Deep Learning Models for Tabular Data Curation

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Author:

Riccardo CAPPUZZO

Scheduled for defense on the 1st APRIL, 2022, before a committee composed of:

Reviewers		
Prof.	**Melanie HERSCHEL**	University of Stuttgart, Germany
Prof.	**Yannis VELEGRAKIS**	Utrecht University, Netherlands
Examiners		
Prof.	**Raphaël TRONCY**	EURECOM, France
Prof.	**Paolo MERIALDO**	Università Roma Tre, Italy
Under the supervision of		
Prof.	**Paolo PAPOTTI**	EURECOM, France

Modèles d'apprentissage profond pour le nettoyage des données tabulaires

Thèse

soumise à

Sorbonne Université

pour l'obtention du Grade de Docteur

présentée par:

Riccardo CAPPUZZO

Soutenance de thèse prévue le 1er AVRIL 2022 devant le jury composé de:

Rapporteurs		
Prof.	**Melanie HERSCHEL**	Université de Stuttgart, Allemagne
Prof.	**Yannis VELEGRAKIS**	Université de Utrecht, Pays-Bas
Examinateurs		
Prof.	**Raphaël TRONCY**	EURECOM, France
Prof.	**Paolo MERIALDO**	Università Roma Tre, Italie
Sous la supervision de		
Prof.	**Paolo PAPOTTI**	EURECOM, France

To my parents and my sister, who will always have my back.

Abstract

Data curation, defined as the problem of organizing and maintaining data so that they can be employed for data-centric tasks, is a pervasive and far-reaching subject, touching all fields from academia to industry. Data curation has a number of sides to it, and no good solution for all of them. Current solutions rely on manual work from domain users, but this does not scale with the number of datasets to clean. In the last few years, many challenging problems in the NLP and computer vision fields have been solved by deploying Deep Learning (DL) models. In this work, we explore the potential of applying deep learning to data curation. However, three main challenges make the problem hard.

First, supervised systems work well, but they require labeled data that is not always available for any given dataset, and can put a large burden on the domain expert. Second, categorical values are pervasive in real data, however they cannot be modeled directly by many DL-based models, instead requiring to be encoded in some numerical form; this can lead to problems such as the "curse of dimensionality", which reduce if not nullify the gains that a DL model could provide. Third, DL-based models excel at representing homogeneous data (e.g., images, text, speech), while tabular data are not homogeneous and feature structural information that is not present in other types of data.

To tackle these challenges, we focus our work on methodologies that originate from different fields to design unsupervised solutions for the data integration and imputation problems. At the core of the solution, we develop models to organically produce distributed representations (embeddings) of mixed data types by transforming relational tables into graphs. We finally implement methods that apply deep learning techniques to tabular data to obtain embeddings that are fed to the target applications.

Our first focus is data integration and specifically the tasks of Schema Matching and Entity Resolution. To this end, we implement EmbDI, a system that generates embeddings for tabular data. We first explore applications of these embeddings in a one-table scenario, then we highlight how the geometric properties of the embeddings can be employed to achieve state-of-the-art results in entity resolution and schema matching.

We then move to the data imputation problem by using Graph Neural Networks in a multi-task learning framework named GRIMP. We use tabular embeddings as inputs for a classification objective to select the correct values for filling vacancies. We also explore methods for introducing external information in imputation systems. We leverage the concept of attention to pilot the training of GRIMP and another ML imputation algorithm on attributes in relationship with each other, and show that algorithms aware of functional dependencies improve their quality results.

Abrégé

La conservation des données, définie comme le problème de l'organisation et de la maintenance des données afin qu'elles puissent être utilisées pour des tâches centrées sur les données, est un sujet omniprésent et de grande envergure, qui touche tous les domaines, du monde universitaire à l'industrie. Le problème de la conservation des données présente de nombreux aspects et il n'existe pas de bonne solution pour chacun d'entre eux. Les solutions actuelles reposent sur le travail manuel des utilisateurs du domaine, mais elles ne sont pas adaptées au nombre d'ensembles de données à nettoyer. Au cours des dernières années, de nombreux problèmes difficiles dans des domaines tels que le langage naturel et la vision par ordinateur ont été résolus en déployant des modèles d'apprentissage profond. Dans ce travail, nous explorons le potentiel de l'application de l'apprentissage profond aux tâches de curation de données. Cependant, trois défis principaux rendent le problème difficile.

Tout d'abord, les systèmes supervisés fonctionnent bien, mais ils nécessitent des données étiquetées qui ne sont pas toujours disponibles pour un ensemble de données donné, et peuvent imposer une charge importante à l'expert du domaine. Deuxièmement, les valeurs catégoriques sont omniprésentes dans les données réelles, mais elles ne peuvent pas être modélisées directement par de nombreux modèles basés sur le langage DL, et doivent être codées sous une forme numérique ; cela peut conduire à des problèmes tels que le "fléau de la dimensionnalité", qui réduisent, voire annulent, les gains qu'un modèle DL pourrait apporter. Troisièmement, les modèles DL excellent dans la représentation de données homogènes (par exemple, des images, du texte, de la parole), alors que les données tabulaires ne sont pas homogènes et comportent des informations structurelles qui ne sont pas présentes dans d'autres types de données.

Pour relever ces défis, nous concentrons notre travail sur le développement de systèmes de conservation de données non supervisés qui peuvent fonctionner sans intervention humaine supplémentaire, la conception de systèmes de conservation qui modélisent intrinsèquement les valeurs catégorielles dans leur forme brute, et la mise en œuvre de méthodes et de techniques qui appliquent des modèles d'apprentissage profond aux données tabulaires.

Nous tirons parti de méthodologies issues de différents domaines pour concevoir des solutions non supervisées capables de résoudre les problèmes d'intégration et d'imputation de données sur des données tabulaires de type mixte. Au cœur de la solution, nous développons des modèles pour produire organiquement des représentations distribuées (embeddings) de données discrètes en transformant des tableaux relationnels en graphes, puis en alimentant les graphes aux algorithmes d'embeddings de graphes.

Nous nous concentrons d'abord sur l'intégration des données, et plus particulièrement sur les tâches de résolution d'entités et de correspondance de schémas. À cette fin, nous implémentons EmbDI, un système qui génère des enchâssements pour les données tabulaires ; nous explorons les applications de ces enchâssements dans un scénario simple, à une table, puis nous soulignons comment les propriétés géométriques des enchâssements peuvent être utilisées pour obtenir des résultats de pointe dans la résolution d'entités et la correspondance de schémas.

Nous passons ensuite au problème de l'imputation des données en utilisant des réseaux de neurones graphiques dans un cadre d'apprentissage multi-tâches appelé GRIMP. Nous utilisons des encastrements de tableaux comme entrées pour un objectif de classification visant à sélectionner les valeurs correctes pour remplir les postes vacants. Nous explorons également des méthodes pour introduire des informations externes dans les systèmes d'imputation. Nous exploitons le concept d'attention pour piloter l'entraînement de GRIMP et d'un autre algorithme d'imputation ML sur des attributs en relation les uns avec les autres et montrons que les algorithmes conscients des dépendances fonctionnelles améliorent la qualité de leurs résultats.

Acknowledgements

I would like to thank my Supervisor, Professor Paolo Papotti, whose continuous support, patience, and advice have been paramount for completing this PhD.

I would also like to thank the entire EURECOM Data Science department, and in particular Dr. Dimitris Milios, Dr. Simone Rossi, Dr. Giulio Franzese and Mr. Mohammed Saeed for their unfailing assistance in both technical and personal matters.

I would like to thank Dr. Saravanan Thirumuruganathan for his invaluable contributions towards the research that has ultimately led to this manuscript.

I would like to thank Prof. Melanie Herschel and Prof. Yannis Velegrakis for their insightful reviews on this manuscript, as well as Prof. Paolo Merialdo and Prof. Raphaël Troncy for their feedback during the course of the PhD.

Finally, I would like to thank my parents and my sister for their understanding and encouragement during the high and lows of the past few years.

Contents

List of Figures

List of Tables

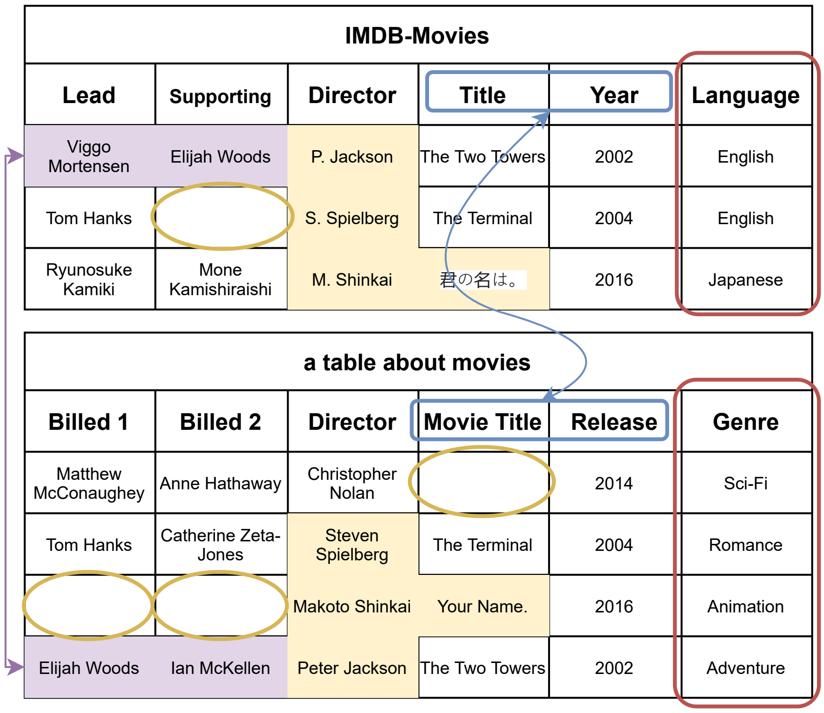
Chapter 1

Introduction

In the beginning, there were data. Huge, vast reservoirs of data to be employed by users, organizations, researchers. Unfortunately, most of these data are tainted by bad formatting, duplicates, missing and erroneous values [2,3]. We show an example of this in Figure 1.1, which reports a pair of (dirty) tables that contain data from the same domain, as well as multiple imperfections. In this Figure, the data imputation problem is shown as the missing values, which should be filled. For data integration, the schema matching problem consists in matching attributes with inconsistent labels (e.g. "Title" and "Movie Title", or "Lead" and "Billed 1"), while for entity resolution the objective is linking entities (in this case, tuples) to find duplicates and possibly combine information coming from the different sources (e.g. by combining both the original and English version of the title of the movie "Your Name.").

Curating data is a tedious, time consuming process. Unfortunately, this process cannot be overlooked if one is keen on avoiding negative or misleading outcomes in downstream applications [4, 5]. Data curation is a remarkably practical field of research: indeed, handling dirty and inconsistent data is a problem that must be dealt with on a daily basis in the industry. According to a widely cited statistic, data scientists spend 60 to 80% of their time cleaning and curating data [6–8]. This makes data curation a far-reaching and widely studied problem [9], which is made harder by the large burden it poses on humans [10], both in the data curation step and in downstream applications.

Clean and open access to data has a number of ramifications depending on the use that is to be made of the data. For some applications, dirty data cannot be used at all due to the fact that erroneous or missing values would introduce unacceptable biases; in the industry, this might lead to wrong decisions that lead to loss of capital down the line [11]; in medical fields, dirty data could drive incorrect therapies [4,12]. In other situations, data might be "dirty" in the sense that multiple data sources contain similar information which should be combined in a "cleaned" view by consolidating duplicates: this can be the case when different companies merge and need to combine their databases. Performing proper data curation is a pervasive subject that touches all kinds of data-centric disciplines. For these reasons, advancements in the field of data curation can directly reach a large set of applications [13], whilst remaining of paramount importance in the database domain where most of its research originates.



IMDB-Movies

Lead	Supporting	Director	Title	Year	Language
Viggo Mortensen	Elijah Woods	P. Jackson	The Two Towers	2002	English
Tom Hanks		S. Spielberg	The Terminal	2004	English
Ryunosuke Kamiki	Mone Kamishiraishi	M. Shinkai	君の名は。	2016	Japanese

a table about movies

Billed 1	Billed 2	Director	Movie Title	Release	Genre
Matthew McConaughey	Anne Hathaway	Christopher Nolan		2014	Sci-Fi
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	Romance
		Makoto Shinkai	Your Name.	2016	Animation
Elijah Woods	Ian McKellen	Peter Jackson	The Two Towers	2002	Adventure

Figure 1.1: An example of dirty relational data.
In this example, some tuples are duplicated; there are inconsistencies between values in duplicated tuples (purple background); some values have different format but represent the same entity (yellow background); some values are missing (yellow ovals); some attributes do not have the same name (blue boxes); some attributes are present only in one table and not the other (red boxes).

There is no easy solution for the problem. Constraints (domain constraints, functional dependencies, conditional functional dependencies, denial constraints etc.) are employed to enforce properties in the data and for identifying errors by observing what values violate them. While some of these constraints can be introduced during the design phase of the database (e.g. by specifying an attribute's datatype and domain, then enforcing the constraints), this is not always possible.

However, it is possible to detect and generate these rules by observing the data. Constraints of different kinds can be employed to enforce some properties in the data, to extract information or to match entities [14]. These constraints are usually hand-crafted, or must be discovered by observing the data. Their discovery has been the focus of a number of works going from rule discovery [15], to the discovery of FDs [16, 17], Conditional FDs [18–20], Denial Constraints (DCs) [21]. This collection of tools can allow to detect and correct some of the errors that can be found in a dirty dataset [4].

Unfortunately, these tools are not perfect: constraints are not always applicable to all situations, functional dependencies do not always exist, the generation of rules can become too expensive, automatically-generated rules may be too many to be checked by a human observer [14]. Moreover, these tools are often heavily reliant on human expertise [3, 22–26]: this is a large issue, as solutions that rely on human interaction are time-consuming, costly, and not scalable to large quantities of data [27].

The presence of dirty or missing data is a major issue, as the generation of constraints over dirty data can lead to incorrect rules [28–30]. Each dirty dataset has its own idiosyncrasies and quirks, which means that ad-hoc solutions are very often the only possible solutions. Even when data are not missing, values might not be clean due to typos (e.g. "balck" instead of "black"), issues with encoding (does "10.002" mean "10 thousands and 2", or "10 and 2 thousandths"?), formatting ("The Who" in one dataset and "Who, The" in another).

The datatype of the values in a dataset can also be problematic: while numerical errors can be handled through naïve approaches such as linear regression, or modeled through more sophisticated Deep Learning-based systems [31–35], categorical data (i.e., most textual data) cannot be handled in such a way. This either means that categorical values are converted into numerical form, or categorical-only systems must be designed. The latter is often the only workable solution for the majority of cases in which the number of categories becomes too large to be handled by one-hot encoding or similar methods [36].

There are a number of sides to the data curation problem, and no good solution for all of them. In this work, we focus on three main challenges:

- **Supervised and unsupervised systems**: Systems that have a human-in-the-loop architecture do not scale well and, in some cases, it is hard to find a domain expert. Supervised systems may rely on golden records and labeled data that are not always available for a given dataset.

- **Embeddings for tabular data**: While text can be encoded by employing existing embeddings algorithms, this is not the case for tabular data.

- **Categorical data**: Categorical values can hang up simpler architectures that work on numerical values by causing an explosion in the number of features, which leads to scalability issues (the well-known "Curse of dimensionality").

Our work leverages methodologies originating from different fields to produce systems that elegantly handle data integration and data imputation for mixed, tabular data in an unsupervised fashion.

1.1 Challenges

Lack of human supervision. As mentioned, human-in-the-loop systems can be employed with success in the field of data curation. However, the human factor is both a blessing and a curse for these methods. On the one hand, human experts can select the best possible set of rules, constraints, corrections to be applied to the data under observation; on the other hand, there is frequently far too much data for a human to handle properly [14]. Another downside is the fact that experts in how to curate the data are not necessarily experts in how to put their recommendations in code, that is how to "inform the system" of what it should do [6–8]. For these reasons, we choose to work with unsupervised methods: all the systems we discuss in this thesis are trained exclusively on the data at hand and do not require labels, golden records, or human-defined rules to carry out their tasks. While we do focus on unsupervised solutions, the embeddings they produce can still be employed by external supervised methods and lead to improvements in the final result. We developed our systems in such a way that it should be possible to apply them without a large coding background, so that a domain expert would be able to run the system, then correct the result if necessary.

Embeddings for tabular data. "Vector space models" (VSMs) describe algebraic models for representing entities through *vectors* (or *embeddings*) in a high-dimensional vector space [37]. The position in space of each vectorized entity is decided in relation to all other entities, in such a way that correlated entities are placed closely to each other in the vector space. Crucially, geometric properties apply to the relationship between these vectors: numeric distances between strings can be measured, and it is possible to "navigate" around the vector space by carrying out vector operations [37]. By designing a system that takes the structural elements of tabular data (such as tuples and attributes) into consideration, it becomes possible to employ the geometric properties of these structural elements for further use.

Categorical data. For this work, we direct our attention mostly at datasets that can contain categorical, numerical, or mixed data types. Categorical data can assume only a limited number of different values that can be separated in "categories". While categorical data is normally textual, it need not be: at times, integer-valued attributes should be treated as categorical attributes (for example, when working with ZIP codes or numerical IDs). Due to their discrete nature, categorical values (i.e., values that can be assigned to a category, such as strings or numerical IDs) are particularly problematic in Machine Learning as most models require numerical features in order to function [36, 38]. The lack of good answers to this problem is what motivated us to focus specifically on datasets that feature a majority of categorical data. We design a set of algorithms and systems that are not only able to handle categorical values, but also "embrace" their discreteness through a discrete representation, that is a graph. To combine ML models and categorical data, we elect to use vector representations to encode the latter, so that it is possible to employ systems that require numerical features on categorical values.

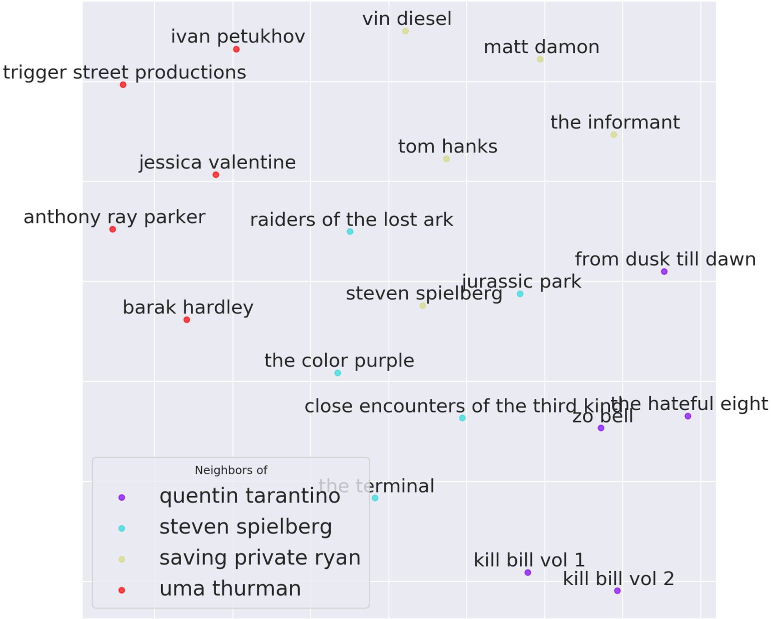


Figure 1.2: 2D T-SNE projection of embeddings of a relational table that contains data about movies.

In early applications, these VSMs have been used to generate vector representations of words and documents. The introduction of later frameworks such as word2vec, however, opened the door to their use for a wide variety of applications. Indeed, a fundamental advantage of embedding models is that any kind of object can be transformed into a numerical vector, provided that entities can be represented in a form suitable to the training model. Our work leans heavily on this characteristic by employing embedding techniques to encode categorical tabular data, then acting on the resulting numerical vectors to carry out further operations that would not be possible on the original data. Embedding vectors allow to mitigate this problem by converting categorical values into comparatively well-behaved high-feature vectors, this in turn allows to reduce the overall number of dimensions required by the model.

Figure 1.2 depict a 2D projection of embeddings generated on a subset of entities related to movies (title, directors and actors), and their closest neighbors. All the points in the space are placed based on their relationship with other points in such a way that related values are positioned closer to each other than unrelated values.

The second reason for us to employ VSM representations ("embeddings") is how they can be trained in an unsupervised fashion given a properly prepared training corpus.

This allows us to develop pre-processing tools that, given a target dataset, produce a suitable input representation (be it a sentence-based training corpus, or a feature-enriched vector representation of table values) for the embedding algorithm to "digest". With this approach, the algorithms can be executed with little required input from human experts other than proper setup and pre-processing.

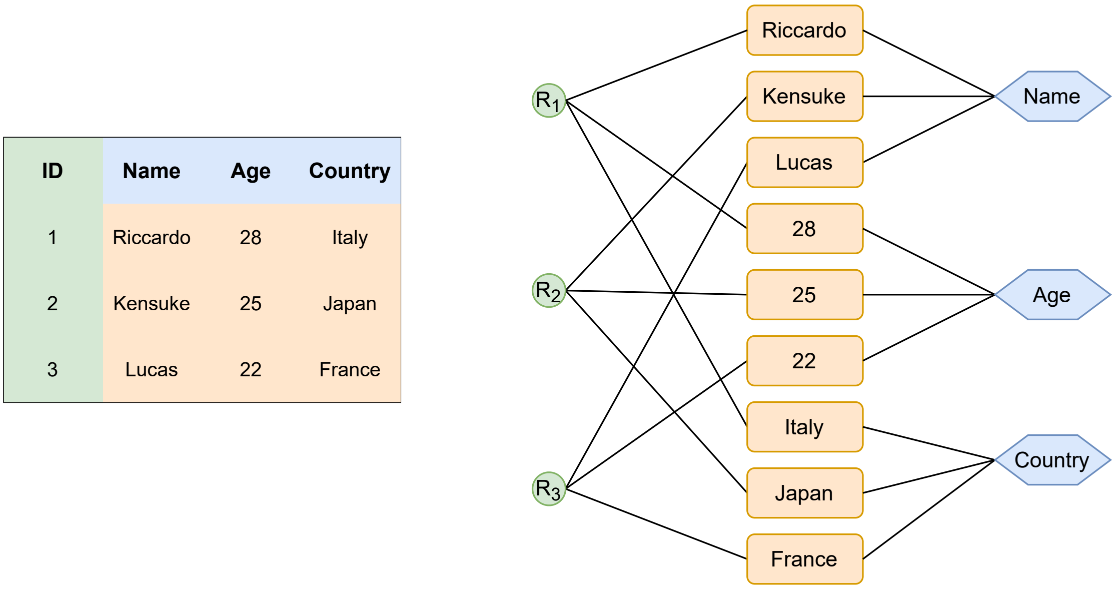
Encoding tabular data through embedding is not a trivial endeavor [39, 40]: while natural language is inherently redundant and homogeneous, tabular data do not share the same homogeneity and are far less redundant, especially when considering only one table at a time. Tabular data also contain syntactic structures that are absent from natural language, namely tuples, attributes and a concept of "belonging" shared by the tokens found in those structures. Furthermore, while it is possible to mitigate some of the issues caused by categorical data, models based on Deep Learning techniques suffer heavily from their characteristic: domains with large cardinality and highly imbalanced data distributions are very difficult to model, which becomes a large problem whenever classification problems come into play.

1.2 Approach and Contributions

In this work, we propose a novel, **modular** procedure for handling **Data Curation** (specifically **Data Integration** and **Data Imputation**) that relies on non-traditional representations of tabular data and Deep Learning-based models. We provide examples of how to represent relational tables **in graph form**, and how to generate vector representations for the graph nodes. These graph representations are built in such a way that the graph nodes include table tuples and attributes, thus allowing to organically generate vector representations for these entities as well: this is a marked change from previous work, in which column and row representations were generated by combining the vectors of the values in the respective column or row.

A major advantage of this modular methodology is that it becomes possible to "plug and play" different methods: this allows to piggyback on the research carried out in the different fields to leverage different state of the art methods and therefore enhance the performance of the system. We provide examples of this by employing different graph representations of tables, as well as multiple methods for generating graph node embeddings (namely, based on DeepWalk [41, 42] and on Graph Neural Networks [43]). For example, by treating random walks as sentences it becomes possible to leverage pre-existing algorithms originally developed for Natural Language Processing (NLP) problems for tasks completely different from what they were designed for. Vector representations enjoy geometric properties that can be exploited to carry out a number of operations: here, we use them to perform entity resolution, schema matching and data imputation. Finally, a key feature of the systems we propose is that they are **unsupervised**, so that there is no hard requirement on external human involvement to run the training.

It is important to note that Deep Learning is not a silver bullet for all applications, and suffers from some drawbacks that must be kept in consideration when applying these solutions, or any DL solution [39, 44]. We explore some of these limitations in the latter



ID	Name	Age	Country
1	Riccardo	28	Italy
2	Kensuke	25	Japan
3	Lucas	22	France

Figure 1.3: This figure shows the graph built by EmbDI on the right, with the small table that acts as source on the left.

part of this manuscript.

This work focuses on attacking the data curation problem from two angles: **unsupervised learning** and **modeling of categorical data**. We employ the former to get around the human-in-the-loop issue: by designing unsupervised models, it becomes possible to carry out multiple data curation tasks with no need for human-provided data other than the starting datasets and the location of missing values. We design our models around the need to model categorical data in order to expand the use of Deep Learning in data curation: indeed, DL-based models are penalized by categorical data exactly because these data are harder to model numerically. In our contributions, we introduce a novel representation of relational data that allows to elegantly generate numerical representations of categorical data: through the transformation of a relational table into a graph such as the one displayed in Figure 1.3, it becomes possible to leverage embedding techniques originally developed for Natural Language Processing to generate vector representations of the nodes in the graph. We design our systems in a modular fashion, so that it is possible to apply different strategies to represent the tables as graphs, to generate the embeddings and to work with the resulting embeddings for different tasks.

Overall, we propose the following contributions:

- We introduce a novel architecture for restructuring tabular data which converts relational tables into a graph. This allows to employ embedding generation techniques to produce distributed, numerical vector representations of the content of the table. As a consequence, the discrete representation can directly be used for representing discrete, categorical datatypes. We show how to generate the embeddings on top of the graph and how to study their quality. *This contribution tackles the challenge* *of generating tabular embeddings for categorical data.*

- We then move to a two-table scenario in which data integration is performed by building a graph on top of the concatenation of the two tables. By doing this, the graph embeddings encode information about the structure of both tables. The geometric properties of these embeddings can be exploited once again, this time to perform **Schema Matching** (SM) and **Entity Resolution** (ER). We develop EmbDI (**EMB**eddings for **D**ata **I**ntegration) to implement these contributions. *With* EmbDI*, we propose an unsupervised solution to the problem of data integration for categorical data.*

- We then pivot to the data imputation task by developing GRIMP (**G**raph embeddings for **R**elational data **IMP**utation), a data imputation system that combines Graph Neural Networks with a Multi-Task classification objective function to perform Data Imputation over relational data. We study how to integrate Functional Dependencies in GRIMP, as well as already established systems with FunForest (**Fun**ctional Miss**Forest**), a data imputation system based on the well-known MissForest algorithm [45]. FunForest can make use of user-provided Functional Dependencies during its training. *For the data imputation problem, we once again propose unsupervised solutions for categorical data.*

For every contribution, we carry out extensive experimental campaigns to study the system main properties and compare each of them against a number of different baseline algorithms.

1.3 Thesis Outline

The remainder of the thesis is organized as follows:

Chapter 2. This chapter introduces the basic terminology and discusses previous work on the different subjects treated in the rest of the thesis.

Chapter 3. This chapter presents a method for generating embeddings for relational data. The different sections of the EmbDI system are described here in detail. We present a suite of tests that let us get a measure of the information that has been encoded in the embeddings by relying on their geometric properties, akin to the Analogy test proposed for word2vec embeddings.

Chapter 4. This chapter explores how EmbDI embeddings can be employed in a two-tables scenario to perform the data integration tasks of Schema Matching and Entity Resolution. We explore how the tables are pre-processed to improve overlapping and how to perform the matching to maximize match precision. Finally, we describe how vector neighbor indexing can be used to speed up the matching procedure. We also introduce some promising applications of the table embeddings, such as geometric querying of embeddings and token matching.

Chapter 5. This chapter focuses on data imputation, our contributions and an in-depth discussion of our results in comparison to a number of data imputation baselines.

We introduce GRIMP and its major components. We show the effect of Functional Dependencies on DL-based systems and develop FunForest, an improved version of the MissForest algorithm that makes use of FDs to achieve better imputation results.

Chapter 6. In this chapter we wrap up the results achieved in this thesis. We highlight some of the drawbacks of the proposed solutions and discuss several promising starting points for future developments in the field.

1.4 Publications

Chapter 3 is partially based on the demo:

> Riccardo Cappuzzo, Paolo Papotti, Saravanan Thirumuruganathan: **EmbDI: Embeddings Generation for Integrating Relational Datasets** (Demo paper in submission, 4 pages).

Chapters 3 and 4 are based on the following papers:

> Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. **Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks**. (14 pages) In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). The paper has received the ACM badges for **Results Reproduced**, **Artifacts Available** and **Artifacts Evalued and Reusable**

> Riccardo Cappuzzo, Paolo Papotti, Saravanan Thirumuruganathan: **EmbDI: Generating Embeddings for Relational Data Integration** (Discussion Paper, 8 pages). SEBD 2021

> Riccardo Cappuzzo, Paolo Papotti, Saravanan Thirumuruganathan: **Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks** (Short Paper, 2 pages). BDA 2021

The content of Chapter 5 is based on the paper:

> Riccardo Cappuzzo, Paolo Papotti, Saravanan Thirumuruganathan: **Combining Graph Neural Networks and Multi-Task Learning for Tabular Data Imputation**. Paper in preparation.

Chapter 2

Background

In this chapter, we describe the background of the subjects that are touched by this thesis. We introduce a set of terms and concepts that are referred to over the course of this work, as well as related work in the field. This is not an exhaustive discussion of all the facets of the subjects in data curation and deep learning: the objective of this section is instead to "standardize" the concepts that are the focus of this thesis.

With the final objective of innovating the field of data curation with deep learning methods, this thesis falls at the intersection of different domains: **Relational Tables**, **Graphs**, **Vector Space Models**, **Multi-task Learning**, and **Data Curation**.

2.1 Relational tables

While relational tables are an ubiquitous concept, this section will nonetheless discuss some basic concepts to summarize their use in this thesis.

Definition 2.1.1 (Relational Table)**.** With *relational table* (in the following table or dataset) we refer to a collection of data organized in table form which includes *rows* (or *tuples*) and *columns* (or *attributes*), with *cells* being the intersection of a row with a column. Each column is defined over a *domain*, that is a set of values that share a *datatype*.

Relational tables will be used over the course of this manuscript.

2.1.1 Datatypes

Broadly speaking, datatypes can be divided in *numerical datatypes*, and *categorical datatypes*; additional datatypes (ordinal values, dates etc.) exist, however we do not consider them in this work. Numerical data include integers and real-valued numbers; they can be handled using traditional distance metrics. It must be noted that, in some cases, numerical values could instead be considered as categorical: for example, ZIP codes would be categorical as the numerical distance between two ZIP codes does not necessarily reflect the difference between the two numbers. Categorical datatypes are datatypes whose values must be split in categories. It is not possible to use a numerical distance to measure the difference between categories, although string metrics that can measure the difference between strings exist, such as the Jaccard similarity, or the Levenshtein distance. Unfortunately, these string metrics do not provide any information relative to the meaning of the strings. To this end, ontologies and lexical databases such as WordNet [46] can be used. Indeed, part of the motivation behind embedding models is finding distances between non-numeric entities that do not rely on string similarity [47].

Depending on the application, all datatypes (categorical, integers and reals) can cause problems for different reasons: NN-based models cannot handle categorical values as-is, and require them to be represented through some kind of encoding such as the one-hot encoding [36]; for some integer values it might be unclear whether they should be considered as numbers or as categories; real numbers by their own nature cannot be represented properly in a discrete form. Integers can be considered as ordinal types to better represent the order relationship between different values [48], but the order relation is not necessarily available.

