Compression picks Item Sets that Matter

Matthijs van Leeuwen, Jilles Vreeken, Arno Siebes

Department of Computer Science
Universiteit Utrecht
{mleeuwen, jillesv, arno}@cs.uu.nl

Abstract. Finding a comprehensive set of patterns that truly captures the characteristics of a database is a complicated matter. Frequent item set mining attempts this, but low support levels often result in exorbitant amounts of item sets. Recently we showed that by using MDL we are able to select a small number of item sets that compress the data well [14]. Here we show that this small set is a good approximation of the underlying data distribution. Using the small set in a MDL-based classifier leads to performance on par with well-known rule-induction and association-rule based methods. Advantages are that no parameters need to be set manually and only very few item sets are used. The classification scores indicate that selecting item sets through compression is an elegant way of mining interesting patterns that can subsequently find use in many applications.

Keywords: frequent item sets, MDL, classification.

1 Introduction

Ever since the first paper on association rule mining [1], mining for frequent item sets has been a popular topic, as it has many useful applications. By now there are many algorithms that discover the frequent item sets efficiently [1,6].

Another problem, however, is far from solved: the explosion of the number of results. High minimum support (*min-sup*) thresholds yield only well-known results, while low thresholds yield very large numbers of frequent item sets; often more than there are transactions in the original database. Over the years, many solutions have been proposed [8,10], e.g., closed [18] and maximal [2] item sets. Most, if not all of these methods can be understood as a compression of the result set, some methods are lossless (closed item sets) while others are lossy (maximal item sets).

Recently, we proposed a radically different solution to this problem [14]. A set of item sets is interesting iff *it yields a good (lossless) compression of the database* rather than a good compression of the set of all frequent item sets.

To determine whether or not a subset of the set of frequent item sets yields a good compression of the database, we used the *Minimum Description Length Principle* (MDL) [7]. The MDL principle gave us a fair way to balance the size of the compressed database and the size of the code table; see Section 2 for more details.

As shown in [14], heuristic algorithms yield sets of frequent item sets that are easily four orders of magnitude smaller than the complete set of frequent item sets and

give high compression ratios. Clearly, the MDL principle indicates that these small sets characterise the database. But, how characteristic are they? That is, do these small sets differentiate between different databases? In this paper we investigate this problem using *classification*.

The small set of item sets (well, actually the code table they come from) compresses the original database well. Of course, this compression scheme can be used not only for transactions in the database but for all transactions over the underlying set of items. If the code compresses such an arbitrary transaction well, it "belongs" to the database.

This observation suggests a classification algorithm. Split the training database according to class and remove the item(s) indicating the class from all transactions. Then, compute a code table for each of these databases. The set of code tables so derived form a classifier: a transaction t gets assigned to the class C whose code table compresses t best.

The accuracy of this classifier is an independent characterisation of the quality of the (small) set of frequent item sets our compression based mechanism picks from the huge set available. The experiments of this paper show that this classifier performs on par with well-known rule-induction and association-rule based methods. In other words, compression picks those sets that capture the characteristics of the data: the patterns that matter.

The roadmap of this paper is as follows. Section 2 gives a short outline of the compression algorithm, after which section 3 details the classification method based on the compression results. In section 4, we discuss some advanced rule-association-based classifiers, to which we compare accuracies obtained with the experiments presented in section 5. We conclude this paper with discussion and conclusions in sections 6 and 7.

2 Compression

In this section we give a brief description of our compression based technique to filter out a small set of descriptive item sets from a vast amount of (frequent) item sets; a full description can be found in [14]. To make referring to our compression method easier, we will name it Krimp from now on (which is Dutch for "to shrink").

The first essential element of Krimp is a *code table*. Such a code table has item sets on the left-hand side and the code for this 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, but their lengths are. To explain how these lengths are computed we first have to introduce the second essential element of Krimp, the coding algorithm.

