rules: 266
```

```
Runtime: 0.018151521682739258 seconds

MinSupp = 0.6, MinConf = 0.6:
Number of frequent itemsets: 51
Number of association rules: 266
Runtime: 0.017128944396972656 seconds

MinSupp = 0.6, MinConf = 0.7:
Number of frequent itemsets: 51
Number of association rules: 223
Runtime: 0.023814678192138672 seconds

MinSupp = 0.7, MinConf = 0.5:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.022629737854003906 seconds

MinSupp = 0.7, MinConf = 0.6:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.02228260040283203 seconds

MinSupp = 0.7, MinConf = 0.7:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.02294754981994629 seconds
```

FP-Growth Tìm các luật kết hợp

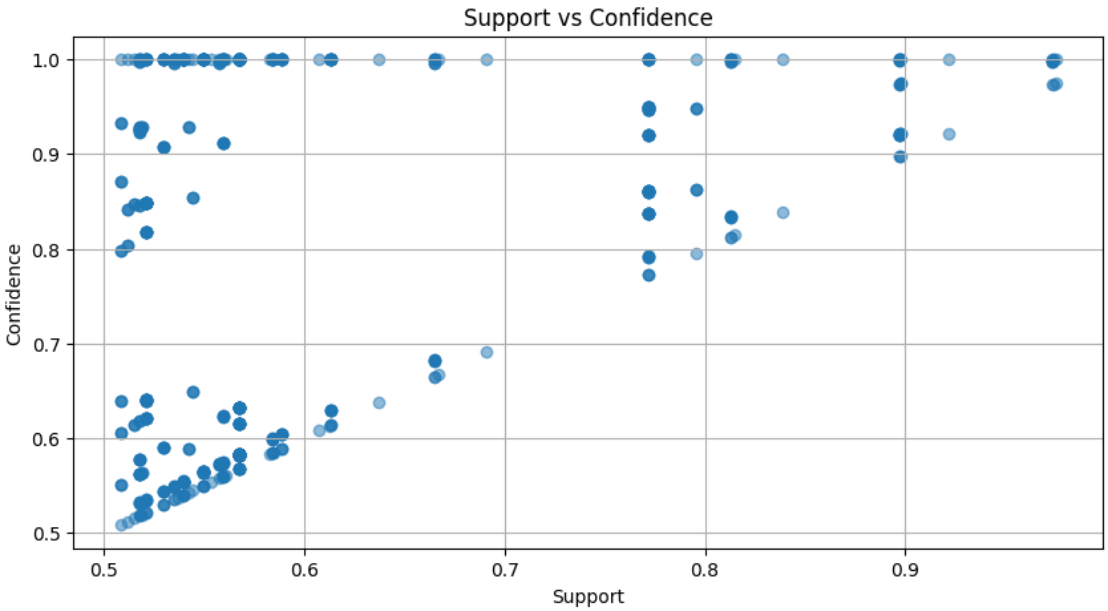
[ ]:
```python
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules

# Sử dụng giải thuật fpgrowth để tìm các luật kết hợp
frequent_itemsets_fpgrowth = fpgrowth(X_mushroom_encoded, min_support=0.5,␣
 ↪use_colnames=True)