2.1.2 Functional Dependencies

Definition 2.1.2 (Functional Dependencies (FDs))**.** *Functional Dependencies* [49] are integrity constraints modeled as relationships between multiple columns, such that a set of columns *functionally determines* (or, *implies*) the value in a different column. Given a relation R, attribute Y is functionally dependent on attribute or set of attributes X if for every valid instance of X, the values in X uniquely determine the value of Y. This relationship is normally represented as:

$$X \rightarrow Y \tag{2.1}$$

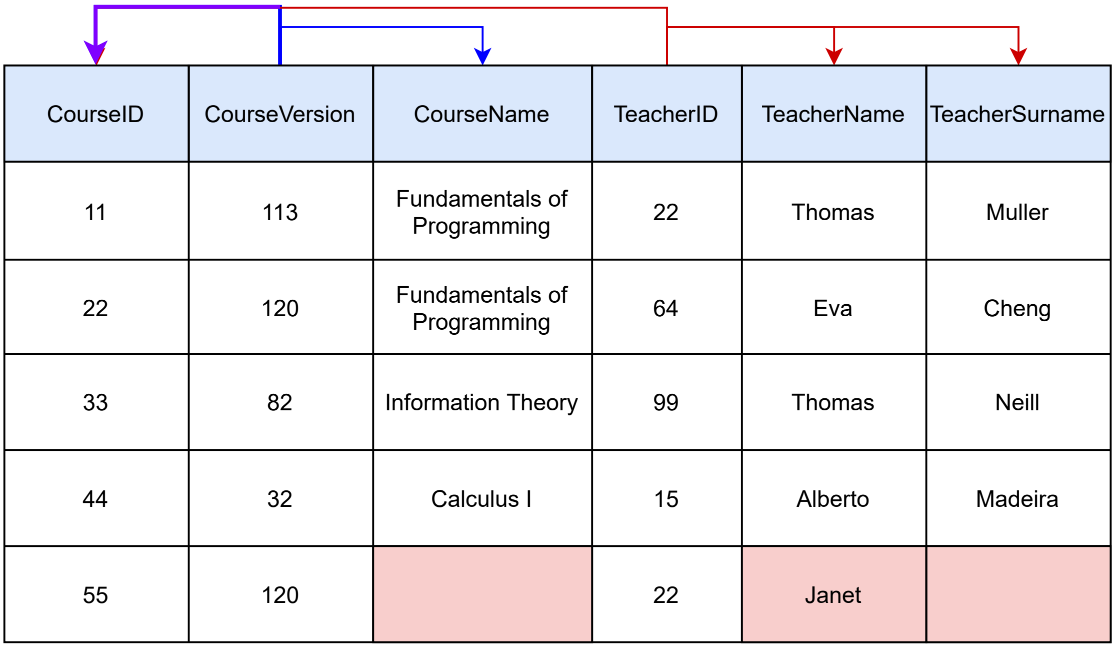
Definition 2.1.3 (Determinant, Dependent)**.** X is the *determinant*, or Left Hand Side (LHS), while Y is the *dependent* or Right Hand Side (RHS). The LHS might include multiple attributes that taken together imply the RHS, but that are not enough to force the relationship when they are taken singularly .

We use Figure 2.1 as a running example for this section. Figure 2.1 gives an example of a table that includes a number of Functional Dependencies:

- $TeacherID \rightarrow TeacherName$
- $TeacherID \rightarrow TeacherSurname$
- $CourseVersion \rightarrow CourseName$
- $TeacherID, CourseVersion \rightarrow CourseID$

$CourseID$ is by itself the LHS of all other attributes as it is the table's primary key. For this example, tuples are be denoted as t_i, where i is the value found in $CourseID$.

While traditionally used for schema design, Functional Dependencies can be used to detect and correct errors, as long as said errors appear only in the Right Hand Side (the Dependent): if the full LHS is available somewhere else in the table, then it is possible to detect and correct errors in the RHS; if part of the LHS is not available, then error detection or correction cannot be done through FDs alone.



CourseID	CourseVersion	CourseName	TeacherID	TeacherName	TeacherSurname
11	113	Fundamentals of Programming	22	Thomas	Muller
22	120	Fundamentals of Programming	64	Eva	Cheng
33	82	Information Theory	99	Thomas	Neill
44	32	Calculus I	15	Alberto	Madeira
55	120		22	Janet	

Figure 2.1: Example of a simple table that includes functional dependencies.

Going back to the example in Figure 2.1: here the known FDs are enough to repair the missing values in $CourseName$ and $TeacherSurname$, as well as to detect the erroneous value in $TeacherName$. Since $CourseVersion \rightarrow CourseName$, tuple t_{55} can be repaired by using the value found in t_{22} and attribute $CourseName$, "$Fundamentals\ of\ Programming$". Unfortunately, FDs are not always the solution. Consider now the value $t_{55}[TeacherName] = Janet$. This is violating the FD found in t_{22}. However, by relying exclusively on FDs, we do not have enough information to decide whether the correct value is $Thomas$, or $Janet$. If more records are available, it may be possible to solve this problem by majority voting: if there exists another tuple t_{66}, in which $TeacherID = 22$ and $TeacherName = Janet$, we could rule $Thomas$ as the erroneous value. This information is not always accessible.

In the example we showed how "strict" Functional Dependencies can be employed to spot and correct some errors, and how they are not infallible. Extensive work has been done in the literature both to extend the reach of FDs by "relaxing" their requirements [50], and to profile data with the objective of detecting them. Examples that belong to the first category include Conditional Functional Dependencies (CFDs) [51], Temporal Functional Dependencies [52]; [17, 53, 54] are some examples of the latter. Denial Constraints (DCs) [21] represent an evolution of Functional Dependencies, with far more power of expression and the capacity of handling more complex cases, such as range-based constraints.

Functional Dependencies are used in Chapter 5.

2.1.3 Deep Learning Representations of Tabular Data

Among the various types of data that can be handled by deep neural networks (e.g. image, video, audio, text data), tabular data stick out as the most problematic [40, 55]: tabular data are heterogeneous, with dense numerical features and sparse categorical features; correlation between features is weaker than semantic or spatial relationships in text or image data; variables can be correlated or independent; features have no positional information [56]. A number of surveys [39, 40, 44] have studied how the problem has been tackled in the literature. Overall, there are a number of challenges that temper the effectiveness of DL-based methods in their application to tabular data [40]:

1. Training data might feature missing values (like in the simple example in Fig. 2.1), outliers, inconsistent data. Moreover, training classes are often imbalanced.

2. There is no spatial correlation between features in tabular data [57], as the order of columns in a table does not necessarily reflect any kind of inherent correlation between them.

3. Categorical data require extensive pre-processing and must be encoded in a way that can be handled by DL models. [36] surveys different encoding solutions.

4. Deep neural network models tend to be very fragile and sensitive to small modifications in the training data [58, 59]. This is less problematic for homogeneous data, but tabular data do not belong to this category. Small changes even in a binary value can lead to large changes in the output, exacerbating the issue described in point 1).

In fact, works such as [39] and [44] have shown that systems that rely exclusively on Deep Neural Networks do not necessarily outperform "simpler" models in all cases. However, DL-based solutions still work well when paired with shallower methods [44] such as GDBT (Gradient Boosted Decision Tree) [60].

Different DL-based methods are employed in Chapter 3 and 5.

2.2 Graphs

While most of this work is based on graphs, we do not delve deeply in the graph theory field. Still, to summarize the concepts that will come into play the most in the rest of this work we report the following definitions.

Definition 2.2.1 (Graph, Vertex, Edge)**.** A *Graph* is defined to be a pair $G = (V, E)$, where V is a set of n *vertices* v_i such that $v_i \in V$, and E is a set of m *edges* (or *links*), where each edge $e_{vu} = (v, u) \in E$ denotes a pair of vertices (v, u) that are put in relation with each other.

Definition 2.2.2 (Directed graph, undirected graph)**.** A graph $G = (V, E)$ is said to be directed if its set of edges E is made of ordered pairs of vertices in V: $e_{uv} = (u, v) \neq e_{vu} = (v, u)$. Conversely, a graph is undirected if its set of edges E is composed of unordered pairs: $e_{uv} = e_{vu} = \{u, v\}$.

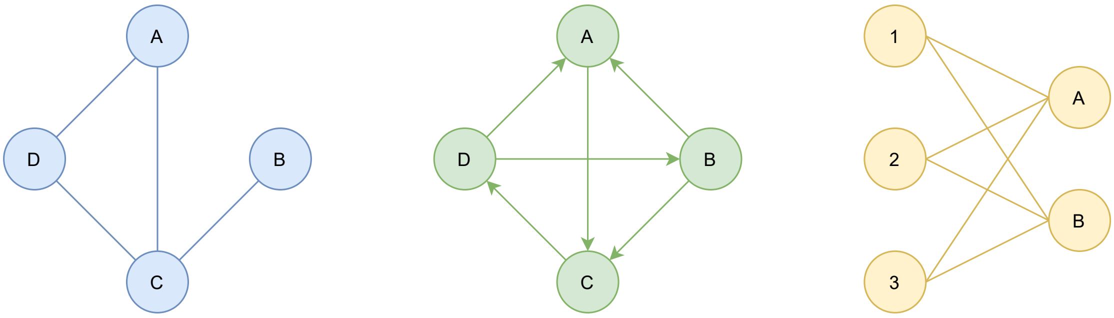
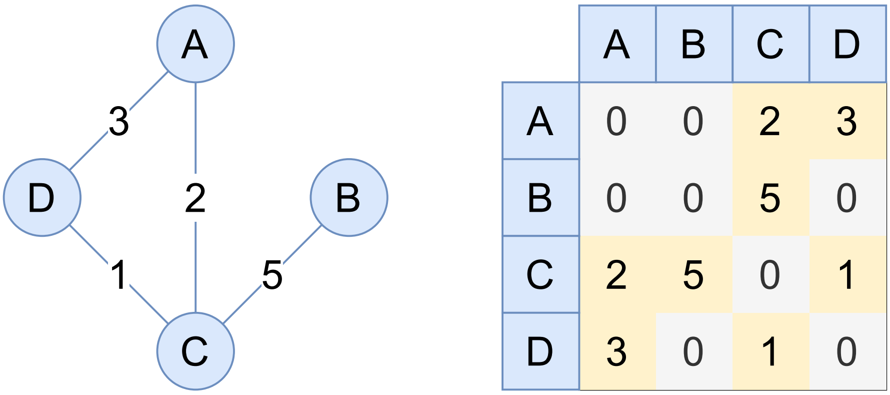


Figure 2.2: An example of three different types of simple graphs: an undirected graph, a directed graph and a bipartite undirected graph.

Definition 2.2.3 (Node neighborhood)**.** The neighborhood of a node v is defined as $\mathcal{N}(v) = \{u \in V|(v,u) \in E\}$, and it represents the set of nodes u_k that share an edge with node v.



	A	B	C	D
A	0	0	2	3
B	0	0	5	0
C	2	5	0	1
D	3	0	1	0

Figure 2.3: Example of adjacency matrix with weighted edges.

Definition 2.2.4 (Adjacency matrix)**.** The adjacency matrix $\mathbf{A}$ is a $n \times n$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ if $e_{ij} \notin E$.

Definition 2.2.5 (Node features)**.** A matrix $\mathbf{X} \in \mathcal{R}^{n \times d}$ is a node feature matrix with $\mathbf{x}_v \in \mathcal{R}^d$ being the feature vector of a node v.

Definition 2.2.6 (Edge features)**.** A matrix $\mathbf{X}^e$ where $\mathbf{X}^e \in \mathcal{R}^{m \times c}$ is an edge feature matrix, with $\mathbf{x}^e_{u,v} \in \mathcal{R}^c$ being the feature vector of edge $e_{uv} = (u, v)$. Edge features might be mono-dimensional, so that the edges might be weighted. Weights might be a measure of the edge (for example, the distance between two cities).

Figure 2.3 shows an example of adjacency matrix with weighted edges.

Definition 2.2.7 (Bipartite graph)**.** A *Bipartite Graph* is a graph in which vertices can be partitioned in two sets A and B, and such that no edges exist between any two vertices within each of A and B (right side of Fig. 2.2).

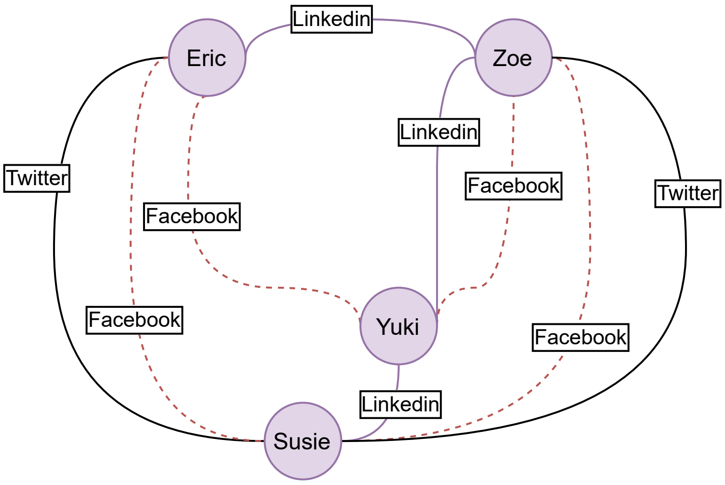


Figure 2.4: Example of multigraph with typed edges.
Here, each node is a different person, while each edge is typed and denotes a link on a specific social network. In some cases, a pair of people is connected by edges with different types.

Definition 2.2.8 (Multigraph)**.** A *Multigraph* (Fig. 2.4) is a generalized graph that allows to have more than one edge connecting the same two vertices; edges in multigraphs can be typed, so that two vertices might be connected by multiple edges with different types.

Different types of graphs are employed in Chapters 3 and 5.

2.2.1 Knowledge Graphs

Knowledge Graphs (e.g. YAGO [61] and WikiDATA [62]) are data structures built with a graph-like architecture to store real world entities and their relationship with other entities. Knowledge Graphs are modeled using multigraphs whose nodes and edges have properties that encode information relative to those entities. Entities in a Knowledge Graph can be any abstract or concrete object of fiction or reality [63], such as *Paul Rudd*, *Paris*, *WHO*. Entities can have properties (e.g. birth date, location, height etc.), and can have relationships with other entities. Relationships connect different entities to describe links between them.

A common standard for representing data in knowledge graphs is the Resource Description Framework (RDF) [64]. RDF statements compose a directed graph, with each statement being a triple that includes a node for the subject, a node for the object and a directed edge that connects the two and acts as a predicate. For example, a possible triple could be $< PaulRudd,\ ActedIn,\ Ant - Man >$.

For this work, we do not employ Knowledge Graphs directly, however we do represent tabular values by relying on triplets reminiscent of RDF statements in Chapter 5.

2.3 Vector Space Models

Vector Space Models (VSMs) describe algebraic models for encoding entities in the form of high-dimensional, real-valued vectors in such a way that related entities are positioned close to each other in the vector space. In the first VSM applications, each entity (word, sentence or document) was assigned a vector with a size equal to the number of terms in the domain, so that each vector acts as an indicator that specifies whether an entity in the domain is present in the document at hand. This is naturally problematic when the size of the domain is very large: vectors can become very large and sparse, which makes carrying out operations particularly difficult. “Bag-of-word” models are one example of Vector Space Models.

A number of advances have led to more compact VSM representations, which removes the need for an indicator for every value in the domain. This makes it possible to compress the information in a smaller number of dimensions, while at the same time maintaining the algebraic properties of the “extended” VSM representation. The CBOW (Continuous Bag of Words) and Skip-gram [37, 65] models were instrumental for bringing scalability in the generation of the VSM, which ultimately allowed to train the models on huge amounts of data for their application in the representation of words for Language Models. Indeed, VSMs came to prominence thanks to their applications in the Natural Language Processing field. We demonstrate that they are not reserved to those applications, nor are the advancements made in the NLP field applicable exclusively to Language Models.

In this work, we focus on ***Word Embedding*** *and* ***Graph Embedding*** *(already described in Section 2.3.2) to perform* ***Table Embedding****. Table embeddings are methods are used in Chapters 3 through 5.*

2.3.1 Word Embedding

As hinted above, the use of VSMs to encode information about words is what has led to the explosion in development of this field. Early word embedding algorithms include Word2Vec [37, 65], GloVe [66] and fastText [67]: these algorithms rely on “bag-of-words” training corpora and rely on the co-occurrence of values in their context to generate word embeddings.

Word2Vec [37] is a neural network-based method that trains vector spaces based on the context each word is found in. The context is a n-sized (an n-gram) window of words centered on the target word, which slides over training sentences in the training corpus. It is a relatively simple algorithm: by only considering the co-occurrence of words within the n-gram, the sequence of words in the window is not considered, and the content (or the meaning) of each “word” is unknown to the algorithm. For pure language modeling, this is a major drawback since word order and sub-word tokens carry a lot of information that the algorithm cannot make use of; on the other hand, this is beneficial if one is trying to adopt Word2Vec to generate embeddings for self-contained entities, since splitting them in sub-entities would not provide additional benefit. This is the case, for example, when the training corpus contains random walks over a graph, like in [42] or EmbDI itself. Word2Vec is trained by using either the CBOW or the Skipgram model:

in the first case, the model is traned by predicting a word from its context, while in the latter the model predicts the context from a word. Word2Vec makes use of a number of optimizations to reduce the training cost involved in the handling of the training corpora that are required for training models for NLP. The second word embedding algorithm we have been working is fastText [67]. fastText is trained in a similar fashion to Word2Vec, with the main difference being how the former splits each word in a training sentence into subwords. This is advantageous when representing natural language, since subwords carry additional information that is lost when only the full word is considered. A useful feature provided by the fastText library is the generation of a vector for unknown words or sentences. For sentences, this is done by dividing each sub-word vector in the sentence by its norm, then averaging all vectors together. Modeling sub-words on top of words is a major advantage of fastText as it allows to handle unknown or misspelled terms better than Word2Vec.

In order to properly train a Language Model, a very large (in the order of billions of tokens) training corpus is necessary; as a result, the training procedure is very computationally expensive. Pre-trained models are available in a number of different flavors: different algorithms, different languages, different vocabulary sizes, different sources. These pre-trained models are made available in various repositories [68, 69] for use in a number of applications, and there are works that employ these embeddings to represent tabular data [70, 71]. A disadvantage inherent in the use of pre-trained models is the fact that vectors may include biases that are not reflected in the table at hand.

The use of pre-trained embeddings is explored in Chapters 3 and 4.

Tokenization

With tokenization we refer to the procedure of parsing a string, then splitting it in smaller units (*tokens*), which have a specific representation and meaning. The "tokenization" procedure has recently come in the spotlight thanks to NLP. Specifically, tokenization is used in the preparation of a Language Model's training corpus to identify what words should be contained by the vocabulary. One of the most widely adoperated tokenizers is the Wordpiece tokenizer model [72], which is employed, among others, by BERT [73]: the Wordpiece tokenizer is trained over the dump of Wikipedia in the largest 100 languages, which generates a multilingual vocabulary that contains tokens from each of those languages. While in many cases (due to how the tokenizer vocabulary is trained, this is more evident in the English language) tokens correspond to full words, in some cases words are split into multiple tokens (*sub-words*) with flag characters to denote that a token should be taken together its predecessor in the sentence.

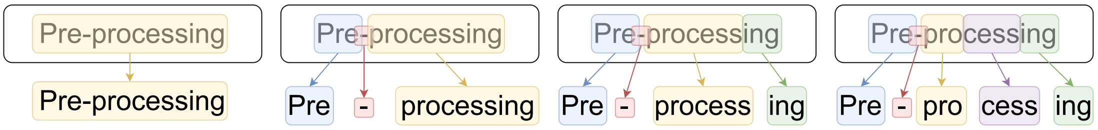


Figure 2.5: Example of different tokenization strategies.

Figure 2.5 displays an example of how it is possible to tokenize a word according to different rules, producing different sets of tokens in the process.

In this work we do not focus on the training of a language model. However, tokenization is still a major concern due to the fact that some datasets feature very long and complex textual strings: these strings should be "unpacked" in some way to not lose the information contained therein. This is where tokenization comes into play, although in our applications it is sufficient to split strings by whitespaces.

We go into more detail about this in Chapter 3 and 4.

Contextual Embeddings and Attention

A major drawback of "bag-of-words" methods is their representation of homonyms, i.e. words that have the same spelling, but unrelated meanings: for example, the word "bat" indicates both an animal and a club-like object; "bag-of-words" methods would generate a singular embedding for "bat" which combines the two different interpretations.

ELMo [74] was one of the first solutions to the problem of contextual embeddings, which it solved by generating representations in such a way that each token is assigned a vector that is a function of the entire input sentence: this allows to have word representations that can draw from the context to better represent the meaning of the word in the sentence.

The real revolution came to be with the application of the concept of Attention and its use in the representation of language. "Attention" describes the process of focusing on a specific, discrete aspect of information, while ignoring other perceptible information [75]. In a similar fashion to how neural networks attempt to mimic neurons in a brain, attention methods in DL attempt to imitate the attention process by focusing on more relevant parts of the piece of information at hand. DL methods can therefore "attend" to specific parts of the input that are more relevant to the task at hand: this allows models to better allocate resources in the training and improve results. Attention mechanisms can be applied to all kinds of information, from images [76], to speech recognition [77], to natural language [73, 78], to graph structures [79], to data imputation [80]. The effectiveness of the attention mechanism has been demonstrated by the very large corpus of work that rely on it [75], with language models such as the Transformer architecture [78] and BERT [73] standing at the forefront thanks to the degree of disruption they caused in the NLP field.

Interestingly, there are remarkable similarities between attention-based systems: NLP transformer-based [78] models can be connected to GATs [79] by imagining each sentence studied by the NLP model as a fully connected graph, with the graph attention network *attending* to each word as if it were a normal graph node [81].

BERT-related models are renowned for their capability to be fine-tuned and modified to be applied to a variety of different problems, which allows to skip the very expensive training step and slightly modify the training weights (usually in the later layers) to better fit to one's application. A different, but related research direction revolves around the production of computer-generation realistic-looking text. The GPT architectures [82–84] are some of the main examples of generative language models.

The concept of attention is used in Chapter 5.

2.3.2 Graph Embedding

There exist a wide variety of graph analysis tasks [85, 86], with just as large a number of algorithms for tackling them. DL-based algorithms for data analysis can be distinguished between algorithms that produce low-dimensional representations of nodes to be used for further tasks [85], and architectures that model an entire task end-to-end [87]. We refer to the first category as **Graph embeddings** and to the second as **Graph Neural Networks** (GNNs).

Graph embedding algorithms employ the low-dimensional node representations as inputs to ML-based techniques, so that a single set of node representations can be used for multiple applications [85, 86], such as link prediction [88], classification [89], clustering [90]. For example, t-SNE [91] or PCA projections can be used in conjunction with the embeddings generated by graph embedding methods for the visualization problem [92], while clustering algorithms such as k-means or DBSCAN [93] can be employed to perform node clustering. Conversely, GNNs include architectures that are designed with the objective of creating an end-to-end solution that can carry out graph analysis tasks (e.g. node classification, link prediction) without relying on external methods. *We use the first approach in Chapters 3 and 4 with* EmbDI *and the second in Chapter 5 with GRIMP.*

Graph Embeddings

Graph embeddings are defined as follows [85]:

Definition 2.3.1 (Graph embedding)**.** Given a graph $G = (V, E)$, a graph embedding is a mapping $f : v_i \rightarrow \mathbf{y_i} \in \mathcal{R}^d \; \forall i \in [1, \ldots, n]$ such that $d \ll |V|$ and the function f preserves some proximity measure defined on graph G.

Graph embeddings generation techniques can be categorized as factorization-based, random walk-based and deep learning-based [85]. Examples of the first category include [94–97]. Random Walk-based models include DeepWalk [41], walklets [98], *node2vec* [42] and HARP [99]. *node2vec* and DeepWalk work by generating random walks, then optimizing the likelihood of observing the last k nodes and the next k nodes in a random walk centered at a node v_i. *node2vec* improves over DeepWalk by employing biased random walks to balance breadth-first and depth-first movement. Optimizing the likelihood of observing nodes in a random walk is akin to what the word2vec [37] algorithm is doing to optimize the co-occurrence of words in a sentence: we make use of this in the development of EmbDI.

Chapters 3 and 4 are based on graph embeddings generated with these approaches.

Graph Neural Networks

GNNs are deep learning-based methods that are inspired from the success of CNNs [100], RNNs [101], and autoencoders [102]: these techniques have been generalized over the past years to handle graph data [87]. In a generic GNN architecture, the network takes as input the graph (with its adjacency matrix A) and a set of node features X, then aggregates the features of each node with its neighbors via a message passing operation [103]; finally, the features are transformed by a non-linear function. As a result, the representation of each node encodes some information relative to the node itself, and to its direct neighbors. For example, a graph convolution can be generalized from a 2D convolution of an image by considering each pixel as a node, and the convolution to be performed over the node neighbors. Naturally, the main difference is that graph nodes do not have a fixed number of neighbors, unlike pixels in images. Figure 2.6 depicts an example of a graph convolution centered on the red node, which is connected directly to the yellow nodes. Similar nodes (either in features or neighborhoods) are placed close to each other in the vector space. By adding more layers on top of each other, it becomes possible to propagate a node's features to longer distances (that is, a larger number of hops from the starting node). The output of the stack of layers contains the distributed representations of the graph nodes, which can then be used for node classification [43], link prediction [104, 105] and more. Unlike previous works in which the node embeddings are generated on the plain graph, and then they are employed for carrying out a task [85], GNNs model the entire task from end to end by combining graph structure and node features using the message passing operation. The resulting features are then fed to a final layer that handles the task (classification, link prediction etc.). Depending on whether the entire graph is operated on at the same time or not, GNNs can be divided into two categories, Spectral Convolutional GNNs (Spectral ConvGNNs) and Spatial Convolutional GNNs (Spatial ConvGNNs) [87] respectively.

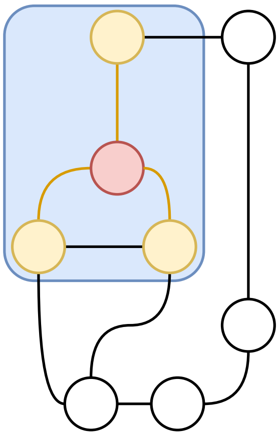


Figure 2.6: Graph Convolution applied to a graph node.

Spectral ConvGNNs assume that graphs are undirected, then apply a series of transforms over the Laplacian of the entire graph to obtain the new representation, which is then passed on to the final task. Spectral ConvGNNs include Spectral CNN [106], ChebNet [107] and Graph Convolutional Network (GCN) [43]. GCN reduces the scale of the convolution from taking the entire graph at the same time, down to handling the neighborhood of one node at a time. This is more scalable, and it is still possible to work on the entire graph by iterating over all nodes.

In Spatial ConvGNNs, the neighborhood of a node can be expanded by increasing the depth of the network and keeping the weights in each layer independent from each other. Spatial ConvGNNs examples include Neural Networks for Graphs (NN4G) [108], Message Passing Neural Network (MPNN) [103], Graph Attention Network (GAT) [79] and GraphSAGE [109].

In Spectral ConvGNNs, the combination operation is modeled as the matrix multiplication of the graph adjacency matrix A with the node features, filtered by a matrix that contains learnable parameters. In these models, the output of a layer has the form [110]:

$$H^i \;=\; f \cdot (H^i, A) \;=\; \sigma(AH^{(i-1)}W^i) \qquad (2.2)$$

Here, A is the graph adjacency matrix, σ is the activation function (generally, a ReLU), H^i is the feature matrix output of layer i, W^i is the weight matrix of layer i. $H^0 = \mathcal{X}$ is a feature matrix that contains the embedding representation of each node in the graph. The adjacency matrix A is often [43, 111] modified by adding an identity matrix I_n to obtain $\bar{A} = I_n + A$: this effectively allows the GNN to make use of a node's own features when the combination operation is carried out, which improves the overall training performance. Adding I_n to the adjacency matrix has the practical effect of introducing self-loops in the graph.

Due to scalability issues, it is not always possible (or even necessary) to work with the full adjacency matrix. Spatial ConvGNNs represent a more efficient solution which revolves around performing the convolution operation on one node at a time, by aggregating only that node's neighbors. To propagate the result of the convolution of one node, a "message passing" [103] operation is performed. With message passing, the results of the convolution can be passed over the edges that link different nodes; during this operation, the features of each node are combined with those of the node's neighbors (either all, or a sample of the neighbors [109]). The message passing function (or spatial graph convolution) is defined as [87, 103]:

$$\mathbf{h_v^{(k)}} = U_k(\mathbf{h}_v^{(k-1)}, \sum_{u \in N(v)} M_k(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}_{vu}^e)) \tag{2.3}$$

Here, $\mathbf{h_i^{(k)}}$ describes the features of node i at layer k, $N(v)$ is the set of neighbors of v and $\mathbf{x}_{vu}^e$ is the vector of features of edge vu; $\mathbf{h_v^{(0)}} = \mathbf{x}_v$, while $U_k(\cdot)$ and $M_k(\cdot)$ are functions with learnable parameters.

These spatial-based convolutional GNNs are far more efficient and versatile than spectral-based GNNs, and a number of solutions that incorporate the general message-passing framework have been developed [79, 109, 112]. Spatial ConvGNNs are preferred over Spectral models for a number of reasons [87]. First, spectral models are computationally less efficient than spatial modes as they require to either compute eigenvectors on the graph's matrix, or to handle the whole graph in memory at the same time. Spatial models do not have this issue, since they can propagate the convolution through message passing, and computations can be completed in batch. Second, spectral graphs assume that the graph is fixed and generalize poorly to new graphs, with any perturbation requiring expensive recomputations. Third, spectral-based models cannot operate on directed graphs, whereas spatial-based models do not have this limitation and can can handle edge inputs, directed graphs and heterogeneous graphs [113, 114]. Regardless of the internal structure, given the graph adjacency matrix and the node features, a GNN will produce, for each node, new features that aggregate the starting features with those of its neighbors.

We go into the detail of how to use GNNs for data imputation in Chapter 5.

2.4 Multi-Task Learning

In Machine Learning, Multi-task Learning (MTL) [115] is a learning paradigm whose aim is to jointly learn multiple related tasks in such a way that the knowledge contained in a task can be leveraged by other tasks [116]. In MTL, rather than optimizing a model on a singular objective, multiple different objective functions are optimized at the same time, while sharing information over the course of the training. Training multiple tasks at the same time allows each task to learn from the others, while improving the generality of the final model.

We use the definition of Multi-Task Learning reported in [116]:

Definition 2.4.1 (Multi-Task Learning)**.** Given a set $\{\mathcal{T}_i\}_{i=1}^m$ of m learning tasks, where all or a subset of the tasks are related, the objective of multi-task learning is to train the m tasks together, in order to improve learning of a model for each task $\mathcal{T}_i$ by using the knowledge contained in all or a subset of other tasks.

Depending on whether tasks lie in the same feature space or not, MTL can be either homogeneous or heterogeneous respectively. The heterogeneous case includes settings in which tasks include classification and regression, however it can be generalized to include supervised learning, unsupervised learning, semi-supervised learning. On the other hand, the homogeneous case includes tasks with only one type.

A second, orthogonal direction over which MTL methods can be characterized is the knowledge sharing procedure used in a given architecture. Depending on what knowledge is shared among tasks, it is possible to distinguish among feature-based, instance-based and parameter-based multi-task learning. Feature-based MTL tries to learn features common to different tasks as a way of sharing knowledge. Instance-based MTL identifies useful data instances in a task for other tasks to employ. Finally, parameter-based MTL uses model parameters in a task to help learn parameters in other tasks.

Furthermore, MTL architectures can be distinguished by their method for sharing parameters, and thus, information, across different tasks [117]. In soft parameter sharing, each task receives its own set of parameters, and parameter sharing architectures are used to connect the different tasks. In hard parameter sharing, the set of parameters is divided into shared and task-specific operations, often by having a shared layer that then branches into task-specific structures.

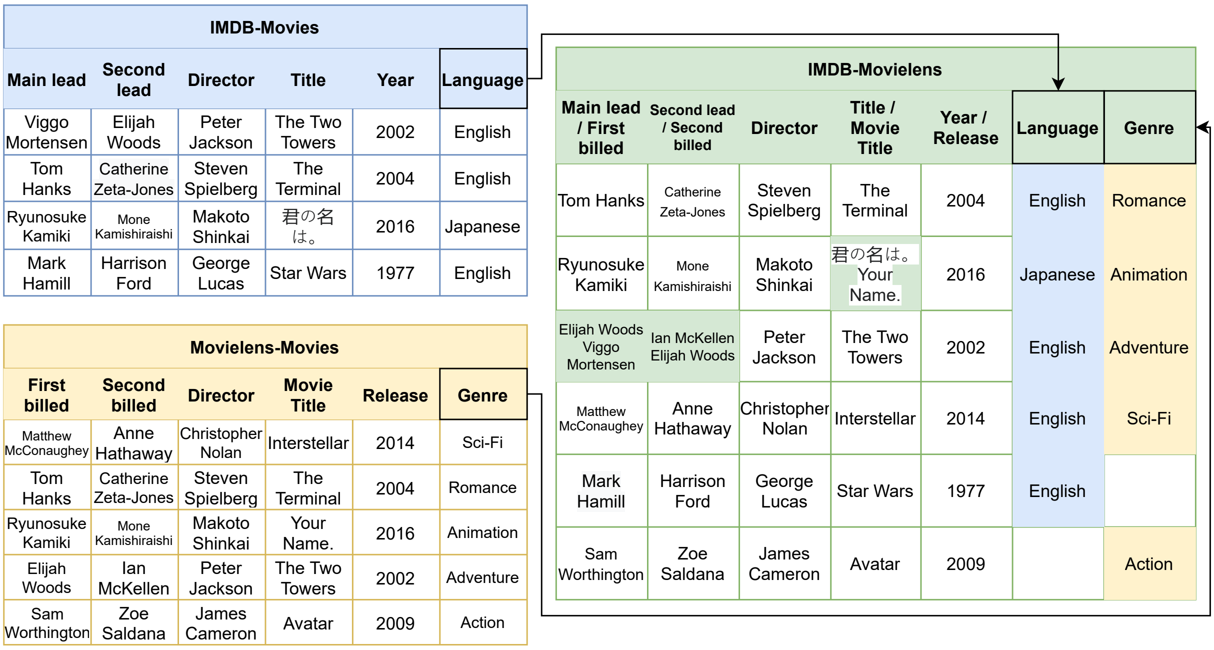
Multi-task learning is a major part of the contribution described in Chapter 5.

