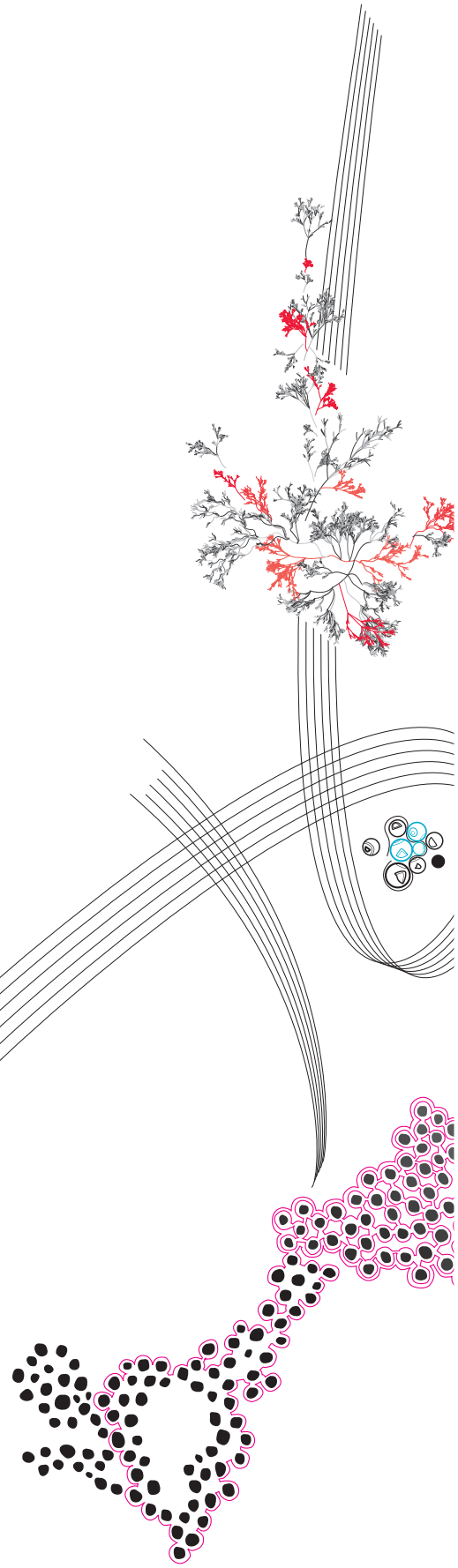


BSc Thesis Applied Mathematics

Improving data quality in a probabilistic database by means of an autoencoder

R.R. Mauritz

Supervisors: Dr. ir. J. Goseling & Dr. ir. M. van Keulen

January, 2020

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Abstract

In the field of *data integration*, the final result often contains uncertainties regarding the resulting data. A way to deal with these uncertainties is to use a *probabilistic database* (PDB) that stores not only just the static values but allows multiple data possibilities by assigning probabilities to each possibility. In this process of *probabilistic data integration*, an important step is to improve the data quality of the data in the PDB once it has been merged into the PDB [21]. Doing such normally requires external experts to manually account for this. However, based on the notion that the probabilistic data $\mathcal{D}_{PDB}$ (in the form of uncertainty/probability parameters) from the PDB indirectly contains *evidence* from its underlying *ground truth data generating distribution* $P(\mathcal{D}_{GT})$, we develop a model that both captures and uses this evidence to achieve *data quality improvement* in a PDB storing *categorical nominal data*.

In order to do so, we first model the problem of data quality improvement in a PDB and state that 'improving data quality' is about decreasing the distance between the probabilistic data $\mathcal{D}_{PDB}$ and its associated underlying ground truth $\mathcal{D}_{GT}$. We then approach the problem by modeling $P(\mathcal{D}_{GT})$ by means of a *Bayesian Network* (BN) and develop a *Probabilistic Inference Bayesian Network* (PIBN) model that approaches data quality improvement by combining the notions of *probabilistic inference* [5] and the propagation of *virtual evidence* [17] in such a BN. In the development of this model, we see that data quality improvement can be achieved by for each record $\boldsymbol{x}_i \in \mathcal{D}_{PDB}$ combining the information from $\boldsymbol{x}_i$ itself together with the prior information defined by $P(\mathcal{D}_{GT})$.

As this latter model is only applicable given $P(\mathcal{D}_{GT})$ is known, we use this knowledge to develop a new model that is applicable in an unsupervised setting by learning $P(\mathcal{D}_{GT})$ indirectly from $\mathcal{D}_{PDB}$. We do this by means of a *denoising autoencoder* (DAE) [14] that is directly trained on the uncertainty parameters $\mathcal{D}_{PDB}$ and is learned to capture evidence from $P(\mathcal{D}_{GT})$ by using the denoising autoencoder principles as *regularization* technique.

After having developed several quality measures, it turns out that this DAE model is well able to achieve data quality improvement when we test it on several synthetic data sets. We also compare its performance with the performance of the supervised PIBN model to conclude that the performance of the unsupervised DAE model on these data sets is only slightly less good and advice to do future research on hyperparameter tuning of the DAE model.

Acknowledgements

Foremost I would like to thank my supervisors Dr. ir. Jasper Goseling and Dr. ir. Maurice van Keulen, for their essential support and guidance during the entire time of my bachelor thesis. During the frequent meetings that we had, they provided me with useful insights and posed the right questions that helped me to overcome several difficulties and led me to new, important insights. Without their help, I would not have been able to present to the reader this paper as it is now.

On top of that, I would like to thank the University of Twente that has taught me the knowledge and provided me the tools that I needed in order to approach the research question of this thesis.

Last but not least I would like to thank all of my friends, Huan Wu and Bram Jonkheer in particular, and family that helped me to make progress by providing me positive energy and good distraction.

- Rutger Mauritz

Contents

1 Introduction

In the field of *data-integration*, i.e. combining several data sources into a single and unified view, the result often contains uncertainties regarding the extracted and merged data. A way to incorporate these uncertainties is to store the data in a *probabilistic database* (PDB), a database that does not just store static values but allows storing multiple possibilities called a *possible world*, each with an associated probability, representing the uncertainty. This process is called *probabilistic data integration* (PDI) and is depicted in Fig. 1.

The PDI process consists of two main phases:

1. **Phase I - Initial data integration:**
   The integration of different data sources into a single and unified view in a PDB.
2. **Phase II - Continuous improvement:**
   Improving the quality of the data by reducing the uncertainty in the data based on evidence.

In the last phase, the 'Gather evidence' step usually means that human experts manually inspect the integrated data view and based on their knowledge give feedback or 'provide evidence' so that the uncertainty regarding specific possible worlds is increased or decreased.

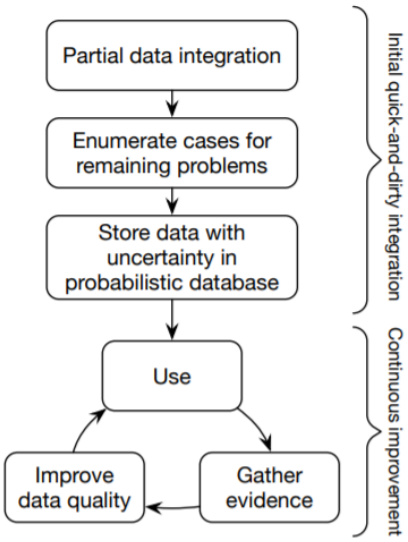


FIGURE 1: Probabilistic Data Integration [21]

In practice the attributes in a database table are almost always correlated with each other. As an example, when having an attribute *[City]* representing the names of cities from all over the world and attribute *[Temperature]* representing the corresponding measured average temperature in that city, it is obvious that these two attributes are not independent. Mathematically speaking, when all samples in data set $\mathcal{D}$ are i.i.d., $\mathcal{D}$ can be seen as a set of samples drawn from one and the same underlying data generating probability distribution $P(\mathcal{D})$, a process which is called the *data generating process*. Since each record in a database is a tuple of multiple attribute values, $P(\mathcal{D})$ should be regarded as a multivariate joint probability distribution over the attributes $A_j, j = 1, 2, \ldots, M$ of the database.

When having such dependencies between the attributes of a database table, a consequence is that after Phase I of the PDI process (Fig. 1), the remaining uncertainties are in essence correlated with each other as well. As a continuation of the above example, say a record *['Amsterdam', '11.2']* has to be extracted and merged into a PDB and for simplicity, assume 'Amsterdam' and 'Rome' are the only two cities in the world. Say that during the data integration process, uncertainty has arisen yielding a probability of 0.5 that the corresponding city is *'Amsterdam'* and a probability of 0.5 that the corresponding city is *'Rome'*. Since we know that the average temperature in Rome is much higher than '11.2' (encoded by $P(\mathcal{D})$), we will say that the probability of the city being *'Rome'* should be decreased and the probability of the city being *'Amsterdam'* should be increased. Moreover, given that we were not entirely sure about the measured average temperature, we expect that the uncertainty regarding the measured temperature changes as well, as there is a possibility

that that corresponding city was indeed *'Rome'*, having a higher average temperature. In other words, the uncertainty regarding the extracted *[City]* attribute and the uncertainty regarding the extracted *Temperature* attribute have influence on each other, given that they both arose independently.

The above example illustrates that Phase II of the PDI process is about massaging the probability parameters such that the dependencies defined by $P(\mathcal{D})$ are incorporated into these resulting probabilities with the aim of obtaining an end result that is closer to the ground truth. This is a very costly and time-consuming process, especially when experts from outside the system have to manually account for it. However, since information from the data generating distribution $P(\mathcal{D})$ is indirectly present in the PDB, we propose to design a model that captures this information automatically so that the data quality may be improved.

In order to do so, we first formally define what it means to 'improve data quality' and define a measure that indicates whether the data quality has improved. We do so by introducing the notion of *ground truth* and use this notion in combination with the notion of a *data generating distribution* to develop a probabilistic model - named '*Probabilistic Inference Bayesian Network*' (PIBN) - that can improve the data quality given that we know $P(\mathcal{D})$. This latter model is built around a *Bayesian Network* (BN) and the notions of *virtual evidence* and *probabilistic inference* in such a BN. This model helps us to understand the fundamental concepts in data quality improvement in a PDB. A problem that comes with this model, however, is that it is unclear how to use it in case we do not know the underlying data generating distribution. Based on the things we learned from the development of the PIBN model, we therefore propose to use a *Denoising Autoencoder* [13] (DAE) that takes the probability parameters as input and exploit its denoising features to improve the quality of the data residing in the PDB by changing these probability parameters. As it is not straightforward how to use an autoencoder in combination with probabilistic data, we extensively describe our approach and the construction of this DAE model.

It turns out that both the PIBN and DAE model are well able to improve the data quality when we test their performance on synthetic data sets. What's more, the comparison of both models leads us to some interesting insights that will form the basis of future research.

Summarized, we have contributed to the problem of data quality improvement in a PDB by

- ... describing the underlying problem of data quality improvement in a PDB by reformulating it as a process of incorporating the indirectly present data dependencies from the underlying ground truth data generating distribution into it, Sec. 2.2 and Sec. 2.3.
- ... describing how data quality can be quantified by means of a proper distance metric between the data and the corresponding ground truth, Sec. 2.3.
- ... constructing a probabilistic model (PIBN) that uses these notions to improve data quality in case the ground truth data generating distribution is known, Sec. 3.
- ... constructing an autoencoder model (DAE) that can improve the data quality in an unsupervised setting. What's more, we connect this model to the previously

constructed PIBN model, Sec. 4.

- ... defining a proper testing scheme for the performance of such a DAE model, modeling the noise present in the data, and by defining good quality measures, Sec. 5.1 and Sec. 5.2.
- ... comparing the performance of the DAE model to the performance of the PIBN model for better model insights, Sec. 5.3.
- ... describing the future work that originates from this thesis, Sec. 6.

2 Problem statement and problem modeling

The goal of this research is to improve the data quality in the PDB by replacing the manual 'Gather evidence' step in the PDI process by an automated process. This automated process should capture the data-dependencies in the data and incorporate them into each of the records $\boldsymbol{x_i}, i = 1, 2, \ldots, N$ from the data in the probabilistic database, $\mathcal{D}_{PDB}$. In order to better describe this process and to come up with a mathematical approach to this problem, we will first define a model for a PDB and specify the notation that will be used and the assumptions that will be made in the rest of this paper. We then describe how $\mathcal{D}_{PDB}$ can be seen as a corrupted version of a set of *ground truth* data $\mathcal{D}_{GT}$ drawn from a data-generating distribution $P(\mathcal{D}_{GT})$. We continue on this by relating this notion to the notion of a decreasing divergence between $\mathcal{D}_{GT}$ and $\mathcal{D}_{PDB}$, so that data quality can be quantified. Finally, we use all of the above to formally describe what it means to improve data quality and how this can be quantified.

