Krimp Minimisation for Missing Data Estimation

Jilles Vreeken & Arno Siebes

Department of Information and Computing Sciences
Utrecht University
Technical Report UU-CS-2008-034
www.cs.uu.nl
ISSN: 0924-3275

KRIMP Minimisation for Missing Data Estimation

Jilles Vreeken and Arno Siebes
Department of Computer Science
Universiteit Utrecht
{jillesv,arno}@cs.uu.nl

Abstract

Many data sets are incomplete. For correct analysis of such data, one can either use algorithms that are designed to handle missing data or use imputation. Imputation has the benefit that it allows for any type of data analysis. Obviously, this can only lead to proper conclusions if the provided data completion is both highly accurate and maintains all statistics of the original data.[1]

In this paper, we present three data completion methods that are built on the MDL-based KRIMP *algorithm. Here, we also follow the MDL principle, i.e. the completed database that can be compressed best, is the best completion because it adheres best to the patterns in the data.*

By using local patterns, as opposed to a global model, KRIMP *captures the structure of the data in detail. Experiments show that both in terms of accuracy and expected differences of any marginal, better data reconstructions are provided than the state of the art, Structural EM.*

1 Introduction

Many data sets are incomplete. Whether dealing with surveys, DNA micro arrays or medical data, missing values are commonplace. However, properly dealing with missing values remains an open problem in data analysis. While some specialised algorithms exist that are designed to handle missing data, many are not, and can at most ignore missing values. From statistics, we know [14] this leads to biases in the outcome of the analysis.

There are two ways to properly analyse incomplete data:

- Using specialised algorithms designed to handle missing data
- Completing the data by imputation

Of these two, imputation has the practical advantage that one can analyse the completed database using any tool or method desired. Obviously, this does require the imputation to be as accurate as possible. That is, all statistics that one computes from the completed database should be as close as possible to those of the original data. The problem of imputation is thus: complete the database as well as possible.

However, determining what is 'good' cannot just be measured through accuracy: only for 100% correct estimations we know for sure that all statistics are maintained. In this paper we consider 0/1 databases in particular and categorical databases in general. This allows us to properly validate the quality of a completed database in the following manner: we compare the support of a random item set in the original (complete) database with its support in the completed database. The rationale is as follows. Analysing binary data is largely based on counting. If the support of all item sets are correct, all counts will be correct. So, if for random item sets that difference in support is nil, we know that all counts are identical. Consequently, all statistics computed on the completed database will be correct.

To achieve such high quality imputation we use the practical variant of Kolmogorov complexity, MDL (minimum description length), as our guiding principle: the completed database that can be compressed best is the best completion. The driving thought behind this approach is that a completion should comply to the local patterns in the database: not just filling in what globally would lead to the highest accuracy. By taking into account how specific values co-occur locally, not only the global statistics on the data will be correct but also those measured on the local level.

We approximate this best result using KM, which stands for KRIMP Minimisation. It is an iterative approach in which each successive completion has a lower complexity as measured through the compressibility of the data. KM is built on the MDL-based KRIMP algorithm, that provides high quality data descriptions through compression of the data using frequent item sets [21]. The code tables that form the outcome of KRIMP have been successfully used in classification [16], characterising differences

[1] This is an extended version of work published at IEEE International Conference on Data Mining (ICDM'08) [22]

between databases [23] and generating data virtually indiscernible from the original [24].

Most good algorithms for missing data first estimate a model on the data. Structural EM [11] is a good example of this. Within the iterative EM process the learning of a Bayes net is integrated. This leads to very good approximations of the networks underlying the data, as well as state of the art imputations.

KM is different in that it looks at the local patterns in the data, rather than a building global model; such local patterns are often smoothed out from a global stance. The experiments show that the local approach is indeed superior to the global approach, both in terms of accuracy and quality of the completed databases.

2 The Problem

2.1 Preliminaries

Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of binary (0/1 valued) attributes. That is, the domain D_i of item I_i is $\{0, 1\}$. A transaction (or tuple) over $\mathcal{I}$ is an element of $\prod_{i \in \{1,\ldots,n\}} D_i$. A database $\mathcal{D}$ over $\mathcal{I}$ is a bag of tuples over $\mathcal{I}$. This bag is indexed in the sense that we can talk about the i-th transaction.

An item set J is, as usual, a subset of $\mathcal{I}$, i.e., $J \subseteq \mathcal{I}$. The item set J *occurs* in a transaction $t \in \mathcal{D}$ if $\forall I \in J : \pi_I(t) = 1$. The *support* of item set J in database $\mathcal{D}$ is the number of transactions in $\mathcal{D}$ in which J occurs. That is, $supp_{\mathcal{D}}(J) = |\{t \in \mathcal{D} |\ J \text{ occurs in } t\}|$. An item set is called *frequent* if its support is larger than some user-defined threshold called the *minimal support* or *min-sup*. Given the A Priori property,

$$\forall I, J \in \mathcal{P}(\mathcal{I}) : I \subset J \rightarrow supp_{\mathcal{D}}(J) \leq supp_{\mathcal{D}}(I)$$

frequent item sets can be mined efficiently levelwise, see [1] for more details.

Note that while we restrict ourself to binary databases in the description of our problem and algorithms, there is a trivial generalisation to categorical databases. In the experiments, we use such categorical databases.