2.5 Data Curation

Data Curation can be defined as the acquisition, care and documentation of data, its proper packaging for reuse, and its maintenance so that data preserves its value over time [118, 119]. For this work, we concentrate our attention on two specific problems: **Data Integration** and **Data Imputation**.

From here on, for the sake of brevity we use the term “data curation” to refer to both subjects at the same time, while we specify the specific task when needed.

2.5.1 Data Integration



IMDB-Movies

Main lead	Second lead	Director	Title	Year	Language
Viggo Mortensen	Elijah Woods	Peter Jackson	The Two Towers	2002	English
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	English
Ryunosuke Kamiki	Mone Kamishiraishi	Makoto Shinkai	君の名は。	2016	Japanese
Mark Hamill	Harrison Ford	George Lucas	Star Wars	1977	English

Movielens-Movies

First billed	Second billed	Director	Movie Title	Release	Genre
Matthew McConaughey	Anne Hathaway	Christopher Nolan	Interstellar	2014	Sci-Fi
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	Romance
Ryunosuke Kamiki	Mone Kamishiraishi	Makoto Shinkai	Your Name.	2016	Animation
Elijah Woods	Ian McKellen	Peter Jackson	The Two Towers	2002	Adventure
Sam Worthington	Zoe Saldana	James Cameron	Avatar	2009	Action

IMDB-Movielens

Main lead / First billed	Second lead / Second billed	Director	Title / Movie Title	Year / Release	Language	Genre
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	English	Romance
Ryunosuke Kamiki	Mone Kamishiraishi	Makoto Shinkai	君の名は。 Your Name.	2016	Japanese	Animation
Elijah Woods Viggo Mortensen	Ian McKellen Elijah Woods	Peter Jackson	The Two Towers	2002	English	Adventure
Matthew McConaughey	Anne Hathaway	Christopher Nolan	Interstellar	2014	English	Sci-Fi
Mark Hamill	Harrison Ford	George Lucas	Star Wars	1977	English	
Sam Worthington	Zoe Saldana	James Cameron	Avatar	2009		Action

Figure 2.7: Example of integration of two tables about movies.

With **Data Integration**, we refer to the task of "integrating", that is "combining" data available from different sources with the objective of producing a clean view of the data for the user to perform further operations on [118, 120]. This involves the identification of those attributes that have the same domain, thus mapping the schema of one table with another (**Schema Matching**), and the matching of tuples that refer to the same entity in the domain (**Entity Resolution**). Performing these tasks allows to produce a new version of the table that "integrates" information coming from multiple sources.

We use Figure 2.7 as a running example for both entity resolution and schema matching. On the left we have two tables that contain information about movies (their title, cast, director, release year and additional attributes); we would like to obtain a new version of the tables similar to what is depicted on the right, produced by an ideal Entity Resolution algorithm. A first pass in the algorithm would perform schema matching by aligning (matching) those columns that (despite their different name) refer to the same domain, such as "Main lead" and "First billed", or "Release" and "Year". Unmatched columns ("Language" and "Genre") are not dropped: the information they contain can still be employed after the entity resolution step. In the entity resolution step, rows that refer to the same entity (in this case, to the same movie) should be matched and their information combined. This is what happens with "The Terminal", "Interstellar", "Your Name." and "The Two Towers". The latter two movies contain collisions (green background), i.e., cells that share the same attribute, but different values. In this example, we keep the values coming from both sources. Finally, some cells are left empty: this is due to the fact that some of the movies in one table do not have a match in the other. As a result, we lack the information required to complete the tuple.

Schema Matching

Part of the information extraction problem revolves around the problem of integrating data coming from different sources [121, 122].

Integration can be achieved by translating data in a common format, thus materializing it in the target schema. This is related to the problem of generating executable data transformations between two schemas, where a schema is a formal structure that represents an engineered artifact. Schemas need not be relational (e.g. XML), as long as they are structured. Such problem is known as schema mapping generation [123, 124]. Schema Matching (SM) is the step that precedes schema mapping generation, in which correspondences are sought between disconnected information atoms (e.g. attributes in relational data), which are then used to build mappings with.

A correspondence (or matching) is a relationship between one or more elements of one schema and one or more elements of the other; when performing schema matching with relational tables, a match would link columns that refer to the same concepts. Similarly to entity resolution, schema matching is a widely researched topic [122, 125–128] with a wide array of different techniques employed to treat the subject [129, 130]. As we focus on relational tables, with schema we refer in the following as the set of attributes in the relational table.

Depending on the type of data that is taken as input by the SM algorithm, it is possible to classify methods that rely on schema-level information, on instance data or a combination of the two [128]. Examples of the first category include Cupid [131] and COMA [132], GLUE [133] is an example of the second category, while QOM [134] and Similarity Flooding [135] are examples of the latter. Depending on how the matching operation is carried out, it is possible to further distinguish among SM methods. [130] and [129] provide a good taxonomy of the different matching components:

- **Value Overlap Matchers** match attributes whose values are overlapping enough. **Data Type Matchers** flag columns as relevant or irrelevant depending on their datatype. **Distribution Matchers** flag columns based on their value distributions.

- Alternatively, matching can be done on the schema structure and metadata. **Attribute Overlap Matchers** rely on the similarity between attribute names, and match attributes whose similarity is higher than a given threshold. **Semantic Overlap Matchers** employ external sources of knowledge (such as knowledge bases) to derive labels for each attribute. Then, two attributes match if their labels overlap enough.

- Finally, matching can be done by employing other types of resources, such as auxiliary information (thesauri, dictionaries, acronyms) or matching based on usage statistics. **Embeddings Matchers** match columns based on the similarity between the columns' embeddings. Embeddings can be generated on the data at hand, or by reusing pre-trained embeddings.

Many SM algorithms combine and employ different matcher types at the same time, such as Cupid [131] (attribute, semantic, data type), Similarity Flooding [135] (attribute, data type), COMA [132] (attribute, value, semantic, data type, distribution). Schema matching methods that rely on embedding matchers include Seeping Semantics [136] and REMA [137].

Entity Resolution

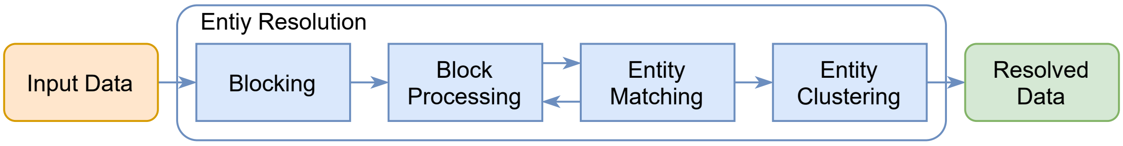


Figure 2.8: General Entity Resolution end-to-end pipeline [1]

Entity Resolution (ER) is the task of identifying different descriptions (normally, records) that refer to the same real-world entity, when entity identifies are not available [1, 138–140]. Entity resolution is normally carried out over multiple datasets (*record linkage*) or within the same data source (*deduplication*), and it aims to link descriptions that correspond to the same entity into *matches*. Depending on what are the characteristics of the input, ER problems can be split into [1, 141]:

- Clean-Clean ER, when the input consists of two overlapping, but “clean” (that is, that do not contain duplicates) data sources.
- Dirty ER, when there exists a single, “dirty” input which contains duplicated values.
- Multi-source ER, when the input consists of more than two sources.

All ER problems listed above share the same general processing tasks that are describe in the end-to-end pipeline depicted in Figure 2.8 [1]. We briefly discuss each step. *Blocking* [142, 143] is typically applied to the input data to reduce the number of comparisons to be performed by discarding entities that are unlikely to be matches. Similar entities are placed in “blocks”, which are identified by some *blocking key*. Comparisons are then performed exclusively among entities that belong to the same block. *Block processing* [144] has the objective of optimizing blocks so to reduce the number of comparisons without significant impact on the final matching accuracy. This can be done by removing redundant comparisons (comparisons repeated across multiple blocks) and superfluous comparisons (comparisons between values known to not be in match). The *Entity Matching* task consists in the application of a similarity function F such that, given two entities e_i and e_j, F outputs $F(e_i, e_j) = true$ if the entities are in match, and $F(e_i, e_j) = false$ if they do not match. Finally, the task of *Entity Clustering* [145] is performed to infer indirect matches between matches that have been detected by the similarity function, with the objective of overcoming possible shortcomings in the similarity function. The clustering step produces as output a set of disjoint sets of entity descriptions $R = \{r_1, r_2, \ldots, r_m\}$ such that 1) $\forall e_i, e_j \in r_k,\ F(e_i, e_j) = true$; 2) $\forall e_i \in r_k, \forall e_j \in r_l,\ F(e_i, e_j) = false$; 3) $\bigcup_{r_i} r_i \in = \varepsilon$, where ε describes the input entity collection. While not every ER system implements all the functions and optimizations described here and in Figure 2.8, all employ a combinations of at least some of them.

ER systems can be distinguished in two categories: rule-based systems such as [146–150], which produce sets of rules for matching entities (e.g., "if these two books have the same ISBN, then they should be matched"), and ML-based systems, which instead train a model to extract matches [70, 71, 151–156]. ML-based methods often have higher effectiveness than rule-based methods [1, 146], however they are not as interpretable by humans; on the other hand, rule-based methods enjoy better interpretability with tradeoffs in effectiveness; there has been a drive towards bringing interpretability in ML-based models for ER with works such as Mojito [157].

Both schema matching and entity resolution are the focus of Chapter 4.

2.5.2 Data Imputation

With **data imputation** we describe the task of pre-processing data that contain missing or erroneous values with the objective of correcting the errors on the basis of the context provided by the existing features. This task is an important step in any pipeline since the presence of missing values can be harmful when the data is used for training models [158, 159].

Formally, the problem of data imputation can be defined as follows [80]:

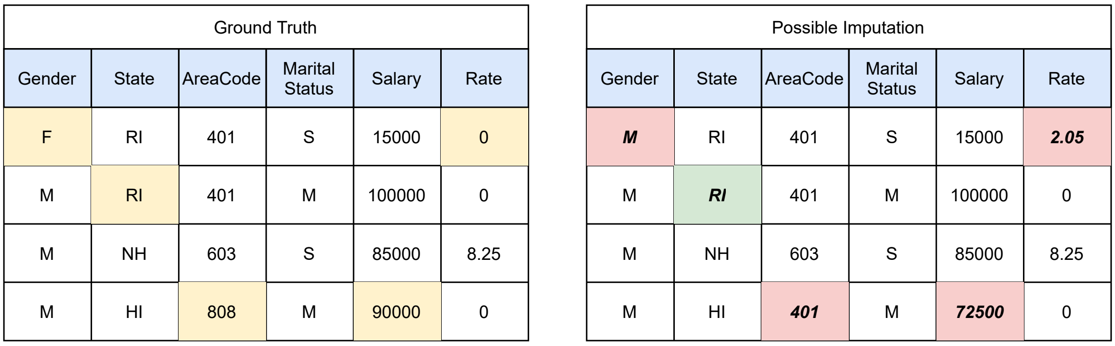
Definition 2.5.1 (Data Imputation)**.** Given a dataset $\mathcal{D}$ with schema $\mathcal{R}$, each attribute $A_j \in \mathcal{R}$ can be either n numerical or categorical, with $N(\mathcal{R})$ and $C(\mathcal{R})$ denoting numerical and categorical attributes respectively. The domain of attributes $A_j \in N(\mathcal{R})$ is $dom(A_j) = \mathbb{R}$; the index domain of $A_i \in C(\mathcal{R})$ is $\mathcal{I} = \{1, \ldots, |A_i|\}$ where $|A_i|$ is the cardinality of A_i.

Definition 2.5.2 (Missing Value)**.** A cell with a missing value in the i-th tuple $t_i \in \mathcal{D}$ on the j-th attribute $A_j \in \mathcal{R}$ is denoted as $t_i[A_j] = \varnothing$. The set of all missing values $t_i[A_j]$ in dataset $\mathcal{D}$ is $\mathcal{M}_\mathcal{D}$, such that $t_i[A_j] \in \mathcal{M}_\mathcal{D} \iff t_i[A_j] = \varnothing$.

Definition 2.5.3 (Imputed dataset)**.** Let $\tilde{\mathcal{D}}$ be the imputed version of $\mathcal{D}$, where $\forall i, j$ with $t_i[A_j] \in \mathcal{D}$, $t_i[A_j] = \varnothing$, while for $\tilde{t}_i[A_j] \in \tilde{\mathcal{D}}$, $\tilde{t}_i[A_j] \neq \varnothing$.
$\mathcal{D}^*$ denotes the ground truth dataset without missing data.

Depending on the datatype of an attribute, we measure the imputation performance using different metrics. For numerical attributes, we rely on RMSE to measure the imputation accuracy. It is possible to use alternatve metrics for specific distributions (e.g. displacement error for ordinal variables [48]), however we do not use them in this work. For categorical attributes, we use accuracy. With a slight abuse of notation, we set $t_i[A_j] = t_{ij}$ for the next equation.

$$accuracy(\mathcal{D}^*, \tilde{\mathcal{D}}) = \frac{1}{|\mathcal{M}_\mathcal{D}|} \sum_{\tilde{t}_{ij} \in \mathcal{M}_\mathcal{D}} 1(\tilde{\mathcal{D}}_{ij} = \mathcal{D}^*_{ij}) \tag{2.4}$$



Ground Truth					
Gender	State	AreaCode	Marital Status	Salary	Rate
F	RI	401	S	15000	0
M	RI	401	M	100000	0
M	NH	603	S	85000	8.25
M	HI	808	M	90000	0

Possible Imputation					
Gender	State	AreaCode	Marital Status	Salary	Rate
M	RI	401	S	15000	***2.05***
M	***RI***	401	M	100000	0
M	NH	603	S	85000	8.25
M	HI	***401***	M	***72500***	0

Figure 2.9: Example of missing value imputation.
Cells with yellow background need to be imputed, cells with red background have been imputed incorrectly, cells with green background have been imputed correctly. .

Figure 2.9 shows an example of the result (exaggerated for the sake of the explanation) of imputing the missing values in the ground truth. The dataset $\mathcal{D}$ is mixed, with categorical attributes Gender, State and Marital Status, and numerical attributes AreaCode, Salary and Rate. AreaCode is an example of numerical attribute that should, however, be treated as a categorical value. In the example, the "Ground truth" table features a Functional Dependency such that the values in the AreaCode attribute *imply* the values in State (so that $401 \rightarrow RI$, for example). The table "Possible Imputation" contains a number of wrong imputations, resulting in a low imputation accuracy: indeed, the missing value in the State column is the only value that has been imputed correctly. Here we see how the imputation algorithm is deviating to the Mode in columns Gender and AreaCode, while the values in columns Salary and Rate are imputed by using the average of all other missing values. While this is a toy example, it should be enough to give an idea of some of the problems that imputation algorithms can face when handling mixed-type relational data.

Handling missing data is not straightforward, and since, by definition, there is no information about missing data, filling vacancies may introduce some degree of bias in the corrected data. For this reason, a common solution to the problem is simply dropping all training samples that include missing values [160]. It goes without saying that this is not ideal: after all, we could be suppressing a sizeable amount of training data, especially if nulls are distributed somewhat uniformly over the entire training corpus.
Missing data in a target dataset can belong to one of three different distributions [160–162]:

- **MCAR** (Missing Completely At Random): the sampling phenomenon that introduces missing values is such that every single missing value is completely independent of all factors, both present in the data and external to it. The independence assumption is very strong, and it does not generally hold in real cases.

- **MAR** (Missing At Random): in this case, missing values are distributed according to a distribution that can be reconstructed from information present in variables that present full information.

- **MNAR** (Missing Not At Random): any data that is neither MCAR nor MAR is MNAR. In this case, missing values are related to the reason they're missing. No assumption is made on the distribution, so real cases fall mostly in this case. MNAR errors are prevalent in medical literature [160, 162, 163]

When the missing data are numeric, simple imputation techniques such as mean substitution or regression are sometimes applied [160]. Neither solution, however, can reliably fix errors due to the inherent lack of information about said missing values. MICE [164] represents a further step in sophistication and carries out imputation by solving a chain of regression problems.

There are a wide variety of different missing data imputation methods, that work on various degrees of complexity [165]. In fact, there exist a number of simple methods that can be applied to the imputation problem, ranging from imputing with the most common catgorical value (or global average for numerical variables) [166], to K-Nearest Neighbor imputation [167]. **Rule-based methods** produce a set of more or less interpretable rules, which are then used to carry out imputation; examples of rule-based imputation systems include RIPPER [168] and CN2 [169]. Non-interpretable models tend to include ML-based algorithms. These can be distinguished in two categories, depending on how the imputation is being generated. **Discriminative models** are trained to select a solution in the domain to impute missing values. MIDA [170], MICE [164], MissForest [45], Aimnet [80] belong to this category. **Generative models** employ a different approach in which the imputed version of the dataset is generated by the model. Systems that belong to this category include SVM-based imputation methods [171], HI-VAE [48], GAIN [35], MIWAE [172].

The development of a novel data imputation method is at the center of Chapter 5.

2.6 Summary

In this chapter we introduced the fundamental concepts and ideas we work with in the rest of the thesis: relational tables, graphs and Vector Space Models. We then discussed the main topics of Data Integration and Data Imputation, describing the main issues in the subjects, as well as previous work. In the next chapter we introduce EmbDI, our first step into the design and development of a Data Integration system based on Vector Space Models.

Chapter 3

Generating Table Embeddings

In this chapter, we focus on the development of a system that, given a relational table that contains mixed data, is able to produce an embedding representation of the entities involved in the dataset. With this we consider not only the cell values or the rows/attributes in the table, but both categories at the same time. Indeed, EmbDI is able to organically generate embeddings for all entities thanks to a novel representation of the table content based on a "tripartite" graph.

Deep learning based techniques have been recently used with promising results for data integration problems [70, 136]. Some methods use *pre-trained* embeddings that were trained on a large corpus such as Wikipedia. Unfortunately, these pre-trained embeddings may not always be an appropriate choice for tables that feature custom vocabularies, such as enterprise datasets. Other methods adapt techniques from natural language processing to obtain embeddings for the enterprise's relational data. However, this approach blindly treats a tuple as a sentence, thus losing a large amount of contextual information present in the tuple.

We propose algorithms for obtaining *local embeddings* that are effective for data integration tasks on relational databases. We make four major contributions. First, we describe a compact graph-based representation that allows the specification of a rich set of relationships inherent in the relational world. Second, we propose how to derive sentences from such a graph that effectively "describe" the similarity across elements (tokens, attributes, rows) in the two datasets; the embeddings are learned based on such sentences. Third, we propose effective optimizations to improve the quality of the learned embeddings and the performance of integration tasks. Finally, we design and implement a diverse collection of criteria to evaluate relational embeddings and perform an extensive set of experiments validating them against multiple baseline methods. Our experiments show that our framework, EmbDI, produces meaningful results for data integration tasks such as schema matching and entity resolution both in supervised and unsupervised settings.

The chapter contains part of the paper "**Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks**" [173], as well as work from the Demo under submission "**EmbDI: Embeddings Generation for Integrating Relational Datasets**". The rest of the paper focuses on Data Integration, and will be reported in Chapter 4. The code we developed for this contribution is available in the repository `https:`

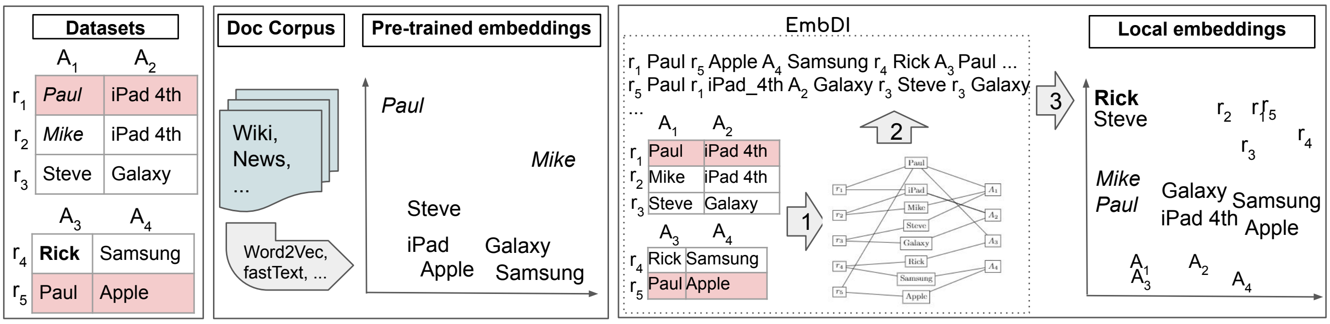


Figure 3.1: Illustration of a simplified vector space learned from text (prior approaches) and from data (EmbDI).

//gitlab.eurecom.fr/cappuzzo/embdi. [173] has received the ACM badges for **Results Reproduced**, **Artifacts Available** and **Artifacts Evalued and Reusable**.

3.1 Introduction

3.1.1 Local Embeddings for Data Integration

We advocate for the design of local embeddings that leverage both the relational nature of the data and the downstream task of data integration.
Tuples are not sentences. Simply adapting embedding techniques originally developed for textual data ignores the richer set of semantics inherent in *relational* data. Consider a cell value $t[A_i]$ belonging to an attribute A_i in tuple t, e.g., "Mike" in the first relation from the top in Figure 3.1. Conceptually, it has a semantic connection with both other attributes of tuple t (such as "iPad 4th") and other values from the domain of attribute A_i (such as "Paul"). Existing embedding techniques cannot model these semantic connections.
Embedding generation must span different datasets. Embeddings must be trained using heterogeneous datasets, so that they can meaningfully leverage and surface similarity across data sources. A notion of similarity between different types of entities, such as tuples and attributes, must be developed. Tuple-tuple and attribute-attribute similarity are important features for entity resolution and schema matching.

There are multiple challenges to overcome. First, it is not clear how to encode the semantics of the relational datasets into the embedding learning process. Second, datasets may share very limited amount of information, have radically different schemas, and contain a different number of tuples. Finally, datasets are often incomplete and noisy. The learning process is affected by low information quality, generating embeddings that do not correctly represent the semantics of the data.

3.1.2 Contributions

We present EmbDI, a framework for building relational, local embeddings for data integration that introduces a number of innovations to overcome the challenges above.

We identify crucial components and propose effective algorithms for instantiating each of them. EmbDI is designed to be modular so that anyone can customize it by plugging in other algorithms and benefit from the continuing improvements from the deep learning and the database communities. The right-hand side of Figure 3.1 shows the main steps in our solution.

1. Graph Construction. We leverage a compact "tripartite" graph-based representation of relational datasets that can effectively represent a rich set of syntactic and semantic relationships between cell values. Specifically, we use a heterogeneous graph with three types of nodes. *Token* nodes correspond to the content of each cell in the dataset. *Record Id* nodes (RIDs) assign a unique token to each tuple. *Column Id* nodes (CIDs) assign a unique token to each column/attribute. These nodes are connected by edges based on the structural relationships in the schema. This graph is a compact representation of the original datasets that highlights overlap and explicitly represent the primitives for data integration tasks, i.e., records and attributes.

2. Embedding Construction. We formulate the problem of obtaining local embeddings for relational data as a graph embeddings generation problem. We use random walks to quantify the similarity between neighboring nodes and to exploit metadata such as tuple and attribute IDs. This method ensures that nodes that share similar neighborhoods will be in close proximity in the final embeddings space. The corpus that is used to train our local embeddings is generated by materializing these random walks.

3. Optimizations. Learning embeddings can be a difficult task in the presence of noisy and incomplete heterogeneous datasets. For this reason, we introduce an array of optimization techniques that handle difficult cases and enable refinement of the generated embeddings. The flexibility of the graph enables us to naturally represent external information, such as data dictionaries, to merge values in different formats, and data dependencies, to impute values and identify errors. We propose optimizations to handle imbalance in the datasets' size and the presence of numerical values (usually ignored in textual word embeddings).

Experimental Results. We propose an extensive set of desiderata for evaluating relational embeddings for data integration. Specifically, our evaluation focuses on three major dimensions that measure how well do the embeddings (a) learn the tuple-, attribute- and constraint-based relationships in the data, (b) learn integration specific information such as tuple-tuple and attribute-attribute similarities, and (c) improve the behavior of DL-based data integration algorithms. As we shall show in the experiments, our proposed algorithms perform well on each of these dimensions.

3.1.3 Outline

Section 3.2 introduces background about embeddings and data integration. Section 3.3 shows a motivating example that highlights the limitations of prior approaches and identifies a set of desiderata for relational embeddings. Section 3.4 details the major components of the framework. Section 3.5 reports the experiments we conducted to measure the quality of the embeddings that we generated.

3.2 Background

Embeddings. Embeddings map an entity such as a word to a high dimensional real valued vector. The mapping is performed in such a way that the geometric relation between the vectors of two entities represents the co-occurrence/semantic relationship between them. Algorithms used to learn embeddings rely on the notion of "neighborhood": intuitively, if two entities are similar, they frequently belong to the same contextually defined neighborhood. When this occurs, the embeddings generation algorithm will try to force the vectors that represent these two entities to be close to each other in the resulting vector space.

Word Embeddings [174,175] are trained on a large corpus of text and produce as output a vector space where each word in the corpus is represented by a real valued vector. Usually, the generated vector space has either 100 or 300 dimensions. The vectors for words that occur in similar context – such as SIGMOD and VLDB – are in close proximity to each other. Popular architectures for learning embeddings include continuous bag-of-words (CBOW) or skip-gram (SG). Recent approaches rely on using the context of word to obtain a contextual word embedding [176,177].

Node Embeddings. Intuitively, node embeddings [42] map nodes to a high dimensional vector space so that the likelihood of preserving node neighborhoods is maximized. One way to achieve this is by performing random walks starting from each node to define an appropriate neighborhood. Popular node embeddings are often based on the skip-gram model, since it maximizes the probability of observing a node's neighborhood given its embedding. By varying the type of random walks used, one can obtain diverse types of embeddings [178].

Embeddings for Relational Datasets. The pioneering work of [179] was the first to apply embedding techniques for extracting latent information from relation data. Recent extensions [180,181] leverage the learned embeddings to develop a "cognitive" database system with sophisticated functionality for answering complex semantic, reasoning and predictive queries. Termite [182] seeks to project tokens from structured and unstructured data into a common representational space that could then be used for identifying related concepts through its Termite-Join approach. Freddy [183] and RetroLive [184] produce relational embeddings that combine relational and semantic information through a retrofitting strategy. There has been prior work that learn embeddings for specific tasks like entity matching (such as DeepER [70] and DeepMatcher [71]) and schema matching (Rema [137]). Our goal is to learn relational embeddings that is tailored for data integration and can be used for multiple tasks. All of the prior approaches rely on viewing the tuple as a textual document. As we shall show later, our choice to use a graph based representation results in better embeddings.

3.3 Motivating Example

In this section, we discuss an illustrative example that highlights the weaknesses of current approaches and motivates us to design a new approach for relational, local embedding.

Consider the scenario where one utilizes popular pre-trained embeddings such as word2vec, GloVe, or fastText. Figure 3.1 shows a hypothetical filtered vector spaces for the tokens in an example with two small customer datasets. We observe that the pre-trained embeddings suffer from a number of issues when we use them to model the two relations.

1. A number of words in the dataset, such as “Rick”, are not in the pre-trained embedding. This is especially problematic for enterprise datasets where tokens are often unique and not found in pre-trained embeddings.

2. Embeddings might contain geometric relationships that exist in the corpus they were trained on, but that are missing in the relational data. For example, the embedding for token “Steve” is closer to tokens “iPad” and “Apple” even though this is not implied in the data.

3. Relationships that do occur in the data, such as between tokens “Paul” and “Mike”, are not observed in the pre-trained vector space.

Naturally, learning local embeddings from the relational data often produces better results. However, computing embeddings for non integrated data sources is a non trivial task. This becomes especially challenging in settings where data is scattered over different datasets with heterogeneous structures, different formats, and only partially overlapping content. Prior approaches express such datasets as sentences that can be consumed by existing word embedding methods. However, we find that these solutions are still sub-optimal for downstream data integration tasks.

3.3.1 Technical Challenges

We enumerate four challenges that must be overcome to obtain effective embeddings.

1. Incorporating Relational Semantics. Relational data exhibits a rich set of semantics. Relational data also follows set semantics where there is no natural ordering of attributes. Representing the tuple as a single sentence is simplistic and often not expressive enough for these signals.

2. Handling Lack of Redundancy. A key reason for the success of word embeddings is that they are trained on large corpora where there are adequate redundancies and co-occurrence to learn relationships. However, databases are often normalized to remove redundant information. This has an especially deleterious impact on the quality of learned embeddings. Rare words, which are very common in relational data, are typically ignored by word embedding methods.

3. Handling Multiple Datasets. We cannot assume that each of the datasets have the same set of attributes, or that there is sufficient overlap in the tuple values, or even that there is a common dictionary for the same attribute.

4. Handling Hierarchical Data. Databases are inherently hierarchical, with entities such as cell values, tuples, attributes, dataset. Incorporating these hierarchical units as first class citizens in embedding training is a major challenge.

3.4 Constructing Local Relational Embeddings

In this section, we provide a description of our approach and how these design choices address the aforementioned technical challenges. Our framework, EmbDI, consists of three major components, as depicted in the right-hand side of Figure 3.1.

1. In the *Graph Construction* stage, we process the relational dataset and transform it into a compact tripartite graph that encodes various relationships inherent in it. Tuple and attribute IDs are treated as first class citizens.

2. Given this graph, the next step is *Sentence Construction* through the use of biased random walks. These walks are carefully constructed to avoid common issues such as rare words and imbalance in vocabulary sizes. This produces as output a series of sentences.

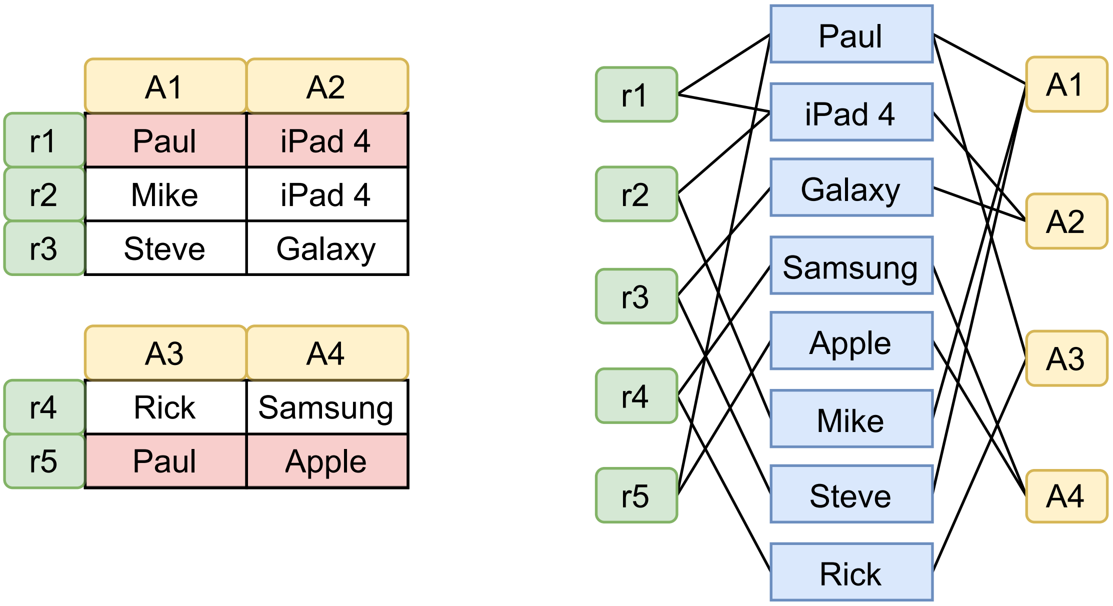
3. In *Embedding Construction*, the corpus of sentences is passed to an algorithm for learning word embeddings. Depending on available external information, we perform optimizations to the graph and the workflow to improve the embeddings' quality.