2.1 PDB modeling and assumptions

It is important to specify the structure of the PDB, the nature of the data it contains and the type of uncertainty it carries as well as the assumptions we made regarding all of the above. Since each of these different types require a different approach, it is necessary to limit this research to a particular type of data containing a particular type of uncertainty.

The probabilistic database that this research will be applied to has a following structure:

- A database in general consists of multiple tables. However, without loss of generality, we assume that the database we are working with comprises of just one table R, having M columns - column j representing attribute A_j - and N rows - row i representing instance $\boldsymbol{x_i}$ - which will be called a *tuple/record* from now on.
- Each attribute A_j contains categorical, nominal data. That is, each attribute A_j contains K_j categories $\{C_{j,1}, C_{j,2}, \ldots, C_{j,K_j}\}$. This is a strong assumption since many other possible data types could have been chosen. These other data types however, are beyond the scope of this thesis.
- Each record $\boldsymbol{x_i}$ is a set of attribute values $a_{i,j}, j = 1, 2, \ldots, M$. That is, $\boldsymbol{x_i} = \{a_{i,1}, a_{i,2}, \ldots, a_{i,M}\}$.
- The uncertainty residing in a PDB can come in many types and intensities. In this thesis, the focus will be on *attribute uncertainty*, meaning that we may be unsure about the value that an attribute of a tuple may take. It is for this reason that each attribute value $a_{i,j}$ is a tuple of probability parameters $[p_i(C_{j,1}), \ldots, p_i(C_{j,K_j})]$,

where each element $p_i(C_{j,k})$ represents the *marginal probability* (certainty/belief) that attribute A_j takes value $C_{j,k}$ in record $\boldsymbol{x_i}$. As a consequence, we have that

$$\sum_{k=1}^{K_j} p_i(C_{j,k}) = 1, \quad \forall \quad (i,j). \tag{1}$$

This thus means that when we mention, 'data in the PDB' ($\mathcal{D}_{PDB}$), we mean those probability parameters.

- A missing value ('no information') for an attribute value $a_{i,j}$ is chosen to be modelled as each category having the same associated marginal probability, that is: $p_i(C_{j,1}) = \ldots = p_i(C_{j,K_j}) = \frac{1}{K_j}$. One should note that many more solutions exist [11], such as using a different, non-uniform prior. We could also have chosen to use an approach such as mean replacement, EM imputation, etc., but we chose to use this simple approach as this is not in the scope of this thesis.

By means of an example, a simplified database that satisfies the above specified properties can be found in Fig. 2.

	Eye colour		**Hair colour**	
	Blue	*Brown*	*Light*	*Dark*
1	0.7	0.3	1.0	0.0
2	0.8	0.2	0.9	0.1
3	0.0	1.0	0.5	0.5

$\mathcal{D}_{PDB}$

FIGURE 2: Example PDB

In this database:

- $\boldsymbol{x}_1 = \{0.7, 0.3, 1.0, 0.0\}$
- $a_{1,1} = \boldsymbol{x}_1$.Eye colour $= [0.7, 0.3]$
- $p_2(C_{1,1}) = p_2(\text{Eye colour} = \text{Blue}) = 0.8$

2.2 Ground truth

We model the data $\mathcal{D}_{PDB}$ residing in a PDB as a corrupted/noisy version of the underlying ground truth data $\mathcal{D}_{GT}$, where 'corrupted'/'noisy' means that uncertainty (noise) is added to the ground truth data, as a result of the integration process. In other words, each record $\boldsymbol{x_i} \in \mathcal{D}_{PDB}$ is derived from an underlying ground truth record $\boldsymbol{x_i^{GT}} \in \mathcal{D}_{GT}$ that for each attribute A_j carries a 100% certainty for which category is observed (such that $\boldsymbol{x_i^{GT}}$ can be seen as a concatenation of one-hot encodings). By means of an example, this might be visualized as follows:

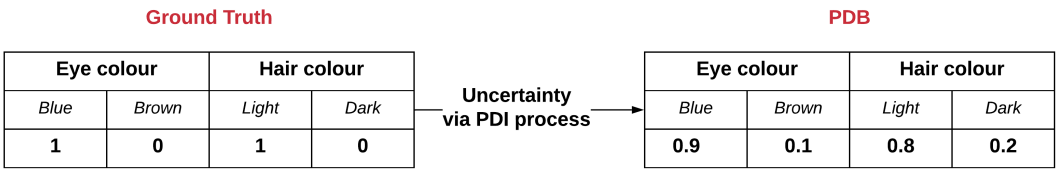


FIGURE 3: From $\mathcal{D}_{GT}$ to $\mathcal{D}_{PDB}$

As the assumption is made that all of the records $\boldsymbol{x_i^{GT}}$ are i.i.d., this data set $\mathcal{D}_{GT}$ should be regarded as a set of samples drawn from one and the same underlying data-generating distribution $P\big(\mathcal{D}_{GT}\big)$. This $P\big(\mathcal{D}_{GT}\big)$ can be seen as a multivariate joint probability distribution over the discrete random variables $A_1, A_2, \ldots, A_M$ resembling the attributes of the PDB.

2.3 Improving data quality

Based on this notion of ground truth, we can define what it means to improve the data quality of data residing in the PDB.

The notion of a ground truth allows us to say that 'improving data quality' in essence means that given a corrupted record $\boldsymbol{x_i} \in \mathcal{D}_{PDB}$, we incorporate evidence into it - where the evidence is the collection of data dependencies defined by $P(\mathcal{D}_{GT})$, being indirectly present in $\mathcal{D}_{PDB}$ - so that its corresponding updated record $\boldsymbol{x}_i^n$ is closer to its corresponding ground truth record $\boldsymbol{x}_i^{GT}$. Ideally, we want to reverse the corruption process as depicted in Fig. 3. In order to quantify the 'closeness' as mentioned above, we need to find a proper distance metric, which will be explained in Sec. 2.3.2.

2.3.1 Data quality improvement and corruption requirement

Before we do so, we should make an important remark first. By using this notion of 'data quality improvement', we get that by incorporating the dependencies defined by $P(\mathcal{D}_{GT})$ in a record $\boldsymbol{x}_i \in \mathcal{D}_{PDB}$, we may sometimes end up with a new record that has *lower* data quality, as it has diverged from its corresponding ground truth record $\boldsymbol{x}_i^{GT}$ due to particular noise in the data. To illustrate this, say we have two ground truth records $\boldsymbol{x}_1^{GT} = [1, 0, 1, 0]$ and $\boldsymbol{x}_2^{GT} = [0, 1, 1, 0]$, such that $P_{GT}(\boldsymbol{x}_1) \gg P_{GT}(\boldsymbol{x}_2)$. Now say that $\boldsymbol{x}_2^{GT}$ is corrupted such that $\boldsymbol{x}_2^{GT} \rightarrow \boldsymbol{x}_2 \in \mathcal{D}_{PDB} = [0.5, 0.5, 1, 0]$. If we were to update this record - $\boldsymbol{x}_2 \rightarrow \boldsymbol{x}_2^n$ - based on what we know from $P(\mathcal{D}_{GT})$, we would increase $p_2(C_{1,1})$, which results in the distance $d(\boldsymbol{x}_2^{GT}, \boldsymbol{x}_2^n) > d(\boldsymbol{x}_2^{GT}, \boldsymbol{x}_2)$, meaning that the data quality has decreased.

In other words, this notion of data quality improvement poses a *corruption requirement* by implying that data quality of a record $\boldsymbol{x}_i$ can only be improved given that the corruption is not such that based on $P(D_{GT})$, a different ground truth record $\boldsymbol{x}_j^{GT}$ is more likely based on $\boldsymbol{x}_i$.

2.3.2 Distance metric

Before we can define a proper distance metric to quantify the distance between $\boldsymbol{x}_i^n$ and $\boldsymbol{x}_i^{GT}$, we first need to understand that both $\boldsymbol{x}_i^n$ and $\boldsymbol{x}_i^{GT}$ can be seen as ensembles of parameters from *categorical (multinoulli) distributions*. The categorical probability distribution is a discrete probability distribution over random variable X whose sample space is the set of K individually identified categories. When having K categories $\{1, 2, \ldots, K\}$, the probability that X belongs to category i is defined by the probability mass function $f\big(X = i \big| \boldsymbol{p}\big) = p_i$, with $\boldsymbol{p} = (p_1, p_2, \ldots, p_K)$ and $\sum_{i=1}^{K} p_i = 1$, each p_i being the probability that observation i is made. We can thus regard each element $a_{i,j}$ as a set of p_i's corresponding to random variable A_j, resembling attribute A_j in the PDB.

Because $\boldsymbol{x}_i^n$ and $\boldsymbol{x}_i^{GT}$ can be seen as ensembles of parameters from categorical distributions, we should have that the distance metric d is a distance measure between probability

distributions. This implies that d should satisfy that the absolute difference between two categorical probability parameters is penalized more as the certainty of these parameters increases. As an example:

	Input		**Output**	
	A_1	A_2	A_1	A_2
x_1	0.9	0.1	0.8	0.2
x_2	0.8	0.2	0.7	0.3
x_3	0.8	0.2	0.9	0.1
x_4	0.7	0.3	0.8	0.2

TABLE 1: From input to output

The first tuple x_1 in the table above should receive a higher penalty than the second tuple x_2 and x_3 should receive a higher penalty than x_4, even though the eucledian distances are the same for both tuple pairs.

A distance metric that satisfies these properties is the *Kullback-Leibler* (KL) *divergence* [1]. This KL divergence is a measure of how one probability distribution is different from another probability distribution. For discrete probability distributions P and Q defined on the same probability space, the KL divergence is defined as follows:

$$D_{KL}\big(P||Q\big) = \sum_{x\in\Omega} P(x)\log\Big(\frac{P(x)}{Q(x)}\Big), \tag{2}$$

with Ω being the sample space. When applying this KL divergence to two discrete categorical distributions P and Q defined on the same probability space, both with n parameters, the KL divergence is evaluated as follows:

$$D_{KL}\big(P||Q\big) = \sum_{i=1}^{n} p_i\log\Big(\frac{p_i}{q_i}\Big), \tag{3}$$

with p_i and q_i being the i-th parameters of the probability distributions from P and Q respectively. A worked out example of Eq. (3) on table 1 can be found in the appendix, Sec. A.1.1. In order to quantify the distance between e.g. $\boldsymbol{x}_i$ and $\boldsymbol{x}_i^{GT}$, we can thus evaluate the KL divergence on each pair of attribute values $[a_{i,j}, a_{i,j}^{GT}], j = 1, 2, \ldots, M$ and add them all up together:

$$D_{KL}(\boldsymbol{x}_i||\boldsymbol{x}_i^{GT}) = \sum_{j=1}^{M} D_{KL}(a_{i,j}||a_{i,j}^{GT}) = \sum_{j=1}^{M}\sum_{k=1}^{K_j} p_i(C_{j,k})\cdot\log\Big\{\frac{p_i(C_{j,k})}{p_i^{GT}(C_{j,k})}\Big\}. \tag{4}$$

A disadvantage of using the KL divergence, however, is that $D_{KL}(P||Q)$ is only defined when $Q(x) = 0$ implies that $P(x) = 0$, which means that P has to be absolutely continuous with respect to Q [7]. This can cause problems as we don't have a guarantee that the records we apply the KL divergence on, are absolutely continuous with respect to each other. Furthermore, the KL divergence is asymmetric, as in general $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. A way to solve this problem is to use another (KL-based) divergence, the *Jensen-Shannon Divergence* (JSD) [4]. This divergence measure is a measure based on the Shannon Entropy H and is defined as

$$JSD_{\pi}\Big(P||Q\Big) = H(\pi_1 P + \pi_2 Q) - \pi_1 H(P) - \pi_2 H(Q), \tag{5}$$

where π is a set of weights $[\pi_1, \pi_2]$. When $\pi = [\frac{1}{2}, \frac{1}{2}]$, this is equivalent to

$$JSD_{\frac{1}{2}}\Big(P||Q\Big) = \frac{1}{2}D_{KL}\Big(P||M\Big) + \frac{1}{2}D_{KL}\Big(Q||M\Big), \tag{6}$$

with $M = \frac{1}{2}(P+Q)$. A proof for this can be found in the appendix, Sec. A.1.2. We use the $\text{JSD}_{\frac{1}{2}}$ divergence as defined in Eq. (6) and call it just 'JSD' from now on. What's more, when evaluating e.g. $JSD(\boldsymbol{x}_i||\boldsymbol{x}_i^{GT})$, this is calculated via attribute-wise summing, just as in Eq. 4:

$$\begin{aligned} JSD(\boldsymbol{x}_i||\boldsymbol{x}_i^{GT}) &= \sum_{j=1}^{M} JSD(a_{i,j}||a_{i,j}^{GT}) \\ &= \frac{1}{2}\sum_{j=1}^{M}\sum_{k=1}^{K_j} p_i(C_{j,k}) \cdot \log\Big\{\frac{p_i(C_{j,k})}{m_i(C_{j,k})}\Big\} + p_i^{GT}(C_{j,k}) \cdot \log\Big\{\frac{p_i^{GT}(C_{j,k})}{m_i(C_{j,k})}\Big\}, \end{aligned} \tag{7}$$

with $m_i(C_{j,k}) = \frac{1}{2}\big\{p_i(C_{j,k}) + p_i^{GT}(C_{j,k})\big\}$.

