DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis

A Probabilistic Modeling Framework for Unclean Databases with Embeddings of Categorical Values

Daoping Wang

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis

A Probabilistic Modeling Framework for Unclean Databases with Embeddings of Categorical Values

Author:	Daoping Wang
Supervisor:	PD Dr. rer. nat. Martin Kleinsteuber
Advisor:	Dr. rer. nat. Rudolf Sailer
Submission Date:	Submission date

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, Submission date Daoping Wang

Acknowledgments

I wish to express my sincere appreciation to PD Dr. rer. nat. Martin Kleinsteuber, who allowed me the possibility to work on such an exciting, yet challenging topic and the freedom of selecting research direction according to my personal preferences and strengths.

I wish to express my deepest gratitude to Dr. rer. nat. Rudolf Sailer for the steady guidance throughout the thesis, which would not have been possible without the countless, fruitful discussions and evaluations with him. Dr. Sailer provided me with his scientific opinions and advice, which precisely pointed out potential research opportunities and opened my mind time and again. Also, I really appreciate the precious time Dr. Sailer gave me whenever needed.

Many thanks go to Muhammad Umer Anwaar and Rayyan Ahmad Khan for taking the time to give me valuable insights into the topic from their perspectives and fields of expertise. Every discussion with them opened up fresh ideas and brought in positive energy.

I am sincerely grateful to Mercateo for inviting me to their wonderful team. Here, I find myself next to true experts in an environment full of freedom and friendliness, while I received all the resources and support necessary for accomplishing the thesis. I thank the fellow students and colleagues, especially the members of team Analytics Amber, for showing great interest in this work and giving helpful suggestions.

Abstract

As a consequence of growing reliance on data-driven innovations across industries, data inconsistencies like incorrect or missing data have become a major bottleneck due to their severe negative impact on the performance of downstream applications. Identifying anomalies in unclean databases is a particular challenging problem due to the heterogeneity of error types and, in many cases, the lack of ground truth labels. Most data cleaning models fail to meet these challenges by either restricting the kinds of anomalies or being unable to accommodate the high level of noise present in the database.

In this thesis, the problem of anomaly detection in databases with high uncertainty and limited external knowledge is addressed. We propose a domain-independent probabilistic modeling framework that models unclean databases with probabilistic graphical models and performs anomaly detection using statistical reasoning techniques. A probabilistic model generated by the framework associates data records with confidence scores by leveraging external information on attribute-level dependencies and quantitative statistics extracted from the observed database.

Furthermore, we describe the high cardinality problem of categorical attributes and propose a solution based on vector representation learning. By taking an unclean database as input, a neural network model assigns each categorical value a high-dimensional vector such that values that share common characteristics are located close to one another in the vector space. We show that using meaningful vector representations significantly increases anomaly detection performance.

Comprehensive experiments with two datasets are conducted to evaluated the effectiveness of our approach. As result, we successfully show that the probabilistic models generated by the framework are performant in terms of anomaly detection under difficult real-world conditions.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Our Goals . . . 2
1.3 Outline . . . 3

2 Related Work **4**
2.1 Anomaly Detection . . . 4
2.2 Data Repairing . . . 5

3 Preliminaries **7**
3.1 Database Theories . . . 7
3.1.1 Relational Databases . . . 7
3.1.2 Probabilistic Databases . . . 7
3.1.3 Integrity Constraints . . . 9
3.1.4 Quantitative Statistics . . . 10
3.2 Probabilistic Graphical Models . . . 11
3.3 Vector Representation for Categorical Attributes . . . 13
3.3.1 High Cardinality Problem of Categorical Attributes . . . 14
3.3.2 Learning Embedding Vectors . . . 15
3.4 Probabilistic Programming with *Gen* . . . 18

4 Probabilistic Modeling Framework **22**
4.1 Generative Functions for Functional Dependencies . . . 22
4.1.1 Mathematical Background . . . 22
4.1.2 General Implementation . . . 25
4.2 Generative Functions for Matching Dependencies . . . 29
4.2.1 Mathematical Background . . . 29
4.2.2 General Implementation . . . 30
4.3 Generative Functions for Occurrence Statistics . . . 31
4.3.1 Mathematical Background . . . 31
4.3.2 General Implementation . . . 32

5 Embedding Training Model **33**
5.1 Optimization Target . . . 33
5.2 Positive and Negative Tuples as Training Samples . . . 34
5.3 Model Architecture . . . 36

6 Experimental Setup and Results **38**
6.1 Datasets . . . 38
6.1.1 Rental Apartments Dataset . . . 38
6.1.2 Mercateo Article Database . . . 39
6.1.3 Generating Anomalous Tuples . . . 40
6.2 Anomaly Detection . . . 42
6.2.1 Evaluation Methodology . . . 42
6.2.2 General Performance and Effect of Embeddings . . . 46
6.2.3 Effect of Integrity Constraints . . . 47
6.2.4 Effect of PDB Parameters . . . 50
6.3 Embedding Training . . . 53
6.3.1 Embedding Quality . . . 53
6.3.2 Effect of Embedding Dimension . . . 53
6.3.3 Effect of Batch Size . . . 55
6.3.4 Implementation Details . . . 55

7 Summary and Outlook **57**

List of Figures **59**

List of Tables **60**

Bibliography **61**

1 Introduction

1.1 Motivation

In the last several years, there has been an unparalleled surge of interest in nearly everything around data across industries and academia. Given the obvious success of tech giants such as Apple, Microsoft, Amazon and Alphabet, which are the top four most valuable listed companies in the world by the end of 2019, more corporations are joining a new kind of "arms race", while they concentrate more on developing innovative technologies that can handle the volume and variety of acquired data and apply it smartly to provide decisive advantage in the according business area. For this reason, data has been equated to *the new oil of the century* [49] and considered the most valuable resource for technological advancement and profit making.

Amongst various use cases of data in various application areas, it plays a major role in constructing machine-learning-based systems that are successfully employed for practical tasks, including e.g. automated driving [8], image recognition [31], and even playing go [45]. While these methods have shown to possess superhuman abilities in numerous fields, their success heavily relies on both quantity and quality of the input data. Taking as example the family of *deep learning* techniques, they can learn representations of high-dimensional, high-variance data that is impossible to fully capture with simple heuristics or hand-engineered features [20] [28], though require massive training sets of labeled examples of high quality to learn from. In fact, owning a database with large amounts of records by itself is not sufficient for realizing powerful algorithms, but the data must be pre-processed and cleaned before being made use of.

Real-world databases often contain noisy and erroneous data, which causes a major

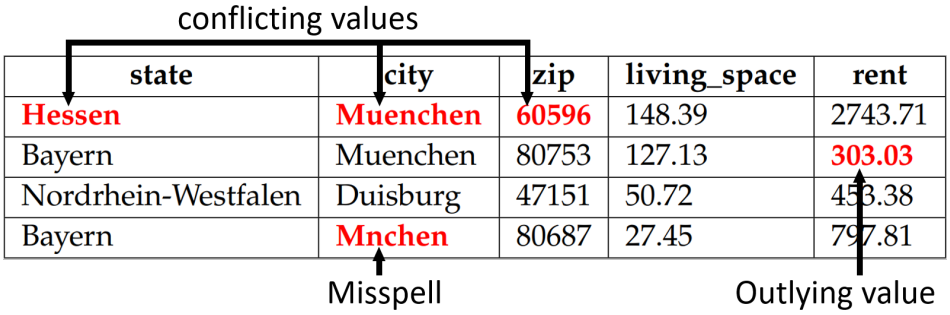


Figure 1.1: A variety of anomalies can occur in real-world datasets.

bottleneck in analytics. For instance, a snippet from a housing offering database in Figure 1.1 can have flaws like misspelled city names, wrongly entered rent prices, or conflicts between zip code and address. At first glance, the task of correcting such data errors seems to be trivial for humans given some work instructions and prior information about the treated data. However, cleaning large-scale datasets in a manual fashion is tremendously expensive regarding the amount of occupied man-hours, and it remains an unreliable process as new errors can be induced during manual editing. In addition, having good prior information about domain-specific datasets requires in-depth expert knowledge, which is not a guaranteed factor and further limits the potential benefits concealed in the data. To address these problems, much research focus has been put into data cleaning and aiming at automated solutions.

To assure the effectiveness of automated data cleaning tools, a number of aspects must be taken into consideration during tool design and implementation. Datasets from different sources differ significantly from each other in terms of types of occurred anomalies and their distributions, availability of external knowledge that is useful for cleaning, and expected application of the dataset after cleaning. In real-world situations, these factors, in particular anomaly distribution and external knowledge, are unknown or inaccessible and thus make the cleaning problem more challenging. Considering the first data record in Figure 1.1, one must understand the distinct mapping between zip code, state and city in order to identify the conflicting values, and incorporate statistical reasoning to generate a repair suggestion. This holistic cleaning process, which combines user-specified heuristics (i.e. zip code, state and city are unambiguously allocated) with probabilistic techniques, is regarded as a viable, generalizable solution that is capable of handling data with high uncertainty and limited prior knowledge.

1.2 Our Goals

In this master's thesis, we propose a novel probabilistic modeling framework that uses *probabilistic graphical models* as representation for unclean databases and converts data cleaning tasks into statistical reasoning problems. Our goal is a domain-independent, generalized modeling framework for holistic incorporation of a variety of information sources, including external knowledge in form of integrity constraints and dictionaries, pre-assumed anomaly distributions, and quantitative statistics extracted from observed databases.

We address the problem of limited availability of external knowledge and sparse occurrence of categorical values by introducing a novel neural network model for learning vector representations, also referred to as *embeddings*, that capture latent correlations between categorical attributes.

Probabilistic Modeling Framework

The proposed framework generates probabilistic models for unclean databases. Probabilistic models capture uncertainties over data by regarding column attributes as random variables, with each random variable obeying certain probability distribution. Under the assumption

that the majority of data entries of an observed database are free of error, the probability distribution of each column attribute is parametrized with the statistical information extracted from the observation. To evaluate the confidence score for a data tuple, i.e. the likelihood of it being a correct tuple, the marginal probabilities of its attribute values are calculated and multiplied together.

Our probabilistic modeling framework is realized with *Gen*, a probabilistic programming system that comes as a package for the *Julia* programming language.

Vector Representation Learning

Databases are structured collections of data values whose type can be numerical or categorical. For categorical attributes that have high cardinality, i.e. they accept too many values which then occur sparsely in the observation, it is difficult to reason about their probabilities based on occurrence statistics. Additionally, if this problem emerges together with the lack of external knowledge on those categorical attributes, it is tremendously difficult for automated programs or even humans to perform data cleaning.

We accept this challenge as an important focus of this thesis and introduce a neural network model that is inspired by the *Work2vec* model from the area of *natural language processing*. As the learnt high-dimensional embedding vectors capture latent interactions between categorical values, they are used to replace categorical values in probabilistic models and thereby enable us to perform numerical techniques and leverage similarities between embeddings during the cleaning process. We show that having meaningful vector representations adds valuable information to probabilistic models and significantly increases anomaly detection performance.

1.3 Outline

In Section 2 we review existing research work related to data cleaning. In Section 3 we provide background preliminaries that are necessary for understanding the addressed problems and proposed solution. In Section 4 we introduce mathematical foundation and practical implementation of our proposed probabilistic modeling framework. In Section 5 we introduce the architecture and working mechanisms of our model for embedding training. In Section 6 we show experimental results of our framework and compare anomaly detection performances between different settings and different datasets. Finally, we summarize the overall outcome of this thesis and discuss open research directions for future work in Section 7.

2 Related Work

Data cleaning can be separated in two tasks: (i) *anomaly detection*, where data inconsistencies, including constraint violations, duplicate data, and incorrect or missing data values are identified, and (ii) *data repairing*, which involves updating the available data to remove any detected anomalies. Both tasks have caught great attention in the last decades, while there have been continuously emerging research projects addressing them from both theoretical and practical perspective.

2.1 Anomaly Detection

Traditional anomaly detection approaches construct models of normal data and detect deviations from the normal model in observed data. A survey and comparison of such techniques is presented by Warrender et al. in [52]. Studied techniques include a method based on sequence analysis for anomaly detection [32], a decision tree model over normal data [36] and a neural network modeling approach[26]. These methods are based on classification and work in a supervised setting with labeled data, which cause them to suffer from three major drawbacks: First, manually obtaining labeled data is a particular expensive and discouraging process. Second, models whose construction relies on training data have difficulty at detecting new types of anomalies that are previously unseen. At last, classification-based techniques assign a label to each record, which can become a disadvantage when a meaningful anomaly score is preferred.

To overcome these limitations, unsupervised anomaly detection techniques are investigated and developed to a large extent. An important group of them is known as nearest-neighbor-based techniques that measure similarity between records and establish local neighborhoods. They are based on the assumption that normal data occur in dense neighborhoods, while anomalies occur far from their closest neighborhoods [9]. Nearest-neighbor-based techniques evaluate the anomaly score based on either the distance of a record to its k^{th} nearest neighbor [24] [41] [55] or the relative density of each record [48] [47] [10] [30]. A key advantage of them is that they are purely data-driven and unsupervised in nature. However, if the formed neighborhoods do not imply any notion of normality or correctness, such techniques fail to label them correctly, resulting in low detection performance. Furthermore, the computational complexity of $O(N^2)$ of the neighborhood construction process, i.e. computing the distances between each record and all other records, is also a challenging factor.

Another broad category of anomaly detection techniques are based on statistics with the underlying principle: "An anomaly is an observation which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed" [4]. The idea is

to fit a statistical model, which we also refer to as a probabilistic model, to the observed data and to apply a inference test to determine if an unseen record belongs to this model. Records that have a low probability based on the inference test are declared as anomalies. Probabilistic anomaly detection techniques can be divided into two branches: Parametric techniques assume the knowledge of underlying distribution, construct probabilistic model accordingly and perform parameter estimation from the observed data [2] [23] [35]. On the contrary, non-parametric techniques do not define model structure a priori, but instead determine it from observed data [21] [54]. Comparing to parametric techniques, non-parametric techniques make fewer assumptions regarding the data.

Probabilistic techniques are advantageous respecting that they provide for each data instance an anomaly score and an associated confidence interval instead of discriminative labels. However, they strongly rely on the assumptions regarding the underlying data distribution: If the assumptions hold true, these techniques provide an statistically justifiable solution for unsupervised anomaly detection [3]; If the assumptions are false, they would have difficulty obtaining satisfactory detection results.

The database modeling technique proposed in this work falls under the category of probabilistic techniques. To address the problem of interest, i.e. performing anomaly detection and repair suggestion on probabilistic databases, we introduce a modeling approach based on *probabilistic graphical models* with special focus on capturing and leveraging the interactions between attributes of multivariate data.

2.2 Data Repairing

Among practical tools that aim to automate data repairing, the majority adopts a deterministic interpretation of databases and uses a set of logical heuristics to identify anomalies and possible repairs [7] [13] [51] [53]. If multiple possible repair options exist for a single database, the proposed methods rely on the notion of *minimality* [11] to prioritize across repair options, i.e. to prefer repairs that involve less changes with respect to the original database. This use of minimality as an operational principle for repair checking and query answering is a practical artifact to limit the search space. However, as these practical methods do not reason about the "likelihood" of repair options and thus use combinatorial principles such as minimality for prioritization, they are unable to capture data's statistical features that provide valuable insights and are much needed for generating correct repairs.

A data repairing tool, proposed by Rekatsinas et al. in [42], incorporates statistical reasoning by applying statistical learning and inference. Instead of a minimal repair, it reasons about a most probable repair and shows more accurate data cleaning results than minimality-based tools for a series of real-world data cleaning scenarios [42]. The tool addresses the data cleaning problem from a holistic perspective and combines a multitude of heterogeneous signals and information sources, e.g. integrity constraints, external data and statistical properties of observed data. Based on this combined knowledge, it models how clean data is generated and how errors are injected by learning a probabilistic model using training data, which are clean data records separated from the observed data using external anomaly

detection methods. Repair suggestions are generated by performing statistical inference over the probabilistic model. Regarding repair performance, the tool yields an average $F1$ improvement of $2\times$ than state-of-the-art methods.

As a result of its success, a formal framework is established upon its concept and aims to provide theoretical foundation for probabilistic approaches for data cleaning tasks [19]. It regards an observed database as the outcome of a noisy channel model that is comprised of an intention model, which represents a belief of how (intended) clean data is generated, and a realization model, which maps intention model's output to the observed database by inducing noise. This theoretical framework brings up the notion of *probabilistic unclean databases* (PUBs) and formally defines computational problems related to PUBs, including data cleaning, probabilistic query answering and supervised learning of parametric PUBs via parameter estimation with training data.

In this thesis, the proposed probabilistic modeling approach adopts the PUB concept and the holistic data cleaning method which incorporates various signals and information sources. Knowing that databases with limited external knowledge and high cardinality attributes are of our central interest, we have done more work on anomaly detection while planning to address repair suggestion as the immediate next step. In fact, as our approach is based on graphical models and realized with probabilistic programming, it intrinsically supports various inference algorithms that come in handy for data repairing tasks. This conveniently makes our approach extendable with advanced data repairing functionalities.

3 Preliminaries

3.1 Database Theories

In this section, we introduce definitions and concepts of the data model that is considered throughout this thesis.

3.1.1 Relational Databases

Notations used for referring to our database model follow the semantics of the *relational data model* proposed in [14]. A relational database is comprised of a *schema* S that has a finite set of *relation symbols*. Each relation symbol R of $\mathbf{S}$ is associated with a set of *attributes* $\mathbf{A}_R = \{A_1, ..., A_k\}$, where k is the arity of $\mathbf{A}_R$. In this thesis, we only consider schemas with one single R and thus abbreviate $\mathbf{A}$ for $\mathbf{A}_R$. The *domain* of an attribute $A \in \mathbf{A}$, denoted by $dom(A)$, defines the set of admissible values and their data type. A relation symbol R, also known as *table* in the SQL terminology, is a subset of the Cartesian product of the domains of $\mathbf{A}$. R stores the subset in form of row entries, each of them being referred to as a *tuple*. A tuple t is a set of *cells* denoted as $Cells[t] = (A_i[t], ..., A_k[t])$, where each cell c corresponds to a different attribute in $\mathbf{A}$. If $t = (c_1, ..., c_k)$ is a tuple over R and $\mathbf{A} = \{A_1, ..., A_k\}$, then we refer to the cell c_j as $t.A_j$, where $j = 1, ..., k$. The set of all tuples over R is referred to as $tuples(R)$. A *database* D over $\mathbf{S}$ is an instance of R, where each $t \in tuples(D)$ has a unique *identifier* $i \in ids(D)$ that denotes the set of identifiers of D.

