Preserving Privacy through Data Generation

Jilles Vreeken, Matthijs van Leeuwen and Arno Siebes
Universiteit Utrecht
{jillesv,mleeuwen,arno}@cs.uu.nl

Abstract

Many databases will not or can not be disclosed without strong guarantees that no sensitive information can be extracted. To address this concern several data perturbation techniques have been proposed. However, it has been shown that either sensitive information can still be extracted from the perturbed data with little prior knowledge, or that many patterns are lost.

In this paper we show that generating new data is an inherently safer alternative. We present a data generator based on the models obtained by the MDL-based KRIMP [12] algorithm. These are accurate representations of the data distributions and can thus be used to generate data with the same characteristics as the original data.

Experimental results show a very large pattern-similarity between the generated and the original data, ensuring that viable conclusions can be drawn from the anonymised data. Furthermore, anonymity is guaranteed for suited databases and the quality–privacy trade-off can be balanced explicitly.

1. Introduction

Many databases will not or can not be disclosed without strong guarantees that no sensitive information can be extracted from it. The rationale for this ranges from keeping competitors from obtaining vital business information to the legally required protection of privacy of individuals in census data. However, it is often desirable or even required to publish data, leaving the question how to do this without disclosing information that would compromise privacy.

To address these valid concerns, the field of privacy-preserving data mining (PPDM) has rapidly become a major research topic. In recent years ample attention is being given to both defender and attacker stances, leading to a multitude of methods for keeping sensitive information from prying eyes. Most of these techniques rely on perturbation of the original data: altering it in such a way that given some external information it should be impossible to recover individual records within certainty bounds.

Data perturbation comes in a variety of forms, of which adding noise [2], data transformation [1] and rotation [3] are the most commonly used. At the heart of the PPDM problem is the balance between the quality of the released data and the amount of privacy it provides. While privacy is easily ensured by strongly perturbing the data, the quality of conclusions that can be drawn from it diminishes quickly. This is inherent of perturbation techniques: sensitive information cannot be fully masked without destroying non-sensitive information as well [6]. This is especially so if no special attention is given to correlations within the data by means of multidimensional perturbation [5], something which has hardly been investigated so far [9].

An alternative approach to the PPDM problem is to generate new data instead of perturbing the original. This has the advantage that the original data can be kept safe as the generated data is published instead, which renders data recovery attacks useless. To achieve this, the expectation that a data point in the generated database identifies a data point in the original database should be very low, whilst all generated data adhere to the characteristics of the original. Data generation as a means to cover-up sensitivities has been explored in the context of statistical databases [8], but that method ignores correlations as each dimension is sampled separately.

We propose a novel method that uses data generation to guarantee privacy while taking important correlations into account. For this we use the MDL-based KRIMP algorithm [12] that has been shown to provide accurate pattern-based approximations of data distributions. The high quality of the approximations was verified through classification [7], and consequently put to use for determining and characterising dissimilarities between datasets [13]. Using the patterns picked by MDL, we construct a model that generates data very similar (but not equal) to the original data. Experiments show that the generative model is well suited for producing data that conserves the characteristics of the original data while preserving privacy.

2. The Problem

A good PPDM technique should not only preserve privacy but also quality, which can be formulated as:

A database db_{priv} induced from a database db_{orig} is privacy and quality preserving *iff:*

a. no sensitive information in db_{orig} can be derived from db_{priv} given a limited amount of external information (privacy requirement);

b. models and patterns derived from db_{priv} by data mining techniques are also valid for db_{orig} (quality requirement).

From this statement follows a correlated data generation approach to induce privacy and quality preserving databases, for which the above requirements can be translated into concrete demands.

It is hard to define an objective measure for the privacy requirement, as all kinds of 'sensitive information' can be present in a database. We guarantee privacy in two ways. Firstly, the probability that a transaction in db_{orig} is also present in db_{priv} should be small. Secondly, the more often a transaction occurs in db_{orig}, the less harmful it is if it also occurs in db_{priv}. This is encapsulated in the Anonymity Score, in which transactions are grouped by the number of times a transaction occurs in the original database (support):

Definition 1: for a database db_p based on db_o, define the Anonymity Score (AS) as:

$$AS(db_p, db_o) = \sum_{supp \in db_o} \frac{1}{supp} P(t \in db_p \mid t \in db_o^{supp}) \qquad (1)$$

In this definition, db^{supp} is defined as the selection of db with only those transactions having a support of *supp*. For each support level in db_o, a score is obtained by multiplying a penalty of 1 divided by the support with the probability that a transaction in db_o with given support also occurs in db_p. These scores are summed to obtain *AS*. Note that when all transactions in db_o are unique (i.e., have a support of 1), *AS* is equal to the probability that a transaction in db_o also occurs in db_p.

