Data Min Knowl Disc (2009) 19:176–193
DOI 10.1007/s10618-009-0137-2

Identifying the components

Matthijs van Leeuwen · Jilles Vreeken · Arno Siebes

Received: 11 June 2009 / Accepted: 20 June 2009 / Published online: 17 July 2009

Abstract Most, if not all, databases are mixtures of samples from different distributions. Transactional data is no exception. For the prototypical example, supermarket basket analysis, one also expects a mixture of different buying patterns. Households of retired people buy different collections of items than households with young children. Models that take such underlying distributions into account are in general superior to those that do not. In this paper we introduce two MDL-based algorithms that follow orthogonal approaches to identify the components in a transaction database. The first follows a model-based approach, while the second is data-driven. Both are parameter-free: the number of components and the components themselves are chosen such that the combined complexity of data and models is minimised. Further, neither prior knowledge on the distributions nor a distance metric on the data is required. Experiments with both methods show that highly characteristic components are identified.

Keywords MDL · Database components · Clusters

1 Introduction

Most, if not all, databases are mixtures of samples from different distributions. In many cases, nothing is known about the source components of these mixtures. Therefore,

Responsible editors: Aleksander Kołcz, Wray Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor.

M. van Leeuwen (✉) · J. Vreeken · A. Siebes
Department of Computer Science, Universiteit Utrecht, Utrecht, The Netherlands
e-mail: mleeuwen@cs.uu.nl

J. Vreeken
e-mail: jillesv@cs.uu.nl

A. Siebes
e-mail: arno@cs.uu.nl

many methods that induce models regard a database as sampled from a single data distribution. While this greatly simplifies matters, it has the disadvantage that it results in suboptimal models.

Models that do take into account that databases actually are sampled from mixtures of distributions are often superior to those that do not, independent of whether this is modelled explicitly or implicitly. A well-known example of explicit modelling is mixture modelling (Titterington et al. 1985). This statistical approach models data by finding combinations of several well-known distributions (e.g., Gaussian or Bernoulli) that are assumed to underlie the data.

Boosting algorithms (Freund and Schapire 1997) are a good example of implicit modelling of multiple underlying distributions. The principle is that a set of weak learners can together form a single strong learner. Each separate learner can adapt to a specific part of the data, implicitly allowing for modelling of multiple distributions.

Transaction databases are no different with regard to data distribution. As an illustrative example, consider supermarket basket analysis. One also expects a mixture of different buying patterns: different groups of people buy different collections of items, although overlap may exist. By extracting both the groups of people and their corresponding buying patterns, a company can learn a lot about its customers.

Part of this problem is addressed by clustering algorithms, as these group data points together, often based on some distance metric. However, since we do not know upfront what distinguishes the different groups, it is hard to define the appropriate distance metric. Furthermore, clustering algorithms such as k-means (MacQueen 1967) only find the groups of people and do not give any insight in their corresponding buying behaviours. The only exception is bi-clustering (Pensa et al. 2005), however, this approach imposes strong restrictions on the possible clusters.

Frequent item set mining, on the other hand, does give insight into what items customers tend to buy together, e.g., the (in) famous *Beer* and *Nappies* example. But, not all customers tend to buy *Nappies* with their *Beer* and standard frequent set mining does not distinguish groups of people. Clearly, knowing several groups that collectively form the client base as well as their buying patterns would provide more insight. So, the question is: can we find these groups and their buying behaviour? That is, we want to partition the database db in sub-databases $db_1, \ldots, db_k$ such that

- the buying behaviour of each db_i is different from all db_j (with $j \neq i$), and
- each db_i itself is homogeneous with regard to buying behaviour.

But, what does "different buying behaviour" mean? It certainly does not mean that the different groups should buy completely different sets of items. Also, it does not mean that these groups cannot have common frequent item sets. Rather, it means that the *characteristics* of the sampled distributions are different. This may seem like a play of words, but it is not. Sampled distributions of transaction data can be characterised precisely through the use of *code tables*.

In Siebes et al. (2006) we introduced the KRIMP algorithm.[1] This MDL-based algorithm picks a few descriptive frequent item sets that compress the data well. These

[1] Although, not yet by that name.

item sets are selected from the gigantic number of frequent item sets that are produced at low *minsup* settings; reductions of up to 7 orders of magnitude are attained. In subsequent research it has been shown that these small sets (or better, the *code tables* they come from; see Sect. 2.2) characterise the sampled distribution of the underlying database very well. More in particular, van Leeuwen et al. (2006) shows how these code tables naturally induce good classifiers. In Vreeken et al. (2007) it is shown that the code tables also induce a dissimilarity measure on transaction databases over the same set of items. In Sect. 3 we will give a brief introduction to KRIMP.

We use code tables and MDL to formalise our problem statement. That is: find a partitioning of the database such that the total compressed size of the components is minimised. This problem statement agrees with the compression perspective on data mining as recently positioned by Faloutsos and Megalooikonomou (2007).

We propose two orthogonal methods to solve the problem at hand. The first method is based on the assumption that KRIMP implicitly models all underlying distributions in a single code table. If this is the case, it should be possible to extract these and model them explicitly. We propose an algorithm to this end that optimises the compressed total size by specialising copies of the original compressor to the different partial distributions of the data.