A transaction t is encoded by Krimp by searching for the first item set c in the left-hand side of 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 coding 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 code 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 [7] we have the optimal code length for c as:

$$l_{CT}(c) = -\log(\mathrm{P}(c \mid db)) = -\log\left(\frac{freq(c)}{\sum_{d \in CT} freq(d)}\right) \quad (\mathbf{1})$$

The length of the encoding of a transaction is now simply the sum of the code lengths of the item sets in its cover. The size of the encoded database is the sum of the size of the encoded transactions, i.e.

$$L_{CT}(db) = -\sum_{c \in CT} freq(c) \cdot \log\left(\frac{freq(c)}{\sum_{d \in CT} freq(d)}\right) \quad (\mathbf{2})$$

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 left-hand side column, note that the simplest code table possibly consists only of the singleton item sets. This is the *standard encoding* 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:

$$L(CT) = \sum_{c \in CT : freq(c) \neq 0} l_{st}(c) + l_{CT}(c) \quad (\mathbf{3})$$

In [14] we defined the optimal set of frequent item sets as that one whose associated code table minimizes:

$$L(CT) + L_{CT}(db) \quad (\mathbf{4})$$

The compression algorithm starts with a valid code table (generally 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 (Naïve-Compression).

An alternative algorithm, Compress-and-Prune, considers each existing code table element for pruning when a new candidate has been added: when deleting an existing element does not reduce the compressed size it is put back, otherwise it is permanently pruned. For more details, please see [14].

3 Classification

As usual in data mining, and data analysis in general, we assume that our database of transactions is an i.i.d. sample from some underlying data distribution. The result of any data mining algorithm is only useful if it reflects structure in the underlying distribution rather than spurious structure in the sample. Translated to Krimp, this means that we expect Krimp to compress an arbitrary transaction sampled from the underlying distribution well. Phrased differently, if it doesn't compress an arbitrary transaction well, we expect that this transaction is generated by another distribution.

To make this more precise, assume that our code table CT has no zero-frequency item sets. Let t be an arbitrary transaction over the items $\mathcal{I}$, then:

$$\begin{aligned} l_{CT}(t) &= \sum_{c \in \text{cover}(t)} l_{CT}(c) = \sum_{c \in \text{cover}(t)} -\log(\text{P}(c \mid db)) \\ &= -\log\left(\prod_{c \in \text{cover}(t)} \text{P}(c \mid db) \right) \\ &= -\log(\text{P}(t \mid db)) \end{aligned} \tag{5}$$

The last equation is a Naïve Bayes [5] like assumption: we assume that the item sets that cover a transaction are independent. Clearly, this assumption is overly optimistic because the item sets in a cover are not allowed to overlap! However, Naïve Bayes is known to perform well, even if the independence assumption is violated. Therefore, we ignore this violation for the moment.

Now, assume that we have two databases db_1 and db_2 generated from two different underlying distributions. We apply Krimp to both databases and get two code tables, CT_1 and CT_2. We are given a new transaction t that is generated under either the distribution for db_1 or the one for db_2, but we are not told which one. How do we decide to which distribution t belongs? Under the Naïve Bayes assumption, we have:

$$l_{CT_1}(t) < l_{CT_2}(t) \rightarrow \text{P}(t \mid db_1) > \text{P}(t \mid db_2) \tag{6}$$

Hence, the Bayes optimal choice is to assign t to the distribution that leads to the shortest code length. In other words, we can do classification by comparing code lengths! We already know that the result of Krimp is a small set of frequent item sets that is optimal with regard to the MDL principle. The above observation suggests an independent way to assess the quality of this result: how well does it classify?