# Tìm các luật kết hợp
rules = association_rules(frequent_itemsets_fpgrowth, metric="lift",␣
 ↪min_threshold=1)

# In ra các luật kết hợp
print(len(rules), rules[:10])
```

```
708                         antecedents                 consequents  \
0                     (veil-color_w)                (veil-type_p)
1                      (veil-type_p)               (veil-color_w)
2                (gill-attachment_f)                (veil-type_p)
```

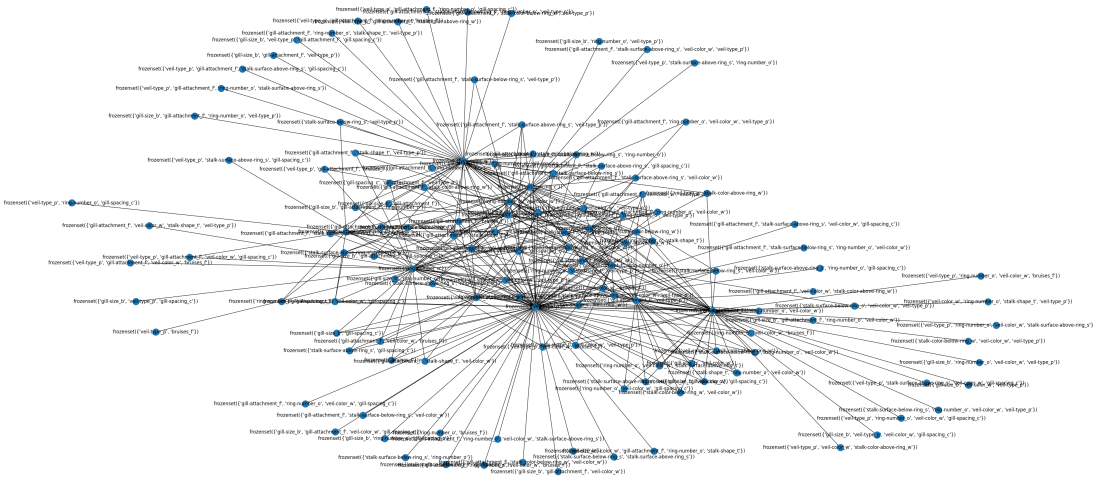
```
3                     (veil-type_p)          (gill-attachment_f)
4                (gill-attachment_f)               (veil-color_w)
5                     (veil-color_w)          (gill-attachment_f)
6  (gill-attachment_f, veil-color_w)                (veil-type_p)
7   (gill-attachment_f, veil-type_p)               (veil-color_w)
8        (veil-color_w, veil-type_p)          (gill-attachment_f)
9                (gill-attachment_f)  (veil-color_w, veil-type_p)

   antecedent support  consequent support   support  confidence      lift  \
0            0.975382            1.000000  0.975382    1.000000  1.000000
1            1.000000            0.975382  0.975382    0.975382  1.000000
2            0.974151            1.000000  0.974151    1.000000  1.000000
3            1.000000            0.974151  0.974151    0.974151  1.000000
4            0.974151            0.975382  0.973166    0.998989  1.024203
5            0.975382            0.974151  0.973166    0.997728  1.024203
6            0.973166            1.000000  0.973166    1.000000  1.000000
7            0.974151            0.975382  0.973166    0.998989  1.024203
8            0.975382            0.974151  0.973166    0.997728  1.024203
9            0.974151            0.975382  0.973166    0.998989  1.024203

   leverage  conviction  zhangs_metric
0  0.000000         inf       0.000000
1  0.000000    1.000000       0.000000
2  0.000000         inf       0.000000
3  0.000000    1.000000       0.000000
4  0.022997   24.353767       0.914199
5  0.022997   11.379452       0.959909
6  0.000000         inf       0.000000
7  0.022997   24.353767       0.914199
8  0.022997   11.379452       0.959909
9  0.022997   24.353767       0.914199
```

Trực quan hóa các luật thu được

[ ]:
```python
import matplotlib.pyplot as plt

# Trực quan hóa luật kết hợp bằng Scatter plot
plt.figure(figsize=(10,5))
plt.scatter(rules['support'], rules['confidence'], alpha=0.5)
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Support vs Confidence')
plt.grid()
plt.show()
```



[ ]:
```
import networkx as nx

# Tạo đồ thị từ các luật kết hợp
G = nx.from_pandas_edgelist(rules, 'antecedents', 'consequents')

# Vẽ đồ thị
plt.figure(figsize=(40,20))
nx.draw(G, with_labels=True)
plt.show()
```



Sử dụng các luật tìm được để dự đoán cho record mới

```
def predict_with_association_rules(new_data_encoded, rules):
    predictions = {}

    for index, row in new_data_encoded.iterrows():
        predicted_labels = set()
        for _, rule in rules.iterrows():
            if rule['antecedents'].issubset(row.index) and rule['consequents'].issubset(row.inc
                predicted_labels.update(rule['consequents'])

        predictions[index] = predicted_labels

    return predictions

# Dự đoán sử dụng luật kết hợp
predictions = predict_with_association_rules(new_data_encoded, rules)

# Hiển thị dự đoán
for index, labels in predictions.items():
    print(f"Index: {index}, Predicted labels: {labels}")
```

Thử nghiệm thuật toán với các gái trị min_support, min_confidence khác nhau

[ ]:
```
import time

