COMP7630 – Web Intelligence and its Applications

Association Rules

Valentino Santucci

(valentino.santucci@unistrapg.it)

Mining Association Rules

- Mining association rules allows to discover regularities in data
- Objective: to find all co-occurrence relationships among data items
- Classic application: market basket data analysis, which aims to discover how items purchased by customers in a supermarket (or a store) are associated

Example Association Rule

$$\text{Cheese} \rightarrow \text{Beer} \qquad [\text{support} = 10\%, \text{confidence} = 80\%].$$

- (Support) 10% of the customers buy Cheese and Beer together
- (Confidence) Customers who buy Cheese also buy Beer 80% of the times

- Support and Confidence are two measures of rule strength

Possible applications in the web

- Purchases patterns in e-commerce websites
- Word co-occurrence relationships
- Hashtag suggestion in social networks
- Web usage patterns
- …

Definitions

- $I = \{i_1, i_2, \dots, i_m\}$ is the universe set of items
- $T = \{t_1, t_2, \dots, t_n\}$ is the set (or database) of transactions
- $t_i \subseteq I$, i.e., each transaction is a subset of items

- An association rule is an implication of the form

$$X \rightarrow Y, \text{ where: } X \subset I, Y \subset I, X \cap Y = \emptyset$$

- X and Y are called itemsets

An example

Example 1: We want to analyze how the items sold in a supermarket are related to one another. I is the set of all items sold in the supermarket. A transaction is simply a set of items purchased in a basket by a customer. For example, a transaction may be:

{Beef, Chicken, Cheese},

which means that a customer purchased three items in a basket, Beef, Chicken, and Cheese. An association rule may be:

Beef, Chicken $\rightarrow$ Cheese,

where {Beef, Chicken} is X and {Cheese} is Y. For simplicity, brackets "{" and "}" are usually omitted in transactions and rules. ■

Some other definitions

- The transaction $t_i \in T$ contains the itemset $X \subseteq I$ iff $X \subseteq t_i$
- In that case, we also say that X covers t_i

- The support count of X in T is the number of transactions in T that contain X. The support count is denoted by $X.count$

Support and Confidence of a rule

- Given an association rule $X \rightarrow Y$, we define its support and confidence:

$$support = \frac{(X \cup Y).count}{n}.$$

$$confidence = \frac{(X \cup Y).count}{X.count}.$$

Relationships with probabilities

- Support is the percentage of transactions in T that contains $X \cup Y$, i.e., that contains both X and Y.
- Support is an estimate of $P(X \cup Y)$ or, better, it estimates $P(X \text{ and } Y) = P(X, Y)$

- Confidence is the percentage of transactions in T containing X that also contain Y
- Confidence is an estimate of $P(Y|X)$

- RECALL conditional probability definition:

$$P(Y|X) = \frac{P(X,Y)}{P(X)} \Rightarrow P(X,Y) = P(Y|X)P(X) = P(X|Y)P(Y)$$

Why Support and Confidence?

- Support is a useful measure because if it is too low, the rule may just occur due to chance. Furthermore, in a business environment, a rule covering too few cases (or transactions) may not be useful because it does not make business sense to act on such a rule (not profitable).

- Confidence determines the predictability of the rule. If the confidence of a rule is too low, one cannot reliably infer or predict Y from X. A rule with low predictability is of limited use.

Mining of Association Rules

Given a transaction set *T*, the problem of mining association rules is to discover all association rules in *T* that have support and confidence greater than or equal to the user-specified minimum support (denoted by **minsup**) and minimum confidence (denoted by **minconf**).

Example

INPUT

t_1:	Beef, Chicken, Milk
t_2:	Beef, Cheese
t_3:	Cheese, Boots
t_4:	Beef, Chicken, Cheese
t_5:	Beef, Chicken, Clothes, Cheese, Milk
t_6:	Chicken, Clothes, Milk
t_7:	Chicken, Milk, Clothes

minsup = 30%
minconf = 80%

The following association rules are valid:

Expected OUTPUT

Rule 1:	Chicken, Clothes $\rightarrow$ Milk	[sup = 3/7, conf = 3/3]
Rule 2:	Clothes, Milk $\rightarrow$ Chicken	[sup = 3/7, conf = 3/3]
Rule 3:	Clothes $\rightarrow$ Milk, Chicken	[sup = 3/7, conf = 3/3].

Apriori Algorithm

- Apriori is one of the best known algorithm for mining association rules

- The Apriori algorithm has been proposed in

Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.

Apriori algorithm: how it works

- Apriori works in two steps:

1. **Generate all frequent itemsets**: A frequent itemset is an itemset that has transaction support above minsup

2. **Generate all confident association rules from the frequent itemsets**: A confident association rule is a rule with confidence above minconf

- Note: in practical applications, sometimes only the first step is enough to reach the objective at hand

Apriori step 1: generate all frequent itemsets

Downward Closure Property: If an itemset has minimum support, then every non-empty subset of this itemset also has minimum support.