3.1.2 Probabilistic Databases

The model of *probabilistic databases* (PDB) extends the relational data model by associating probabilities to measure the degree of uncertainty [46]. For our model, we consider PDBs with the property of *tuple-independence*, where tuples are regarded as independent probabilistic events coming from the probability distributions over the domains of $\mathbf{A}$. Formally, a *tuple-independent PDB* (D, p) over $\mathbf{S}$ is a pair consisting of D, an instance of the relation symbol R of $\mathbf{S}$, and a *probability measure* $p : tuples(D) \rightarrow [0, 1]$, which assigns a probability value $p(t)$ to each tuple $t \in tuples(D)$. The probability value $p(t)$, also referred to as confidence score, denotes the confidence in the validity of t in D, i.e. a higher $p(t)$ stands for a higher confidence in t being valid. Note that the notion of validity is oftentimes ambiguous and traces back to the composition of p: If p implements integrity constraints over $\mathbf{S}$, the resulting low $p(t)$ value would directly indicate the invalidity of t; On the contrary, if p implements statistical correlations between attributes, a low $p(t)$ only shows the improbability of t [22].

id	state	city	zip	living_space	construct_year	rent	p
129	Baden-Wuerttemberg	Karlsruhe	76195	73.76	1933	1089.88	0.94
130	Hessen	Hanau	63450	40.19	1944	615.5	0.93
131	Hessen	Muenchen	82393	148.39	2010	2743.71	0.05
132	Bayern	Muenchen	80753	127.13	2001	303.03	0.1
133	Hessen	Darmstadt	64291	270.99	1934	1028.35	0.25
134	Nordrhein-Westfalen	Duisburg	47151	50.72	1961	453.38	0.95
...	...	...	...	...	...	...	...

(a)

state	*prob*
Hessen	0.1
Bayern	0.2
Niedersachsen	0.08
...	...

(b)

ext_state	ext_city
Baden-Wuerttemberg	Karlsruhe
Bayern	Erlangen
Bayern	Dachau
...	...

(c)

$c1$: zip → state, city
$c2$: city=ext_city → state=ext_state

(d)

$\mathbf{S}_h$	Schema of relation symbol R_h and attributes $\mathbf{A}_h$
$\mathbf{A}_h$	Set of attributes given by $\mathbf{A}_{h,categorical} \cup \mathbf{A}_{h,numerical}$
$\mathbf{A}_{h,categorical}$	Set of categorical attributes: {state, city, zip, construct_year}
$\mathbf{A}_{h,numerical}$	Set of numerical attributes: {living_space, rent}

(e)

Figure 3.1: An example database D_h of apartment rental offers.
(a) Relation instance R_h
(b) External information on the probability distribution of $dom(state)$
(c) External information on the belonging between cities and states
(d) Integrity constraints over S_h
(e) Symbols used in R_h

A probability value $p(t)$ is calculated by the product of a series of *factors* that measure uncertainty at attribute level. In the current context, a factor can represent either the probability distribution of one single attribute, or the joint probability distribution of multiple attributes. A more general definition of factors is introduced in Section 3.2.

Example 1 We illustrate the PDB model and the need of modeling uncertainties and correlations through a concrete application scenario. In Figure 3.1, a simple housing dataset D_h that contains apartment rental offers in Germany is manually created and serves as a running example for the purpose of demonstrating conceptions and problems throughout the thesis. $tuples(D_h)$ can contain different types of anomalies, including misspellings, confusions between attribute values, inconsistent names for the same establishment, and numerical values that heavily deviate from the average.

In the example dataset in Figure 3.1, we observe various types of uncertainties that interact in complex ways. Although the examination of such uncertainties can be trivial for human

personnel with certain level of training, their automation requires an efficient modeling methodology and a holistic compilation of external knowledge and statistical reasoning techniques. While the PDB model is applied to D_h, the probability measure p that can be interpreted as the joint probability distribution over $\mathbf{A}_h$ is constructed. The factors of p model the uncertainty within each $A_i \in \mathbf{A}_h$ in form of probability distributions, incorporating external knowledge (e.g. Figure 3.1b, 3.1c), quantitative statistics of D_h, and integrity constraints.

3.1.3 Integrity Constraints

Integrity constraints are conditions used to judge the consistency of a tuple, i.e. the consistency over $Cells[t_i], i \in ids(D_h)$. They subsume numerous types of conditions, including e.g. functional dependencies [50] [17], matching dependencies [6] [25] and denial constraints [1] [12]. Since databases with limited external knowledge are of our central interest, integrity constraints considered in our PDB model are limited to functional dependencies (FDs) and matching dependencies (MDs).

Definition 3.1.1 *(Functional dependency) A functional dependency defined on database instance D is an expression of the form $X \rightarrow Y$, where X and Y are attribute sets. $X \rightarrow Y$ is satisfied by a database instance D of relation symbol R if $X \rightarrow Y$ is over R and*

$$\forall u, v \in tuples(D) : u[X] = v[X] \rightarrow u[Y] = v[Y]. \tag{3.1}$$

Intuitively, a functional dependency is a constraint between two sets of attributes X and Y in a database, specifying that the values of Y are determined by the values of X. Under our setting, the fact of two tuples in D_h having the same zip code but different state and/or city value would violate the FD c_1 in Figure 3.1d and thus decrease, if not eliminate, the corresponding confidence scores.

Definition 3.1.2 *(Matching dependency) A matching dependency defined on database instances (D_1, D_2) is an expression of the form*

$$\bigwedge_{j \in [1,k]} (D_1[X_1[j]] \approx_j D_2[X_2[j]]) \rightarrow D_1[Y_1] \approx D_2[Y_2] \tag{3.2}$$

where (X_1, X_2) and (Y_1, Y_2) are compatible attribute lists over (D_1, D_2), X_1 and X_2 have the same length k, and $\approx_j$ and $\approx$ are similarity operators, for all $j \in [1, k]$.

Matching dependencies (MDs) specify that a pair of attribute values in two database tuples are to be matched, i.e., made equal, if similarities hold between other pairs of values in the same tuples. In our PDB model, they define look-ups to available external dictionaries or labeled data. In Figure 3.1d, c_2 is an MD implying that *t.state* must be equal to the state value associated with *t.city* in the external dictionary.

3.1.4 Quantitative Statistics

As aforementioned, the automation of anomaly detection and repair suggestion task heavily relies on the quality of external knowledge, e.g. dictionaries and integrity constraints on data, which usually cannot be guaranteed in real-world applications. As the correctness and effectiveness of external knowledge usually depends on the expertise of the corresponding data management personnel, the provided knowledge might be incomplete and erroneous. Also, considering that any deviation between expert knowledge and true distribution of the data will affect probability evaluation and thus data cleaning results, relying solely on expert knowledge is not always a good choice.

In such cases, *quantitative statistics*, which are comprised of information on occurrences and co-occurrences of attribute values in the database, become a reliable source of insides into the ground truth. To extract these statistics and convert them into probabilities, the *categorical distribution* is used and defined as follows:

Definition 3.1.3 *(Categorical distribution) A categorical distribution categorical($\boldsymbol{\theta}$) is a discrete probability distribution with a finite set of outcomes $1, ..., m$. It is parametrized by $\boldsymbol{\theta} = (\theta_1, ..., \theta_m)$, where*

$$Pr(X = j|\boldsymbol{\theta}) = \theta_j \tag{3.3}$$

and

$$\sum_{j=1}^{m} \theta_j = 1. \tag{3.4}$$

For instance, if the information in Figure 3.1c and 3.1d is unavailable, the only left method of evaluating the confidence score for $t_i \in tuples(D_h)$ with the PDB model would be to implement factors to encode Figure 3.1b and the quantitative statistics of $Cells[t_i]$. Then, the reason for assigning a low confidence score to a tuple $t_{low} \in tuples(D_h)$ would be one of the followings:

- Dictionary knowledge: $t_{low}.state$ has a low *prob* according to Figure 3.1b;
- Occurrence statistics: The attribute value $t_{low}.A$, $A \in \mathbf{A}_{h,categorical}$ has a low occurrence in D_h and thus results into the low confidence score of t_{low};
- Co-occurrence statistics: The pair $(t_{low}.A_i, t_{low}.A_j)$, where $A_i, A_j \in \mathbf{A}_h$, $i \neq j$, has a low number of co-occurrences in D_h and thus results into the low confidence score of t_{low}.

Following the above ideas, the majority of tuples with low confidence scores would be those tuples with attribute values or value combinations that are rarely observed in D_h. This result would be desirable if most of the observed tuples are clean, and if the plain quantitative statistics are informative for confidence evaluation. However, this cannot be expected in general cases, as the occurrence information of an attribute value does not necessarily imply its correctness.

It is a major objective of our approach to overcome the problem of insufficient external knowledge and limited information gain from quantitative statistics. The solution we propose

is an PDB model that captures the uncertainties and complex correlations appearing in real-world databases, yet at the same time allows the flexibility to capture statistical regularities.

3.2 Probabilistic Graphical Models

Among various techniques for modeling uncertainty, the techniques referred to as *probabilistic graphical models* (PGMs) are the ones of the most recognized and developed [34] [42] [44] [29]. Also, there is a wide literature on effective performance of probability computation and inference. PGMs are known for their compactness and generality, which are optimal properties necessary for solving our PDB modeling problem. We start with presenting the definition of the general PGM model. Then, we show the suitability of PGMs as practical realization techniques for PDB models by drawing connections between their concepts.

Let X denote a random variable that can be assigned any value from a domain of assignments, denoted by $dom(X)$.

Definition 3.2.1 *(Factor) A factor $f(\mathbf{X})$ is a function over a set of random variables $\mathbf{X} = (X_1, ..., X_n)$ such that $f(\mathbf{x}) \geqslant 0$, $\forall \mathbf{x} \in dom(X_1) \times ... \times dom(X_n)$.*

Definition 3.2.2 *(Probabilistic graphical model) A probabilistic graphical model $P = \langle \mathbf{F}, \mathbf{X} \rangle$ defines a joint distribution over the list of random variables $\mathbf{X} = (X_1, ..., X_n)$ via a set of factors $\mathbf{F}$, while each factor is defined over a subset of $\mathbf{X}$. Given a complete joint assignment $\mathbf{x} \in dom(\mathbf{X})$ to the variables in $\mathbf{X}$, the joint distribution is defined by*

$$Pr(\mathbf{x}) = \frac{1}{Z} \prod_{f \in \mathbf{F}} f(\mathbf{x}_f), \tag{3.5}$$

where $\mathbf{x}_f$ denotes the assignments restricted to the arguments of f and $Z = \sum_{\mathbf{x}'} \prod_{f \in \mathbf{F}} f(\mathbf{x}'_f)$ is a normalization constant referred to as the partition function.

Since the basic building blocks of PGMs are random variables and factors, they can be directly used to model joint probability distributions and to construct the probability measure p of a PDB model. Let the set of attributes $\mathbf{A}$ be a set of random variables. Then, following the concept of PGMs, the joint probability

$$Pr : dom(\mathbf{A}) \rightarrow \mathbb{R}^{\geqslant 0} \text{ such that } \sum_{\mathbf{a} \in dom(\mathbf{A})} Pr(\mathbf{a}) = 1 \tag{3.6}$$

can be represented by the product of a collection of factors, where each factor takes a subset of $\mathbf{A}$, denoted as $\mathbf{A}_{arg}$, as argument and outputs a confidence value for the joint assignment $\mathbf{a}_{arg}$. Here, each factor resembles a probability distribution over $\mathbf{A}_{arg}$ that incorporates schema-level knowledge on the prior distribution of $\mathbf{A}_{arg}$ and/or correlation between the attributes in $\mathbf{A}_{arg}$. For our running example, this knowledge comes from Figure 3.1b (prior distribution), 3.1c (correlation) and 3.1d (integrity constraints).

By using factors for modeling $p(t)$, numerous probabilistic computations on PDB instances can be performed conveniently. In particular, the evaluation of confidence scores for tuples and

inference of missing values can be accomplished by performing *maximum a priori estimation*. Consider the running database example D_h in Figure 3.1 with probability measure now defined as

$$\begin{aligned} p(t) = f_1(t.state) \cdot f_2(t.city|t.state) \cdot f_3(t.zip|t.state, t.city) \cdot f_4(t.living\_space) \\ \cdot f_5(t.construct\_year) \cdot f_6(t.rent|t.living\_space, t.construct\_year), \end{aligned} \tag{3.7}$$

we can

- identify tuples with low confidence score by computing probability values $p(t)$ for all $t \in tuples(D_h)$,
- compute the marginal distribution for a single random variable $A_{marginal} \in \mathbf{A}$ by summing over the distributions for $\mathbf{A}_{rest} \subset \mathbf{A} \backslash A_{marginal}$, and
- compute the most probable assignment to random variables $\mathbf{A}_{missing} \subset \mathbf{A} \backslash \mathbf{A}_{assigned}$ given assignments to $\mathbf{A}_{assigned}$ and thus predict missing values in $tuples(D_h)$, which is also known as the *most probable explanation task* [29].

By breaking up the joint probability distribution, in our context the probability measure p of a PDB instance, into a product of factors, various graphical models from the PGM family can be applied to the modeling process of PDB instances. In this thesis, we focus on the approach of *Bayesian networks*, which ranks among the most common and most widely applied techniques for compactly modeling uncertainty for large collections of random variables [16] [34].

Bayesian networks are directed acyclic graphical models that represent causal or asymmetric interactions amongst a set of random variables [40]. In Bayesian networks, factors used to break up the joint probability distribution are called *conditional probability factors*:

Definition 3.2.3 *(Conditional probability factor, discrete) Let $f(X_d|\Pi)$ denote a factor that takes as arguments $X_d \cup \Pi$ where X_d denotes a random variable that takes value from a domain $dom(X_d)$ of discrete values and Π denotes a set of random variables such that*

$$\sum_{x \in dom(X_d)} f(x|\pi) = 1, \, \forall \pi \in dom(\Pi), \tag{3.8}$$

where X_d is referred to as the child and Π as the parents of X_d. Note that Π can be the empty set.

Definition 3.2.4 *(Conditional probability factor, continuous) Let $f(X_c|\Pi)$ denote a factor that takes as arguments $X_c \cup \Pi$ where $X_c \in \mathbb{R}$ denotes a random variable and Π denotes a set of random variables such that*

$$\int_{-\infty}^{+\infty} f(x|\pi)dx = 1, \, \forall \pi \in dom(\Pi), \tag{3.9}$$

where X_c is referred to as the child and Π as the parents of X_c. Note that Π can be the empty set.

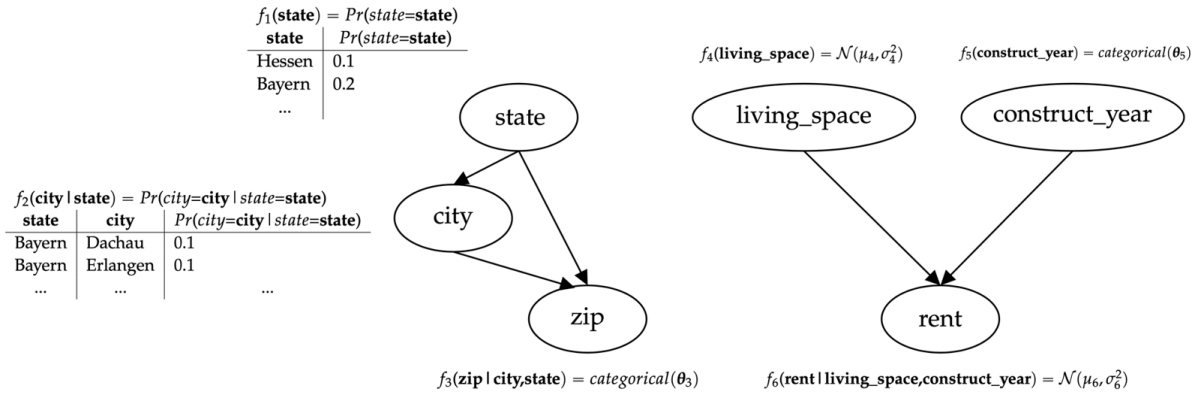


Figure 3.2: An exemplary Bayesian network that models $p(t)$ from Equation 3.7 for running example D_h.

A conditional probability factor represents a conditional probability distribution over the child given an assignment to its parents. To represent a conditional probability factor, a *conditional probability table* that performs the mapping between $x \in dom(X)$ and $f(x|\pi)$ is used. Furthermore, conditional probability factors can be used to enforce deterministic constraints:

Definition 3.2.5 *(Deterministic conditional probability factor) A conditional probability factor $f(X|\Pi)$ is a deterministic factor if*

$$\exists x \in dom(X) \; s.t. \; f(x|\pi) = 1, \forall \pi \in dom(\Pi) \tag{3.10}$$

By having the conditional probability factors defined, the complete joint distribution of the Bayesian network is given by the product of all the factors

$$Pr(x_1, ..., x_n) = \prod_{i=1}^{n} f(x_i|\pi_i) \tag{3.11}$$

that sums up to 1 and is a normalized legal distribution [34].

To interpret the graphical structure of Bayesian networks in our context, their vertices represent random variables, while a direct edge from random variable X_i to random variable X_j indicates direct influence of X_i on X_j. For our running example database from Figure 3.1, a Bayesian network is constructed and shown in Figure 3.2, illustrating how a probability measure $p(t)$ (see Equation 3.7) can be implemented with conditional probability probabilities and a graph structure.

3.3 Vector Representation for Categorical Attributes

In Section 1 and 3.1 we have emphasized the fact that real-world databases are often coupled with the problem of insufficient external knowledge and the lack of information gain from plain quantitative statistics. In this section, we introduce a further crucial problem that is

A	$\lvert dom(A)\rvert$
state	5
city	67
zip	154
construct_year	60

Table 3.1: Cardinalities of $A \in \mathbf{A}_{h,categorical}$

commonly encountered in various application domains, which is the high cardinality of categorical attributes. We then propose a practical solution to the problem, i.e. a model for learning vector representations for categorical attribute values.