Throughout our research we observed (van Leeuwen et al. 2006) that by partitioning a database, e.g., on class label, compressors are obtained that encode transactions from their 'native' distribution shortest. This observation suggests an iterative algorithm that resembles *k*-means: without any prior knowledge, randomly split the data in a fixed number of parts, induce a compressor for each and re-assign each transaction to the compressor that encodes it shortest, etcetera. This scheme is very generic and could also be easily used with other types of data and compressors. Here, as we are concerned with transaction data, we employ KRIMP as compressor.

Both algorithms are implemented and evaluated on basis of total compressed sizes and component purity, but we also look at (dis)similarities between the components and the code tables. The results show that both our orthogonal methods identify the components of the database, without parameters: the optimal number of components is determined by MDL. Visual inspection confirms that characteristic decompositions are identified.

2 Problem statement

2.1 Preliminaries

As usual in (frequent) item set mining (Han et al. 2000) we assume a set of items $\mathcal{I}$. A *transaction* t is simply a set of items, i.e., $t \subseteq \mathcal{I}$. A database *db* is a bag of transactions.

The patterns we are interested in are called *item sets*. Again, an item set I is simply a set of items, i.e, $I \subseteq \mathcal{I}$. An item set I *occurs* in a transaction t iff $I \subseteq \mathcal{I}$. The *support* of an item set I in database *db* is denoted by $\text{sup}_{db}(I) = |\{t \in db | I \subseteq t\}|$.

For a given minimum support threshold *minsup*, an item set I is called *frequent* if its support exceeds *minsup* on *db*, i.e., $\sup_{db}(I) \geq minsup$.

2.2 MDL for item sets

MDL (minimum description length; Grünwald 2005) is a practical version of Kolmogorov Complexity (Li and Vitányi 1993). Both embrace the slogan *Induction by Compression*. For MDL, this principle can be roughly described as follows.

Given a set of models $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimises

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

in which

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

In order to use this principle for our problem statement, we need to define our collection of models and how to encode the data with such a model. Moreover, we need to determine how many bits are necessary to encode a model and how many are necessary for it to encode the data.

The remainder of this subsection is taken mostly taken from Siebes et al. (2006) and provided here for the convenience of the reader. We use (sets of) *code tables* as our models. Such a code table is defined as follows.

Definition 1 *Let $\mathcal{I}$ be a set of items and $\mathcal{C}$ a set of code words. A code table CT for $\mathcal{I}$ and $\mathcal{C}$ is a two column table such that:*

1. *The first column contains item sets over $\mathcal{I}$, this column contains at least all singleton item sets. Elements are ordered descending on (1) length, (2) support and (3) lexicographically.*
2. *The second column contains elements from $\mathcal{C}$, such that each element of $\mathcal{C}$ occurs at most once.*

An item set $I \in \mathcal{P}(\mathcal{I})$ occurs in CT, denoted by $I \in CT$ iff I occurs in the first column of CT, similarly for a code $C \in \mathcal{C}$. For $I \in CT$, $code_{CT}(I)$ denotes its code, i.e., the corresponding element in the second column.

To encode a transaction t from database *db* over $\mathcal{I}$ with code table *CT*, we use the COVER algorithm from Siebes et al. (2006) given in Algorithm 1. Its parameters are a code table *CT* and a transaction t, the result is a disjoint set of elements of *CT* that cover t. Note that COVER is a well-defined function on any code table *CT* and any transaction t, since *CT* contains at least the singletons.

To encode database *db*, we simply replace each transaction $t \in db$ by the codes of the item sets in its cover: $t \rightarrow \{\text{code}_{CT}(I) \mid I \in \text{COVER}(CT, t)\}$. Note that to ensure that we can decode an encoded database uniquely we assume that $\mathcal{C}$ is a *prefix code*.

Since MDL is concerned with the best compression, the codes in *CT* should be chosen such that the most often used code has the shortest length. That is, we use

Algorithm 1 Cover

```
COVER($CT, t$)
1 $S$ := first element $I \in CT$ for which $I \subseteq t$
2 if $t \setminus S = \emptyset$
3     then $Res := \{S\}$
4     else $Res := \{S\} \cup$ COVER($CT, t \setminus S$)
5 return $Res$
```

an optimal prefix code, i.e., for which the Shannon entropy provides us the optimal lengths. Note that we are not interested in materialised codes, but only in their lengths. To determine these, we need to know how often a certain code is used. We define the *frequency* of an item set I in *CT* as the number of transactions in *db* in which I is in its cover. Normalised, this frequency represents the probability that that code is used in the encoding of an arbitrary $t \in db$,

$$P(I|db) = \frac{\text{freq}_{db}(I)}{\Sigma_{J \in CT} \text{freq}_{db}(J)}.$$

The optimal code length is then $-\log$ of this probability and the code table is optimal if all its codes have their optimal length. That is, a code is optimal for *db* iff $|\text{code}_{CT}(I)| = -\log(P(I|db))$ and *CT* is *code-optimal* for *db* if all its codes $C \in CT$ are optimal for *db*. From now on, we assume that code tables are code-optimal, unless we state differently.

For any database *db* and code table *CT*, we can now compute $L(db|CT)$. The encoded size of a transaction is the sum of the sizes of the codes of the item sets in its cover, $l(t|CT) = \Sigma_{I \in \text{COVER}(CT,t)} - \log(P(I|db))$. The size of *db*, denoted by $L(db|CT)$, is simply the sum of the sizes of the transactions, $L(db|CT) = \Sigma_{t \in db} l(t|CT)$.