The construction of the Krimp classifier works as follows:

1. Split the training database according to class
2. Remove the item(s) that indicate the class from each transaction
3. Apply Krimp to each of the databases. This yields a code table CT_i for each class C_i

Then, to classify an unseen transaction t:

1. Compute $l_{CTi}(t)$ for all classes C_i
2. Assign t to the class that minimizes $l_{CTi}(t)$

During compression without pruning it is possible that item sets loose their function as transactions are covered in different ways. Some item sets may have a usage frequency *freq*(*c*) of zero and thus an undecidable code length, as can be seen from Equation 1. This has no effect during the compression phase; they are unused and can simply be ignored in calculating the code table size. However, when encoding unseen test instances during classification these sets might be used and therefore require a code length. Simply deleting them from *CT* would be easy, but this would come down to pruning even when we do not prune. Therefore, we do a Laplace correction. That is, all usage frequencies are increased by one when classifying.

Not all data is equally compressible and this shows even between different classes from the same dataset. The density and size of (the partial) databases have a huge influence on the characteristics and amount of frequent item sets that are generated as code table candidates. As the compression ratios achieved per class differ, it would not be fair to only use the code tables of the final *min-sup* for classification. Therefore we also generate code tables at fixed support intervals *report-sup* during compression, and use these for classification as well.

To pair the code tables for classification we use two methods: absolute and relative. In absolute pairing, the code tables that have been generated at the same support levels are matched. Classification therefore starts at the *max-sup* of the smallest class and continues to *min-sup* with steps of report-sup. Relative pairing matches code tables of the same relative support between 100% of *max-sup* (per class) and 1% in intervals of 1%.

4 Advanced Classifiers

Many algorithms that can be used for classification have been proposed, many of which fall into either the class of rule-induction-based or that of association-rule-based methods. Because we use classification as a quality measure for the patterns that Krimp picks, we will compare our results with those obtained by some of the best existing classifiers. Comparison can be done with rule-induction-based methods such as C4.5 [12], FOIL [13] and CPAR [17], but we are more interested in an in-depth comparison with association-rule-based algorithms like iCAEP [19], HARMONY [15], CBA [9] and LB [11]. We believe this comparison is more interesting because these methods also use a collection of item sets for classification. Because we argued that our method is strongly linked to the principle of Naïve Bayes (NB) [5] its imperative we also compare to this method. Please note that all of these methods were devised with the goal of classification in mind, not that of extracting a small set of interesting patterns. We would therefore expect these methods to outperform the Krimp classifier.

Most related to our method is probably iCAEP. In iCAEP, the database is split in the same per-class way we do, after which the set of Emergent Patterns (EPs) for each class is determined. An Emergent Pattern from a partial database to a full database is an item set that has a much larger (relative) support in the partial than in the full database. All mined EPs together are then considered to be the ‘code table’ and are used

to encode test instances with a different probability distribution for each class. Following the MDL principle, the class giving the minimum code length is then assigned.

Somewhat similar as well is Large Bayes (LB), which mines frequent item sets according to an interestingness measure based on cross-entropy. For each of these item sets, the support per class is determined. After this, classification is done by assigning the class that gives the highest posterior probability; the class-specific supports are used as probability distributions.

A recent classification method based on item sets is HARMONY, which selects the so-called 'Highest Confidence Covering Rule' (HCCR) for each transaction in a training database. Each HCCR is of the form $I \rightarrow c$ (where c denotes the class of the transaction), all HCCRs together form the classification model. Lastly, CBA mines all association rules off the form $I \rightarrow c$ and uses a greedy selection method to build a classifier that consists of a sequence of rules.

5 Experiments

In order to assess the quality of Krimp's data distribution approximation, we tested the constructed classifier on a plethora of UCI databases and compared accuracies to a large range of existing classification algorithms. To show the general applicability of Krimp, we've chosen datasets with a broad range of specifics, ranging from sparse to dense and from tiny to large. As the algorithm currently only deals with item sets, we used discretised versions [3,6] of the databases.

We compare to accuracy scores taken from the publications in which the respective classifiers were described [11,15,19]. The missing scores for Naïve Bayes and C4.5 have been acquired using Weka [16].

