CS 6604: Data Mining Fall 2007

Lecture 16 — October 15, 2007

Lecture: Naren Ramakrishnan *Scribe: Xiaomo Liu*

1 Overview

In the previous several lectures, we mainly discussed algorithms for ILP, i.e., supervised learning of relational predicates. In this class, we focus on unsupervised algorithms for learning relational patterns. In particular, we look at algorithms that are the relational equivalent of traditional enumerative search algorithms.

Take, for instance, the most famous algorithm in the association rules literature, i.e., **Apriori**. Its relational twin, call it **Relational Apriori**, is its generalization to first-order logic. To recall the details of Apriori, one of the properties it exploits is anti-monotonicity, namely if a set of items X does not have support, a set $Y \supseteq X$ cannot have support.

2 Relational Apriori

2.1 Queries

Suppose we have a relational database consisting of three tables, i.e., ”Customer”, ”Parent”, and ”Buys” as follows:

Customer_ID
allen
bill
carol
diana

Parent_ID	Child_ID
allen	bill
allen	carol
bill	zoe
carol	diana

Customer_ID	Item
allen	wine
bill	candy
bill	pizza
diana	pizza

Table 1: Relation database RD with customer information

The type of pattern we try to mine can be viewed as queries. We say that a query-pattern matches the database if the set of tuples returned by the query is not empty. Take an ordinary SQL query for example, if we need to inquire the relational database for the customer who buys its child ”candy”:

```
SELECT Customer.Customer_ID, Parent.Child_ID
FROM   Customer, Parent, Buys
WHERE  Customer.Customer_ID = Parent.Parent_ID
AND    Parent.Child_ID = Buys.Customer_ID
AND    Buys.Item = 'candy'
```

Let us convert this query into predicate logic form. Thus, the above SQL query is translated into the following logical clause form, Q_1.

```
Customer(X), Parent(X,Y), Buys(Y,candy)
```

We say query-pattern Q_1 matches the relational database RD because there exists an answer $\{X = allen, Y = bill\}$.

2.2 "Relational Itemsets"

Queries can be view as "relational itemsets". Compare example query Q_1 to itemset:

```
{Customer, Parent, Child, Candy_buyer}.
```

In the single table database framework, such an itemset has a 'market-basket' interpretation so that in a single transaction, a customer, a parent, a child, and a candy buyer occur. In the relational case, we need to define the notion of a 'transaction'. Consider again query Q_1 and assume we observe that 25% of our customers are parents of children who buy candy (Allen).

```
Customer(X), Parent(X,Y), Buys(Y, candy)
Frequency: 0.25
```

At the same time, we can also count the occurrence for another query Q_2 and observe that 75% of our customers are parents (Allen, Bill and Carol).

```
Customer(X), Parent(X,Y)
Frequency: 0.75
```

The frequency calculation, i.e. occurrence counting, for queries in the relational database framework is slightly different from that for itemsets in a single table. The details of the query occurrence counting method is described by the **Algorithm 2: WARMR-EVAL** in [1].

2.3 Relational Association Rules

In the original single table database frame, we take two itemsets, corresponding to queries Q_1 and Q_2:

```
{Customer, Parent, Child, Candy_buyer}
Frequency: 0.25
```

and

```
{Customer, Parent, Child}
Frequency: 0.75
```

and construct an association rule

```
{Customer, Parent, Child} -> {Candy_buyer}
Confidence: 0.25/0.75=0.33
```

Now let us consider a similar rule generation in the relational framework. One approach is to construct:

```
    Customer(X), Parent(X,Y) -> Buy(Y, candy).
```

However, the problem is that, if we interpret this rule, the left hand side is read as 'some customers are parents' and the right hand side is interpreted as 'someone buys candy' which holds for all the customers. So this rule is obviously not what we are looking for. The following rule is correct, read as 'if a customer has a child, then that customer also has a child that buys candy'.

```
    Customer(X), Parent(X,Y) -> Customer, Parent(X,Y), Buy(Y, candy).
```

If using a more strict predicate logic representation, the above rule can be written as

$$\exists x \exists y, Customer(x) \wedge Parent(x,y) \longrightarrow Customer(x) \wedge Parent(x,y) \wedge Buy(y, candy).$$

As mentioned in the previous section 2.1.2, query Q_1 has the frequency 0.25 and query Q_2 has the frequency 0.75 and the confidence of this rule is 0.33.