2.2 Missing Data

A database $\mathcal{D}$ has *missing data,* if some of its values are denoted by '?'. The ?-values denote that we do not know what the actual value is, it might be a 0 or a 1. In the traditional market-basket example, this means, e.g., that we do not know whether or not *Beer* was bought in a given transaction.

The literature, see, e.g., [17, 20], distinguishes the following three different types of missing data mechanisms.

MCAR which stands for *missing completely at random*. It means that the fact that a data value is missing does not depend on any of the values in the transaction, including itself.

MAR which stands for *missing at random.* This means that the fact that a data value is missing may depend on one or more of the observed values, it does *not* depend on the "true" value of any of the missing values.

NMAR which stands for *not missing at random.* This means that the fact that a data value is missing may depend on the true value of a missing data value.

NMAR is a very problematic case. Without background knowledge, unbiased analysis of the data is impossible. As the vast majority of the literature, we restrict ourselves to MAR and MCAR only.

2.3 Database Completion

Completing a database simply means that each question mark is replaced by a definite value, i.e., a 1 or a 0. More formally we have the following definition.

Definition 1. *Let $\mathcal{D}$ and $\mathcal{D}_c$ be two databases over $\mathcal{I}$. Moreover, let $\mathcal{D}$ have missing data, whereas $\mathcal{D}_c$ is* complete*, i.e., $\mathcal{D}_c$ has no missing data. Then, $\mathcal{D}_c$ is a* completion *of $\mathcal{D}$ iff*

1. *$\mathcal{D}$ and $\mathcal{D}_c$ both have k transactions, $\mathcal{D} = \{t_1, \ldots, t_k\}$ and $\mathcal{D}_c = \{s_1, \ldots s_k\}$;*
2. *$\forall i \in \{1, \ldots, k\}\ \forall I \in \mathcal{I} : \pi_I(t_i) \in \{0, 1\} \rightarrow \pi_I(t_i) = \pi_I(s_i)$*

An algorithm $\mathcal{A}$ that completes any incomplete database is called a completion algorithm*.*

There are many possible completions of an incomplete database. In fact, if $\mathcal{D}$ has k unknown values, there are 2^k completions. Clearly, not all completions are equally useful. To define quality measures, we assume that we know the true complete database, denoted by $\mathcal{D}^t$. The most obvious quality measure of a completion is accuracy.

Definition 2. *Let $\mathcal{D}, \mathcal{D}^t$, and $\mathcal{D}_c$ be databases over $\mathcal{I}$, such that $\mathcal{D}$ is incomplete, $\mathcal{D}^t$ is the true completion of $\mathcal{D}$ and $\mathcal{D}_c$ is an arbitrary completion of $\mathcal{D}$. Moreover, let m be the number of missing values in $\mathcal{D}$ and n the number of missing data values in $\mathcal{D}$ on which $\mathcal{D}^t$ and $\mathcal{D}_c$ agree. The accuracy of $\mathcal{D}_c$ is given by*

$$acc(\mathcal{D}_c) = \frac{n}{m}$$

Clearly, a 100% accurate completion of $\mathcal{D}$ will allow for unbiased estimates on $\mathcal{D}_c$. However, accuracy is a very strict measure. If $\mathcal{D}_c$ is simply a permutation of the rows

of $\mathcal{D}^t$, the accuracy can be arbitrarily low. Whereas such a permutation still allows for unbiased estimates! Still, accuracy is the most generally used quality measure for data completion [17].

Alternatively, we could define accuracy upto permutations. However, this would yield its computation rather hard. It would require the search for a permutation that yields maximal accuracy (in the strict sense as defined above).

To define a less strict quality measure, recall that the goal of a completion is to allow unbiased statistics. That is, statistics or models computed on $\mathcal{D}_c$ should be as close as possible to their counterparts computed on $\mathcal{D}^t$. Most statistical analysis of categorical data depends crucially on counts and sums. Often subtables are created using selections and projections, and counts and sums on these subtables are computed.

In the case of 0/1 data, selections correspond to item sets; in fact we have the following simple result.

Theorem. *Let $\mathcal{D}$ be a complete database over $\mathcal{I}$, let $\mathcal{J}, \mathcal{K} \subseteq \mathcal{I}$, with $\mathcal{J} \cap \mathcal{K} = \emptyset$, and let $I \in \mathcal{I} \setminus (\mathcal{J} \cup \mathcal{K})$. The number of 1's I has in the subtable created by the selection*

$$\bigwedge_{J \in \mathcal{J}} J = 1 \ \wedge \ \bigwedge_{K \in \mathcal{K}} K = 0$$

on $\mathcal{D}$, is given by

$$supp_{\mathcal{D}}(\mathcal{J} \cup \{I\}) - supp_{\mathcal{D}}(\mathcal{J} \cup \mathcal{K} \cup \{I\})$$

Similarly, the number of 0's for I is given by

$$supp_{\mathcal{D}}(\mathcal{J}) - supp_{\mathcal{D}}(\mathcal{J} \cup \mathcal{K} \cup \{I\})$$

Proof. A transaction t satisfies the selection if it has 1's for all elements of $\mathcal{J}$ and 0's for all elements of $\mathcal{K}$. In other words, it should be in the support of $\mathcal{J}$, but not in the support of $\mathcal{K}$. □

Given that sums are counts on 1's on 0/1 databases and that the above theorem is invariant under suitable projections, we have the following corollary.

