Unsupervised Learning: Association Rule Learning

AAA-Python Edition

Plan

- 1- Association Rules Learning
- 2- Apriori
- 3- apriori optimization
- 4- apriori implementation
- 5- apriori implementation part 2
- 6- apriori

Concept and terminology

- The learning concerns identifying **associations** between data **attributes**.

- This association is expressed by : **associations rules** in the form:
    - If X then Y   Or in the form: X ==> Y
  - Where X and Y are subsets of attributes. These attributes are generally called: **items.** And the subsets of attributes are called: **itemsets.**
  - The data samples described by these attributes are called **transactions.**

- These rules are obtained by identifying the **frequent itemsets.**

- The rules permit to **predict** the presence of items knowing the existence of other ones.

Measures

- The following measures are used in the association rules identification process: (where X and Y are itemsets from the transactions set T, and form the rule **X ==> Y**)

- **Support_count (X):** $= \text{frequency of X in T} = \text{Number of occurences of X in T}$

- **Support (X):** $= \dfrac{\text{frequency of X in T}}{\text{size of T}}$

- **Support (X,Y) =** $= \dfrac{\text{frequency of X and Y together in T}}{\text{size of T}} = Support\left(X ==> Y\right)$

- **Confidence(X==>Y):** $= \dfrac{\text{frequency of X an Y together in T}}{\text{frequency of X}}$

- **Lift :** $= \dfrac{\text{Support(X,Y)}}{Support\left(X\right) \times Support\left(Y\right)}$

1- Association Rule Learning

Frequent itemsets and association rules

- An **itemset A** is **frequent** if :
  - Support(A) >= **minimum support threshold**

- An association rule (X ==> Y) is generated as follow:
  - Select **All** itemsets that are **frequents**
  - **Split each frequent** itemset in all possible subsets:X and Y that satisfy the condition:
    - Confidence (X ==Y) >= **minimum confidence threshold**

- In association rules **learning,** we apply **specific algorithms** on the **transactions dataset** (the training data) to identify the **frequent itemsets** in order to generate the **association rules.**

- Some of these algorithms:
  - **Apriori** (Breath First Search)
  - **FP-growth** (Frequent Pattern Growth)
  - **Eclat** (Depth First Search)

[By Amina Delali]

Naive Apriori: frequent itemsets

2- Apriori

- There is the “**naive”** approach, which describe the original **apriori** algorithm. Some improvements were introduced to this algorithm, which lead to different versions. We are going to describe the steps of the naive algorithm described in [Agrawal et al., 1994]

- The steps of frequent itemsets generation:
  - Define the L_1={frequent 1-itemset} (k-itemset = itemset with k items).
  - For (k=2, $L_{k-1} \neq \emptyset$, k++)
    - C_k = from L_{k-1} generate-all candidates k-itemsets (using only frequent itemsets)
    - For all transactions $t \in T$ (T is the set of all transactions)
      - Increment the count of all itemsets in C_k and contained in t
    - L_k = frequent itemsets in C_k (itemsets with count >= min support threshold)
  - The final frequent itemsets $is\ \cup_k L_k$

[By Amina Delali]

Naive apriori: rules generation

- The steps of rules generation are:
  - For all frequent itemsets l_k, k>=2
    - Generate all valid rules $\bar{a} \rightarrow (l_k - \bar{a})$ for each $\bar{a} \subset l_k$
      a valid rule is the one that have **confidence >= min confidence threshold**

- To generate all valid rules $\bar{a} \rightarrow (l_k - \bar{a})$ for each $\bar{a} \subset l_k$
  - 1- Set $a_m = l_k$
  - 2- A = all a_{m-1} itemsets that are subsets of a_m
  - 3- For each $a_{m-1} \in A$
    - Compute confidence of the rule r= (a_{m-1} ==> l_k- a_{m-1}) == support (l_k) /support(a_{m-1})
    - If (confidence (r) **>= min confidence threshold)** then
      - select r as a valid rule
      - If (m-1 >1) set a_m= a_{m-1} and go to 2.