All experimental results were obtained using 10-fold cross-validation. The data sets were randomly sorted and split into folds. Each resulting training database was then split into its classes for individual compression, as described in Section 3. The actual *min-sup* thresholds we used for each dataset are reported.

The chess (king-rook vs. king-pawn) and connect-4 datasets are so dense that mining and storing all frequent item sets for low *min-sup* thresholds is impossible. To circumvent this, we removed the False resp. Blank features from these datasets. Though this makes database transactions less informative, it decreases the density such that we can mine candidates for compression at much lower minimum support levels.

Although virtually any collection of item sets could be considered as candidates, our focus has been on using all frequent and closed frequent item sets.

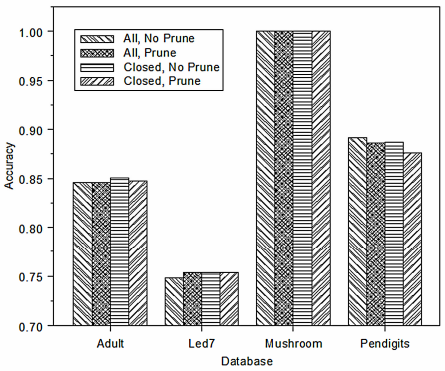

Fig. 1. Accuracies for different candidate sets (all/closed frequent item sets) and pruning enabled/disabled, for 4 datasets.

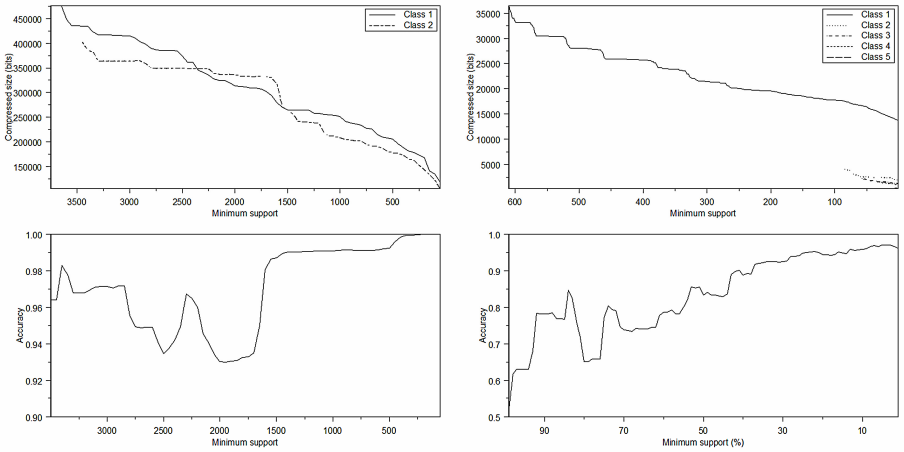

Fig. 2. Minimum support vs total compressed size per class and accuracy for mushroom (absolute pairing, left) and anneal (relative pairing, right).

Figure 1 shows that using either all or closed frequent item sets and Naïve-Compression or Compress-and-Prune hardly influences classification. Because the reduction of the number of item sets is largest with all frequent item sets and on-the-fly pruning [14], we use this combination in all experiments presented in the rest of this section.

Figure 2 shows that better compression generally leads to better classification. This is not always the case though: for the mushroom dataset, a drop in total compressed size for one of the classes always results in a change in accuracy, but this change is not always positive. The drops in accuracy left of the centre of the graph could have several causes, the most likely of which is that item sets that are characteristic for both classes are added to the code tables at that support level. This would make the

Table 1. Statistics on 11 UCI datasets using Compress-and-Prune with all frequent item sets as candidates. Numbers of candidates, resulting code table sizes |CT| (excluding singleton sets) and compression ratios at given min-sups are summed for all classes and 10-fold cross validated. The best 10-fold cross validated accuracies are given (not always belonging to *min-sup*).

