CS 6604: Data Mining Fall 2007

Lecture 2 — Wednesday, August 22, 2007

Lecture: Naren Ramakrishnan Scribe: Clifford Owens

1 Searching for Sets

The canonical data mining problem is to search for frequent subsets from a set of sets (sometimes called a "structured set"). The typical instantiation is known as market based analysis where we are given a set (database) of sets (transactions) of items. An example is:

transaction ID	items
t1	{a,b}
t2	{a,c}
t3	{b,c,a}

Databases are typically stored in one of two formats. A horizontal representation, as shown above, is indexed by transaction ID and stores items in each transaction. A vertical representation, on the other hand, is indexed by item and stores transactions that each item occurs in. Different algorithms are tuned to work with different representations. In this lecture, we will focus on horizontal representations.

Given I, the set of all items in the database, the frequency (or *support*) of $s \subseteq I$ is given by the number of transactions that contain s, as a fraction of the total number of transactions. The frequent itemset mining problem is to find all subsets of I that have support greater than or equal to some specified threshold θ.

It is folklore in data mining that in mining consumer transactions from supermarkets the itemset {beer,diapers} was observed to be frequent. Given this observation, a marketer would like to know whether it is more likely that beer purchasers also buy diapers (represented by 'beer' $\rightarrow$ 'diapers'), or the other way round (i.e., 'diapers' $\rightarrow$ 'beer'), or both. The *confidence* captures these distinctions, where the confidence of rule $s \rightarrow t$, $s, t \subseteq I$ is given by $\frac{\text{support}(s \cup t)}{\text{support}(s)}$. Observe that the confidence of 'beer' $\rightarrow$ 'diapers' and the confidence of 'diapers' $\rightarrow$ 'beer' utilize the same numerator but have different denominators. Similarly, from an itemset of size k, we can investigate $2^k - 2$ possible rules.

The association rule mining problem is to first find frequent itemsets that satisfy a given support threshold and then, for each itemset, identify confident rules. We will skip the second problem since the first one is more intensive (why?) and because there is a version of the first problem hiding inside the second (think about it).

2 Algorithms for Searching Set Spaces

A trivial algorithm would work as follows. It scans one transaction at a time. For each transaction, it increments the count of every subset occurring in that transaction. For a transaction that has k items, it will increment the count of $2^k - 1$ subsets. It is easy to see that this algorithm requires only one pass through the database. However it is very wasteful since most subsets will not be frequent but will anyways be counted.

The AIS (Agrawal-Imielinski-Swami) algorithm, presented in [1] and named after its authors, is credited with establishing the modern day focus of data mining from secondary storage. It makes multiple passes through the database. In each pass it generates and evaluates 'candidates.' In the first pass, all singleton itemsets, such as $\{a\}$, $\{b\}$, etc., are considered candidates. For each transaction read, for every item in that transaction, the count of the corresponding singleton itemset is incremented. At the end of the pass, those singleton itemsets that do not have sufficient support are discarded. In the second pass, it constructs itemsets of size 2 (by using the results of the first pass) and counts their support. The exact way it does this is as follows: for each transaction read, it determines if one of the frequent singleton itemsets are present in it. If so, that singleton itemset is 'extended' with every other item present in the transaction. These become the candidates for this pass. For instance, if $\{a\}$ is found frequent in the first pass, and we encounter a transaction containing $\{a, b, d\}$ in the second pass, then we create two candidates: $\{a, b\}$ and $\{a, d\}$. Similarly, at pass 3, we create and evaluate candidates of size 3 by extending those itemsets found frequent in pass 2 by one more item. In general, at pass k, we extend frequent itemsets of size $k-1$ by one more item.

It doesn't seem to be widely known that the actual algorithm presented in [1] presents a variety of ways to 'configure' this search. First, in any given pass, it can be made more aggressive by extending itemsets with not just one more item but more items. The reason for doing this is that even if the extended-by-one itemset is frequent, yet another pass is required to ensure that *its* extension(s) are not frequent. In other words, we need to 'step across the border' to know that we have crossed the border. The approach in [1] is to first conduct a probabilistic modeling of the frequency of the individual items, i.e., the singleton itemsets. Using these, given a frequent itemset, we can determine which of its extensions are *expected* to be frequent. To allow for stepping across the border, the AIS algorithm identifies those extensions that are expected to be frequent and also those that are expected to be infrequent. All of them are evaluated in the current pass.