Worst case is when all transactions in db_{orig} also occur in db_{priv}. In other words, if we choose db_{priv} equal to db_{orig}, we get the highest possible score for this particular database, which we can use to normalise between 0 (best possible privacy) and 1 (no privacy at all):

Definition 2: for a database db_{priv} based on db_{orig}, define the Normalised Anonymity Score (NAS) as:

$$NAS(db_{priv}, db_{orig}) = \frac{AS(db_{priv}, db_{orig})}{AS(db_{orig}, db_{orig})} \qquad (2)$$

To conform to the quality requirement, the frequent pattern set of db_{priv} should be very similar to that of db_{orig}. We will measure pattern-similarity in two ways: 1) on database level through a database dissimilarity measure [13] and 2) on the individual pattern level by comparing frequent pattern sets. For the second part, pattern-similarity is high iff the patterns in db_{orig} also occur in db_{priv} with (almost) the same support. So:

$$P(|\mathrm{supp}_{priv} - \mathrm{supp}_{orig}| > \delta) < \varepsilon \qquad (3)$$

The probability that a pattern's support in db_{orig} differs much from that in db_{priv} should be very low: the larger δ, the smaller ϵ should be. Note that this second validation implies the first: only if the pattern sets are highly similar, the code tables become similar, which results in low measured dissimilarity. Further, it is computationally much cheaper to measure the dissimilarity than to compare the pattern sets.

3. Preliminaries

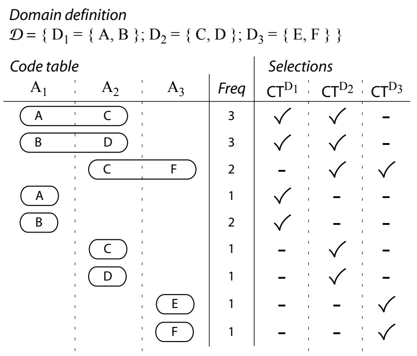
In this paper we discuss categorical databases. A database *db* is a bag of tuples (or transactions) that all have the same attributes $\{A_1,\ldots,A_n\}$. Each attribute A_i has a discrete domain of possible values $D_i \in \mathcal{D}$.

The KRIMP algorithm operates on item set data, as which categorical data can easily be regarded. The union of all domains $\cup D_i$ forms the set of items $\mathcal{I}$. Each transaction t can now also be regarded as a set of items $t \in \mathcal{P}(\mathcal{I})$. An item set $I \in \mathcal{I}$ occurs in a transaction $t \in db$ iff $I \subseteq t$. The support of I in *db* is the number of transactions in the database in which I occurs.

KRIMP is a heuristic method based on MDL: it attempts to find that (small) set of patterns that compresses a database best. Such a KRIMP code table (pattern set) consists of item sets with a usage frequency. The compression process starts with an empty code table, a database and a set of candidate (frequent) patterns. Each of these patterns is tested individually: it is kept in the code table iff it helps to better compress the database. For more details, see [12]. The performance and effectiveness of the method was further investigated through classification [7], recently we used the method to devise a database dissimilarity measure and several ways to characterise differences [13].

4. KRIMP Categorical Data Generator

In this section we present our categorical data generation algorithm. We start off with a simple example, after which we will detail the algorithm formally.


Domain definition
$\mathcal{D} = \{ D_1 = \{ A, B \}; D_2 = \{ C, D \}; D_3 = \{ E, F \} \}$

Code table				*Selections*		
A_1	A_2	A_3	*Freq*	CT^{D_1}	CT^{D_2}	CT^{D_3}
A	C		3	✓	✓	-
B	D		3	✓	✓	-
	C	F	2	-	✓	✓
A			1	✓	-	-
B			2	✓	-	-
	C		1	-	✓	-
	D		1	-	✓	-
		E	1	-	-	✓
		F	1	-	-	✓

Figure 1. Example for 3-column database. Each frequency is Laplace corrected by 1.