3.3.1 High Cardinality Problem of Categorical Attributes

To illustrate the problem of high cardinality, we again use the running example from Figure 3.1 and assume that D_h is a small dataset of 200 tuples, within which every value in $dom(\mathbf{A}_{h,categorical})$ (see Table 3.1) has occurred at least once. Recall that the probability measure $p(t)$ for $t \in D$ is interpreted as a joint probability distribution and calculated as the product of a set of factors $f \in \mathbf{F}$, where each f resembles a probability distribution over a subset of attributes $\mathbf{A}$ taken as argument.

We consider a factor f that models the conditional probability distribution of *construct_year* given assignment $\mathbf{city} \in dom(city)$. As $dom(construct\_year)$ and $dom(city)$ consist of categorical values, f is a discrete conditional probability factor that fulfils the following equation:

$$\sum_{\mathbf{construct\_year} \in dom(construct\_year)} f(\mathbf{construct\_year}|\mathbf{city}) = 1,\ \mathbf{city} \in dom(city).$$

The intuition behind $f(\mathbf{construct\_year}|\mathbf{city})$ is that we suppose the construction year of an apartment correlates with the city it is located in, as different cities find themselves in different development stages. Also, we assume that attributes $\mathbf{A}_h \backslash \{construct\_year, city\}$ are conditionally independent with *construct_year* given a assignment of *city*. Note that these assumptions do not necessarily correspond to the truth and only serve for illustration purposes.

As we have no external knowledge on the exact correlation between *construct_year* and *city* nor on their prior distribution, the conditional probability $Pr(construct\_year|city)$, implemented by f, is evaluated by incorporating occurrence statistics, i.e. by counting the number of occurrence in D_h for each possible combination of assignment. However, since both cardinalities, $|dom(construct\_year)|$ and $|dom(city)|$, are very high in relation to the size of D_h, tuples that contain a certain assignment pair $(\mathbf{construct\_year}, \mathbf{city})$ are rarely, if ever, observed in D_h.

In this case, it is very difficult to implement f in a way such that it gives a reasonable marginal probability value for every possible combination of **construct_year** and **city**. Considering, for instance, the assignment pair $(2010, Muenchen)$, it is impossible for f to plausibly

measure $Pr(construct\_year = 2010|city = Muenchen)$ if 2010 occurs only once and *Muenchen* twice in D_h.

To address this problem, an intuitive approach would be to seek "neighbors" of *Muenchen*, i.e. to find a subset $\mathbf{M} \subseteq dom(city)$ whose elements share similar characteristics with *Muenchen*. Once a reasonably sized $\mathbf{M}$ is obtained, f can evaluate $Pr(construct\_year = 2010|city = Muenchen)$ by taking tuples with assigned attribute value $\mathbf{city} \in \mathbf{M}$ into consideration instead of being stuck with the two tuples where *Muenchen* actually occurred.

The job of creating meaningful neighborhoods for categorical attributes is easier said than done, as the only available source of ground truth information is the noisy database itself. Moreover, since we rely on other attributes to measure the similarity between entities (e.g. "*Muenchen* and *Nuernberg* are similar cities because they share the same average rent."), the knowledge on correlations between attributes is mandatory for the success of seeking meaningful neighbors.

In summary, the idea is to leverage quantitative information and attribute-level correlations to an extent that enables us to also reason about categorical values that have rarely occurred in D_h. To achieve this objective, we look for techniques that are capable of extracting latent correlations between attributes in D_h despite its small size, uncertainties and lack of ground truth. Then, the latent correlations shall be exploited and brings the notion of similarity into categorical attributes.

3.3.2 Learning Embedding Vectors

We address the problem of high cardinality categorical attributes by proposing a neural network model for learning vector representations, also known as *embeddings*. The solution concept is to assign each categorical value an embedding vector in a high-dimensional space. Embedding vectors are positioned in the vector space such that categorical values that share common characteristics are located close to one another in the space.

Our model for learning embeddings is a *feed forward neural network* with an architecture that is inspired by *Word2vec* [38] [39], a group of related models that are popular in the area of *natural language processing* (NLP). Word2vec models take as input a large corpus of text and produces dense vector representations for words during a iterative process of reconstructing linguistic contexts.

In the following, we focus on the architecture of the *Skip-gram model*, an important and widely recognized model from the Word2vec family, and introduce our modified version of it. We explain the reason why it is considered as a crucial component of our PDB modeling approach, while we show similarities between the word embedding task from the NLP area and our task of handling categorical attributes with high cardinality.

The Skip-gram Model

The training objective of the original Skip-gram model is to find word representations that are useful for predicting the surrounding words of a given center word. It uses each word in the text corpus as an input to a log-linear classifier with continuous projection layer, and

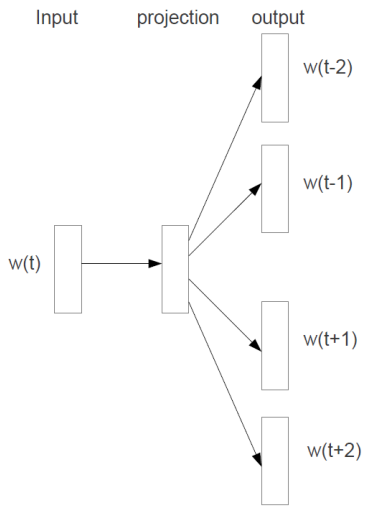


Figure 3.3: The Skip-gram model architecture [39]

predict words within a certain range before and after the current word. Given a sequence of training words $w_1, w_2, w_3, ..., w_T$, the formal optimization target of the Skip-gram model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leqslant j\leqslant c, j\neq 0}\log p(w_{t+j}|w_t) \tag{3.12}$$

where c is the window size of the training context. The probability $p(w_{t+j}|w_t)$ is defined using the softmax function

$$p(w_O|w_I) = \frac{\exp(u_{w_O}^{\mathsf{T}} v_{w_I})}{\sum_{w=1}^{W}\exp(u_w^{\mathsf{T}} v_{w_I})} \tag{3.13}$$

where v_{w_I} denotes the embedding vector of input word w_I and u_w denotes the weight vector of word w that is multiplied with v_{w_I} during softmax activation in the output layer. In Figure 3.3 a diagram of the Skip-gram architecture is shown [39].

The Skip-gram model has been profoundly analyzed by various research papers [27] [5] [37]. It is shown that words occurred in similar contexts receive similar embeddings, which is the core strength of the Skip-gram model that we want to borrow for our embedding task. More precisely, domains of categorical attributes are regarded as vocabularies, while each tuple is considered as a "sentence" that contains cell values ("words") that share common contexts. The aim is to use this contextual information, provided by the tuples, to train embedding vectors for categorical values. Vector spaces are established such that categorical values from the same attribute are mapped into the same vector space, where the distance between vectors implies the contextual similarity between them.

Differences between Continuous Text and Relational Data

Recalling the example database D_h in Figure 3.1 with the set of categorical attributes $\mathbf{A}_{h,categorical} = \{\text{state}, \text{city}, \text{zip}, \text{construct\_year}\}$ and the set of numerical attributes $\mathbf{A}_{h,numerical} = \{\text{living\_space}, \text{rent}\}$, the trained embeddings will be representing

$$dom(\mathbf{A}_{h,categorical}) = \{dom(state), dom(city), dom(zip), dom(construct\_year)\}.$$

To do this, $tuples(D_h)$ are used as source of context knowledge and fed iteratively into the embedding training model. An example training tuple is

$$t = (\text{state: “Bayern”, city: “Muenchen”, zip: “80753”, living\_space: 127.13,}$$
$$\text{construct\_year: “2001”, rent: 303.03),}$$

whereas a word sequence for training the Skip-gram model looks like

$$\mathbf{w} = \text{“The quick brown fox jumps over the lazy dog”.}$$

t is different from $\mathbf{w}$ in two aspects: t has a fixed size and contains both categorical and numerical attributes. For the Skip-gram model, the length of input word sequences is a hyper-parameter and influences the training process in terms of both accuracy and training time, whereas the size of $t \in tuples(D_h)$ is a constant equal to the number of attributes.

Regarding data value type, the Skip-gram model considers each input word sequence as a sequence of categorical values. It is not capable of incorporating information from numerical data during the training process, which makes it impossible for us to directly apply the model to our embedding learning task. The consideration is that, in relational data, numerical values belong to the context of a tuple and often carry crucial information.

For instance, if we neglect $\mathbf{A}_{h,numerical} = \{living\_space, rent\}$ and train vector representations for $dom(\mathbf{A}_{h,categorical})$ using only the categorical attributes, we are throwing away a great amount of contextual information and causing a decrease, if not a demolition, of the quality of the resulting embeddings. Therefore, our own model must accept both categorical and numerical values and exploit such contexts of mixed types in a meaningful and effective way.

Another difference between text corpus and $dom(\mathbf{A}_{h,categorical})$ is that the latter has a clear tabular structure with separate vocabularies, while the former is a unstructured collection of words. The way the Skip-gram model handles input text corpus is to establish a solitary vocabulary for it and to initialize a single embedding matrix that contains trainable vector representations for distinct words from the vocabulary. During the training process, words are treated equally without being divided based on their type (e.g. nouns, verbs and adjectives) or pre-classified using prior knowledge.

In our problem setup, $dom(\mathbf{A}_{h,categorical})$ are vocabularies that are separate by the nature of attributes, while each vocabulary implies its own notion of similarity. For instance, the vector space for *city* does not necessarily encode the same notion of similarity as the vector space for *state*. For our PDB model, we are ultimately interested in the contextual information between values of the same attribute and not in cross-comparing values of different attributes. For this purpose, our proposed model creates for each categorical attribute a separate vocabulary

and gives special attention to leveraging structural information, which are content that is not included in the scope of the Skip-gram model.

In section 5, the optimization target, architecture and training process of the proposed model are presented.

3.4 Probabilistic Programming with *Gen*

In contrast to relational databases that can be modeled and maintained by a variety of industrially reliable *relational database management systems* like Oracle Database or MySQL, there has not yet been a standard software system or a modeling paradigm for probabilistic databases. Most existing frameworks related to PDB models are evolving research projects [42][13][44] whose effectiveness and applicability have not yet been fully verified in real-world applications.

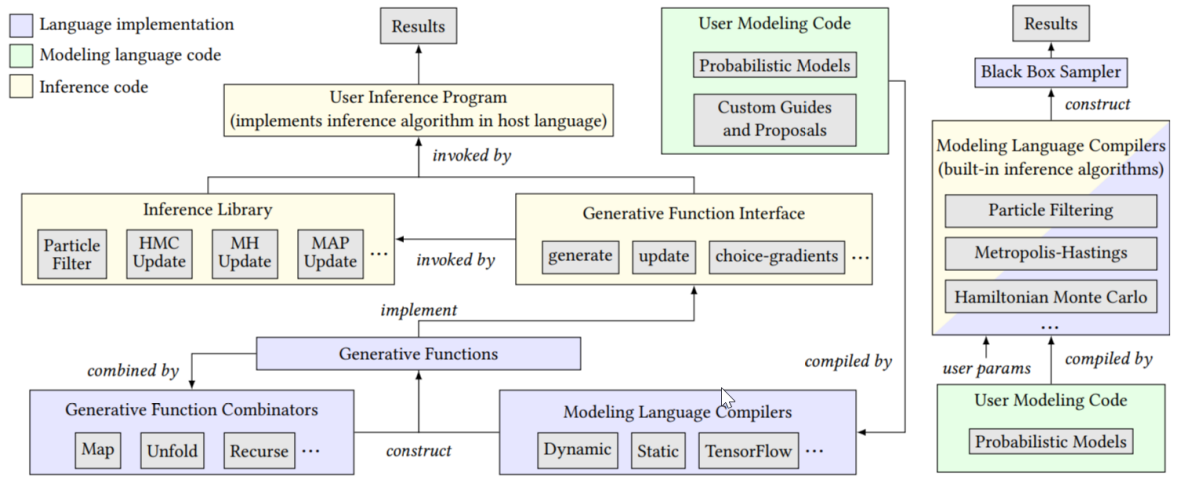
In this thesis, the software of our PGM-based modeling framework is established on the basis of *probabilistic programming*. Probabilistic programming languages (PPLs) describe probabilistic models and the mechanics to perform inference in those models. They are powerful tools for dealing with uncertainties regarding their competence of combining the inference capabilities of statistical methods with the representational power of programming languages.

Traditionally, constructing probabilistic models is an expert-only process that is done by hand in mathematical notation on paper. The transition process of such models into software programs often involves implementing the basic elements of probability theory from scratch, e.g. random variables, probability distributions and inference algorithms. A probabilistic programming language provides elementary building blocks for probabilistic models and abstractions that allow users to implement custom sampling and inference algorithms. In a PPL-program, assumptions relating to observed data are encoded with prior distributions over the variables of the model. During execution, the posterior distributions of the parameters of the model can be computed based on observed data by an automatic inference procedure. In our context, this allows us to construct graphical models and adjust, or even extract, prior knowledge precisely from the observed data tuples via inference.

Gen and its Architecture

Our software framework is realized with *Gen* [15], a novel general-purpose probabilistic programming system with language constructs for modeling and for user customization and optimization of inference. *Gen* comes as a package for the *Julia* programming language and consists of multiple modeling languages that are implemented as *domain-specific languages* (DSLs) in Julia and a Julia library for inference programming.

Like other probabilistic programming languages, *Gen* includes universal modeling languages that can represent any model, including models with stochastic structure, discrete and continuous random variables, and simulators. However, *Gen* is distinguished by the flexibility that it affords to users for customizing their inference algorithm. It is possible to use built-in



(a) *Gen*'s architecture (b) Standard architecture

Figure 3.4: Comparison of *Gen*'s architecture to a standard probabilistic programming architecture [15].

algorithms that require only a couple lines of code, as well as develop custom algorithms that are able to meet scalability and efficiency requirements.

Two major differences between *Gen* and other PPLs can be determined in its architecture, depicted in Figure 3.4. First, *Gen* uses the *generative function interface*, a black box abstraction for probabilistic computations, to separate low-level operations from the implementation of custom inference algorithms. Second, it contains *generative function combinators*, which produce efficient generative functions by exploiting common patterns of repeated computation and conditional independence that arise in probabilistic models.

The implementation process of probabilistic models with *Gen* is shown in Figure 3.5, where a *generative function* is implemented for the running PDB example D_h from Figure 3.1.

To define probabilistic models with *Gen*, users write generative functions in *probabilistic DSLs* that generate realizations of stochastic processes. Generative functions represent computations that accept some arguments, use randomness internally, return an output, and cannot mutate externally observable state. They are identified by a `@gen` annotation in front of a regular Julia `function` and make random choices with functions that sample from an extensible set of probability distributions. In Figure 3.5, `choose_proportionally(elements, probs)` is a probabilistic DSL that draws a random value from `elements` with associated probabilities `probs`.

Each execution of a generative function generates an *execution trace* (trace). A trace contains the values of random choices that are sampled during function call and annotated with `@trace`. Additionally, it contains a *weight* value that is the log-likelihood of the joint probability, i.e. the product of probabilities of the sampled random choices.

The generative function `generate_rent_tuple` models a Bayesian Network for D_h by translating factors into probabilistic DSLs and sampling random choices for $\mathbf{A}_h$ according

```
@gen function choose_proportionally(elements, probs)
   index = @trace(categorical(probs), :index)
   return elements[index]
end

@gen function generate_rent_tuple()
   state = @trace(choose_proportionally(states, prob_states), :state)

   prob_cities = normalize(number_occurrence(cities, [state]))
   city = @trace(choose_proportionally(cities[state], prob_cities), :city)

   prob_zips = normalize(number_occurrence(zips, [state, city]))
   zip = @trace(choose_proportionally(zips, prob_zips), :city)

   mean_l_s = mean(living_spaces[state, city])
   std_l_s = std(living_spaces[state, city])
   living_space = @trace(normal(mean_l_s, std_l_s), :living_space)

   prob_c_y = normalize(number_occurrence(construct_years, [state, city]))
   construct_year = @trace(choose_proportionally(construct_years, prob_c_y), :construct_year)

   mean_rent = mean(rent[living_space, construct_year])
   std_rent = std(rent[living_space, construct_year])
   rent = @trace(normal(mean_rent, std_rent), :rent)
   return rent
end
```

Figure 3.5: An illustrative generative function in *Gen* for modeling the example database D_h in figure 3.1.

to their probability distributions. For instance, in Line 2, a `state` value is drawn from `states` (that is $dom(state)$) with prior probabilities given in Figure 3.1b. In Line 7-8, as no external knowledge on distribution of $dom(zip)$ is available, the generative function seeks co-occurrences of $\mathbf{zip} \in dom(zip)$ and the already sampled `state` and `city` in D_h. Then, co-occurrence numbers are normalized to a valid categorical distribution and sampled from.

4 Probabilistic Modeling Framework

In this chapter, we provide an overview of our probabilistic modeling framework and describe its implementation. In Figure 4.1, the overall framework structure is shown.

In the current implementation, our framework takes as input an unclean database instance D along with a set of integrity constraints Σ that is the sole external information available for anomaly detection. As the framework focuses on databases with limited external knowledge, Σ only contains functional dependencies (FDs) and matching dependencies (MDs). Recall that FDs correspond to attribute-level dependencies pre-assumed for D, and MDs specify look-ups to available external dictionaries or labeled (clean) data.

In the following sections, we describe how Σ is translated into *Gen*-specific generative functions and how to chain these generative functions to create a holistic probabilistic model for D that resembles a Bayesian network. In addition, we show how occurrence statistics of solitary attributes can also be incorporated with generative functions.

From now on, let p denote the probability measure for $tuples(D)$, and $\mathbf{A} = \{A_1, ..., A_k\}$. For each $t \in tuples(D)$,

$$p(t) = Pr(t.A_1, ..., t.A_k) = \prod_{f \in \mathbf{F}} f(t.\mathbf{A}_f), \tag{4.1}$$

where $\mathbf{A}_f$ is the set of arguments taken by f.

4.1 Generative Functions for Functional Dependencies

In this section, generative functions that implement FDs in our PPL-program are presented. First, we introduce the associated mathematical background and give a general implementation. Then, we show how high-dimensional embeddings, which represent categorical values in D, can be leveraged in generative functions for FDs.

4.1.1 Mathematical Background

Let $c : \mathbf{A}_{left} \rightarrow \mathbf{A}_{right}$ denote an FD, where $\mathbf{A}_{left}$ is the set of attributes that are antecedents of the condition and $\mathbf{A}_{right}$ the set of attributes that are consequents of the condition and conditionally independent given $\mathbf{A}_{left}$. Let $\mathbf{a}_{left}$ denote the determined values of $\mathbf{A}_{left}$. Furthermore, we assume c to be the only external knowledge about the confidence in $\mathbf{A}_{right}$.