The second configurable in the AIS algorithm is how to determine those itemsets that are to be extended in a given pass (these are called 'frontier' itemsets in [1]). So far we have implicitly assumed that we will attempt to extend only those that are found frequent in the previous pass. This might be wasteful (why?). Another option is to extend only those that are 'maximal', i.e., that are not contained in another itemset that has also been determined to be frequent. The paper [1] shows that this can lead to missing some itemsets. The correct approach is, besides the maximal itemsets, to incldue those itemsets that were expected to be infrequent but turned out to be frequent.

We have slyly introduced three important lessons in data mining. First, the support criterion for itemsets is *anti-monotonic*, i.e., if an itemset does not have support, then none of its supersets can have support. This is also known as a downward closure property (since all subsets of frequent set are frequent). The second lesson derives from the first. Due to the downward closure, there is a border separating frequent patterns from infrequent patterns and hence algorithms that can quickly find the border are preferred. Finally, effective data mining is often a matter of efficiency in counting. So, the more itemsets that can be discarded without counting, the better the algorithm. Data mining algorithms are heavily tuned in this regard. Even for those itemsets that have to be counted, we can ensure that we do not encounter the same extension twice by ordering the items in some default (e.g., lexicographic) order.

The next algorithm we will study is the *Apriori* algorithm, due to Srikant and Agrawal [2]. Unlike AIS, it doesn't try to jump multiple levels in a given pass; it proceeds strictly one level in a given pass. But it is slightly more clever. Whereas the AIS algorithm extends itemsets by looking at which other items occur in a given transaction, the *Apriori* algorithm determines the candidates for a given pass before looking at even a single transaction (for that pass). Given frequent itemsets at level k, F_k, candidates at level $k+1$ are obtained by picking two sets from F_k that have the first $k-1$ elements the same and unioning them (keep in mind that we will assume sets are lexicographically ordered). For example, if we have $\{n, q, r, t\}$,

$\{n, q, s, x\}$, and $\{n, q, s, z\}$ as frequent in the fourth pass, we will create $\{n, q, s, x, z\}$ as a candidate in the fifth pass. But there will be no candidates containing $\{n, q, r\}$.

This is sufficient to create candidates, as no valid candidates will be overlooked. However, it is not very strict, as there may be candidates unecessarily generated. This is because we are using only two sets from F_k to create a candidate at level $k + 1$, whereas any itemset that is frequent at level $k + 1$ must have $k + 1$ subsets that are frequent in level F_k. This is again something that can be verified before actually counting (i.e., before undertaking another pass through the database). To use the previous example, given $\{n, q, s, x, z\}$, we can verify that $\{n, q, x, z\}$, $\{n, s, x, z\}$, and $\{q, s, x, z\}$ all are frequent.

Notice that just picking two sets from F_k that have $k - 1$ elements in common is not as aggressive as picking two sets from F_k that have the first $k - 1$ elements the same. This idea was used in [3].

Unlike a traditional algorithms course, we will assess complexity of data mining algorithms in terms of the number of patterns that are generated. This is often referred to as output-sensitivity in the algorithms literature. Ideally, for an algorithm that generates M patterns, we would like the complexity to be $O(M)$. What is the complexity of the above algorithms?

This underscores the importance of setting good parameters for data mining, such as the support threshold. Too small a threshold will produce too many patterns. Too large a threshold might not result in any patterns.

3 Classes of patterns

One way to avoid finding too many patterns is to define a new pattern class that is smaller than the set of all frequent patterns. Consider the database:

1	ACTW
2	CDW
3	ACTW
4	ACDW
5	ACDTW
6	CDT

Assuming a support threshold of 0.5, the following are frequent:

Level 1: A C D T W

Level 2: AC AT AW CD CT CW DW TW

Level 3: ACT ACW ATW CDW CTW

Level 3: ACTW

These frequent itemsets can be arranged in a *lattice* as shown in Fig. 1. Recall that a lattice is a poset where every pair of elements has a greatest lower bound, and every pair of elements has a lowest upper bound.