Corollary 1. *Let $\mathcal{D}, \mathcal{D}^t$, and $\mathcal{D}_c$ be databases over $\mathcal{I}$, such that $\mathcal{D}$ is incomplete, $\mathcal{D}^t$ is the true completion of $\mathcal{D}$ and $\mathcal{D}_c$ is an arbitrary completion of $\mathcal{D}$. If for all item sets $J \subseteq \mathcal{I}$,*

$$supp_{\mathcal{D}_c}(J) = supp_{\mathcal{D}^t}(J)$$

then all counts and sums on project-select subtables on $\mathcal{D}_c$ equal their counterpart on $\mathcal{D}^t$.

With this result in mind, our new quality measure, we can measure how good the support of item sets in a completed database is.

Definition 3. *Let $\mathcal{D}, \mathcal{D}^t$, and $\mathcal{D}_c$ be databases over $\mathcal{I}$, such that $\mathcal{D}$ is incomplete, $\mathcal{D}^t$ is the true completion of $\mathcal{D}$ and $\mathcal{D}_c$ is an arbitrary completion of $\mathcal{D}$. Moreover, let $\epsilon, \delta \in \mathbb{R}$. $\mathcal{D}_c$ is* (ϵ, δ)-correct *if for a random (frequent) item set I*

$$P(|supp_{\mathcal{D}^t}(I) - supp_{\mathcal{D}_c}(I)| > \epsilon) \leq \delta$$

In other words, the support of item sets on (ϵ, δ)-correct completions are almost always close to the correct value. The lower ϵ and δ are, the better the completion is. As an aside, note that (ϵ, δ)-correctness is invariant under permutations; the sum is a commutative operator.

There is a simple relation between accuracy and (ϵ, δ)-correctness, as given by the following theorem.

Theorem. *Let $\mathcal{D}, \mathcal{D}^t$, and $\mathcal{D}_c$ be databases over $\mathcal{I}$, such that $\mathcal{D}$ is incomplete, $\mathcal{D}^t$ is the true completion of $\mathcal{D}$ and $\mathcal{D}_c$ is an arbitrary completion of $\mathcal{D}$. If $\mathcal{D}_c$ is $(0,0)$-correct, then there exists a permutation σ of the rows of $\mathcal{D}_c$ such that $\sigma(\mathcal{D}_c) = \mathcal{D}^t$*

Proof. Let ψ be a maximal injective partial function $\psi : \mathcal{D}^t \to \mathcal{D}_c$ such that $\psi(t) = t$. Moreover, let $s \in \mathcal{D}_c \setminus \psi(\mathcal{D}^t)$. Let the pair $(\mathcal{J}, \mathcal{K})$ be the partition of $\mathcal{I}$, such that s has value 1 for all items in $\mathcal{J}$ and value 0 for all items in $\mathcal{K}$. This means that s is in the support of $\mathcal{J}$ minus the support of $\mathcal{K}$. Since the support of all item sets are equal on $\mathcal{D}_c$ and $\mathcal{D}^t$, this means that we can extend ψ to have s in its image. This contradicts maximality and, hence, $\mathcal{D}_c \setminus \psi(\mathcal{D}^t) = \emptyset$. Thus ψ is a bijection between $\mathcal{D}$ and $\mathcal{D}_c$. □

Clearly, if a completion is 100% accurate, it is also $(0,0)$-correct. If $(0,0)$-correctness is not attainable, the two measures differ. Due to the invariance of (ϵ, δ)-correctness, it is a more flexible quality measure.

(ϵ, δ)-correctness is defined for a random item set, whatever the support of an item set. Many of these item sets will have a very low support, in fact, the vast majority will have support 0. For statistical analysis, however, item sets with a very low support are not very interesting. Statistics computed on small data sets, including the frequency of an item set(!), are not very stable. Small pertubations to the database may cause large changes of these statistics. Hence, for the purposes of subsequent statistical analysis it is better to have a high (ϵ, δ)-correctness considering *frequent* item sets only, rather than having a high (ϵ, δ)-correctness considering all item sets.

All frequent item sets is, unfortunately, still a large space to sample from. It is well-known that the set of all closed frequent item sets is often far smaller than the set of all frequent item sets. Moreover, for any frequent item set I, there is a closed frequent item set J such that the support of I equals the support of J. Hence, if we know that for the closed frequent item sets $supp_{\mathcal{D}^t}(J) \approx supp_{\mathcal{D}_c}(J)$, then this also holds for the frequent item sets.

Given these observations, we sample the closed frequent item sets to estimate (ϵ, δ)-correctness in our experiments.

2.4 The Completion Problem

With these quality measures at hand we formalise our completion problem as follows.

The Completion Problem:

Devise a completion algorithm $\mathcal{A}$ that yields an (ϵ, δ)-correct completion for any incomplete database with ϵ and δ as low as possible.

We settle for *as low as possible* because there may not be enough information in the database to derive the $(0, 0)$-correct completion. For example, if $\mathcal{I}$ has only one item, the database has only one transaction and its value is missing! Either $\{1\}$ and $\{0\}$ are possible, and there is no algorithm that will reliably choose correctly. For all practical purposes, though, $(0, 0)$ is well approximable.

