Topics in Algorithmic Data Analysis
Assignment 2: Grabbing It

Muhammad Yaseen - 2577833
s8muyase@stud.uni-saarland.de

1 Introduction

In this report we consider three pattern mining algorithms: KRIMP [1], SLIM [2], and GRAB [3]. All three algorithms were proposed in the context of solving the pattern explosion problem in Data Mining using the Minimum Description Length (MDL) principle. Both KRIMP [1] and SLIM [2] can mine representative itemsets (in terms of MDL) whereas GRAB [2] can additionally mine rule sets of the form X and includes patterns as a special case: $X = \emptyset$.

In the following sections we compare these methods and look at their similarities and differences. We use the same notation as that in the respective papers and as such do not explicitly describe it.

2 Conceptual differences and similarities

All methods consider new itemsets to be added to the model from a universe $\mathcal{F}$ of sets. For KRIMP $\mathcal{F}$ is the set of pre-mined candidates according to some criteria, usually *minsup* whereas in SLIM $\mathcal{F}$ is the collection of all itemsets in database $\mathcal{D}$. SLIM uses pairwise union of itemsets present in CT to identify the next best candidate. This results in iterative refining the of code table in a systematic manner as redundant codes are replaced with succinct combinations. Thus SLIM considers the whole database while KRIMP is restricted to a given set. Compared to these two, n GRAB candidate set is constructed via condensing rules together and thus is more similar to SLIM in this respect and it is also not limited to a pre-mined set. In all three algorithms the models always contains the singletons.

KRIMP and SLIM also share many similarities between them. Both use the same model i.e. a code table CT with itemsets in the left column and their code in the right column and start from a singleton code table called Standard Code Table (ST) and also use it to encode the final itemsets X_i belonging to the optimal code table CT via optimal prefix codes for singletons. In either of them a longer frequent itemset is considered first to cover a transaction. This makes intuitive sense as it would result in covering more data and result shorter codes. A major perspective shift from KRIMP to SLIM is that it introduces a cover space formed by replacing transactions with their cover in which it searches for

frequent itemsets and treats pairwise union of them as candidates for next CT. In so doing it treats each set X_i as a kind of "extended"-singleton.

Finally, in both KRIMP and SLIM the problem is formalized as finding Minimal Coding Set represented by CT whereas in GRAB the we look for a directed acyclic graph (DAG) of Minimal Rule Sets.

3 Algorithmic similarities and differences

KRIMP and SLIM both use the same coding method i.e. optimal prefix codes based on usage or support of singleton itemsets to encode the items in ST, which in turn is used to encode CT. Thus any difference in performance of the two is due to the search strategy used, candidates used to cover the data, and pruning.

Two principle limitations of KRIMP are that (1) expects a candidate set already available e.g. generated from some frequent itemset mining algorithm and then works to iteratively filter the redundant rules to identify a minimal set of non-redundant rules that describe the data best (2) it considers the cover only in a fixed order. However, in large databases specially at small support level the number of patterns generated it exponential and thus it is not feasible to even generate and test that many candidates and thus the method is not practical in that setting. Similarly, and the code table obtained might not be the best approximation to optimal result. SLIM solves these problem by directly mining patterns from the database (2) removing an itemset from CT if it no longer contributes to compression.

An important algorithmic difference from KRIMP which makes a huge difference in practice is that as both SLIM and GRAB explore a much larger search space compared to KRIMP, they use a two-level gain estimates for efficient search i.e. the first estimate is easier to computer but in-exact while the second is exact but computationally costly to compute. Doing this avoids materialization and costly gain evaluation on every potential candidate at each step and reserve it only for candidates that show the best gain in terms of the cheaper first estimate.

4 Encoding and overlapping patterns

Encoding process of KRIMP and SLIM is rather straightforward. But to illustrate the encoding method of GRAB we use the following binary database in context of the rule $\{a, b\} \rightarrow \{c, e\}$.

tid	a	b	c	d	e	f
1	1	1	0	1	0	0
2	1	1	0	0	0	1
3	1	1	1	1	0	0
4	0	1	0	1	1	0
5	1	1	1	0	0	0
6	0	1	1	0	1	0
7	0	0	1	1	1	1
8	1	1	1	0	0	1
9	1	1	1	0	1	0
10	0	0	0	1	1	1