This means that the JSD divergence is just a smoothed version of the KL divergence. This divergence measure satisfies the properties mentioned earlier and moreover is symmetric and is numerically stable since it does not require P and Q to be absolutely continuous with respect to each other.

2.3.3 Data quality improvement and distance minimization

Based on the notion of data quality improvement and the distance metric above, improving data quality thus means that for a given update of $\boldsymbol{x}_i \in \mathcal{D}_{PDB} : \boldsymbol{x}_i \rightarrow \boldsymbol{x}_i^n$, we want that $d\big(\boldsymbol{x}_i^n, \boldsymbol{x}_i^{GT}\big) < d\big(\boldsymbol{x}_i, \boldsymbol{x}_i^{GT}\big) \implies JSD\big(\boldsymbol{x}_i^n||\boldsymbol{x}_i^{GT}\big) < JSD\big(\boldsymbol{x}_i||\boldsymbol{x}_i^{GT}\big)$. When we are talking about 'improving the data quality of data residing in a PDB', we thus mean that the quality has improved once the average JSD distance over $\mathcal{D}_{PDB}$ has improved, that is

$$\frac{1}{N}\sum_{i=1}^{N} JSD\big(\boldsymbol{x}_i^n||\boldsymbol{x}_i^{GT}\big) < \frac{1}{N}\sum_{i=1}^{N} JSD\big(\boldsymbol{x}_i||\boldsymbol{x}_i^{GT}\big). \tag{8}$$

3 Probabilistic model

As mentioned in Sec. 2.2, we can regard $\mathcal{D}_{PDB}$ as a corrupted version of data sampled from an underlying data generating distribution $P(\mathcal{D}_{GT})$. In this section, we describe how we can exploit this notion to construct a probabilistic model - hereafter called '*Probabilistic Inference Bayesian Network*' (PIBN) model - that can be used to achieve data quality improvement via a probabilistic modelling approach in a supervised setting ($P(\mathcal{D}_{GT})$ is known). As this model is built around the notions of *probabilistic inference* in a BN based on *virtual evidence*, we will first explain those concepts after which we explain how these concepts can be used to achieve data quality improvement. We will then end this section by concluding that this model is useful for insight in data quality improvement and useful for comparison, but cannot be used in an unsupervised setting. Based on this conclusion, we will then propose to develop a different model which *can* be used in an unsupervised setting and uses knowledge from the PIBN model.

3.1 Bayesian Network

As mentioned in Sec. 2.2, $P(\mathcal{D}_{GT})$ is a multivariate joint probability distribution over the discrete random variables $A_1, A_2, \ldots, A_M$, resembling the attributes of the PDB. By repeatedly using the *product rule of probability* (called *factorization*), we obtain the following expression for this joint probability distribution:

$$P\big(\mathcal{D}_{GT}\big) = P(A_1, A_2, \ldots, A_M) = \\ P(A_M|A_1, \ldots, A_{M-1}) \cdot P(A_{M-1}|A_1, \ldots, A_{M-2}) \cdot \ldots \cdot P(A_2|A_1)P(A_1). \tag{9}$$

Such a data-generating distribution $P\big(\mathcal{D}_{GT}\big)$ can be well described by a *Bayesian Network* (BN), also called a *Belief Network*. A Bayesian Network is a couple (G, Ω), where $G = (\boldsymbol{V}, \boldsymbol{E})$ a directed acyclic graph (DAG) with each node $V \in \boldsymbol{V}$ representing a random variable and each edge $E \in \boldsymbol{E}$ representing the conditional dependence between its head and tail, defined by component Ω. By using such a graphical model and the factorization of the joint distribution in Eq. (9), we can express the joint distribution $P(\mathcal{D}_{GT})$ as follows:

$$P(\mathcal{D}_{GT}) = \prod_{k=1}^{M} P(A_k|pA_k), \tag{10}$$

where pA_k denotes the set of parents of node A_k. In other words, the value for the joint probability is just the product of each of the individual posterior probabilities defined by the BN.

An example of a Bayesian Network with discrete variables is depicted in Fig. 4.

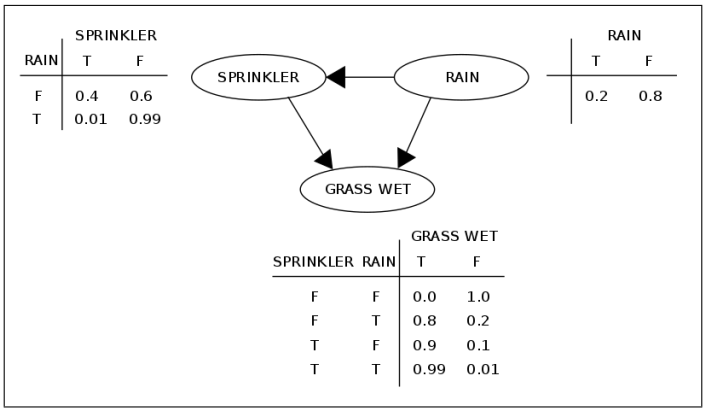


FIGURE 4: Simple Bayesian Network example

In this example, $P(R, S, G)$ can be modelled as

$$P(R, S, G) = \prod_{k=1}^{3} P(A_k|pA_k) = P(G|S, R) \cdot P(S|R) \cdot P(R).$$

The probability that the sprinkler is on whilst it does not rain and the grass is wet, can then be calculated as follows:

$$\begin{aligned} &P(S = T, R = F, G = T) \\ &= P(G = T|S = T, R = F) \cdot P(S = T|R = F) \cdot P(R = F) \\ &= 0.9 \cdot 0.4 \cdot 0.8 = 0.288. \end{aligned}$$

3.2 Probabilistic inference in Bayesian Networks with virtual evidence

Since the Bayesian Network is a network that fully describes the variables and their relationships, it can be used well to answer probabilistic queries about them. This is called *probabilistic inference*. Probabilistic inference on a BN is the process of computing the conditional probability $P(X = x|\boldsymbol{E} = \boldsymbol{e})$. This means that we want to determine the probability of r.v. X being in state x, given our observations (evidence) $\boldsymbol{e}$ for the set of r.v.'s $\boldsymbol{E}$ [5]. Probabilistic inference on graphical models is called *belief propagation* and was first proposed by J. Pearl [2], who formulated his algorithm as an exact inference algorithm on trees. Algorithms based on this that apply probabilistic inference in a discrete BN, do so by by first computing a secondary structure called the *join tree* (JT). This JT is used for propagating the evidence, which is called *join tree propagation* (JTP). Several of these exact algorithms exist for performing JTP, algorithms such as the Shafer-Shenoy algorithm [12], the Lauritzen-Spiegelhalter algorithm [3], the Hugin algorithm [6] and Lazy Propagation [8].

The aforementioned evidence on some variable X can come in many forms and shapes. In this paper, we distinguish *regular evidence* and *uncertain* evidence:

1. **Regular evidence**:
   Regular evidence on a variable X can be subdivided into multiple types [5]. A so-called *observation* is the knowledge that X definitely has a particular value. An observation comes with an evidence vector containing all 0's and one 1 corresponding to the state X is observed to be in. A *finding* is evidence that tells us that that X is definitely *not* in some state(s). The evidence vector contains 0's for the states we are sure X is not in, and 1's for the other states. This thus means that this evidence contains some uncertainty, as it does not specify in which state X must be, only in which it will not be.

2. **Uncertain evidence**:
   Uncertain evidence can be subdivided into *virtual evidence/likelihood evidence* (VE) [17] and *soft evidence* (SE) [9].

   - VE can be interpreted as evidence *with* uncertainty and a VE on variable A is represented by a likelihood ratio $L(A) = \big(P(obs|a_1) : \ldots : P(obs|a_n)\big)$ where $P(obs|a_i)$ denotes the probability of the observed event given A is in state a_i. Note that by definition, the elements of $L(A)$ thus not need to sum to 1.

   - SE can be interpreted as evidence *of* uncertainty and is represented as a probability distribution of one or more variables [16].

As this paper focuses on virtual evidence only (explained in Sec. 3.3), this notion is further explained by means of an example as can be found in the work of Mrad et al. [18]:

Example of virtual evidence, OCR system:
A Bayesian network includes a variable X representing a letter of the alphabet that the writer wanted to draw. The state space of X is the set of letters of the alphabet. A piece of uncertain information on X is received from a system of Optimal Character Recognition (OCR). The input of this system is an image of a character and the output is a vector of similarity between the image of the character and each letter of the alphabet. Let o represent the observed image. Consider a case where, due to lack of clarity, o can be recognized as either the letter 'v', 'u' or 'n'. The OCR technology provides the indices such

that $P(Obs = o|X = v) = 0.8$, $P(Obs = o|X = u) = 0.4$, $P(Obs = o|X = n) = 0.1$ and $P(Obs = o|X = x) = 0$ for any letter x other than 'u', 'v' or 'n'. This means that there is twice as much chance of observing o if the writer had wanted to draw the letter 'v' than if she had wanted to draw the letter 'u'. Such a finding on X is a VE on X, specified by $L(X) = (0 : ... : 0 : 0.1 : 0 : ...0 : 0.4 : 0.8 : 0 : 0 : 0 : 0)$.

This example illustrates that the prior probability distribution $P(X|pX)$ as defined by the BN includes the knowledge about the distribution of letters in the language of the text from which the character comes, whereas the OCR technology does not integrate that knowledge. In other words, it provides information about X without prior knowledge. In order to update the belief in the value of the character, the information provided by the OCR (being the likelihood vector) has to be combined with the prior knowledge of the frequency of letters.

3.3 Probabilistic inference and improving data quality

The question remains how the theory explained above is connected to the goal of this research. In other words, how is probabilistic inference in a BN connected to incorporating the data-dependencies from $\mathcal{D}_{GT}$ into each observation $\boldsymbol{x_i} \in \mathcal{D}_{PDB}$ such that the data quality is improved (Sec. 2.3)?

In fact, because of the specific kind of probabilistic nature of our data, each observation $\boldsymbol{x_i} \in \mathcal{D}_{PDB}$ exactly is a set of virtual evidences. Each attribute value $a_{i,j}$ provides a VE on attribute A_j and thus represents the likelihood vector on attribute A_j such that each $p_i(C_{j,k}) \in a_{i,j}$ is the likelihood that in the i-th observation, category $C_{j,k}$ is observed given that attribute A_j is in category C_k. Now, just as in the OCR example in Sec. 3.2, the beliefs in the values of the observed attributes in observation $\boldsymbol{x_i}$ can be updated by combining the likelihood vectors with the prior information defined by the BN, being the data-generating distribution where $\boldsymbol{x_i}$ is indirectly derived from.

For each parameter $p_i(C_{j,k})$ we thus update its value to the probability of that attribute A_j being in state $C_{j,k}$ given the evidence of the rest of the observation $\boldsymbol{x_i}$, that is

$$p_i(C_{j,k}) \rightarrow \hat{p}_i(C_{j,k}) = \underbrace{P(A_j = C_{j,k}|\underbrace{L(A_1, \ldots, A_M) = \boldsymbol{x_i}}_{\text{evidence}})}_{\text{defined by } P(\mathcal{D}_{GT})}, \tag{11}$$

where $L(A_1, \ldots, A_M)$ denotes the concatenation of the likelihood vectors for $A_1, \ldots, A_M$.

In other words, when the data-generating distribution $P(\mathcal{D}_{GT})$ is known, it can be described by a BN such that observation $\boldsymbol{x_i} \in \mathcal{D}_{PDB}$ can be updated by using a JTP algorithm to propagate the evidence into the BN, yielding - given the corruption requirement as mentioned in Sec. 2.3.1 - $\boldsymbol{x_i}^{new}$ that is closer to its corresponding clean value $\boldsymbol{x_i}^{GT}$. This process is repeated for each observation $\boldsymbol{x}_i$, where the evidence from the previous updates is erased from the BN. For each observation $\boldsymbol{x_i}$, we thus update its parameter values by propagating itself as evidence through the BN after which we extract the posterior distributions given the evidence. Pseudo-code for this can be found in Algorithm 1.