3 The KRIMP Algorithm, a brief introduction

In this work, the KRIMP algorithm plays an important role. This MDL based algorithm for item set mining was recently introduced by Siebes et al. [21], although not yet by that name. For the convenience of the reader we provide a brief introduction to this algorithm here.

The MDL principle [13] 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.

The key idea of this 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 the coding algorithm needs to be introduced. 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 it is insisted 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 $\mathcal{D}$. The frequency of an item set $c \in CT$ is the number of transactions $t \in \mathcal{D}$ which have c in their cover. The relative frequency of $c \in CT$ is the probability that c is used to encode an arbitrary $t \in \mathcal{D}$. For optimal compression of $\mathcal{D}$, the higher P(c), the shorter its code should be. In fact, from information theory [8], we have the Shannon code length for c, which is optimal, as:

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

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 $t \in \mathcal{D}$ 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:

$$\begin{aligned} L_{CT}(\mathcal{D}) &= \sum_{t \in \mathcal{D}} L_{CT}(t) \\ &= -\sum_{c \in CT} freq(c) \log\left(\frac{freq(c)}{\sum_{d \in CT} freq(d)}\right) \end{aligned}$$

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)*, of which we use the codes to compute the size of the item sets in the left-hand side column. Hence, the size of the code table is given by:

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

Siebes et al. [21] defines the optimal set of (frequent) item sets as that one whose associated code table minimises the total compressed size:

$$L(CT) + L_{CT}(\mathcal{D})$$

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, all other elements of CT are reconsidered to see if they still positively contribute to compression. For more details see [21].

4 Completion Algorithms

Another way to paraphrase the MDL principle is: the better a model compresses the database, the closer it approximates the underlying data distribution. The key point of both MAR and MCAR is that they do not perturb this underlying distribution. Hence, in the spirit of MDL, one can say: the best completion is the completion that allows for the best compression.

A straightforward implementation of this idea, however, runs the risk of surpressing the natural variation in the data. The more data that is missing, the higher this risk becomes. How susceptible an algorithm is to this risk can be analysed by estimating (ϵ, δ)-correctness. The more susceptible an algorithm is, the worse the (ϵ, δ)-correctness will be.

Next, note that for $\mathcal{D}$ with n missing binary values the search space consists of 2^n possible completions. Clearly, finding the best completion quickly becomes infeasible, even for moderate amounts of missing values. Further, there is no structure that we can exploit to prune this search space. Therefore, we settle for heuristics that approximate the best compressable $\mathcal{D}_c$ by making local decisions.

We introduce three such completion algorithms, based on KRIMP, in this section. For the first two algorithms, Simple Completion and sc Krimp Completion, we assume that there is enough complete data to allow sc Krimp to discover good code tables. Moreover, KRIMP Completion uses randomisation to minimise the risk of variation surpression. The third algorithm, called KRIMP Minimisation, does no longer rely on the assumption that there is enough complete data.

4.1 Simple Completion

A simple way to impute a missing value is using the maximal likelihood estimator. For KRIMP this reduces to shortest encoded length. Let $t \in \mathcal{D}$ be a transaction with missing values and denote by $C(t)$ the set of all its possible completions. The *Simple Completion* algorithm SC replaces t by that element of $C(t)$ that has the shortest encoded length.

More precisely, let $\mathcal{D} = \mathcal{D}_{comp} \cup \mathcal{D}_{inc}$ such that all transactions in $\mathcal{D}_{comp}$ are complete, while all transactions in $\mathcal{D}_{inc}$ are incomplete. The SC algorithm, Fig. 1, first computes a code table CT, by running KRIMP on $\mathcal{D}_{comp}$. Next, each incomplete transaction by its shortest completion.

SC($\mathcal{D}$)
1 CT := KRIMP($\mathcal{D}_{comp}$)
2 **foreach** $t \in \mathcal{D}_{inc}$
  $t := \text{argmin}_{s \in C(t)} L_{CT}(s)$
3 **return** $\mathcal{D}_{comp} \cup \mathcal{D}_{inc}$

Figure 1. The SC Algorithm

4.2 KRIMP Completion

By always choosing the most likely candidate, the SC algorithm may have a detrimental effect on the support of some item sets. Suppose, e.g., that the encoded length of $(1, 1)$ is slightly shorter than $(1, 0)$. Then each occurrence of $(1, ?)$ will be replaced by $(1, 1)$ by SC. This leads to an overestimate of the support of $(1, 1)$ and an underestimate of the support of $(1, 0)$. The KRIMP *Completion* algorithm KC, see Fig. 2, remedies this by choosing an element of $C(t)$ with a chance proportional to its encoded length. More precisely, we again assume that $\mathcal{D} = \mathcal{D}_{comp} \cup \mathcal{D}_{inc}$ as before. Again KRIMP is first run on $\mathcal{D}_{comp}$. The resulting code table CT defines a probability distribution $P_{CT}(t)$ on $C(t)$ given by:

$$P_{CT}(t)(s) = \frac{2^{-L_{CT}(s)}}{\sum_{u \in C(t)} 2^{-L_{CT}(u)}}$$

The completion of t is chosen from $C(t)$ according to this distribution. The function CHOICE($C(t), CT$) makes this random choice.