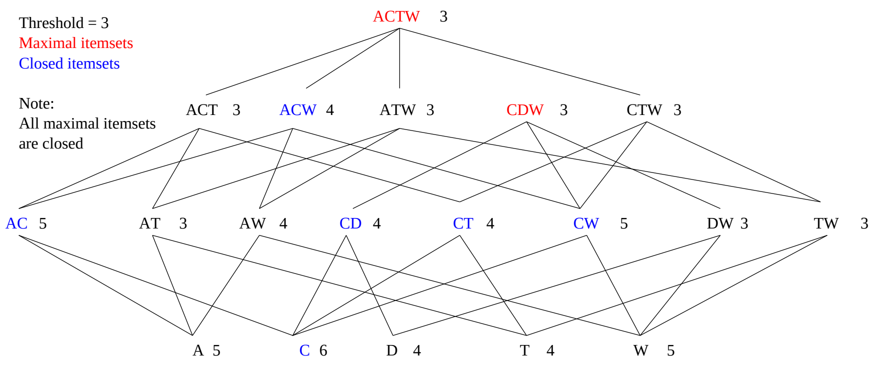

Figure 1: Lattice of frequent itemsets. The support threshold requires that every itemset be present in at least 3 transactions.

Instead of mining all frequent itemsets, we can try to find only *maximal* ones, i.e., those frequent itemsets that are not contained in any other frequent itemsets. In Fig. 1, CDW and ACTW are maximal. Many times it is good just to find the maximal frequent patterns. While they don't tell us everything, given the maximal frequent itemsets, we know that all supersets thereof are not frequent, and all subsets thereof are frequent. Hence the maximal frequent itemsets identify the border. The number of maximal frequent itemsets can be an order of magnitude smaller than the number of frequent itemsets.

However, the maximal frequent itemsets are not a lossless representation. What we mean by this is that, given just the maximal frequent itemsets and their frequencies (supports), we cannot determine the frequencies of any of the other frequent sets. Another pattern class, called the *closed frequent sets*, serves this purpose.

A closed frequent itemset is a frequent itemset that has no superset with exactly the same support, i.e., that is present in exactly the same set of transactions.

¿From the above example, we see that the sets A and AC occur in:

A: 1, 3, 4, 5
AC: 1, 3, 4, 5

Hence AC is closed, but A is not. We can store only AC and its support. Fig. 2 depict the relationship between frequent itemsets, maximal frequent itemsets, and closed frequent itemsets. The closed frequent itemsets can be an order of magnitude smaller than frequent itemsets and the maximal frequent itemsets can be an order of magnitude smaller than the closed ones as well.

4 Food for thought

Think about algorithms for mining rules. We have only investigate algorithms for mining frequent itemsets.

Why should we search bottom-up? Can we search top-down? What is the over-riding consideration when deciding whether to search bottom-up or top-down? Checkout [4] that tries to search from both directions.

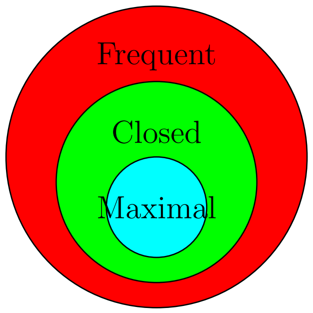

Figure 2: Classes of itemsets

References

[1] Rakesh Agrawal, Tomasz Imielinski, Arun Swami, Mining Association Rules between Sets of Items in Large Databases, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pages 207-216, 1993.

[2] Rakesh Agrawal, Ramakrishnan Srikant, Fast Algorithms for Mining Association Rules in Large Databases, Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94), pages 487-499, 1994.

[3] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient Algorithms for Discovering Association Rules, in Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, July 1994.

[4] Dao-I Lin, Zvi M. Kedem. Pincer Search: A New Algorithm for Discovering the Maximum Frequent Set, in Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, 1998, pages 105-119, 1998.