CS 6604: Data Mining Fall 2007

Lecture 8 — Wednesday, September 12, 2007

Lecture: Naren Ramakrishnan *Scribe: Don Conry*

1 Overview

It is convenient to recall a few terms from previous lectures [1]:

1. A *monotonic* constraint C_M is one that preserves order, i.e. if $x \subseteq y$ and $C_M(x)$ is satisfied, then $C_M(y)$ is also satisfied. It follows that monotone constraints are *upward-closed*, meaning that if any set A satisfies the constraint, all supersets of A will also satisfy that constraint.

2. An *anti-monotonic* constraint C_{AM} is one that reverses order, i.e. if $x \subseteq y$ and $C_{AM}(x)$ is not satisfied, then $C_{AM}(y)$ is also not satisfied. Anti-monotone constraints are *downward-closed*, meaning that if any set B fails to satisfy the constraint, all supersets of B will also fail to satisfy that constraint.

Either of these properties induces a border. Either can be used with a top-down or bottom-up search, or both (like Pincer Search [2]).

1.1 An example

Consider the following database of item prices and transactions:

transaction ID	items
1	{b, c, d, g}
2	{a, b, d, e}
3	{b, c, d, g, h}
4	{a, e, g}
5	{c, d, f, g}
6	{a, b, c, d, e}
7	{a, b, d, f, g, h}
8	{b, c, d}
9	{b, e, f, g}

item	price
a	5
b	8
c	14
d	30
e	20
f	15
g	6
h	12

Suppose we want to mine itemsets from the above database with 2 constraints:

1. Itemsets must have support of 4 or greater (minsupp(i) $>= 4$).

2. Itemsets must have total price of 45 or greater (sum(i.price) $>= 45$).

Notice that the first constraint is anti-monotonic (downward-closed), while the second constraint is monotonic (upward-closed). How can we mine sets meeting the above constraints, without doing too much work? In particular, we want to avoid unnecessary database scans.

To begin, observe that the two constaints differ from each other in more ways than monotonicity. Given an itemset, the second constraint can be verified without scanning the database; it only requires knowing the prices of items. This is not true for the first constraint. This idea is used in [4] to push the constraint deeper into the candidate generation and pruning process. Here is how it works.

Step 1: Count singleton items and save their support values in an index.

item	support
a	4
b	7
c	5
d	7
e	4
f	3
g	6
h	2

Step 2: Next eliminate singleton items that fail to satisfy the support constraint. In this case, f and h can be eliminated. This is called μ reduction in [4].

Step 3: Claim: items a and e can also be eliminated. Why? Notice that transaction 6 $\{a, e, g\}$ has a total cost of 14, which is less than 45. Therefore, since we know $\{a, e, g\}$ alone cannot satisfy the second constraint, we can remove transaction 6 from consideration, reducing the support values for the included items by 1. In this case, the support for items a and e drops to 3 which is below the threshold, so a and e can also be removed due to lack of support. This is known as α reduction in [4].

Step 4: Remove items eliminated in steps 2 & 3 from *all* transactions, and recalculate the total cost of each transaction. If necessary we can return to step 3 and eliminate other transactions which no longer meet the total cost constraint.

In general, steps 3 & 4 can be iterated until only transactions satisfying both constraints remain. This method works because both α- and μ-reduction can prune items or transactions from the database *without losing any solutions*. A formal proof of this method can be found in [4].

2 Variance

Another type of constraint is of the following form: find all itemsets such that the variance of the item cost is at least p. This variance constraint is neither monotonic nor anti-monotonic. However, as proved in [5], variance is *loosely* anti-monotonic (and also loosely monotonic). Loosely anti-monotonic means that if an itemset X satisfies a constraint, there exists at least one itemset $Y \subset X$ with $|Y| = |X| - 1$ (i.e. exactly one less item in size) that also satisfies the constraint; this process can be repeated to find a suitable subset of Y satisfying the constraint, and so forth down the lattice (until level 1... single item sets have a variance of 0). Loosely monotonic sets have a similar property in the reverse direction (going up the itemset lattice).

In the case of variance, appropriate subsets and supersets are easy to find at each level; to go down the lattice (loosely anti-monotonic), we simply drop the item that is the closest to a set's mean value. To go up the lattice, we select the superset with an additional item that is furthest from the set's mean.

For a given itemset with constraints $Var(x.price) <= p$ and $supp(x) >= \theta$, we can adapt the algorithm from the previous section to mine transactions that meet the constraints. Scan the database once to count support for singleton items (variance is 0 for singletons). After eliminating singletons that fail to satisfy the minimum support constraint, then combine the survivors. At each subsequent level, due to the loosely anti-monotonic property of variance, if *none* of the subsets of an itemset meets *both* constraints, we can eliminate that itemset from further consideration.