KC($\mathcal{D}$)
1 CT := KRIMP($\mathcal{D}_{comp}$)
2 **foreach** $t \in \mathcal{D}_{inc}$
  $t :=$ CHOICE($C(t), CT$)
3 **return** $\mathcal{D}_{comp} \cup \mathcal{D}_{inc}$

Figure 2. The KC Algorithm

4.3 KRIMP Minimisation

For KC to work, we need enough complete data for KRIMP to compute a good code table. If there is not enough complete data, the result of KC may be arbitrarily bad. To handle such a lack of sufficient complete data, we take an EM-like [9] approach.

KM($\mathcal{D}$)
1 $\mathcal{D}_c :=$ random completion of $\mathcal{D}$
2 **while** not converged
  $CT :=$ KRIMP($\mathcal{D}_c$)
  $\mathcal{D}_c :=$ KC($\mathcal{D}$)
3 **return** $\mathcal{D}_c$

Figure 3. The KM Algorithm

The KRIMP *Minimisation* algorithm KM starts with a random completion of the incomplete database $\mathcal{D}$. Then it iterates through a number of KRIMP and KC steps. In the KRIMP step it compresses the current *complete* database. In the KC step it completes the *incomplete* database $\mathcal{D}$ using the code table computed in the KRIMP step. This is continued as long as the total encoded length of the completed database shrinks. The algorithm returns the final completed database. It has the shortest encoded length of the considered completions, hence the name of the algorithm.

Note that KM will always terminate. The total encoded size shrinks with every step. Since the encoded size of any finite database is finite, KM can only execute a finite number of iterations.

5 Related Work

Imputation has a long history. One of the first known examples, Hot Deck imputation [2], was employed by the US census bureau in the fifties. It replaces missing records by random draws from complete records from the same local area. As such, it may be regarded as a crude form of k nearest-neighbour imputation [25]. Since, more advanced systems for editing survey data have been developed, in particular for hierarchical demographic data. Examples include GEIS and SPEER [15] for continuous and DISCRETE [6] and CANCEIS [3] for discrete survey data. These systems all rely on nearest-neighbour algorithms for imputation [4]. As such, they require a distance function on the data, unlike parameter-free methods.

Regression, mean substitution and mean-mode [14] imputation have a greedy nature that harms the variance in the completed data [2]. Using some randomness circumvents this, which is why both Multiple Imputation (MI) [19] and Expectation Maximisation (EM) [9] are still the current state of the art.

To start with the latter, imputation through EM is the process of maximising the likelihood of the data given a distribution. Iteratively, it adapts the model to the data and re-imputes it. EM has been shown to provide very accurate probability estimations. Its model, however, has obviously to be chosen according to the data. For categorical data the log-linear model may be used. Still, by its exponential size in the number of attributes, this is only feasible for datasets with only few variables [20]. KRIMP Minimisation follows a similar iterative approach. However, it optimises the compressed size of the database, not its likelihood.

Integrating a structure learner into the EM process leads to even better results. Structural EM (SEM) [10, 11] learns Bayes nets during the modeling phases. SEM has been shown to provide high quality probability estimates, and very good approximations of the original Bayes nets. However, it is computationally expensive and thereby only feasible for moderately sized datasets. A stark difference to our approach is that SEM learns a global Bayes network on the data, whereas KRIMP considers the data in more detail by using local patterns.

Multiple imputation [19] states that the data should be imputed multiple times, thus providing different datasets. It does not dictate which data completion algorithm should be used, though typically either by sampling from predefined distributions [5] or by applying EM. The resulting datasets need to be analysed individually, after which the results are aggregated. For many data mining approaches this is non-trivial.

6 Experiments

In this section we empirically evaluate the performance of the three proposed data completion methods. All results of the experiments in this section are averaged over 10 independent runs, unless indicated otherwise. Further, in these experiments we consider the case of 1 missing value per transaction, on average. Again, unless indicated otherwise. The (ϵ, δ)–correctness is calculated over the closed frequent item set collection as mined on the complete test data.

6.1 Datasets

We use a range of data sets to validate our methods. From the widely used UCI repository [7] we took seven databases. Further, for fair comparison to the Bayesian Structural EM method, we generated data from the well-known *alarm* network. This data was generated using the *GenInstance* program, made available by Nir Friedman in the LibB Bayes Network tool library [12].

Table 1. Statistics of the data sets used in the experiments.

Dataset	$\#attributes$	$\|\mathcal{I}\|$	$\|\mathcal{D}\|$	$acc.random$	$acc.baseline$	% min-sup
alarm	37	105	5000	37.8%	79.9%	50.0
chess (kr vs k)	7	58	28056	16.5%	23.0%	0.7
led7	8	24	3200	50.0%	61.2%	0.03
let.Recog	17	102	20000	20.2%	57.5%	1.2
mushroom	23	119	8124	28.8%	57.6%	1.2
penDigits	17	86	10992	21.8%	35.7%	0.9
tic–tac–toe	10	29	958	33.2%	45.2%	0.1
wine	14	68	178	20.5%	43.5%	0.6

Table 2. Imputation quality measurements. Missing values estimated using SIMPLE COMPLETION (SC) and KRIMP COMPLETION (KC), trained on complete data, using 10-fold cross-validation. For the (ϵ, δ)-measurements, ϵ was fixed, calculating δ per imputed database. All values relative (%).