The remaining problem is to determine the size of a code table. For the second column this is clear, as we know the size of each of the codes. For encoding the first column, we use the simplest code table, i.e. the code table that contains only the singleton elements. This code table, with optimal code lengths for database *db*, is called the standard code table for *db*, denoted by *ST*. With this choice, the size of *CT*, denoted by $L_{db}(CT)$ is given by $L_{db}(CT) = \Sigma_{I \in CT} |\text{code}_{ST}(I)| + |\text{code}_{CT}(I)|$.

With these results we know the total size of our encoded database. It is simply the the size of the encoded database plus the size of the code table. That is, we have the following theorem.

Theorem 1 *Let db be a transaction database over $\mathcal{I}$ and let CT be a code table that is code-optimal for db. The total compressed size of the encoded database and the code table, denoted by $L(CT, db)$, is given by $L(CT, db) = L(db|CT) + L_{db}(CT)$.*

Now that we know how to compute $L(CT, db)$, we can formalise our problem using MDL.

2.3 The problem

Our goal is to discover an optimal partitioning of the database; optimal, in the sense that the characteristics of the different components are different, while the individual components are homogeneous.

As discussed before, code tables characterise the sampled distributions of the data. Hence, we want a partitioning for which the different components have different characteristics, and thus code tables. In terms of MDL, this is stated as follows.

Formal Problem Statement *Let $\mathcal{I}$ be a set of items and let db be a bag of transactions over $\mathcal{I}$. Find a partitioning $db_1, \ldots, db_k$ of db and a set of associated code tables $CT_1, \ldots, CT_k$, such that the total encoded size of db*

$$\sum_{i \in \{1,\ldots,k\}} L(CT_i, db_i)$$

is minimised.

There are a few things one should note about this statement. Firstly, we let MDL determine the optimal number of components for us, by picking the smallest encoded size over every possible partitioning and all possible code tables. Note that this also ensures that we will not end up with two components that have the same or highly similar code tables. It would be far cheaper to combine these.

Secondly, asking for both the database partitioning and the code tables is in a sense redundant. For any partitioning, the best associated code tables are, of course, the optimal code tables. The other way around, given a set of code tables, a database partitions naturally. Each transaction goes to the code table that compresses it best. This suggests two ways to design an algorithm to solve the problem. Either one tries to find an optimal partitioning or one tries to find an optimal set of code tables.

Thirdly, the size of the search space is gigantic. The number of possible partitions of a set of n elements is the well-known Bell number B_n. Similarly, the number of possible code tables is enormous. It consists of the set of all sets of subsets of $\mathcal{I}$ that contain at least the singletons. Further, for each of these code tables we would have to consider all possible combinations of covers per transaction. Moreover, there is no structure in this search space that we can use to prune. Hence, we will have to introduce heuristic algorithms to solve our problem.

3 KRIMP preliminaries

3.1 Finding the optimal code table

Recently, we proposed a heuristic algorithm called KRIMP to approximate the optimal code table for a database (Siebes et al. 2006). For this it needs a database and a set of candidate item sets as input. As candidates, frequent item sets up to a given *minsup* are used. The candidate set is ordered first descending on support, second descending on item set length and third lexicographically. KRIMP starts with the standard code table *ST*. One by one, each pattern in the candidate set is added to the code table to see if it helps to improve database compression. If it does, it is kept in the code table, otherwise it is removed. After this decision, the next candidate is tested. In all experiments reported in this paper, pruning is applied, meaning that each time an item set is kept in the code table all other elements are tested to see whether they still contribute to compression. Elements that do not are permanently removed. See Siebes et al. (2006) for further details.

3.2 Dissimilarity

As code tables are very specific for different distributions, it is possible to define a dissimilarity measure on entire databases (Vreeken et al. 2007). Of course, this only works for databases over the same set of items $\mathcal{I}$. The encoded size of database y compressed by the code table induced by KRIMP for database x, $L\left(y|CT_x\right)$, is denoted by the shorthand notation $CT_x\left(y\right)$. Now define the code table dissimilarity *DS* between x and y as

$$DS(x, y) = \max \left\{ \frac{CT_y(x) - CT_x(x)}{CT_x(x)}, \frac{CT_x(y) - CT_y(y)}{CT_y(y)} \right\}.$$

The databases are deemed very similar (possibly identical) iff the score is 0, higher scores indicate greater dissimilarity.

3.3 Datasets

We use a number of UCI datasets (Coenen 2003), the Retail dataset (Brijs et al. 1999) and the Mammals dataset (Heikinheimo et al. 2007) for experimental validation of our methods. The latter consists of presence records of 121 European mammals in geographical areas of 50 × 50 km.[2] The properties of these datasets are listed in Table 1. For fair comparison between the found components and the original classes, the class labels are removed from the databases in our experiments. In addition, KRIMP candidates (all or closed frequent item sets up to *minsup*), the regular (single-component) KRIMP compressed size of the database and class dissimilarities are given. Average class dissimilarities are computed by splitting the database on class label, computing pair-wise code table dissimilarities as defined above and taking the average over these. Both Retail and Mammals datasets do not contain class labels.

4 Model-driven component identification

In this section we present an algorithm that identifies components by finding an optimal set of code tables.

[2] The full dataset (Mitchell-Jones et al. 1999) is available upon request from the Societas Europaea Mammalogica, http://www.european-mammals.org.

4.1 Motivation

A code table induced for a complete database captures the entire distribution, so the multiple underlying component distributions are modelled implicitly. This suggests that we should be able to extract code tables for specific components from the code table induced on the whole database, the *original* code table.