# Chức năng tính thời gian chạy FP-Growth
def run_fpgrowth(min_support, min_confidence):
    start_time = time.time()

    # Sử dụng giải thuật FP-Growth để tìm các tập mục phổ biến
    frequent_itemsets = fpgrowth(X_mushroom_encoded, min_support=min_support,␣
 ↪use_colnames=True)

    # Sử dụng các tập mục phổ biến để tạo ra các luật kết hợp
    rules = association_rules(frequent_itemsets, metric="confidence",␣
 ↪min_threshold=min_confidence)

    end_time = time.time()
    run_time = end_time - start_time

    return len(frequent_itemsets), len(rules), run_time

# Giá trị min_support và min_confidence
min_support_values = [0.5, 0.6, 0.7]
min_confidence_values = [0.5, 0.6, 0.7]
```

```python
# Lặp qua các giá trị min_support và min_confidence và tính số lượng tập mục␣
 ↪phổ biến, số lượng luật và thời gian chạy
for min_support in min_support_values:
    for min_confidence in min_confidence_values:
        num_itemsets, num_rules, run_time = run_fpgrowth(min_support,␣
 ↪min_confidence)
        print(f"MinSupp = {min_support}, MinConf = {min_confidence}:")
        print(f"Number of frequent itemsets: {num_itemsets}")
        print(f"Number of association rules: {num_rules}")
        print(f"Runtime: {run_time} seconds")
        print()
```

```
MinSupp = 0.5, MinConf = 0.5:
Number of frequent itemsets: 151
Number of association rules: 1146
Runtime: 0.24596738815307617 seconds

MinSupp = 0.5, MinConf = 0.6:
Number of frequent itemsets: 151
Number of association rules: 877
Runtime: 0.1521461009979248 seconds

MinSupp = 0.5, MinConf = 0.7:
Number of frequent itemsets: 151
Number of association rules: 666
Runtime: 0.18129968643188477 seconds

MinSupp = 0.6, MinConf = 0.5:
Number of frequent itemsets: 51
Number of association rules: 266
Runtime: 0.12752079963684082 seconds

MinSupp = 0.6, MinConf = 0.6:
Number of frequent itemsets: 51
Number of association rules: 266
Runtime: 0.15660381317138672 seconds

MinSupp = 0.6, MinConf = 0.7:
Number of frequent itemsets: 51
Number of association rules: 223
Runtime: 0.13071727752685547 seconds

MinSupp = 0.7, MinConf = 0.5:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.11772871017456055 seconds
```

```
MinSupp = 0.7, MinConf = 0.6:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.1594688892364502 seconds

MinSupp = 0.7, MinConf = 0.7:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.25206804275512695 seconds
```

Max Miner Vì Max Miner chưa có code thư viện triển khai vậy nên hãy xem phần code build from scratch (tự code thuật toán) ở phía dưới để biết thêm thông tin.

3.2.3 Cách 2: Build from scratch

Apriori

[ ]:
```python
from __future__ import division, print_function
import numpy as np
import itertools


class Rule:
    def __init__(self, antecedent, concequent, confidence, support):
        self.antecedent = antecedent
        self.concequent = concequent
        self.confidence = confidence
        self.support = support


class Apriori:
    """A method for determining frequent itemsets in a transactional database
    and
    also for generating rules for those itemsets.

    Parameters:
    -----------
    min_sup: float
        The minimum fraction of transactions an itemets needs to
        occur in to be deemed frequent
    min_conf: float:
        The minimum fraction of times the antecedent needs to imply
        the concequent to justify rule
    """

    def __init__(self, min_sup=0.3, min_conf=0.81):
```

```
        self.min_sup = min_sup
        self.min_conf = min_conf
        self.freq_itemsets = None  # List of freqeuent itemsets
        self.transactions = None  # List of transactions

    def _calculate_support(self, itemset):
        count = 0
        for transaction in self.transactions:
            if self._transaction_contains_items(transaction, itemset):
                count += 1
        support = count / len(self.transactions)
        return support

    def _get_frequent_itemsets(self, candidates):
        """Prunes the candidates that are not frequent => returns list with
        only frequent itemsets"""
        frequent = []
        # Find frequent items
        for itemset in candidates:
            support = self._calculate_support(itemset)
            if support >= self.min_sup:
                frequent.append(itemset)
        return frequent