To encode the knowledge of c in probability measure p, a conditional probability factor is established and denoted as f_c. As c is the only available external information on $\mathbf{A}_{right}$, f_c considers co-occurrences of $\mathbf{A}_{right}$ and $\mathbf{A}_{left}$ in $tuples(D)$ as source of evidence. To extract

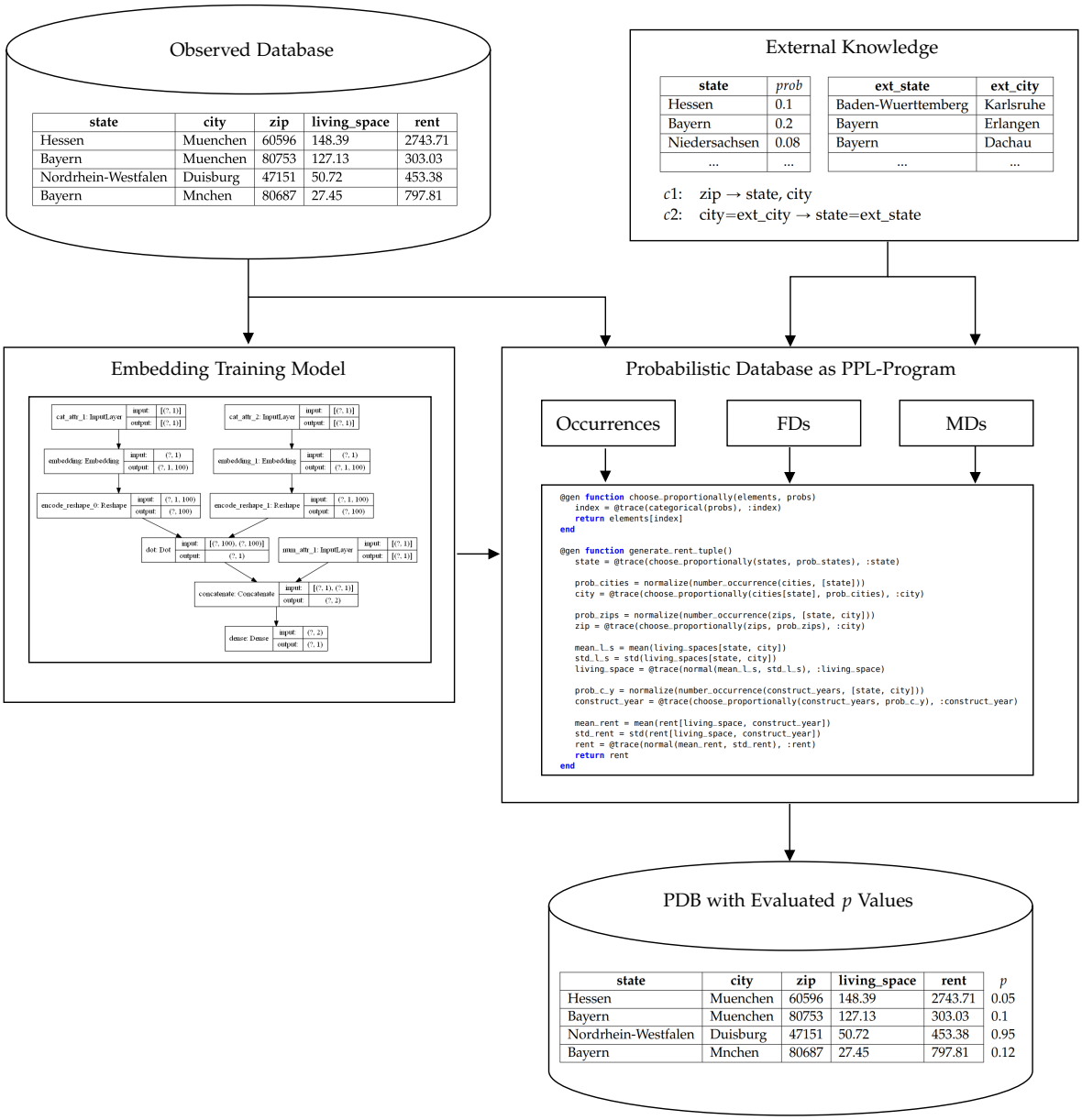


Figure 4.1: An overview of the proposed probabilistic modeling framework for unclean databases. The observed database along with a collection of functional dependencies and matching dependencies are passed on as input to construct a PPL-program, while vector representations for categorical values are learnt in a Python program and passed on to the PPL-program.

co-occurrence statistics from D, consider a subset $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \subseteq D$ that contains tuples with antecedent values that are either identical with, or similar to, $\mathbf{a}_{left}$. That is,

$$\forall t \in tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}) : t.\mathbf{A}_{left} \approx \mathbf{a}_{left}, \tag{4.2}$$

where $\approx$ determines whether $t.\mathbf{A}_{left}$ and $\mathbf{a}_{left}$ are similar based on certain matching criteria.

The conditional probability factor $f_c(\mathbf{A}_{right}|\mathbf{A}_{left} \approx \mathbf{a}_{left})$ is defined as

$$f_c(\mathbf{A}_{right}|\mathbf{A}_{left} \approx \mathbf{a}_{left}) = Pr(\mathbf{A}_{right}|\mathbf{A}_{left} \approx \mathbf{a}_{left}). \tag{4.3}$$

Due to conditional independence, f_c can be rewritten into the product form

$$f_c(\mathbf{A}_{right}|\mathbf{A}_{left} \approx \mathbf{a}_{left}) = \prod_{A_i \in \mathbf{A}_{right}} Pr(A_i|\mathbf{A}_{left} \approx \mathbf{a}_{left}). \tag{4.4}$$

To evaluate $Pr(A_i|\mathbf{A}_{left} \approx \mathbf{a}_{left})$, two different cases must be taken into consideration: If A_i is a numerical attribute, $Pr(A_i|\mathbf{A}_{left} \approx \mathbf{a}_{left})$ is a continuous distribution, whereas it is a discrete distribution if A_i is categorical.

Numerical Consequent

Selecting probability distribution for modeling a continuous numerical random variable is a task specific to both prior knowledge and observed samples of the variable. In our current implementation for the general case without other assumptions, $Pr(A_i|\mathbf{A}_{left} = \mathbf{a}_{left})$ with numerical A_i is a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with

$$\mu = \frac{1}{|tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}})|} \sum_{t \in tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}})} t.A_i \tag{4.5}$$

and

$$\sigma^2 = \frac{1}{|tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}})|} \sum_{t \in tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}})} (t.A_i - \mu)^2. \tag{4.6}$$

Categorical Consequent

In case that A_i is a categorical attribute, $Pr(A_i|\mathbf{A}_{left} = \mathbf{a}_{left})$ is a categorical distribution $categorical(\boldsymbol{\theta})$, where $\boldsymbol{\theta} = \{\theta_1, ..., \theta_n\}$, $n = |dom(A_i)|$ is a list of probabilities associated with $dom(A_i)$. The probability θ_i for the i^{th} value $a_i^{(i)} \in dom(A_i)$ is calculated as

$$\theta_i = \frac{|tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}, A_i \approx a_i^{(i)}})|}{|tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}})|}, \tag{4.7}$$

where $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}, A_i \approx a_i^{(i)}}$ is a subset of $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$, and

$$\forall t \in tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}, A_i \approx a_i^{(i)}}) : t.A_i \approx a_i^{(i)}. \tag{4.8}$$

In summary, the knowledge of a functional dependency c is incorporated in p via f_c, which is comprised of a set of multipliers, i.e. $Pr(A_i|\mathbf{A}_{left} \approx \mathbf{a}_{left})$, $A_i \in \mathbf{A}_{right}$. Each multiplier is modeled with a generative function, whose implementation is introduced in the upcoming sections.

4.1.2 General Implementation

Algorithm 1: Generative function for FDs

Input: Antecedents $\mathbf{A}_{left}$, determined values $\mathbf{a}_{left}$, consequent A_{right}, observation $tuples(D)$

Output: Sampled value a_{right}, probability p

`/* Find all tuples with desired co-occurred values` $\mathbf{a}_{left}$ `*/`

1 $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \leftarrow get\_subset(tuples(D), \mathbf{A}_{left}, \mathbf{a}_{left})$

`/* Case distinction for consequent */`

2 **if** A_{right} *is numerical attribute* **then**

3 $\quad \mu \leftarrow$ mean of $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}.A_{right}$

4 $\quad \sigma \leftarrow$ standard deviation of $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}.A_{right}$

5 $\quad a_{right}, p \leftarrow normal(\mu, \sigma)$ `// Samples a value and gives its probability`

6 **else**

7 $\quad \boldsymbol{\theta} \leftarrow$ normalized occurrence statistics of $dom(A_{right})$ in $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$

8 $\quad a_{right}, p \leftarrow categorical(\boldsymbol{\theta})$

9 **end**

10 **return** a_{right}, p

The idea of translating FDs into generative functions of *Gen* is generalized with Algorithm 1. In line 1, a $get\_subset()$ function extracts $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \subset D$ by comparing $t.\mathbf{A}_{left}$, $t \in tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}})$, with $\mathbf{a}_{left}$ and collecting tuples that satisfy certain matching criteria depending on the type of attributes in $\mathbf{A}_{left}$.

For numerical attributes in $\mathbf{A}_{left}$, consider the *living_space* attribute in Figure 3.1 as an example. Knowing that *living_space* is a numerical attribute that accepts continuous values to two decimal places, there are much less apartment offers with the exact same *living_space* value than offers with close values. For numerical attributes, $get\_subset()$ would very likely yield no matching results if it does not incorporate the closeness between numerical values as a matching criterion. For this reason, $get\_subset()$ examines the type of each $A_{left} \in \mathbf{A}_{left}$ and, if A_{left} is numerical, seeks matches for a_{left} in the interval

$$[a_{left} - |\epsilon|, a_{left} + |\epsilon|], \tag{4.9}$$

where ϵ is a hyper-parameter specified during the modeling process.

For categorical attributes in $\mathbf{A}_{left}$, the mostly applied matching criterion is to search for tuples with the same $\mathbf{A}_{left}$ values as $\mathbf{a}_{left}$. However, as discussed in Section 3.3.1, this method would not be effective for high cardinality categorical attributes and databases with

limited amount of tuples. In the next sections, two variants of $get\_subset()$'s implementation are presented, where the first one uses the "exact" matching criterion and the second one leverages embeddings and seeks matches in neighborhoods.

Once $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$ is evaluated, the generative function determines $a_{right} \in dom(A_{right})$ by sampling from a probability distribution whose parameters are evaluated from $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$. Again, it is distinguished between categorical and numerical A_{right}. In case of a numerical attribute, a_{right} is drawn, under our current setting, from a normal distribution with mean and variance calculated from $tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}).A_{right}$. Note that the normal distribution can be altered to another continuous probability distribution which A_{right} assumedly obeys.

If A_{right} is categorical, a_{right} is drawn from a categorical distribution with $\boldsymbol{\theta}$ evaluated from $tuples(D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}).A_{right}$: For values $\mathbf{a} \in dom(A_{right})$ that have occurred in $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$, the corresponding $\boldsymbol{\theta}$ are proportional to their occurrence statistics (see Equation 4.7).

For values in $dom(A_{right}) \backslash \mathbf{a}$ that are unseen in $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$, there are different options of how to calculate their probabilities: First, $dom(A_{right}) \backslash \mathbf{a}$ can be assigned zero probability, which is derived from the assumption that tuples with $(A_{left} : a_{left}, A_{right} : a_{right})$ are certainly anomalous if this value combination has not been observed in the database. Second, if the problem statement considers that the observed database does not rule out unseen value combinations, values in $dom(A_{right}) \backslash \mathbf{a}$ can be assigned a non-zero low_score, which makes them possible, but still less probable than $\mathbf{a}$. In general, we regard the choice between above options for handling unseen value combinations and the choice of low_score as context-specific tasks.

When making a random draw for A_{right}, the probability of the sampled value is stored in the execution trace. By chaining generative functions to implement an FD, the probability of a tuple regarding that FD is evaluated by calculating the product of the probabilities in each execution trace. In fact, the whole PDB model in our approach is implemented with chained generative functions that encode integrity constraints and other meaningful information for probability evaluation.

Matching without Embeddings

Algorithm 2: *get_subset*$(tuples(D), \mathbf{A}_{left}, \mathbf{a}_{left})$ without embeddings

Input: Antecedents $\mathbf{A}_{left}$, determined values $\mathbf{a}_{left}$, observation $tuples(D)$
Output: Desired subset $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$

```
1  D_{A_left ≈ a_left} ← ∅
2  for each t in tuples(D) do
3      flag ← true
4      for each (A_left, a_left) in (A_left, a_left) do
5          if A_left is categorical and t.A_left ≠ a_left then
6              flag ← false
7              break
8          else if A_left is numerical and t.A_left ≉ a_left then
9              flag ← false
10             break
11         end
12     end
13     if flag then
14         D_{A_left ≈ a_left} ← D_{A_left ≈ a_left} ∪ t
15     end
16 end
17 return D_{A_left ≈ a_left}
```

The implementation of *get_subset*() without incorporating embeddings is shown in Algorithm 2. In line 5-15, the function examines the type of A_{left} and appends matching tuples to the return array. The $\not\approx$ operation in line 8 considers an specified interval around a_{left}, this is,

$$\forall t \in tuples(D) : t.A_{left} \notin [a_{left} - |\epsilon|, a_{left} + |\epsilon|] \implies t.A_{left} \not\approx a_{left}. \tag{4.10}$$

The $\neq$ operation in line 5 for comparing categorical values strictly excludes tuples with unequal antecedent values, causing the shrinkage of the return array $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$ and thus a loss of potentially useful information for sampling consequent values.

Matching with Embeddings

Algorithm 3: $get\_subset(tuples(D), \mathbf{A}_{left}, \mathbf{a}_{left})$ with embeddings

Input: Antecedents $\mathbf{A}_{left}$, determined values $\mathbf{a}_{left}$, observation $tuples(D)$, size of neighborhood k
Output: Desired subset $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$

$D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \leftarrow \varnothing$
for *each t in tuples(D)* **do**
  $flag \leftarrow$ true
  for *each* (A_{left}, a_{left}) *in* $(\mathbf{A}_{left}, \mathbf{a}_{left})$ **do**
    if A_{left} *is categorical* ***and*** $t.A_{left} \neq a_{left}$ **then**
      $flag \leftarrow$ false
      break
    else if A_{left} *is numerical* ***and*** $t.A_{left} \not\approx a_{left}$ **then**
      $flag \leftarrow$ false
      break
    end
  end
  if *flag* **then**
    $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \leftarrow D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \cup t$
  end
end
$n\_matches \leftarrow |D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}|$
if $n\_matches < k$ **then**
  /* $\mathbf{a}_{left}$ is sparse, seek neighbors around it */
  $neighborhood \leftarrow$ A set $\{t_1, ..., t_{k-n\_matches}\} \subseteq tuples(D) \backslash D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$ that contains $(k - n\_matches)$ tuples whose values of $\mathbf{A}_{left}$ are the closest to $\mathbf{a}_{left}$ in the embedding space, measured with cosine similarity
  $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \leftarrow D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}} \cup neighborhood$
end
return $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$

The second implementation of $get\_subset()$ addresses the problem of diminishing matching tuples by establishing neighborhoods for $\mathbf{a}_{left}$ in case that $\mathbf{a}_{left}$ is sparsely observed in D. That is, if the number of matching tuples in $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$ is less than a specified size k, $get\_subset()$ will fill up $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$ by seeking tuples whose values of $\mathbf{A}_{left}$ are the closest to $\mathbf{a}_{left}$ in the high-dimensional embedding space.

Denoting the embedding vector for a categorical value a as $\mathbf{e}(a)$ with dimension dim_e, the

distance between two embeddings is measured with the cosine similarity

$$dist(\mathbf{e}(a^{(1)}), \mathbf{e}(a^{(2)})) = cos(\alpha) = \frac{\sum_{i=1}^{dim_e} e_i(a^{(1)})e_i(a^{(2)})}{\sqrt{\sum_{i=1}^{dim_e} e_i^2(a^{(1)})}\sqrt{\sum_{i=1}^{dim_e} e_i^2(a^{(2)})}}, \tag{4.11}$$

where α denotes the angle between $\mathbf{e}(a^{(1)})$ and $\mathbf{e}(a^{(2)})$. It is a measure of similarity between two non-zero vectors and judges orientation of the vectors and not their magnitude. The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, while in-between values indicate intermediate similarity or dissimilarity.

The enriched $D_{\mathbf{A}_{left} \approx \mathbf{a}_{left}}$ is then passed on to the generative function and used for parameter evaluation for probability distributions.

4.2 Generative Functions for Matching Dependencies

Matching dependencies are another kind of integrity constraints considered in this thesis and express relations between external dictionaries and attributes. Similar to FDs, MDs also consist of antecedents $\mathbf{A}_{left}$ and consequents $\mathbf{A}_{rights}$. However, they are different from FDs in that the evidence used to evaluate probability of $(\mathbf{A}_{left} : \mathbf{a}_{left}, \mathbf{A}_{right} : \mathbf{a}_{right})$ does not originate from the observed database but dictionaries that provide ground truth information on correctness of value combinations.

4.2.1 Mathematical Background

MDs are modeled with discrete conditional probability factors. Let $c : \mathbf{A}_{left} \rightarrow \mathbf{A}_{right}$ denote an MD, $\mathbf{a}_{left}$ the set of assigned values and $dict$ an external dictionary. Assume that $\mathbf{A}_{right}$ are conditionally independent given $\mathbf{A}_{left}$. The corresponding factor f_c is defined as

$$f_c(\mathbf{A}_{right}|\mathbf{A}_{left} = \mathbf{a}_{left}) = \prod_{A_i \in \mathbf{A}_{right}} Pr(A_i|\mathbf{A}_{left} = \mathbf{a}_{left}). \tag{4.12}$$

Again, it must be distinguished between numerical and categorical $A_i \in \mathbf{A}_{right}$ as they obey different probability distributions.