Remarks

- When we generate the C_k candidates, we eliminate all the ones created by subsets that are not frequent. We call it the **prune** step.

- As an improvement, we can consider only transactions that contain frequent itemsets

- The min support and min confidence thresholds must be chosen wisely:
  - A small threshold will lead to more iterations of the algorithm
  - A high threshold can eliminate rare items.

3- Apriori illustration:
Frequent itemsets

The data

- We will run the algorithm on the example cited in [Gollapudi, 2016] (after correction)
- We suppose we have the dataset T of transactions that represent the items bought together in each purchase.

T=

1	A, B,E
2	B, D
3	B, C
4	A, B , D
5	A, D
6	B, C
7	A, D
8	A, B, C, E
9	A, B,C

- Where each letter represents an item:
  - A = iPad
  - B = iPad case
  - C = iPad scratch guard
  - D = Apple care
  - E = iPhone

- The numbers in left column represent the TID: transaction identifier

Frequent itemsets

- We suppose that the **minimum support count =2,(min shupport threshold (2/9)**

T

1	A, B,E
2	B, D
3	B, C
4	**A**, B , **D**
5	**A, D**
6	B, C
7	**A, D**
8	A, B, C, E
9	A, B,C

L_1

Itemset	Support count >= 2
A	6
B	7
C	4
D	4
E	2

C_2

Itemset	Support count
A,B	4
A,C	2
A,D	**3**
A,E	2
B,C	4
B,D	2
B,E	2
C,D	0
C,E	1
D,E	0

L_2

Itemset	Support count >= 2
A,B	4
A,C	2
A,D	3
A,E	2
B,C	4
B,D	2
B,E	2

[By Amina Delali]

Frequent items sets: prune steps

3- Apriori: illustration

L_2 → C_3 → Prune → C_3

L_2

Itemset	SC>=2
A,B	4
A,C	2
A,D	3
A,E	2
B,C	4
B,D	2
B,E	2

Not found in L_2

C_3

Itemset	Subsets
A,B,C	AB, AC, BD
A,B,D	AB, AD, BD
A,B,E	AB, AE, BE
A,C,D	AC, AD, **CD**
A,C,E	AC, AE, **CE**
A,D,E	AD, AE, **DE**
B,C,D	BC, BD, BE
B,C,E	BC, BE, **CE**
B,D,E	BD, BE, **DE**

C_3

Itemset	Support count
A,B,C	2
A,B,D	1
A,B,E	2
B,C,D	0

L_3

Itemset	SC>=2
A,B,C	2
A,B,E	2

We stop here, because if we want to prune $C_{4,}$ we will eliminate the generated subsets (ABCD, ABCE) since they contain subsets of 3 items that are not in L_3

C_4

Itemset	SC

L_4

Itemset	SC

[By Amina Delali]

4- Apriori Application

Ml-xtend The data

- **mlxtend** library implments the **apriori** algorithm.
- But, before applying the algorithm on the previous example, we have to create the corresponding data.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

# the previous data
data = [["A","B","E"],["B","D"],["B","C"],["A","B","D"],["A","D"],["B","C"],["A","D"],["A","B","C",

# encode the data, so we can apply the algorithm
TE = TransactionEncoder()
dataEnc = TE.fit(data).transform(data)
df = pd.DataFrame(dataEnc, columns=TE.columns_)
df
```

	A	B	C	D	E
0	True	True	False	False	True
1	False	True	False	True	False
2	False	True	True	False	False
3	True	True	False	True	False
4	True	False	False	True	False
5	False	True	True	False	False
6	True	False	False	True	False
7	True	True	True	False	True
8	True	True	True	False	False

[By Amina Delali]

Ml-extend frequent itemsets

4- Apriori Application