If the data in a database is a mixture of several underlying data distributions, the original code table can be regarded as a mixture of code tables. This implies that all patterns required for the components are in the code table and we only need to 'extract' the individual components. In this context, each possible subset of the original code table is a candidate component and thus each set of subsets a possible decomposition. So, given an original code table *CT* induced for a database *db*, the optimal model driven decomposition is the set of subsets of *CT* that minimises the total encoded size of *db*.

Obviously, the optimal decomposition could be found by simply trying all possible decompositions. However, although code tables consist of only few patterns (typically hundreds), the search space for such approach is enormous. Allowing only partitions of the code table would strongly reduce the size of the search space. But, as different distributions may have overlapping common elements, this is not a good idea. The solution is therefore to apply a heuristic.

4.2 Algorithm

The algorithm, presented in detail in pseudo code in Algorithm 2, works as follows: first, obtain the overall code table by applying KRIMP to the entire database (line 1). Then, for all possible values of k, i.e., $k \in [1, |db|]$, identify the best k components. We return the solution that minimises the total encoded size (2–3).

Algorithm 2 Model-Driven Component Identification

```
     IdentifyTheComponentsByModel ($db$, $minsup$)
1      $CT_{\text{orig}}$ := KRIMP($db$, MineFreqSets ($db$, $minsup$))
2      $b$ := argmin$_{k \in [1, |db|]}$ CalcEncodedSize(IdentifyKComponentsByModel ($db$, $CT_{\text{orig}}$, $k$))
3      return IdentifyKComponents ($db$, $CT_{\text{orig}}$, $b$)

     IdentifyKComponentsByModel ($db$, $CT_{\text{orig}}$, $k$)
4      $C$ := {$k$ Laplace corrected copies of $\text{CT}_{\text{orig}}$}
5      do $e$ := FindBestElimination ($db$, $C$, $k$)
6        ApplyElimination ($C$, $e$)
7      while *compressed size decreases*
8      return $C$

     FindBestElimination ($db$, $C$, $k$)
9      ($C_b$, $B$) := argmin $C_i \in C$, $I \in C_i$ CalcEncodedSize ($db$, $(C \backslash C_i) \cup (C_i \backslash I)$, $k$)
10     return ($C_b$, $B$)

     CalcEncodedSize ($db$, $C$, $k$)
11     for each transaction $t \in db$ : assign $t$ to $CT_i \in C$ that gives the shortest code
12     for each $CT_i \in C$: ComputeOptimalCodes ($db_i$, $CT_i$)
13     return $\Sigma_i L(C_i, db_i)$
```

To identify k components, start with k copies of the original code table (4). A Laplace correction of 1 is applied to each copy, meaning that the frequencies of all item sets in the code table are increased by one. This correction ensures that each code table can encode any transaction, as no item sets with zero frequency (and therefore no code) can occur. Now, iteratively eliminate that code table element that reduces the total compressed size most; until compression cannot be improved any further (5–7). Iteratively, each element in each code table is temporarily removed to determine the best possible elimination (9). To compute the total compressed size, each transaction is assigned to that code table that compresses it best (11). This can be translated as being the Bayes optimal choice (van Leeuwen et al. 2006). After this, re-compute optimal code lengths for each component (frequencies may have changed) and compute the total encoded size by summing the sizes of the individual components (12–13).

4.3 Experimental results

The results of the experiments with the algorithm just described are summarised in Table 2. By running IdentifyTheComponentsByModel for all values of k and choosing the smallest decomposition, MDL identifies the optimal number of components. However, the search space this algorithm considers grows enormously for large values of k, so in our experiments we imposed a maximum value for k.

The compression gain is the reduction in compressed size relative to that attained by the original code table; the regular KRIMP compressed size as given in Table 1. The compression gains in Table 2 show that the algorithm ably finds decompositions that allow for a much better compression than when the data is considered as a single component. By partitioning the data, reductions from 9 up to 46% are obtained. In other words, for all datasets compression improves by using multiple components.

We define purity as the weighted sum of individual component purities, a measure commonly used in clustering. Hence, the baseline purity is the percentage of

Table 1 Statistics on the datasets used in the experiments

Dataset	Basic statistics				KRIMP			
	$\|db\|$	$\|\mathcal{I}\|$	$\|C\|$	Purity	Cand's	$minsup$	$L(CT, db)$	ClassDS
Adult	48842	95	2	76.1	Closed	20	841604	0.8
Anneal	898	65	6	76.2	All	1	21559	6.3
Chess (kr-k)	28056	40	18	16.2	All	10	516623	1.3
Mammals	2183	121	–	–	Closed	150	164912	–
Mushroom	8124	117	2	51.8	Closed	1	272600	8.4
Nursery	12960	21	9	33.3	Closed	1	240537	4.2
PageBlocks	5473	33	11	89.8	All	1	7160	10.6
Retail	88162	16470	–	–	All	16	10101135	–

As basic statistics, provided are the number of transactions, number of items, number of classes and purity (the accuracy by majority class voting). KRIMP-specific are the candidates used (all or closed frequent item sets), the *minsup* at which the candidates are mined, the total compressed size in bits and the average code table dissimilarity between the classes