4.1 Generating a transaction, an example

Suppose we need to generate a new transaction for a simple three-column categorical database. To apply our generation scheme, we need a domain definition $\mathcal{D}$ and a KRIMP code table *CT*, both shown in Figure 1.

We start off with an empty transaction and fill it by iterating over all domains and picking an item set from the code table for each domain that has no value yet. We first want to assign a value for the first domain, D_1, so we have to select one pattern from those patterns in the code table that provide a value for this domain. This subset is shown as selection CT^{D_1}.

Using the frequencies of the code table elements as probabilities, we randomly select an item set from CT^{D_1}; elements with high frequency occur more often in the original database and are thus more likely to be picked. Here we randomly pick 'BD' (probability 3/9). This set selects value 'B' from the first domain, but also assigns a value to the second domain, namely 'D'.

To complete our transaction we only need to choose a value for the third domain. We do not want to change any values once they are assigned, as this might break associations within an item set previously chosen. So, we do not want to pick any item set that would re-assign a value to one of the first two domains. Considering the projection for the third domain, CT^{D_3}, we thus ignore set CF(2), as it would re-assign the second domain to 'C'. From the remaining sets E(3) and F(3), both with frequency 3, we randomly select one – say, 'E'. This completes generation of the transaction: 'BDE'.

4.2 Definition of the generator

Here we will detail our data generator more formally. First, define the projection CT^D as the subset of item sets in *CT* that define a value for domain $D \in \mathcal{D}$. To generate a database, our categorical data generator requires: the original database, a Laplace correction value, a *min-sup* value for mining candidates for the KRIMP algorithm and the number of transactions to generate. We present the full algorithm below.

Generation starts with an empty database *gdb* (line 2). To obtain a code table *CT*, the KRIMP algorithm is applied to the original database *db* (3). A Laplace correction *laplace* is added to all elements in the code table (4 and 5). We return the generated database when it contains *num-trans* transactions (7 to 9).

Generation of a transaction is started with an empty transaction *t* (line 11). As long as $\mathcal{D}$ is not empty (12), our transaction is not finished and we continue. First, a domain *D* is randomly selected (13). From the selection CT^D, one item set is randomly chosen, with probabilities defined by their relative frequencies (14). After the chosen set is added to *t* (15), we filter from *CT* all sets that would redefine a value – i.e. those sets that intersect with the definitions of the domains for which *t* already has a value (16 and 17). Further, to avoid reconsideration we also filter these domains from $\mathcal{D}$ (18). After this the next domain is picked from $\mathcal{D}$ and another item set is selected; this is repeated until $\mathcal{D}$ is empty (and *t* thus has a value from each domain).

Note that code table elements are treated fully independently, as long as they do not re-assign values. Correlations between dimensions are stored explicitly in

ALGORITHM KRIMPGENERATOR

```
1  GenerateDatabase(db, laplace, min-sup, num-trans)
2      $gdb = \emptyset$
3      $CT$ = KRIMP(db, MineCandidates(db, min-sup))
4      for each item set e in CT
5          e.frequency += laplace
6      $\mathcal{D}$ = db.getDomains
7      while(|gdb| < num-trans)
8          gdb = gdb + GenerateTransaction(CT, $\mathcal{D}$)
9      return gdb

10 GenerateTransaction(CT, $\mathcal{D}$)
11     $t = \emptyset$
12     while $\mathcal{D} \neq \emptyset$
13         pick a random $D \in \mathcal{D}$
14         is = PickRandomItemSet($CT^D$)
15         $t = t \cup is$
16         for each domain C for which is has a value
17             $CT = CT \setminus CT^C$
18             $\mathcal{D} = \mathcal{D} \setminus C$
19     return t

20 PickRandomItemSet(CT)
21     weights = $\{ e.\text{frequency} \mid e \in CT \}$
22     is = WeightedSample(weights, CT)
23     return is
```

the item sets and are thus taken into account implicitly.