Algorithm 1: Record updating in the PDB via the PIBN model

Input: $\mathcal{D}_{PDB}$
Output: updated data $\mathcal{D}^n_{PDB}$

```
for every record x_i in D_PDB do
    Propagate the evidence defined by x_i through the BN;
    for every marginal probability p_i(C_{j,k}) in x_i do
        update p_i(C_{j,k}) → P(A_j = C_{j,k}|evidence) via probabilistic inference on the BN
          in which the evidence is propagated;
    end
    Erase evidence defined by x_i from the BN;
end
```

PDB probabilistic inference example:
As an example of the application of the PIBN model, let's say that the data-generating distribution $P(\mathcal{D}_{GT})$ is described as follows:

$$P(A) = \big[0.5, 0.5\big], \quad P(B|A) = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}, \quad P(C|A) = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}.$$

Now say that we have an observation $\boldsymbol{x}^{GT} = [1, 0, 1, 0, 1, 0]$, meaning that $(A, B, C) = (0, 0, 0)$ is observed. This observation is then extracted via a fuzzy extraction system such that we are not certain anymore whether its value for C was either 0 or 1, e.g., $\boldsymbol{x}^{GT} \rightarrow \boldsymbol{x} = [1, 0, 1, 0, 0.5, 0.5]$. By using an exact inference algorithm such as Lazy Propagation [8] to propagate this evidence[1], we obtain $\boldsymbol{x}^{new} = [1, 0, 1, 0, 0.9, 0.1]$. In this example, this solution can be computed and understood easily, as for example

$$\begin{aligned} &P\Big(C = 0\Big|L(A) = \{1, 0\}, L(B) = \{1, 0\}, C = \{0.5, 0.5\}\Big) = \\ &P\Big(C = 0\Big|L(A) = \{1, 0\}, L(B) = \{1, 0\}\Big) = \\ &P\Big(C = 0\Big|A = 0, B = 0) = P(C = 0|A = 0\Big) = 0.9, \end{aligned}$$

where the first equality follows as the likelihood for C doesn't favourize any state and the second to last inequality follows as C is conditionally independent of B given A, $(C \perp\!\!\!\perp B) \mid A$. For an indication, in Table 2 one can find several other update scenario's with the same data-generating distribution as above, together with the corresponding Jensen-Shannon divergence before and after the update.

GT	Corrupted	New	JSD before	JSD after
[1, 0, 1, 0, 1, 0]	[1.0, 0.0, 0.2, 0.8, 1.0, 0.0]	[1.0, 0.0, 0.69, 0.31, 1.0, 0.0]	0.4228	0.1207
[0, 1, 1, 0, 1, 0]	[0.5, 0.5, 1.0, 0.0, 1.0, 0.0]	[0.98, 0.02, 1.0, 0.0, 1.0, 0.0]	0.2158	0.6361
[1, 0, 1, 0, 1, 0]	[1.0, 0.0, 0.7, 0.3, 0.8, 0.2]	[1.0, 0.0, 0.95, 0.05, 0.97, 0.03]	0.1922	0.0255
[0, 1, 0, 1, 0, 1]	[0.0, 1.0, 0.5, 0.5, 0.5, 0.5]	[0.0, 1.0, 0.2, 0.8, 0.1, 0.9]	0.4315	0.1109
[1, 0, 1, 0, 0, 1]	[0.5, 0.5, 0.5, 0.5, 0.5, 0.5]	[0.5, 0.5, 0.55, 0.45, 0.5, 0.5]	0.6473	0.6206

TABLE 2: Record updates via the PIBN model

Note that in most cases, the end-result is closer to the ground-truth than it was before. An exception is the second record, where we see that the average JSD has increased. As the probability of observing $(A = 0, B = 0, C = 0) = 0.405$ is much larger than the probability of observing $(A = 1, B = 0, C = 0) = 0.01$ (Sec. 2.3.1).

[1]We used **aGrUM/pyAgrum** [19] for Lazy Propagation inference

3.4 Data quality improvement in an unsupervised setting

Now we have theoretically defined what it means to improve the data quality in a PDB and built a probabilistic model that can be used for data quality improvement given that we know its underlying data-generating distribution $P(\mathcal{D}_{GT})$, we wish to apply this knowledge to the real-life, unsupervised case where we do not know this $P(\mathcal{D}_{GT})$.

Because in such a case we only posses $\mathcal{D}_{PDB}$, we cannot directly apply the probabilistic inference as defined by Eq. (11). Given that the corruption of $\mathcal{D}_{PDB}$ is small compared to its corresponding GT data $\mathcal{D}_{GT}$, we can try to estimate $P(\mathcal{D}_{GT})$ from $\mathcal{D}_{PDB}$, but for the PIBN approach this requires us first to transform the probabilistic data $\mathcal{D}_{PDB}$ to non-probabilistic data. Besides being unclear how this latter step can be performed (if it even makes sense), we encounter the problem that when using Bayesian inference, estimating a BN from data quickly becomes computationally intractable as the number of latent variables and their dimensionalities increase. This is also the main disadvantage of the proposed PIBN model, as in the unsupervised case we need to estimate $P(\mathcal{D}_{GT})$ by constructing a BN from $\mathcal{D}_{PDB}$ (*structure learning*), which can become computationally intractable (in fact it is NP-hard [10]), let alone exact/approximate inference in a BN [20].

In order to overcome this difficulty and to find a proper approach to the problem in case $\mathcal{D}_{PDB}$ is very complex, a solution might be to use approximate inference techniques such as *variational inference* or *Markov chain Monte Carlo* (MCMC) sampling, however we then run again into the problem that it is not straightforward how to do so when our data has a probabilistic nature.

It is for this reason that we propose a different approach, a model that *can* use probabilistic data as input and that does not assume to know $P(\mathcal{D}_{GT})$. This model is built around an autoencoder and will be explained in Sec. 4.

4 Autoencoder model

Because of the problematic requirements of the PIBN model in an unsupervised setting, we need to develop a model that can use the probabilistic data $\mathcal{D}_{PDB}$ directly without assuming to know $P(\mathcal{D}_{GT})$. We propose to do this by means of an autoencoder that uses the probabilistic data $\mathcal{D}_{PDB}$ as input and indirectly learns to capture the data dependencies from $P(\mathcal{D}_{GT})$ via $\mathcal{D}_{PDB}$.

4.1 Traditional autoencoder model

The *autoencoder* (AE) dealt with in this paper is a feedforward, non-recurrent neural network having an input layer, a number of hidden layers and an output layer with the same number of nodes as the input layer. The purpose of such an autoencoder is to reconstruct its input by means of learning the outputs to be the same as the inputs. This makes this autoencoder to be an unsupervised learning model since no prior knowledge about the data (i.e. in terms of targets) is required.

The autoencoder consists of an encoder $g(\cdot) : \mathbb{R}^K \rightarrow \mathbb{R}^L$ parameterized by ϕ and a decoder $f(\cdot) : \mathbb{R}^L \rightarrow \mathbb{R}^K$ parameterized by θ, where ϕ and θ represent the weights and biases of the neural network. The encoder $g_\phi(\cdot)$ is a deterministic mapping between the input $\boldsymbol{x} \in \mathbb{R}^K$ and a hidden or 'latent' representation $\boldsymbol{z} \in \mathbb{R}^L$, whereas the decoder $f_\theta(\cdot)$ deterministi-

cally maps the hidden representation $\boldsymbol{z} \in \mathbb{R}^L$ back to the autoencoder's output $\boldsymbol{x'} \in \mathbb{R}^K$, visualized in Fig. 5.

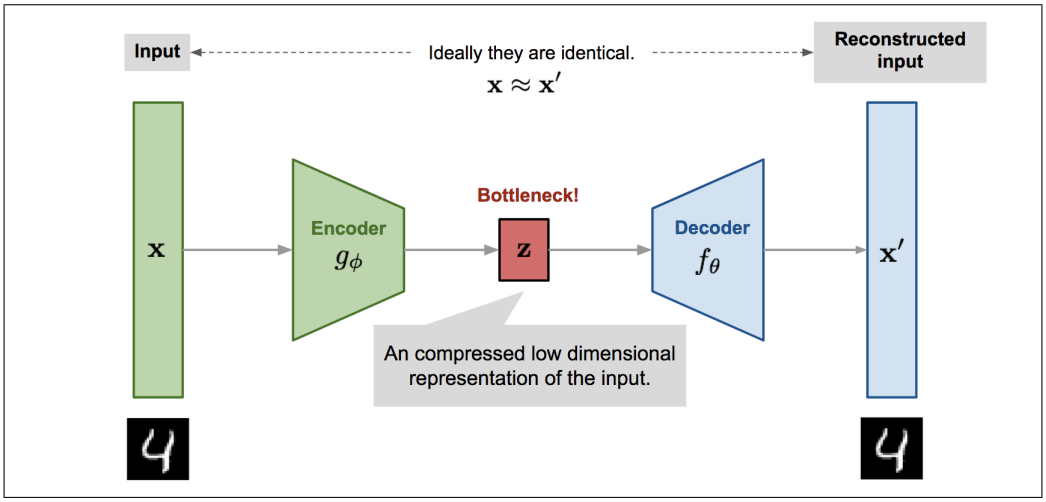


FIGURE 5: Basic autoencoder architecture

The autoencoder is trained by minimizing the reconstruction error/loss $\mathcal{L}$ with respect to the parameters $\boldsymbol{W} = [\phi, \theta]$ over the training data set:

$$\boldsymbol{W} = \arg\min_{\phi,\theta} \sum_{\boldsymbol{x} \in \boldsymbol{X}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{x'}) = \arg\min_{\phi,\theta} \sum_{\boldsymbol{x} \in \boldsymbol{X}} \mathcal{L}\Big\{\boldsymbol{x}, (f_\theta \circ g_\phi)\boldsymbol{x}\Big\}, \tag{12}$$

with $\boldsymbol{X} \in \mathbb{R}^{N \times K}$ being the training data set containing N observations. This optimization problem is solved by using the backpropagation of the loss (e.g. via gradient descent), just as in a regular neural network optimization problem.

4.2 Autoencoder and feature extraction

When no further restrictions are placed on the capacity of the AE, the AE will tend to learn the identity mapping from its input $\boldsymbol{x}$ to its output $\boldsymbol{x'}$. This means that the AE is just overfitting on the training data, making it a useless network as it is not generalizing. In order to make sure that the AE has a good reconstruction error on unseen data, we need the AE to learn a mapping $\boldsymbol{x} \to \boldsymbol{z}$ such that $\boldsymbol{z}$ is a good representation being robust to noise in $\boldsymbol{x}$.

For a representation to be *good*, we need this representation to at least retain a significant amount of information about the input. In information-theoretic terms, this means that the mutual information $\mathbb{I}(X, Z)$ between input random variable X and its corresponding constructed hidden representation Z is maximized. As shown by Vincent et al. [14], an AE is exactly doing that when being trained to minimize the reconstruction error, since it is maximizing a lower bound on this mutual information. In other words, when an AE is trained to minimize the reconstruction error of input X, it has learned to retain as much of the information of X as possible.

Only this criterion however, is not enough for the mapping to be able to separate noisy details from the useful information, or in other words, distinguish the important data-dependencies from the noise in the data. As mentioned above, the mutual information $\mathbb{I}$ can simply be maximized by learning the identity mapping. We also need the mapping

to be robust to noise, meaning that the representations $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ for inputs $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ respectively yield a similar reconstruction when $\boldsymbol{x}_2$ is a slightly corrupted version of $\boldsymbol{x}_1$. This robustness can be incorporated in several ways, of which the most popular methods are as follows:

- **Undercomplete AE**: by making the dimension L of the middle hidden layer ('*bottleneck*') smaller than the input dimension K, $\boldsymbol{z}$ becomes a compressed representation of input $\boldsymbol{x}$, such that not all information can be retained, meaning that a good reconstruction requires $\boldsymbol{z}$ to capture the most important information. This is the standard method and is depicted in Fig. 5.
- **Sparse AE** [13]: also called the *overcomplete autoencoder*, this AE has a hidden layer with a dimensionality at least the input dimensionality K, but adds a sparsity constraint to the reconstruction loss $\mathcal{L}$: $Loss = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x'}) + \Omega(\boldsymbol{z})$, where Ω is an increasing function of the average activity of the nodes in $\boldsymbol{z}$, encouraging less nodes to be active.