Algorithm Apriori(T)

$C_1 \leftarrow$ init-pass(T); // the first pass over T
$F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq minsup\}$; // n is the no. of transactions in T
for ($k = 2$; $F_{k-1} \neq \emptyset$; k++) **do** // subsequent passes over T
    $C_k \leftarrow$ candidate-gen(F_{k-1});
    for each transaction $t \in T$ **do** // scan the data once
        for each candidate $c \in C_k$ **do**
            if c is contained in t **then**
                $c.count$++;
        endfor
    endfor
    $F_k \leftarrow \{c \in C_k \mid c.count/n \geq minsup\}$
endfor
return $F \leftarrow \bigcup_k F_k$;

Function candidate-gen(F_{k-1})

$C_k \leftarrow \emptyset$; // initialize the set of candidates
forall $f_1, f_2 \in F_{k-1}$ // find all pairs of frequent itemsets
    with $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$ // that differ only in the last item
    and $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\}$
    and $i_{k-1} < i'_{k-1}$ **do** // according to the lexicographic order
        $c \leftarrow \{i_1, \ldots, i_{k-1}, i'_{k-1}\}$; // join the two itemsets f_1 and f_2
        $C_k \leftarrow C_k \cup \{c\}$; // add the new itemset c to the candidates
        for each $(k-1)$-subset s of c **do**
            if ($s \notin F_{k-1}$) **then**
                delete c from C_k; // delete c from the candidates
        endfor
endfor
return C_k; // return the generated candidates

Note: Not necessary to load the whole data into memory before processing, only one transaction must reside in memory.

Apriori step 1 ... an "animated" example

- We have the following dataset of transactions

TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

and we assume minsup=40%, i.e., at least 2 transactions need to be covered, so in the following I simplify support to be the number of transactions covered by a given itemset

Apriori step 1 ... an "animated" example

- Find candidate itemsets of length 1

TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

C1

Itemset	Support
{1}	3
{2}	3
{3}	4
{4}	1
{5}	4

Apriori step 1 … an "animated" example

- Filter-out candidates with support<minsup and obtain F1 (frequent itemsets of length 1)

C1

Itemset	Support
{1}	3
{2}	3
{3}	4
{4}	1
{5}	4

F1

Itemset	Support
{1}	3
{2}	3
{3}	4
{5}	4

Apriori step 1 … an "animated" example

- Create candidate of length 2 and filter-out low-support itemsets

TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

Only Items present in F1

C2

Itemset	Support
{1,2}	1
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3

F2

Itemset	Support
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3

Apriori step 1 … an "animated" example

- Create candidates of length 3

TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

C3

Itemset	In F2?
{1,2,3}, {1,2}, {1,3}, {2,3}	NO
{1,2,5}, {1,2}, {1,5}, {2,5}	NO
{1,3,5},{1,5}, {1,3}, {3,5}	YES
{2,3,5}, {2,3}, {2,5}, {3,5}	YES

Apriori step 1 … an "animated" example

- Filter-out C3 to generate frequent itemsets of length 3

TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

F3

Itemset	Support
{1,3,5}	2
{2,3,5}	2

Apriori step 1 … an "animated" example

- Create candidates of length 4

TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

F3

Itemset	Support
{1,3,5}	2
{2,3,5}	2

C4

Itemset	Support
{1,2,3,5}	1

Since all candidates in C4 are filtered-out, the algorithm terminates and its result is the union of F1, F2, and F3.

Apriori step 2: generate association rules

- Simple strategy:

  for every frequent itemset f and for each subset $\alpha \subset f$:

  output the rule $(f - \alpha) \rightarrow \alpha,\ \text{if}$

$$confidence = \frac{f.count}{(f - \alpha).count} \geq minconf,$$

where $f.count$ (or $(f - \alpha).count$) is the support count of f (or $(f - \alpha)$), which can be easily obtained from the supports computed in step 1.

Apriori step 2... a more efficient strategy

- In order to design an efficient strategy, let's observe that the support count of f does not change as α changes
- Therefore, if $(f - \alpha) \rightarrow \alpha$ is valid, then all the rules of the form $(f - \alpha_{sub}) \rightarrow \alpha_{sub}$, with $\alpha_{sub} \subset \alpha$, are valid

- Example: if $A, B \rightarrow C, D$ is valid, then also $A, B, C \rightarrow D$ and $A, B, D \rightarrow C$ are valid

- Therefore, an efficient algorithm very similar to "candidate-gen" (seen some slides before) can be devised

Lift: another measure for association rules

- $Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Y.count/n}$

- Lift is an estimate of $\frac{P(X,Y)}{P(Y)P(X)}$
- Therefore *lift* is a kind of "correlation" between X and Y

- Informally speaking, a high *lift* indicates that the importance of an association rule is not just a coincidence