This concludes our study of itemsets.

3 Classification

Given a target variable and one or more features, predicting the value of the target variable based on these features is called *classification*. One example of a classification problem is filtering unwanted spam messages from email. Given some rules including specific features and a known target variable, spam filters predict the target variable for future instances:

email	feature 1	feature 2	feature 3	spam?
#1	y	n	y	spam
#2	y	y	n	not spam
#3	n	y	y	spam
...	...	...	...	...
#k	y	n	n	?

One method of prediction is by using association rules. Consider instances as transactions, features and the target variable as itemsets in each transaction, and mine the database for frequent itemsets, but only those itemsets involving the target variable. Next, obtain rules from these itemsets whose consequent is the target variable and rank them by *confidence*. One problem with this approach is that contradictory rules may occur. For example, if you have rules that feature b predicts $spam$ and feature c predicts not spam, how can you classify an instance containing both features?

This type of method where lots of rules simultaneously predict classes in possibly overlapping ways is called a *simultaneous covering*. A *covering* of a set S is a collection C of subsets of S such that $\bigcup C = S$. In other words, the union of all sets in C contain every element in S at least once. A covering is *simultaneous* if the rules are not required to be ordered. A *sequential covering* contains rules that must be applied in a certain order. The rules of a sequential covering are similar to the if-then-else construct in many programming languages, also known as a decision list (see figure). A *partition* of S is a minimal covering, i.e. it is both collectively exhaustive (covers all elements of S) and mutually exclusive (contains no repeated elements).

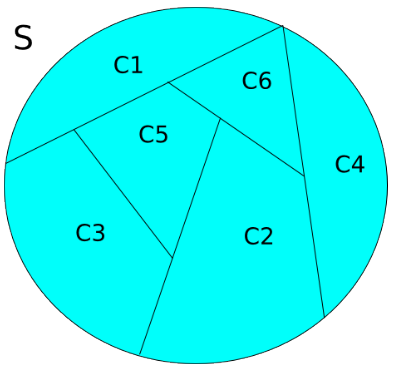

Figure 1: The sets C1, ..., C6 form a *covering* as well as a *partition* of S.

In reality, most coverings consist of a majority of points inside the target set S, but also some points hanging outside. The problem of adjusting such sets so they perfectly cover S is called the Red Blue Set Cover problem. This problem, and even finding approximations of its solution, are intractable.

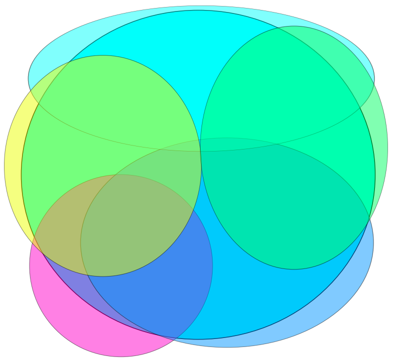

Figure 2: This covering is exhaustive but not mutually exclusive. It also 'spills' over to cover unnecessary elements.

4 Next lexture

Next class we will discuss how to learn decision trees (and lists). One way to grow these kinds of trees is to ask informative questions. Next time we will define what we mean by 'informative'.

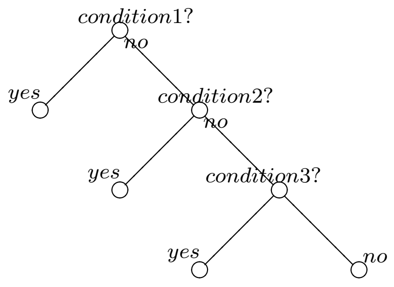

Figure 3: A simple decision list. Every node has a leaf for (at least) one of its children.

References

[1] A. Ceglar, J.F. Roddick. *Association Mining*, in ACM Computing Surveys, volume 38, number 2, article number 5, July 2006.

[2] 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, pages 105-119, 1998.

[3] Jian Pei, Jiawei Han. *Can we push more Constraints into Frequent Pattern Mining?*, in KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 350-354, 2000.

[4] F. Bonchi, F. Giannotti, A. Mazzanti, D. Pedreschi. *ExAnte: Anticipated Data Reduction in Constrained Pattern Mining*, in Proceedings of PKDD'03, pages 59-70, 2003.

[5] F. Bonchi, C. Lucchese, and R. Trasarti, *Pushing Tougher Constraints in Frequent Pattern Mining*, in Proceedings of PAKDD'05, pages 114-124, 2005.