It can be seen that the rule applies to: $T_{\{a,b\}} = \{1, 2, 3, 5, 8, 9\}$

The rule holds strictly on $T^{Mstrict}_{\{c,e\}|\{a,b\}} = \{8, 9\}$

Or if we allow tolerance of 1 missing item ($k = 1$): $T^{Mtol}_{\{c,e\}|\{a,b\}} = \{3, 5, 8, 9\}$

The complement set in sctrict case is $T^{Mstrict}_{not\{c,e\}|\{a,b\}} = \{1, 2, 3, 5\}$

Or in noise tolerant case: $T^{Mtol}_{not\{c,e\}|\{a,b\}} = \{1, 2\}$

The final part which needs to be encoded as data is given as $D_{Y|X} = \Pi_{\{c,e\}}(T_{\{a,b\}})$:

tid	c	e
1	0	0
2	0	0
3	1	0
5	1	0
8	1	1
9	1	1

Considering these sets we can see that given any T_X where the rule applies it can hold in $\binom{|T_X|}{|T^M_{Y|X}|}$ ways, similar to a subset selection problem. Each of this possibility can be enumerated in a pre-determined canonical way and thus an index over it could be defined so each possibility corresponds to a natural number. Clearly, to transmit the index for a specific rule we need $log\binom{|T_X|}{|T^M_{Y|X}|}$ bits. This explains the first term in data encoding for GRAB.

The second and third terms can be understood as encoding a binary sub-matrix corresponding to the partition of $\Pi_{\{c,e\}}(T_{\{a,b\}})$ where the rule is said to hold (resp. not hold). In our case this would be the sub-matrix corresponding to tids 8,9 in strict case and 3,5,8,9 in the noise tolerant case. Given that we know the row and columns the whole sub-matrix can be represented as a single number which is an index over the enumeration of all possible matrices, ways in which rule could hold (resp. not hold). This representation also

preserves the number of items of Y actually present (resp. not present) when the rule is said to hold (resp. not hold).

Another interpretation of second and third part is that it measures the degree to which the consequent holds (resp. doesn't hold) in $T_{Y|X}$ (resp. $T_{\overline{Y}|X}$). As such both parts vanish when the partitioning is perfect and there is no partial overlap with consequent in T_X. In our case this would happen if tids 3 and 5 did not exist.

Thus, we see that overlapping patterns in a set of transactions can be captured by a single consequent. This is different from SLIM or KRIMP, which would require additional disjoint itemsets to cover each possible overlap.

5 Noise and Error matrices

Of the three methods we consider only GRAB accounts for noise explicitly. This noise is intended as a robustness measure against small or spurious violations of a given rule. Thus we can say that a rule $A \to BCD$ applies to transaction $t = ABC$ even if $D \notin t$. In GRAB this robustness incurs an additional cost as we have to also then send for each rule and transaction which elements of Y are actually present in t. Thus the coding method used in GRAB incurs least cost when we use exact condition for rule holding and in each transaction to which the rule applies it holds exactly i.e. either whole of Y is present in t or no element of Y is in t. In the cases of partial fulfilment/violation we have to incur the extra cost.

Noise in context of GRAB is defined as the maximum number of items k in the consequent of a rule allowed to be missing in a transaction t while still saying that the rule holds. Interestingly, noise is not dealt with on a global level (e.g. assumed to be Gaussian or Uniform) but rather it is inferred from the set of transactions T_X where the rule $X \to Y$ applies. Doing so allows different level of tolerance for every rule.

Modelling noise in this way can be beneficial in cases where occasional violations are expected in an otherwise fixed process e.g. in a market transaction database caused by an out-of-stock item (not buying toilet paper because there aren't any), promotions or special occasions like Christmas, New Year etc. It can also be used to account for missing values given that the number is small or to account for cases where the violation of rule occurs only within a group e.g. a person buying {apple,banana,cola} or {apple,banana,pepsi}. In both cases the third items represents a beverage and a specific beverage may not be important but knowing that a person buys one is.

References

[1] J. Vreeken, M. van Leeuwen, and A Siebes. "Krimp: Mining Itemsets that Compress". In: *Data Mining and Knowledge Discovery* (2011).

[2] K. Smets and J Vreeken. "Slim: Directly Mining Descriptive Patterns". In: *In Proceedings of the 12th SIAM International Conference on Data Mining.* (2012), pp. 236–247.

[3] J. Fischer and J. Vreeken. “Sets of Robust Rules, and How to Find Them”. In: *In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Data* (2019).