The two parameters *laplace* and *min-sup* control the amount of privacy provided in the generated database. The Laplace correction parameter directly controls the data diversity and strength of the correlations within the data. Before the generation process, a small constant is added to the frequencies of all code table elements. As code tables always contain all single values, this ensures that all values have a small probability of being chosen. Thus, 1) a complete transaction can always be generated and 2) all possible transactions can be generated. For this purpose the correction needs only be small. However, the strength of the correction influences the chance an otherwise unlikely code table element is used; with larger correction, the influence of the original data distribution is dampened and diversity is increased.

The second parameter, *min-sup*, has a strong relation to the *k*-anonymity blend-in-the-crowd approach [11]. The *min-sup* parameter has (almost) the same effect as *k*: patterns that occur less than *min-sup* times in the original database are not taken into account by KRIMP. As they cannot get in the code table, they cannot be used for generation. Particularly, complete transactions have to occur at least *min-sup* times in order for them to make it to the code table. In other words, original transactions that occur less often than *min-sup* can only be generated by chance if often occurring patterns are combined. It follows that when more patterns have to be combined, it becomes less likely that such 'private' transactions are generated.

5. Experiments

5.1 Experimental Setup

In our experiments, we use a selection of datasets from the UCI repository [4]. Table 1 reports the minimum support levels we use for mining the frequent item sets that function as candidates for KRIMP.

For all experiments we use a Laplace correction parameter of 0.001, an arbitrarily chosen small value to ensure that all code table elements can be chosen in generation. All experimental results presented below are averaged over 10 runs and all generated databases have the same number of transactions as the originals.

5.2 Results

To quantify the likeness of the generated databases to their original counterparts, we use a database dissimilarity measure [13]. To judge these measurements, we also provide the dissimilarity between the original

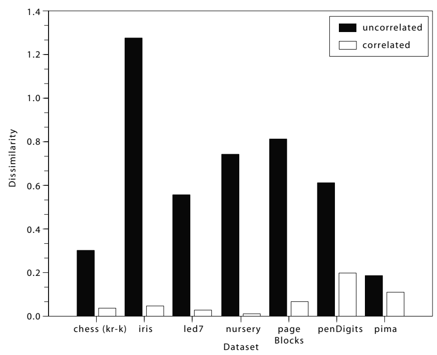


Figure 2. Dissimilarity scores between generated and original databases.

database and independent random samples of half the size from the original database.

In Table 1 we show both these internal dissimilarity scores and the dissimilarity measurements between the original and generated databases. To put the reported dissimilarities in perspective, note that the dissimilarity measurements between the classes in the original databases range from 0.29 up to 12 [13]. The measurements thus indicate clearly that the generated databases adhere very closely to the original data distribution.

Databases generated at higher values of the *min-sup* parameter show slightly larger dissimilarity. The bar diagram of Figure 2 shows a comparison of the dissimilarity scores between uncorrelated (only single values in the code table) and correlated generation (regular code table obtained with KRIMP). As expected, when correlations are taken into account, the generated databases are far more similar to the originals.

Now, let us consider quality on the level of individual patterns; for this, we mined frequent item sets from both the generated and original databases. A comparison between the resulting sets is presented in Table 1.

Large parts of the generated and original frequent pattern sets consist of exactly the same items sets, as can be seen from the fifth column. For example, for Led7 and Nursery about 90% of the mined item sets are equal. Due to the relatively high *min-sup* used for PenDigits, a comparatively low 25% of the original patterns is found – although more than 90% of the sets mined from the generated database are original.

For item sets found in both cases, the average difference in support between original and generated is very small, as the next column shows. Iris is a bit of an outlier here, caused by the very small size of the dataset. Not only the average is low, standard deviation is also small: as can be seen from Figure 3, almost all sets have a very small support difference. The generated databases thus fulfil the support difference demands we formulated in Equation 3.

Table 1. Dissimilarity measurements (between original and generated datasets), frequent pattern set comparisons and Normalised Anonymity Scores for a range of UCI datasets. As candidates, frequent item sets up to the given minimum support level were used.