Dataset	**#rows**	**|C|**	**|$\mathcal{I}$|**	**#candidates**	**|CT|**	**Compr ratio**	**Acc %**	**min sup**
Adult	48842	2	97	1679483	994	3.40	84.6	50
Anneal	898	6	71	2117941	110	2.47	97.0	1
Chess (kr-kp)	3196	2	40	867877	458	1.68	94.1	1
Connect-4	67556	3	87	2284323	1988	2.29	69.9	50
Ionosphere	351	2	157	42908227	107	1.47	90.6	35
Led7	3200	10	24	12815	720	3.49	75.3	1
Letrecog	20000	26	102	1470766	2164	1.97	68.1	50
Mushroom	8124	2	119	92163948	391	3.94	100.0	50
Pendigits	10992	10	86	255838056	1996	2.19	88.6	1
Waveform	5000	3	101	942271	741	1.71	77.2	100
Wine	178	3	68	1049213	167	1.26	97.7	1

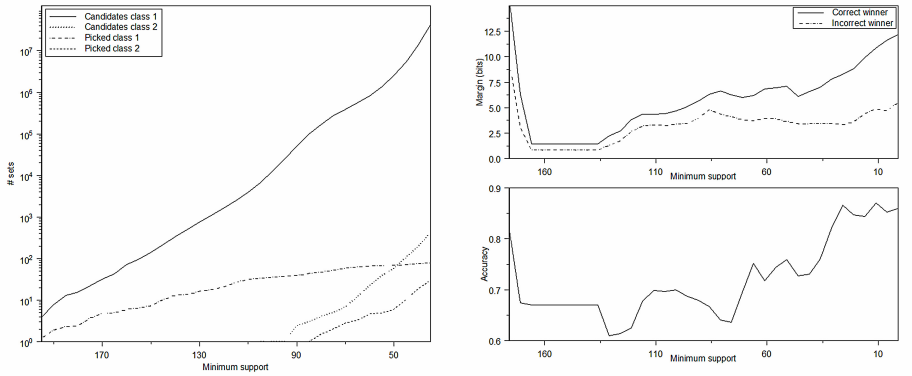

Fig. 3. a) Minimum support against total number of candidates and selected item sets for ionosphere. b) Accuracy and encoded length margins for tic-tac-toe. The margins plotted are the average differences in encoded lengths for all test instances between the two classes, where correct/incorrect winners are aggregated.

encoded probability distributions look more alike and make discrimination more difficult. The (generally longer) candidates with lower support can make an important difference between the classes though, as a 100% classification score is obtained later on.

In case of the mushroom dataset, code tables for the two classes are paired absolute, e.g. having the same minimum support. This is not possible for the anneal dataset, as it consists of 5 classes that have a very skewed a priori distribution. Neither multi-class datasets nor skewed class distributions pose a problem that cannot be tackled by our classifier though: using relative code table pairing, competitive scores can be obtained for anneal. The accuracy graph also emphasizes that better compression doesn't always result in better classification: the highest accuracy is achieved shortly before *min-sup* is reached. This might be caused by slight overfitting on the training data at very low support levels.

In Table 1 we provide an overview of the achieved compression together with classification scores of Krimp on a variety of UCI datasets. The numbers clearly indicate that Krimp can cope with a very wide range of datasets and is always able to strongly reduce the number of candidates. In most cases the reduction of the number of item sets is huge, up till a giant 5 orders of magnitude. Looking at the min-sups, only waveform seems like a strange outlier: because of its characteristics and density, compressing is computationally very intensive, despite the seemingly small numbers of rows and candidates.

Figure 3a visualizes the enormous explosion of the number of frequent item sets that occurs for lower support levels of the ionosphere database. Even on a logarithmic scale, the line representing the number of candidates of class 1 is super-linear. The number of item sets picked stays at a much more reasonable level though. The figure also shows the large differences that may exist between different classes: one class is extremely large, while the other is much smaller. Because of the smaller number of transactions, far less candidates are generated for this class. The differences between the sizes of the resulting code tables are not so extreme though.

