Data Mining: Sequential Patterns

Nathan Rountree

15 Aug 2001 (7 August 2000)

1 Introduction

This summary is based on material which may be found in Agrawal and Srikant (1995). An expanded version is available as Agrawal and Srikant (1994).

Imagine you have a database of customer transactions, each one a market basket of items. Each transaction is identifiable by both customer ID and transaction ID, where transaction ID is temporally ordered (i.e. you can tell when a transaction occurred by its ID).

If you wanted to express patterns about which items are bought together (in one transaction), that would be *association mining.* Associations are therefore an *intra-transaction* model on which to base patterns. If on the other hand you were interested in what *item(set)s* are bought *after having bought previous item(set)s*, that would be **sequence** mining, and an **inter**-transaction model.

A sequential pattern may look something like this:

> *Customers who buy "The Fellowship of the Ring" tend to buy* **both** *"The Two Towers" and "The Return of the King" in a subsequent transaction.*

which seems nice and simple. However there are some important things to remember:

1. If possible, we want groups of items (itemsets) as each member in a pattern.
2. We want sequences to be as long as possible.
3. Sequences don't have to be contiguous in the database ([A then B] may be implied from [A then X then B]), but we still have to recognise them.
4. Sequences should meet some kind of minimum support criterion.

2 Terminology

itemset: a group of items co-occurring in a single transaction. An itemset is said to have **support** of $x\%$ if $x\%$ of the customers have a transaction with the itemset in it. Note that this is different to the notion of support in association mining, where it is calculated as the percentage of transactions in which it occurs.

sequence: a list of itemsets which belong to one customer in chronological order.

containment: a sequence a is **contained** in another sequence b if for each itemset i_a in a there is an itemset i_b in b which is a superset of i_a, AND the order of the itemsets is the same. For example:

$\langle (ab)(c)(def) \rangle$ is contained in $\langle (abc)(d)(cd)(def) \rangle$
$\langle (ab)(c)(def) \rangle$ is not contained in $\langle (abc)(def) \rangle$
$\langle (abc) \rangle$ is not contained in $\langle (a)(b)(c) \rangle$ and vice versa.

maximal: a sequence is maximal if it is not contained within any other sequence.

customer sequence: if we order the database first on customer ID then on transaction ID, we can view all the transactions of each customer as a sequence. This is a *customer-sequence.*

support: A customer supports a sequence if it is contained within that customer's customer sequence. The support for a sequence is therefore the proportion of customers who support a sequence.

3 Finding Sequential Patterns

Let's define a minimum support for a sequence as *minsupp*. There are 5 phases to finding sequences with support higher than *minsupp*:

1. Sort the database by customer ID and transaction time.
2. Find all the itemsets with support higher than *minsupp* using a modified *a priori* algorithm. We call these **large** itemsets.
3. Transform the database by replacing each transaction with a token representing which large itemsets are present in the transaction. For instance, if $\{a\}$, $\{b\}$ and $\{ab\}$ are all large with tokens 1, 2 and 3 then we would replace the transaction { a b c d e } with tokens { 1 2 3 }. (The d and e items are discarded, since they are not large. Tokens 2 and 1 are NOT subsumed by 3: make sure you understand why not.)
4. Use the transformed database to find the sequences.
5. Remove from the sequences any which are not maximal.

4 Example

customer	time	items
c1	1	a b
c1	2	c
c1	3	b c d
c2	1	a c
c2	2	b c
c2	3	d
c3	1	b d
c3	2	b c
c3	3	c

Let's set *minsupp* at 0.66%

Step 1: is already done.

Step 2: the large itemsets are a, b, c, d, bc and bd which we shall tokenise as 1, 2, 3, 4, 5, 6.

Step 3: gives us the new table:

customer	time	large items
c1	1	1 2
c1	2	3
c1	3	2 3 4 5 6
c2	1	1 3
c2	2	2 3 5
c2	3	4
c3	1	2 4 6
c3	2	2 3 5
c3	3	3

Step 4: The tricky part. Initial candidate generation is the large itemsets:

```
1 2 3 4 5 6
```

Using a similar idea to *a priori*'s candidate generation we get

```
1 1    2 1    3 1    4 1    5 1    6 1
1 2    2 2    3 2    4 2    5 2    6 2
1 3    2 3    3 3    4 3    5 3    6 3
1 4    2 4    3 4    4 4    5 4    6 4
1 5    2 5    3 5    4 5    5 5    6 5
1 6    2 6    3 6    4 6    5 6    6 6
```

of which only

```
1 2    2 2    3 2
1 3    2 3    3 3
1 4    2 4    3 4
1 5    2 5    3 5
```

actually appear as sequences in the database. Once more we do an extension; this time we can avoid any subsequences that aren't in the set above:

```
1 2 2    2 2 2    3 2 2
1 3 2    2 3 2    3 3 2
1 2 3    2 2 3    3 2 3
1 3 3    2 3 3    3 3 3
1 2 4    2 2 4    3 2 4
1 3 4    2 3 4    3 3 4
1 2 5    2 2 5    3 2 5
1 3 5    2 3 5    3 3 5
```

Now check those in the database to find that only

```
1 3 4    2 3 3
```

are actually there. No point in generating 4-sequences, since each customer only has 3 transactions listed.

Step 5: Get rid of sequences which are not maximal, giving us:

```
1 2*   2 2*   3 2*   1 3 4
1 5    2 4    3 5    2 3 3
       2 5
```

since 1 4, 1 3, 2 3, 3 3, and 3 4 are all contained within the two 3-sequences. Also the starred sequences can be cut, because they are contained within 1 5, 2 5, or 3 5 (think carefully about why).

That's it—translate back to a, b, c, d for the actual sequences.

References

R. Agrawal and R. Srikant. Mining sequential patterns. Technical Report RJ9910, IBM, October 1994.

R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the International Conference on Data Engineering*, Taipei, Taiwan, march 1995.