Table 2 Experimental results for Model-Driven Component Identification

Dataset	Model-Driven Component Identification				
	Compression gain (%)	Component purity (%)	Average DS	Optimal *k*	Maximum *k*
Adult	8.7	76.1	6.44	2	2
Anneal	18.1	80.8	5.96	19	30
Chess (kr-k)	14.5	18.2	2.86	6	7
Mushroom	25.7	88.2	11.32	12	15
Nursery	11.7	45.0	1.77	14	20
Pageblocks	46.4	91.5	804.9	6	40

Given are the gain in compression over the single component KRIMP compression, component purity by majority class voting, average dissimilarity between the components, the optimal value of *k* and the maximum value of *k*

transactions belonging to the majority class. If we look at the obtained purities listed in Table 2 and compare these to the baseline values in Table 1, we notice that these values range from baseline to very good. Especially the classes of the Mushroom and PageBlocks datasets get separated well. The average (pairwise) dissimilarity between the *Optimal k* components, *Average DS,* shows how much the components differ from each other. We obtain average dissimilarities ranging from 1.7 to 804.9, which are huge considering the values measured between the actual classes (as given in Table 1). Without any prior knowledge the algorithm identifies components that are at least as different as the actual classes. While for the Adult database the obtained purity is at baseline, the data is very well partitioned: the dissimilarity between the components measures 6.4, opposed to just 0.8 between the actual classes.

Arguably, the most important part of the results is the code tables that form the end product of the algorithm: these provide the characterisation of the components. Inspection of these provides two main observations. First, a number of the original elements occur in multiple components, others occur only in a single component and some lost their use. Secondly, the code lengths of the elements are adapted to the specific components: codes become shorter, but lengths also change relative to each other. Example original and resulting code tables are depicted in Fig. 1.

4.4 Discussion

The significantly smaller total encoded sizes after decomposition show that the proposed algorithm extracts different underlying distributions from the mixture. The purities and component dissimilarities show that components are different from each other but homogeneous within themselves.

One of the properties of the method is that the number of (unique) patterns required to define the components is never higher than the number of item sets in the original code table, which consists of only few patterns (van Leeuwen et al. 2006). As the total number of patterns actually used in defining the components is often even smaller,

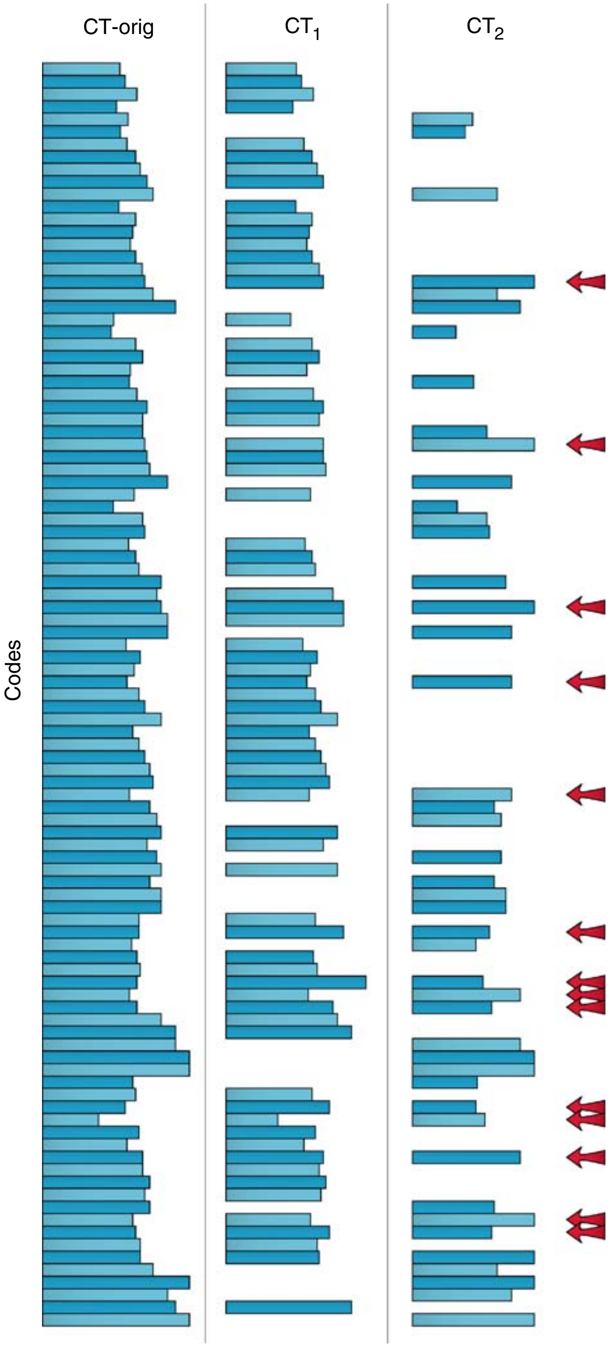

Fig. 1 The components of Anneal. Codes and their lengths (in bits, represented by width) for original and component code tables, $k = 2$. Elements in common are marked by *arrows*

the resulting code tables can realistically be manually inspected and interpreted by domain experts.

The running times for the reported experiments ranged from a few minutes for Anneal up to 80 h for Adult; which is the reason we imposed a *Maximum k* of only two for this dataset. Obviously, more computation time is needed for very dense and/or very large databases, which is why in this section no results are presented for Retail and Mammals. While parallelisation of the algorithm is trivial and should provide a significant speedup, there is another approach to handle large datasets: the data driven method presented in the next section is inherently much faster.