			MCAR						*MAR*			
			baseline		SC		KC		SC		KC	
Dataset	*% miss*	ϵ	δ	*acc.*	δ	*acc.*	δ	*acc.*	δ	*acc.*	δ	*acc.*
alarm	2.7	0.1	39.1	79.9	4.2	84.0	5.5	80.0	6.9	85.1	5.6	81.2
chess	14.3	0.4	50.2	23.0	1.1	28.3	0.5	25.4	1.4	28.2	0.3	25.8
led7	12.5	0.2	43.0	61.2	3.1	85.9	2.8	80.8	3.5	87.3	3.0	82.2
let.Recog	5.9	0.1	26.0	57.6	0.2	70.1	0.0	67.1	0.4	71.4	0.0	68.6
mushroom	4.3	0.5	30.3	57.6	0.0	76.9	0.0	74.7	0.3	77.1	0.5	76.3
penDigits	5.9	0.04	82.1	35.7	5.3	70.0	5.4	67.3	6.5	68.7	6.6	65.6
tic–tac–toe	10.0	0.5	10.0	45.2	0.0	84.9	0.0	82.8	0.0	84.8	0.0	82.3
wine	7.1	1.1	6.0	43.5	3.0	53.0	3.0	51.9	3.0	53.0	3.0	49.7

The details for these data sets are depicted in Table 1. Apart from their base statistics, we provide the imputation accuracies on MCAR data as achieved by 1) choosing a random possible value and 2) the baseline estimator that chooses the most frequent of the possible values.

We use the closed frequent pattern set as candidates for KRIMP. Being generated from a Bayes net, the *alarm* dataset contains no local structure. The absence hereof leads to a gigantic explosion in the number of patterns; hence we have to use a high min–sup for this database.

6.2 Creating Missing Values

We consider two types of missing data: missing completely at random (MCAR) and missing at random (MAR).

To create MCAR test data, we start with complete data. From it, for as many missing values we want to create, transactions are independently uniformly sampled. From each, one value is uniformly chosen and removed.

For the MAR case, we use the class labels and the three-valued *bp* variable for *alarm* as the depending attributes. For each of their values, a probability table was generated to indicate the chance of the other attributes being missing. We sample over this table to remove values, while still choosing the target transactions uniformly.

6.3 Sufficient Complete Data

First, we study how our methods perform provided with complete training data. Here, we thus consider Simple Completion and KRIMP Completion. This experiment was set up using 10-fold cross validation. Leaving the training-data undamaged, we created missing values in the test–folds.

The results of this experiment are presented in Table 2. The accuracy scores of both our methods are much better than that of the baseline methods (scores for the random chooser in Table 1). This is even more the case for correctness: for all these databases the random estimator scores a 100% chance of finding a marginal differing more than ϵ. Choosing the most frequent value is a better strategy, but still scores up to 50% chance of finding incorrect counts. Both SC and KC, however, do get very close to the optimal (0,0)–score.

Comparing between our two methods directly, we see that the greedy SC acquires better accuracy scores, as expected. However, this comes at a surprisingly slight cost on the (ϵ, δ)–correctness. Given sufficient complete training data even the greedy method maintains the proper local variance of the data. Both the SC and KC imputed data can be regarded statistically identical to the original. This goes

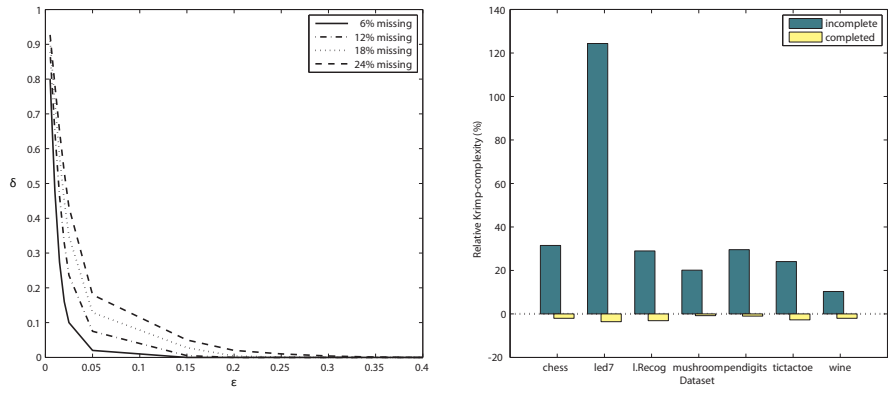

Figure 4. (left) (ϵ, δ)-correctness of SIMPLE COMPLETION on the Letter Recognition dataset. 10 fold cross-validated, trained on complete data. One missing value missing on average per transaction equals 5.9% missing data, 2 values 11.8%, 3 values 17.6% and 4 values 23.5%. ϵ in % (!). (right) Relative difference in KRIMP complexity for incomplete and KM-completed data.

for both MCAR and MAR, for which no strong differences in performance were found.

These experiments further show that there is no strong degradation of performance for increasing amounts of missing data. To show this in more detail, in the left hand side plot in Figure 4 we show a plot of the (ϵ, δ)-correctness of Simple Completion of the *let.Recog* dataset for 6 to 24% missing data. While the related accuracy scores only drop marginally (from 70.1 to 69.1!), the attained (ϵ, δ)-correctness is even more impressive. Even with almost a quarter of the data missing, SC still achieves (0.003, 0.01)-correctness.