Numerical Consequent

For numerical A_i, $Pr(A_i|\mathbf{A}_{left} = \mathbf{a}_{left})$ is a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with

$$\mu = dict(\mathbf{a}_{left}).A_i \tag{4.13}$$

and

$$\sigma^2 = \sigma_{low}^2, \tag{4.14}$$

where $dict(\mathbf{a}_{left}).A_i$ is the lookup value of A_i given $\mathbf{a}_{left}$ in the dictionary, and σ_{low}^2 is a preset variance close to 0. The reason of σ_{low}^2 is to ensure the validity of $\mathcal{N}(\mu, \sigma^2)$ while allowing a sampled $a_i \in dom(A_i)$ to slightly deviate from the lookup value.

Categorical Consequent

For categorical A_i, $Pr(A_i|\mathbf{A}_{left} = \mathbf{a}_{left})$ is a categorical distribution with high probability of sampling the lookup value given by *dict* and low probability of sampling a value that violates *dict*. That is,

$$Pr(A_i = a_i|\mathbf{A}_{left} = \mathbf{a}_{left}) = \begin{cases} 1 - (|dom(A_i)| - 1) \cdot \theta_{low}, & dict(\mathbf{a}_{left}).A_i = a_i \\ \theta_{low}, & dict(\mathbf{a}_{left}).A_i \neq a_i \end{cases}, \tag{4.15}$$

where $\theta_{low} \in [0, 1]$ is specified regarding model configuration. If $\theta_{low} = 0$, any violation of c will directly result into assigning zero probability to the corresponding tuple, whereas violations of c are less punished and "allowed" if $\theta_{low} > 0$. Intuitively speaking, the determination of θ_{low} depends on the credibility of *dict*.

4.2.2 General Implementation

Algorithm 4: Generative function for MDs

Input: Antecedents $\mathbf{A}_{left}$, determined values $\mathbf{a}_{left}$, consequent $\mathbf{A}_{right}$, dictionary *dict*
Output: Sampled value a_{right}, probability p

```
1  a_right ← ∅
2  p ← ∅
3  for each A_right in A_right do
4  |   a_right ← dict(a_left).A_right
5  |   if A_right is numerical attribute then
6  |   |   a_right, p ← normal(a_right, σ_low)
7  |   else if A_right is categorical attribute then
8  |   |   θ ← [θ_low, ..., θ_low]
9  |   |   θ[a_right] ← 1 − (|dom(A_right)| − 1) · θ_low
10 |   |   a_right, p ← categorical(θ)
11 |   end
12 |   a_right ← a_right ∪ a_right
13 |   p ← p ∪ p
14 end
15 return a_right, p
```

To integrate MDs into a PPL-program, they are translated into generative functions following Algorithm 4. Depending on attribute type of each $A_{right} \in \mathbf{A}_{right}$, the generative function samples values $\mathbf{a}_{right} \in dom(\mathbf{A}_{right})$ from either a normal distribution (line 5) or a categorical distribution (line 7-9). In both situations, the probability of sampling $dict(\mathbf{a}_{left}).A_{right}$, i.e. the lookup value associated with $\mathbf{a}_{left}$ in *dict*, should be much higher than the probability of sampling a different value in order to make use of dictionary knowledge.

4.3 Generative Functions for Occurrence Statistics

Besides integrity constraints in Σ that contribute to modeling the database and its probability measure, external knowledge can also include reasoning about occurrence statistics of solitary attributes that are independent from other attributes. In this section, we discuss how such occurrence statistics can be included in the overall probability measure p via generative functions.

4.3.1 Mathematical Background

Let A denote an attribute whose occurrence statistics are to be incorporated in probability measure p of database D. The probability distribution over $dom(A)$ is parametrized with occurrences of $dom(A)$ in the observed database and realized with probability factor

$$f(A) = Pr(A). \tag{4.16}$$

Numerical Attribute

In case of A being a numerical attribute, $Pr(A)$ is a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with

$$\mu = \frac{1}{|tuples(D)|} \sum_{t \in tuples(D)} t.A \tag{4.17}$$

and

$$\sigma^2 = \frac{1}{|tuples(D)|} \sum_{t \in tuples(D)} (t.A - \mu)^2, \tag{4.18}$$

where μ and σ^2 are the observed mean and variance of values of A in D.

Categorical Attribute

Let $D_{A=a_i} \subseteq D$ be a subset of D, where $tuples(D_{A=a_i}).A = a_i$ and $a_i \in dom(A)$. If A is categorical, $Pr(A)$ is a categorical distribution $categorical(\boldsymbol{\theta})$ with $\boldsymbol{\theta} = \{\theta_1, ..., \theta_n\}$, $n = |dom(A)|$, where θ_i is proportional to the number of occurrence of a_i in D. That is,

$$\theta_i = \frac{|tuples(D_{A=a_i})|}{|tuples(D)|}. \tag{4.19}$$

4.3.2 General Implementation

Algorithm 5: Generative function for Occurrence Statistics

Input: Attributes **A**, observation $tuples(D)$
Output: Sampled value **a**, probability p

```
1  a ← ∅
2  p ← ∅
3  for each A in A do
4      if A is numerical attribute then
5          μ ← mean of tuples(D).A
6          σ ← standard deviation of tuples(D).A
7          a, p ← normal(μ, σ)
8      else if A is categorical attribute then
9          θ ← occurrence statistics of dom(A) in tuples(D)
10         a, p ← categorical(θ)
11     end
12     a ← a ∪ a
13     p ← p ∪ p
14 end
15 return a, p
```

With Algorithm 5, occurrences statistics of A are integrated into the PPL-program and contribute to p.

5 Embedding Training Model

In this chapter, we introduce our neural network model for training vector representations for categorical values of our PDB model.

5.1 Optimization Target

We first consider the case of applying the Skip-gram model directly to probabilistic databases. Let t denote a tuple from $tuples(D)$ where D is the considered database with attributes $\mathbf{A}$ and $\mathbf{A}_{categorical} \subset \mathbf{A}$. Let $e(t.A_i)$ denotes the embedding vector for the categorical value $t.A_i \in dom(A)$, $A \in \mathbf{A}_{categorical}$. Our model is a supervised neural network that, in each training iteration, regards t as input, selects the corresponding embeddings $\{e(t.A_i), ..., e(t.A_k)\}$ where $\{A_i, ..., A_k\} = \mathbf{A}_{categorical}$, and updates these embeddings based on the aggregate context that is contained in t.

Recalling the main cost function of the Skip-gram model (Equation 3.12, 3.13), for each t, the optimization objective is to maximize the average log-likelihood

$$J_{sg}(\theta) = \frac{1}{k} \sum_{t=1}^{k} J_i(\theta) \tag{5.1}$$

where

$$J_i(\theta) = \sum_{j \in \{1,...,k\} \setminus i} \log \frac{\exp(u_{t.A_j}^{\intercal} \, e(t.A_i))}{\sum_{w=1}^{|dom(A_j)|} \exp(u_w^{\intercal} \, e(t.A_i))} \tag{5.2}$$

is the cost for $t.A_i$ being the target and $\{t.A_j\}$, $j \in \{1, ..., k\} \setminus i$ being the context.

In Equation 5.2, the normalization factor

$$\sum_{w=1}^{|dom(A_j)|} \exp(u_w^{\intercal} \, e(t.A_i))$$

requires a sum over the entire vocabulary size $dom(A_j)$. If $|dom(A_j)|$ is large, this term becomes very expensive and slow to normalize every training tuple by summing over the outputs of every value in $dom(A_j)$. Considering that we are only interested in extracting contextual knowledge to solve the high cardinality problem, the ability of the Skip-gram model to estimate probability for every admissible value in $dom(A_j)$ of being the context is not necessarily needed.

Therefore, instead of predicting the context window given a target, we change the problem statement into a *binary classification problem*, where $tuples(D)$ are labeled as *positive samples*

and trained against. This solution very much resembles the *negative sampling* method [39] which concentrates on reinforcing embeddings of words that find themselves in the same input sequence, rather than propagating the contextual information of each input sequence through the entire embedding matrix.

As we no longer estimate probabilities for the complete vocabulary but consider a problem of classifying tuples into positive and negative samples, our model's output layer now uses the *sigmoid* function as activation and performs a *logistic regression* [33]. For each t, the log-likelihood becomes

$$J_t(\theta) = y_t \log h_\theta(x_t) + (1 - y_t)\log(1 - h_\theta(x_t)) \tag{5.3}$$

where $h_\theta(x_t)$ is the hypothesis

$$h_\theta(x_t) = p(y_t = 1|x_t; \theta) = \frac{1}{1 + \exp(-\theta^\intercal x_t)}, \tag{5.4}$$

x_t is the input vector that carries contextual information of t by merging its embeddings and numerical values, and y_t is the label of t. Finally, our aim is to maximize the overall objective function of our model

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} J_{t^{(i)}}(\theta) = \frac{1}{m}\Big[\sum_{i=1}^{m} y_{t^{(i)}} \log h_\theta(x_{t^{(i)}}) + (1 - y_{t^{(i)}})\log(1 - h_\theta(x_{t^{(i)}}))\Big] \tag{5.5}$$

where m is the number of positive samples ($|tuples(D)|$) and negative samples used for training.

5.2 Positive and Negative Tuples as Training Samples

The training process is realized with pairs of $(t^{(i)}, y_{t^{(i)}})$, where $t^{(i)}$ is an input tuple, $y_{t^{(i)}} \in \{0, 1\}$ is a label that indicates whether $t^{(i)}$ is a positive sample, and $i \in \{1, ..., m\}$ where m is the total number of training pairs, in our case $|tuples(D)|$. Under the assumption that $tuples(D)$ are mostly correct and do not introduce too much error [19], they are regarded as a credible source of contextual information. Thus we label $t \in tuples(D)$ with $y_t = y_{positive} = 1$ to appreciate its contextual relevance.

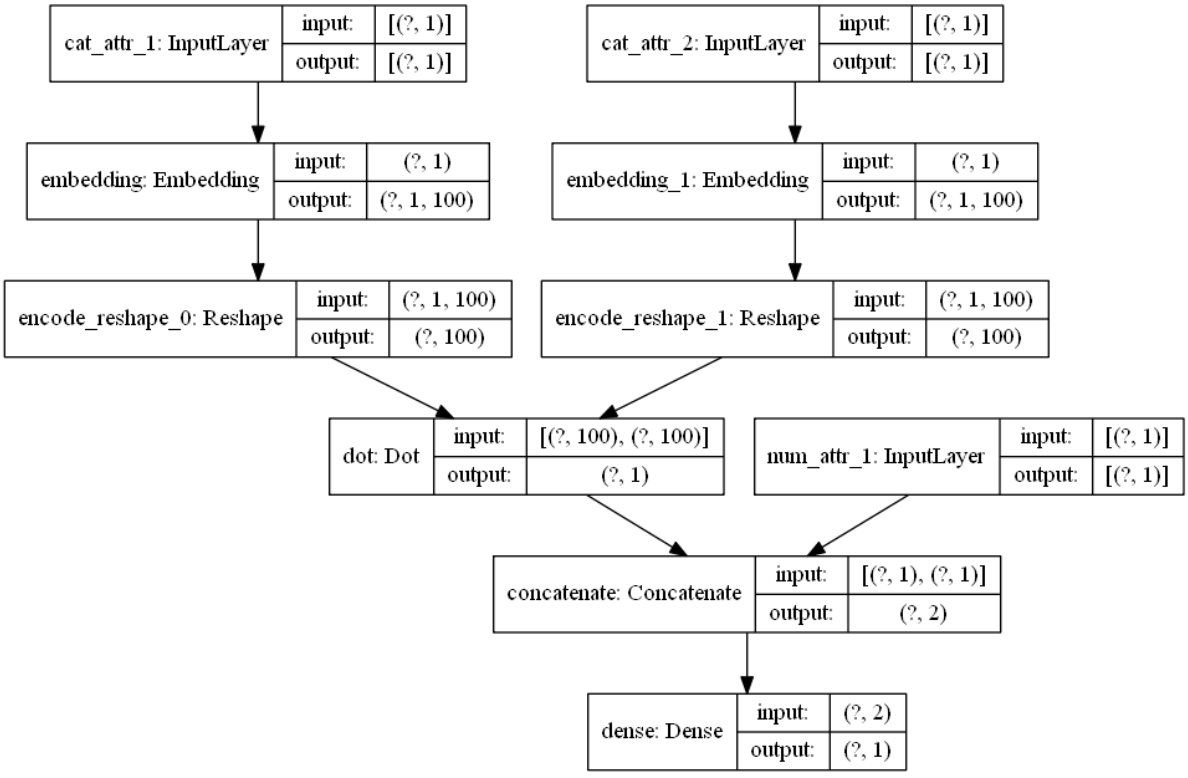
In addition to positive samples, *negative samples* are generated by duplicating a small subset $D_{negative}$ of D and inducing error in $t_{negative} \in tuples(D_{negative})$. For each $t_{negative}$, $t_{negative}.A_i$ is retained while $\{t_{negative}.A_j\}$, $j \in \{1, ..., k\}\backslash i$ are replaced by values from $dom(A_j)$ that have never occurred together with $t_{negative}.A_i$ in $tuples(D)$. This is to say, in contrast to our belief in the credibility of $tuples(D)$, values that have rarely or never co-occurred in $tuples(D)$ share less common contexts with each other and thus should have distant vector representations. From our model's perspective, its objective is to increase the similarity between embeddings of categorical values in $Cells[t_{positive}]$ while distancing embeddings of categorical values in $Cells[t_{negative}]$.

As we cannot feed categorical values directly into our network, they must be encoded to numerical values beforehand. Among various encoding techniques, we choose for simplicity

Layer (type)	Output Shape	Param #	Connected to
cat_attr_1 (InputLayer)	[(None, 1)]	0	
cat_attr_2 (InputLayer)	[(None, 1)]	0	
embedding (Embedding)	(None, 1, 100)	600	cat_attr_1[0][0]
embedding_1 (Embedding)	(None, 1, 100)	6800	cat_attr_2[0][0]
encode_reshape_0 (Reshape)	(None, 100)	0	embedding[0][0]
encode_reshape_1 (Reshape)	(None, 100)	0	embedding_1[0][0]
dot (Dot)	(None, 1)	0	encode_reshape_0[0][0] encode_reshape_1[0][0]
num_attr_1 (InputLayer)	[(None, 1)]	0	
concatenate (Concatenate)	(None, 2)	0	dot[0][0] num_attr_1[0][0]
dense (Dense)	(None, 1)	3	concatenate[0][0]

Total params: 7,403
Trainable params: 7,403
Non-trainable params: 0

(a) Summary



(b) Architecture

Figure 5.1: An example model with $|\mathbf{A}_{categorical}| = 2$, $|\mathbf{A}_{numerical}| = 1$, $k = 3$

and convenience the *integer encoding* method that assigns to each unique categorical value an integer label and creates a dictionary to save the assignment information. These integer labels do not imply any numerical meaning and only serve as identifiers for the dictionary.

5.3 Model Architecture

Understanding the architecture of the embedding training model is straightforward. It is essentially a *feed-forward neural network* with few layers.

To take in training samples $t^{(i)}$, $i \in \{1, ..., m\}$, it has n_{input} *input nodes*, where

$$n_{input} = |\mathbf{A}| = |Cells[t^{(i)}]|$$

is the total number of attributes of $t^{(i)}$. Note that $\mathbf{A} = \mathbf{A}_{numerical} \cup \mathbf{A}_{categorical}$.

Recalling that we establish for each categorical attribute a separate vocabulary, the input nodes that take in integer-encoded $\mathbf{A}_{categorical}$ are each connected to a dedicated node in the *embedding layer*. With a total of $n_{embedding} = |\mathbf{A}_{categorical}|$ nodes in the embedding layer, each node contains a trainable, randomly initialized embedding matrix that corresponds to the vocabulary of a categorical attribute $A_{categorical} \in \mathbf{A}_{categorical}$ and has the size of $(|dom(A_{categorical})|, dim_e)$ with dim_e denoting the dimension of embedding vectors. Note that due to model design, all embedding matrices have the same dim_e.

Given input integer labels, each node in the embedding layer selects its corresponding matrix row $\mathbf{e}$, which is the embedding of its input (the integer-encoded categorical value), reshapes $\mathbf{e}$ and passes it on to a *merge layer* that condenses similarity information between categorical values in $t^{(i)}$ by calculating the similarities between their embeddings.

For $n_{embedding}$ nodes in the embedding layer, the number of nodes in the merge layer n_{merge} is the binomial coefficient

$$n_{merge} = \binom{n_{embedding}}{2} = \frac{n_{embedding}!}{2!(n_{embedding} - 2)!}.$$

That is, for each unique pair of nodes in the embedding layer, a dedicated node in the merge layer extracts pair-wise similarity between the embeddings. Here, the considered measure is the *dot product* between vectors, which is defined as

$$\mathbf{e}^{(1)} \cdot \mathbf{e}^{(2)} = \sum_{i=1}^{dim_e} e_i^{(1)} e_i^{(2)},$$

where $\mathbf{e}^{(1)}$, $\mathbf{e}^{(2)}$ are two embeddings. The intention of using the dot product measure to merge embeddings is to ensure that categorical values that share common contexts end up having similar embeddings after the training process, whereas embeddings of categorical values in negative tuples eventually become distant from each other in the vector space.

The outputs of the merge layer converge at the *concatenation layer*, where $\mathbf{A}_{numerical}$ are also passed on to. Since $\mathbf{A}_{numerical}$ provide rich contextual information and, in our setting, do not

require vector representation, they are concatenated with the dot products from the merge layer in the concatenation layer. Now, we obtain a vector

$$x_{t^{(i)}} = \begin{bmatrix} \mathbf{e}^{(1)} \cdot \mathbf{e}^{(2)} \\ \vdots \\ \mathbf{e}^{(1)} \cdot \mathbf{e}^{(|\mathbf{A}_{categorical}|)} \\ \mathbf{e}^{(2)} \cdot \mathbf{e}^{(3)} \\ \vdots \\ \mathbf{e}^{(|\mathbf{A}_{categorical}|-1)} \cdot \mathbf{e}^{(|\mathbf{A}_{categorical}|)} \\ t^{(i)}.A_{numerical,1} \\ \vdots \\ t^{(i)}.A_{numerical,k} \end{bmatrix}$$

where $k = |\mathbf{A}_{numerical}|$. The number of elements in $x_{t^{(i)}}$ equals

$$\binom{|\mathbf{A}_{categorical}|}{2} + |\mathbf{A}_{numerical}|.$$

Recalling the overall objective function of the embedding training model (see equation 5.5), $x_{t^{(i)}}$ is the input for the binary classifier and depicts the complete contextual information of $t^{(i)}$.