However, another very interesting approach, is the so-called **Denoising AE** (DAE) proposed by Vincent et al. [14]. In this set-up, each input observation $\boldsymbol{x}$ is corrupted[2] into $\tilde{\boldsymbol{x}}$ via stochastic mapping $\tilde{\boldsymbol{x}} \sim q(\tilde{\boldsymbol{x}}|\boldsymbol{x})$, which is specified in Sec. 4.6. The model is then trained to minimize the difference between the output $\boldsymbol{x'}$ corresponding to input $\tilde{\boldsymbol{x}}$ and the corresponding clean version $\boldsymbol{x}$. That is:

$$\boldsymbol{W} = \arg\min_{\phi,\theta} \sum_{\boldsymbol{x}\in\boldsymbol{X}} \mathcal{L}\Big\{\boldsymbol{x}, (f_\theta \circ g_\phi)\tilde{\boldsymbol{x}}\Big\}. \tag{13}$$

In this set-up, the hidden representation $\boldsymbol{z}$ is thus a result of the deterministic mapping $g_\phi(\tilde{\boldsymbol{x}})$ rather than $g_\phi(\boldsymbol{x})$. By doing such, the DAE learns to clean partially corrupted input, which results in a better hidden representation $\boldsymbol{z}$ that can be used for denoising, a property that can be used to improve the data quality of our input data $\mathcal{D}_{PDB}$. In this set-up, the definition of a good representation can be reformulated as: *a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input*[14]. The two ideas that are implicit in this approach are:

- A higher level representation should be rather stable and robust under corruptions of the input.
- It is expected that by performing a denoising task, the hidden layer should extract features that capture the useful structure of the data generating distribution of the input data.

Note that the given definition above fits exactly our purpose of improving the data quality in the PDB. It is for this reason that we chose to use the DAE structure as a method for feature extraction.

4.3 DAE and data quality improvement in a PDB

The central goal of this research is to increase the quality of the data residing in a PDB, which requires to capture the dependencies in $P(\mathcal{D}_{GT})$. As mentioned in Sec. 4.2, the DAE can be used to capture the data dependencies of its input data. This means that when the corruption in $\mathcal{D}_{PDB}$ is relatively low, using $\mathcal{D}_{PDB}$ as input to the DAE, the DAE

[2]One should note that this is not the same as the corruption of a record with respect to its corresponding ground truth version as mentioned in Sec. 2.2

indirectly learns the data dependencies in $P(\mathcal{D}_{GT})$. In other words, by training the DAE on $\mathcal{D}_{PDB}$, the DAE should be able to learn a good hidden representation $\boldsymbol{z}$ for each input $\boldsymbol{x}$ that works denoising and can be used to bring the marginal probabilities $p_i(C_{j,k})$ closer to their corresponding ground truth values. As taking probability parameters as input to the AE with an output being of the same nature is not straightforward, we further explain this in Sec. 4.4.1.

4.4 Model input and output

An important part of model construction is thinking about the nature of the model's input and output. Since this paper deals with an autoencoder model, the nature of the input and output are the same, which means that the choice of the nature of the input data is fully depending on the desired output from the model, which at its turn should fit the purpose of the model's construction.

4.4.1 Probabilistic input and output

As explained in Sec. 4.3, the DAE can be used to bring the marginal probabilities $p_i(C_{j,k})$ closer to their corresponding ground truth value by making use of its denoising property. This thus requires that instead of static data as input and output, we will use the probability parameters themselves as input. In case of $\mathcal{D}_{PDB}$, this thus means that we take $\boldsymbol{x}_i \in \mathcal{D}_{PDB}$ as input.

Using this kind of probabilistic input, the autoencoder model outputs for each observation $\boldsymbol{x}_i$ a tensor $\boldsymbol{x}_i'$ with the same number of elements representing the marginal probabilities, however with these probabilities being massaged, differently distributed, which is a direct consequence of the dependencies and patterns in the entire data set $\mathcal{D}_{PDB}$, as well as a consequence of the corresponding input observation $\boldsymbol{x}_i$. Note that in fact this is similar to combining the prior information as defined by the underlying $P(\mathcal{D}_{GT})$ together with the information provided by the record itself, as was mentioned in Sec. 3.2.

4.4.2 Input implementation in the autoencoder model

Implementation-wise, the above means that the data from the probabilistic database needs to be compatible with a neural-network type of model. Such a model has an input layer consisting of D nodes where each node corresponds to a feature from a D-dimensional observation $[x_1, x_2, \ldots, x_D]$.

Based on the description in Sec. 4.4.1, this means that each category $C_{j,k}$ of attribute A_j should have a corresponding node in the input layer. In other words, the input is an ensemble of the parameters from categorical distributions (Sec. 2.3.2). This means that for each attribute A_j with K_j number of different categories, the model has K_j number of corresponding input nodes. In total, the model then has $\sum_{j=1}^{M} K_j$ number of input nodes. This idea is visualized in Fig. 6 by means of a simple example.

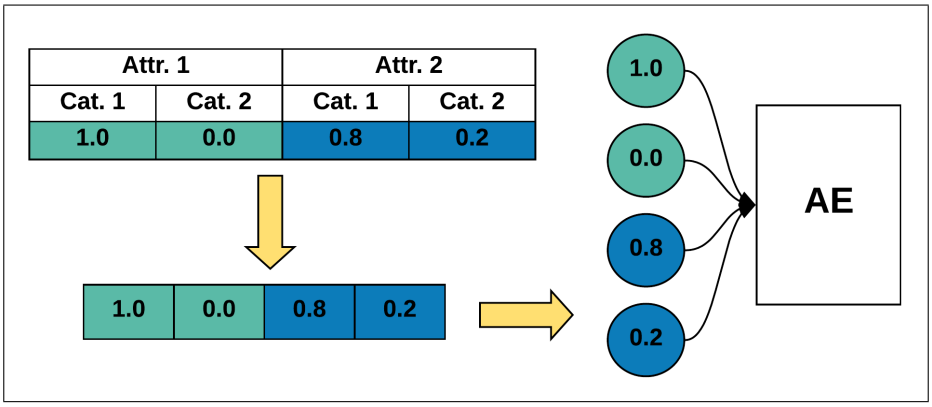


FIGURE 6: Probabilistic data input

4.4.3 Output implementation in the autoencoder model

As stated in Sec. 4.4.1, given an input $\boldsymbol{x}_i$, the output $\boldsymbol{x}_i'$ from the DAE should be of the same nature as its input, being an ensemble of parameters from categorical distributions. The nature of such an output requires that for each element $p_i'(C_{j,k}) \in \boldsymbol{x}_i'$ its value is between 0 and 1 and that the sum of the outputted probability parameters corresponding to attribute j equals 1, as motivated in Eq. (1). This constraint can be implemented into the autoencoder model by applying the Softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ to each set of output nodes corresponding to one and the same attribute. This function is element wise defined as follows

$$\sigma(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}. \tag{14}$$

It is easy to verify that the Softmax function takes a K-dimensional input and returns a K-dimensional input where each element $\sigma(\boldsymbol{x})_i$ is squeezed into an interval $[0, 1]$ and the sum of the elements equals 1. It is for this reason that the Softmax function can be used to make the output to represent the parameters a categorical distribution.

Applying these Softmax functions to the output of the autoencoder is visualized in Fig. 7 by means of an example.

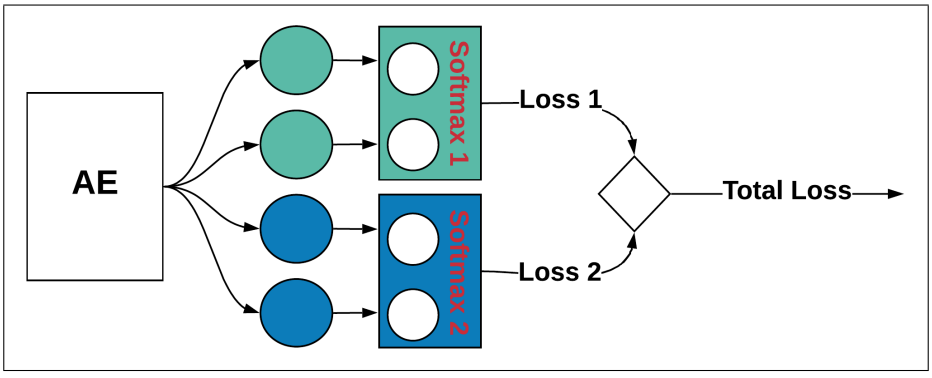


FIGURE 7: Example of a Softmax function on an AE output
as continuation on Fig. 6.

Each Softmax output $\in \mathbb{R}^{K_j}$ for attribute A_j is used as input parameter for a loss function

$\mathcal{L}_j$ that is a then combined with other losses to obtain one total loss which can then be used for training the model, as further explained in Sec. 4.5.

4.5 Loss function

In order to obtain a solid model performance, the model should be trained with a proper training criterion. Since the autoencoder is trained by minimizing a reconstruction error (Eq. 12), this means choosing a suitable loss function $\mathcal{L}$. As explained in Sec. 2.3.2, a proper distance metric for this would be the JSD, because of the nature of the output from the autoencoder model.

4.6 Data corruption

As mentioned in Sec. 4.2, the reconstruction loss of the DAE, $\mathcal{L}$, is a function of $\tilde{\boldsymbol{x}}$, where $\tilde{\boldsymbol{x}}$ is a corrupted version of $\boldsymbol{x}$ according to a stochastic mapping q: $\tilde{\boldsymbol{x}} \sim q(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. Popular corruption processes are:

- **Gaussian Noise**: $\tilde{\boldsymbol{x}}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}, \sigma^2 I)$
- **Salt and Pepper Noise**: a fraction v of the elements of $\mathbf{x}$ is set to their minimal (salt) or maximal (pepper) values, being 0 and 1 in our case.
- **Masking Noise**: a fraction v of the elements of $\boldsymbol{x}$ is set to 0.

These noise processes do not make much sense for our type of data, however, as for each observation $\boldsymbol{x_i}$ in $\mathcal{D}_{PDB}$, the condition should hold that for each attribute $a_{i,j}$ the sum of the probability parameters is 1, Eq. (1). It would make more sense to randomly select one or multiple attribute values $a_{i,j}$ from $\boldsymbol{x_i}$ and set all of its corresponding probability parameters to $1/K_j$, making $a_{i,j}$ a missing value, as explained earlier in Sec. 2.1. By doing such, the DAE has to learn to 'fill in the blanks' based on the values of the other attributes, or in other words, the DAE has to learn to predict the category-distribution for that particular attribute, based on the category-distributions of the other attributes.

This noise process can be parameterized by parameter v, denoting the fraction of attribute values in the data set that is corrupted. Hence a $v = 0.20$ means that 20% of all the attribute values are corrupted, i.e. set to a uniform distribution as explained above.

5 Evaluating and testing

In order to test the general performance of the DAE model, we need to evaluate its performance on unseen test-data. Since we defined 'improving data quality' as decreasing the distance between $\mathcal{D}_{PDB}$ and $\mathcal{D}_{GT}$ (Sec. 2.3), this means that we need to know the ground truth of the data in order to do so.

Evaluating the performance of the model is in particular important since we need to account for the values of the hyper-parameters of the model. In the case of the DAE model, this mostly applies to finding a good value for the corruption level v (Sec. 4.6) and the complexity of the DAE, that is, the number of hidden nodes in the hidden layer. What's more, the DAE is trained by minimizing the reconstruction loss $\mathcal{L}$, Eq.(13). This, however, gives us no guarantee that the learned mapping $\boldsymbol{x} \rightarrow \boldsymbol{z} \rightarrow \boldsymbol{x'}$ is indeed a mapping that satisfies the properties as specified in Sec. 4.2. The model may for instance be overfitting on training data (as a consequence of a too high complexity of the autoencoder) or on the

contrary may be underfitting (as a consequence of a too low complexity of the autoencoder).

As already explained in Sec. 3.4, in a real world application, we do not posses the ground truth $\mathcal{D}_{GT}$ for $\mathcal{D}_{PDB}$ and have to unsupervisedly develop the model and account for a choice of architecture and the tuning of hyper-parameters of the model. This is done however, based on results from supervised model construction in a similar case in an earlier stage. By creating *synthetic data sets*, the DAE model can be tested on data sets of which the nature of the data dependencies is known.

It is for this reason that we first develop a way to evaluate the performance of the DAE in a supervised setting. Thereafter we evaluate the performance of the DAE model on several synthetic data sets. What's more, we compare its performance with the performance of the PIBN model that assumes to know $P(\mathcal{D}_{GT})$.