6.4 Insufficient Complete Data

Second, we consider the problem for when no (sufficient) complete training data is available. We therefore now employ KRIMP Minimisation instead of KRIMP Completion.

The results of this set of experiments is presented in Table 3. If we first look at the accuracy measurements, we notice that the imputation accuracies are actually often higher than we saw just before. Through the (ϵ, δ)–correctness we can see that no magic is going on, as for all datasets these scores have actually decreased. Thus, through the incompleteness of the training data both methods are hindered in grasping the true data distribution.

Further, we see that the greedy nature of SC expectedly leads to a decrease in the data quality. For all datasets the accuracy provided by KM are only slightly lower than SC. This small loss in accuracy is compensated for by a strong gain in (ϵ, δ)–correctness. For KM, these scores are up to an order better than SC and often approximate the scores attained on the complete training data.

The right hand plot of Figure 4 shows further proof of the data reconstruction ability of KM. To compress the data with missing values, KRIMP typically requires 30% more bits than it does to encode the original data. Clearly, the missing values refrain it from encoding using the most appropriate patterns. However, through iterative imputation, KM is able to approximate the KRIMP complexity of the original data within a single percent. As noise is canceled, the KM-imputed data has slightly lower complexity than the unseen original.

These experiments also showed that KM rapidly converges to this approximate original complexity: only three iterations were required for six of the datasets, two more for the data on *mushroom* edibility. Further, we noticed that while KM is nondeterministic in initialisation and imputation, the resulting accuracies, correctness and data complexities are all virtually equal.

6.5 Comparing to SEM

Now that we have verified that KM provides good data completions, we will compare it to the state of the art in imputation: Bayes Structural EM (SEM) [11]. Structural EM incorporates the learning of a Bayes net on within the EM [9] process. Iteratively it is learned and used to re-impute the data, until the process converges.

In order to learn Bayes Nets on incomplete training data, we used Friedman's publicly available implementation of Structural EM [12]. Its search process can be initialised in various ways, here we used initialisation with random trees.

Table 3. Imputation quality measurements. Missing values estimated using SIMPLE COMPLETION (SC) and KRIMP MINIMISATION (KM), trained on incomplete data. For the (ϵ, δ)-measurements, ϵ was fixed, calculating δ per imputed database. All values relative (%).

			MCAR				*MAR*			
			SC		KM		SC		KM	
Dataset	*% miss*	ϵ	δ	*accuracy*	δ	*accuracy*	δ	*accuracy*	δ	*accuracy*
alarm	2.7	0.7	16.4	84.0	3.0	82.5	17.3	85.7	5.4	83.6
chess	14.3	0.1	15.8	35.4	3.6	33.7	18.8	32.7	4.2	28.2
led7	12.5	1.0	32.7	72.7	1.2	79.2	17.2	82.8	1.2	81.1
let.Recog	5.9	0.8	16.4	64.2	4.8	61.9	14.2	67.1	5.9	65.3
mushroom	4.3	0.5	4.4	76.3	4.1	74.2	4.1	74.5	0.3	70.9
penDigits	5.9	0.1	8.8	66.6	3.8	67.2	11.8	64.6	5.3	65.7
tic–tac–toe	10.0	0.5	3.6	50.5	2.7	46.4	5.2	47.6	3.7	42.3
wine	7.1	1.1	5.3	55.2	4.9	54.6	6.9	50.3	5.1	48.8

Table 4. Imputation quality measurements for MCAR and MAR test data. Missing values estimated using KRIMP MINIMISATION (KM) and Structural EM (SEM), trained on incomplete data. For the (ϵ, δ)-measurements, ϵ was fixed, calculating δ per imputed database. All values relative (%).

			MCAR				*MAR*			
			KM		*SEM*		KM		*SEM*	
Dataset	*% miss*	ϵ	δ	*accuracy*	δ	*accuracy*	δ	*accuracy*	δ	*accuracy*
alarm	2.7	0.5	6.7	82.5	15.2	80.9	8.0	83.6	37.8	82.5
chess	14.3	0.1	4.1	33.7	24.5	23.5	4.2	28.2	33.2	22.7
led7	12.5	0.5	0.9	79.2	2.6	76.1	0.8	81.1	2.3	79.1
tictactoe	10.0	1.0	0.2	46.4	3.1	36.3	0.4	42.3	3.9	41.4
wine	7.1	1.6	4.9	54.6	7.9	46.1	1.1	48.8	12.8	27.5

Other settings were explored, but no significant differences in performance were found. For the actual inference on these Bayes nets for the missing values, we used the Bayes Network Toolbox for Matlab (BNT) [18]. As both learning the Bayes nets and inference are computationally expensive, we do not consider all datasets in this comparison.

The results of this experiment are presented in Table 4. From it, we first notice that KM attains higher imputation accuracies than SEM for three out of the five datasets. However, the general quality of the KM imputed databases, as measured through the (ϵ, δ)–correctness scores, is quite dramatically better than the Bayes net driven SEM approach. This shows that the detail provided by the local-pattern based KRIMP code tables allow for imputation that adheres much better to the local statistics of the original data than possible from a global model. Even for the *alarm* dataset (with relatively few missing values), SEM is unable to score a win. Although this dataset contains no local structure, and we were forced to use high min–sup values for KRIMP, the resulting code tables still allow for better reconstruction of the original data than with the SEM induced global Bayes Net model.