Finally, $x_{t^{(i)}}$ is passed on to the *output layer*, where a node with sigmoid activation function performs classification by outputing a predicted label $\hat{y}_{t^{(i)}} = h_\theta(x_{t^{(i)}})$, which is either 1 or 0, and comparing it with the actual label $y_{t^{(i)}}$. The error of $(\hat{y}_{t^{(i)}}, y_{t^{(i)}})$ is evaluated using the *binary cross-entropy loss* defined as

$$L(\hat{y}_{t^{(i)}}, y_{t^{(i)}}) = -\big(y_{t^{(i)}} \log(\hat{y}_{t^{(i)}}) + (1 - y_{t^{(i)}}) \log(1 - \hat{y}_{t^{(i)}})\big) = -J_{t^{(i)}}(\theta).$$

The error is back-propagated through the network to adjust the weights in the output layer and the embedding matrices, which are extracted once the prediction accuracy of the model converges after certain training epochs.

In figure 5.1, an example embedding training model with $|\mathbf{A}_{categorical}| = 2$, $|\mathbf{A}_{numerical} = 1|$ and $dim_e = 100$ is depicted.

Note that under current architecture, $|\mathbf{A}_{categorical}| \overset{!}{\geqslant} 2$ as at least one node is required in the merge layer for proceeding. For cases where $|\mathbf{A}_{categorical}| = 1$ and $|\mathbf{A}_{numerical}| \geqslant 1$, this requirement could be fulfilled by converting $\mathbf{A}_{numerical}$ into categorical attributes through *bucketing*, i.e. splitting numerical values into intervals based on conditions or manual boundaries. However, to perform this assignment, we need to increase the overall complexity of the model by inducing more hyper-parameters (e.g. number of buckets and their boundaries) which again require additional reasoning and justification. As we focus more on the conceptual feasibility of embedding categorical attributes, we leave out this point for now and consider it as a possible extension that can be researched on in future work.

6 Experimental Setup and Results

In this chapter, we describe the datasets, metrics and experimental setups used to evaluate both the performance of anomaly detection and quality of the trained embeddings.

6.1 Datasets

6.1.1 Rental Apartments Dataset

This dataset consists of apartment rental offers and is used as running example (shown in Figure 3.1) throughout the thesis. Data tuples in the rental apartments (RA) dataset are manually generated using a sampling script that takes as input the ground truth distributions over attributes' domains and information on attribute-level dependencies.

The intuition behind the RA dataset is to tailor a simplistic dataset for the purpose of proving the fundamental concept of the PDB modeling framework, which is crucial considering its novelty and high difficulty of the addressed problem. An additional advantage of having this artificial dataset is the total control over its properties, including e.g. amount of induced uncertainties and ratio between number of tuples and cardinality of categorical attributes. This allows us to simulate various situations and difficulties that can occur in real-world datasets and thus enables a comprehensive evaluation of our approach.

Attribute A	**Attribute type**	**Cardinality** $\lvert dom(A) \rvert$
state	Categorical	5
city	Categorical	65
zip	Categorical	1316
living_space	Numerical	n.a.
construct_year	Categorical	150
rent	Numerical	n.a.

(a) Attributes and their properties

MD_1: state=ext_state → city=ext_city
MD_2: city=ext_city → zip=ext_zip
MD_3: zip=ext_zip → avg_rent=ext_avg_rent
MD_4: zip=ext_zip → std_rent=ext_std_rent

(b) Integrity constraints

Table 6.1: Attributes of the rental apartments dataset and integrity constraints used by the sampler during data generation.

The sampler for generating the RA dataset is a PDB model by itself, because it utilizes the same types of knowledge and is modeled in the same way as a PDB model intended for data cleaning. i.e. it is a generative function that follows certain integrity constraints. However, since this sampler is intended for generating correct data, it is given for each categorical attribute in Table 6.1a a dictionary containing admissible values and their unambiguous allocations, and a set of integrity constraints that comprehensively define attribute-level correlations, which is a great amount of external knowledge that is often unavailable for real-world datasets. In Algorithm 6, we show how external information is embedded in the generative function and how each artificial tuple is sampled.

Algorithm 6: Sampler for creating the RA database

Input: Dictionaries *dict*
Output: Sampled tuple t

$t \leftarrow \varnothing$
$t.state \leftarrow uniform\_categorical(dict.state)$ `// Sample a state value uniformly`
$t.city \leftarrow generative\_function\_MD(:state, t.state, :city, dict)$
$t.zip \leftarrow generative\_function\_MD(:city, t.city, :zip, dict)$
$t.living\_space \leftarrow uniform(dict.living\_space)$ `// Sample from a given range`
$t.construct\_year \leftarrow uniform\_categorical(dict.construct\_year)$
$avg\_rent \leftarrow dict.avg\_rent(t.zip)$
$std\_rent \leftarrow dict.std\_rent(t.zip)$
$additional\_cost \leftarrow dict.cost(t.living\_space) + dict.cost(t.construct\_year)$
$t.rent \leftarrow normal(avg\_rent + additional\_cost, std\_rent)$
return t

Until now, only positive tuples that comply with the specifications in Table 6.1 are being generated. In Section 6.1.3, we show how negative tuples are created by manually inducing anomalies. Negative tuples are then regarded as targets needed to be detected by the PDB model for data cleaning.

6.1.2 Mercateo Article Database

In contrast to the artificial RA dataset, the *Mercateo* article (Mercateo) database is a dump of real article data available on the Mercateo e-procurement portal. It contains a multitude of information related to offered articles including product categories, manufacturers, supplier IDs, prices, units etc.. While some of these attributes are provided directly by suppliers and thus can have various anomalies such as missing values, inconsistent units and manufacturer names, some other attributes are extracted from supplier specifications based on certain algorithms and can be misled by those anomalies, resulting in further untrue data.

Evaluating the Mercateo database is a particularly difficult task due to several reasons:

- In total, the Mercateo database contains millions of articles of different categories from different suppliers. Comparing the number of distinct products, manufacturers and

Attribute A	**Attribute type**	**Cardinality** $\lvert dom(A) \rvert$
article_id	Categorical	7489
catalog_id	Categorical	210
unit	Categorical	10
keyword	Categorical	167
manufacturer	Categorical	35
set_id	Categorical	200
price	Numerical	n.a.

Table 6.2: Attributes of the Mercateo article dataset with 9049 tuples.

other attribute domains to the overall database size, their cardinalities are too high. Additionally, as some products are offered more widely than some others, they have completely different occurrence statistics.

- Unlike many other datasets, where the correctness of a data value can be reasoned with its occurrences, quantitative statistics of article data do not necessarily imply correctness. That is to say, the fact that a certain manufacturer occurs less than the others must not indicate an error, but it can be merely due to uneven numbers of existing manufacturers on the market.

- Most attributes in the Mercateo dataset are categorical, while many of them are sparse ID numbers or other identifiers that encode catalogue, product set and other complex business-internal information. For these attribute domains, it is challenging to establish a measure of semantic similarity between cryptic values, which although would be particularly valuable for our PDB model as discussed in Section 3.3.1.

- Regarding ground truth information: Not only are there no labels on correctness of article data, but it is also hard to define whether a tuple is erroneous. For instance, if an article is given a price that is much higher than the average of its kind, this fact does not necessarily imply an error and can be seller's intention.

For the purpose of evaluating performance of our PDB modeling framework and quality of trained embeddings in real-world scenarios, and notably, under the aforementioned difficult conditions, we fetched 9049 articles that are pre-classified into 200 product sets by the system, and selected 7 attributes that are considered informative. They are listed i n Table 6.2. Note that article tuples with missing data are discarded for now as detecting them would be trivial. Text descriptions and attributes that involve language processing and information extraction techniques are consider out of scope and therefore also neglected.

6.1.3 Generating Anomalous Tuples

As both the RA and Mercateo database are without ground truth labels, anomalous tuples are manually generated based on observation and Algorithm 7. That is, observed tuples are

considered positive samples, while manual anomalies are negative samples that should be found out by the corresponding PDB model.

Algorithm 7: Sampler for generating anomalous tuples

Input: Observed dataset D, attributes $\mathbf{A}$, pivot attribute A_{pivot}, error rate parameter p
Output: Sampled negative tuple t

$t \leftarrow$ a random tuple from $tuples(D)$
/* Randomly induce negative values for attributes except A_{pivot} */
$a_{pivot} \leftarrow t.A_{pivot}$
$flag \leftarrow$ false
for A *in* $\mathbf{A} \backslash A_{pivot}$ **do**
    /* Sample a negative value with probability p */
    if $bernoulli(p)$ **then**
        $negative\_A\_values \leftarrow$ values in $dom(A)$ that have not co-occurred with a_{pivot}
        $t.A \leftarrow uniform\_categorical(negative\_A\_values)$
        $flag \leftarrow$ true
        continue
    end
end
if $!flag$ **then**
    /* No negative value was induced */
    $A \leftarrow uniform\_categorical(\mathbf{A})$
    $negative\_A\_values \leftarrow$ values in $dom(A)$ that have not co-occurred with a_{pivot}
    $t.A \leftarrow uniform\_categorical(negative\_A\_values)$
end
return t

The size of generated anomalies is determined by the number of function calls to the sampler. The amount of negative values in each negative tuple can be controlled with a parameter p that specifies the probability of inducing error at each attribute. When determining anomaly detection performance and embedding quality, we also vary p and the ratio between positive and negative tuples to evaluate their impact.

For the RA dataset, the assumption that all observed tuples are correct is justified by the fact that its sampling process is based on given ground truth and does not allow any violation of the integrity constraints. For the Mercateo database, however, the amount of noise existing in the observation is unclear, while there exists no simple method for measuring the noise level (which would intrinsically do anomaly detection). If the implicit noise level is high, a negative tuple generated based on observation could be "less erroneous" than an observed tuple and cause a missed detection, while the latter could cause a false alarm.

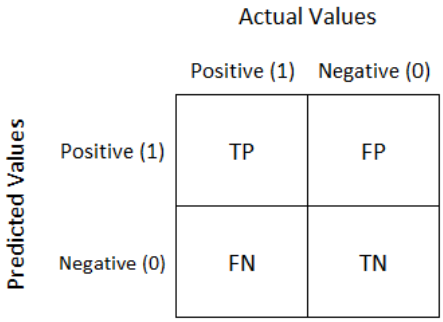


Figure 6.1: Confusion matrix

6.2 Anomaly Detection

In this section, we evaluate anomaly detection performance of the PDB models generated by the proposed probabilistic modeling framework for the RA and Mercateo database. The evaluation is conducted from the following perspectives:

- We show that our PDB models are capable of giving meaningful probability estimations for both datasets with minimal amount of external knowledge;
- We show that replacing categorical values with embeddings significantly increase detection performance;
- We investigate how availability of integrity constraints impacts detection performance;
- We investigate the effect of different dataset parameters.

6.2.1 Evaluation Methodology

The evaluation of anomaly detection performance is conducted by converting estimated probability values of tuples into binary labels indicating whether a tuple is positive or negative and reasoning about the resulting *confusion matrix* (Figure 6.1) that shows the number of correctly and incorrectly labeled tuples. Note that positive labels counter-intuitively indicate anomalous tuples, while clean tuples, which are previously referred to as positive tuples, should be given negative labels. Furthermore, as tuple probabilities are very likely to vanish during multiplication of factors, we take their *common logarithms* (with base equaling 10) during the entire evaluation process.

Two measures are used for the evaluation:

Definition 6.2.1 *(Recall) Recall is the fraction of true positives, i.e. correctly labeled anomalies, over the total number of negative tuples:*

$$Recall = \frac{TP}{TP + FN}. \tag{6.1}$$

Definition 6.2.2 *(Precision) Precision is the fraction of true positives over the total number of tuples labeled as positive:*

$$Precision = \frac{TP}{TP + FP}. \tag{6.2}$$

As there exists yet no general technique of seeking a meaningful probability threshold for labeling negative and positive tuples, we reason about performance of generated PDB models by sorting tuples after their probabilities and inspecting the distribution of anomalous tuples in this sorted list.

Two methods of converting sorted probabilities into positive and negative labels are considered. The first method is to label k tuples with the smallest estimated probabilities as positive, where k is the total number of anomalous tuples. For both datasets, k is always known because anomalous tuples are artificially generated. By doing so, the recall of a PDB model can be calculated by inspecting how many of the k identified tuples are actually anomalous.

The second applied method is to initialize $k = 1$, increase it step-wise and observe the behavior of precision and recall by drawing *receiver operating characteristic* (ROC) and *precision-recall* (PR) curves, which are commonly used measures for evaluating the predictive performance of classification algorithms [18]. In PR curves, precision is plotted against recall, whereas the true positive rate TPR is plotted against the false positive rate FPR in ROC curves. These quantities are defined as

$$\begin{aligned} TPR &= \frac{TP}{TP + FN}, \\ FPR &= \frac{FP}{FP + TN}. \end{aligned} \tag{6.3}$$

ROC curves quantify predictive performance in terms of the *area under the curve* (AUC) that lies in the range $[0, 1]$. In case of an ideal classifier, i.e. it does not make any prediction errors and thus achieves $TPR = 1$ before producing any false positives, its ROC curve would have an $AUC = 1$. The ROC curve of a random classifier that makes binary decision by tossing a fair coin would have an $AUC \approx 0.5$.

PR curves are an alternative to ROC curves and expose differences between algorithms that are not apparent in ROC space. They are especially meaningful for datasets with a large skew in the class distribution: If the number of negative examples greatly exceeds the number of positive examples, the number of false positives can greatly increase, which however would only lead to a small change of FPR used in ROC space. In contrast, this effect of large number of negative samples is captured by precision used in PR space.

PDB Parameters						Embedding Parameters		
$\|tuples(D)\|$	$\|\mathbf{A}\|$	k	$k/\|tuples(D)\|$	p	*neighborhood*	dim_e	*batch_size*	# Epochs
2976	6	149	0.05	0.2	10	100	1024	1000

Integrity Constraints	
FD_1:	state → city
FD_2:	state, city → zip
FD_3:	state, city → living_space
FD_4:	city, living_space → construct_year
FD_5:	city, living_space, construct_year → rent

(a) The RA dataset

PDB Parameters						Embedding Parameters		
$\|tuples(D)\|$	$\|\mathbf{A}\|$	k	$k/\|tuples(D)\|$	p	*neighborhood*	dim_e	*batch_size*	# Epochs
9049	7	452	0.05	0.2	10	100	4096	2000

Integrity Constraints	
FD_1:	keyword → unit
FD_2:	keyword → manufacturer
FD_3:	keyword, manufacturer, unit → set_id
FD_4:	catalog_id, set_id → price

(b) The Mercateo dataset

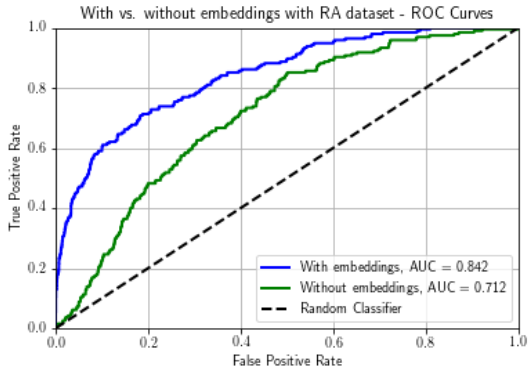
$\|tuples(D)\|$	Number of tuples in database D
$\|\mathbf{A}\|$	Number of attributes of D
k	Number of artificial negative tuples
p	Probability of inducing a negative value at each attribute (see Section 6.1.3)
neighborhood	Size of neighborhood formed by embeddings
dim_e	Dimension of embeddings
batch_size	Batch size during embedding training
# Epochs	Number of training epochs

(c) List of parameters

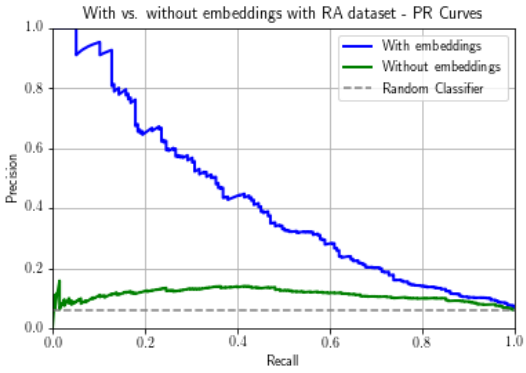
Table 6.3: Dataset and parameter configuration.

In our context, as both the RA and Mercateo datasets contain notably less anomalies than correct tuples, PR curves provide valuable insights into the performance of PDB models. The goal in ROC space is to be in the upper-left-hand corner, while the goal in PR space is to be in the upper-right-hand corner.

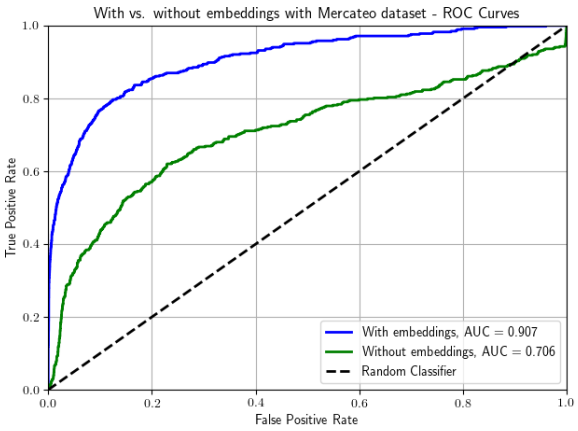
The parameter configuration of the RA and Mercateo dataset used for the experiments in this section are listed in Table 6.3.



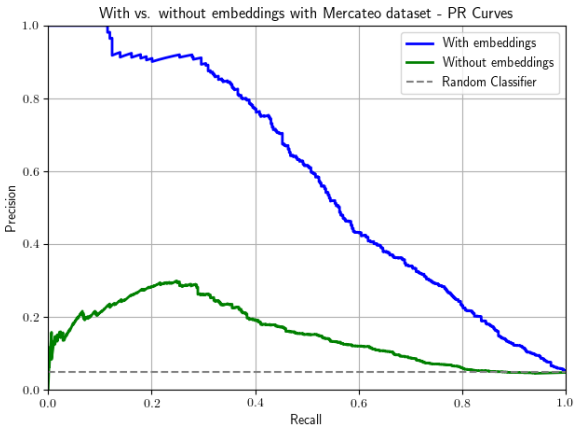
(a)



(b)



(c)



(d)

D	**With embeddings**	**Without embeddings**
RA	0.49	0.11
Mercateo	0.54	0.29

(e) Precision at $TP + FP = k$

Figure 6.2: General performance and effect of embeddings.

