MDL in Pattern Mining A Brief Introduction to Krimp

Arno Siebes

Algorithmic Data Analysis Group
Universiteit Utrecht, The Netherlands
arno@cs.uu.nl

Abstract. In this short paper we sketch a brief introduction to our Krimp algorithm. Moreover, we briefly discuss some of the large body of follow up research. Pointers to the relevant papers are provided in the bibliography.

1 Patterns

Arguably *patterns* are the most important contribution of the data mining community to data analysis. On way to define patterns is as *partial* models, i.e., they do not necessarily concern all tuples (a.k.a. objects, individuals, ...) nor do they necessarily comprise all attributes (a.k.a. variables, features, ...). Another way is to identify patterns with subsets of the data that are for some reason deemed interesting.

The prototypical pattern mining problem is undoubtedly frequent item set mining [1], which is usual formulated in the context of transaction data. Each such transaction is a set of items, e.g., the items a customer buys at a supermarket and a transaction database is simply a bag of transactions.

The patterns are also sets of items called item sets. An item set I occurs in a transaction t if $I \subseteq t$. The support of an item set I in a database db, denoted by $supp_{db}(I)$ is the number of transactions in db in which I occurs. The frequent item set problem is then to find *all* item sets whose support exceeds some user defined threshold. Because of the apriori property

$$I \subseteq J \Rightarrow supp_{db}(I) \geq supp_{db}(J)$$

all frequent item sets can be found relatively efficiently using level-wise search; relatively because there may be exponentially many frequent item sets.

In [10] the authors generalize frequent item set mining into a much wider class of pattern mining problems known as *theory mining*. To keep our discussion simple, we stay in the realm of item set mining, but anything we do can be generalized to this wider context.

Frequent item set mining also illustrates rather well why pattern mining is important. The set of customers of a supermarket is hardly homogeneous. While the famous[1] frequent item set example {beer, diapers} may be applicable to

[1] Famous but highly likely an urban legend.

young couples it is probably not relevant to fifty-something couples. While the discerning palate implicated by the item set {Condrieu, Saint Marcellin} carves up the customer space in a completely different – and independent – way.

2 The Pattern Explosion

As our example item sets above show patterns can provide useful insight in the data, but, there is an problem. If the support threshold is set high, a few patterns will be discovered, but mostly patterns that are already well known to domain experts. If the support threshold is set low, however, the number of patterns discovered explodes. It is not uncommon to discover more patterns than one has transactions in the database!

Given that one of the main goals of pattern mining is to provide insight in the data, this explosion of the number of patterns is rather embarrassing. So it is not surprising that this problem received much attention and that a wide variety of more or less successful solutions have been put forward.

One of the earliest and best-known is actually closely related to Formal Concept Analysis, viz., *closed item set* mining [11]. Closed item sets are those item sets for which each superset has a strictly smaller support. In other words, they are the closure of the obvious Galois connection between transactions and item sets.

While the number of closed frequent item sets is clearly smaller that the number of frequent item sets, there is no guarantee that their number is far smaller – and often it isn't. The collection of closed frequent item sets has the property that all frequent item sets can be derived from it. That is, the collection of closed frequent item sets is a *condensed representation* of the set of all frequent item sets.

This observation gave rise to other condensed representations, such as *free* item sets [4] and non-derivable item sets [5]; the latter being the smallest condensed representation. However, all of these suffer from the fact that they may still yield very large result sets.

The popular alternative approach is through *constraints* [8]. Clearly, using filters a user can – in principle – easily search through the large set of frequent patterns to find the truly interesting ones. One of the main goals of constraint based pattern mining is to generate only interesting patterns, e.g., by pushing the constraints into the discovery process. While it is relatively easy to remove definitely uninteresting patterns using filters and/or constraints it turns out that it is hard to define – and thus mine for – the small group of interesting ones only.

3 What Is the Problem?

The reason why it is so hard to delineate the interesting patterns is that with low(er) support, many patterns describe essentially the same subset of the database. In more detail,

- a database has many small subsets,
- many of these subsets can be described by patterns
- many of the small subsets described by patterns differ in at most a few objects.

Let both A and $A \cup \{a\}$ be closed item sets. If their difference in support is only 1, it is hard to imagine how both can be interesting to a user.

In other words, to decide whether or not a pattern is interesting we also have to look at other patterns as well as the subset of the data that is described by these patterns. That is, we should not be looking for interesting patterns, but for interesting *sets of patterns* and to make such a set of patterns interesting, its members should not describe the same subset of the data over and over again. To formalize this, we use MDL.