Similar to the previous experiments, no strong trend presents itself when we compare between MCAR and MAR. For KM accuracy is harmed slightly on average, but its (ϵ, δ)-correctness remains stable or even improves. For SEM, however, we do see that for both *alarm* and *chess* the data quality does suffer significantly.

7 Discussion

The experimental results of our methods show that compression is a very viable approach to the data completion problem. All three our parameter-free data completion algorithms show that approximating the best compressible completed database leads to high quality imputation.

Provided with undamaged training data, SC provides highly accurate estimates. The method was shown to be robust: even up to 24% missing values, both accuracy and (ϵ, δ)–correctness of the completed data are very high.

For the realistic case of only insufficient complete training data being available, KRIMP Minimisation is the right choice for a data completion algorithm. It finds good approximations of the best compressible completed database, and is shown to provide both provide high accuracy and very good (ϵ, δ)–scores. Further, the complexity of the orig-

inal data is approximated within a single percent.

Compared to the state of the art in missing value estimation, Structural EM, the completions that KM offers provide both higher accuracy and adhere better to all count statistics of the original data.

Here we only consider the code tables from the KRIMP compressor. As the proposed methods operate straightforwardly, it is possible to employ other compression schemes instead, for instance to better suit other data types.

8 Conclusions

In this paper we considered the problem of high quality imputation of missing data. To test this objectively we propose (ϵ, δ)–correctness to measure the difference between two databases in terms of count statistics.

We presented three KRIMP–based methods for imputation of incomplete datasets. All follow the MDL–principle: the completed database that can be compressed best is the best completed database. This, because then the imputations adhere to the local patterns that are present in the database, instead of only keeping its global statistics correct.

Both the greedy Simple Completion and randomised KRIMP Completion offer high performance when sufficient complete training data is available. By minimising the compressed size of the imputed data, KRIMP Minimisation performs evenly well when no complete data is available. Besides providing high accuracy, all three completion algorithms render imputations that particularly respect the variance of the original data.

Our methods consider local patterns in the data, rather than a global model; such local patterns are often smoothed out from a global stance. The experiments show that the local approach is indeed superior to the global approach, both in terms of accuracy and quality of the completed databases.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI, 1996.

[2] P. Allison. *Missing Data – Quantitative Applications in the Social Science*. Sage Publishing, 2001.

[3] M. Bankier. Canadian census minimum change donor imputation methodology. In *Proceedings of the UN/ECE Workshop on Data Editing*, 2000.

[4] R. Bruni. Discrete models for data imputation. *Discrete Applied Mathematics*, 144:59–69, 2004.

[5] S.van Buuren. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research*, 16:219–242, 2007.

[6] B. Chen, W. Winkler, and R. Hemmig. Using the DISCRETE edit system for ACS surveys. Technical report, U.S. Bureau of the Census, 2000.

[7] F. Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library: http://www.csc.liv.ac.uk/˜frans/KDD/Software/LUCS_KDD_DN/. 2003.

[8] T. Cover and J. Thomas. *Elements of Information Theory, 2nd ed.* John Wiley and Sons, 2006.

[9] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[10] N. Friedman. Learning bayesian networks in the presence of missing values and hidden variables. In *Proceedings of International Conference on Machine Learning*, pages 125–133, 1997.

[11] N. Friedman. The bayesian structural EM algorithm. In *Proceedings of the International Conference on Uncertainty in AI*, pages 129–138, 1998.

[12] N. Friedman and G. Elidan. LibB for Windows/Linux programs 2.1: http://www.cs.huji.ac.il/labs/compbio/LibB. 2008.

[13] P. D. Grünwald. Minimum description length tutorial. In P. Grünwald and I. Myung, editors, *Advances in Minimum Description Length*. MIT Press, 2005.

[14] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[15] J. Kovar and W. Winkler. Comparison of GEIS and SPEER for editing economic data. Technical report, U.S. Bureau of the Census, 2000.

[16] M.van Leeuwen, J. Vreeken, and A. Siebes. Compression picks the item sets that matter. In *Proceedings of the 10th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 585–592, 2006.

[17] R. Little and D. Rubin. *Statistical Analysis with Missing Data (2nd Edition)*. John Wiley and Sons, 2002.

[18] K. Murphy. Bayes net toolbox for Matlab: http://www.cs.ubc.ca/˜murphyk/Software/BNT/. 1997.

[19] D. Rubin. *Multiple imputation for nonresponse in surveys*. John Wiley and Sons, 1987.

[20] J. Schafer. Analysis of incomplete multivariate data. *Monographs on Statistics and Applied Probability*, 72, 1997.

[21] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *Proceedings of the SIAM Conference on Data Mining*, pages 393–404, 2006.

[22] J. Vreeken and A. Siebes. Filling in the blanks – Krimp minimisation for missing data. In *Proceedings of the IEEE International Conference on Data Mining*, 2008.

[23] J. Vreeken, M. van Leeuwen, and A. Siebes. Characterising the difference. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 765–774, 2007.

[24] J. Vreeken, M. van Leeuwen, and A. Siebes. Preserving privacy through data generation. In *Proceedings of the IEEE International Conference on Data Mining*, pages 685–690, 2007.

[25] I. Wasito and B. Mirkin. Nearest neighbour approach in the least-squares data imputation algorithms. *Journal of Information Sciences*, 167:1–25, 2005.