6.2.2 General Performance and Effect of Embeddings

The general anomaly detection performance of PDB models for the RA and Mercateo dataset is evaluated in Figure 6.2, where blue curves correspond to models that involve embeddings and green curves to models that operate only after integrity constraints and quantitative statistics. The precision, calculated by labeling the top k improbable tuples as positive, of both PDB models are listed in Figure 6.2e.

Without Embeddings

Under current dataset and parameter configuration, our PDB models are not performing well without embeddings.

For the RA dataset, although the green ROC curve in Figure 6.2a indicates that the model works better than a random classifier, this is mainly due to the enormous imbalance between negative and positive tuples. The green PR curve in Figure 6.2b shows that, after sorting the tuples after their estimated probabilities, the anomalous tuples are evenly distributed in the sorted dataset.

For the Mercateo dataset, a similar situation is observed. Comparing to the PDB model for the RA dataset, the green PR curve in Figure 6.2d shows better anomaly detection performance as the majority of anomalous tuples are located in the first half of the sorted dataset. However, the overall performance is poor and must be improved.

There are several possible reasons for the unsatisfying behavior of our models:

- As many categorical values occur sparsely in the RA and Mercateo dataset, there is no way for the PDB model to know which values and value combinations are correct. To solve this problem, one needs to either add more external knowledge, label data manually, or come up with a method to create neighborhoods around sparse categorical values, which is done by replacing them with embeddings in this thesis.
- If the integrity constraints do not comply with the latent ground truth, they add additional difficulty to the detection task. More details regarding the effect of integrity constraints are presented in the following sections.
- The original noise level in the datasets could be so high that observed tuples exceed manual corruptions in terms of "inplausibility" or "inprobability". In this case, the anomaly detection performance of a PDB model cannot be effectively evaluated with manually sampled anomalies, because the assumption of general credibility of what is observed does not hold anymore. Therefore, more human effort into data cleaning and knowledge gathering is required.

With Embeddings

For both RA and Mercateo dataset, the benefits brought by embeddings are significant.

In ROC space, the predictive performance of both PDB models drastically increase; In PR space, the fact that the PR curves initiate from the top-left-hand corner and remain in that

area within certain recall indicates the success of detecting the most improbable anomalies. However, the precision values gradually decrease with increasing recall, indicating that the PDB models are struggling with distinguishing between observed tuples and unobvious, anomalous tuples.

Detecting anomalous tuples that do not deviate significantly from the general distribution is a particular challenging task and has a deep impact on baseline performance. Although establishing neighborhoods drastically improved the PDB models, increasing baseline performance requires work on other peripherals such as integrity constraints and external knowledge, and fine-tuning of model's hyper-parameters according to specifications of the application domain.

6.2.3 Effect of Integrity Constraints

We evaluate how integrity constraints influence anomaly detection performance by reducing the sets of integrity constraints for both datasets. Instead of building PDB models using the integrity constraints in Table 6.3, they are replaced with integrity constraints that give minimal information on attribute-level correlations.

For the RA dataset, we replace FD_i, $i = 1, ..., 5$ in Figure 6.3a with

$$FD_{RA,minimal} : \text{state, city, zip, living\_space, construct\_year} \rightarrow \text{rent}, \tag{6.4}$$

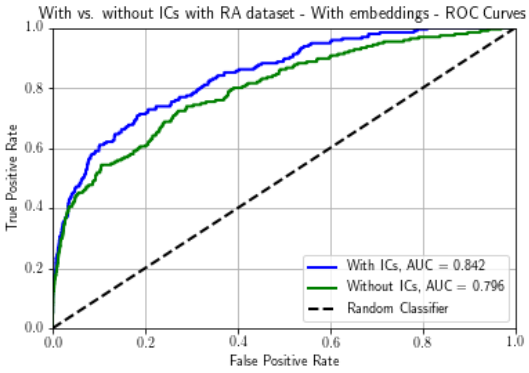
which states that the antecedents have influence on rent price but gives no other information such as correlations between the antecedents. For the Mercateo dataset, consider

$$FD_{Mercateo,minimal} : \text{catalog\_id, unit, keyword, manufacturer, set\_id} \rightarrow \text{price} \tag{6.5}$$

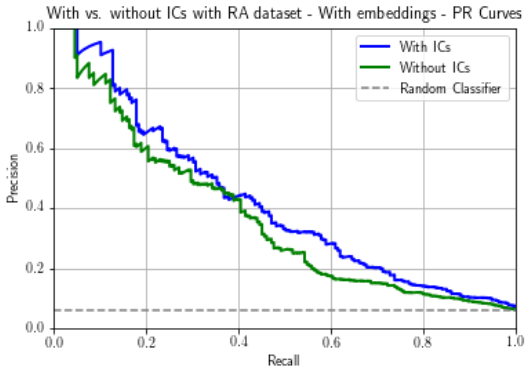
as the only available integrity constraint.

By inspecting the ROC and PR curves of the resulting PDB models in Figure 6.3 and 6.4, we come to the following observations regarding the reduction of integrity constraints:

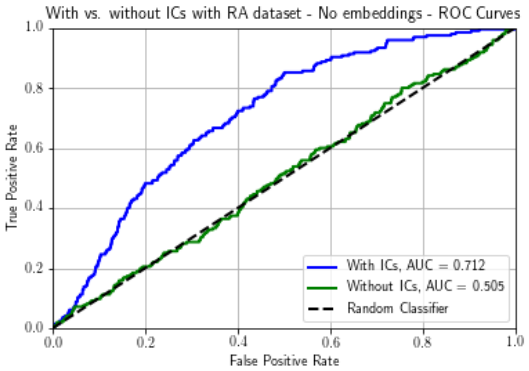
- Meaningful integrity constraints are a key factor for the success of anomaly detection. Their absence has decreased predictive performance for both datasets.
- The presence of embeddings compensates for a major portion of the information loss. Considering Figure 6.3a and 6.3b, the PDB model for the RA dataset that incorporates embeddings suffered marginally from the reduced integrity constraint and remains a meaningful classifier.
- The ROC and PR curve of the PDB model for the RA dataset, under the absence of embeddings and integrity constraints, show characteristics of a random classifier. This verifies the expectation that the model becomes a coin-tossing machine if it does not have access to any external knowledge.



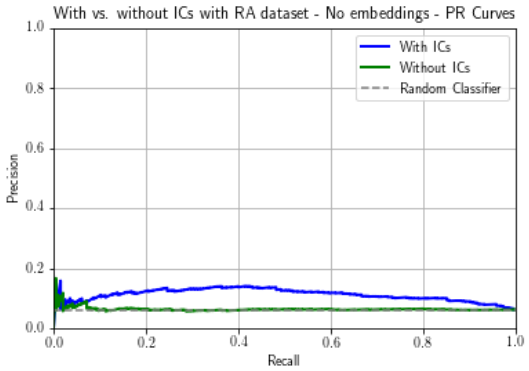
(a)



(b)



(c)

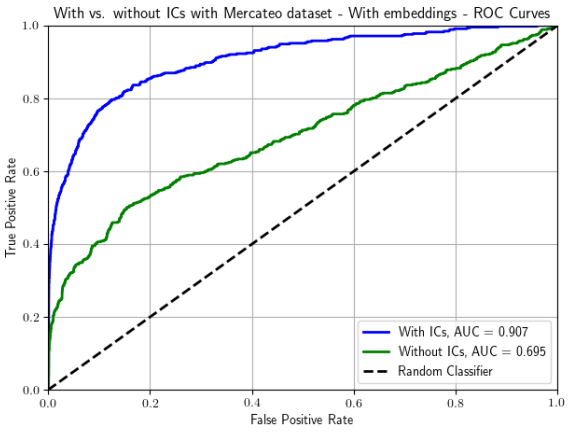


(d)

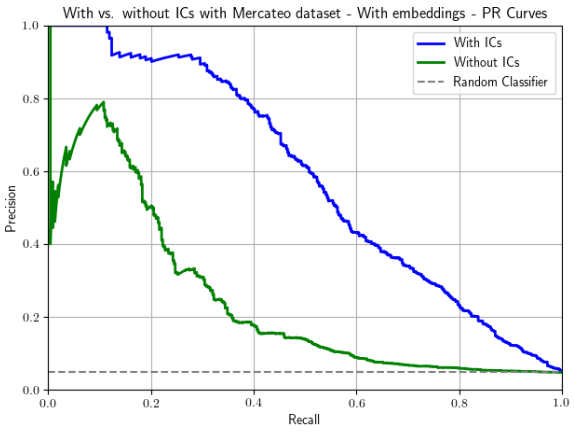
D	**With ICs**	**Without ICs**
RA, with embeddings	0.49	0.42
RA, without embeddings	0.11	0.07

(e) Precision at $TP + FP = k$

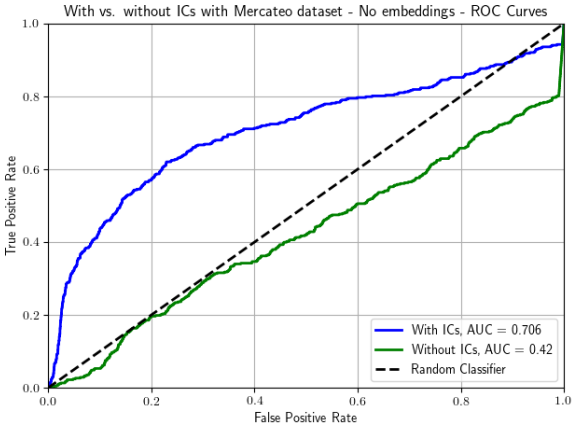
Figure 6.3: Effect of integrity constraints with the RA dataset.



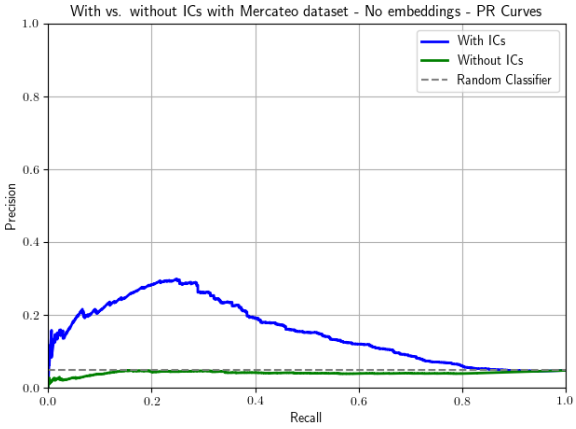
(a)



(b)



(c)



(d)

D	**With ICs**	**Without ICs**
Mercateo, with embeddings	0.54	0.29
Mercateo, without embeddings	0.30	0.02

(e) Precision at $TP + FP = k$

Figure 6.4: Effect of integrity constraints with the Mercateo dataset.

6.2.4 Effect of PDB Parameters

To investigate how different PDB parameters, i.e. characteristics of a considered dataset, influence anomaly detection performance of the PDB model, we uses the sampler, which created the RA dataset, to generate datasets of different sizes and properties.

In this section, the following parameters are considered potential influencing factors and therefore examined closely:

- $|tuples(D)|$, which is the size of a dataset, is strongly connected with the problem of high cardinality. For the same set of attributes $\mathbf{A}$ with constant cardinalities $|dom(\mathbf{A})|$, a large $|tuples(D)|$ means that categorical values are observed more frequently in D, and thus provides more statistical and relational evidence that can be leveraged by both our probabilistic modeling framework and our embedding training model. Therefore, a positive correlation between $|tuples(D)|$ and anomaly detection performance is expected.
- k, which is the number of manually sampled anomalies, can influence the overall performance due to the participation of anomalous tuples in the embedding training process. During training, the embedding training model has no information on whether a tuple is manually corrupted or actually observed. Therefore, the quality of embeddings depends on k and can have a negative impact on predictive performance of the PDB model.
- The probability for inducing each anomalous value in a negative tuple is defined by p. We expect p to be a primary "influencer" that determines detection performance, because PDB models can easily identify obvious anomalous tuples containing multiple conflicting values, whereas they struggle with tuples that deviate only slightly from what has been observed.

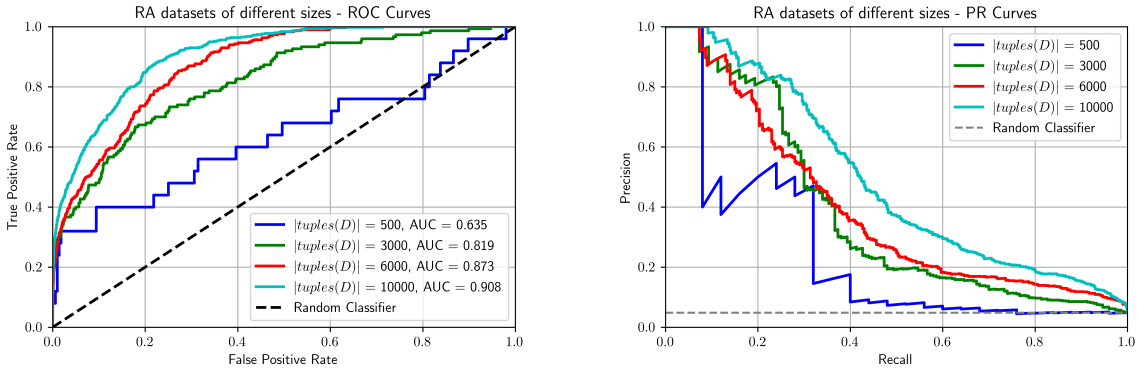
During evaluation process, we use the RA dataset and vary each of the mentioned parameter, while keeping other parameters fixed and unchanged from Table 6.3a. For each distinct parameter configuration, the embeddings are retrained. We use the same measures (ROC and PR curves) from the previous sections.

Number of Observed Tuples — $|tuples(D)|$

In Figure 6.5, a performance increase of the PDB models is noticed for larger datasets, where $|tuples(D)|$ compensates for part of the high cardinality problem.

By having richer occurrence statistics of categorical values, the PDB model gains evidence for identifying anomalous tuples that are less obvious. This is observed in the PR space in Figure 6.5b: For $|tuples(D)| = 500$, the PDB model comes close to a random classifier after detecting 40% of the anomalies. As $|tuples(D)|$ increases, predictive performance of unobvious tuples is gradually improved.

Furthermore, by letting the PDB model to classify the top k improbable tuples as positive (anomalous), the results in Figure 6.5c show positive correlation between $|tuples(D)|$ and precision.



(a) (b)

$\|tuples(D)\|$	**500**	**3000**	**6000**	**10000**
Precision at $TP + FP = k$	0.32	0.37	0.39	0.45

(c)

Figure 6.5: Effect of $|tuples(D)|$ in case of the RA dataset.

Ratio between Artificial Anomalies and Observed Tuples — $k/|tuples(D)|$

The impact of $k/|tuples(D)|$, the ratio between anomalous and observed tuples, on predictive performance is ambiguous in both ROC and PR space.

While the ROC curves in Figure 6.6a show no significant performance difference between PDB models of datasets with various $k/tuples(D)$, the PR curves of larger $k/|tuples(D)|$ are smoother. Before reaching a recall of 0.3, the PDB model with $k/|tuples(D)| = 0.01$ outperforms other models, then falls back rapidly, whereas the PR curve of $k/|tuples(D)| = 0.4$ converges to its baseline precision in a linear fashion.

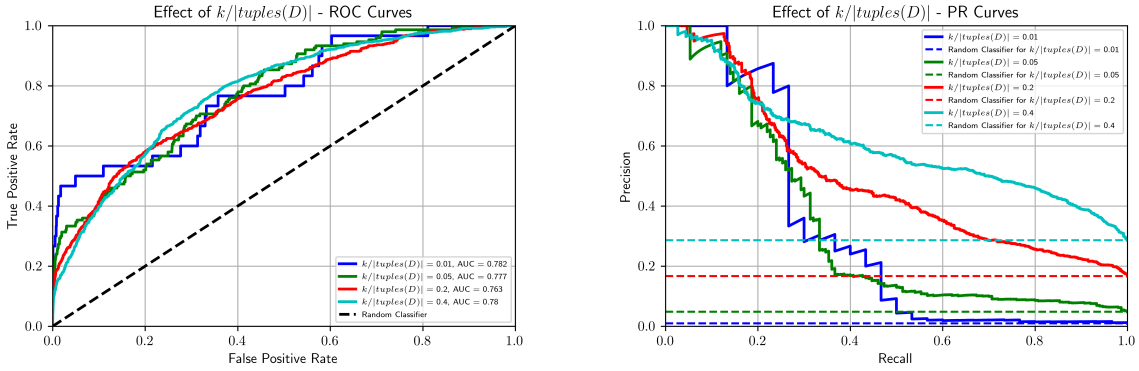
One possible reason for this inconsistent behavior is the change of baseline precision caused by the alternated balance between positive and negative tuples. In case of a large imbalance (small $k/|tuples(D)|$), the resulting precision would drastically decrease for successive false positives. If $k/|tuples(D)|$ is large, the precision decrease would not be that radical, which however does not mean better predictive performance.

Besides the ambiguities, a clear effect of mixing more manual anomalies into the observed dataset is that the PDB model becomes less accurate even when classifying the most improbable anomalies.

Probability of Error Induction — p

Coming to p, the probability of inducting a conflicting value at each attribute used by the anomaly sampler in Algorithm 7, it is observed that a large p has an unambiguous, positive effect on anomaly detection performance.