    def _has_infrequent_itemsets(self, candidate):
        """True or false depending on the candidate has any
        subset with size k - 1 that is not in the frequent itemset"""
        k = len(candidate)
        # Find all combinations of size k-1 in candidate
        # E.g [1,2,3] => [[1,2],[1,3],[2,3]]
        subsets = list(itertools.combinations(candidate, k - 1))
        for t in subsets:
            # t - is tuple. If size == 1 get the element
            subset = list(t) if len(t) > 1 else t[0]
            if not subset in self.freq_itemsets[-1]:
                return True
        return False

    def _generate_candidates(self, freq_itemset):
        """Joins the elements in the frequent itemset and prunes
        resulting sets if they contain subsets that have been determined
        to be infrequent."""
        candidates = []
        for itemset1 in freq_itemset:
            for itemset2 in freq_itemset:
                # Valid if every element but the last are the same
                # and the last element in itemset1 is smaller than the last
```

```
                # in itemset2
                valid = False
                single_item = isinstance(itemset1, int)
                if single_item and itemset1 < itemset2:
                    valid = True
                elif (
                    not single_item
                    and np.array_equal(itemset1[:-1], itemset2[:-1])
                    and itemset1[-1] < itemset2[-1]
                ):
                    valid = True

                if valid:
                    # JOIN: Add the last element in itemset2 to itemset1 to
                    # create a new candidate
                    if single_item:
                        candidate = [itemset1, itemset2]
                    else:
                        candidate = itemset1 + [itemset2[-1]]
                    # PRUNE: Check if any subset of candidate have been␣
↪determined
                    # to be infrequent
                    infrequent = self._has_infrequent_itemsets(candidate)
                    if not infrequent:
                        candidates.append(candidate)
        return candidates

    def _transaction_contains_items(self, transaction, items):
        """True or false depending on each item in the itemset is
        in the transaction"""
        # If items is in fact only one item
        if isinstance(items, int):
            return items in transaction
        # Iterate through list of items and make sure that
        # all items are in the transaction
        for item in items:
            if item not in transaction:
                return False
        return True

    def find_frequent_itemsets(self, transactions):
        """Returns the set of frequent itemsets in the list of transactions"""
        self.transactions = transactions
        # Get all unique items in the transactions
        unique_items = set(
            item for transaction in self.transactions for item in transaction
        )
```

```
        # Get the frequent items
        self.freq_itemsets = [self._get_frequent_itemsets(unique_items)]
        while True:
            # Generate new candidates from last added frequent itemsets
            candidates = self._generate_candidates(self.freq_itemsets[-1])
            # Get the frequent itemsets among those candidates
            frequent_itemsets = self._get_frequent_itemsets(candidates)

            # If there are no frequent itemsets we're done
            if not frequent_itemsets:
                break

            # Add them to the total list of frequent itemsets and start over
            self.freq_itemsets.append(frequent_itemsets)

        # Flatten the array and return every frequent itemset
        frequent_itemsets = [
            itemset for sublist in self.freq_itemsets for itemset in sublist
        ]
        return frequent_itemsets

    def _rules_from_itemset(self, initial_itemset, itemset):
        """Recursive function which returns the rules where confidence >=␣
↪min_confidence
        Starts with large itemset and recursively explores rules for subsets"""
        rules = []
        k = len(itemset)
        # Get all combinations of sub-itemsets of size k - 1 from itemset
        # E.g [1,2,3] => [[1,2],[1,3],[2,3]]
        subsets = list(itertools.combinations(itemset, k - 1))
        support = self._calculate_support(initial_itemset)
        for antecedent in subsets:
            # itertools.combinations returns tuples => convert to list
            antecedent = list(antecedent)
            antecedent_support = self._calculate_support(antecedent)
            # Calculate the confidence as sup(A and B) / sup(B), if antecedent
            # is B in an itemset of A and B
            confidence = float("{0:.2f}".format(support / antecedent_support))
            if confidence >= self.min_conf:
                # The concequent is the initial_itemset except for antecedent
                concequent = [
                    itemset for itemset in initial_itemset if not itemset in␣
↪antecedent
                ]
                # If single item => get item
                if len(antecedent) == 1:
                    antecedent = antecedent[0]
```

```
                if len(concequent) == 1:
                    concequent = concequent[0]
                # Create new rule
                rule = Rule(
                    antecedent=antecedent,
                    concequent=concequent,
                    confidence=confidence,
                    support=support,
                )
                rules.append(rule)