3.4.1 Graph Construction

Why construct a Graph? Prior approaches for local embeddings seek to directly apply an existing word embedding algorithm on the relational dataset. Intuitively, all tuples in a relation are modeled as sentences by breaking the attribute boundaries. The collection of sentences for each tuple in the relation then makes up the corpus, which is then used to train the embedding. This approach produces embeddings that are customized to that dataset, but it also ignores signals that are inherent in relational data. We represent the relational data as a graph, thus enabling a more expressive representation with a number of advantages. First, it elegantly handles many of the various relationships between entities that are common in relational datasets. Second, it provides a straightforward way to incorporate external information such as "two tokens are synonyms of each other". Finally, when multiple relations are involved, a graph representation enables a unified view over the different datasets that is invaluable for learning embeddings for data integration.

Simple Approaches. Consider a relation R with attributes $\{A_1, A_2, \ldots, A_m\}$. Let t be an arbitrary tuple and $t[A_i]$ be a cell, that is the value of attribute A_i for tuple t. A naive approach is to create a chain graph where tokens corresponding to adjacent attributes such as $t[A_i]$ and $t[A_{i+1}]$ are connected. This will result in m edges for each tuple. Of course, if two different tuples share the same token, then they will reuse the same node. However, relational algebra is based on set semantics, where the attributes do not have an inherent order. So, simplistically connecting adjacent attributes is doomed to fail. Another extreme is to create a complete subgraph, where an edge exists between all possible pairs of $t[A_i]$ and $t[A_{i+1}]$. Clearly, this will result in $\binom{m}{2}$ edges per tuple. With this approach, the number of edges is quadratic in the number of attributes and ignores other token relationships such as "token t_1 and token t_2 belong to the same attribute".

Relational Data as Heterogeneous Graph. We propose a heterogeneous graph with three types of nodes. *Token* nodes correspond to information found in the dataset (i.e. the content of each cell in the relation). Multi-word tokens may be represented as a single entity, get split over multiple nodes or use a mix of the two strategies. We describe the effect of each strategy more in depth in Section 3.5. *Record Id* nodes (RIDs) represent each tuple in the dataset, *Column Id* nodes (CIDs) represent each column/attribute. These nodes are connected by edges according to the structural relationships in the schema. This representation can produce a vector for all RIDs (CIDs) rather than representing them by combining the vectors of the values in each tuple (column). By using this representation, it is possible to handle challenges (1) and (4) in Section 3.3.1.



	A1	A2
r1	Paul	iPad 4
r2	Mike	iPad 4
r3	Steve	Galaxy

	A3	A4
r4	Rick	Samsung
r5	Paul	Apple

Figure 3.2: The EMBDI graph for the two tables also shown in Figure 3.1.

Consider a tuple t with RID r_t. Then, nodes for tokens corresponding to $t[A_1], \ldots, t[A_m]$ are connected to the node r_t. Similarly, all the tokens belonging to a specific attribute A_i are connected to the corresponding CID, say c_i. This construction is generic enough to be augmented with other types of relationships. Also, if we know that two tokens are synonyms (e.g. via wordnet), this information could be incorporated by reusing the same node for both tokens. Note that a token could belong to different record ids and column ids when two different tuples/attributes share the same token. Numerical values are rounded to a number of significant figures decided by the user, then they are assigned a node like regular categorical values; null values are not represented in the graph. We discuss more sophisticated approaches for handling numeric, noisy, and null values in Section 4.3.

Algorithm 1 shows the operations performed during the graph creation with hybrid representation of multi-word tokens (the "combination" approach). Figure 3.2 shows a graph constructed for the datasets in Figure 3.1. Note that this could be considered as a variant of tripartite graph. A key advantage of this choice is that it has the same expressive power as the complete sub-graph approach, while requiring orders of magnitude fewer edges.

Algorithm 1 GenerateTripartiteGraph

```
Input: relational dataset $D$
let $G$ = empty graph
for all $c_i$ in columns($D$) do
  G.addNode($c_i$)
for all $r_i$ in rows($D$) do
  G.addNode($R_i$)    //$R_i$ is the record id of $r_i$
  for all value $v_k$ in $r_i$ do
    if $v_k$ is multi-word then
      for all word in tokenize($v_k$) do
        G.addNode(word)
        G.addEdge(word, $R_i$), G.addEdge(word, $c_k$)
    else if $v_k$ is single-word then
      G.addNode($v_k$)
      G.addEdge($v_k$, $R_i$), G.addEdge($v_k$, $c_k$)
Output: graph $G$
```

3.4.2 Sentence Construction

Graph Traversal by Random Walks. To generate the distributed representation of each node in the graph, we produce a large number of random walks and gather them in a training corpus where each random walk will correspond to a sentence. Using graphs and random walks allows us to have a richer and more diverse set of neighborhoods than what would be possible by encoding a tuple as a *single* sentence. For example, a walk starting from node 'Paul' could go to node A_3, and then to node 'Rick'. This walk implicitly defines the neighborhood based on attribute co-occurrence. Similarly, the walk from 'Paul' could have gone to 'r_5' and then to 'Apple', incorporating the row level relationships. Our approach is agnostic to the specific type of random walk used, with different choices yielding different embeddings. For example, one could design random walks that are biased towards other nodes belonging to the same tuple, or towards rare nodes. To better represent all nodes, we assign a "budget" of random walks to each of them and guarantee that all nodes will be the starting point of at least as many random walks as their budget. This ensures that even rare values are represented in the embeddings (thus handling challenge (2) in Section 3.3.1). After choosing the starting point T_i, the random walk is generated by choosing a neighboring RID of T_i, R_j. The next step in the random walk will then be chosen at random among all neighbors of node R_j, for example by moving on C_a. Then, a new neighbor of C_a will be chosen and the process will continue until the random walk has reached the target length.

We use uniform random walks in most of our experiments to guarantee good execution times on large datasets, while providing high quality results. We compare alternative random walks in the experiments.

Algorithm 2 GenerateRandomWalk

```
**Input**: starting node $n_j$, random walk length $l$
$r_j$ = findNeighboringRID($n_j$)
$W$ = seq($r_j$, $n_j$)
currentNode = $n_j$
**while** length($W$) $<$ $l$ **do**
  nextNode = findRandomNeighbor(currentNode)
  $W$.add(nextNode)
  currentNode = nextNode
**Output**: walk $W$
```

From Walks to Sentences. It is important to note that the path on the graph represented by a random walk does not necessarily reflect the sentence that will be inserted in the training corpus. For example, a possible random walk could be the following: $R_aT_bR_cT_dC_eT_fC_gT_h$, where T_*, R_*, C_* correspond to nodes of type tokens, record ids, and column ids, respectively. We note that the random walks include nodes corresponding to RIDs and CIDs. We noticed that the presence (or absence) of CIDs and RIDs in the sentences that build the training corpus has large effects on the data integration performance of the algorithm. Indeed, we observe that treating these as first order citizens, we can represent them as points in the vector space in the same way as any other token. For example, two nodes corresponding to different attributes might co-occur in many random walks, resulting in embeddings that are closer to each other: this may imply that these two attributes represent similar information. A similar phenomenon could also be obtained for tuple embeddings. A number of prior approaches such as DeepER [70] or DeepMatcher [71] only learn embeddings for tokens and then obtain embeddings for tuples by averaging them or combining by using a RNN. The use of our random walks as sentences provides additional information about the neighborhood of each node, which would not be so easily obtained by using only the structured data format.

3.4.3 Embedding Construction

The generated sentences are then pooled together to build a corpus that is used to train the embeddings algorithm. Our approach is agnostic to the actual word embedding algorithm used. We piggyback on the plethora of effective embeddings algorithms such as Word2Vec, GloVe, fastText, and so on. Every year, improved embedding training algorithms are released, and this has a transitive effect on our approach. Broadly, these techniques can be categorized as word-based (such as Word2Vec) or character-based (such as fastText). We discuss the hyperparameters for embedding algorithms such as learning method (either CBOW or Skip-Gram), dimensionality of the embeddings, and size of context window in Section 3.5.

3.4.4 Algorithm So Far

Algorithm 3 provides the pseudocode for learning the local and relational embeddings based on our discussion.

Algorithm 3 Meta Algorithm for EmbDI

```
Input: relational datasets $D$, number of random walks $n_{walks}$, number of nodes $n_{nodes}$

$W$ = [ ]
$G$ = GenerateTripartiteGraph($D$)
for all $n_j \in nodes(G)$ do
    for i = 1 to $(n_{walks}/n_{nodes})$ do
        $w_i$ = GenerateRandomWalk($n_j$)
        $W$.add($w_i$)
$E$ = GenerateEmbeddings($W$)
Output: Local relational embeddings $E$
```

3.5 Experiments

In this section we demonstrate that our proposed embeddings learn the major relationships inherent in structured data (Section 3.5.3). We then analyze the contributions of our design choices in Section 4.4.4. We will show the positive impact of our embeddings for multiple data integration tasks in supervised and unsupervised settings in Chapter 4.

3.5.1 Datasets

Name (shorthand)	# tuples	# columns	# distinct values	# matches	# sentences	% overlap
IMDB-Movielens (IM)	49875	15	118779	4115	2810900	8.79
Amazon-Google (AG)	4589	3	5390	1166	166316	6.01
Walmart-Amazon (WA)	24628	5	45454	961	1168033	3.10
Itunes-Amazon (IA)	62830	8	53079	131	1931816	5.84
Fodors-Zagats (FZ)	864	6	3282	109	69100	9.08
DBLP-ACM (DA)	4910	7	6555	2223	191083	62.33
DBLP-Scholar (DS)	66879	4	131099	5346	3299633	2.33
BeerAdvo-RateBeer (BB)	7345	4	11260	67	310083	10.18
Million Songs Dataset (MSD)	1000000	5	870841	1292023	31180683	n.a.

Table 3.1: EmbDI dataset properties.

We used 8 datasets from the literature [70, 71, 185, 186] and a dataset with a larger schema (IM) that we created starting from open data (`https://www.imdb.com/interfaces/`, `https://grouplens.org/datasets/movielens/`). Details for the scenarios are in Table 3.1. For the majority of the scenarios, less than 10% of the distinct data values are overlapping across the two datasets. MSD is a dataset with only one table.

3.5.2 Generating the Embeddings

Pre-trained Embeddings. In the following, *pre-trained* word embeddings have been obtained from fastText [187]. We tested also GloVe [188] and obtained comparable quality results. We relied on state of the art methods to combine words in tuples and to obtain embeddings for words that are not in the pre-trained vocabulary [70, 153]. With fastText [67], we create tuple embeddings by concatenating the embeddings of each tuple entry. We observed experimentally that this approach yields the best results. We rely on built-in fastText functions to handle out-of-vocabulary entries.

Embedding Generation Algorithms. We test four algorithms for the generation of local embeddings from relational dataset. All local methods make use of our tripartite graph and exploit record and column IDs in the integration tasks.

The first method is Basic, which creates embeddings from permutations of row tokens and sentences with samples of attribute tokens. As the method is aware of the structure of the database, it can learn representation for tuples and attributes. We fixed the size of the sentence corpus for Basic to contain the same number of tokens in EmbDI's corpus.

The second method is Node2Vec [42], a widely used algorithm for learning node representation on graphs. Given our graph as input, it learns vectors for all nodes. We used the implementation from the paper with default parameters.

The third method is Harp [178], a state of the art algorithm that learns embeddings for graph nodes by preserving higher-order structural features. This method represents general meta-strategies that build on top of existing neural algorithms to improve performance. We used the implementation from the paper with default parameters.

The fourth method is the one presented in Section 3.4, we refer to it as EmbDI in the following (`https://gitlab.eurecom.fr/cappuzzo/embdi`). The default configuration uses our tripartite graph, walks (sentences) of size 60, 300 dimensions for the embeddings space, the Skip-Gram model in word2vec with a window size of 3, and different tokenization strategies to convert cell values in nodes. We report the numbers of generated sentences for each dataset in Table 3.1. The number of sentences depends on the desired number of tokens in the corpus, we discuss a rule-of-thumb to obtain reasonable sizes in the ablation analysis.

By default, EmbDI uses optimizations in data integration tasks. However, to be fair to pre-trained embeddings, our default configuration does not exploit external information, therefore the techniques in Sections 4.3.2, 4.3.3, and 4.3.4 are not used - we show their impact in the ablation study. Experiments have been conducted on a laptop with a CPU Intel i7-8550U, 8x1.8GHz cores and 32GB RAM.

	Basic				Node2Vec				Harp				EmbDI			
	MA	MR	MC	AVG	MA	MR	MC	AVG	MA	MR	MC	AVG	MA	MR	MC	AVG
BB	**.99**	.33	.32	.55	.97	**.66**	.92	**.85**	.96	.65	**.95**	**.85**	.92	.50	.77	.73
WA	.19	.27	.12	.19	mem	mem	mem	mem	.16	.32	.13	.20	**.94**	**1.00**	**.99**	**.98**
AG	**1.00**	**.42**	.10	.51	**1.00**	.39	**1.00**	**.80**	.99	.37	**1.00**	.79	**1.00**	.38	**1.00**	.79
FZ	.08	.30	.00	.13	.84	.88	.62	.78	.80	.86	.89	.85	**.94**	**.99**	**.94**	**.95**
IA	.09	.11	.09	.09	mem	mem	mem	mem	.81	.59	.96	.78	**.89**	**.85**	**.98**	**.90**
DA	.08	.29	.02	.13	**.79**	.77	.18	.58	.51	.74	.49	.58	**.79**	**.91**	**.66**	**.79**
DS	**1.00**	.58	.69	.76	mem	mem	mem	mem	.12	.06	.06	.08	**.90**	**.99**	**.99**	**.96**
IM	**.99**	.34	.64	**.66**	mem	mem	mem	mem	.07	.29	.10	.16	.74	**.42**	**.78**	.65
MSD	.31	.37	.51	.39	mem	mem	mem	mem	t.o.	t.o.	t.o.	t.o.	**.60**	**.95**	**.83**	**.79**

Table 3.2: Quality results for local embeddings generation.

3.5.3 Evaluating Embeddings Quality

We introduce three kinds of tests to measure how well embeddings learn the relationships inherent in the relational data. Each test consists of a set of tokens taken from the dataset as input, while the goal is to identify which token does not belong to the set (function *doesnt_match* in Python library *gensim*). For the *MatchAttribute* (MA) tests, we randomly sample four values from an attribute and a fifth value from a different attribute at random in the same dataset, e.g., given (Rambo III, The matrix, E.T., A star is born, ***M. Douglas***), the test is passed if M. Douglas is identified. In *MatchRow* (MR), we pick all tokens from a row and replace one of them at random with a value from a different row, also selected at random from the same dataset, e.g., (S. Stallone, Rambo III, ***1952***, P. MacDonald). Finally, in *MatchConcept* (MC), we model more subtle relationships. We manually identify two attributes A_1 and A_2 that are in a one to many relationship. For a random token x in A_1, we identify all tuples T such that $(A_1 = x)$, we take three A_2 distinct values in T and we finally add a random value y (not in T) from A_2. The test is passed if y is identified as unrelated from the other tokens, e.g., (Q. Tarantino, Pulp fiction, Kill Bill, Jackie Brown, ***Titanic***). This test observes whether the relationship between co-occurring elements (such as directors and their movies) is stronger than the relationship between elements that belong to the same attribute. We took the union of the (aligned) datasets for each scenario and created between 1000 and 11000 tests, depending on each aligned dataset's size in terms of rows and attributes.

We report the quality results in Table 3.2, where each number represents the fraction of passed tests. With large datasets, some methods either failed the execution or have been stopped after a cut-off time of 10 hours. While on average the local embeddings generated by EmbDI are superior to all other methods, our solution is beaten in few cases. By increasing the percentage of row permutations in Basic, results improve for MR but decrease for MA, without significant benefit for MC. This shows that complex relationships are not modelled by row and attribute co-occurrence. Node2Vec fails on our configuration for the larger scenarios with memory errors (mem), while Harp has been stopped after 10 hours for MSD (t.o.). We do not report results for pre-trained embedding as they are not aware of the relationships in the dataset and perform very poorly for this task. For example, they obtain .33 on average for dataset BB (MA: .49, MR: .27, MC: .24) and 0.16 on average for dataset AG (MA: .03, MR: .22, MC: .22).

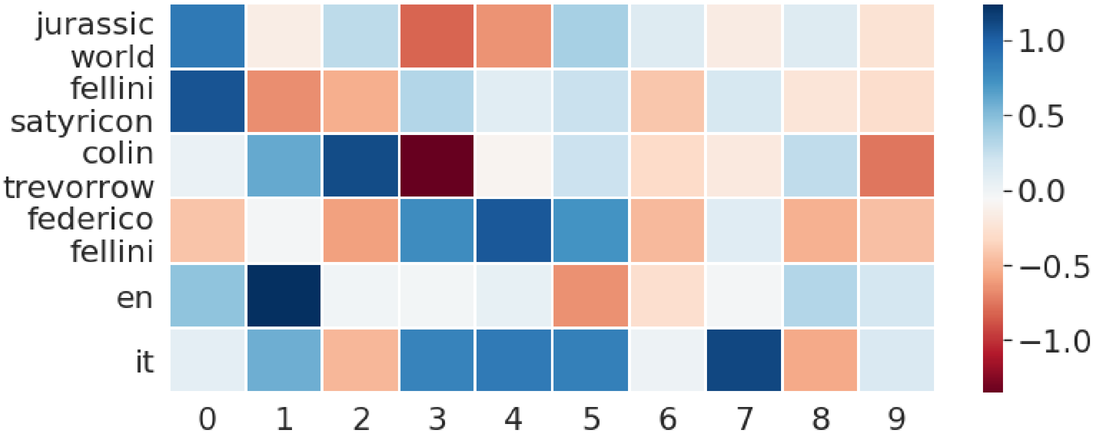


Figure 3.3: Heatmap of the vectors for different entities in the IMDB-Movielens dataset.

A more qualitative indication of how EmbDI embeddings can model information found in the dataset can be gained by projecting the embeddings of specific tuples down to a small number of dimensions using methods such as PCA or t-SNE [91]. One such example is depicted in Figure 3.3, where we plot the vectors of different entities from the IMDB-Movielens dataset as heatmaps: similar colors therefore represent similar positions in each dimension in the space. In this example, we choose a movie in the English language ("Jurassic World", with language "en") and a movie in the Italian language ("Fellini Satyricon", "it") along with their respective directors ("Colin Trevorrow" and "Federico Fellini"). From this example, there are a number of observations that can be made:

- Movies and directors are close to each other in dimension 2.
- All entities related to the Italian language share dimensions 3, 4 and 5.
- Token "en" is shared among an extremely large number of tuples in the original dataset, so most of its dimensions become "averaged out" among all other related tokens.
- On the other hand, "it" is shared among a comparatively smaller subset of movies, thus making it far more correlated to "Federico Fellini", who appears as director with a much higher relative frequency compared to directors that work in the English language.

Take-away: our graph preserves the structure of the dataset and EmbDI generates local embeddings that model column, row, and inter-tuple relationships better than other embedding generation methods.

3.6 Summary

In this chapter, we introduced the EmbDI system, a framework for building embeddings for tabular data. We discussed the modular pipeline of EmbDI and its components. We described the steps involved in the execution of the algorithm, starting from the construction of the graph and the operations required by it. We then concluded with some experimental observations made in the one-table scenario.

Chapter 4

Table Embeddings for Data Integration

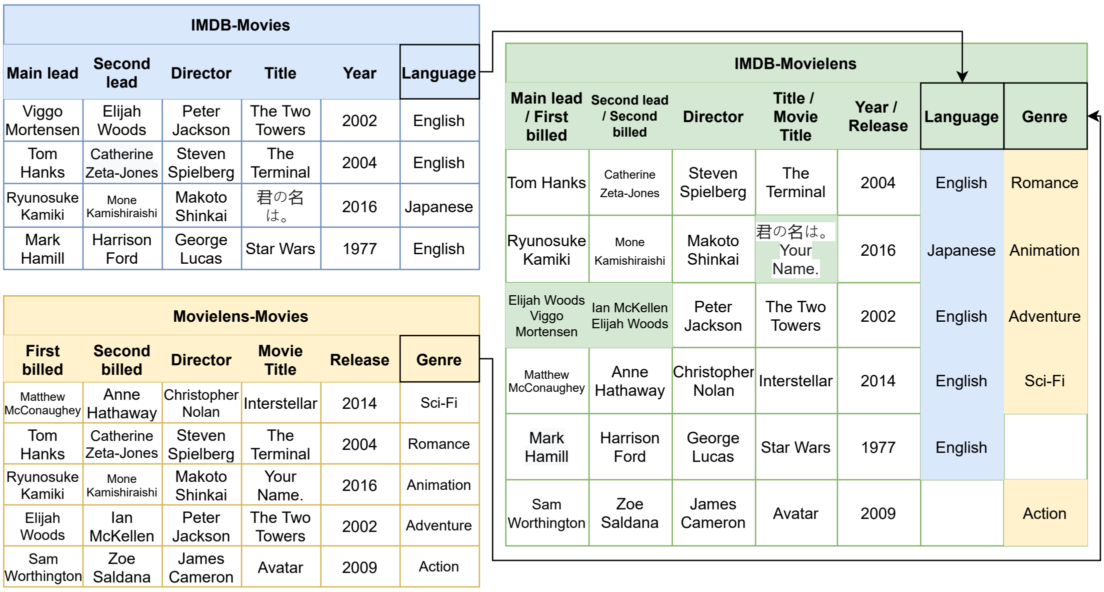
This Chapter is based on the second part of the paper "**Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks**", and focuses specifically on the Data Integration part of the contribution. In this Chapter, we discuss how to exploit the geometric properties of EmbDI-generated embeddings for performing schema matching and entity resolution.

4.1 Introduction

Data in an enterprise is often scattered across information silos. The problem of data integration concerns the combination of information from heterogeneous relational data sources [9]. It is a challenging first step before data analytics can be performed to extract value from data. Unfortunately, it is also an expensive task for humans [10]. An often cited statistic is that data scientists spend 80% of their time integrating and curating their data [6]. Due to its importance, the problem of data integration has been studied extensively by the database community. Traditional approaches require substantial effort from domain scientists to generate features and labeled data or domain specific rules [9]. There has been increasing interest in achieving accurate data integration with dramatically less human effort.

With data integration we describe the task of "integrating" different data sources by combining them in a new table view that contains information originating from the different sources. Figure 4.1 provides an example of this task, where we start from two separate relational tables in the same domain (movies) and attempt to produce a new, integrated table that contains all the information present in the starting two values in an easier-to-parse format. In this work, we concentrate our attention on the tasks of *Entity Resolution* (ER) and *Schema Matching* (SM).

We design EmbDI so that the system can perform these tasks by creating a combined graph from the concatenation of multiple tables. With this, attributes and tuples found in different tables can be correlated through their embeddings. As a result, it becomes



IMDB-Movies

Main lead	Second lead	Director	Title	Year	Language
Viggo Mortensen	Elijah Woods	Peter Jackson	The Two Towers	2002	English
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	English
Ryunosuke Kamiki	Mone Kamishiraishi	Makoto Shinkai	君の名は。	2016	Japanese
Mark Hamill	Harrison Ford	George Lucas	Star Wars	1977	English

Movielens-Movies

First billed	Second billed	Director	Movie Title	Release	Genre
Matthew McConaughey	Anne Hathaway	Christopher Nolan	Interstellar	2014	Sci-Fi
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	Romance
Ryunosuke Kamiki	Mone Kamishiraishi	Makoto Shinkai	Your Name.	2016	Animation
Elijah Woods	Ian McKellen	Peter Jackson	The Two Towers	2002	Adventure
Sam Worthington	Zoe Saldana	James Cameron	Avatar	2009	Action

IMDB-Movielens

Main lead / First billed	Second lead / Second billed	Director	Title / Movie Title	Year / Release	Language	Genre
Tom Hanks	Catherine Zeta-Jones	Steven Spielberg	The Terminal	2004	English	Romance
Ryunosuke Kamiki	Mone Kamishiraishi	Makoto Shinkai	君の名は。 Your Name.	2016	Japanese	Animation
Elijah Woods Viggo Mortensen	Ian McKellen Elijah Woods	Peter Jackson	The Two Towers	2002	English	Adventure
Matthew McConaughey	Anne Hathaway	Christopher Nolan	Interstellar	2014	English	Sci-Fi
Mark Hamill	Harrison Ford	George Lucas	Star Wars	1977	English	
Sam Worthington	Zoe Saldana	James Cameron	Avatar	2009		Action

Figure 4.1: Example of an ideal data integration system.
This is how an ideal data integration system could merge the two tables on the left to obtain the table on the right. Common attributes are combined, conflicts are merged (green background), attributes with no match are added to the table. Nulls are used when data is not available.

possible to leverage the geometric properties of the vectors to find neighbors for each tuple or attribute; these neighbors then become matching candidates. As both tuples and attributes have proper embeddings, the same method can be used to match either category.

4.1.1 Previous Work on Word Embeddings for Data Integration

Embeddings have been successfully used for data integration tasks such as entity resolution [70, 71, 152–155], schema matching [136, 137, 189], identification of related concepts [182], and data curation in general [190, 191]. Typically, these works fall into two dominant paradigms based on how they obtain word embeddings. The first reuses *pre-trained* word embeddings computed for a given task. The second builds *local* word embeddings that are specific to the dataset. These methods treat each tuple as a sentence by reusing the same techniques for learning word embeddings employed in natural language processing.

However, both approaches fall short in some circumstances. Enterprise datasets tend to contain custom vocabulary. For example, consider the small datasets reported in the left-hand side of Figure 4.2. The pre-trained embeddings do not capture the semantics expressed by these datasets and do not contain embeddings for the term "UUID20". Approaches that treat a tuple as a sentence miss a number of signals such as attribute boundaries, integrity constraints, and so on. Moreover, existing approaches do not consider the generation of embeddings from heterogeneous datasets, with different attributes and alternative value formats. These observations motivate the generation of *local* embeddings for the *relational* datasets at hand.

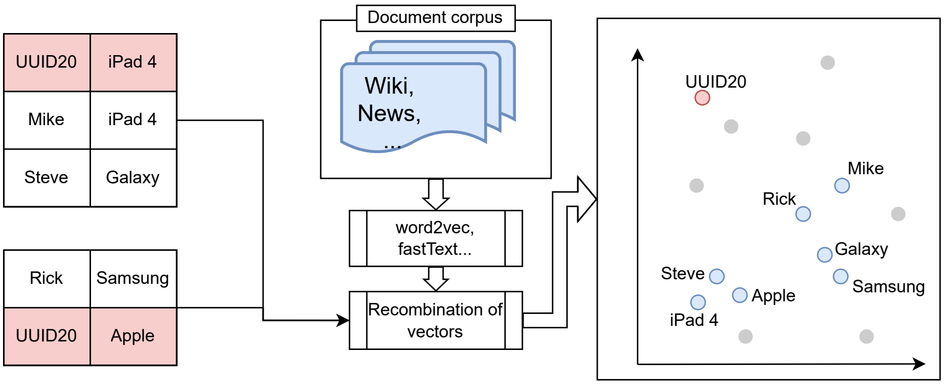


Figure 4.2: Illustration of a simplified vector space learned from text (prior approaches). Value "UUID20" appears correctly in both tables, but it is not present in the training corpus (it is out-of-vocabulary). Values present in the training corpus, but not found in the tables are represented as grey dots.

For the remainder of this chapter, we carry out data integration on relational tables and assume that we have generated suitable embeddings for the given tables according to what has been reported in chapter 3, with the specific optimizations required by the data integration task.

4.2 Using Embeddings for Integration

Once the embeddings are trained, they can be used for common data integration tasks. We now describe *unsupervised* algorithms that employ the embeddings produced by EmbDI to perform two tasks widely studied in data integration, Schema Matching and Entity Resolution.

The matching algorithms we use to perform SM and ER are very similar to each other, since both rely on the same notion of "bidirectional closeness" between nodes. Specifically, two nodes (either CIDs or RIDs) are matched if the first node is the closest to the second, and viceversa. In both algorithms, many elements (either RIDs or CIDs) in one dataset will not have a match that satisfies this condition in the other dataset. However, in datasets that involve many-to-many matches, this approach can lead to a loss in recall due to the fact that correct matches are discarded by the default choice of $n_{top} = 1$: by setting $n_{top} = 1$ to a larger value, the number of positive matches increases. On the other hand, this will also lead to a reduction in the precision, because of the larger pool of "viable" choices. Verifying the symmetry of the relationship has the advantage of increasing the precision by reducing the False Positive Rate, without penalizing the recall. The effect of n_{top} is described in Table 4.5.

The main difference between the procedures used for SM and ER lies in the blocking step that is performed before the execution of the matching operation: in both cases, values that are not involved in the task at hand (non-RIDs for ER, non-CIDs for SM) are removed from the model, thereby reducing the search space of each node's neighbors. While in the ER case, most points remain in the model, for SM the vast majority of embeddings are removed.

4.2.1 Schema Matching (SM)

Traditional approaches rely on grouping attributes based on the value distributions or use other similarity measures. Recently, [136] used embeddings to identify relationships between attributes using both syntactic and semantic similarities. However, they use embeddings only on attribute/relation names and do not consider the instances – i.e. values taken by the attribute.

Algorithm 4 describes the steps taken to perform schema matching between two attributes by exploiting their cosine distance in the vector space. The cosine distance is a widely used metric to measure the distance between points in high-dimensional spaces, and is defined as:

$$\textit{cosine similarity} = S_C(A,B) = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4.1}$$

Consider that, to prevent false positives in the column alignment, we terminate the algorithm after two iterations have been completed, even if some candidate pools may still contain values: if no matches are found for an attribute after two iterations have been completed, then the attribute is considered to be "alone" (e.g. attributes "Language" and "Genre" in Figure 4.1).

4.2.2 Entity Resolution (ER)

Recent works used pre-existing embeddings to represent tuples [70, 71]. In contrast, our approach relies on the use of RIDs as nodes in the heterogeneous graph. This allows EmbDI to learn better embeddings for the entire record from the data itself, rather than relying on combination methods such as averaging or concatenating the embeddings of the terms in the tuple. This information is then used to perform unsupervised ER by computing the distance between RIDs. We will also discuss in the experiments how one can piggyback on prior supervised approaches by passing the trained embeddings as features to [70, 71].

Algorithm 5 describes the steps taken to identify the matches in the Entity Resolution task. We assume that we are working in the Clean-Clean case, so that no matches for r_i are present in D_1.

If appropriate embeddings were learned for the RIDs, then this approach will produce good matches, which is indeed what we observe in our experiments.

Algorithm 4 Schema Matching

```
let $\mathcal{C}_1$ be the set of CIDs of dataset $D_1$ and $\mathcal{C}_2$ be the set of CIDs of dataset $D_2$
let $d(c_i)$ be the list of distances between column $c_i \in \mathcal{C}_1$ and all other columns $c_k \in \mathcal{C}_2$,
sorted in ascending order of distance (and viceversa).
let $\mathcal{T} = \mathcal{C}_1 \cup \mathcal{C}_1$ be the set of columns to be matched
while $\mathcal{T} \neq \emptyset$ do
  for all $c_k \in \mathcal{T}$ do
    if $d(c_k) \neq \emptyset$ then
      $c'_k$ = findClosest($d(c_k)$)
      $c''_k$ = findClosest($d(c'_k)$)
      if $c''_k == c_k$ then
        $c_k$ and $c'_k$ are matched
        remove $c_k$, $c'_k$ from $\mathcal{T}$
      else
        removeCandidate($d(c_k)$, $c'_k$)
        removeCandidate($d(c'_k)$, $c_k$)
    else
      remove $c_k$ from $\mathcal{T}$
```

4.2.3 Token Matching (TM)

We also consider the problem of matching tokens that are conceptual synonyms of each other, a task that is also known as *string matching* [192, 193]. For example, one relation could encode a language as "English" while other could encode it as "EN". Note that this is different from schema matching, where the objective is to identify attributes that represent the same information. Instead, we are interested in finding pairs of *tokens* from different relations that are related conceptually. Given two aligned attributes A_i and A_j, we seek to identify if two tokens $t_k \in Dom(A_i)$ and $t_l \in Dom(A_j)$ are related. Given the token t_k, we identify the set of top-n token ids that are closest to t_k. We announce that the first token $t_l \in Dom(A_j)$ that occurs in the ranked list is the conceptual synonym of t_k.

4.3 Improving Local Embeddings

In this section, we discuss a number of challenging issues that occur when applying EmbDI in practice.