5 Data-driven component identification

In this section, we present an approach to identify the components of a dataset by finding the MDL-optimal partitioning of the data.

5.1 Motivation

Suppose we have two equally sized databases, db_1 and db_2, both drawn from a mixture of distributions D_A and D_B. Further, suppose that db_1 has more transactions from D_A than from D_B and vice versa for db_2. Now, we induce compressors C_1 and C_2 for db_1 and db_2, respectively. Assuming that D_A and D_B are different, C_1 will encode transactions from distribution D_A shorter than C_2, as C_1 has seen more samples from this distribution than C_2. This bias will be stronger if more transactions come from a single distribution – and the strongest when the data consists solely of transactions from one distribution. So, provided a particular source distribution has more transactions in one database than in another, transactions of that distribution will be encoded shorter by a compressor induced on that data.

We can exploit this property to find the components of a database. Given a partitioning of the data, we can induce a code table for each part. Now, we simply reassign each transaction to that part whose corresponding code table encodes it shortest. We thus re-employ the Bayes optimal choice to group those transactions that belong to similar distribution(s) (van Leeuwen et al. 2006). By doing this iteratively until all transactions remain in the same part, we can identify the components of the database. This method can be seen as a form of k-means, without the need for a distance metric and providing insight in the characteristics of the found groupings through the code tables.

5.2 Algorithm

From the previous subsection it is clear what the algorithm will look like, but its initialisation remains an open question. Without prior background knowledge, we start with a random initial partitioning. Because of this, it is required to do a series of runs for each experiment and let MDL pick the best result. However, as transactions are reassigned to better fitting components already in the first iteration, it is expected that

Algorithm 3 Data-Driven Component Identification

```
IdentifyTheComponentsByData ($db$, $minsup$)
  for $k = 2$ to $|db|$ : $r_k$ := IdentifyKComponentsByData ($db$, $k$, $minsup$)
  best := $\text{argmin}_{k \in [1,|db|]}$ CalcEncodedSize ($db$, $r_k$, $k$)
  return $r_{\text{best}}$

IdentifyKComponentsByData ($db$, $k$, $minsup$)
  *parts* := Partition (RandomizeOrder($db$), $k$)
  do
    for each $p_i \in parts$ :
      $CT_i$ := KRIMP($p_i$, MineFreqSets ($p_i$, $minsup$))
      $CT_i$ := ApplyLaplaceCorrection ($CT_i$)
   for each transaction $t \in db$ : assign $t$ to $CT_i$ that gives the shortest code
  while *transactions have been swapped*
  return *parts*
```

Table 3 Experimental results for Data-Driven Component Identification

Dataset	Data-Driven Component Identification			
	Compression gain (%)	Component purity (%)	Average DS	Optimal k
Adult	40.3	82.2	31.7	177
Anneal	4.8	76.2	3.7	2
Chess (kr-k)	18.2	17.8	2.9	13
Mammals	46.2	–	1.7	6
Mushroom	14.7	75.6	7.8	20
Nursery	15.0	43.4	3.6	8
PageBlocks	70.3	92.5	19.6	30
Retail	25.5	–	0.10	2

Per database, the gain in compression over regular KRIMP compression, component purity by majority class voting, average dissimilarity between the components and the optimal value of k are shown

the algorithm will be robust despite this initial non-deterministic step. The algorithm is presented in pseudo-code as Algorithm 3.

5.3 Experiments

Each experiment was repeated ten times, each run randomly initialised. The different runs resulted in almost the same output, indicating that random initialisation does not harm robustness. In Table 3 we report the main results of these experiments. The reported results are those of the run with the shortest global encoded length.

For all datasets decompositions are found that allow for much better compression of the whole. The gains show that very homogeneous blocks of data are identified; otherwise the total encodings would only have become longer, as many code tables have to be included instead of one. The components identified in the Adult, Mammals and Pageblocks datasets are so specific that between 40 and 70% fewer bits are required than when the data is compressed as a single block.

In between, however, the components are very heterogeneous. This is shown by the average dissimilarity measurements. For example, the 19.6 measured for Pageblocks

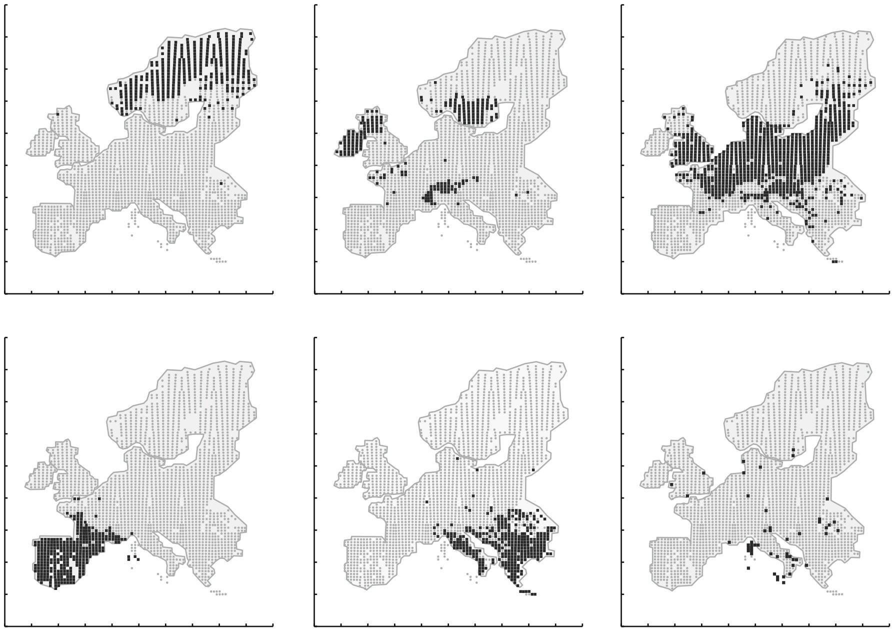

Fig. 2 The components of Mammals ($k = 6$, optimal)

means that on average 1960% more bits would be required for a random transaction, if it were encoded by a 'wrong' component. Also for the other datasets partitions are created that are at least as different as the original classes (see Table 1). Further, the component purities are in the same league as those of the model driven algorithm.