Dataset	**Krimp**	**Dissimilarity**		**Frequent Pattern Set Comparison**			**Anonymity**
Name	*Min-sup*	*Gen. vs. orig.*	*Orig. internal*	*% equal item sets*	*% avg sup diff equal item sets*	*% avg sup new item sets*	*Normalised Anonymity Score*
Chess (kr-k)	1	0.037	0.104	71	0.01	0.01	0.30
Iris	1	0.047	0.158	83	1.69	0.80	0.72
Led7	1	0.028	0.171	89	0.14	0.06	0.66
Mushroom*	20	0.010	0.139	-	-	-	0.09
Nursery	1	0.011	0.045	90	0.04	0.03	0.49
PageBlocks	1	0.067	0.164	75	0.06	0.02	0.77
PenDigits	50	0.198	0.124	25	0.50	0.59	0.22
Pima	1	0.110	0.177	60	0.30	0.14	0.64

* Only closed item sets used as candidates. No values for frequent pattern set comparison as it was infeasible to compute these.

The seventh column of Table 1 contains the average supports of item sets that are newly found in the generated databases; these supports are very low. All this together clearly shows that there is a large pattern-similarity, thus showing a high quality according to our problem statement.

To measure the level of provided anonymity, we calculate the Normalised Anonymity Score as given by Definition 2. These scores are presented in the last column of Table 1. As lower scores indicate better privacy, some datasets (e.g. Mushroom, PenDigits) are anonymised very well, while other datasets (such as PageBlocks) are not. As discussed in Section 4, the *min-sup* parameter of our generation methods doubles as a *k*-anonymity provider.

To show the effect of *min-sup* in action, as an example we increase the *min-sup* for the Chess database to 50. While the so-generated database is still very similar to the original (dissimilarity of 0.19), privacy is considerably increased - which is reflected by a Normalised Anonymity Score of 0.15. For further evidence of the *k*-anonymity obtained, we take a closer look at PenDigits, for which we use a *min-sup* of 50. Of all transactions with support < 50 in the generated database, only 3% is also found in the original database with support < 50. It is thus highly unlikely that one picks a 'real' transaction from the generated database with support lower than *min-sup*.

Although not all generated databases preserve privacy very well, the results indicate that privacy can be obtained. This brings us to the question *when* privacy can be guaranteed. This not only depends on the algorithm's parameters, but also on the characteristics of the data. It is difficult to determine the structure of the data and how the parameters should be set in advance, but during the generation process it is easy to check whether privacy is going to be good.

The key issue is whether transactions are generated by only very few or many code table elements. In Figure 4 we show this relation: for each dataset in Table 1, a cross marks the average number of item sets used to generate a single transaction and the Normalised Anonymity Score. In the bottom-right corner we find the generated databases that preserve privacy well, includ-

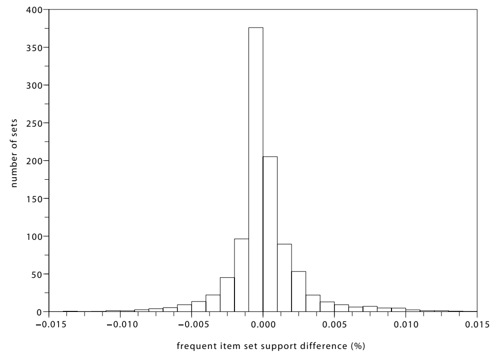


Figure 3. Support differences, for identical item sets in generated and orig. Led7.

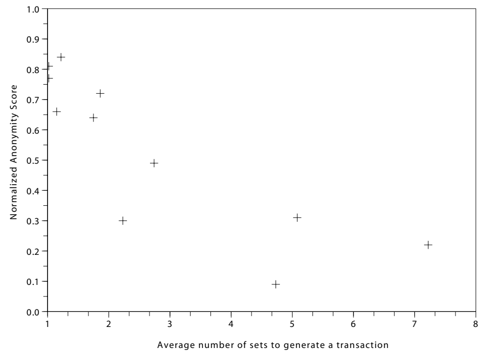


Figure 4. Average number of patterns to generate a transaction versus NAS.

ing PenDigits. At the top-left reside those databases for which too few elements per transaction are used during generation, leading to bad privacy; PageBlocks and Led7 are the main culprits. Thus, by altering the *min-sup*, this relation allows for explicit balancing of privacy and quality of the generated data.