                # If there are subsets that could result in rules
                # recursively add rules from subsets
                if k - 1 > 1:
                    rules += self._rules_from_itemset(initial_itemset,␣
↪antecedent)
        return rules

    def generate_rules(self, transactions):
        self.transactions = transactions
        frequent_itemsets = self.find_frequent_itemsets(transactions)
        # Only consider itemsets of size >= 2 items
        frequent_itemsets = [
            itemset for itemset in frequent_itemsets if not isinstance(itemset,␣
↪int)
        ]
        rules = []
        for itemset in frequent_itemsets:
            rules += self._rules_from_itemset(itemset, itemset)
        # Remove empty values
        return rules
```

[ ]:
```
# Find frequent itemsets and generate rules using Apriori
transactions = X_mushroom.values.tolist()
apriori = Apriori(min_sup=0.5, min_conf=0.8)
frequent_itemsets = apriori.find_frequent_itemsets(transactions)
rules = apriori.generate_rules(transactions)

# Print frequent itemsets
print("Frequent itemsets:")
for itemset in frequent_itemsets:
    print(itemset)

# Print generated rules
print("\nGenerated rules:")
for rule in rules:
```

```
    print(f"{rule.antecedent} -> {rule.consequent} (confidence: {rule.
↪confidence}, support: {rule.support})")
```

FP-Growth

[ ]:
```
from __future__ import division, print_function
import numpy as np
import itertools


class FPTreeNode():
    def __init__(self, item=None, support=1):
        # 'Value' of the item
        self.item = item
        # Number of times the item occurs in a
        # transaction
        self.support = support
        # Child nodes in the FP Growth Tree
        self.children = {}


class FPGrowth():
    """A method for determining frequent itemsets in a transactional database.
    This is done by building a so called FP Growth tree, which can then be mined
    to collect the frequent itemsets. More effective than Apriori for large␣
↪transactional
    databases.

    Parameters:
    -----------
    min_sup: float
        The minimum fraction of transactions an itemets needs to
        occur in to be deemed frequent
    """
    def __init__(self, min_sup=0.3):
        self.min_sup = min_sup
        # The root of the initial FP Growth Tree
        self.tree_root = None
        # Prefixes of itemsets in the FP Growth Tree
        self.prefixes = {}
        self.frequent_itemsets = []

    # Count the number of transactions that contains item.
    def _calculate_support(self, item, transactions):
        count = 0
        for transaction in transactions:
            if item in transaction:
                count += 1
```

```
        support = count
        return support


    def _get_frequent_items(self, transactions):
        """ Returns a set of frequent items. An item is determined to
        be frequent if there are atleast min_sup transactions that contains
        it. """
        # Get all unique items in the transactions
        unique_items = set(
            item for transaction in transactions for item in transaction)
        items = []
        for item in unique_items:
            sup = self._calculate_support(item, transactions)
            if sup >= self.min_sup:
                items.append([item, sup])
        # Sort by support - Highest to lowest
        items.sort(key=lambda item: item[1], reverse=True)
        frequent_items = [[el[0]] for el in items]
        # Only return the items
        return frequent_items

    def _insert_tree(self, node, children):
        """ Recursive method which adds nodes to the tree. """
        if not children:
            return
        # Create new node as the first item in children list
        child_item = children[0]
        child = FPTreeNode(item=child_item)
        # If parent already contains item => increase the support
        if child_item in node.children:
            node.children[child.item].support += 1
        else:
            node.children[child.item] = child

        # Execute _insert_tree on the rest of the children list
        # from the new node
        self._insert_tree(node.children[child.item], children[1:])

    def _construct_tree(self, transactions, frequent_items=None):
        if not frequent_items:
            # Get frequent items sorted by support
            frequent_items = self._get_frequent_items(transactions)
        unique_frequent_items = list(
            set(item for itemset in frequent_items for item in itemset))
        # Construct the root of the FP Growth tree
        root = FPTreeNode()
```