Table 2. Accuracy scores on 13 UCI datasets, 10-fold cross validated. Scores taken from [19], additional scores for Naive Bayes and C4.5 obtained using Weka [16].

Dataset	#cl	#rows	iCAEP	CAEP	NB	LB	CBA	C4.5	Krimp	min sup
Adult	2	48842	80.9	83.1	82.7	85.1	75.2	**85.5**	84.6	50
Anneal	6	898	95.1	85.7	86.3		**98.1**	90.4	97.0	1
Breast (Wisc)	2	699	**97.4**	97.0	96.0	96.9	95.3	85.4	94.1	1
Chess (kr-kp)	2	3196	94.6	85.5	87.9	90.2	98.1	**99.4**	94.1	1
Connect-4	3	67556	69.9	73.0	72.1			**81.0**	69.9	50
Iono	2	351	90.6	87.2	82.6		**92.1**	91.5	90.6	35
Iris	3	150	93.3	94.0	**96.0**		92.9	84.7	**96.0**	1
Mushroom	2	8124	99.8	93.0	95.8			**100.0**	**100.0**	20
Nursery	5	12960	84.7	84.4	90.3			**97.1**	92.4	1
Pima	2	768	72.3	73.3	74.7	**75.8**	73.1	72.5	75.0	1
Tic-tac-toe	2	958	92.6	85.9	69.6		**100**	84.6	87.1	1
Waveform	3	5000	81.7	**83.9**	80.0	79.4	75.3	75.1	77.2	100
Wine	3	178	**98.9**	96.1	96.6		91.6	93.8	97.7	1

Figure 3b shows that when compression proceeds and classification accuracies improve, the difference in average encoded lengths (for all test instances) between the winning and losing code tables also changes. As accuracy increases, the difference between the encoded lengths of the correctly (incorrectly) winning code table and that of the losing code table also increases (decreases). In other words: the approximations of the data distributions for the specific classes become better and better. To find out how well our classifier actually performs, we will now compare the obtained accuracies to those of well-known classifiers.

The comparison of classification accuracies in Table 2 shows that the Krimp code tables are of such high quality they can compete with many methods specifically built for classification. Our elegant though make-shift MDL based classification extension delivers two wins, while in other cases it's found at the head of the pack. Only on the (very dense) Connect-4 dataset it finishes last, together with iCAEP. Note that on average Krimp performs better than Naïve-Bayes, although it based on the same assumption. Unfortunately no further results for LB and CBA were available.

When comparing on some of the largest UCI datasets in Table 3, Krimp proves to

Table 3. Accuracy scores on 10 large UCI databases, 10-fold cross validated. Scores taken from [15]. *Min-sup* for Harmony set to 50.

Dataset	#cl	#rows	FOIL	CPAR	SVM	Harmony	Krimp	min sup
Adult	2	48842	82.5	76.7	84.2	81.9	**84.6**	50
Chess (kr-k)	18	28056	42.6	32.8	29.8	44.9	**58.0**	10
Connect-4	3	67557	65.7	54.3	**72.5**	68.1	69.9	50
Led7	10	3200	62.3	71.2	73.8	74.6	**75.3**	1
LetterRecog	26	20000	57.5	59.9	67.8	**76.8**	68.1	50
Mushroom	2	8124	99.5	98.8	99.7	99.9	**100.0**	20
Nursery	5	12960	91.3	78.5	91.4	**92.8**	92.4	1
PageBlocks	5	5473	91.6	76.2	91.2	91.6	**96.6**	1
PenDigits	10	10992	88.0	83.0	93.2	**96.2**	88.6	1
Waveform	3	5000	75.6	75.4	**83.2**	80.5	77.2	100

do exceptionally well. The bigger datasets allow MDL to better work its magic, making a good selection of which sets to include in its code tables. Though we do not report on this further, we would like to remark that higher scores might be achieved using experimental code table pairing techniques; scores of 76.6% for LetterRecog have been recorded.