In Figure 6.7, both ROC and PR curves show that, for $p = 1.0$, the corresponding PDB

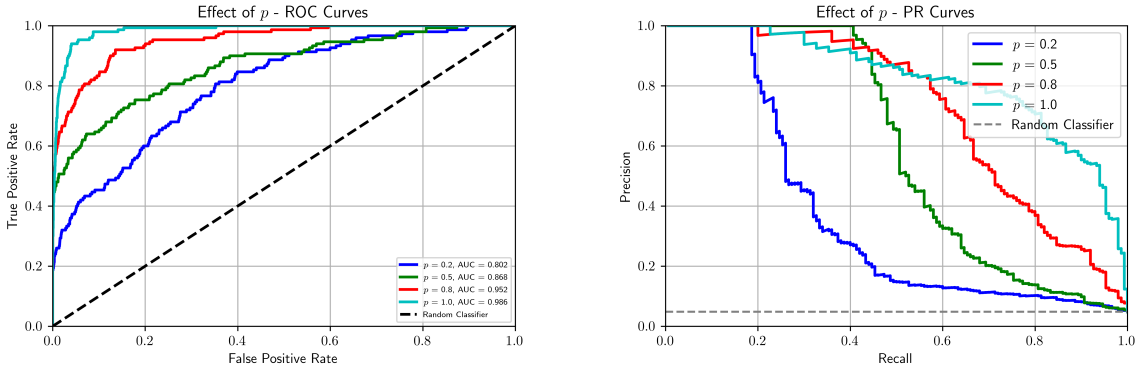


(a) (b)

$k/\|tuples(D)\|$	**0.01**	**0.05**	**0.2**	**0.4**
Precision at $TP + FP = k$	0.30	0.33	0.45	0.54

(c)

Figure 6.6: Effect of $k/|tuples(D)|$ in case of the RA dataset.



(a) (b)

p	**0.2**	**0.5**	**0.8**	**1.0**
Precision at $TP + FP = k$	0.33	0.51	0.65	0.76

(c)

Figure 6.7: Effect of p in case of the RA dataset.

model retains its initial high precision throughout the process and beats other models by large margins. This fact proves the effectiveness of the generated PDB models, particularly when deviation between the anomalies and clean tuples is high in a considered dataset.

6.3 Embedding Training

In this section, we investigate our embedding training model in terms of quality of learnt embeddings and effect of varying model parameters.

6.3.1 Embedding Quality

To measure the quality of embeddings, two major categories of evaluation schemes are present in the NLP area, where our model originates from. The first category includes *intrinsic* evaluation methods, which measure syntactic or semantic relationships between words by involving a pre-selected set of query terms and semantically related target words. On the contrary, *extrinsic* evaluation methods measure changes in performance metrics specific to downstream tasks [43].

In our application, an intrinsic evaluation of embedding quality is infeasible, as there exists no universal method of giving semantic meanings to categorical values. Moreover, since our neural network model is yet the only applicable method of learning embeddings from probabilistic databases, finding renowned studies and using them as benchmark are difficult.

To give a general feeling for the training performance, Figure 6.8 shows embeddings of the attribute *keyword* from the Mercateo dataset. Comparing to other categorical attributes, keywords are human readable values indicating an article's category. The associated embeddings are therefore easier to be judged based on semantic differences between the keywords.

6.3.2 Effect of Embedding Dimension

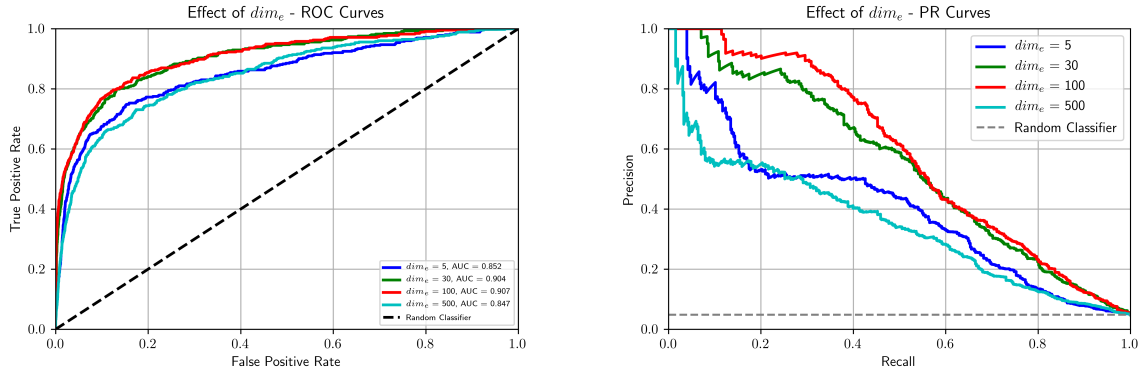
Choosing an appropriate embedding dimension dim_e has significant influence on performance of the downstream task, in our case, the predictive performance of an PDB model.

dim_e values that are too small or too large decrease anomaly performance by large margins, which can be observed in Figure 6.9b.

Another important finding is that high prediction accuracy of the neural network after training does not necessarily result in high predictive performance of the PDB model: For networks with $dim_e = \{5, 30, 100, 500\}$, their prediction accuracies on training data all converged to 1.0. However, 100-dimensional embeddings clearly outperform the others considering the PR curves.

Regarding time consumption, enlarging dim_e drastically increases training duration as the number of trainable parameters increases linearly with dim_e. It also results in longer execution time of the PDB model as it increases dimensionality of the neighborhood seeking process during each probability calculation.

Figure 6.8: From the Mercateo dataset, embeddings of the *keyword* values are visualized in a two-dimensional plane with t-SNE after 100 iterations with perplexity = 5. The figure shows ability of the model to learn meaningful relationships between categorical values in an unsupervised setup.



(a) (b)

dim_e	**5**	**30**	**100**	**500**
Precision at $TP + FP = k$	0.46	0.53	0.54	0.40
Execution time of PDB model [s]	113.06	120.51	125.41	134.20
Embedding training time [s]	32.81	56.35	93.38	393.33

(c)

Figure 6.9: Effect of dim_e in case of the Mercateo dataset.

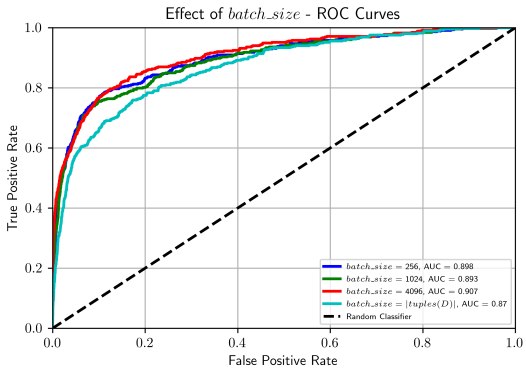
6.3.3 Effect of Batch Size

Similar to embedding dimension, the choice of batch size is an application-specific task. In general, training with larger batches require significantly more iterations in order to reach the same level of "fitness" as with smaller batches, because the calculated gradients of each sample tend to cancel out each other and result in a smaller gradient descent step.

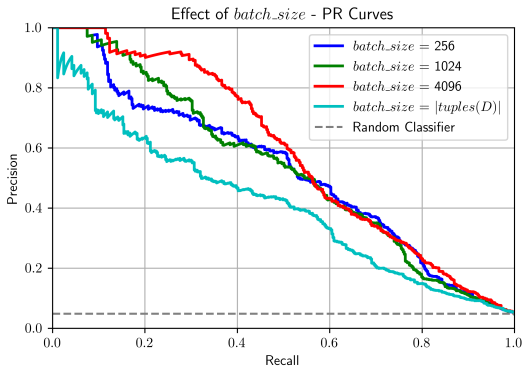
For the Mercateo dataset and with other parameters set according to Table 6.3, the predictive performance peaks with $batch\_size = 4096$ and degrades for larger batches. Results in the ROC and PR space are shown in Figure 6.10.

6.3.4 Implementation Details

The probabilistic modeling framework is implemented in Julia 1.3.1 with *Gen* of version 0.3.2. The embedding training model is implemented in Python 3.6.5 with TensorFlow 2.1.0. All experiments were executed on a machine with a quad-core 2.9 GHz Intel i7-7820HQ CPU and 16GB of RAM, running macOS 10.15. No GPU acceleration is involved in the experiments.



(a)



(b)

batch_size	**256**	**1024**	**4096**	$\|tuples(D)\|$
Precision at $TP + FP = k$	0.52	0.52	0.54	0.44
Embedding training time [s]	219.03	120.71	112.22	103.29

(c)

Figure 6.10: Effect of *batch_size* in case of the Mercateo dataset.

7 Summary and Outlook

In this thesis, we introduced a probabilistic modeling framework for unclean databases that is based on theories of probabilistic graphical models (PGMs) and realized with probabilistic programming language *Gen*. The framework is capable of generating probabilistic database (PDB) models useful for anomaly detection under considerably limited availability of external knowledge and ground truth information on the observed database.

In addition, we addressed the problem of categorical attributes with high cardinality by proposing a novel neural network model for learning vector representations, the so-called embeddings. We showed that embeddings significantly improve anomaly detection performance of the PDB models and compensate for a large amount of information loss in case of lacking integrity constraints.

We conducted practical experiments using two datasets. The synthetic rental apartments dataset allowed us to prove the concept and to permute model parameters to investigate their effect. Experiments with the Mercateo article dataset showed the effectiveness of the framework under real-world conditions.

The following research directions are left for the future due to time limitation:

- Our probabilistic modeling framework can be granted the ability of suggesting repairs for the detected anomalies thanks to the advantageous properties of PGMs and *Gen*. The framework allows custom inference algorithms and proposal functions driven by user-specified models or neural networks, meaning that embeddings not only benefit anomaly detection, but are also potentially useful for seeking probable repairs.

- The embedding training model should be further inspected regarding its structure and, in particular, the way it incorporates information from numerical data during training. Recalling that, in current implementation, a numerical value is concatenated with the pair-wise similarities between categorical values, the model requires the dataset to have at least two categorical attributes to accomplish the training process. This issue could be solved by bucketing numerical values into intervals and treating them as categorical values, though the effect of this process is unknown and needs further reasoning.

- An intrinsic evaluation measure of embedding quality needs to be established for better interpretation. Until now, embeddings of categorical values can only be evaluated based on downstream performance, which is sufficient for our application but not enough in general cases.

- Experiments with larger real-world datasets should be conducted to prove the effectiveness of the probabilistic modeling framework. They are left out due to limited time and local computational resources.

List of Figures

1.1 A variety of anomalies can occur in real-world datasets. 1

3.1 An example database D_h of apartment rental offers. (a) Relation instance R_h (b) External information on the probability distribution of $dom(state)$ (c) External information on the belonging between cities and states (d) Integrity constraints over S_h (e) Symbols used in R_h 8
3.2 An exemplary Bayesian network that models $p(t)$ from Equation 3.7 for running example D_h. 13
3.3 The Skip-gram model architecture [39] 16
3.4 Comparison of *Gen*'s architecture to a standard probabilistic programming architecture [15]. 19
3.5 An illustrative generative function in *Gen* for modeling the example database D_h in figure 3.1. 20

4.1 An overview of the proposed probabilistic modeling framework for unclean databases. The observed database along with a collection of functional dependencies and matching dependencies are passed on as input to construct a PPL-program, while vector representations for categorical values are learnt in a Python program and passed on to the PPL-program. 23

5.1 An example model with $|\mathbf{A}_{categorical}| = 2$, $|\mathbf{A}_{numerical}| = 1$, $k = 3$ 35

6.1 Confusion matrix 42
6.2 General performance and effect of embeddings. 45
6.3 Effect of integrity constraints with the RA dataset. 48
6.4 Effect of integrity constraints with the Mercateo dataset. 49
6.5 Effect of $|tuples(D)|$ in case of the RA dataset. 51
6.6 Effect of $k/|tuples(D)|$ in case of the RA dataset. 52
6.7 Effect of p in case of the RA dataset. 52
6.8 From the Mercateo dataset, embeddings of the *keyword* values are visualized in a two-dimensional plane with t-SNE after 100 iterations with perplexity = 5. The figure shows ability of the model to learn meaningful relationships between categorical values in an unsupervised setup. 54
6.9 Effect of dim_e in case of the Mercateo dataset. 55
6.10 Effect of *batch_size* in case of the Mercateo dataset. 56

List of Tables

3.1 Cardinalities of $A \in \mathbf{A}_{h,categorical}$ 14

6.1 Attributes of the rental apartments dataset and integrity constraints used by the sampler during data generation. 38
6.2 Attributes of the Mercateo article dataset with 9049 tuples. 40
6.3 Dataset and parameter configuration. 44

Bibliography

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. "Detecting data errors: Where are we and what needs to be done?" In: *Proceedings of the VLDB Endowment* 9.12 (2016), pp. 993–1004.

[2] D. Agarwal. "Detecting anomalies in cross-classified streams: a bayesian approach". In: *Knowledge and information systems* 11.1 (2007), pp. 29–44.

[3] C. C. Aggarwal. "Outlier analysis". In: *Data mining*. Springer. 2015, pp. 237–263.

[4] F. J. Anscombe. "Rejection of outliers". In: *Technometrics* 2.2 (1960), pp. 123–146.

[5] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski. "A latent variable model approach to pmi-based word embeddings". In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 385–399.

[6] L. Bertossi, S. Kolahi, and L. V. Lakshmanan. "Data cleaning and query answering with matching dependencies and matching functions". In: *Theory of Computing Systems* 52.3 (2013), pp. 441–482.

[7] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. "Conditional functional dependencies for data cleaning". In: *2007 IEEE 23rd international conference on data engineering*. IEEE. 2007, pp. 746–755.

[8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[9] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[10] S. Chawla and P. Sun. "SLOM: a new measure for local spatial outliers". In: *Knowledge and Information Systems* 9.4 (2006), pp. 412–429.

[11] J. Chomicki and J. Marcinkowski. "Minimal-change integrity maintenance using tuple deletions". In: *Information and Computation* 197.1-2 (2005), pp. 90–121.

[12] X. Chu, I. F. Ilyas, and P. Papotti. "Discovering denial constraints". In: *Proceedings of the VLDB Endowment* 6.13 (2013), pp. 1498–1509.

[13] X. Chu, I. F. Ilyas, and P. Papotti. "Holistic data cleaning: Putting violations into context". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE. 2013, pp. 458–469.

[14] E. F. Codd. "A relational model of data for large shared data banks". In: *Software pioneers*. Springer, 2002, pp. 263–294.

[15] M. F. Cusumano-Towner, F. A. Saad, A. Lew, and V. K. Mansinghka. "Gen: A General-Purpose Probabilistic Programming System with Programmable Inference". In: *CSAIL Technical Reports*. 2018.

[16] A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

[17] C. Date et al. *Database in depth: relational theory for practitioners*. " O'Reilly Media, Inc.", 2005.

[18] J. Davis and M. Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[19] C. De Sa, I. F. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas. "A Formal Framework for Probabilistic Unclean Databases". In: *arXiv preprint arXiv:1801.06750* (2018).

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[21] D. E. Denning. "An intrusion-detection model". In: *IEEE Transactions on software engineering* 2 (1987), pp. 222–232.

[22] M. Dylla and M. Theobald. "Learning tuple probabilities in probabilistic databases". In: (2014).

[23] E. Eskin. "Anomaly detection over noisy data using learned probability distributions". In: (2000).

[24] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. "A geometric framework for unsupervised anomaly detection". In: *Applications of data mining in computer security*. Springer, 2002, pp. 77–101.

[25] W. Fan. "Dependencies revisited for improving data quality". In: *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2008, pp. 159–170.

[26] A. K. Ghosh and A. Schwartzbard. "A Study in Using Neural Networks for Anomaly and Misuse Detection." In: *USENIX security symposium*. Vol. 99. 1999, p. 12.

[27] Y. Goldberg and O. Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *arXiv preprint arXiv:1402.3722* (2014).

[28] A. Graves and J. Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural networks* 18.5-6 (2005), pp. 602–610.

[29] E. Gribkoff, G. Van den Broeck, and D. Suciu. "The most probable database problem". In: *Proceedings of the First international workshop on Big Uncertain Data (BUDA)*. 2014, pp. 1–7.

[30] V. Hautamaki, I. Karkkainen, and P. Franti. "Outlier detection using k-nearest neighbour graph". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. IEEE. 2004, pp. 430–433.

[31] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[32] S. A. Hofmeyr, S. Forrest, and A. Somayaji. "Intrusion detection using sequences of system calls". In: *Journal of computer security* 6.3 (1998), pp. 151–180.

[33] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*. Vol. 398. John Wiley & Sons, 2013.

[34] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[35] J. Laurikkala, M. Juhola, E. Kentala, N. Lavrac, S. Miksch, and B. Kavsek. "Informal identification of outliers in medical data". In: *Fifth international workshop on intelligent data analysis in medicine and pharmacology*. Vol. 1. 2000, pp. 20–24.

[36] W. Lee and S. Stolfo. "Data mining approaches for intrusion detection". In: (1998).

[37] O. Levy, Y. Goldberg, and I. Dagan. "Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225.

[38] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[39] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

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

[41] S. Ramaswamy, R. Rastogi, and K. Shim. "Efficient algorithms for mining outliers from large data sets". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 427–438.

[42] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. "Holoclean: Holistic data repairs with probabilistic inference". In: *Proceedings of the VLDB Endowment* 10.11 (2017), pp. 1190–1201.

[43] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims. "Evaluation methods for unsupervised word embeddings". In: *Proceedings of the 2015 conference on empirical methods in natural language processing*. 2015, pp. 298–307.

[44] P. Sen, A. Deshpande, and L. Getoor. "PrDB: managing and exploiting rich correlations in probabilistic databases". In: *The VLDB Journal—The International Journal on Very Large Data Bases* 18.5 (2009), pp. 1065–1090.

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[46] D. Suciu. *Probabilistic databases*. Springer, 2009.

[47] P. Sun and S. Chawla. "On local spatial outliers". In: *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE. 2004, pp. 209–216.

[48] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung. "Enhancing effectiveness of outlier detections for low density patterns". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2002, pp. 535–548.

[49] *The world's most valuable resource is no longer oil, but data*. `https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data`. Accessed: 2020-02-14.

[50] M. Y. Vardi. *Fundamentals of dependency theory*. IBM Thomas J. Watson Research Division, 1985.

[51] J. Wang and N. Tang. "Towards dependable data repairing with fixing rules". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 457–468.

[52] C. Warrender, S. Forrest, and B. Pearlmutter. "Detecting intrusions using system calls: Alternative data models". In: *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344)*. IEEE. 1999, pp. 133–145.

[53] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. "Guided data repair". In: *arXiv preprint arXiv:1103.3103* (2011).

[54] K. Yamanishi and J.-i. Takeuchi. "Discovering outlier filtering rules from unlabeled data: combining a supervised learner with an unsupervised learner". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 389–394.

[55] J. Zhang and H. Wang. "Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance". In: *Knowledge and information systems* 10.3 (2006), pp. 333–355.