One more example

INPUT

```
*** Dataset of transactions ***
[['A', 'B', 'C', 'D'],
 ['A', 'B'],
 ['A', 'D', 'E'],
 ['C', 'D', 'E'],
 ['B', 'D', 'E']]
********************************
```

minsup = 0.5
minconf = 0.8

maxlen = 2

One more example

INPUT

```
*** Dataset of transactions ***
[['A', 'B', 'C', 'D'],
 ['A', 'B'],
 ['A', 'D', 'E'],
 ['C', 'D', 'E'],
 ['B', 'D', 'E']]
*******************************
```

minsup = 0.5
minconf = 0.8

maxlen = 2

Support Computation

Support(A) = 3/5 = 0.6
Support(B) = 3/5 = 0.6
Support(C) = 2/5 = 0.4
Support(D) = 4/5 = 0.8
Support(E) = 3/5 = 0.6

Support(A,B) = 2/5 = 0.4
Support(A,D) = 2/5 = 0.4
Support(A,E) = 1/5 = 0.2
Support(B,D) = 2/5 = 0.4
Support(B,E) = 1/5 = 0.2
Support(D,E) = 3/5 = 0.6

One more example

INPUT

```
*** Dataset of transactions ***
[['A', 'B', 'C', 'D'],
 ['A', 'B'],
 ['A', 'D', 'E'],
 ['C', 'D', 'E'],
 ['B', 'D', 'E']]
********************************
```

minsup = 0.5
minconf = 0.8

maxlen = 2

Support Computation

Support(A) = 3/5 = 0.6
Support(B) = 3/5 = 0.6
Support(C) = 2/5 = 0.4
Support(D) = 4/5 = 0.8
Support(E) = 3/5 = 0.6

Support(A,B) = 2/5 = 0.4
Support(A,D) = 2/5 = 0.4
Support(A,E) = 1/5 = 0.2
Support(B,D) = 2/5 = 0.4
Support(B,E) = 1/5 = 0.2
Support(D,E) = 3/5 = 0.6

Confidence Computation

Confidence(D->E) = 0.6 / 0.8 = 0.75
Confidence(E->D) = 0.6 / 0.6 = 1

One more example

INPUT

```
*** Dataset of transactions ***
[['A', 'B', 'C', 'D'],
 ['A', 'B'],
 ['A', 'D', 'E'],
 ['C', 'D', 'E'],
 ['B', 'D', 'E']]
*******************************
```

minsup = 0.5
minconf = 0.8

maxlen = 2

Support Computation

Support(A) = 3/5 = 0.6
Support(B) = 3/5 = 0.6
Support(C) = 2/5 = 0.4
Support(D) = 4/5 = 0.8
Support(E) = 3/5 = 0.6

Support(A,B) = 2/5 = 0.4
Support(A,D) = 2/5 = 0.4
Support(A,E) = 1/5 = 0.2
Support(B,D) = 2/5 = 0.4
Support(B,E) = 1/5 = 0.2
Support(D,E) = 3/5 = 0.6

Confidence Computation

Confidence(D->E) = 0.6 / 0.8 = 0.75
Confidence(E->D) = 0.6 / 0.6 = 1

Lift Computation

Lift(E->D) = 1 / 0.8 = 1.25

Practical applications other than market basket

- Any text documents can be seen as a transaction, where each distinct word/lemma is an item (duplicate words are removed)
- Same as before, but consider windows of words of a given maximum length

- Relational tables with categorical values are easily seen as transactions

- If numerical values are present, we need to discretize them to categories
  - Example:
    - $Temperature \leq 0°$ => "very cold"
    - $0° < Temperature \leq 15°$ => "cold"
    - $15° < Temperature \leq 25°$ => "warm"
    - $Temperature > 25°$ => "hot"

Some more (complex) extensions

- Rare items problem: distinctive items may be interesting at different minimum supports.
- Class labels: transactions may be labeled by classes and users may be interested in targeted rules, i.e., rules whose right-hand-side is a class label.
- Sequential patterns: association rules do not consider the order of transactions, so sequences of items (and not simply itemsets) can be considered.

- All these cases have been formally defined and specific mining algorithms (which are extensions of the Apriori algorithm) have been proposed

Hands-on with Python

- Requirements:
  - Python (>=3.8, but it may be ok also an older version)
  - `pip install mlxtend`

- Two examples in `arules.zip`:
  - `basket.py` (mining of rules for a simple market basket)
  - `recipes.py` (mining of rules for a dataset of recipes… where recipes are lists formed by cuisine_type + list of ingredients)

References

- *Liu, Bing. Web data mining: exploring hyperlinks, contents, and usage data. Berlin: springer, 2011. Chapter 2.*

- *Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.* https://courses.cs.duke.edu/compsci516/spring16/Papers/AssociationRuleMining.pdf

- Apriori Algorithm – Know how to find frequent itemset. https://medium.com/edureka/apriori-algorithm-d7cc648d4f1e