```
        for transaction in transactions:
            # Remove items that are not frequent according to
            # unique_frequent_items
            transaction = [item for item in transaction if item in␣
↪unique_frequent_items]
            transaction.sort(key=lambda item: frequent_items.index([item]))
            self._insert_tree(root, transaction)

        return root

    def print_tree(self, node=None, indent_times=0):
        """ Recursive method which prints the FP Growth Tree """
        if not node:
            node = self.tree_root
        indent = "    " * indent_times
        print ("%s%s:%s" % (indent, node.item, node.support))
        for child_key in node.children:
            child = node.children[child_key]
            self.print_tree(child, indent_times + 1)


    def _is_prefix(self, itemset, node):
        """ Makes sure that the first item in itemset is a child of node
        and that every following item in itemset is reachable via that path """
        for item in itemset:
            if not item in node.children:
                return False
            node = node.children[item]
        return True


    def _determine_prefixes(self, itemset, node, prefixes=None):
        """ Recursive method that adds prefixes to the itemset by traversing␣
↪the
        FP Growth Tree"""
        if not prefixes:
            prefixes = []

        # If the current node is a prefix to the itemset
        # add the current prefixes value as prefix to the itemset
        if self._is_prefix(itemset, node):
            itemset_key = self._get_itemset_key(itemset)
            if not itemset_key in self.prefixes:
                self.prefixes[itemset_key] = []
            self.prefixes[itemset_key] += [{"prefix": prefixes, "support": node.
↪children[itemset[0]].support}]
```

```
        for child_key in node.children:
            child = node.children[child_key]
            # Recursive call with child as new node. Add the child item as␣
↪potential
            # prefix.
            self._determine_prefixes(itemset, child, prefixes + [child.item])


    def _get_itemset_key(self, itemset):
        """ Determines the look of the hashmap key for self.prefixes
        List of more strings than one gets joined by '-' """
        if len(itemset) > 1:
            itemset_key = "-".join(itemset)
        else:
            itemset_key = str(itemset[0])
        return itemset_key

    def _determine_frequent_itemsets(self, conditional_database, suffix):
        # Calculate new frequent items from the conditional database
        # of suffix
        frequent_items = self._get_frequent_items(conditional_database)

        cond_tree = None

        if suffix:
            cond_tree = self._construct_tree(conditional_database,␣
↪frequent_items)
            # Output new frequent itemset as the suffix added to the frequent
            # items
            self.frequent_itemsets += [el + suffix for el in frequent_items]

        # Find larger frequent itemset by finding prefixes
        # of the frequent items in the FP Growth Tree for the conditional
        # database.
        self.prefixes = {}
        for itemset in frequent_items:
            # If no suffix (first run)
            if not cond_tree:
                cond_tree = self.tree_root
            # Determine prefixes to itemset
            self._determine_prefixes(itemset, cond_tree)
            conditional_database = []
            itemset_key = self._get_itemset_key(itemset)
            # Build new conditional database
            if itemset_key in self.prefixes:
                for el in self.prefixes[itemset_key]:
                    # If support = 4 => add 4 of the corresponding prefix set
```

```
                    for _ in range(el["support"]):
                        conditional_database.append(el["prefix"])
                # Create new suffix
                new_suffix = itemset + suffix if suffix else itemset
                self._determine_frequent_itemsets(conditional_database,␣
↪suffix=new_suffix)

    def find_frequent_itemsets(self, transactions, suffix=None,␣
↪show_tree=False):
        self.transactions = transactions

        # Build the FP Growth Tree
        self.tree_root = self._construct_tree(transactions)
        if show_tree:
            print ("FP-Growth Tree:")
            self.print_tree(self.tree_root)

        self._determine_frequent_itemsets(transactions, suffix=None)

        return self.frequent_itemsets
```

[ ]:
```
transactions = X_mushroom.values.tolist()

# Find frequent itemsets using FP-Growth
fp_growth = FPGrowth(min_sup=0.5)
frequent_itemsets = fp_growth.find_frequent_itemsets(transactions)

# Print frequent itemsets
for itemset in frequent_itemsets:
    print(itemset)
```

Đã chạy thử nhưng cần cải tiến hơn nữa để tránh chạy quá lâu

Max Miner Code triển khai giải thuật

[ ]:
```
import pandas as pd

class MaxMiner:
    def __init__(self, min_support, min_confidence):
        self.min_support = min_support
        self.min_confidence = min_confidence
        self.transactions = None
        self.frequent_itemsets = []
        self.rules = []

    def fit(self, X):
        self.transactions = X.apply(lambda row: frozenset(row), axis=1)
```

```
        self._generate_frequent_itemsets()
        self._generate_rules()

    def _generate_frequent_itemsets(self):
        items = {}
        transaction_count = len(self.transactions)