```
from mlxtend.frequent_patterns import apriori

frequent_itemsets= apriori(df, min_support=0.22, use_colnames=True)
```

	support	itemsets
0	0.666667	(A)
1	0.777778	(B)
2	0.444444	(C)
3	0.444444	(D)
4	0.222222	(E)
5	0.444444	(B, A)
6	0.222222	(C, A)
7	0.333333	(D, A)
8	0.222222	(E, A)
9	0.444444	(C, B)
10	0.222222	(B, D)
11	0.222222	(E, B)
12	0.222222	(C, B, A)
13	0.222222	(E, B, A)

The result is the union of all previous L_i we found ($L_1 \cup L_2 \cup L_3$)

[By Amina Delali]

4- Apriori Application

Mlxtend: generated rules

- we will generate the **association rules** corresponding to the found **frequent itemsets**

```
from mlxtend.frequent_patterns import association_rules

AR= association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
AR.iloc[:,[0,1,4,5,6]]
```

	antecedents	consequents	support	confidence	lift
0	(D)	(A)	0.333333	0.75	1.125000
1	(E)	(A)	0.222222	1.00	1.500000
2	(C)	(B)	0.444444	1.00	1.285714
3	(E)	(B)	0.222222	1.00	1.285714
4	(C, A)	(B)	0.222222	1.00	1.285714
5	(B, E)	(A)	0.222222	1.00	1.500000
6	(E, A)	(B)	0.222222	1.00	1.285714
7	(E)	(B, A)	0.222222	1.00	2.250000

Min confidence threshold

Correspond to the rule: $E \rightarrow B, A$

[By Amina Delali]

Concept

- It it is based on the relationship between subset inclusion and support values.
- In this algorithm, an item is represented by the list of transactions it belongs to. They are the TidLists
- The support is computed from the intersection of these TidLists.

- If N is the size of T (the transactions dataset), then:

$$support(X, Y) = \frac{|Tid(X) \cap Tid(Y)|}{N} = \frac{frequency\ \ of\ \ X\ \ \text{and}\ \ Y\ \ together}{N}$$

- It also relies on the concept that:
  - If $X \subseteq Y$, and $support(Y) = S \Rightarrow support(X) \geq s$
  - If $Y \subseteq X$, and $support(X) \leq \text{min\_s} \Rightarrow support(Y) < \text{min\_s}$

5- Eclat

[By Amina Delali]

Algorithm

5- Eclat

- The algorithm is defined by a recursive function **eclat**
- A recursive function is a function that calls itself directly or indirectly. It stops when a certain condition is met.
- The steps are:
  - set p= {}, Items ={all items}
  - Call Eclat(P,Items)
- Eclat (P,I) definition:
  - F = {}, C_{it}={}
  - If Items ={} return F
  - else
    - C = for each i in Items and not in P generate (P U i, i) tuples
    - Filter out C so it will contain only frequent P U i itemsets
    - For each (P U i, i) in c add i to C_{it}
    - For each (X, i) in C:
      - C_{it} = C_{it} - {i}
      - F = F U X U Eclat(X, C_{it})
    - Return F

Illustration

- We will run the algorithm on the example cited in [Eclat] (in the reference it is actually run for threshold=2 and not 3):
- I = {a,c,b,e,d,f}, min_support_count_threshold= 3
- T = [[a,b,c],[a,c,d,e,f], [a,b,c],[d,e]]
- P ={} , Items= {a,b,c,d,e}
- Eclat(P= {}, Items={a,b,c,d,e}) --------------(1)
  - Eclat(P= {}, Items={a,b,c,d,e}) (from 1)
    - F= {}; C_{it}={}
    - C= {(a,a),(b,b),(c,c),(d,d),(e,e),(f,f)}
    - C = {(a,a),(c,c)}, C_{it} ={a,c}
      - 1) X=a, i= a
      - C_{it} ={c}
      - F = {a} U Eclat(P={a}, Items={c}) ------- (11)
      - Eclat(P={a}, Items= {c}) (from 11)
        - F= {}, C_{it}={}
        - C = {(ac,c)}

5- Eclat

[By Amina Delali]

Illustration (suite)

- $C = \{(ac,c)\}$, $C_{it} = \{c\}$
  - X={ac}, i= c
  - Items={}
    - F = {ac } U Eclat (P={ac}, Items={})---- (111)
    - Eclat (P={ac}, Items={}) (from 111)

      - F= {}, $C_{it}=\{\}$

      - Items =={}, return F (return to 111)
  - F = {ac}
- Return F (return to ---- (11)

- F ={a, ac}
- 2) X={c}, i = c
- $C_{it}=\{a\}$
- F = {a,ac,c} U Eclat(P={c}, Items={a}) ------- (12)
  - Eclat(P={c}, Items= {a}) (from 1)

AIM

5- Eclat

[By Amina Delali]

5- Eclat

Illustration (suite)

- Eclat(P={c}, Items= a}) (from 12)
  - F= {}, $C_{it=}${}
  - C = {(ca,a)}
  - C_{it}= {a}
    - X= {ca} ,i=a
    - C_{it}= {}
    - F = {ca } U Eclat (P={ca}, Items={})---- (121)
    - Eclat (P={ca}, Items={}) (from 121)
      - F= {}, C_{it}={}
      - Items =={}, return F (return to 121)
    - F = {ca}
  - Return F (return to ---- (12))
- F ={a,ac,c}

➔ Return F (return to (1))

- Final F = {a, ac, c}

[By Amina Delali]

Library and data

- fim is a library comprised of a module that implements a set of functions dedicated to frequent itemset mining.

- The functions are related to the algorithm they implement.
- For example, the library implements “Eclat”, “apriori”, et “fpgrowth” functions.

- To have a list of all the the represented algorithms, take a look at its homepage (PyFIM - Frequent Item Set Mining for Python).
- To install the library, just use the **pip** command:

```
!pip install fim
```

- Concerning the data, we do not need to do any transformation. So, we will use the transactions of the example, as we defined them (list of lists):

```
T = [["a","b","c"],["a","c","d","e","f"], ["a","b","c"],["d","e"]]
```

Eclat application

6-fim library

```
import fim as fim
from fim import eclat
fis = eclat(T, supp=75)
```

```
[(('c',), 3), (('a', 'c'), 3), (('a',), 3)]
```

- Supp =75 means Support = 0.75 (¾)
- By default it prints support_count values

"S" to print the support as fractions

```
fis_p = eclat(T, supp = 75,report = "s")
fis_p
```

```
[(('c',), 0.75), (('a', 'c'), 0.75), (('a',), 0.75)]
```

The same frequent itemsets we found earlier in the illustration

[By Amina Delali]

Apriori application

- We will use the “apriori” function on the previous example we saw in apriori section.

```python
from fim import apriori
fis_a = apriori(data, supp=22,report="s")
```

Support_count = 2 is equivalent to support = 2/9

```
[(('E', 'A', 'B'), 0.2222222222222222),
 (('E', 'A'), 0.2222222222222222),
 (('E', 'B'), 0.2222222222222222),
 (('E',), 0.2222222222222222),
 (('D',), 0.4444444444444444),
 (('D', 'A'), 0.3333333333333333),
 (('D', 'B'), 0.2222222222222222),
 (('C', 'B'), 0.4444444444444444),
 (('C',), 0.4444444444444444),
 (('C', 'A', 'B'), 0.2222222222222222),
 (('C', 'A'), 0.2222222222222222),
 (('A',), 0.6666666666666666),
 (('A', 'B'), 0.4444444444444444),
 (('B',), 0.7777777777777778)]
```

References

- [Agrawal et al., 1994] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB, volume 1215, pages 487–499.
- [Alexey, 2014] Alexey, G. (2014). Eclat. On-line at http://mlwiki.org/index.php/Eclat. Accessed on 30-12-2018.
- [Gollapudi, 2016] Gollapudi, S. (2016). Practical Machine Learning. Community experience distilled. Packt Publishing.
- [Sebastian,] Sebastian, R. mlxtend's documentation. On-line at http://rasbt.github.io/mlxtend/. Accessed on 30-12-2018.

Thank you!

FOR ALL YOUR TIME