The geographic Mammals dataset allows us to meaningfully visually inspect the found components. Figure 2 shows the best found decomposition. Each of the rows (i.e. grid location of 50 × 50 km) has been assigned to one of the six components based on its items (i.e., the mammals that have been recorded at that location). This is a typical example where normally ad-hoc solutions are used (Heikinheimo et al. 2007) as it is difficult to define a meaningful distance measure. Our method only regards the characteristics of the components, the patterns in which the items occur. As can be seen in Fig. 2, this results in clear continuous and geographically sound groupings—even though the algorithm was completely uninformed in this regard. For example, the top row components cover, respectively, the 'polar' region, highlands and more temperate areas of Europe.

5.4 Discussion

Much shorter data descriptions and high dissimilarities found show that highly specific database components are identified: internally homogeneous and heterogeneous in between. Moreover, the improvements in purity over the baseline show that many components have a strong relation to a particular class. Although mainly a property of the data, the optimal number of components we identify is low, which enables experts to analyse and interpret the components by hand.

By definition, each randomly initialised run of the algorithm may result in a different outcome. In all our experiments, however, we found the outcomes to be stable—indicating the robustness of a data compression approach. Nevertheless, to ensure that one finds a very good solution a couple of runs are required. However, as only partitions of the data have to be compressed, the algorithm runs very fast (a run typically takes seconds to minutes) so this poses no practical problems. For example, for the largest dataset, Retail, it took only 6 h in total to run ten independent runs over all k.

6 Discussion

The components of a database can be identified using both model and data driven compression approaches. The experimental results show that both methods return characteristic partitions with much shorter total encoded size than regular single-component KRIMP compression. The distributions of the components are shown to be very different from each other using dissimilarity measurements. These dissimilarities show that the code tables, and thus the patterns present in the data, are very unalike—i.e., the characteristics of the data components are different.

The optimal number of components is determined automatically by MDL by either method: no parameter k has to be set by hand.

Each of the two proposed component identification methods has its own merit and depending on the data and computational power available one may choose for either the data or model driven algorithm. When dealing with (very) large databases, the data driven method is bound to provide good results quickly. For analysis of reasonable amounts of data the model driven method has the advantage of characterising the components very well with only modest numbers of patterns.

Although we here focus on transaction data and the KRIMP encoding, note that without much effort the framework can be generalised to a generic solution based on MDL: given a data type and suited encoding scheme, components can be identified by minimising the total compressed size. Especially the data driven algorithm is very generic and can be easily applied to other data types or adapted to use different compressors. This is trivial if a compressor can encode single transactions, otherwise one should assign each transaction to the component of which the encoded size of transaction and component is minimal. The model driven algorithm is more specific to our code table approach, but can also be translated to other data types.

7 Related work

Clustering is clearly related to our work, as it addresses part of the problem we consider. The best-known clustering algorithm is k-means (MacQueen 1967), to which our data driven component identifier is related as both iteratively reassign data points to the currently closest component. For this, k-means requires a distance metric, but it is often hard to define one as this requires prior knowledge on what distinguishes the different components. Our method does not require any a priori knowledge. In addition, clustering only aims at finding components, while here we simultaneously model these partitions explicitly with small pattern sets.

Local frequency of items has been used by Wang et al. (1999) to form clusters, avoiding pair-wise comparisons but ignoring higher order statistics. Aggarwal et al. (2002) describe a form of k-means clustering with market basket data, but for this a similarity measure on transactions is required. This measure is based on pair-wise item correlations; more complex (dis)similarities between transactions may be missed. Moreover, many parameters (eight) have to be set manually, including the number of clusters k.

Bi-clustering algorithms (Pensa et al. 2005) look for linked clusters of both objects and attribute-value pairs called bi-clusters. These methods impose more restrictions on the possible clusters; for instance, they do not per se allow for overlap of the attribute-value pairs in the clusters. Further, the number of clusters k often has to be fixed in advance. Koyotürk et al. (2005) regarded item data as a decomposable matrix, of which the approximation vectors were used to build hierarchical cluster trees. However, rather many vectors are required for decomposition. Cadez et al. (2001) do probabilistic modelling of transaction data to characterise (profile) what we would call the components within the data, which they assume to be known beforehand.

Recently, a number of information theoretic approaches to clustering have been proposed. The Information Bottleneck (Tishby et al. 1999) can be used to find clusters by minimising information loss. In particular, LIMBO (Andritsos et al. 2004) focuses on categorical databases. Main differences with our approach are that the number of clusters k has to be specified in advance, and as a lossy compression scheme is used MDL is not applicable.