4 MDL for Pattern Sets

The MDL principle [7] can be paraphrased as: *Induction by Compression.* Slightly more formal, it can be described as follows. Given a set of models $\mathcal{H}$, the best model $H \in \mathcal{H}$ for data set D is the one that minimises

$$L(H) + L(D|H)$$

in which

- $L(H)$ is the length, in bits, of the description of H
- $L(D|H)$ is the length, in bits, of the description of the data when encoded with H.

In the remainder of this section we briefly describe how we employ MDL to find small characteristic sets of patterns; see, e.g., [14,22] for more detail.

The key idea of our compression based approach is the code table. A code table has item sets on the left-hand side and a code for each item set on its right-hand side. The item sets in the code table are ordered descending on 1) item set length and 2) support. The actual codes on the right-hand side are of no importance: their lengths are. To explain how these lengths are computed we first have to introduce the coding algorithm. A transaction t is encoded by Krimp by searching for the first item set c in the code table for which $c \subseteq t$. The code for c becomes part of the encoding of t. If $t \setminus c \neq \emptyset$, the algorithm continues to encode $t \setminus c$. Since we insist that each code table contains at least all singleton item sets, this algorithm gives a unique encoding to each (possible) transaction. The set of item sets used to encode a transaction is called its cover. Note that the coding algorithm implies that a cover consists of non-overlapping item sets. The length of the code of an item in a code table CT depends on the database we want to compress; the more often a code is used, the shorter it should be. To compute this code length, we encode each transaction in the database db. The frequency of an item set $c \in CT$ is the number of transactions $t \in db$ which have c in their cover. The relative frequency of $c \in CT$ is the probability that c is used to encode an arbitrary $t \in db$. For optimal compression of db, the higher $\mathrm{P}(c)$, the shorter its code should be. In fact, from information theory [6], we have the Shannon code length for c, which is optimal, as:

$$l_{CT}(c) = -\log(P(c|db)) = -\log\left(\frac{freq(c)}{\sum_{d \in CT} freq(d)}\right)$$

The length of the encoding of a transaction is now simply the sum of the code lengths of the item sets in its cover. Therefore the encoded size of a transaction $t \in db$ compressed using a specified code table CT is calculated as follows:

$$L_{CT}(t) = \sum_{c \in cover(t,CT)} l_{CT}(c)$$

The size of the encoded database is the sum of the sizes of the encoded transactions, but can also be computed from the frequencies of each of the elements in the code table:

$$L_{CT}(db) = \sum_{t \in db} L_{CT}(t) = -\sum_{c \in CT} freq(c) \log\left(\frac{freq(c)}{\sum_{d \in CT} freq(d)}\right)$$

To find the optimal code table using MDL, we need to take into account both the compressed database size as described above as well as the size of the code table. For the size of the code table, we only count those item sets that have a non-zero frequency. The size of the right-hand side column is obvious; it is simply the sum of all the different code lengths. For the size of the left-hand side column, note that the simplest valid code table consists only of the singleton item sets. This is the *standard encoding (st)* which we use to compute the size of the item sets in the left-hand side column. Hence, the size of the code table is given by:

$$L(CT) = \sum_{c \in CT : freq(c) \neq 0} l_{st}(c) + l_{CT}(c)$$

In [14] we defined the optimal set of (frequent) item sets as that one whose associated code table minimises the total compressed size:

$$L(CT) + L_{CT}(db)$$

Krimp starts with a valid code table (only the collection of singletons) and a sorted list of candidates. These candidates are assumed to be sorted descending on 1) support and 2) item set length. Each candidate item set is considered by inserting it at the right position in CT and calculating the new total compressed size. A candidate is only kept in the code table iff the resulting total size is smaller than it was before adding the candidate. If it is kept, we reconsider all other elements of CT to see if they still contribute to compression. If not, they are permanently removed. The whole process is illustrated in Figure 1.

In [14,22] it is shown that this simple heuristic algorithm reduces the number of frequent item sets dramatically – e.g. by retaining only 1 in a million frequent patterns.

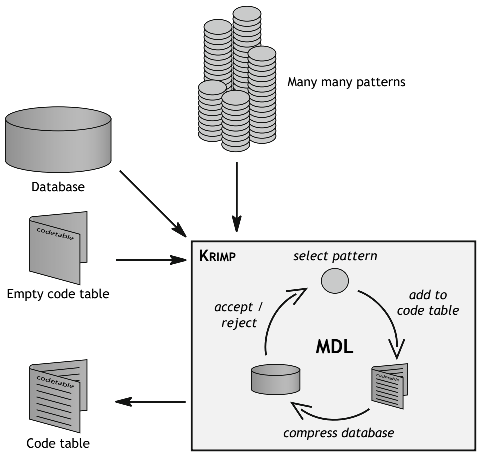

Fig. 1. KRIMP in action

Clearly, there are far simpler approaches that can reduce the number of frequent item sets, e.g., by randomly deleting almost all item sets. Hence, in [22,18] we show that KRIMP returns a *characteristic* set of patterns. This is illustrated by using code tables for classification. The idea is simply that we learn a code table for each class separately and assign a new transaction to the class whose code table compresses the transaction best.