3 Relational Rule Mining

In the single table database framework, the mining algorithms such as Apriori etc. use a level-wise approach to search in the lattice spanned by a specialization relation $\preceq$ between patterns, where $p_1 \preceq p_2$ denotes 'pattern p_1 is more general than pattern p_2' or 'p_2 is more specific than p_1'. Pruning is based on anti-monotonicity of $\preceq$ with respect to frequency: if a pattern is not frequent then none of its specializations are frequent [1].

Recall that in the *candidate generation* phase of Apriori, it prunes the infrequent patterns, or itemsets, based on the property of anti-monotonicity on $\subset$, namely if a itemset X (e.g.$\{a\}$) is infrequent then its superset Y (e.g.$\{a, b, c\}$) is also infrequent. Here $\preceq$ is equivalent to $\subset$ because set X is more general than its superset Y.

In the case of the relational data mining framework, we again need to perform a level-wise search from the bottom of the lattice for queries. For instance, the level-1 queries consist of one literal such as $a(x)$ and $d(z)$. From these we construct level-2 queries such as $a(x)b(x)$ and $b(x)c(y)$. In the *candidate generation* phase of 'Relational-Apriori', we also need to determine a specialization relation $\preceq$ for queries so as to perform the pruning process. With such as specialization relation, if we can find a query Q at level-i is infrequent then all the queries $Q^{'}$ at level-j ($i < j$) are also infrequent.

However, the subset relation from the traditional Apriori framework is not suitable for queries. For example, consider the query:

```
    Likes(Z,Z), Buys(Z, candy)
```

which is more specific than the query:

```
    Likes(X,Y), Likes(Y,X)
```

although the first query is not a subset of the second. Fortunately, the notion of θ-subsumption satisfies this purpose. However, the original definition of θ-subsumption introduced in the previous lectures need to be flipped in the context of 'Relational-Apriori'. The reason for this flip is that the query is rather a clause with the conjunction of all literals, whereas in the previous definition a clause is a disconjunction of all literals. Therefore, a query Q_1 θ-subsumes (abbreviated to: ts) another query Q_2 if and only if there exists a substitution of the variables of Q_2 such that $Q_1 \supseteq Q_2\theta$ (recall that this was: $Q_1\theta \subseteq Q_2$ in the original definition). Take the above example:

```
    Likes(Z,Z), Buys(Z, candy) ts Likes(X,Y), Likes(Y,X)
```

with $\theta = \{X/Z, Y/Z\}$, because

$$Likes(Z,Z) \wedge Buys(Z, candy) \supseteq Likes(Z,Z) \wedge Likes(Z,Z).$$

In this case, $Likes(Z,Z), Buys(Z, candy)$ is more specific than $Likes(X,Y), Likes(Y,X)$. If we can prove that the query $Likes(X,Y), Likes(Y,X)$ is infrequent, then we can determine that the query $Likes(Z,Z)$, $Buys(Z, candy)$ must also be infrequent and, thus, we can prune it. The details of the pruning process and candidate generation is described by **Algorithm 3: WARMR-GEN** in [1].

4 Summary

As a summary of the 'relational' version of Apriori, such a rule mining algorithm for queries is similar to the ordinary Apriori algorithm except for three modifications:

- Determine how to count the occurrence of a query in the relational database.
- Cannot cancel out the literals for rule generation as the ordinary association rule.
- Flip the definition of θ-subsumption in order to perform the pruning.

References

[1] L. Dehaspe, H. Toivonen, *Discovery of Relational Association Rules*, In N. Lavrac and S. Dzeroski, editors, *Relational Data Mining*, pages 189-212, Springer-Verlag, 2001.