Entropy measures allow for non-linearly defined clusters to be found in image segmentation tasks (Gokcay and Principe 2002). The MDL principle was used for vector quantization (Bischof et al. 1999), where superfluous vectors were detected via MDL. Böhm et al. (2006) used MDL to optimise a given partitioning by choosing specific models for each of the parts. These model-classes need to be pre-defined, requiring premonition of the component models in the data. Cilibrasi and Vitányi (2005) used pairwise compression to construct hierarchical cluster trees with high accuracy. The method works well for a broad range of data, but performance deteriorates for larger (>40 rows) datasets. Kontkanen et al. (2004) proposed a theoretical framework for data clustering based on MDL which shares the same global code length criterion with our approach, but does not focus on transaction databases.

8 Conclusions

Transaction databases are mixtures of samples from different distributions; identifying the components that characterise the data, without any prior knowledge, is an important problem. We formalise this problem in terms of total compressed size using MDL: the optimal decomposition is that set of code tables and partitioning of the data that minimises the total compressed size. No prior knowledge on the distributions, distance metric or the number of components has to be known or specified.

We presented two approaches to solve the problem and provide parameter-free algorithms for both. The first algorithm provides solutions by finding optimal sets of code tables, while the second does this by finding the optimal data partitioning. Both methods result in significantly improved compression when compared to compression of the database as a whole. Component purity and dissimilarity results confirm our hypothesis that characteristic components can be identified by minimising total compressed size. Visual inspection shows that MDL indeed selects sound groupings.

The approach we present can easily be adopted for other data types and compression schemes. The KRIMP-specific instantiation for categorical data in this paper shows that the MDL principle can be successfully applied to this problem. The data driven method especially is very generic and only requires a compression scheme that approximates the Kolmogorov Complexity of the data.

References

Aggarwal CC, Procopiuc C, Yu PS (2002) Finding localized associations in market basket data. IEEE Trans Knowl Data Eng 14(1):51–62

Andritsos P, Tsaparas P, Miller RJ, Sevcik KC (2004) LIMBO: scalable clustering of categorical data. In: Proceedings of the EDBT'04, pp 124–146

Bischof H, Leonardis A, Sleb A (1999) MDL principle for robust vector quantization. Pattern Anal Appl 2:59–72

Böhm C, Faloutsos C, Pan J-Y, Plant C (2006) Robust information-theoretic clustering. In: Proceedings of the KDD'06, pp 65–75

Brijs T, Swinnen G, Vanhoof K, Wets G (1999) The use of association rules for product assortment decisions: a case study. In: Proceedings of the KDD'99, pp 254–260

Cadez IV, Smyth P, Mannila H (2001) Probabilistic modeling of transaction data with applications to profiling, visualization, and prediction. In: Proceedings of the KDD'01, pp 37–46

Cilibrasi R, Vitányi PMB (2005) Clustering by compression. IEEE Trans Inf Theory 51(4):1523–1545

Coenen F (2003) The LUCS-KDD discretised/normalised ARM and CARM data library. http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html. Accessed 30 June 2009

Faloutsos C, Megalooikonomou V (2007) On data mining, compression, and kolomogorov complexity. Data Min Knowl Discov 15(1):3–20

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comp Sys Sci 55(1):119–139

Gokcay E, Principe JC (2002) Information theoretic clustering. IEEE Trans Pattern Anal Mach Intell 24(2):158–171

Grünwald PD (2005) Minimum description length tutorial. In: Grünwald PD, Myung IJ, Pitt MA (eds) Advances in minimum description length. MIT Press, Cambridge

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the SIGMOD'00, pp 1–12

Heikinheimo H, Fortelius M, Eronen J, Mannila H (2007) Biogeography of European land mammals shows environmentally distinct and spatial coherent clusters. J Biogeogr 34(6):1053–1064

Kontkanen P, Myllymäki P, Buntine W, Rissanen J, Tirri H (2004) An MDL framework for clustering. Technical Report 2004–6, HIIT

Koyotürk M, Grama A, Ramakrishnan N (2005) Compression, clustering, and pattern discovery in very high-dimensional discrete-attribute data sets. IEEE Trans Knowl Data Eng 17(4):447–461

Li M, Vitányi PMB (1993) An introduction to Kolmogorov complexity and its applications. Springer, New York

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the symposium on mathematical statistics and probability, pp 281–297

Mitchell-Jones AJ, Amori G, Bogdanowicz W, Krystufek B, Reijnders PJH, Spitzenberger F, Stubbe M, Thissen JBM, Vohralik V, Zima J (1999) The atlas of European mammals. Academic Press, New York

Pensa R, Robardet C, Boulicaut J-F (2005) A bi-clustering framework for categorical data. In: Proceedings of the PKDD'05, pp 643–650

Siebes A, Vreeken J, van Leeuwen M (2006) Item sets that compress. In: Proceedings of the SDM'06, pp 393–404

Titterington D, Smith A, Makov U (1985) Statistical analysis of finite mixture distributions. Wiley, New York

Tishby N, Pereira FC, Bialek W (1999) The information bottleneck method. In: Proceedings of the allerton conference on communication, control and computing, pp 368–377

van Leeuwen M, Vreeken J, Siebes A (2006) Compression picks item sets that matter, In: Proceedings of the ECML/PKDD'06, pp 585–592

Vreeken J, van Leeuwen M, Siebes A (2007) Characterising the difference. In: Proceedings of the KDD'07, pp 765–774

Wang K, Xu C, Liu B (1999) Clustering transactions using large items. In: Proceedings of the CIKM'99, pp 483–490