        # Đếm tần suất xuất hiện của từng item
        for transaction in self.transactions:
            for item in transaction:
                if item not in items:
                    items[item] = 1
                else:
                    items[item] += 1

        # Lọc các frequent itemsets
        frequent_itemsets = {}
        for item, count in items.items():
            support = count / transaction_count
            if support >= self.min_support:
                frequent_itemsets[frozenset([item])] = support

        self.frequent_itemsets.append(frequent_itemsets)

        # Tạo các frequent itemsets lớn hơn
        k = 2
        while len(self.frequent_itemsets[k - 2]) > 0:
            candidates = self._generate_candidates(self.frequent_itemsets[k -␣
↪2], k)
            frequent_itemsets = self._get_frequent_itemsets(candidates)
            self.frequent_itemsets.append(frequent_itemsets)
            k += 1

    def _generate_candidates(self, itemsets, k):
        candidates = set()
        for itemset1 in itemsets:
            for itemset2 in itemsets:
                union = itemset1.union(itemset2)
                if len(union) == k:
                    candidates.add(union)
        return candidates

    def _get_frequent_itemsets(self, candidates):
        frequent_itemsets = {}
        for transaction in self.transactions:
            for candidate in candidates:
                if candidate.issubset(transaction):
```

```
                    if candidate not in frequent_itemsets:
                        frequent_itemsets[candidate] = 1
                    else:
                        frequent_itemsets[candidate] += 1
        transaction_count = len(self.transactions)
        frequent_itemsets = {itemset: support / transaction_count for itemset,␣
↪support in frequent_itemsets.items() if support / transaction_count >= self.
↪min_support}
        return frequent_itemsets

    def _generate_rules(self):
        for i in range(1, len(self.frequent_itemsets)):
            for itemset in self.frequent_itemsets[i]:
                self._generate_rules_from_itemset(itemset)

    def _generate_rules_from_itemset(self, itemset):
        for item in itemset:
            antecedent = itemset - {item}
            consequent = frozenset([item])
            support_itemset = self.frequent_itemsets[len(itemset) - 1][itemset]
            support_antecedent = self.frequent_itemsets[len(antecedent) -␣
↪1][antecedent]
            confidence = support_itemset / support_antecedent
            lift = confidence / self.frequent_itemsets[0][consequent]
            if confidence >= self.min_confidence and lift >= 1:
                self.rules.append((antecedent, consequent, confidence, lift))

    def print_rules(self, n=None):
        if n is None:
            n = len(self.rules)
        print("Number of rules:", len(self.rules))
        for rule in self.rules[:n]:
            print(rule)
```

[ ]:
```
# Sử dụng thuật toán Max-Miner để tìm các luật kết hợp
max_miner = MaxMiner(min_support=0.5, min_confidence=0.5)
max_miner.fit(X_mushroom_encoded)

# In ra các luật kết hợp
max_miner.print_rules(10)
```

```
Number of rules: 2
(frozenset({True}), frozenset({False}), 1.0, 1.0)
(frozenset({False}), frozenset({True}), 1.0, 1.0)
```

Thử nghiệm với nhiều min_support & min_confidence khác nhau

[ ]:
```python
import time

# Chức năng tính thời gian chạy
def run_maxminer_scratch(min_support, min_confidence):
    start_time = time.time()

    # Sử dụng giải thuật Apriori để tìm các tập mục phổ biến
    max_miner = MaxMiner(min_support=min_support, min_confidence=min_confidence)
    max_miner.fit(X_mushroom_encoded)

    frequent_itemsets = max_miner.frequent_itemsets

    # Sử dụng các tập mục phổ biến để tạo ra các luật kết hợp
    rules = max_miner.rules

    end_time = time.time()
    run_time = end_time - start_time

    return len(frequent_itemsets), len(rules), run_time

# Giá trị min_support và min_confidence
min_support_values = [0.5, 0.6, 0.7]
min_confidence_values = [0.5, 0.6, 0.7]

# Lặp qua các giá trị min_support và min_confidence và tính số lượng tập mục␣
 ↪phổ biến, số lượng luật và thời gian chạy
for min_support in min_support_values:
    for min_confidence in min_confidence_values:
        num_itemsets, num_rules, run_time = run_maxminer_scratch(min_support,␣
 ↪min_confidence)
        print(f"MinSupp = {min_support}, MinConf = {min_confidence}:")
        print(f"Number of frequent itemsets: {num_itemsets}")
        print(f"Number of association rules: {num_rules}")
        print(f"Runtime: {run_time} seconds")
        print()
```