4.3.1 Handling Imbalanced Relations

In a real-world scenario, there often are multiple relations and local embeddings must be learned for each of them. For a single relation, one can simply perform multiple random walks from each token node. This approach directly ameliorates the issue of infrequent words that plagues word embedding approaches, by guaranteeing that even rare words will appear frequently enough to be properly represented. A further complication arises

Algorithm 5 Entity Resolution

```
let $\mathcal{R}_1$ be the set of RIDs $\in D_1$
let $\mathcal{R}_2$ be the set of RIDs $\in D_2$
let $d(r_i)$ be the list of distances between RID $r_i \in \mathcal{R}_i$ and the closest $n_{top}$ RIDs $\in D_j$,
  with $i \neq j$.
for all $r_i \in D_1 \cup D_2$ do
    $d(r_i)$ = findClosest($r_i$, $n_{top}$)
for all $r_k \in D_1$ do
    $r'_k$ = findClosest($d(r_k)$)
    $r''_k$ = findClosest($d(r'_k)$)
    if $r''_k == r_k$ then
        $r_k$ and $r'_k$ are matched
        remove $r_k$ and $r'_k$ from $d(r_i)$ $\forall r_i \neq r_k, r'_k$
```

when one relation contains many more nodes than the other. If we perform an equal amount of random walks starting from each node, the signals from the larger dataset might overwhelm those coming from the smaller dataset. We found that an effective heuristic is to start random walks only from nodes that co-occur in both datasets. The tokens contained by random walks generated with this heuristic tend to be distributed more uniformly over the different datasets, even if there is a large difference in the size of the datasets. Furthermore, these nodes also happen to be the most informative ones as they connect two relationships and often quite useful for integrating these two relations. Even with datasets with a minimum amount of overlap (less than 2%), this approach ensures adequate coverage of all nodes and minimizes the issues due to relation imbalance.

The overlapping tokens are the bridge between the two datasets to be integrated. To maximize their impact in the embedding creation, one could start every sentence with a RID or CID, randomly picked from those connected to the token at hand. This small change in the random walk creation affects the results by creating evidence of similarity for the corresponding rows and columns.

Example. Assume that node token T_a appears in two rows R_a and R_b over two large datasets. Since the token is rare, it will appear most likely only once as the first node in the walk, therefore the embedding algorithm will only see it in few patterns, such as $T_aR_bT_c$ or $T_aC_dT_e$. To improve the modeling of the T_a we start the sentence with a RID or CID connected to T_a, such as $C_dT_aC_c$ and $R_aT_aR_b$. This way, even if the token is rare, it gives strong signals that the attributes and the row that contain it are related.

4.3.2 Handling Missing and Noisy Data

Many real-world datasets contain a large amount of missing data, so any effective approach for learning embeddings must have a cogent strategy for this scenario. The ideal approach employs imputation techniques to minimize the number of missing values. Unfortunately, this might not always be possible, since algorithms for imputation and data repair often do not provide good results in a relational setting. Prior approaches for learning relational embeddings skip missing values when computing embeddings. However, this approach is often counter-productive as missing data can be an indication of systemic error. Approaches where all missing values are treated as if they were the same entity (so one node for all nulls), or unique entities (individual nodes for each null) are not appropriate. The first approach creates a super node to store all NULL values, which has multiple negative effects on the result and produces no benefit. The second approach creates a unique node for each NULL: this does not cause any issues, but does not provide any additional information either. Moreover, if the number of NULLs is large, this approach increases the processing time without any commensurate benefit.

We propose a simple mechanism to use classical database techniques such as Skolemization [194] to handle missing data. Approaches for data repairs [195] are very accurate in identifying the errors, but struggle to identify the correct updated value [196, 197]. When there is no certain update to make, most methods put a *placeholder*, like a variable or the output of a function that is related to Skolemization. Our model is able to naturally consume and model these placeholders to obtain better embeddings. Hence, the data repairing task could be used to address both missing and noisy values.

Consider the scenario with two relations R_1 and R_2. Without loss of generality, let us assume that they both have attributes A_1, A_2, A_3, A_4. Suppose there are two tuples:

$$R1(a, N_1, c, N_2) \text{ and } R2(a, b, c', N_3)$$

Here N_1, N_2, N_3 denote the null values. If A_1 is the key attribute, we can derive three important updates in the data, including the creation of two placeholders, and rewrite the two tuples are follows:

$$R1(a, b, X_1, X_2) \text{ and } R2(a, b, X_1, X_2)$$

where X_1 models the conflict between c and c' and X_2 merges the two nulls. This reduces the heterogeneity of the data and improves the quality of the embeddings. Consider also that all occurrences of c and c' are merged in the graph, even in tuples that do not satisfy the pattern of this functional dependency. A single placeholder may end up merging a large number of token occurrences in the original dataset.

4.3.3 Incorporating External Information

Node Merging. Our graph representation allows one to incorporate external information such as wordnet or other domain specific dictionaries in a seamless manner. This is an *optional* step to improve the quality of embeddings. For example, consider two attributes from different relations – one stores country codes while the other contains complete country names. If some mapping between these two exists, then we can *merge* the nodes corresponding to, say, Netherlands and NL. The same reasoning applies to tuples (attributes): if trustable information about possible token matches is available, we merge different RIDs (CIDs) in the same node. Merging of nodes could be achieved by using external functions, such as matchers based on syntactic similarity, pre-trained embeddings, or clustering. This often increases the number of overlapping tokens across datasets and produces better embeddings for data integration.

Node Replacement in Random Walks. Merging of nodes is only viable if we are confident that the two tokens refer to the same underlying entity. In practice, the mapping between two entities is imperfect. For example, one could have a machine learning algorithm that says that tokens T_i and T_j are similar with confidence of 0.8. The extreme approaches of merging the two nodes (such as by applying a fixed threshold) or ignoring this strong information are both sub-optimal. We propose the use of a *replacement* strategy where, during the construction of the sentence corpus, token T_i is replaced by T_j (and vice versa) with a probability proportionate to their closeness. Note that this only affects the sentence construction. The random walk by itself is not affected. Specifically, if the random walk is at node T_i, it might output T_j in the sentence instead of T_i. However, when choosing the next node, it will only pick the neighbors of node T_i.

Handling Numeric Data. Integer and real-valued attributes are very common in relational data. A straightforward approach is to treat them as strings, so that each distinct value is assigned to a node in the graph. However, this simplistic approach does not always work well, as it ignores geometric relationships between numbers such as the Euclidean distance. One way to use this distance information is to *replace* two numbers if they are within a threshold distance. Unfortunately, identifying an effective threshold is quite challenging in general. Consider two set of tokens $\{1, 2, 3, \ldots, \}$ and $\{1, 1.00001, 1.00002, \ldots, 2\}$. In the former, we can plausibly replace 1 with 2 while it would not be appropriate in the latter scenario. We apply an effective heuristic that combines node replacement with data distribution-aware distance between two numbers. Typically, most numeric attributes can be approximated by a small number of distributions, such as Gaussian or Zipfian. As an example, if a particular attribute is Gaussian, we can efficiently estimate its parameters – mean and variance. Then, given a number i, we generate a random number r around i in accordance with the learned parameters. If the new random number is part of the domain of the attribute, then we replace i with r.

4.3.4 Embedding Alignment

Typically, embeddings for multiple relations are trained using two extreme approaches – either by training embeddings one relation at a time or by pooling all the relations and training a common space. The individual approach is more scalable, but misses out on patterns that could be inferred by pooling the data. The pooled approach must ensure that signals from larger relations do not overpower those from smaller ones. We advocate for a novel embedding alignment approach, adapted from multilingual translation [198].

We begin by training embeddings each relation individually. This may cause RID and CID vectors that represent different instances of the same entity to differ from each other when the datasets share a small number of common tokens. To mitigate this problem, we *align* the embeddings of the values contained by the two datasets that were trained in the initial execution by pivoting on the new information, basically changing the vector space that represents one dataset to better match the vector space of the other. This allows us to better materialize relationships between tokens, even if they do not co-occur in a single relation. Furthermore, this approach ensures that the geometric relationships between tokens within each individual dataset are retained.

Algorithm 6 AlignEmbeddings

```
Input: relations $\mathbb{R}_1$, $\mathbb{R}_2$, $\mathbb{E} = \textsc{EmbDI}\ (\text{concat}(\mathbb{R}_1,\ \mathbb{R}_2))$
let $U_i$ be the set of unique words in $\mathbb{R}_i\ \forall i \in 1, 2$
let $\mathcal{A} = U_1 \cap U_2$
$A\ =\ \mathbb{E}(w_i)\ \forall\ w_i \in \mathbb{R}_1$
$B\ =\ \mathbb{E}(w_j)\ \forall\ w_j \in \mathbb{R}_2$
$W^*\ =\ \text{argmin}_{W,\mathcal{A}}(WA - B)$
$A' = W^*A$
for all $w_i \in \mathbb{R}_1 \cup \mathbb{R}_2$ do
    if $w_i \in \mathbb{R}_1 \cap \mathbb{R}_2$ then
        $\mathbb{E}'(w_i)\ =\ \text{average}(A'(w_i),\ B(w_i))$
    else if $w_i \in \mathbb{R}_1$ then
        $\mathbb{E}'(w_i)\ =\ A'(w_i)$
    else
        $\mathbb{E}'(w_i)\ =\ B(w_i)$
Output: Aligned embeddings $\mathbb{E}'$
```

Assume that we have two relations $\mathbb{R}_1$ and $\mathbb{R}_2$ with adequate overlap, and that A and B represent the embeddings of words in $\mathbb{R}_1$ and $\mathbb{R}_2$, respectively. It is possible to formulate an orthogonal Procrustes problem [198] by seeking a translation matrix $W^*\ =\ \text{argmin}_{W,\mathcal{A}}(WA - B)$, with $\mathcal{A} = U_1 \cap U_2$ being the intersection of unique values (the *anchors*) in common between the two starting relations. Applying the translation matrix W^* to A yields a translated matrix A', which minimizes the distance between anchor points. To employ this technique in the ER and SM tasks, we use matching CIDs and RIDs in the original embeddings as anchors to perform the rotation. We then match again on the rotated embeddings. Algorithm 6 describes the embedding alignment.

4.3.5 Handling Multi-Word Tokens

Multi-word tokens (such as “Adobe Photoshop CS3”) are common in relational dataset. There are two key problems: how to tokenize a multi-world cell and how to aggregate the token embeddings to get the cell embeddings. There are no simple answers to this problem. Indeed, there are a number of ways in which multi-word cells could be tokenized: one simple option is to treat the entire word sequence as a single token; the other option is to tokenize the word sequence, compute the word embeddings for each of the tokens, and then aggregate these token embeddings to get the embedding for the multi-word cell. In some cases, these multi-word tokens contain substrings that would yield additional information if they were represented as stand-alone nodes (in the example above, “Adobe” and “Photoshop” are possible candidates). Unfortunately, in the general case it is very hard to pinpoint cases where performing the expansion would improve the results; consider a counterexample such as “Saving Private Ryan”: in this case, we would rather have a single node to represent the movie title as it likely is a “primary key” in the dataset and as such would help when performing integration tasks.

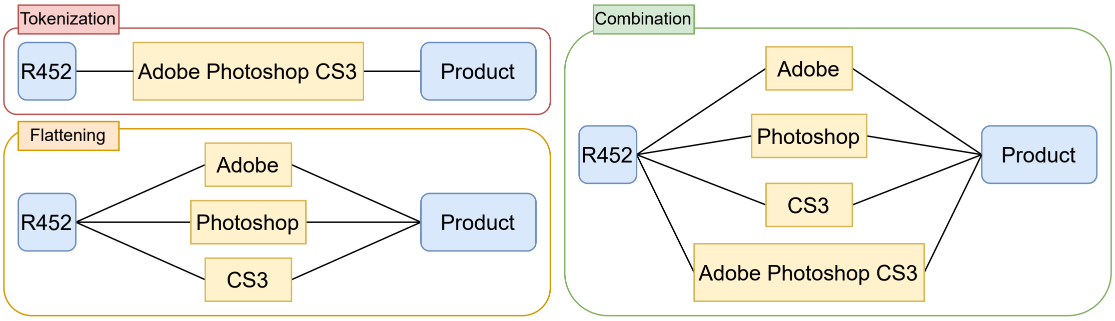


Figure 4.3: Different tokenization results for the string “Adobe Photoshop CS3”.

We tested multiple solutions to mitigate these issues, depicted in Figure 4.3. *Tokenization* describes the case in which the content of a cell is kept where as is, regardless of how many sub-tokens are present in the string. *Flattening* is the opposite solution, in which each token in a multi-word token is assigned a node in the graph. Finally, the *combination* of the two works by creating a node for the full string (“Adobe Photoshop CS3”) and one node for each of the substrings. This solution increases the size of the graph, but as we will show in the experimental section, it leads to major improvements in the results. An additional solution we employed specifically for our applications in Data Integration consists in a simple yet effective heuristic that allows us to handle both multi-word tokens and rare tokens at the same time. Instead of representing all unique values in both datasets in the same way, we make a distinction between nodes that are present in both datasets and those that appear only in one dataset. Then, we tokenize the shared tokens and expand those that are not in common. This effectively allows us to extract the information present within multi-word tokens and, possibly, introduce connections that would be missed otherwise. Moreover, representing the common values as unique tokens introduces “bridges” between the datasets, which can be exploited during the step of random walks generation to introduce semantic connections that would not be identified otherwise. Proper handling of tokens mitigates challenge (3) in Section 3.3.1.

4.4 Experimental Results

We test schema matching and entity resolution in every integration scenario with two datasets and report preliminary results on token matching. In the following, we measure the quality of the results w.r.t. hand crafted ground truth for each task with precision, recall, and their combination (F-measure). Execution times are reported in seconds.

4.4.1 Schema Matching

We test an unsupervised setting using Algorithm 4.2.1 with overlap of columns treated as bag-of-words (Base) and with local embeddings. We also report for an existing system with both pre-trained embeddings (Seep$_P$), as in the original paper [136], and EmbDI local embeddings (Seep$_L$), as they are the ones with competitive performance that we

	Unsupervised					
	Base	EmbDI	Node2Vec	Harp	SEEP$_P$	SEEP$_L$
BB	**1.00**	**1.00**	**1.00**	**1.00**	.75	.75
WA	**1.00**	**1.00**	mem	.60	.60	.80
AG	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
FZ	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
IA	**1.00**	**1.00**	mem	**1.00**	.50	.75
DA	**1.00**	**1.00**	mem	.50	.75	.81
DS	**1.00**	.50	mem	**1.00**	.60	.73
IM	.60	**.78**	mem	**.78**	.68	.75

Table 4.1: F-Measure results for Schema Matching (SM).

could generate in all cases.

Table 4.1 reports the results w.r.t. manually defined attribute matches. All methods are unsupervised, but we distinguish two groups. In the first group, local embeddings are generated and then used with Algorithm 4.2.1 from Section 4.2. Basic local embeddings lead to 0 attribute matches in this experiment and we do not report them in the table. While EmbDI embeddings lead to the best results in most cases, for DS Harp gets better results. While we can get comparable results with optimizations (Section 4.3), this shows that our graph enables other, more complex embedding schemes to get good results. Base performs very well across most datasets and it is outperformed by local embeddings in one case.

In the second group, we compare pre-trained and EmbDI embeddings with an existing matching system. We have two main remarks. First, the simple unsupervised method with EmbDI embeddings outperforms by at least an absolute 10% the Seep$_P$ baseline in terms of F-measure in all scenarios. Second, the baseline method improves by an average absolute 6% in F-measure when it is executed with EmbDI embeddings, showing their superior quality for SM w.r.t. pre-trained ones.

We observe that results for Seep$_P$ depend on the quality of the original attribute labels. If we replace the original (expressive and correct) labels with synthetic ones, Seep$_P$ obtains F-measure values between .30 and .38. Local embeddings from EmbDI do not depend on the presence of the attribute labels. Finally, we tested a traditional instance-based schema matcher that does not use embeddings [199, 200], whose results are lower than the ones obtained by EmbDI in all scenarios.

We employ the heatmap representation introduced in the previous chapter to display the similarities between matched columns in the vector space. The heatmap in Fig. 4.4 displays each entry on the vertical axis (`0_actor`, `1_year`, `0_director` ...), while the horizontal axis reports the value of the vector representations in each dimension. It is quite evident how attributes in different datasets have similar positions in the same dimensions if they have the same domain. It is also possible to see how some tokens (`0_actor`, `1_director`, `0_title`) share a very strong similarity in some dimensions when they share the same dataset.

Take-away: EmbDI local embeddings are more effective than pre-trained ones for the

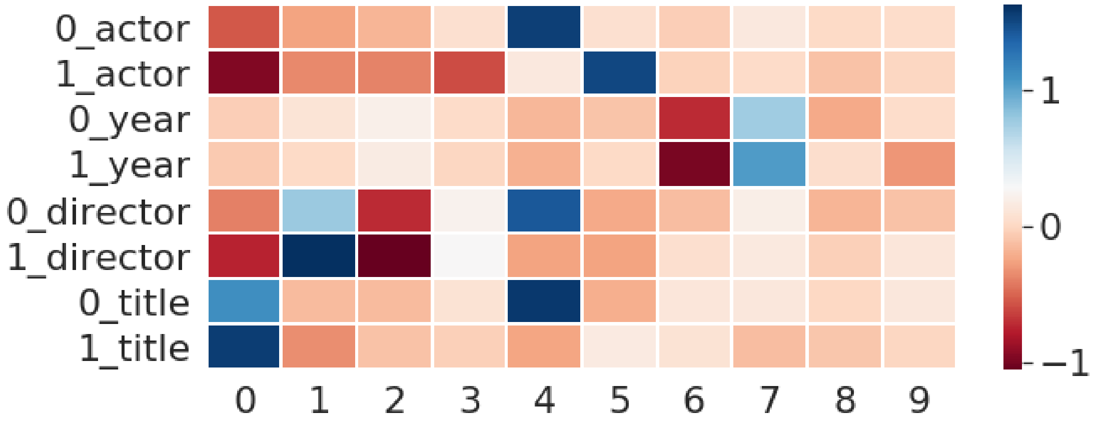


Figure 4.4: Heatmap of the vector representations of attributes in the IM dataset.

	Pre-trained	Local				
	FASTTEXT	EMBDI-S	EMBDI-F	EMBDI-O	NODE2VEC	HARP
BB	.59	.50	.82	**.86**	**.86**	**.86**
WA	.58	.59	.75	**.81**	mem	.78
AG	.18	.14	.57	.59	.70	**.71**
FZ	.99	.98	.99	.99	**1.00**	**1.00**
IA	.10	.09	.09	.11	mem	.14
DA	.72	.95	.94	.95	.87	**.97**
DS	.80	.85	.75	**.92**	mem	.81
IM	.31	.90	.64	.94	mem	**.95**

Table 4.2: Unsupervised Entity Resolution results comparing different baselines.

schema matching task when tested with two different unsupervised algorithms.

4.4.2 Entity Resolution

For ER, we study both unsupervised and supervised settings. To enable baselines to execute this scenario, we aligned the attributes with the ground truth. EMBDI can handle the original scenario where the schemas have not been aligned with a limited decrease in ER quality.

Figure 4.5 depicts two examples of movie titles that are in match. Similarly to previous examples in this form, it is possible to recognize some patterns that are shared between pairs in match (e.g. dimension 2, dimension 6), and that are not shared across different pairs. This is yet another demonstration of how information can be encoded by the embeddings.

As baseline for the *unsupervised* case, we use Algorithm 4.2.2 with pre-trained embeddings (FASTTEXT). We report for our integration algorithm with EMBDI embeddings in three variants of the way in which we tokenize the cell values in the dataset. EMBDI-S

	Supervised (5% labelled)		Task specific (5% labelled)	
	DeepER$_P$	DeepER$_L$	DeepER$_P$	DeepER$_L$
BB	0.51	0.53	0.54	0.58
WA	0.58	0.62	0.62	0.63
AG	0.53	0.56	0.58	0.62
FZ	**1.00**	**1.00**	**1.00**	**1.00**
IA	.76	.81	.77	**0.82**
DA	.84	.89	.86	.90
DS	.80	.87	.82	.91
IM	.82	.88	.84	.91

Table 4.3: Supervised Entity Resolution results comparing pre-trained (DeepER$_P$) and local (DeepER$_L$) embeddings.

DATASET	HEURISTIC			COMBINED WIN 3			COMBINED WIN 5		
	P	R	F	P	R	F	P	R	F
AG	71.51	49.6	58.57	81.40	63.77	71.52	**83.48**	67.07	**74.38**
BB	92.74	81.37	86.67	91.80	82.35	86.82	95.93	**86.76**	**91.12**
DA	98.55	91.73	95.02	**99.41**	**97.66**	**98.53**	**99.41**	**97.66**	**98.53**
DS	96.23	87.92	91.88	98.90	**97.13**	**98.01**	98.94	96.76	97.84
FZ	99.69	98.79	99.24	99.09	99.09	99.09	**100.00**	99.09	99.54
IM	95.91	91.18	93.65	98.98	96.65	97.80	99.01	**96.82**	**97.90**
IA	14.29	8.59	10.73	25.81	6.25	10.06	25.00	6.25	10.00
WA	**86.69**	77.23	81.69	82.89	76.17	79.39	86.27	**80.40**	**83.23**

Table 4.4: Updated ER results with combined graph structure.

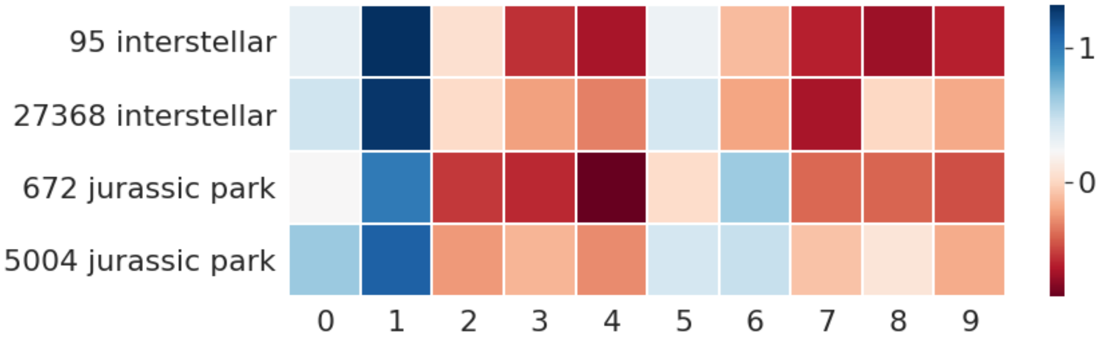


Figure 4.5: Heatmap of vectors in match for ER.

(Simple) uses the original value as a token node in the graph (e.g., “iPad 4th 2012”), while EmbDI-F (Flatten) models it as single words (e.g., nodes “iPad“, “4th”, “2012” connected to the same RID and to the same CID). The first strategy is more accurate in the modeling on tokens with more than one word as each token gets its own embedding; this is more precise than the one derived from combining the embeddings of the single words. However, a finer granularity is mandatory for heterogeneous datasets with long texts in the cell values for two reasons. First, accurate node merging is challenging with long sequences of words. Second, in different datasets the same entities can be split across attributes or grouped in one attribute. As an example, the BB datasets contain attributes “beer name” and “brewing company” but in one dataset oftentimes the name of the brewing company appears in the beer name `brewing_company_A beer_name_1`, while in the other dataset only `beer_name_1` appears in the name column. As we do not assume any user-defined pre-processing of the attribute values, modeling the words individually is beneficial in these cases. The third tokenization strategy, EmbDI-O (Overlap) is a trade off between the two that preserves as token nodes the cell values that are overlapping across the two datasets and models as single words the others.

We then tested the ER case with the *combination* strategy described in Section 4.3.5, and observed that the combination of the equivalent of EmbDI-S and EmbDI-F yields, in general, a major improvement over even the best results achieved in the previous cases. These results are shown in table 4.4.

We also test our local embeddings in the *supervised* setting with a state of the art ER system (DeepER$_L$), comparing its results to the ones obtained with pre-trained embeddings (DeepER$_P$).

Results in Table 4.2 for unsupervised settings show that EmbDI-O embeddings obtain the best quality results in three scenarios and second to the best in four cases. In every case, local embeddings obtained from our graph outperform pre-trained ones. For supervised settings, as in the SM experiments, using local embeddings instead of pre-trained ones increases the quality of an existing system. In this case, supervised DeepER shows an average 5% absolute improvement in F-measure with 5% of the ground truth passed as training data. The improvements decrease to 4% with more training data (10%). Similarly to SM, in the ER case local embeddings obtained with the Basic

n_{top}	P						R						F					
	AG	BB	DA	IA	IM	WA	AG	BB	DA	IA	IM	WA	AG	BB	DA	IA	IM	WA
1	**.803**	**.929**	**.991**	**.278**	**.973**	**.925**	.407	.765	.884	.039	.862	.634	.540	**.839**	.935	.068	.914	.752
5	.716	.885	.986	.132	.963	.853	.494	**.794**	**.917**	.055	.911	.748	.585	.837	**.950**	.077	**.936**	**.797**
10	.715	.885	.986	.137	.963	.841	**.496**	**.794**	**.917**	**.078**	**.912**	.757	**.586**	.837	**.950**	**.100**	**.936**	**.797**
100	.714	.885	.986	.125	.962	.834	**.496**	**.794**	**.917**	**.078**	**.912**	**.764**	.585	.837	**.950**	.096	**.936**	**.797**

Table 4.5: Effects of n_{top} on ER results.

method lead to 0 rows matched.

Finally, we investigated if our task agnostic embeddings can be *fine-tuned* for a specific task. This process of pre-training followed by fine-tuning is a common workflow in NLP. Specifically, we start with the relational embeddings learned by EmbDI and allow it to be fine-tuned for each individual tuple pair if it improves performance. We achieve this by modifying the embedding lookup layer of DeepER. By default, this layer does a "lookup" of a given token from the embedding dictionary. We allow DeepER to learn an additional weight matrix W such that the original EmbDI embeddings can be tuned for ER. Table 4.3 shows the results.

4.4.3 Token Matching

Differently from the previous experiments, we do not claim an unsupervised solution for this integration task. In fact, we argue that our embeddings should be used as an additional signal to be combined with the other similarity measures used for this task, e.g., edit distance, Jaccard, TF/IDF. We evaluated the accuracy of this approach on the IM scenario in matching of tokens across the two datasets in two (aligned) pairs of columns. We picked this dataset and these columns as it was possible to manually craft the ground truth for their matches. Two columns in a pair have the information about the same entities, but expressed in different formats, such as "DK" for "Denmark", "UK" for "Great Britain", and so on. We used the unsupervised matching based on nearest neighbor also used for ER.

For the column expressing information about countries, pre-trained embeddings and Jaccard similarity obtain matches with 0.13 and 0.19 F-measure, respectively, while EmbDI embeddings get 0.31. For the column about languages, the baselines obtain 0.17 and .20, while EmbDI obtains 0.30. These results suggest that local embeddings can bring a stronger signal than pre-trained embeddings and Jaccard distance in string matching systems.

4.4.4 Ablation Analysis

We now show the effect of the different parameters, design choices, and optimizations in our framework.

Parameters. Several parameters in EmbDI affect the quality of the local embeddings. All the results reported above have been obtained using a single configuration, but the quality of the results for the different tasks increases significantly by tuning the parameters for the specific tasks.

The default setting uses walks of length 60, 300 dimensions for the embeddings space, and the Skip-Gram model in Word2Vec with a window size of 3. We noticed that CBOW performs better than Skip-Gram on the ER task, while having worse results in the EQ and SM. For example, executing the ER task with CBOW increases F-measure by at least 2 absolute points for IM and DS. Similarly, decreasing the size of the walks to 5 for the SM task raises the F-measure for DS to 1. This is because embeddings from shorter walks better model the value overlap across columns. As this signal drives the matching task, a lower value increases the quality of the SM matches, but reduces the quality for EQ and ER. We also observe that an even lower value (3) decreases the results also for SM, demonstrating that a semantic characterization in also needed. The window size has proven to be an important parameter, with a large effect on the final performance of the algorithm. Its effect is highly dependent on the chosen tokenization method: while window size 3 achieves better results with the simpler methods (*tokenize*, *flatten* and *heuristic*), *combination* strongly benefits from a larger window size (5). Reducing the number of dimensions has limited, mixed effects on average, thus showing that our method is robust to this parameter.

A larger corpus leads to better results in general, but we empirically observed diminishing returns after a certain size. As a rule of thumb, we fix the total number of tokens n_{tokens} in the training corpus with the following formula:

$$n_{tokens} = (n_{dist} + n_{rows}) \times 1000 \tag{4.2}$$

where n_{dist} is the number of value nodes in the graph after taking into account the tokenization method explained in Section 4.3.5 and n_{rows} is the total number of tuples in the table. The number of walks is derived by dividing the number of tokens by the walk length. For example, the IM dataset contains 49875 tuples and 118779 distinct values. If we use sentences with length 60, then the resulting number of sentences will be $(118779 + 49875) \times 1000 \div 60 = 2810900$, which is the value reported in Table 3.1. While effective, this rule of thumb can be "overly generous" and generate far more tokens than required if the starting graph is quite large: experimentally, we observed that a number of tokens ranging from 1.5M to 2.5M is enough for all datasets we have worked with, with the execption of MSD.

We set $n_{top} = 10$ in our ER experiments; by varying n_{top} we observe the expected trade-offs between P and R, as reported in Table 4.5 for six datasets. Results for the FZ scenario do not change with different n_{top} values and results for DS are close in values and trend to those reported for DA.

Optimizations. We tested optimizations of the original default configuration for EmbDI. For replacement (Section 4.3.3), we used an external dictionary for one column in each dataset, e.g., different formats of country codes. The biggest improvement is in ER with an absolute 3% on average, while the quality is stable for SM and EQ. For alignment (Section 4.3.4), we fed the optimization step with the outcome of the default model, i.e., we got candidate RIDs and CIDs from a first execution and then refined the embeddings with this information. This leads to a an absolute 2% increase in F-measure for ER, with the higher contribution coming from the better recall.

Figure 4.6 shows the impact on ER of inserted missing values in the IM dataset. We defined the FD *Title,Director* → *Year* and inserted increasing amount of noise at random in the column Year. As the number of records in common across the two datasets is very low, most of the NULLs are modifying records that are only in one dataset. Surprisingly, this has a visible effects on the results in terms of F-measure. While the default Skip solution (ignore NULL values in the graph creation) stays stable until a large number of NULLs is introduced, the results improve for the optimization that enforces the FD in the graph construction. This improvement is driven by the increasing precision. In fact, there are non duplicate movies that have a large number of attribute values in common, including the year, and that are identified as duplicates by our unsupervised method (based on neighbor RIDs). However, the FD enforces that any missing value is treated as a new value, distinct from the others, and this information moves the embedding of the RID with the NULL away from the similar tuple that is not a duplicate.

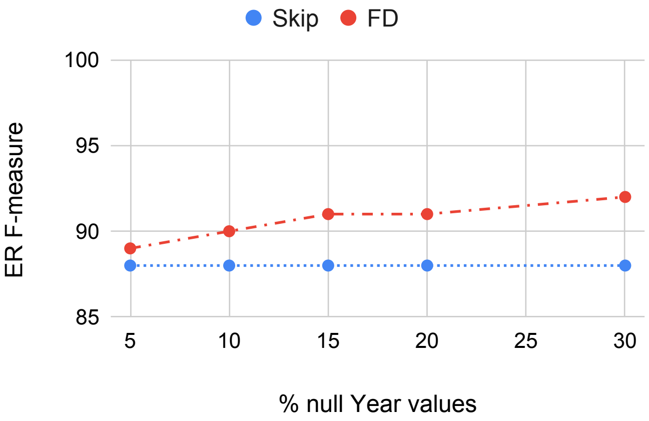


Figure 4.6: EmbDI ER F-measure for IM with increasing amount of missing values in the data.

Dataset	Graph	Walks	Embeddings	Walks+Embeddings	N2V	HARP
BB	2.47	66.7	133	200	1663	732
WA	13.4	329	1113	1442	mem	2394
AG	1.19	34.4	122	156	953	135
FZ	0.3	12.0	40.7	52.6	178	27.0
IA	32.0	533	1360	1893	mem	9122
DA	2.08	43.6	130	173	920	128
DS	33.9	919	3027	3947	mem	21659
IM	31.6	768	2772	3540	mem	8001
MSD	146	6377	27050	33427	mem	t.o.

Table 4.6: Execution times (in seconds) for embeddings generation for EmbDI, Node2Vec (N2V) and Harp.