5 There Is More

KRIMP is just one algorithm to compute code tables, it is rather wasteful as it first generates all frequent item sets and only then starts looking for the code table. SLIM [16] is much faster because it generates candidate item sets on the fly; by taking the current code table into account large parts of the search space can be ignored.

KRIMP (and SLIM) searches for one optimal model. GROEI [12] aims to find a collection of good models, each optimal for a given level of complexity.

We specified KRIMP in the context of transaction data. Because categorical data sets are easily transformed to transaction data, it is obvious that KRIMP also works for that type of data. The basic ideas underlying KRIMP have also been applied to different types of data, e.g., relational data [9], streaming data [17], data from evolutionary biology [2], and recently to seismographic data [3]

Next to classification, code tables can also be used for other kinds of data mining tasks, such as clustering [19], change detection [17], and outlier detection [15]. Moreover, they are also useful in traditionally more statistical tasks such as data generation [21] – which allows for a strong form of privacy protection – data imputation [20], and data smoothing [13].

And there is more, much more than I can describe here in this brief introduction; both by the originators of the KRIMP algorithm and, more importantly, by others. The papers referenced here will give the reader at least a start in the fascinating new area at the intersection of data mining and algorithmic information theory.

Acknowledgements. The author is supported by the Dutch national COMMIT project.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Inkeri Verkamo, A.: Fast discovery of association rules. In: Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.) Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI/MIT Press (1996)
2. Bathoorn, R., Siebes, A.: Constructing (almost) phylogenetic trees from developmental sequences data. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 500–502. Springer, Heidelberg (2004)
3. Bertens, R., Siebes, A.: Characterising seismic data. In: ICDM 2014 Proceedings. IEEE (2014)
4. Boulicaut, J.-F., Bykowski, A., Rigotti, C.: Free-sets: a condensed representation of boolean data for the approximation of frequency queries. Data Mining and Knowledge Discovery 7(1), 5–22 (2003)
5. Calders, T., Goethals, B.: Non-derivable itemset mining. Data Mining and Knowledge Discovery 14(1), 171–206 (2007)
6. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley- Interscience, New York (2006)
7. Grünwald, P.: The Minimum Description Length Principle. MIT Press (2007)
8. Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.): Constraint-Based Mining and Inductive Databases. LNCS (LNAI), vol. 3848. Springer, Heidelberg (2005)
9. Koopman, A., Siebes, A.: Characteristic relational patterns. In: KDD 2009 Proceedings, pp. 437–446 (2009)
10. Mannila, H., Toivonen, H., Inkeri Verkamo, A.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery 1(3), 241–258 (1997)
11. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 398–416. Springer, Heidelberg (1999)
12. Siebes, A., Kersten, R.: A structure function for transaction data. In: SDM 2011 Proceedings, pp. 558–569. SIAM (2011)
13. Siebes, A., Kersten, R.: Smoothing categorical data. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012, Part I. LNCS, vol. 7523, pp. 42–57. Springer, Heidelberg (2012)
14. Siebes, A., Vreeken, J., van Leeuwen, M.: Item sets that compress. In: SDM 2006 Proceedings, pp. 393–404. SIAM (2006)
15. Smets, K., Vreeken, J.: The odd one out: Identifying and characterising anomalies. In: SDM 2011Proceedings, pp. 804–815 (2011)

16. Smets, K., Vreeken, J.: Slim: Directly mining descriptive patterns. In: SDM 2012 Proceedings, pp. 236–247 (2012)
17. van Leeuwen, M., Siebes, A.: Streamkrimp: Detecting change in data streams. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part I. LNCS (LNAI), vol. 5211, pp. 672–687. Springer, Heidelberg (2008)
18. van Leeuwen, M., Vreeken, J., Siebes, A.: Compression picks item sets that matter. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 585–592. Springer, Heidelberg (2006)
19. van Leeuwen, M., Vreeken, J., Siebes, A.: Identifying the components. Data Mining and Knowledge Discovery 19(2), 173–292 (2009)
20. Vreeken, J., Siebes, A.: Filling in the blanks: Krimp minimisation for missing data. In: ICDM 2008 Proceedings, pp. 1067–1072. IEEE (2008)
21. Vreeken, J., van Leeuwen, M., Siebes, A.: Preserving privacy through data generation. In: ICDM 2007 Proceedings, pp. 685–690. IEEE (2007)
22. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: Mining itemsets that compress. Data Mining and Knowledge Discovery 23(1), 169–214 (2011)