6. Discussion

The experimental results in the previous section show that the databases generated by our data generator are of very high quality; pattern similarity on both database level and individual pattern level is very high. Furthermore, we've shown that it is possible to generate high quality databases while privacy is preserved. The Normalised Anonymity Scores for some datasets are very low, showing that hardly any transactions that occur few times in the original database also exist in the generated database. As expected, increasing *min-sup* leads to better privacy, while dissimilarity remains good. The trade-off between quality and privacy can thus be balanced explicitly.

A natural link between our method and *k*-anonymity is provided by the *min-sup* parameter, of which we've shown that it works in practice. While we haven't explored this parameter in this work, it is also possible to mimic *l*-diversity [10], as in our method the *laplace* parameter acts as diversity control. The higher the Laplace correction, the lesser the characteristics of the original data are taken into account (thus degrading quality, but increasing diversity).

To obtain even better privacy, one can also directly influence model construction: for example, by filtering the KRIMP candidates prior to building the code table. Correlations between specific values and/or domains can be completely filtered.

7. Conclusions

We introduce a pattern-based data generation technique as a solution to the privacy-preserving data mining problem in which data needs to be anonymised. Using the MDL-based KRIMP algorithm we obtain accurate approximations of the data distribution, which we transform into high-quality data generators with a simple yet effective algorithm. Experiments show that the generated data meets the criteria we posed in the problem statement. For more extensive results and analysis, see [14].

Preserving privacy through data generation does not suffer from the same weaknesses as data perturbation. By definition, it is impossible to reconstruct the original database from the generated data, with or without prior knowledge. The privacy provided by the generator can be regulated and balanced with the quality of the conclusions drawn from the generated data. For suited databases, the probability of finding a 'real' transaction in the generated data is extremely low.

8. References

[1] Aggarwal, C. C., and Yu. P. S. "A condensation approach to privacy preserving data mining", *Proc. EDBT*, 2004, pp.183-199.

[2] Agrawal, R., and Srikant, R. "Privacy-preserving data mining", *Proc. SIGMOD,* 2000, pp.439-450.

[3] Chen, K., and Liu, L. "Privacy Preserving Data Classification with Rotation Pertubation", *Proc. ICDM,* 2005, pp.589-592.

[4] Coenen, F. *The LUCS-KDD Discretised/normalised ARM and CARM Data Library*, http://www.csc.liv.ac.uk/~frans/KDD/Software/, 2003.

[5] Huang, Z., Du, W., and Chen, B. "Deriving private information from randomized data", *Proc. SIGMOD,* 2005.

[6] Kargupta, H., Datta, S., Wang, Q., and Sivakumar, K. "Random-data perturbation techniques and privacy-preserving data mining", *Knowledge and Information Systems* 4(7), 2005, pp.387-414.

[7] Van Leeuwen, M., Vreeken, J., and Siebes, A. "Compression Picks Item Sets That Matter", *Proc. PKDD*, 2006, pp.585-592.

[8] Liew, C.K., Choi, U.J., and Liew, C.J. "A data distortion by probability distribution", *ACM Trans. Database Systems* 3(10), 1985, pp.395-411.

[9] Liu, K., Giannella, C., and Kargupta, H. "An Attacker's View of Distance Preserving Maps for Privacy Preserving Data Mining", *Proc. PKDD*, 2006, pp.297-308.

[10] Machanavajjhala, A., Gehrke, J., Kifer, D., and Venkitasubramaniam, M. "*l*-Diversity: Privacy Beyond *k*-Anonymity", *Proc. ICDE,* 2006, pp.24-35.

[11] Samarati, P. "Protecting respondents' identities in microdata release", *IEEE Trans. Knowledge and Data Engineering*, 2001, pp.1010-1027.

[12] Siebes, A., Vreeken, J., and Van Leeuwen, M. "Item Sets That Compress", *Proc. SIAM SDM,* 2006, pp.393-404.

[13] Vreeken, J., Van Leeuwen, M., and Siebes, A. "Characterising the Difference", *Proc. SIGKDD,* 2007.

[14] Vreeken, J., Van Leeuwen, M., and Siebes, A. *Privacy Preservation through Data Generation*, Technical Report UU-CS-2007-020, Universiteit Utrecht, 2007.