Execution times. As the graph generation is common to all methods, we compare our solution with Node2Vec (N2V) and Harp in terms of time to generate walks and produce embeddings (W+E). EmbDI is faster in most cases, up to 7x in two datasets, and, in contrast with Harp, never hits the time out (t.o.) of 10 hours. With larger datasets, Node2Vec raised a memory error on our 32GB reference machine. EmbDI does not suffer from this problem, even in a laptop with 16GB of main memory, we have been able to run all tests, including the ones for the biggest dataset of 1M tuples

(139MB).

4.5 Summary

In this chapter, we moved to the two-tables scenario and introduced the data integration part of EmbDI. We described the algorithms employed by EmbDI to perform the Schema Matching and Entity Resolution tasks. In our experimental campaign, we compared EmbDI against different baselines in the unsupervised case and as local pre-trained features for a supervised algorithm. We showed that EmbDI has comparable performance to SOA graph embeddings algorithms, and outperforms them in some cases; in the supervised case, local embeddings produced by EmbDI always outperform pre-trained embeddings trained on external corpora such as fastText. In the ablation analysis, we discussed how different parameters influence the imputation performance of EmbDI. We then shifted our attention to the study of the EmbDI embeddings, and how to employ them to perform token matching and querying over embeddings.

Chapter 5

Relational Data Imputation with GNNs and Multi-task Learning

So far, we have developed a set of solutions the data integration problem, introducing a novel approach for handling the data integration problem in an unsupervised, deep learning-based fashion. Spurred by the very encouraging results we achieved in data integration, we work on implementing the same tools and ideas to the data imputation problem, which will be the focus of this Chapter.

5.1 Introduction

We introduced data imputation in Chapter 2 as the task of pre-processing dirty data by fixing errors in cells, where an "error" could be a value that is tagged as incorrect by an error detection algorithm, or a missing value that should be filled.

The data imputation problem is an important step in any pipeline since missing or erroneous values can be harmful when the data is used for training models [158, 159]. There are two challenges involved in the development of systems for data imputation that we focus on:

1. **The need for supervision.** Data imputation methods show good performance when supervision is available. Unfortunately, supervision is costly and not always accessible.

2. **Deep Learning-based systems cannot naturally model tabular data.** Tabular data is structured, discrete, and can contain mixed-type data. Deep Learning-based systems work best in presence of streamlined, homogeneous data to better translate it in numerical form. Even when transformed into numerical features via some kind of encoding, categorical values do not have the same distribution as "real" numerical values, which causes issues when training DL-based methods on features with mixed origin.

In this chapter, we introduce two novel algorithms for tackling these challenges, and then we carry out an experimental campaign with the objective of understanding how

ML-based imputation methods learn to provide some recommendations on how to use them. In the following, we consider datasets in which the erroneous values have already been identified and marked as “nulls”: we do not, therefore, consider the problem of Error Detection and instead focus on the problem of filling the vacancies present in the datasets at hand. For the sake of brevity, we refer to a dataset that contains missing values as “dirty dataset”.

The major contributions we introduce in this Chapter are GRIMP and FunForest, two data imputation algorithms for mixed type (categorical and numerical) tabular data. Both systems are designed to work in high-error rate environments and can easily handle multiple missing values in a tuple.

Our first contribution is GRIMP (**G**raph embeddings for **R**elational data **IMP**utation), a Data Imputation algorithm that combines a graph representation of tabular data, Graph Neural Networks and an self-supervised multi-task classification objective to perform imputation of mixed relational data. With GRIMP, we address the challenges listed above as follows:

1. GRIMP is self-supervised, and does not require labeled or fully-clean tuples. GRIMP is robust to noise and high error fractions.

2. GRIMP is designed to transparently handle mixed (categorical and numerical) data types by generating an encoded vector representation of the dataset values where categorical and numerical values are treated equally.

3. GRIMP introduces a first attempt at combining external information modeled as functional dependencies with GNNs by piloting the training objective of columns involved in FDs.

Similarly to EmbDI, in GRIMP the table is transformed into a graph; for GRIMP, however, we generate the node embeddings by employing GNNs in tandem with a classifier, rather than using the EmbDI’s word2vec-based architecture. The multitask classifier includes an attention layer to combine attribute vectors and value vectors. This layer can be bootstrapped by using external information such as Functional Dependencies.

During the development of the GRIMP architecture to handle Functional Dependencies (FDs), we considered the problem of introducing this information in alternative, pre-existing data imputation algorithms. As a result, we develop an enhanced version of one the most popular data imputation algorithms, MissForest [45], and name it FunForest (**FUN**ctional Miss**FOREST**). FunForest works by exploiting external information under the form of Functional Dependencies. This is done by allocating a fraction of the random forest estimator budget to a subset of attributes, so that a part of the ensemble is being exposed exclusively to the attributes involved in FDs. Applying this constraint allows the model to focus more on attributes that are known to be (via the FDs) directly related to each other. As it is based on MissForest, which relies on random forests, FunForest is unsupervised and can handle mixed data types.

In the latter part of this chapter, we carry out an in-depth experimental campaign where we compare GRIMP and FunForest to a number of data imputation baselines, showing their performances and how different architectures tend to fall in common pitfalls and have very close results in practice. We try to give an explanation of why this happens, what DL-based models are doing in their training phase and how this issue can be tested and be accounted for.

5.2 GRIMP

GRIMP (**G**raph embeddings for **R**elational data **IMP**utation) is a data imputation system that employs a combination of deep learning techniques to impute missing data in relational tables. The architecture of GRIMP is based on two modules. First, a preprocessing module prepares the graph representation of the input table and the training corpus required by the second module. The second module consists of a multi-task architecture in which each attribute of the table is assigned a task-specific head; each head then carries out imputation by using either a multi-class classification or a regression objective, depending on whether the head is working with categorical or numerical values respectively.

Encouraged by the promising results achieved with EmbDI, we rely once more on a graph representation to represent the input dirty dataset. Indeed, we have shown in the previous chapters that graphs can be used to elegantly model tabular data and that such a representation can be employed with success to apply Deep Learning-based methods on tabular data. In this work, we leverage the advancements in the field of graph embedding through the application of Graph Neural Networks [43, 109], which allow us to combine known node embeddings with the graph's structure to produce an enhanced vector-based representation of nodes. In short, GRIMP takes as input a table, transforms it into a graph to feed it to a GNN, the node embeddings produced by the GNN are then passed as input to the final step in the training procedure.

This final step is the second major innovation in this work: the application of multi-task learning [116, 117] to the data imputation problem. We design and implement a multi-task architecture where each sub-task (defined as *head*) takes as input the vector representations of the graph nodes (that is, the cell values), then uses the information contained in the vectors for selecting the correct imputations. In the rest of this work, what we refer to as "heads" are task-specific sub-modules: unlike other works that feature attention mechanisms, our current implementation includes a singular attention module (rather than multiple "attention heads", such as BERT's [73]). Each head in the multi-task architecture models the imputation problem for a single table attribute, be it numeric or categorical: on one hand, this allows to carry out imputation in a single shot, rather than training the model iteratively and imputing over a single column at at time, on the other hand it allows to organically model categorical attributes, without the conflicts that might arise between true numerical variables and numerical encodings of categorical variables. A further advantage of the multi-task approach is how it allows to spread the classification operation over multiple sub-tasks, each of them working over a smaller domain than what a simpler architecture would require: this is how a naïve architecture that employs a GNN for classification could be implemented. GRIMP is highly robust to noise and can exploit all non-missing values in a dirty dataset by leveraging once again its multi-task architecture: thanks to the fact that each head is separated from all others, it becomes possible to feed the same information to different heads and obtain multiple, different and valid outputs from each of them. This advantage is explored further in Section 5.2.3.

GRIMP is self-supervised: given a dataset that features missing values, GRIMP produces an imputed version of said dataset with no labels or human supervision required by generating its own labels on the training data. While the algorithm itself is self-supervised, some preparation is needed before it can be executed: specifically, node embeddings must be generated on the target dirty dataset so that the graph nodes have features for the GNN to operate on. These features can be generated by any suitable method, which means that unsupervised algorithms such as EmbDI or simple sentence embeddings generated by fastText can be utilized to this end. As already mentioned, GRIMP does not require additional external information or labels to run other than the aforementioned embeddings. However, external information can be fed to GRIMP in two ways: by employing pre-trained embeddings that incorporate external information (e.g. fastText embeddings trained on Common Crawl), or by providing attribute-level information such as Functional Dependencies.

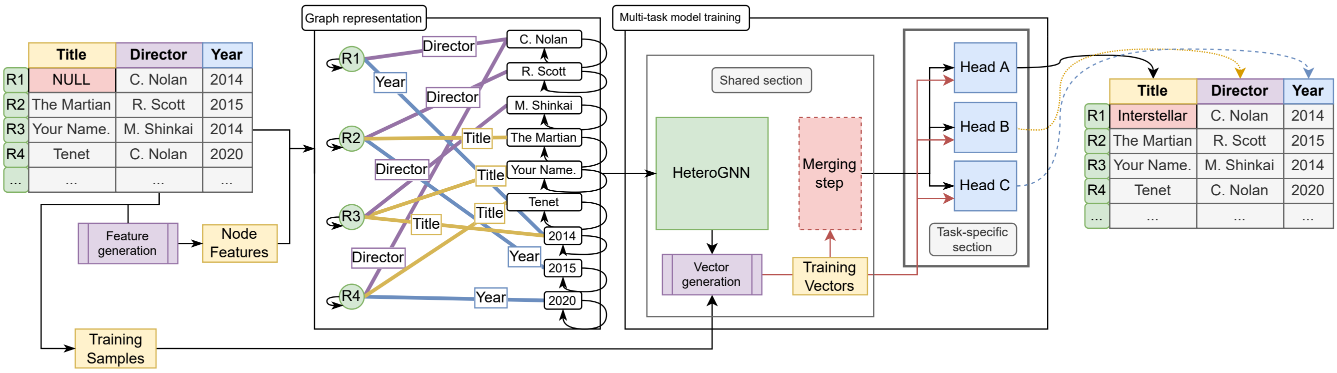
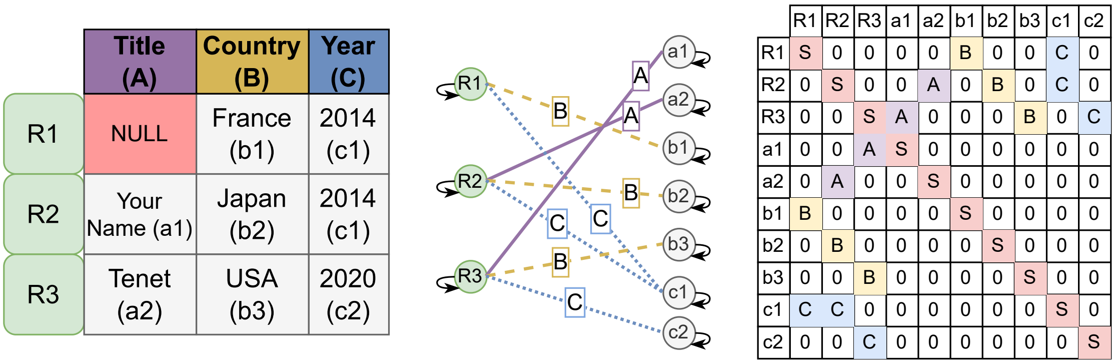


Figure 5.1: Overview of the GRIMP architecture.

The GRIMP pipeline is shown in Figure 5.1. Given a target table that contains missing values (in the example, a table on movies), two main operations are carried out: the training corpus (denoted as *training samples*) is generated, and the graph representation of the table is prepared. For the first operation (explained in more detail in Section 5.2.3), GRIMP creates a training sample for each non-null value in the dirty table: these training samples are then split by attribute among the various heads. For efficiency reasons, the training samples do not contain tuple values, instead the tuple contains the index assigned by the graph to each value: these indexes are used to build the vector representation of each tuple at run-time far more efficiently than a more naïve reconstruction operation based on look-ups would allow. In the graph generation (Section 5.2.1), a quasi-bipartite, heterogeneous graph encodes the table information by generating a node for each row and value, with typed edges linking the nodes on either side. We say the graph is quasi-bipartite because self-loops are added to all edges,



	Title (A)	Country (B)	Year (C)
R1	NULL	France (b1)	2014 (c1)
R2	Your Name (a1)	Japan (b2)	2014 (c1)
R3	Tenet (a2)	USA (b3)	2020 (c2)

	R1	R2	R3	a1	a2	b1	b2	b3	c1	c2
R1	S	0	0	0	0	B	0	0	C	0
R2	0	S	0	0	A	0	B	0	C	0
R3	0	0	S	A	0	0	0	B	0	C
a1	0	0	A	S	0	0	0	0	0	0
a2	0	A	0	0	S	0	0	0	0	0
b1	B	0	0	0	0	S	0	0	0	0
b2	0	B	0	0	0	0	S	0	0	0
b3	0	0	B	0	0	0	0	S	0	0
c1	C	C	0	0	0	0	0	0	S	0
c2	0	0	C	0	0	0	0	0	0	S

Figure 5.2: Example of GRIMP graph and adjacency matrix on a table. Nodes for record IDs and values are color-coded, edge types are color-coded as well and replicate the color of the attribute they refer to. The same attribute colors are used in the adjacency matrix, with "S" describing a self-loop.

following a common procedure in GNN literature [43, 87]. GNNs expect graph nodes to have features that encode information about them: since we are generating the graph from scratch starting from an unrelated data structure (that is, a relational table), an additional step for generating these features is required. Section 5.2.2 reports different methods for doing so.

The training step contains the shared section of the multi-task classifier, which comprises the GNN and a merging layer that combines the vector representations generated by the GNN. The structure of the shared layer and of the GNN in particular is explored in Section 5.2.4. Once the vector representations are ready, they are forwarded to the task-specific section of the pipeline. This section is composed of multiple task-specific sub-modules named *heads* (explored in Section 5.2.4), whose structure depends on their task: each task to corresponds to an attribute in the table, and the output of each head depends on the attribute datatype (categorical or numerical) and on its domain. If a head's attribute is categorical, then the head's output becomes a multi-class classifier with as many classes as there are distinct values in the attribute's domain; if the attribute is numerical, then the classifier has a single output for the specific value (number) that should be used for the given training sample. Each head measures its own training loss independently and according to the data type of its attribute: categorical attributes use cross entropy loss, while numerical attributes rely on RMSE. At the end, all losses are merged together.

We detail each component in the GRIMP pipeline in the following sections.

5.2.1 Building the Graph

Given a relational table that includes missing values, GRIMP builds a heterogeneous, bipartite graph which encodes both the table's content and its structure. Figure 5.2 shows an example table on the left, together with the corresponding graph and its adjacency matrix. In the graph, each record is assigned a RID node (highlighted in green), each unique value in the dataset is assigned a cell node (shown in grey), then RIDs and cell nodes are connected via a typed edge. The edge type is defined by the attribute the cell is found in, with the number of types being equal to the number of attributes in the table. Typed edges are color coded according to the color of the respective attribute: for example, record R1 is connected to value b1 via a yellow edge with type 'Country' ('B').

The adjacency matrix encodes the relative position of nodes in the graph and is a very important part of the pipeline: in fact, the graph adjacency matrix is how the GNN combines the structure of the graph with the node embeddings that carry information about each node.

Values that appear in multiple attributes are preemptively disambiguated so that each occurrence is connected exclusively to the attribute it is found in:

$$a_{ik} \neq a_{jl} \quad \forall a_{ik} \in A_i \ \wedge \ \forall a_{jl} \in A_j \ \Leftrightarrow \ i \neq j \tag{5.1}$$

The graph is constructed by iterating over the dirty dataset row by row: a new node is created for each tuple, and a new node is created for each unique value encountered in the tuple; for each attribute, a typed edge connects the tuple node to the value that is found in that attribute. If a cell contains a missing value, no additional edges are added to the graph and the empty cell is ignored at this stage. After the creation of the graph, the node features are assigned to each node by probing the pre-trained feature corpus. Algorithm 7 reports the graph generation procedure.

Algorithm 7 GenerateGraph

```
Input: table $\mathcal{D}$, node features $\mathcal{X}$
let $G$ = empty graph
for all $r_i$ in rows($\mathcal{D}$) do
  G.addNode($R_i$, $\mathcal{X}(R_i)$) {$R_i$ is the record id of $r_i$}
  for all value $v_k$ in $r_i$ do
    G.addNode($v_k$, $\mathcal{X}(v_k)$)
    G.addEdge($v_k$, $R_i$, $A_k$)
Output: graph $G$
```

Modeling Multi-Word Tokens and Numerical Values

The problem of multi-word tokens is still present here in the same way as it was in EmbDI (see Section 4.3.5). Differently from the data integration case, for the data imputation task we assume that each missing value should be replaced by *exactly the same* value as in the ground truth. Thus, we start from the assumption that the value to be imputed is present in the attribute domain *as-is*, and we should not expand the attribute domain to include all possible substrings: this is a common assumption for the data imputation task [35, 80, 165].

The final imputation operation is modeled as a classification task. By not splitting the content of a cell, the overall size of the domain can be kept low. This reduces the complexity of the task and avoids introducing noise with overlapping sub-strings that belong to different cells (e.g. common words such as “the”). This simplifies the training objective by not forcing the model to learn both the “combined” view of the cell value and all its sub-strings.

Numerical values are treated as strings and represented as nodes in the same way. To handle real numbers, we first round them to a set number of decimal places (we choose 8 places by default), then we treat them as strings and assign them a node. We considered the idea of binning similar numeric values based on the attribute’s distribution, but we observed experimentally that this solution did not bring improvements in the imputation performance of numerical values, and it introduced noise in the model by creating connections between tuples that were previously unrelated to each other. The combination of these two drawbacks led to a general worsening of the imputation performance. To avoid scale issues in the loss computations, numerical values are normalized before training the model, and then de-normalized before measuring the imputation accuracy.

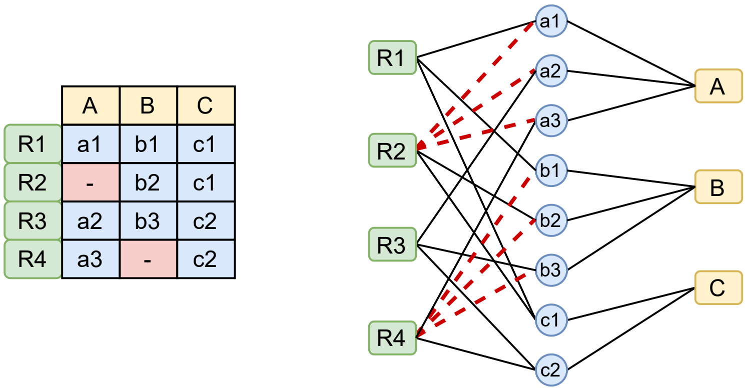
5.2.2 Generation of node features

The construction of GRIMP’s table graph is the first step in the preparation of the table for the imputation procedure carried out by the multi-task model. There is, however, the need for an interface that transforms the table graph into a form that can be “ingested” by the multi-task classifier: to this end, we rely on the latest advancements in graph embedding [85, 87], represented by the so-called Graph Neural Networks (GNNs). GNN architectures combine the features of neighbors of a node with the node’s own features to produce a merged representation of the node itself. GRIMP requires feature representations for tuples, attributes and cell values in order to function. These features can either be randomly generated, or prepared by relying on external methods such as those described in the rest of this section. Unsurprisingly, randomly generated features do not work as well as reasoned (pre-trained) features do; however, the model can still converge to an extent even when using random features.

To this end, we propose three strategies to initialize the node features. In all strategies, features are generated using the dirty version of the dataset, with its missing values; no ground truth information is visible to the embeddings before training occurs.

The first solution relies on pre-trained embeddings (specifically FASTTEXT embeddings): cell values are passed to the pre-trained embeddings algorithm, which then generate a new vector representation for each value. Then, the vector representation of each tuple is prepared by averaging the vector representations of the values in the tuple. Similarly, the vector representation of each attribute is prepared by averaging the vectors of the values in the attribute. Works such as [70] and [71] apply this tabular representation technique to the Entity Resolution problem.

As a second solution, we rely on local embeddings generated by EMBDI. We slightly modify the graph used by EMBDI to better handle null values by introducing new “possible” edges to the edge set of the EMBDI graph. For example, given a missing value $t_i[A_j]$ in tuple t_i and attribute A_j, the algorithm generates a new edge connecting t_i’s node to all values in the domain of attribute A_j. In Figure 5.3 we show a simple table



	A	B	C
R1	a1	b1	c1
R2	-	b2	c1
R3	a2	b3	c2
R4	a3	-	c2

Figure 5.3: Modified version of the EmbDI graph as it is used to generate the features.

with a missing value and the corresponding modified EmbDI graph: node R2 is now connected to both $a1$ and $a2$ via the dashed lines, whereas no edges would be present there in the original EmbDI graph. By introducing these edges, the graph is now aware to an extent of the fact that a missing value can take any of the values in the attribute domain. To model the fact that these edges are "possible", rather than "exact", each edge e_{ix} connecting tuple t_i to cell value $x = t_i[A_j]$ is weighted proportionally to the frequency of x in attribute A_j, with more frequent values having a higher weight. As a safeguard against imbalanced classes, weights are scaled by a function W with has a parameter k and frequencies in the range $[f_{min}, f_{max}]$; given a node n_t with frequency f_t, the weight w_t of each edge connecting n_t to a tuple node are assigned by function W, which scales all weights in the range. The function W scales the values to a range $[f_{max}, k \cdot f_{max}]$, then normalizes the range to $[1/k, 1]$. This procedure assigns less-frequent values a weight that is, at least $1/k$ with a parameter k; as a result, edges connected to rare values are far more likely to be selected than they would if there were no re-scaling. To avoid an explosion in the number of graph edges, we set a cap of maximum 20 new edges for each missing value: without a cap, errors attributes with very large domains would lead to the generation of too many edges.

Example 5.2.1. *Given the frequency list* $f = \{1, 15, 1, 2, 50\}$ *with* $f_{max} = 50$ *and scaling parameter* $k = 4$*, the scaled output of* W *is* $W(f) = \{\boldsymbol{0.25}, 0.4643, 0.25, 0.2653, \boldsymbol{1}\}$.

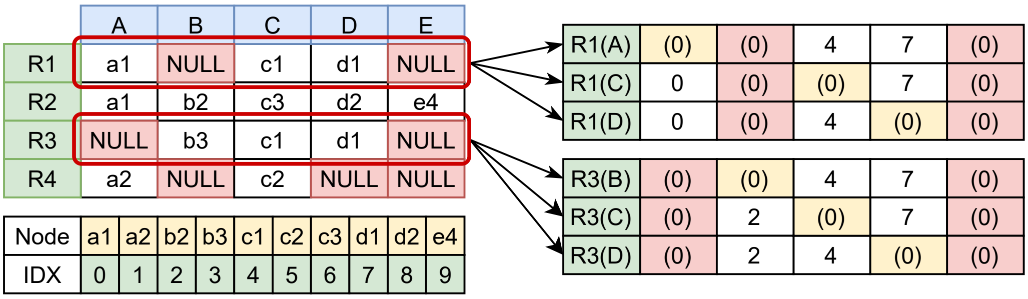
As our third solution, we combine the embeddings generated by fastText with the embeddings generated on the modified EmbDI graph. This approach is motivated by the fact that, in some cases, the embeddings generated by fastText outperform the local embeddings generated by fastText as they contain external information coming from the pre-training on the text corpus. By combining the two versions, we leverage information coming from both vector spaces, which includes both the local knowledge of the domain present in the EmbDI embeddings and external knowledge encoded by the fastText embeddings. To combine the embeddings, we first normalize the embeddings coming from both sources, then we concatenate them to obtain a representation of each node that combines the features. To counter the issues (memory footprint, training time) caused by the concatenation of very large vectors, a PCA projection of the normalized and concatenated vectors is carried out to reduce the number of dimensions down to a value provided by the user (by default, 64).

5.2.3 Creating the Training Samples

As mentioned in the introduction, GRIMP employs a representation of the tuples that is efficient, and at the same time robust to noise. In GRIMP, each non-missing value (that is, each value that is available in the dirty dataset) is used for training the model: this is done by creating a copy of the tuple it is found in, where the value is removed. Since we know what the correct imputation for this "fake" missing value is, we can impute this value and measure a loss during the training. Therefore, a tuple's training samples are copies of the tuple, where one value at a time is replaced by a null: this null is what is used to train the classifier.

In reality, the classifier is not expecting a training sample which contains table values: it takes as input a vector that represents information about that training sample. This *training vector* is built by replacing each value in the training sample with its vector representation, as it is generated by the GNN. This can be a very slow operation, if done in a naïve way, as it requires a large number of look-ups for each training sample; the fact that the GNN updates the node representations during each epoch signifies that the look-ups should be done in each epoch.

To carry out this operation more efficiently, we index all unique values in the table (i.e. values that will be assigned a node in the table graph), then we use the training samples to store the indexes of each value. The index of a value can be used to retrieve both the value, and the vector for that value as it was generated by the GNN. In this way, given a training sample that contains indexes, it becomes possible to build a training vector for that training sample by selecting the vectors corresponding to the indexes found in the training sample.



	A	B	C	D	E
R1	a1	NULL	c1	d1	NULL
R2	a1	b2	c3	d2	e4
R3	NULL	b3	c1	d1	NULL
R4	a2	NULL	c2	NULL	NULL

Node	a1	a2	b2	b3	c1	c2	c3	d1	d2	e4
IDX	0	1	2	3	4	5	6	7	8	9

R1(A)	(0)	(0)	4	7	(0)
R1(C)	0	(0)	(0)	7	(0)
R1(D)	0	(0)	4	(0)	(0)

R3(B)	(0)	(0)	4	7	(0)
R3(C)	(0)	2	(0)	7	(0)
R3(D)	(0)	2	4	(0)	(0)

Figure 5.4: Example of training sample generation in GRIMP.
GRIMP training samples are generated starting from all tuples in the dataset. Given a tuple, a training sample is generated for all attributes that contain a non-null value.

Precisely, for every tuple t_i in the table, GRIMP generates a training sample for each attribute in the table by adding a new missing value to the tuple, unless the tuple already contains a missing value for the attribute at hand. This is shown in Figure 5.4:

given tuple $R1$, it creates training samples for attributes A, C and D, since they are the attributes for which $R1$ contains a known value; it is not possible to generate a training sample for attributes B and E because those attributes already contain a "true" missing value. For each target attribute, the corresponding value is removed (shown as (0) with a yellow background in the figure). Then, the training sample is built by concatenating either the index of each value in the graph adjacency matrix, or a flag index that will be replaced by a null vector.

During training, the classifier uses the indices in each training sample to generate a *training vector* by probing the index for the node representation of each value, then concatenating them.

Example 5.2.2. *Given tuple* $R1 = [a_1, NULL, c_1, d_1, e4]$*, the training sample* $R1(A)$ *for attribute* A *is* $R1(A) = [-1, -1, 4, 7, -1]$ *with* -1 *denoting a "null flag". During training, the resulting training vector becomes* $\mathbf{v} = [\varnothing, \varnothing, \mathbf{c_1}, \mathbf{d_1}, \varnothing]$.

An advantage of this approach is how it allows to employ even tuples that contain a large fraction of "true" missing values for generating training samples. This can be done by inserting missing values in the attributes that are still available. Looking again at record R_1 in Figure 5.4, it is possible to generate training samples for attributes A, C and D, since attributes B and E contain missing values. Similarly, record R_2 can be used to create training samples for attributes B, C and D. Furthermore, training sample $R_3(B)$ shows another side of how the multi-task architecture is robust to high amounts of noise: by imputing each attribute using a separate head, samples $R_1(A)$ and $R_3(B)$ can be used to train each head separately, even though the input to each head is exactly the same. In a multi-class classifier where all values are in a single domain, by passing the same exact input, we would expect the same output: in this case, the classifier would likely select one of either $R_1(A)$ or $R_3(B)$, thus failing the imputation for one of the training samples. Thanks to the multi-task nature of the architecture, this is not a problem: each training vector is fed to a separate head, with a distinct domain. As a result, the same input fed to different heads leads to different imputations.

Algorithm 8 shows the procedure building the training samples.

5.2.4 Multi-Task Learning Component

The "heart" of GRIMP is an imputation module that employs a multi-task learning (MTL) architecture to conduct a multi-class classification operation on every attribute in the dirty dataset. GRIMP employs a multi-task, *hard parameter sharing* architecture with a *shared section* where all parameters are shared among all tasks, and a *task-specific* section where each task is implemented by a specific head. MTL improves training efficiency by reducing the training time compared to a normal multi-class classifier and by exploiting all available data for training, since there is no need to remove duplicates from the training samples (see Section 5.2.3). Furthermore, by having heads with distinct imputation domains, the MTL model can only select values found in a head's domain; a traditional multi-class classifier would instead select a value from the entire domain of the table, which increases the likelihood of selecting a value that does not belong to the

Algorithm 8 GenerateTrainingSamples

```
Input: Table $T$, node index $\mathcal{I}$
let $S$ = empty list
for all $r_i$ in rows($T$) do
    let $s_i$ = empty list
    for all $A_j$ in attributes($T$) do
        $c = r_i[A_j]$ {Content of the cell $r_i[A_j]$}
        if $c$ is NULL then
            $s_i[A_j] = \varnothing$ {$\varnothing$ will be replaced by an empty vector.}
        else
            $s_i[A_j] = \mathcal{I}(c)$ {Find the index of value $c$. }
    $S.addSample(s_i)$
Output: sample set $S$
```

attribute's domain: this increases the final imputation accuracy. Finally, this approach allows the model to handle mixed-type variables.

We report more details for this design choice in Table 5.1. Under *task efficiency* we describe improvements in the process execution time, with *imputation accuracy* we describe the effectiveness in the imputation task while *task generality* is the possibility of applying the model to a wider set of tasks (e.g. datasets with very large fractions of missing values, mixed-type datasets).

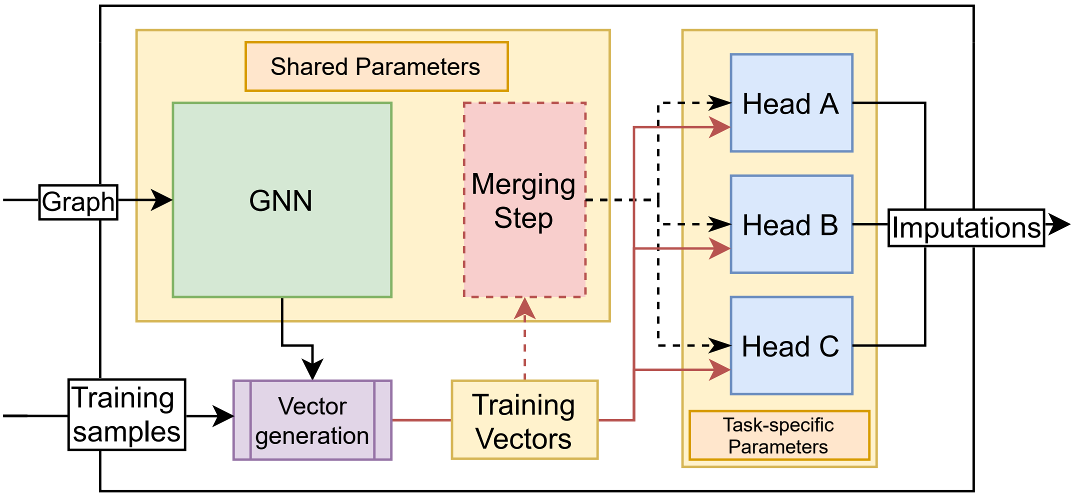


Figure 5.5: Schema of the GRIMP Multi-task component

To explain the MTL structure implemented in GRIMP, we follow Figure 5.5. The MTL component takes as input the table graph and the set of training samples described in Section 5.2.3. The table graph contains all the graph information needed by the GNN, i.e., the graph adjacency matrix and the node features. Figure 5.5 highlights the different sections (shared and task-specific) of the MTL classifier. The shared section includes the GNN and an additional (optional) shared layer, while the task-specific section is composed of multiple *heads*, one head for each attribute. Depending on the attribute's datatype, the corresponding head's output can be either a multi-class classifier or a single valued output if the attribute is categorical or numerical respectively.

Each sub-component of the MTL classifier is be explored in more detail in its own section.

Motivation	Training efficiency	Imputation accuracy	Task generality
The domain size of each task is much smaller compared to the full domain size that would be required by a multi-class classifier trained with the same objective. This increases imputation accuracy and reduces the training time.	$\surd$	$\surd$	
It is possible to reuse the same training tuple for different columns, since the relative tasks are not aware of each other and will thus produce different outputs given the same input.			$\surd$
This architecture allows to train the model only on a subset of the columns in the entire dataset, which is advantageous in some circumstances.	$\surd$		$\surd$
There is no risk that the model will select imputation values belonging to other attribute domains, which is more likely to happen when some values appear very frequently in the table.		$\surd$	
Different tasks might use different loss functions, as long as they can be combined together. This allows to employ both numerical- and categorical-focused loss functions.		$\surd$	$\surd$

Table 5.1: Features of the MTL model and how they benefit training efficiency, imputation accuracy and task generality compared to a single-task classification model.