5.1 Evaluation structure

In order to be able to evaluate the performance of the DAE model and to select the best possible hyper-parameters in such a supervised setting, the following evaluation structure as depicted in Fig. 8 is used:

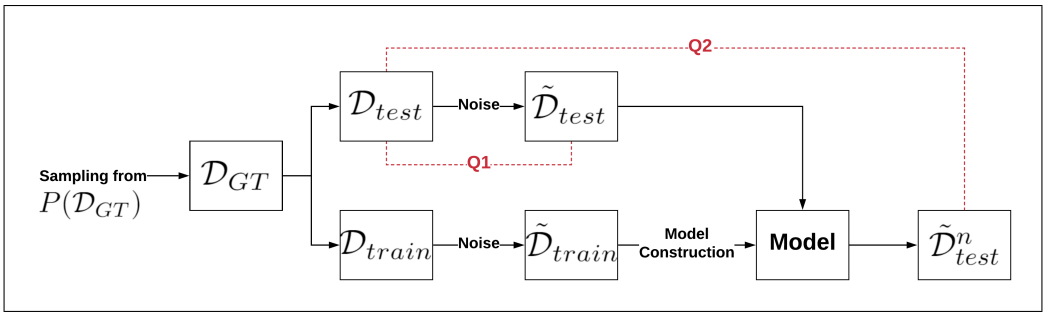


FIGURE 8: Evaluation Process

1. **Step I, GT data set**:
   Data set $\mathcal{D}_{GT}$ is generated according to sampling from the data generating distribution $P(\mathcal{D}_{GT})$, as explained in Sec. 2.2. This $P(\mathcal{D}_{GT})$ is explicitly defined via a BN, which is modeled by using the PyAgrum [22] library in Python. An example of this implementation (both the BN construction and the sampling process) can be found in the appendix, Sec. A.2.

2. **Step II, train and test set + corruption**:
   The $\mathcal{D}_{GT}$ is partitioned into a training and test set, $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ respectively. Those two data sets are then both corrupted with the same type (structure and intensity) of noise, leading to two corrupted data sets $\tilde{\mathcal{D}}_{train}$ and $\tilde{\mathcal{D}}_{test}$. This noise represents the uncertainty as residing in a PDB. More about this proces and the type of noise can be found in Sec. 5.2.
   Note: this is not the same as the deliberate corruption in the DAE training process!

3. **Step III, model training**:
   The DAE model is trained based on $\tilde{\mathcal{D}}_{train}$. It does so by minimizing the reconstruction loss $\mathcal{L}$, Eq. (13), meaning the autoencoder gets $\tilde{\mathcal{D}}_{train}$ as input which it tries to reconstruct. When the amount of noise in $\tilde{\mathcal{D}}_{train}$ relative to its corresponding ground truth $\mathcal{D}_{train}$ data set is relatively low, the idea is that by training in this way,

the autoencoder will not be influenced much by this noise and will thus indirectly capture the data dependencies as residing in $\mathcal{D}_{train}$ (and thus $\mathcal{D}_{GT}$, given that $\mathcal{D}_{train}$ is large enough).

4. **Step IV, test set mapping**:
   The DAE model is applied to the corrupted test data $\tilde{\mathcal{D}}_{test}$ yielding a new data set $\tilde{\mathcal{D}}^n_{test}$.

5. **Step V, distance measures**:
   In this step, the quality measures $Q_2 = Q\big(\mathcal{D}_{test}, \tilde{\mathcal{D}}^n_{test}\big)$ is evaluated on $\tilde{\mathcal{D}}^n_{test}$ with ground truth $\mathcal{D}_{test}$ as reference point. Furthermore, this quality measure is compared with $Q_1 = Q\big(\mathcal{D}_{test}, \tilde{\mathcal{D}}_{test}\big)$ to see if the quality of the data has improved. This step is crucial in evaluating the model performance. We use the following types of quality measures:

   (a) The average distance measured by the JSD, as given by Eq. (8). This average distance is calculated both on the entire data sets as well as only on the corrupted records in that data set, denoted with '*JSD*' and '*JSD cr*' respectively.

   (b) Another interesting quality measure Q is a measure that measures the fraction of cases in which the output $\tilde{\boldsymbol{x}}^n_i \in \tilde{\mathcal{D}}^n_{test}$ from the autoencoder on input $\tilde{\boldsymbol{x}}_i \in \tilde{\mathcal{D}}_{test}$ has diverged from the ground truth $\boldsymbol{x}_i \in \mathcal{D}_{test}$:

   $$Q = \frac{1}{|I|} \sum_{i \in I} [JSD(\boldsymbol{x}_i || \tilde{\boldsymbol{x}}^n_i) > JSD(\boldsymbol{x}_i || \tilde{\boldsymbol{x}}_i)], \tag{15}$$

   where $[\cdot]$ represent the Iverson brackets. What's more, we would like to know the average divergence for those 'false updates'. These two properties are both calculated for the entire data set as well as for only the corrupted records, denoted with '*Faul*' and '*Faul cr*' respectively.

   (c) Finally, a measure that can be used in an unsupervised setting as well, is a measure for the total uncertainty in the data set. For this we can perfectly use the Shannon Entropy H. Since in our type of data, A_j can be seen as a categorical random variable, the uncertainty of each observation x_i in our data sets can be measured by summing the entropy of its attributes $a_{i,j}$:

   $$H(\boldsymbol{x}_i) = \sum_{j=1}^{M} H(a_{i,j}) = -\sum_{j=1}^{M} \sum_{k=1}^{K_j} p_i(C_{j,k}) \log_b \Big(p_i(C_{j,k}) \Big). \tag{16}$$

   We do this for every record $\boldsymbol{x}_i$ in the data set and sum to obtain the total uncertainty residing in the data set.

5.2 Uncertainty parameterization

Within the domain of attribute uncertainty (Sec. 2.1), there is still a lot of freedom in which uncertainty can be distributed among the (synthetic) data. This is why we specify the 'corruption process' as mentioned in Sec. 2.2 by means of the following parameters:

- How many records of the PDB contain uncertainty?
  - $0 \leq \lambda \leq 1$: the fraction of records that contains uncertainty.
- Is the uncertainty in such a record only with respect to one or multiple attributes?

- $0 \leq\ n_a \leq M$: the number of attributes that contain uncertainty.

- Are these attributes the same for all the uncertain records or is this fully random?
  - $R \in \{\text{True, False}\}$: governing whether the attribute(s) containing uncertainty is the same for each uncertain record (R = False) or is pure random (R=False).

- How much noise is added per attribute value $a_{i,j}$?
  - $0 \leq \epsilon \leq 1$: the amount of noise distributed over the other categories of $a_{i,j} = [p_{i,j}(C_{j,1}), p_{i,j}(C_{j,2}), \ldots, p_{i,j}(C_{j,K_j})]$.

- How is the uncertainty in a record distributed among the attribute(s), that is, are the parameters of $a_{i,j}$ nearly all uniformly distributed or is this distribution more skewed, and how much? This can be implemented by using the Dirichlet distribution [15] which in fact is the conjugate prior of the categorical distribution:

$$\frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}. \tag{17}$$

  This distribution is used to get a tuple of observations $\boldsymbol{x} = [x_1, x_2, \ldots, x_K]$ summing to 1. This tuple is then multiplied with ϵ to obtain how the noise is spread over the other K categories. The vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_K]$ is used to define how skewed the distribution over $\boldsymbol{x}$ is. If all elements of $\boldsymbol{\alpha}$ are equal, this means that the distribution is symmetric and $\boldsymbol{\alpha}$ can be written as $\boldsymbol{\alpha} = \alpha \cdot [1, 1, \ldots]$ and the Dirichlet distribution is then fully parameterized by the scalar α. What's more, a small value of $\alpha < 1$ yields a sparse (skewed) distribution over $\boldsymbol{x}$ whereas a large value of $\alpha > 1$ yields dense (more uniform) distribution over $\boldsymbol{x}$.
  - $0 < \alpha < \infty$: governing the skewness of the distribution of noise over the other categories.

5.3 Performance on synthetic data sets

In this section we will test the performance of the DAE model on several synthetic data sets. We compare its performance with the performance of the PIBN model assuming that it knows $P(\mathcal{D}_{GT})$. As mentioned in Sec. 5.1, the synthetic data sets are sampled from a data generating distribution $P(\mathcal{D}_{GT})$ defined by a BN. Therefore, for each of the data-sets below, the structure of the corresponding $P(\mathcal{D}_{GT})$ can be found in the appendix, Sec. A.3. What's more, in each case:

1. $|\mathcal{D}_{GT}| = 10,000$, $|\mathcal{D}_{train}| = 9,000$, $|\mathcal{D}_{test}| = 1,000$.
2. Settings for the DAE model:
   The implementation is done in Python with the *Functional API* from *Keras*. The used hyper-parameters for the DAE model are as follows:

Parameter	**Parameter value**
Optimizer	Stochastic Gradient Descent
Learning rate	$5e-3$
Decay	$1e-6$
Momentum	0.9
Nesterov	True
Batch-size	20
Epochs	100
Activation hidden layer	RELU
DAE corruption v	0.30

TABLE 3: DAE (training) hyper-parameter values

In this paper, we test the performance of the DAE and PIBN model on 5 synthetic data sets that are all corrupted according to the same corruption structure. Information on the type of corruption and the statistical properties of test data sets before application of the models can be found in Table 4.

Corruption settings: $\lambda = 0.20, n_a = 2, R = \text{True}, \epsilon = 0.5, \alpha = 5$			
Data set	**JSD**	**JSD cr**	**Entropy**
$\mathcal{D}_1$	0.03819	0.21576	0.23750
$\mathcal{D}_2$	0.04315	0.21576	0.30299
$\mathcal{D}_4$	0.04272	0.21576	0.30870
$\mathcal{D}_3$	0.04143	0.21576	0.27553
$\mathcal{D}_5$	0.04294	0.21576	0.32648

TABLE 4: Corruption settings & statistical properties of the used synthetic data sets

After applying the DAE and PIBN model, we obtain the results[3] as can be found in Table 5 and Table 6, respectively. For the JSD quality measure, we used e as base of the *log* function. For the Entropy measure, we used 2 as base.

	DAE model				
	JSD	*JSD cr*	*Entropy*	*Faul*	*Faul cr*
$\mathcal{D}_1$	0.00569	0.03164	0.03558	3.8% - 0.10759	6.2% - 0.36401
$\mathcal{D}_2$	0.00903	0.04378	0.06515	6.5% - 0.12498	6.5% - 0.56834
$\mathcal{D}_3$	0.00923	0.04524	0.09169	14.9% - 0.04072	3.0% - 0.78620
$\mathcal{D}_4$	0.01028	0.04712	0.08016	14.6% - 0.07718	8.9% - 0.53040
$\mathcal{D}_5$	0.01468	0.06266	0.07641	16.6% - 0.20968	9.6% - 1.59828

TABLE 5: Quality measures of updated data sets after application of the DAE model

[3]Because the DAE model contains randomness in the training process, the results might slightly vary each time the model is trained. Because of lack of time, we could not include a confidence interval for those results. This is an action point for future work.

	PIBN model				
	JSD	*JSD cr*	*Entropy*	*Faul*	*Faul cr*
$\mathcal{D}_1$	0.00673	0.03802	0.04029	0.6% - 0.33316	3.4% - 0.33316
$\mathcal{D}_2$	0.00997	0.04987	0.06904	0.9% - 0.55640	4.5% - 0.55640
$\mathcal{D}_3$	0.01370	0.06919	0.12581	2.6% - 0.16462	13.1% - 0.16462
$\mathcal{D}_4$	0.01188	0.06186	0.08531	1.1% - 0.94218	5.7% - 0.94218
$\mathcal{D}_5$	0.01217	0.06115	0.06554	2.1% - 0.8273	10.6% - 0.82730

TABLE 6: Quality measures of updated data sets after application of the PIBN model

In Table 5 and Table 6, we can see multiple interesting things:

1. First of all, we note that for each of the data sets it is true that, after update via the DAE or PIBN model, it has become closer to its corresponding ground truth, *meaning that the data quality has been improved*. In addition to this, we see that the uncertainty (measured by the entropy H) has been decreased for each of the data sets, which is as expected because of our observation of a decreased difference between the data sets and their ground truth counterparts.

2. Secondly, we see that as the data sets get more complex (Sec. A.3), the DAE model has a higher '*faul*' and '*faul cr*' ratio, which is true till a lesser extent for the PIBN model. This is as expected, as the denoising autoencoder has to learn a more complicated dependency structure with relatively more occurrences of observations with a low probability of occurrence. What's more, with a more complicated dependency structure comes an increasing probability of updating a record 'into the wrong direction' (Sec. 2.3.1).

3. Thirdly, we see that the DAE model slightly outperforms the PIBN model on data sets $1-4$, whereas the performance of the PIBN model is better on data set 5. We suspect that this is a consequence of the DAE model being slightly overfitting on the data, as for these data sets with a relatively simple underlying generating probability distribution, the data in the test set is quite similar[4] to the data on which the model is trained, yielding a very good result for the test set when the data is overfitting on the train set. Because data set 5 is of a more complex structure, overfitting on the train set will give a worse performance on the test set. To prevent this overfitting even more, better values should be picked for the hyper-parameters, which is an action point for future research.