Harmony achieved a score of 58.4% for the large chess dataset at a *min-sup* of 10, which is comparable to that obtained by our Krimp classifier. We could also improve slightly on this dataset by disabling on-the-fly pruning: this would raise the obtained accuracy to 58.9%. Overall, Krimp performs very well, as it wins on 5 out of the 10 large databases and is pretty close to the best scores in other cases.

6 Discussion

Outstanding classification was not the ultimate goal of the experiments we presented in the previous section, as we explained before. We are very content as our actual intention was to verify that our method is very capable at representing data distributions with only few item sets. The results of the experiments clearly show this hypothesis is correct.

Although Krimp has not been designed for classification and no attempts have been made to enhance the differences between code tables for different classes, the results show that it performs well compared to the best known classifiers. As the mentioned association-rule-based classifiers also select item sets that characterise the classes, it is interesting to compare our method with those in a qualitative way.

The selection method iCAEP uses is far less effective than MDL: the amount of Emergent Patterns may grow enormously and a lot of item sets may be required in the end. Also, Krimp is independent of the base distribution, iCAEP is not. Large Bayes does succeed in selecting only a small set of item sets that is used to determine a class-based probability distribution, but it uses an interestingness measure that requires a parameter that needs to be chosen manually.

HARMONY selects at most one rule per transaction in the training database. Although it is likely that equal rules are selected and therefore merged, the final set of rules can still be quite large. In Krimp, an item set is only used if it helps to compress the whole training database; we therefore believe that HARMONY is more prone to noise and overfitting than our method and the rules do not represent the data as well. CBA has a very rough selection method that is likely to result in large numbers of rules that do not represent the complete database well.

The elegancy of the Krimp classifier lies in 1) the use of only MDL for both building and applying the classifier, 2) the small amount of item sets required and 3) the natural way it deals with multi-class problems and skewed class distributions.

Although the compression of the item sets is already enormous (in case of PenDigits, the beastly amount of 255 million item sets is cut back to only 1996!), several paths might lead to an even stronger reduction. An example would be to allow overlap in the cover of transactions. Though this would make covering and proper selection of item sets computational more complex, fewer item sets would be necessary for a full database cover.

This leads us to another important issue in data mining: what about scalability? Our current highly optimised implementation is about 150 times faster than the implementation we used for the previous paper. The time required to consider a single candidate scales linearly with its support, meaning that the speed at which candidates can be considered increases considerably at low support levels. Our implementation therefore scales extremely well, making compression of large databases feasible.

Nevertheless, for certain datasets the number of candidates explodes extremely at low *min-sup* levels. Despite our smart implementation this can become a problem as one would like to get the best compression and (generally longer) candidates generated at low support might be required.

For many datasets, using only closed frequent sets as candidates saves a lot of time. As our experiments show that using only closed frequent item sets as candidates hardly influences classification, the closed reduction is good at preserving the most important item sets and is therefore an attractive alternative for the full frequent set. Also, on-the-fly pruning speeds up compression quite a bit and gives smaller code tables; the (on average) slightly worse classification is negligible.

A big advantage of Krimp is the ease with which it can be distributed. As only very few candidates are kept in the code table, one can easily slice up their testing. The overhead of code table synchronization and partial re-evaluation would be much smaller than the time won by speeding up with a factor as big as the distribution employed. Further, for classification it's straightforward that the per-class compression tasks can be run on different computers. Even classification itself could be split up, as each row has to be compressed individually.

Still, in our view it would be even better to find a clever way of generating candidates on-the-fly and pruning the search space based on the current database cover, making use of bounds on their effect on compression. This has our focus, as it would allow a dramatic reduction of the number of candidates while maintaining the quality of the resulting item sets.

7 Conclusion

Krimp picks the item sets that matter. From staggering amounts it selects only handfuls of item sets that not only attain high compression ratios, but compete head on with today's cutting edge classifiers as well. We therefore conclude that Krimp is very well suited for capturing the characteristics of the data.