Shared Layer and GNN

Figure 5.6 displays the heterogeneous Graph Neural Network, or Heterograph component of GRIMP. The GNN is heterogeneous because it is built by combining a number of different sub-modules, one for each datatype. Each sub-module can use a different graph representation method (e.g. plain GCN, GraphSAGE, etc.) to handle edges that belong to its type. A layer $i \in \{1, 2, \ldots n_{layers}\}$ comprises N sub-modules (where N is the number of columns in the original dataset), where a sub-module is defined as GNN_{ij} $\forall i \in \{1, 2\} \wedge \forall j \in [1, 2, \ldots, N]$ and each column in the starting dataset is assigned a sub-module. Each sub-module GNN_{ij} performs its convolution exclusively on nodes connected by edges of the type it pertains to (e.g. values belonging to column 2 are

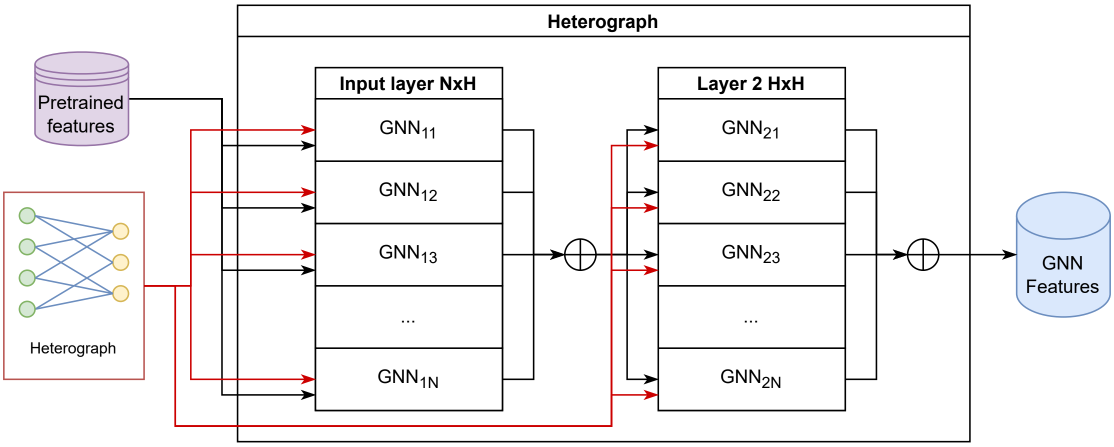


Figure 5.6: Schema of the GNN component in GRIMP.

described by sub-modules $GNN_{12}, GNN_{22}, \ldots, GNN_{n_{layers}2}$).

The GNN module takes as input the adjacency matrix of the heterograph built on top of the dataset and combines it with a set of pre-trained features for each node in the graph in each of the two convolutional layers. Each of the layers combines the pre-trained features with the locality features given by the adjacency matrix. The output of the overall network is the output of the final layer: the result is a new representation for each node in the graph. In-between the convolutional layers, a pooling component combines the node representations; the vectors that are produced at the end of the final layer are used in the *vector generation* procedure to be transformed in the training vectors that are used in the next steps of the procedure.

The vector representation of a vector v after layer k is modeled as follows:

$$h_v^{(k)} \;=\; \sigma(\gamma(\mathbf{W}^{(k,i)} \cdot f_k^{(i)}(h_v^{(k-1)}, \{h_u^{(k-1)}, \forall u \in S_{\mathcal{N}(v)}\})\ \forall i \in [1, N])) \qquad (5.2)$$

Here, σ is a nonlinearity, γ is an aggregation function that combines the vector representation produced by each sub-module GNN_i

From our experimental observations, two is the optimal number of layers: one layer is not enough to satisfactorily represent the spatial characteristics (that is, the neighborhood of nodes), while architectures with more than two layers produce worse results and require a longer training. The trainable weights are not shared among sub-modules, which allows some independence between each column while modeling each node's feature representation. In our implementation, we employ GraphSAGE [109] as the convolution operation used in each sub-module, although it is in principle possible to use a different operator in each sub-module by leveraging the fact that edges are typed.

The "merging step" box shown in Figure 5.5 is an additional, optional layer that can be implemented with linear or attention-based components and whose role is adding a further pooling step to the model. We have observed experimentally that this module can improve the imputation performance for some datasets.

Once the vector representation of each node is available, it becomes possible to prepare the training vectors by probing the node index for the indexes found in the

Algorithm 9 BuildTrainingVectors

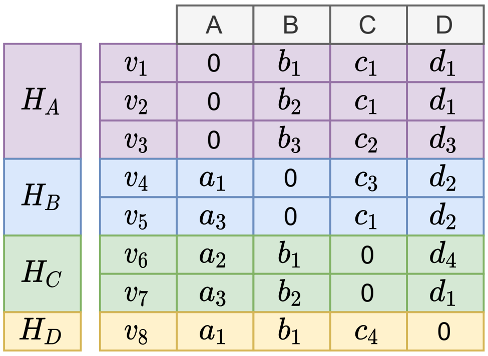
```
Input: node representations $h$, training samples $S$, node index $\mathcal{I}$
let $V$ = empty list
let $\emptyset$ = empty vector of size $|h|$ {$|h|$ = number of dimensions of the node representations}

for all $s_i$ in $S$ do
  let $v_i$ = empty list
  for all $s_{ij}$ in $s_i$ do
    if $s_{ij} = \varnothing$ then
      $v_{ij} = \emptyset$ {$\emptyset$ is an empty vector.}
    else
      $v_{ij} = h(\mathcal{I}(s_{ij}))$ {Find the representation of $s_{ij}$}
  $V.addVector(v_i)$
Output: vector set $V$
```

training samples. This indexing procedure (explained in Algorithm 9) allows to efficiently reconstruct all training vectors through what practically amounts to a "substitution" operation, rather than a far more time-consuming iterative concatenation of vectors: the reconstruction operation is carried out in each epoch, so an inefficient implementation of this procedure would make training impractical. Finally, the training vectors that are generated in this step are forwarded to the task-specific layer for the imputation operation to be carried out.

MTL Task-specific Layers

The task-specific layer is where the actual imputation operation is carried out. Firstly, each attribute in the dirty dataset is assigned a *head*, whose characteristics depend on the type of attribute: if the attribute is categorical, then the head will be a multi-class classifier whose domain is the same as the domain of the attribute; if the attribute is numeric, then the head will instead be a regressor with a single output.



		A	B	C	D
H_A	v_1	0	b_1	c_1	d_1
	v_2	0	b_2	c_1	d_1
	v_3	0	b_3	c_2	d_3
H_B	v_4	a_1	0	c_3	d_2
	v_5	a_3	0	c_1	d_2
H_C	v_6	a_2	b_1	0	d_4
	v_7	a_3	b_2	0	d_1
H_D	v_8	a_1	b_1	c_4	0

Figure 5.7: Example of the distribution of training samples over different heads.

Each head takes as input a subset of the training vectors relative to the attribute the head pertains to. Figure 5.7 shows an example of this. Training vectors v_1, v_2, v_3 are prepared for head H_A, and thus their value in the attribute A is set to be empty. Similarly, vector v_8 is prepared for head H_D. While the entire matrix is sent to the task-specific layer, each head has access only to its vectors: H_A does not see vectors v_6 and v_7, for example. The entire collection of training vectors becomes matrix V. Depending on the distribution of missing values in the dataset, each head might take a different number of vectors to train on (e.g. H_A has three vectors, while H_D only has one). Experimentally, this did not cause issues. Moreover, the previous shared steps help with supplying additional information to the attributes that have fewer training vectors to work with.

In GRIMP, heads can be implemented using linear layers that rely exclusively on the training vectors, or through an attention layer that combines attribute-level information with the training vectors. While the first solution is much faster to train, the latter has better results. Linear heads provide a simple architecture for keeping some of the parameters isolated from all other heads, so that the classification (or regression) objective in each head is not as influenced by the other tasks. Very shallow architectures (up to three linear layers) are normally enough to obtain good classification results. Simple linear heads cannot make use of attribute node features, which is a major downside compared to the attention heads we describe in the next section.

Algorithm 10 shows the steps necessary for building the task-specific heads.

Algorithm 10 BuildMultiTaskHeads

```
Input: table $\mathcal{D}$ with schema $\mathcal{R}$
let $N(\mathcal{R})$ = set of numerical attributes
let $C(\mathcal{R})$ = set of categorical attributes
let $\mathcal{H}$ = empty list
for all $A_j$ in $C(\mathcal{R})$ do
    $N_j = |A_j|$ {Find the cardinality of the attribute. }
    $out_{A_j} = N_j$ {Size of the head's output layer. }
    $h_i$ = CreateHead($out_{A_j}$)
    $\mathcal{H}$.addHead($h_i$)
for all $A_j$ in $N(\mathcal{R})$ do
    $out_{A_j} = 1$
    $h_i$ = CreateHead($out_{A_j}$)
    $\mathcal{H}$.addHead($h_i$)
Output: $\mathcal{H}$
```

5.2.5 Attention Structures

Inspired by the extensive work on attention mechanisms and by the very positive results achieved in a number of fields by systems that implement them, we add an attention mechanism to GRIMP's classifier architecture.

We modify and incorporate the architecture introduced in AimNet [80] after optimizing it for our scenario. The objective of our attention structure is combining the attribute- and tuple-level features prepared in the previous steps in order to leverage the information they contain. By employing attribute-level information on top of the already available tuple-level information, GRIMP improves the imputation performance on all tested datasets. We implement an attention layer both in the merging step (attention shared across all training vectors) and in the task-specific sections of the classifier, but we observe experimentally that the shared attention layer does not improve the imputation performance and increases the training time. For this reason, we elected to use the simpler linear model in the shared layer and reserve the attention structure to the multi-task heads. We observed that the attention structure tends to smooth out the training vectors, which are then forwarded to the later layers: it is possible that this smoothing operation removes some of the features that the heads are exploiting to perform their own operations.

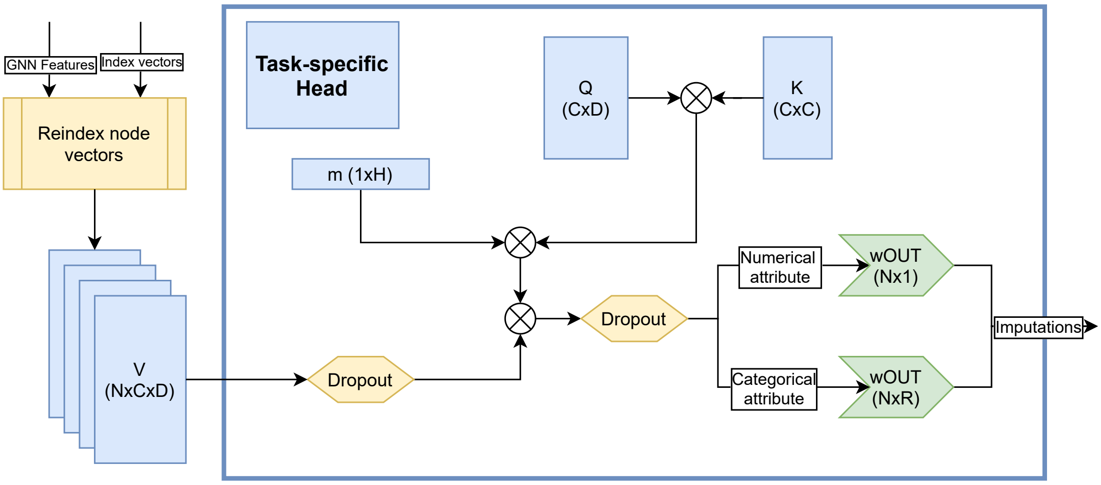


Figure 5.8: Internals of a classification head in the multi-task model.

Head Attention

GRIMP's head attention structure is shown in Figure 5.8. The "attention matrices" Q, K and V are shown in light blue, along with the pooling vector $\mathbf{m}$. For this example, we focus on the input to a single head H_A for attribute A: each head in the multi-task model receives a set of parameter specific to that head, these parameters have the same suffix as the head (for example, K_A refers to the version of matrix K that is built for head H_A).

Given the collection of node vectors prepared by the GNN, and the index vectors prepared as shown in Section 5.2.3 (shown as "GNN Features" and "Index vectors" respectively in Figure 5.8), the matrix V is prepared by the "reindex node vectors" step. The result is a N-dimensional matrix that contains all training vectors. As explained in Section 5.2.4, head H_A has only access to a subset of the vectors contained in V: we define this subset as V_A.

Matrix V_A contains the vector representation of the training tuples that was prepared in the previous step by the GNN and the shared layer. It has shape $N_A \times C \times D$ where N_A is the number of training vectors prepared for attribute A, C is the number of columns, D is the number of dimensions of the vector representation of each node, i.e. the number of output dimensions of the shared layer.

Matrix Q_A has shape $C \times D$ and contains the pre-trained vectors of each column in the dataset; matrix Q_A holds the attribute information that must be combined with the information produced in the shared layer. The content of the matrix Q_i is the same for all attributes $i \in [1, \ldots, C]$ when the heads are built, but each head H_i modifies its own Q_i independently of other heads.

K_A is a binary matrix that is used for selecting only a set of the columns that each head should work with, assigning each column a given weight. Figure 5.9 shows the result of constructing matrix K using the "weak diagonal" (explained later) strategy. Matrix K_A is assigned to head H_A: since it pertains to attribute A, the weight given to A is maximum, while the other columns receive a lower weight; similarly, K_B refers to H_A, and thus attribute B receives the higher weight.

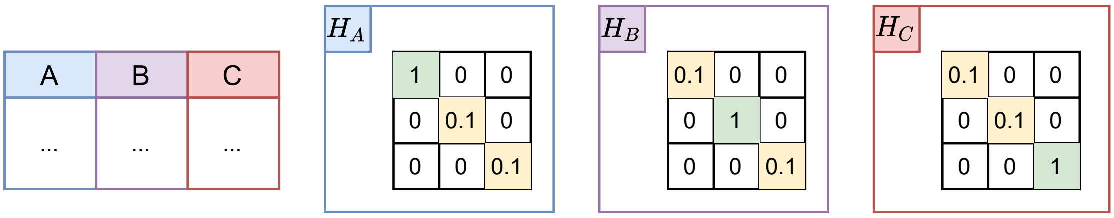


Figure 5.9: Example of K matrices for a table with three attributes.

We tested a number of strategies for building K, which differ in the values found on the diagonal. We have observed that different strategies have some effect on the imputation accuracy, however there does not appear to be a strategy that dominates the others in all cases. Figure 5.10 shows different variants of K for head 2 in a relational table with five columns:

- **Diagonal**: In this variant, all columns have equal weight.
- **Target column**: In this variant, all columns except the head's column are ignored.
- **Weak diagonal**: A "middle ground" between the previous two, where the target column has the highest weight, but other columns are still considered with a lower importance.

K can be used to encode additional, external information by modifying the weight that is assigned to each column. If, for example, functional dependencies that link multiple columns are known, it is possible to "highlight" those columns in the matrix so that the model give them a higher weight during training. The latter two strategies in Figure 5.10

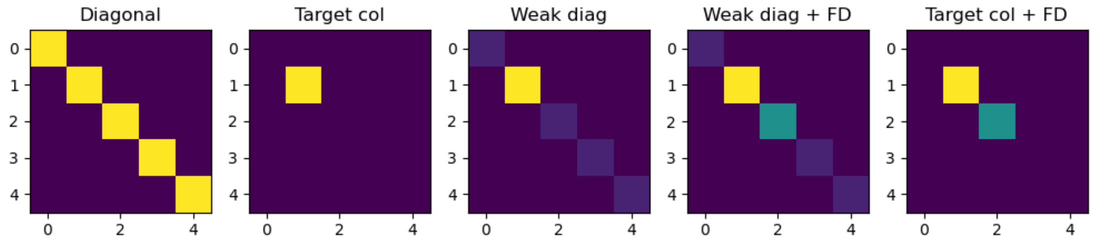


Figure 5.10: Different variants of matrix K in the head relative to attribute 2, with a functional dependency between attribute 2 and 3.

show how this can be done given target column 2 and a functional dependency that links column 3 to column 2: **Weak diagonal + FD** implements the previous **Weak diagonal** strategy, but assigns column 3 a higher weight; similarly, **Target column + FD** considers only the target column and the column related through a FD.

Finally, $\mathbf{m}$ is a vector of size $1 \times C$ which contains only 1. This vector is used to pool the result of the multiplication of K_A and Q_A: this step is done to select the attributes that should be attended to the most by the current head.

At training time, each head H_i is trained by passing V_i, a subset of the training vectors V, which is passed to the attention layer. In each head H_i, matrix Q_i and K_i are multiplied before being pooled by vector $\mathbf{m}$. After multiplying the result with matrix V_i, the final matrix passes through a linear layer whose output size is either equal to the cardinality of the domain of the head's attribute if the column is categorical, or one-dimensional if the column is numerical.

The steps we have described so far represent an extension of the attention structure described in Aimnet [80] for a multi-task architecture. As an added advantage, matrix K allows to encode additional information in the model by tweaking the weights assigned to each column based on known external information. In the experimental section we demonstrate that this attention strategy noticeably improves the imputation accuracy.

5.2.6 Training Procedure

The training procedure of GRIMP is summarized in Algorithm 11. The first step of the pipeline revolves around the construction of the table graph and the generation of training samples. Once the architecture is prepared, the model is trained iteratively over a certain number of epochs. The training duration expressed as number of epochs required varies dataset by dataset, so a fraction of the training samples is held out in a validation step to make it possible to implement early stopping policies and stop the training once the validation error starts increasing. To avoid contaminating the validation step, before the training starts we remove all edges incident in the validation step from the graph representation.

Algorithm 11 Pseudocode of the GRIMP pipeline.

```
Input: Table $\mathcal{D}$
NormalizeNumericalAttributes(table $\mathcal{D}$)
$G$ = GenerateGraph(table $T$)
$S$ = GenerateTrainingSamples (table $\mathcal{D}$)
$\mathcal{H}$ = BuildMultiTaskHeads (table $\mathcal{D}$)
for epoch $E$ in $n_{epochs}$ do
    $h$ = GNN(G) {Generate the node embeddings using the GNN.}
    $V$ = BuildTrainingVectors($h$, $S$)
    $h_{shared}$ = SharedLayer($h$, $V$) {Feed the node embeddings to the shared layer of the
    multitask classifier.}
    for head $H_i$ in $\mathcal{H}$ do
        $h_i$ = HeadLayer($H_i$, $V$, $h_{shared}$)
        if type($H_i$) is numerical then
            $loss_i$ = RMSE($h_i$, $S$)
        else
            if type($H_i$) is categorical then
                $loss_i$ = CrossEntropy($h_i$, $S$)
    $loss_E = \sum_i^{n_{heads}} loss_i$
```

Loss Function

In each epoch, each head measures its own loss independently of the others and according to its type: categorical attributes use *Cross Entropy* loss or *Focal loss*, while numerical attributes use MSE. Numerical values are normalized before the training starts, so that their MSE is comparable in magnitude to the Cross Entropy loss measured for categorical variables: this allows to combine the loss of each head in a total loss without the need for weights.

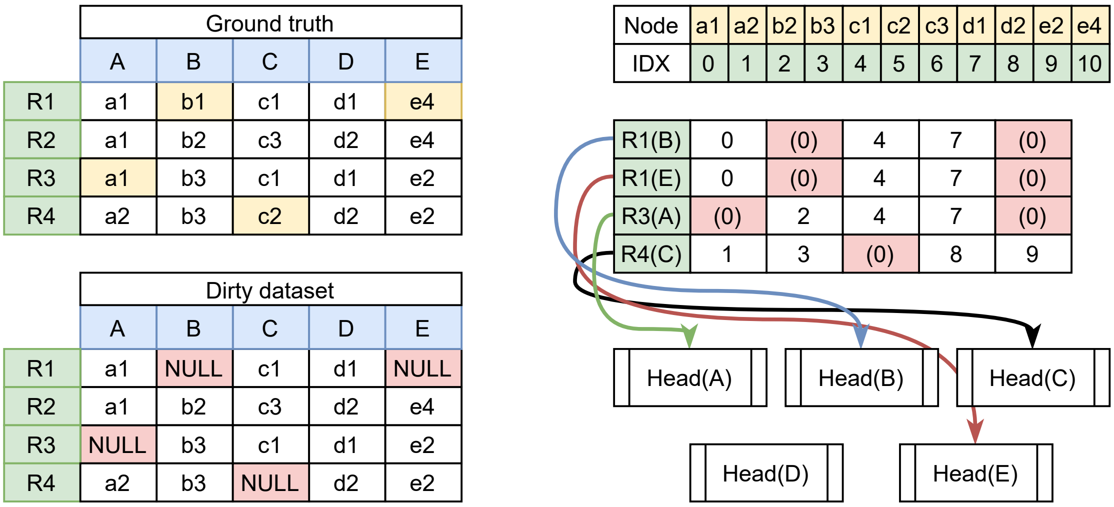
$$loss \;=\; \sum_{i}^{N} loss(H_i) \tag{5.3}$$

Here, N is the number of heads used in the training.

5.2.7 Imputing the Missing Values

After model training ends (either by early stopping or by completing all epochs), the actual imputation procedure is carried out. For GRIMP, we start from the assumption that we do not have access to the ground truth during the training procedure. To test the final imputation accuracy, we introduce errors in datasets that contain no missing values, thus producing “dirty datasets” for which we have access to the ground truth (that is, the original datasets with no missing values).

Imputation of missing values is done by probing the model through the use of testing samples, which are structurally the same as the training samples described in Section 5.2.3, except that no additional missing values are added to the vector, other than the



	Ground truth				
	A	B	C	D	E
R1	a1	b1	c1	d1	e4
R2	a1	b2	c3	d2	e4
R3	a1	b3	c1	d1	e2
R4	a2	b3	c2	d2	e2

	Dirty dataset				
	A	B	C	D	E
R1	a1	NULL	c1	d1	NULL
R2	a1	b2	c3	d2	e4
R3	NULL	b3	c1	d1	e2
R4	a2	b3	NULL	d2	e2

Node	a1	a2	b2	b3	c1	c2	c3	d1	d2	e2	e4
IDX	0	1	2	3	4	5	6	7	8	9	10

R1(B)	0	(0)	4	7	(0)
R1(E)	0	(0)	4	7	(0)
R3(A)	(0)	2	4	7	(0)
R4(C)	1	3	(0)	8	9

Figure 5.11: Example of how GRIMP generates the testing samples for the dataset on the left.

values that are already present (and that must be imputed). Consider Figure 5.11: given the dirty dataset built by injecting random errors in the ground truth, all values with white background in the ground truth become training samples, while all values with yellow background become missing values in the dirty dataset, and therefore testing samples for which we want to test the imputation accuracy. The testing samples for the table in the example are shown on the right hand side of the figure, together with the head each sample is fed to for the imputation.

After generating the test samples, each sample is fed to the GRIMP pipeline and specifically to the head relative to the test sample's attribute: for example, in Figure 5.11, test sample $R1(B)$ is forwarded to head B, while sample $R3(A)$ is forwarded to head A). For categorical values, the imputation is chosen by selecting the value with the highest likelihood, while numerical variables are imputed by sampling the numeric head for a value, then de-normalizing it to obtain the actual value.

A further advantage of the multi-task structure is that it is possible to organically train the model and carry out imputation on a subset of the columns, rather than training the full model on all attributes that include missing values.

5.3 FunForest

While we are working with unsupervised methods to ensure that they can be applied to situations where no knowledge about the data at hand is available, it should still be possible to include external information in the training procedure if it becomes available. Functional Dependencies describe one such kind of information, and GRIMP's attention layer (described in Section 5.2.4) is a viable method for the introduction of functional dependencies in a DL-based data imputation pipeline.

In this Section, however, we introduce FunForest, a different approach to solving the same problem, in which we take a well-known and effective algorithm in MissForest [45] and modify it so that it can make use of known functional dependencies. We introduce lightweight changes that not only improve the imputation results by a large margin, but they also speed up the convergence of the algorithm, thus reducing the execution time.

FunForest is an extension of the MissForest algorithm that exploits external information under the form of Functional Dependencies to pilot the training algorithm so that a larger fraction of the random forest is observing data relevant to the imputation task at hand.

5.3.1 MissForest

The basic implementation of MissForest trains a random forest classifier (or regressor, depending on the imputation data type) on the dataset, then performs imputation by "predicting" for each missing value the most likely imputation. The basic algorithm has proven to work quite well in many circumstances, and as such has been used as a benchmark for other data imputation datasets in the literature.

Thanks to the fact that the decision trees that make up the random forest can cover multiple combinations of attributes in the dataset, MissForest is able (to an extent) to learn relationships between attributes akin to Functional Dependencies. Unfortunately, since the attribute choice is random, a large faction of the estimator budget may be allocated to spurious combinations, thus reducing the effect of the FDs on the imputation result. On the other hand, this distributed training allows to pick up on partial relationships between values that can be missed by systems that focus exclusively on FDs. We develop FunForest to counter this problem.

5.3.2 From MissForest to FunForest

Our contribution enhances MissForest by exploiting external information to focus the attention of the classifier on columns that are known to be related to each other according to specific FDs. By "pointing" the decision trees at a subset of dataset attributes, rather than the entire table, it is possible to reduce the noise introduced by unrelated columns and instead focus on the values found within the more relevant attributes. This is done by allocating a fixed budget of decision trees that are trained exclusively on a small subset of the attributes, while the rest of the decision trees are handled in the same way as the original MissForest algorithm did. If multiple functional dependencies are present in the table, then the budget is split equally among all FDs.

Allocation of the Budget

One of the main hyperparameters of MissForest is the number of estimators that should be generated to perform classification/regression, which should be specified by the user. Since our extension of the original algorithm exploits the same architecture, a fraction of the entire estimator budget must be allocated to those trees that will focus on the attributes involved in FDs. Choosing this budget is not trivial, as different budgeting strategies may lead to different results depending on the circumstances.

Using too many (or too few) FD decision trees may lead to adverse results, with the imputation algorithm either underfitting or overfitting on the given columns. In our experiments we observed both behaviors in different situations. Specifically, in those cases where the Functional Dependencies are not fully reliable, or there are too many of them, high FD budgets led to a noticeable drop in the imputation accuracy. On the other hand, when manually crafted, high quality FDs are available, low FD budgets will produce worse results. Interestingly, in some cases overusing FD trees leads to a slight drop in quality compared to cases where the budget fraction is 50%.

5.4 Experimental study

In this section, we report the results we obtain in our experimental campaign, where we test GRIMP and FunForest against a number of baselines on multiple datasets. We observe how different baselines have very close results on most datasets, and suggest some reasons for why this happens. We focus on datasets with known functional dependencies to test FunForest and GRIMP, and show FDs improve their effectiveness when they are available.

5.4.1 Experimental Setup

Dataset	Abbr.	# rows	# columns	# cat.	# num.	Distinct	R_{avg}
Adult	AD	3016	14	9	5	289	146.10
Australian	AU	690	15	9	6	957	10.82
Contraceptive	CO	1473	10	8	2	65	226.62
Credit	CR	653	16	10	6	918	11.38
Flare	FL	1066	13	10	3	34	407.59
IMDB	IM	4529	11	9	2	9829	5.07
Mammogram	MM	830	6	5	1	93	53.55
Tax	TA	5000	12	5	7	910	65.93
Thoracic	TH	470	17	14	3	255	31.33
Tic-Tac-Toe	TT	958	9	9	0	5	1724.4

Table 5.2: Table statistics for all the datasets we use in this work. *Abbr.* is the abbreviation that will be used across the section instead of the full dataset name for the sake of space. *Distinct* is the number of unique values found in the entire table. R_{avg} is the average redundancy of values in the table.

In this section, we introduce the datasets and the algorithms we use for testing data imputation.

Datasets

We employ mostly mixed-type datasets already known in the literature to study the effectiveness of different imputation methods for categorical data. **Adult**, **Australian**,

Contraceptive, **Credit**, **Flare**, **Mammogram**, **Thoracic** and **Tic-Tac-Toe** are sourced from the UCI repository [201], **IMDB** was prepared to be used in our experiments for EmbDI [173], **Tax** is a synthetic dataset for testing Conditional Functional Dependencies [51] and Denial Constraints [21]. All datasets are pre-processed so that any row that includes missing values is dropped before it is used in the pipeline, so it becomes possible to inject synthetic missing values with a known ground truth for testing purposes.

Table 5.2 reports some additional stats in the number of distinct values contained by a table, together with the average redundancy of values in the same table. Average redundancy describes the average frequency of occurrence of a value that belongs to the given dataset, defined as $R_{avg} = n_{rows} \cdot n_{cols} \ / \ n_{distinct}$. While imperfect, we have found the average redundancy to be a good indication of the difficulty involved in the imputation of errors in a given dataset. Interestingly, tracking the average redundancy over datasets with different percentages of missing values is a valid predictor of the final imputation accuracy for all baselines, with datasets with higher average redundancy being easier to impute. Figure 5.12 shows the average redundancy of the datasets we use under different error fractions (missing values percentages); Tic-Tac-Toe is not shown because of its redundancy being far larger than the average.

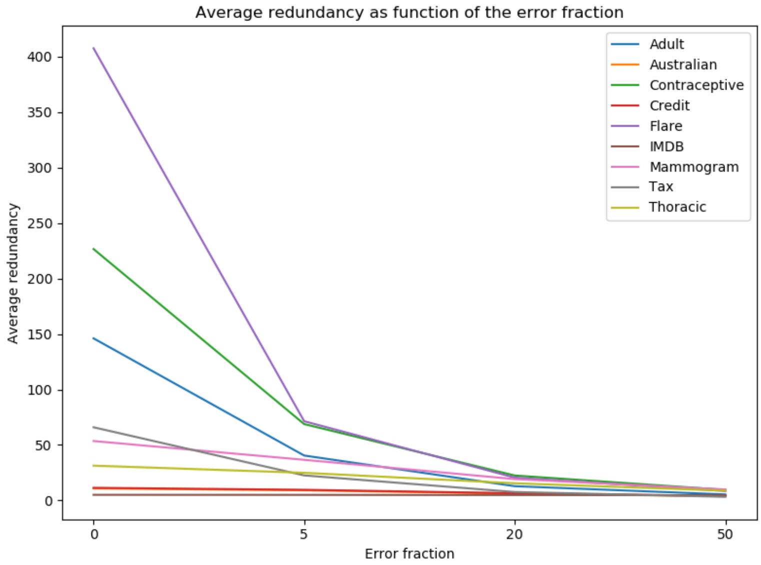


Figure 5.12: Average redundancy with increasing error fractions over all columns.

Algorithms

As our baselines, we use MissForest [45], AimNet [80] and DataWig [202]. We use the default parameters for all systems. For testing GRIMP, we test different pre-trained features and different combinations of parameters. FunForest requires known functional dependencies to work, so it is executed only on the two datasets that feature exact FDs, i.e., Adult and Tax. We compare the baselines against two GRIMP configurations, whose

difference lies in the pre-trained features that are used by the algorithm: GRIMP-FT uses fastText embeddings, while GRIMP-E uses EmbDI embeddings. In both configurations, GRIMP is using attention (as explained in Section 5.2.5) in the task-specific heads. We observed that the combination of fastText and EmbDI embeddings does not outperform the basic versions and therefore do not report it for the sake of space.
Experiments have been conducted on a laptop with a CPU Intel i7-8550U, 8x1.8GHz cores and 32GB RAM.

5.4.2 Imputation Results

We test the data imputation algorithms on a number of clean datasets which we corrupt by injecting increasing amounts of errors, starting from a relatively low error fraction of 5%, then we increase to 20 and finally 50% of missing values. Errors are injected with a MCAR distribution over the entire table, so that each attribute contains the same amount of missing values. The same dirty datasets are presented to each algorithm; we then execute the algorithm and save the output. The imputation accuracy is measured by comparing the dataset version imputed by each imputation algorithm with the ground truth, then measuring the number of correctly imputed categorical values out of the full set of missing values. In the following tables, we report exclusively the accuracy results for categorical attributes. For what concerns numerical attributes and RMSE, Holoclean outperforms other methods, while GRIMP is comparable to MISF and Datawig reports the worst results.

Comparison with Baselines

From the results reported in Table 5.3, GRIMP outperforms the other baselines in most cases, and remains competitive with Holoclean even in the cases in which the latter is the best solution. Overall, Datawig gets outperformed by all other solutions, while MissForest remains competitive w.r.t. the best solutions in most cases.