```
MinSupp = 0.5, MinConf = 0.5:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.22956228256225586 seconds

MinSupp = 0.5, MinConf = 0.6:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.2712745666503906 seconds

MinSupp = 0.5, MinConf = 0.7:
Number of frequent itemsets: 3
```

```
Number of association rules: 2
Runtime: 0.26827263832092285 seconds

MinSupp = 0.6, MinConf = 0.5:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.22156906127929688 seconds

MinSupp = 0.6, MinConf = 0.6:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.35318660736083984 seconds

MinSupp = 0.6, MinConf = 0.7:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.4365541934967041 seconds

MinSupp = 0.7, MinConf = 0.5:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.3810884952545166 seconds

MinSupp = 0.7, MinConf = 0.6:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.37274813652038574 seconds

MinSupp = 0.7, MinConf = 0.7:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.47695422172546387 seconds
```

4 So sánh tổng kết

Với các bộ dữ liệu còn lại thực hiện tương tự nên không có gì đáng để nói

Với Apriori:
MinSupp = 0.5, MinConf = 0.5:
Number of frequent itemsets: 151
Number of association rules: 1146
Runtime: 0.04031634330749512 seconds

MinSupp = 0.5, MinConf = 0.6:
Number of frequent itemsets: 151
Number of association rules: 877
Runtime: 0.04703092575073242 seconds

MinSupp = 0.5, MinConf = 0.7:
Number of frequent itemsets: 151
Number of association rules: 666
Runtime: 0.027643918991088867 seconds

MinSupp = 0.6, MinConf = 0.5:
Number of frequent itemsets: 51
Number of association rules: 266
Runtime: 0.018151521682739258 seconds

MinSupp = 0.6, MinConf = 0.6:
Number of frequent itemsets: 51
Number of association rules: 266
Runtime: 0.017128944396972656 seconds

MinSupp = 0.6, MinConf = 0.7:
Number of frequent itemsets: 51
Number of association rules: 223
Runtime: 0.023814678192138672 seconds

MinSupp = 0.7, MinConf = 0.5:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.022629737854003906 seconds

MinSupp = 0.7, MinConf = 0.6:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.02228260040283203 seconds

MinSupp = 0.7, MinConf = 0.7:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.02294754981994629 seconds

Với FP-Growth:
MinSupp = 0.5, MinConf = 0.5:
Number of frequent itemsets: 151
Number of association rules: 1146
Runtime: 0.24596738815307617 seconds

MinSupp = 0.5, MinConf = 0.6:
Number of frequent itemsets: 151
Number of association rules: 877
Runtime: 0.1521461009979248 seconds

MinSupp = 0.5, MinConf = 0.7:
Number of frequent itemsets: 151
Number of association rules: 666
Runtime: 0.18129968643188477 seconds

MinSupp = 0.6, MinConf = 0.5:
Number of frequent itemsets: 51

Number of association rules: 266
Runtime: 0.12752079963684082 seconds

MinSupp = 0.6, MinConf = 0.6:
Number of frequent itemsets: 51
Number of association rules: 266
Runtime: 0.15660381317138672 seconds

MinSupp = 0.6, MinConf = 0.7:
Number of frequent itemsets: 51
Number of association rules: 223
Runtime: 0.13071727752685547 seconds

MinSupp = 0.7, MinConf = 0.5:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.11772871017456055 seconds

MinSupp = 0.7, MinConf = 0.6:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.1594688892364502 seconds

MinSupp = 0.7, MinConf = 0.7:
Number of frequent itemsets: 31
Number of association rules: 180
Runtime: 0.25206804275512695 seconds

Với Max-Miner:
Number of rules: 2
(frozenset({True}), frozenset({False}), 1.0, 1.0)
(frozenset({False}), frozenset({True}), 1.0, 1.0)
MinSupp = 0.5, MinConf = 0.5:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.22956228256225586 seconds

MinSupp = 0.5, MinConf = 0.6:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.2712745666503906 seconds

MinSupp = 0.5, MinConf = 0.7:
Number of frequent itemsets: 3
Number of association rules: 2
Runtime: 0.26827263832092285 seconds