4. Finally, we observe that both the '*faul*' and '*faul ocr*' rates for the DAE model are higher than those of the PIBN model. However, if we look at the average JSD of those 'false updates', we see that this is much lower for the DAE model. This is because the DAE model may slightly increase the uncertainty in 'certain records' (records that are not corrupted in the corruption process and are thus identical to the ground truth), such that this update is counted as a faul, even though the divergence is not much (low average JSD for these records). The PIBN model will not do this and thus has much lower faul rates.

[4]Every record in the test set is also present in the train set

6 Conclusion and future work

As the results show, in this thesis we successfully developed a model that can achieve data quality improvement in a PDB. This model, the DAE model, does so by learning to capture the most important data dependencies from the underlying ground truth data generating probability distribution $P(\mathcal{D}_{GT})$ that are indirectly present in $\mathcal{D}_{PDB}$. The latter is achieved by a regularization type in the form of deliberately adding noise to the data on which the AE is trained. Before we could develop such a model, we first described what it means to improve data quality and how this can be quantified. Once this was established, we developed a probabilistic model, the PIBN model, that manages to achieve data quality improvement given that we know $P(\mathcal{D}_{GT})$. In the development process of this model, we saw that data quality improvement for a single record $\boldsymbol{x}_i$ can be achieved by combining the evidence provided by that record together with the prior information defined by $P(\mathcal{D}_{GT})$. This notion helped us to establish the DAE model, which does a similar thing, but learns $P(\mathcal{D}_{GT})$ indirectly via $\mathcal{D}_{PDB}$.

We are aware that our results and efforts are just the tip of the iceberg, a first step in the process of data quality improvement in a PDB. To substantiate this, we end this paper by mentioning the most important action points for future work that follows from this thesis:

1. **Statistical data type assumption:**
   In this thesis, we focused on categorical nominal data only. This has a huge impact on the approach to data quality improvement. Using other statistical data types - say, numerical data - totally changes how a PDB is modeled; in such a case, we cannot regard a record $\boldsymbol{x}_i \in \mathcal{D}_{PDB}$ as an ensemble of parameters from categorical distributions. This has as an effect that we can not use those parameters as input to an AE. In such a case it is required to come up with a totally different approach. As the data in probabilistic data integration is often not categorical nominal but categorical ordinal or numeric (or both), it might be interesting to do research on how one would approach data quality improvement given such conditions.

2. **AE Regularization and hyper-parameter tuning:**
   When testing the performance of the DAE model on the synthetic data sets, we used a specific setting for the hyper-parameters of the AE and its training process, as shown in Table 3. We did not thoroughly test for the best parameter values because this is not in the scope of this thesis, although this is very important for improvement of the developed models. As an example, evaluating the model performances as the corruption fraction v (Sec. 4.6) changes (and how this is dependent on the particular data set), may give very useful insights in how the DAE model learns to capture the most important dependency structures and how this influences whether the model is overfitting or not.

3. **Data quality improvement and data set properties:**
   Last but not least, a very interesting action point is to do research on the connection between the statistical properties of data sets and the performance of the proposed PIBN and DAE model on those data sets. As can be seen in Table 5 and Table 6, the performance seems to decrease as the data sets get more complex dependency structures (Sec. A.3). However, till now, this is merely an observation a a result of a small test-case, rather than a general truth. What's more, it is not clear how this 'complexity' can be quantified and qualified (e.g. in information theory terms). In

order to use the DAE model, it may be very useful to be able to know more about this connection so that better claims can be made about the model's performance.

References

[1] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[2] Judea Pearl. *Reverend Bayes on inference engines: A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science . . ., 1982.

[3] Steffen L Lauritzen and David J Spiegelhalter. "Local computations with probabilities on graphical structures and their application to expert systems". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 50.2 (1988), pp. 157–194.

[4] J. Lin. "Divergence measures based on the Shannon entropy". In: *IEEE Transactions on Information Theory* 37.1 (Jan. 1991), pp. 145–151. DOI: 10.1109/18.61115.

[5] Cecil Huang and Adnan Darwiche. "Inference in belief networks: A procedural guide". In: *International journal of approximate reasoning* 15.3 (1996), pp. 225–263.

[6] Finn V Jensen. "Bayesian updating in causal probabilistic networks by local computations". In: *An introduction to Bayesian networks* (1996).

[7] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[8] Anders L Madsen and Finn V Jensen. "Lazy propagation: a junction tree inference algorithm based on lazy evaluation". In: *Artificial Intelligence* 113.1-2 (1999), pp. 203–245.

[9] Marco Valtorta, Young-Gyun Kim, and Jirka Vomlel. "Soft evidential update for probabilistic multiagent systems". In: *International Journal of Approximate Reasoning* 29.1 (2002), pp. 71–106.

[10] David Maxwell Chickering, David Heckerman, and Christopher Meek. "Large-sample learning of Bayesian networks is NP-hard". In: *Journal of Machine Learning Research* 5.Oct (2004), pp. 1287–1330.

[11] Alan C Acock. "Working with missing values". In: *Journal of Marriage and family* 67.4 (2005), pp. 1012–1028.

[12] Prakash P Shenoy and Glenn Shafer. "Axioms for probability and belief-function propagation". In: *Classic Works of the Dempster-Shafer Theory of Belief Functions*. Springer, 2008, pp. 499–528.

[13] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[14] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of machine learning research* 11.Dec (2010), pp. 3371–3408.

[15] "Dirichlet Distribution". In: *Dirichlet and Related Distributions*. John Wiley & Sons, Ltd, 2011. Chap. 2, pp. 37–96. ISBN: 9781119995784. DOI: 10.1002/9781119995784.ch2.

[16] Ali Mrad et al. "Uncertain Evidence in Bayesian Networks : Presentation and Comparison on a Simple Example". In: vol. 299. July 2012, pp. 39–48. DOI: 10.1007/978-3-642-31718-7_5.

[17] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[18] Ali Ben Mrad et al. "An explication of uncertain evidence in Bayesian networks: likelihood evidence and probabilistic evidence". In: *Applied Intelligence* 43.4 (2015), pp. 802–824.

[19] Christophe Gonzales, Lionel Torti, and Pierre-Henri Wuillemin. "aGrUM: a Graphical Universal Model framework". In: *International Conference on Industrial Engi-*

neering, Other Applications of Applied Intelligent Systems. Proceedings of the 30th International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems. Arras, France, June 2017. URL: https://hal.archives-ouvertes.fr/hal-01509651.

[20] Johan Kwisthout. "Approximate inference in Bayesian networks: Parameterized complexity results". In: *International Journal of Approximate Reasoning* 93 (2018), pp. 119–131. ISSN: 0888-613X. DOI: https://doi.org/10.1016/j.ijar.2017.10.029. URL: http://www.sciencedirect.com/science/article/pii/S0888613X17306680.

[21] Maurice van Keulen. "Probabilistic Data Integration". English. In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Zomaya. Springer, Feb. 2018. DOI: 10.1007/978-3-319-63962-8_18-1.

[22] Pierre-Henri Wuillemin Christophe Gonzales. *pyAgrum*. URL: https://agrum.gitlab.io/pages/pyagrum.html.

A Appendix

A.1 Model training and validation

A.1.1 Loss function

When using e as base for the logarithm, we obtain:

$$
\begin{aligned}
D_{KL}(\boldsymbol{x}_1) &= 0.9 \log\left(\frac{0.9}{0.8}\right) + 0.1 \log\left(\frac{0.1}{0.2}\right) \approx 0.0367 \\
D_{KL}(\boldsymbol{x}_2) &= 0.8 \log\left(\frac{0.8}{0.7}\right) + 0.2 \log\left(\frac{0.2}{0.3}\right) \approx 0.0257 \\
D_{KL}(\boldsymbol{x}_3) &= 0.8 \log\left(\frac{0.8}{0.9}\right) + 0.2 \log\left(\frac{0.2}{0.1}\right) \approx 0.0444 \\
D_{KL}(\boldsymbol{x}_4) &= 0.7 \log\left(\frac{0.7}{0.8}\right) + 0.3 \log\left(\frac{0.3}{0.2}\right) \approx 0.0282
\end{aligned}
\tag{18}
$$

When having the following inputs and outputs:

FIGURE 9: From input to output

, the KL divergence is computed as follows (using e as base):

$$
\begin{aligned}
D_{KL}(\boldsymbol{x}^{in} || \boldsymbol{x}^{out}) &= \sum_{j=1}^{M} D_{KL}(\boldsymbol{a}_j^{in} || \boldsymbol{a}_j^{out}) \\
&= \sum_{j=1}^{M} \sum_{k=1}^{K_j} p^{in}(C_{j,k}) \cdot \log\left\{\frac{p^{in}(C_{j,k})}{p^{out}(C_{j,k})}\right\} \\
&= 1.0 \cdot \log\left(\frac{1.0}{0.9}\right) + 0.0 \cdot \log\left(\frac{0.0}{0.1}\right) + 0.8 \cdot \log\left(\frac{0.8}{0.9}\right) + 0.2 \cdot \log\left(\frac{0.2}{0.1}\right) \\
&\approx 0.1498
\end{aligned}
\tag{19}
$$

A.1.2 Jensen Shannon Divergence

Theorem:
When using equal weights $[\frac{1}{2}, \frac{1}{2}]$, we have that

$$
JSD_{\frac{1}{2}}(P||Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M)
$$

with $M = \frac{1}{2}(P + Q)$

Proof.

$$
\begin{aligned}
JSD_{\frac{1}{2}}\Big(P||Q\Big) &= H\Big(\frac{1}{2}P + \frac{1}{2}Q\Big) - \frac{1}{2}H(P) - \frac{1}{2}H(Q) \\
&= -\sum_i \frac{1}{2}(p_i + q_i)\log\Big(\frac{1}{2}(p_i + q_i)\Big) + \frac{1}{2}\sum_i p_i \log(p_i) + \frac{1}{2}\sum_i q_i \log(q_i) \\
&= -\frac{1}{2}\sum_i p_i \log(m_i) + q_i \log(m_i) + \frac{1}{2}\sum_i p_i \log(p_i) + \frac{1}{2}\sum_i q_i \log(q_i) \\
&= \frac{1}{2}\sum_i p_i \log(p_i) - \frac{1}{2}\sum_i p_i \log(m_i) + \frac{1}{2}\sum_i q_i \log(q_i) - \frac{1}{2}\sum_i q_i \log(m_i) \\
&= \frac{1}{2}\sum_i p_i \log\Big(\frac{p_i}{m_i}\Big) + \frac{1}{2}\sum_i q_i \log\Big(\frac{q_i}{m_i}\Big) \\
&= \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M)
\end{aligned}
$$

□

A.2 Synthetic data construction

Both the construction of the Bayesian Network and sampling from it is done in Python by using the *PyAgrum* library [22].

A Bayesian Network is constructed by means of the following steps:

1. Create an instance *bn* of the *BayesNet()* class.
2. Add nodes to *bn* by means of a the *bn.add()* method and the *LabelizedVariable()* class.
3. Add edges between the nodes by means of the *addArc()* method.
4. Define conditional probability tables for each node using the *bn.cpt()* method.
5. Create an instance *dbg* of the *BNDatabaseGenerator()* class.

The samples are taken by the following single step:

1. Sample from *dbg* by means of the *dbg.drawSamples()* and store it in a *.csv* file by means of the *dbg.toCSV()* method.