As an independent measure of the quality of the selected item sets we used classification. The training data is compressed per class, the resulting code tables are used to encode new test instances. Following Bayes optimal choice, the class with the code table assigning the shortest code has the highest probability of being the correct data distribution and is therefore chosen winner. Classification accuracies achieved are on par with the best known classifiers.

Not only is the selected collection of item sets of high quality, the reduction of the number of candidates is huge, generally many orders of magnitude. Krimp thus proves to be a generic method that finds small sets of patterns that encapsulate the probability distribution of the data well.

We are currently investigating whether Krimp is also suited for other types of data, such as bags, trees and graphs. In the future, the general approach could also be applied in unsupervised learning tasks like clustering. Our architecture shows great scalability and is easy to distribute, which would be an interesting line to pursue.

These properties make it very applicable for real-world applications. Recently we've embarked on a bioinformatics collaboration through which we not only obtained usable data, but also the expertise that is required to assess the quality of the found patterns. The humble number of patterns picked by Krimp can be relatively easily interpreted by experts and are expected to reveal valuable knowledge.

References

1. Agrawal, R., Imielinksi, T. & Swami, A. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD conference*, 207-216 (1993).
2. Bayardo, R. Efficiently mining long patterns from databases. In *Proc. ACM SIGMOD conference*, 85-93 (1998).
3. Coenen, F. The LUCS-KDD Discretised/normalized ARM and CARM Data Library. In *www.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html*.
4. Dong, G., Zhang, X., Wong, L. & Li, J. CAEP: Classification by Aggregating Emerging Patterns. In Proc. *Discovery Science* (1999).
5. Duda, R. & Hart, P. *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York (1973).
6. Goethals, B. et al. FIMI website. In *fimi.cs.helsinki.fi*
7. Grünwald, P.D. Minimum description length tutorial. In Grünwald, P.D., Myung, I.J. & Pitt, M.A., editors, *Advances in Minimum Description Length*. MIT Press (2005).
8. Jaroszewicz, S. & Simovici, D.A. Interestingness of Frequent Itemsets Using Bayesian Networks as Background Knowledge. In *Proc. KDD-04*, Seattle (2004).
9. Liu, B., Hsu, W. & Ma, Y. Integrating Classification and Association Rule Mining. In *Proc. KDD-98*, New York (1998).
10. Mannila, H. & Toivonen, H. Multiple uses of frequent sets and condensed representations. In *Proc. of the Conference on Knowledge Discovery and Data Mining* (1996).
11. Meretakis, D. & Wüthrich, B. Extending Naïve Bayes Classifiers Using Long Itemsets. In *Proc. of the Conference on Knowledge Discovery and Data Mining* (1999).
12. Quinlan, J.R. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, Los Altos, California (1993).
13. Quinlan, J.R. FOIL: A Midterm Report. In *Proc. ECML '93* (1993).
14. Siebes, A.P.J.M., Vreeken, J. & Van Leeuwen, M. Item Sets That Compress. In *Proc. of the ACM SIAM Conference on Data Mining* 393-404 (2006).
15. Wang, J. & Karypis, G. HARMONY: Efficiently Mining the Best Rules for Classification. In *Proc. of the ACM SIAM Conference on Data Mining* (2005).
16. Witten, I.H. & Frank, E. *Data Mining: Practical machine learning tools and techniques*, 2nd edition, Morgan Kaufmann, San Francisco (2005).
17. Yin, X. & Han, J. CPAR: Classification based on Predictive Association Rules. In *Proc. SDM '03* (2003).
18. Zaki, M.J. & Orihara, M. Theoretical foundations of association rules. In *Proc. ACM SIGMOD workshop on research issues in KDD* (1998).
19. Zhang, X., Guozhu, D. & Ramamohanarao, K. Information-based Classification by Aggregating Emerging Patterns. In *Proc. IDEAL* (2000).