As an example, the Bayesian Network from Fig. 4 is implemented and is sampled from as follows:

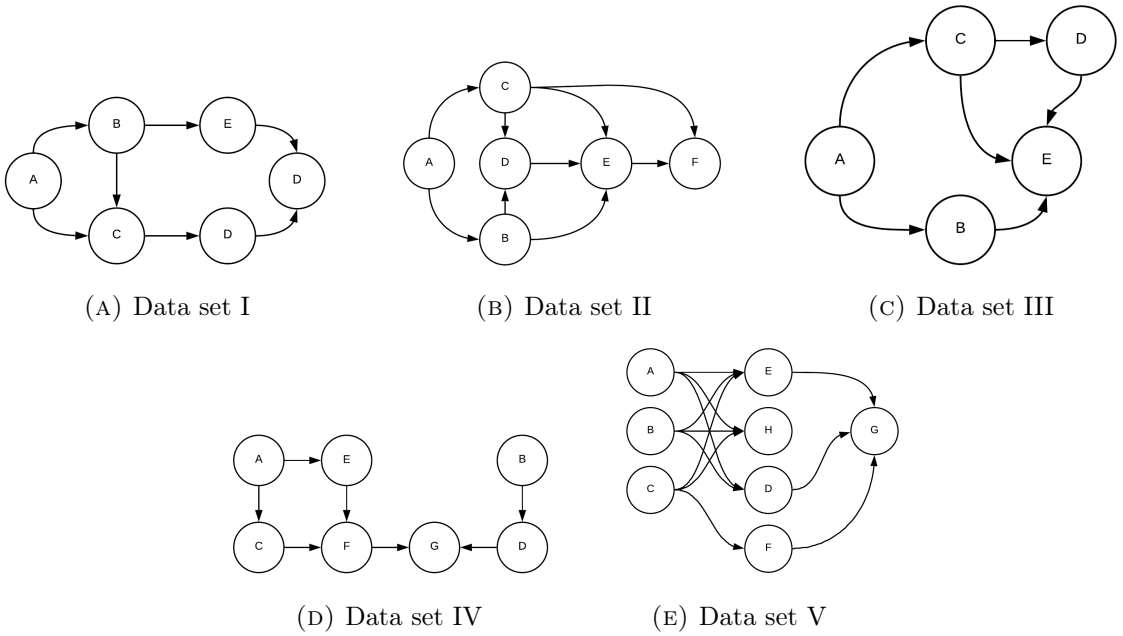
```
import pyAgrum as gum

# Initialize the Bayesian Network
bn = gum.BayesNet('MyNetwork')
# Add nodes
R = bn.add(gum.LabelizedVariable('R', 'Rain', 2))
S = bn.add(gum.LabelizedVariable('S', 'Sprinkler', 2))
G = bn.add(gum.LabelizedVariable('G', 'Grass wet', 2))
# Add edges
for link in [(R, S), (R, G), (S, G)]:
    bn.addArc(*link)
# Define conditional probability tables
bn.cpt(R)[:] = [0.2, 0.8]
```

```
bn.cpt(S)[0, :] = [0.4, 0.6]
bn.cpt(S)[1, :] = [0.01, 0.99]
bn.cpt(G)['R':0, 'S':0] = [0.0, 1.0]
bn.cpt(G)['R':0, 'S':1] = [0.8, 0.2]
bn.cpt(G)['R':1, 'S':0] = [0.9, 0.1]
bn.cpt(G)['R':1, 'S':1] = [0.99, 0.01]
# Create instance of the BNDatabaseGenerator
dbg = gum.BNDatabaseGenerator(bn)
# Sample from it
dbg.drawSamples(1000)
# Store the samples
dbg.toCSV('File.csv')
```

Listing 1: Bayesian Networks via PyAgrum

A.3 Synthetic data sets

In this section, we briefly describe the data generating distributions that were used to sample data from in order to create synthetic data sets. We do this by specifying the graphical structure of the network, by mentioning the cardinality of the state space for each random variable and most importantly by giving the conditional probability tables.
BN graphical structure:



(a) Data set I (b) Data set II (c) Data set III

(d) Data set IV (e) Data set V

Figure 10: Structure for the Bayesian Networks resembling the data generating distributions of the synthetic data sets

State space cardinality per random variable:
The items in the lists below correspond to the random variables $A, B, \ldots$ in that order.

- Data set I: $[3, 3, 3, 3, 2, 2]$.
- Data set II: $[3, 6, 3, 3, 3, 2]$.
- Data set III: $[10, 2, 2, 3, 4]$.
- Data set IV: $[4, 5, 3, 3, 2, 3, 2]$.

- Data set V: $[3, 3, 3, 6, 2, 3, 5, 5]$.

Conditional probability tables:

Data set I:

$$P(A) = \begin{bmatrix} 0.33 & 0.33 & 0.33 \end{bmatrix}, P(B|A) = \begin{bmatrix} 0 & 1 & 0 \\ 0.5 & 0.5 & 0.5 \\ 0.1 & 0 & 0.9 \end{bmatrix}, P(C|A,B) = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P(D|C) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P(E|B) = \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0 & 1 \end{bmatrix}, P(F|D,E) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.8 & 0.2 \end{bmatrix}$$

Data set II: conditional probability tables for $P(A)$, $P(B|A)$, $P(C|A)$, $P(D|B,C)$, $P(E|B,C,D)$, $P(F|C,E)$, respectively.

$$\begin{bmatrix} 0.33 & 0.17 & 0.5 \end{bmatrix}, \begin{bmatrix} 0.33 & 0.33 & 0.33 & 0 & 0 & 0 \\ 0 & 0.17 & 0.33 & 0.33 & 0.17 & 0 \\ 0 & 0 & 0 & 0.33 & 0.33 & 0.33 \end{bmatrix}, \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.3 & 0.5 & 0.2 \\ 0.9 & 0.1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.9 & 0.05 & 0.05 \\ 0.8 & 0.15 & 0.05 \\ 0.7 & 0.15 & 0.15 \\ 0.9 & 0.05 & 0.05 \\ 0.8 & 0.15 & 0.05 \\ 0.7 & 0.15 & 0.15 \\ 0.8 & 0 & 0.2 \\ 0.75 & 0.05 & 0.2 \\ 0.7 & 0.1 & 0.2 \\ 0.8 & 0 & 0.2 \\ 0.75 & 0.05 & 0.2 \\ 0.7 & 0.1 & 0.2 \\ 0.7 & 0 & 0.3 \\ 0.65 & 0.1 & 0.25 \\ 0.6 & 0.15 & 0.15 \\ 0.7 & 0 & 0.3 \\ 0.65 & 0.1 & 0.25 \\ 0.6 & 0.15 & 0.15 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0.9 & 0.1 \\ 1 & 0 \\ 0.5 & 0.5 \\ 0.8 & 0.2 \\ 0.9 & 0 \\ 0.5 & 0.5 \\ 0.7 & 0.3 \\ 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix}$$

Data set III:

$$P(A) = \begin{bmatrix} 0.1 & 0.1 & \dots & 0.1 \end{bmatrix}, P(B|A) = \begin{bmatrix} 0 & 1 \\ 0.1 & 0.9 \\ 0.2 & 0.8 \\ \dots & \dots \\ 0.9 & 0.1 \end{bmatrix}, P(C|A) = \begin{bmatrix} 0 & 1 \\ 0.1 & 0.9 \\ 0.2 & 0.8 \\ \dots & \dots \\ 0.9 & 0.1 \end{bmatrix}$$

$$P(D|C) = \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0.3 & 0.3 & 0.4 \end{bmatrix}, P(E|C,B,D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0.6 & 0 & 0.4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Data set IV:

$$P(A) = \begin{bmatrix} 0.25 & \dots & 0.25 \end{bmatrix}, P(B) = \begin{bmatrix} 0.2 & \dots & 0.2 \end{bmatrix}, P(C|A) = \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.9 & 0.1 \\ 0 & 0.1 & 0.9 \end{bmatrix}$$

$$P(D|B) = \begin{bmatrix} 0.9 & 0.05 & 0.05 \\ 0.8 & 0.15 & 0.05 \\ 0.2 & 0 & 0.8 \\ 0 & 0.5 & 0.5 \\ 0.05 & 0.05 & 0.9 \end{bmatrix}, P(E|A) = \begin{bmatrix} 0.05 & 0.95 \\ 1 & 0 \\ 0.9 & 0.1 \\ 0.95 & 0.05 \end{bmatrix}, P(F|C,E) = \begin{bmatrix} 0.93 & 0.02 & 0.05 \\ 0 & 0.99 & 0.01 \\ 0.9 & 0.05 & 0.05 \\ 0.1 & 0.02 & 0.88 \\ 0.01 & 0.99 & 0 \\ 0.7 & 0.1 & 0.2 \end{bmatrix}$$

$$P(G|F,D) = \begin{bmatrix} 1 & 0 \\ 0.9 & 0.1 \\ 0.1 & 0.9 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0.5 & 0.5 \\ 1 & 0 \end{bmatrix}$$

Data set V: conditional probability tables for $P(A)$, $P(B)$, $P(C)$, $P(D|A,B)$, $P(E|A,B,C)$,

$P(F|C)$, $P(G|D,E,F)$ & $P(H|A,B,C)$, respectively.

$$P(A) = P(B) = P(C) = \begin{bmatrix} 0.33 & 0.33 & 0.33 \end{bmatrix},$$

$$\begin{bmatrix} 0.7 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 0.1 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 & 0.8 \\ 0.3 & 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 & 0.1 & 0.1 \\ 0.1 & 0 & 0 & 0 & 0 & 0.9 \\ 0 & 0 & 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.99 & 0.01 \\ 0.2 & 0.8 \\ 0.99 & 0.01 \\ 0.9 & 0.1 \\ 0.3 & 0.7 \\ 0.8 & 0.2 \\ 0.8 & 0.2 \\ 0.1 & 0.9 \\ 0.8 & 0.2 \\ 0.9 & 0.1 \\ 0.15 & 0.85 \\ 0.8 & 0.2 \\ 0.7 & 0.3 \\ 0.1 & 0.9 \\ 0.7 & 0.3 \\ 0.7 & 0.3 \\ 0.1 & 0.9 \\ 0.8 & 0.2 \\ 0.88 & 0.12 \\ 0.2 & 0.8 \\ 0.7 & 0.3 \\ 0.6 & 0.4 \\ 0.1 & 0.9 \\ 0.9 & 0.1 \\ 0.5 & 0.5 \\ 0.11 & 0.89 \\ 0.8 & 0.2 \end{bmatrix}, \begin{bmatrix} 0.99 & 0.01 & 0 & 0 & 0 \\ 0 & 0.2 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0.85 & 0.15 \\ 0.8 & 0.1 & 0.1 & 0 & 0 \\ 0 & 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 0 & 0.85 & 0.15 \\ 0.85 & 0.1 & 0.05 & 0 & 0 \\ 0 & 0.1 & 0.9 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0.2 \\ 0.8 & 0.1 & 0.1 & 0 & 0 \\ 0 & 0.15 & 0.85 & 0 & 0 \\ 0 & 0 & 0.05 & 0.9 & 0.05 \\ 0.8 & 0.1 & 0.1 & 0 & 0 \\ 0 & 0.1 & 0.9 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 0.1 \\ 0.75 & 0.15 & 0.1 & 0 & 0 \\ 0 & 0.1 & 0.9 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0.2 \\ 0.8 & 0.08 & 0.12 & 0 & 0 \\ 0 & 0.2 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 0.1 \\ 0.82 & 0.1 & 0.08 & 0 & 0 \\ 0 & 0.1 & 0.9 & 0 & 0 \\ 0 & 0 & 0 & 0.92 & 0.08 \\ 0.7 & 0.2 & 0.1 & 0 & 0 \\ 0 & 0.11 & 0.89 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0.2 \end{bmatrix},$$

$$\begin{bmatrix} 0.88 & 0.1 & 0.01 & 0.01 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0.05 & 0.1 & 0.85 & 0 \\ 0 & 0 & 0.01 & 0.99 & 0 \\ 0 & 0 & 0 & 0.1 & 0.9 \\ 0.2 & 0 & 0.2 & 0 & 0.6 \\ 0.85 & 0.15 & 0 & 0 & 0 \\ 0 & 0.15 & 0.85 & 0 & 0 \\ 0 & 0 & 0.1 & 0.85 & 0.05 \\ 0 & 0 & 0 & 0.95 & 0.05 \\ 0 & 0 & 0.05 & 0.05 & 0.9 \\ 0.1 & 0 & 0.05 & 0.05 & 0.8 \\ 0.9 & 0.1 & 0 & 0 & 0 \\ 0 & 0.2 & 0.8 & 0 & 0 \\ 0 & 0 & 0.1 & 0.9 & 0 \\ 0 & 0 & 0.02 & 0.98 & 0 \\ 0 & 0 & 0.05 & 0.05 & 0.9 \\ 0.2 & 0 & 0.1 & 0 & 0.7 \\ 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0.15 & 0.8 & 0.05 & 0 \\ 0 & 0 & 0.15 & 0.85 & 0 \\ 0 & 0 & 0.05 & 0.85 & 0.1 \\ 0 & 0 & 0 & 0.15 & 0.85 \\ 0.05 & 0 & 0 & 0.15 & 0.8 \\ 0.99 & 0.01 & 0 & 0 & 0 \\ 0 & 0.21 & 0.79 & 0 & 0 \\ 0 & 0 & 0.1 & 0.9 & 0 \\ 0 & 0 & 0.05 & 0.95 & 0 \\ 0 & 0 & 0 & 0.05 & 0.95 \\ 0.1 & 0 & 0 & 0 & 0.9 \\ 0.7 & 0.3 & 0 & 0 & 0 \\ 0 & 0.1 & 0.9 & 0 & 0 \\ 0 & 0 & 0.05 & 0.9 & 0.05 \\ 0 & 0 & 0 & 0.95 & 0.05 \\ 0 & 0 & 0 & 0.1 & 0.9 \\ 0.1 & 0 & 0 & 0.1 & 0.8 \end{bmatrix}$$

$$\begin{bmatrix} 0.9 & 0.1 & 0 \\ 0.1 & 0.9 & 0 \\ 0 & 0.1 & 0.9 \end{bmatrix}$$