We compute the overall average imputation accuracy by measuring the average imputation over all runs carried out by a specific configuration (GRIMP-E, GRIMP-FT, Datawig etc.). If we combine the results with this metric to observe general trends, we observe that solutions based on GRIMP obtain the best results on average (last line of the table). Specifically, GRIMP with EmbDI obtains an average accuracy of 0.684, almost a 2% absolute increase over the state of the art method, Holoclean (0.665). Interestingly, even GRIMP linear, not reported in the table, outperforms the non-GRIMP solution with an average accuracy of 0.676. In other words, all variants of GRIMP perform better than the state of the art solutions.

From the point of view of the execution time (reported in Table 5.4), GRIMP with attention is often (but not always) the slowest algorithm, although Datawig is sometimes even slower. Missforest is always among the fastest systems. GRIMP-L (linear heads) is comparable in execution time to the faster algorithms, but tends to produce worse imputation results.

The execution time of both GRIMP and Holoclean decreases as the fraction of missing values increases: this is likely due to the fact that, with a larger fraction of missing values,

Dataset	Error %	GRIMP-FT	GRIMP-E	MISF	HOLO	DWIG
Adult	5	0.752	**0.773**	0.733	0.670	0.275
Adult	20	0.749	0.760	0.729	**0.764**	0.276
Adult	50	0.688	0.694	0.659	**0.699**	0.266
Australian	5	**0.729**	0.706	0.663	0.676	0.588
Australian	20	0.692	0.668	0.691	**0.700**	0.590
Australian	50	0.660	0.639	0.637	**0.669**	0.577
Contraceptive	5	0.659	**0.660**	0.623	0.655	0.380
Contraceptive	20	0.643	0.642	0.618	**0.649**	0.442
Contraceptive	50	**0.631**	0.624	0.595	**0.631**	0.424
Credit	5	0.758	0.753	0.718	**0.764**	0.648
Credit	20	**0.770**	0.748	0.752	0.765	0.650
Credit	50	0.696	0.682	0.682	**0.700**	0.595
Flare	5	**0.841**	0.835	0.828	0.825	0.604
Flare	20	0.794	**0.798**	0.769	0.785	0.585
Flare	50	0.764	0.763	0.744	**0.766**	0.643
IMDB	5	0.351	**0.352**	0.351	0.333	0.105
IMDB	20	0.347	**0.350**	0.345	0.324	0.106
IMDB	50	**0.349**	0.345	0.340	0.321	0.106
Mammogram	5	0.776	**0.790**	0.748	0.776	0.390
Mammogram	20	0.747	**0.752**	0.713	0.741	0.337
Mammogram	50	0.731	**0.718**	0.682	0.715	0.379
Tax	5	0.646	**0.800**	0.689	0.622	0.389
Tax	20	0.615	0.631	**0.661**	0.626	0.385
Tax	50	0.577	**0.585**	0.571	0.561	0.348
Thoracic	5	0.868	**0.871**	0.848	0.854	0.762
Thoracic	20	0.860	**0.859**	0.840	0.857	0.764
Thoracic	50	0.827	0.828	0.820	**0.829**	0.740
Tic-Tac-Toe	5	0.642	**0.648**	0.463	0.609	0.606
Tic-Tac-Toe	20	0.576	**0.581**	0.468	0.565	0.557
Tic-Tac-Toe	50	0.474	0.489	0.444	**0.494**	0.460
Average		0.683	**0.684**	0.648	0.665	0.466

Table 5.3: Imputation accuracy obtained with different imputation algorithms.

Dataset	Error %	GRIMP-L	GRIMP-A	MISF	HOLO	DWIG
Adult	5	88	710	**13**	52	302
Adult	20	79	726	**26**	47	138
Adult	50	71	552	**32**	48	330
Australian	5	35	130	**14**	16	63
Australian	20	32	147	21	**16**	96
Australian	50	32	152	35	**16**	97
Contraceptive	5	28	114	**9**	125	60
Contraceptive	20	28	113	**23**	107	150
Contraceptive	50	38	111	**14**	71	57
Credit	5	36	145	**22**	154	109
Credit	20	32	155	**19**	116	128
Credit	50	31	170	**22**	80	104
Flare	5	36	165	**15**	148	160
Flare	20	34	151	**18**	126	175
Flare	50	29	160	**21**	82	99
IMDB	5	176	495	**120**	991	165
IMDB	20	147	496	**113**	776	178
IMDB	50	**31**	469	61	436	172
Mammogram	5	10	28	**6**	46	55
Mammogram	20	**10**	27	14	41	35
Mammogram	50	9	24	**4**	26	44
Tax	5	156	916	**18**	879	606
Tax	20	151	1027	**23**	737	1068
Tax	50	91	665	**28**	506	1239
Thoracic	5	31	132	**20**	108	173
Thoracic	20	30	134	**16**	84	102
Thoracic	50	28	148	**28**	56	182
Tic-Tac-Toe	5	18	61	**8**	46	341
Tic-Tac-Toe	20	17	63	**13**	41	237
Tic-Tac-Toe	50	17	63	**15**	28	510

Table 5.4: Execution time in seconds required by different baselines on the given datasets.

fewer viable cells remain. As a result, both algorithms work over a smaller quantity of data and terminate earlier. This is mirrored by the other two baselines in MissForest and Datawig, whose models spend longer to train in high-error configurations.

Comparing GRIMP Configurations

Tables 5.5 and 5.6 report the results obtained by GRIMP on the same test datasets described in the previous sections, changing the architecture of the heads. In the "Attention" column, we report the performance of GRIMP with the same structure as in the previous section, that is by relying on the attention structures to combine attribute embeddings with the tuple embeddings generated by the GNN. In the "Linear" column, we instead employ a much simpler architecture, in which each head consists only of a number of linear layers.

It is clear from Table 5.5 that the more complex attention structure improves the results, outperforming the linear structures in almost all situations. On the flip side, Table 5.6 is a demonstration of how much faster the linear architecture is, compared to using attention.

From the results, none of pre-trained features appear to clearly surpass the others in all settings. Solutions based on EmbDI features slightly outperform the others and in general do not fall too far behind the best solution. In any case, all solutions based on pre-trained features outperform the random initialization (not reported in table).

5.4.3 Working with Functional Dependencies

In this section, we focus our attention on two datasets that have known, exact functional dependencies, i.e, Adult and Tax. The first dataset contains a bi-directional functional dependency (2 FDs over 2 attributes) while the second contains six exact FDs (over 10 attributes).

For this experiment, we again increase the fraction of injected errors from 5 to 20 and 50% over all attributes, then we measure the training time and the imputation accuracy. The results are reported in table 5.7. Imputation accuracy is measured as the fraction of correctly imputed categorical values over the entire set of missing values in categorical attributes.

From observing the results, it is clear that FunForest is a major improvement over the original MissForest algorithm when Functional Dependencies are available: indeed, the improved version leverages the information provided by the FDs to improve (by up to an absolute 10% in some cases) the imputation accuracy, whilst at the same time speeding up the convergence time of the algorithm by at least 50%. We also report the results obtained by training GRIMP with attention heads (GRIMP-A) on the same datasets. Modeling the FDs in GRIMP always introduces improvements in the imputation accuracy w.r.t. to the same setting without the external information. We observe that GRIMP outperforms both MissForest and FunForest on the Adult dataset, while the random forest methods work better on the Tax dataset with the higher error rates.

These results suggest that our approach to "piloting" FunForest's decision trees is very effective: by focusing the attention of the ensemble on attributes that are known

Case		Attention			Linear		
Dataset	Error %	EMBDI	FT	COMB	EMBDI	FT	COMB
Adult	5	**0.773**	0.752	0.766	0.766	0.760	0.755
Adult	20	0.760	0.749	**0.764**	0.746	0.753	0.748
Adult	50	0.694	0.688	**0.696**	0.679	0.678	0.676
Australian	5	**0.706**	0.729	0.686	0.678	0.694	0.657
Australian	20	0.668	**0.692**	0.671	0.659	0.675	0.678
Australian	50	0.639	**0.660**	0.647	0.625	0.631	0.638
Contraceptive	5	**0.660**	0.659	0.647	0.618	0.624	0.617
Contraceptive	20	0.642	0.643	**0.652**	0.617	0.624	0.613
Contraceptive	50	0.624	**0.631**	0.619	0.580	0.577	0.583
Credit	5	0.753	**0.758**	0.755	0.739	0.739	0.733
Credit	20	0.748	**0.770**	0.747	0.730	0.748	0.735
Credit	50	0.682	**0.696**	0.678	0.673	0.676	0.676
Flare	5	0.835	**0.841**	0.840	0.830	0.821	0.816
Flare	20	**0.798**	0.794	0.796	0.770	0.771	0.775
Flare	50	0.763	**0.764**	0.763	0.738	0.741	0.735
IMDB	5	**0.352**	0.351	0.350	0.339	0.350	0.345
IMDB	20	**0.350**	0.347	0.349	0.333	0.345	0.334
IMDB	50	0.345	**0.349**	0.344	0.330	0.345	0.327
Mammogram	5	**0.790**	0.776	0.786	0.754	0.767	0.779
Mammogram	20	0.752	0.747	**0.753**	0.738	0.734	0.724
Mammogram	50	0.718	**0.731**	0.727	0.705	0.711	0.717
Tax	5	**0.800**	0.646	0.647	0.785	0.776	0.792
Tax	20	0.631	0.615	0.639	**0.731**	0.729	0.727
Tax	50	0.585	0.577	0.583	**0.586**	0.584	**0.586**
Thoracic	5	0.871	0.868	**0.872**	0.836	0.839	0.858
Thoracic	20	0.859	**0.860**	0.852	0.825	0.835	0.820
Thoracic	50	**0.828**	0.827	0.824	0.801	0.790	0.805
Tic-Tac-Toe	5	0.648	0.642	**0.657**	0.639	0.631	0.648
Tic-Tac-Toe	20	**0.581**	0.576	0.575	0.540	0.541	0.538
Tic-Tac-Toe	50	**0.489**	0.474	0.474	0.444	0.446	0.454

Table 5.5: Comparison between different pre-trained features using linear heads and attention heads.

Case		Attention			Linear		
Dataset	Error %	EMBDI	FT	COMB	EMBDI	FT	COMB
Adult	5	710	819	639	**36**	37	**36**
Adult	20	726	636	661	**40**	42	43
Adult	50	552	550	628	**42**	**42**	45
Australian	5	130	131	129	**19**	**19**	**19**
Australian	20	147	139	138	20	20	**19**
Australian	50	152	152	156	**20**	**20**	21
Contraceptive	5	114	111	110	**15**	**15**	**15**
Contraceptive	20	113	120	112	**16**	**16**	**16**
Contraceptive	50	111	111	111	19	**16**	**16**
Credit	5	145	143	151	**20**	21	21
Credit	20	155	155	155	**21**	**21**	**21**
Credit	50	170	169	170	**21**	22	22
Flare	5	165	167	173	**18**	**18**	**18**
Flare	20	151	151	150	20	**19**	20
Flare	50	160	166	156	**19**	20	20
IMDB	5	495	496	490	54	**53**	54
IMDB	20	496	487	488	52	52	**51**
IMDB	50	469	457	458	**31**	101	45
Mammogram	5	28	28	27	**7**	**7**	**7**
Mammogram	20	27	27	27	**7**	**7**	**7**
Mammogram	50	24	26	24	**7**	**7**	**7**
Tax	5	916	1223	1065	61	70	**60**
Tax	20	1027	770	1156	**68**	78	73
Tax	50	665	702	777	**48**	**48**	**48**
Thoracic	5	132	139	130	**19**	**19**	**19**
Thoracic	20	134	134	134	**20**	**20**	**20**
Thoracic	50	148	148	148	**20**	21	**20**
Tic-Tac-Toe	5	61	60	71	**11**	**11**	**11**
Tic-Tac-Toe	20	63	63	63	**11**	**11**	**11**
Tic-Tac-Toe	50	63	61	63	**11**	**11**	**11**

Table 5.6: Comparison between execution times in seconds for linear and attention heads with different pre-trained features.

Case		Training time			Imputation accuracy		
Dataset	Error %	MISF	FUNF	GRIMP-A	MISF	FUNF	GRIMP-A
AD	5.00	13.03	**2.38**	496.60	0.733	0.737	**0.766**
AD	20.00	25.70	**6.05**	551.22	0.727	0.732	**0.756**
AD	50.00	22.50	**15.23**	537.90	0.657	0.674	**0.693**
TA	5.00	17.47	**6.00**	1117.54	0.689	0.786	**0.808**
TA	20.00	23.18	**7.62**	977.62	0.661	**0.757**	0.632
TA	50.00	27.94	**16.44**	751.93	0.571	**0.630**	0.586

Table 5.7: Imputation results of MissForest (MISF) against FunForest (FUNF) and GRIMP-A in presence of exact functional dependencies.

to be related to each other through external information, the algorithm is not wasting computational budget while searching for correlations between unrelated attributes. Clearly, this is very advantageous and the termination condition employed by MissForest can make full use of this faster convergence.

It must be noted that, while we have been using Functional Dependencies to "infuse" external information in the training procedure, this need not be the case: both algorithms can in principle make use of any kind of attribute-related external information to partition the estimator budget or steer the attention.

5.5 Summary

In this chapter, we explored the problem of implementing a system for imputation of missing values in tabular data in presence of categorical attributes. As a solution to the problem, we implemented GRIMP, a self-supervised imputation algorithm that combines a graph representation of the table, Graph Neural Networks, and multi-task learning to produce state of the art results over a wide variety of datasets. To improve the performance and inject external information in the model, we extended the task-specific heads with an attention mechanism tailored to the multi-task architecture. We then develop FunForest, an improvement over the MissForest imputation algorithm that leverages functional dependencies to pilot the model training. Finally, we carried out an extensive experimental campaign over a number of different datasets to test the performance of our contributions against various baselines.

Chapter 6

Conclusions and Future Research Directions

Over the course of this thesis, we have introduced the concept of data curation and its importance, then we focused on two specific subjects in the data curation domain, namely data integration and data imputation. We developed systems that do not require human supervision and that can naturally handle strings and categorical data.

In Chapter 2, we introduced the various concepts, methodologies and problems that were employed in the implementation of our main contributions.

With Chapter 3, we introduced EmbDI (**Emb**eddings for **D**ata **I**ntegration), a modular, unsupervised framework for generating local embeddings of tabular data. We first represented tabular data in graph form, then we generated the distributed representation of all nodes by traversing the graph with random walks that are used for training a word embeddings algorithm. By doing so, we were able to assign a distributed representation (an embedding) to both categorical (i.e., discrete) data, and the table's structural entities (i.e., tuples and attributes). We suggested a number of optimizations and proposed solutions to the problem of extracting information hidden within textual cells. We introduced novel methods for testing embeddings quality and we demonstrated that the embeddings generated by EmbDI on a single table can be employed for a number of applications in the data integration field and outside.

In Chapter 4, we moved from the one-table scenario explored in Chapter 3 to a two-table scenario, demonstrating how the embeddings generated by EmbDI can be employed with great success to perform two important sub-tasks of data integration, namely schema matching and entity resolution. By exploiting the geometric properties of the table embeddings, we implemented a heuristic that accurately finds duplicates of a given tuple or attribute by searching the vector space for the closest embeddings to the target. We executed an extensive experimental campaign in which we demonstrated that the graph architecture implemented by EmbDI can be used by generic graph embedding algorithms for performing data integration tasks; we also showed that even simple strategies for generating EmbDI embeddings are competitive with ad-hoc graph embedding algorithms, while being more efficient in practice. Finally, we demonstrated that the local embeddings prepared by EmbDI can be used instead of generic pre-trained embeddings to perform supervised entity resolution tasks and produce better results overall.

In Chapter 5, we considered the problem of tabular data imputation in presence of categorical data. We designed and implemented **GRIMP** (**G**raph embeddings for **R**elational data **IMP**utation), a data imputation algorithm that combines Graph Neural Networks with a multi-task mixed classification/regression architecture to repair tables that contain missing data. Through GNNs, we combined pre-trained node features with the structural information contained in the graph to produce refined node representations to be fed to the multi-task architecture. Thanks to the multi-task structure, we improved the training efficiency, enhanced the robustness of the algorithm to high error scenarios and increased the imputation accuracy for categorical data. To better integrate external information in the model training, we implemented attention-based classification tasks, with which it became possible to direct the training by feeding information about known correlations between columns. Finally, we improved the well-known MissForest algorithm by developing FunForest (**Fun**ctional Miss**Forest**), a variant of the original algorithm that can make use of known functional dependencies to direct the training of the random forest ensemble. We demonstrated experimentally that GRIMP outperforms known imputation baselines on a number of datasets, and showed that FunForest is both faster and more accurate than plain MissForest in presence of known functional dependencies.

6.1 Future Work

We describe here some possible research directions for future work.

6.1.1 EmbDI, Tabular Embeddings and Data Integration

While the architecture of EmbDI that has been described in this thesis is already well developed on its own, there are a number of research directions that are yet to be explored; these research options range from improvements to the implementation, to extensions of the algorithms to leverage external information, to expansions of the scope of already developed experiments. We are also considering the idea of querying through embeddings as a possible future research direction.

For the first category, we are planning to modify the structure of the graph on two different scopes. Simpler extensions would entail modifying the graph by introducing new edges or weights to be used in the random walk generation. Preliminary results in this direction were inconclusive: while the addition of new edges does not seem to bring immediate positive results, edge weights can improve the embeddings quality, at the cost of a much broader research space for the optimization of the weights. More extensive modifications would require a major redesign of the code to make use of the GNN architectures that we exploited with success in Chapter 5. Further extensions of the scope of EmbDI include modifications that would allow to perform many-to-many matches, as well as improving the scalability of the data integration infrastructure from two to n tables, e.g., in a data lake scenario.

For the second, we are looking for solutions to the problem of including external, known information in the table graph so that the resulting embeddings can make use of both the system architecture and said external information (e.g. textual data, or data contained in knowledge bases). In [203], the EmbDI architecture has been successfully extended to the KB case, so this research direction has already shown promise.

Querying with Table Embeddings

A further direction of study that can be taken to make use of the geometric properties of embeddings can be described as operating "queries over the embeddings". With this we describe algebraic operations executed on the embeddings that can be used to extract information from the vector space. These operations are akin to the famous example $king - man + woman \implies queen$ from the Word2Vec paper [37], as we are combining vectors corresponding to different concepts, then search for the closest points to the combined vector.

We illustrate some of the operations that can be performed through examples. In the following listings, we take the vector of two movie directors (Quentin Tarantino and Steven Spielberg) and we search for their movies in the vector space: we can do so by subtracting from the vector associated to their name the vector for "director", then adding the vector for "title". This listing for Quentin Tarantino:

```
v1 = model.get_vector('quentin_tarantino')
v2 = model.get_vector('title')
v3 = model.get_vector('director')
v_res = v1 + v2 - v3
res = print_most_similar(model, v_res, topn=50)
```

produces as output

```
             quentin_tarantino       0.71
               kill_bill_vol_2       0.52
               kill_bill_vol_1       0.52
                  pulp_fiction       0.50
           from_dusk_till_dawn       0.49
                         title       0.48
              django_unchained       0.48
                reservoir_dogs       0.48
             the_hateful_eight       0.48
                    grindhouse       0.48
          inglourious_basterds       0.47
                   double_dare       0.47
                  jackie_brown       0.46
```

A similar listing for Steven Spielberg:

```
v1 = model.get_vector('steven_spielberg')
v2 = model.get_vector('title')
v3 = model.get_vector('director')

```

```
v_res = v1 + v2 - v3
res = print_most_similar(model, v_res, topn=50)
```

produces as result:

```
                  steven_spielberg       0.73
           raiders_of_the_lost_ark       0.53
                     jurassic_park       0.51
                             title       0.51
                  the_color_purple       0.50
close_encounters_of_the_third_kind       0.49
                              jaws       0.49
                         war_horse       0.48
                      the_terminal       0.48
indiana_jones_and_the_last_crusade       0.47
                     night_gallery       0.46
               saving_private_ryan       0.46
              voices_from_the_list       0.46
```

In this example we search for the movie “Pulp Fiction” and display the closest vector to the result:

```
test_name = 'pulp_fiction'
v1 = model.get_vector(test_name)
v2 = model.get_vector('title')
v3 = v1 - v2
r = model.similar_by_vector(v3)
```

```
      pulp_fiction       0.79
 quentin_tarantino       0.42
            199409       0.37
       phil_lamarr       0.36
      bruce_willis       0.31
       eric_stoltz       0.31
```

While simple, these examples show that it is possible to rely on the geometric properties of vectors to “probe” the vector space for the values that are most closely related to the vector under observation. However, it is important to note that this approach strongly depends on the position of some values in the vector space, and on how nodes are modeled. Nodes that appear very frequently in the training set have a large number of neighbors in the graph, which “averages out” their position in space. Comparatively rarer nodes (e.g., movie actors that appear in only few titles) are much more correlated to the values they share a tuple with. Moreover, these operations rely solely on information that is already present in the database, which limits the extent of new information that can be gained from these operations.

6.1.2 GRIMP, Data Imputation with GNNs

For what concerns the data imputation part of this work, there are two main avenues to pursue. On one side, we are looking for improvements over the current GRIMP architecture. The architecture of GRIMP is quite complex and features a large number of parameters, whose influence on the final imputation accuracy is noticeable and non-obvious. We are planning to optimize GRIMP by introducing hyperparameter tuning in the pipeline, so that it would become possible to select the optimal configuration for each dataset. Besides looking for means of improving the imputation performance, we are planning to improve the explainability of the model to better understand the overall behavior of the algorithm and how different parameters may influence the imputation performance.

On the other side, we are interested in carrying out a broader study of DL-based imputation models to observe whether it is possible to estimate an upper bound on the imputation accuracy of a generic imputation algorithm on a given, dirty dataset. Indeed, we observed that the imputation accuracy of various algorithms can be approximated by studying the distribution and frequency of distinct values in the dataset. Our hypothesis is that imputation algorithms are heavily biased towards frequent values, and thus have a higher imputation accuracy for them, while underperforming on the rarer values. We model this hypothesis by defining the "expected fraction of incorrect imputations" of a value v in a column A as $E_v^A = 1 - f_v$, where f_v is the frequency of value v in column A: with this simple definition, the expectation is that a generic imputation algorithm will fail most imputations on rare values, while performing better on "easier" values.

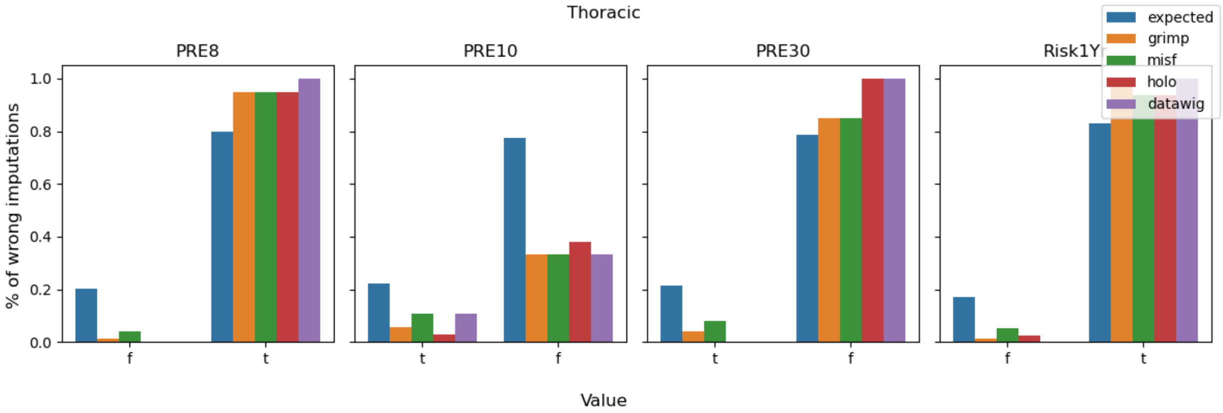


Figure 6.1: Distribution of wrong imputations in "Thoracic" dataset.

In Figures 6.1 and 6.2, we plot the results of our experiments on the Thoracic and Contraceptive datasets, respectively. Each subplot contains data relative to a single attribute in the table, with the fraction of incorrectly imputed values on the y-axis (a value of 0 on the y-axis denotes perfect imputation accuracy for that value, thus the lower the bars, the better), and the different values in the attribute domain on the x-axis, sorted in descending order by frequency so that rare values in the attribute domain are on the right side of the plot. The blue bar (labeled as "expected") displays the expected fraction of erroneous imputations given a value's frequency as defined above, while the other bars show the actual fraction of erroneous imputations as they are produced by the different imputation algorithms we have used in the experimental section. Our hypothesis seems to be confirmed by the results: all algorithms tend to have a very high accuracy on frequent values, while failing frequently on rarer values. While different algorithms may exhibit different behaviors over different columns, there is a clear trend that is common

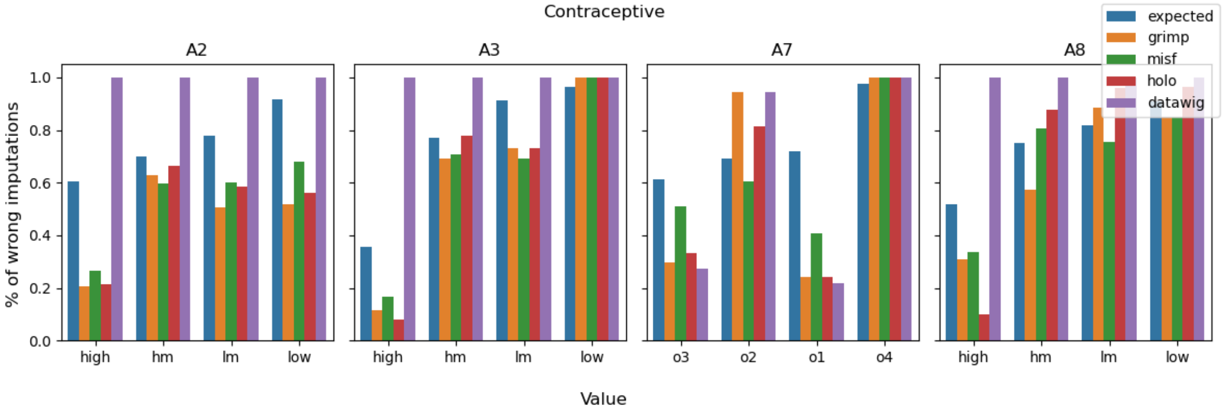


Figure 6.2: Distribution of wrong imputations in “Contraceptive” dataset.

to all of them.

While these results are clearly preliminary, it is nonetheless interesting to observe how algorithms that employ unrelated technologies tend to exhibit remarkably similar behaviors in the performance of the final imputation task. We plan to delve deeper in this subject, with the objective of reaching a better understanding of how these systems model imputation and finding a heuristic that would allow to provide advice on what algorithm to use given a dataset’s characteristics.

Résumé

Présentation du problème

Au début, il y avait des données. D'énormes, de vastes réservoirs de données à exploiter par les utilisateurs, les organisations, les chercheurs, les entreprises. Malheureusement, la plupart de ces données sont entachées d'un mauvais formatage, de fautes de frappe, de doublons, de valeurs manquantes et erronées [2]. La conservation des données est un processus fastidieux qui prend du temps. Malheureusement, ce processus ne peut être négligé si l'on veut éviter des résultats négatifs ou trompeurs dans les applications en aval [4, 5]. La curation des données est un domaine de recherche remarquablement pratique : en effet, la gestion des données sales et incohérentes est un problème auquel il faut faire face quotidiennement dans l'industrie. Selon une statistique largement citée, les data scientists passent 60 à 80 Cela fait de la conservation des données un problème de grande envergure et largement étudié [9], qui est rendu plus difficile par la charge importante qu'il représente pour les humains [10], à la fois dans l'étape de conservation des données et dans les applications en aval.

L'accès propre et ouvert aux données a un certain nombre de ramifications en fonction de l'utilisation qui doit être faite des données. Pour certaines applications, les données sales ne peuvent pas être utilisées du tout parce que des valeurs erronées ou manquantes introduiraient des biais inacceptables ; dans l'industrie, cela pourrait conduire à des décisions erronées qui entraîneraient des pertes de capital en aval [11] ; dans le domaine médical, les données sales pourraient conduire à des thérapies incorrectes [4, 12]. Dans d'autres situations, les données peuvent être "sales" dans le sens où plusieurs sources de données contiennent des informations similaires qui doivent être combinées dans une vue "nettoyée" en consolidant les doublons : cela peut être le cas lorsque différentes entreprises fusionnent et doivent combiner leurs bases de données. La conservation des données est un sujet omniprésent qui touche toutes sortes de disciplines centrées sur les données. Pour ces raisons, les avancées dans le domaine de la curation des données peuvent directement toucher un large éventail d'applications [13], tout en restant d'une importance capitale dans le domaine des bases de données où la plupart de ses recherches trouvent leur origine.

Il n'existe pas de solution simple à ce problème. Les contraintes (contraintes de domaine, dépendances fonctionnelles, dépendances fonctionnelles conditionnelles, contraintes de refus, etc.) sont utilisées pour faire respecter les propriétés des données et pour identifier les erreurs en observant quelles valeurs violent lesdites contraintes. Si certaines de ces contraintes peuvent être introduites pendant la phase de conception de la base de données (par exemple en spécifiant le type de données et le domaine d'un attribut, puis en appliquant les contraintes), cela n'est pas toujours possible.

Il est toutefois possible de détecter et de générer ces contraintes et ces règles en observant les données. Des contraintes de différents types peuvent être employées pour faire respecter certaines propriétés dans les données, pour extraire des informations ou pour faire correspondre des entités [14]. Ces contraintes sont généralement créées à la main, ou doivent être découvertes en observant les données. La découverte de telles contraintes a fait l'objet de nombreux travaux allant de la découverte de règles [15], à la découverte de FDs [16, 17], de FDs conditionnelles [18–20], de contraintes de déni (DCs) [21]. Cette collection d'outils peut permettre de détecter et de corriger certaines des erreurs que l'on peut trouver dans un jeu de données sale [4].

Malheureusement, ces outils ne sont pas parfaits : les contraintes ne sont pas toujours applicables à toutes les situations, les dépendances fonctionnelles n'existent pas toujours, la génération de règles peut devenir trop coûteuse, les règles générées automatiquement peuvent être trop nombreuses pour être vérifiées par un observateur humain [14]. De plus, ces outils sont souvent fortement dépendants de l'expertise humaine [3, 22–26] : c'est un enjeu important, car les solutions qui reposent sur l'interaction humaine sont longues, coûteuses et peu évolutives pour de grandes quantités de données [27].

La présence de données sales ou manquantes est un problème majeur, car la génération de contraintes sur des données sales peut conduire à des règles incorrectes [28–30]. Chaque ensemble de données sales a ses propres idiosyncrasies et bizarreries, ce qui signifie que les solutions ad-hoc sont très souvent la seule solution possible. Même lorsque les données ne sont pas manquantes, les valeurs peuvent ne pas être propres en raison de fautes de frappe (par exemple, ”balck” au lieu de ”black”), de problèmes d'encodage (”10.002” signifie-t-il ”10 mille et 2” ou ”10 pointe 000” ?), de formatage (”The Who” dans un ensemble de données, ”Who, The” dans un autre).

Le type de données des valeurs d'un ensemble de données peut également poser problème : si les erreurs numériques peuvent être traitées par des approches naïves telles que la régression linéaire ou modélisées par des systèmes plus sophistiqués basés sur l'apprentissage profond (Deep Learning), les données catégorielles (c'est-à-dire la plupart des données textuelles) ne peuvent pas être traitées de cette manière. Cela signifie soit que les valeurs catégorielles sont converties sous forme numérique, soit que des systèmes uniquement catégoriels doivent être conçus. Cette dernière solution est souvent la seule viable pour la majorité des cas dans lesquels le nombre de catégories devient trop important pour être traité par un codage à un coup ou des méthodes similaires [36].

Deuxièmement, dans les tableaux relationnels réels, les valeurs catégoriques sont omniprésentes, ce qui est très problématique car de nombreuses méthodes basées sur la DL ne peuvent pas traiter les valeurs catégoriques dans leur forme brute, mais nécessitent leur encodage dans une représentation numérique (généralement un encodage à un coup). Cela peut bloquer les architectures plus simples en provoquant une explosion du nombre de caractéristiques, ce qui entraîne des problèmes d'évolutivité (la fameuse ”malédiction de la dimensionnalité”), un problème malheureusement très fréquent puisque même les petits ensembles de données peuvent avoir des domaines catégoriels très vastes. Enfin, alors que les données homogènes (comme le texte, les images, la parole) peuvent être encodées en employant des algorithmes d'embeddings, ce n'est pas le cas pour les données tabulaires : les données tabulaires sont intrinsèquement discrètes dans leur structure, et peuvent contenir des valeurs numériques ou catégorielles, dont les distributions sont complètement différentes et donc très difficiles à combiner.

Dans le cadre de notre travail, nous fournissons des solutions qui peuvent aborder ces trois points : nous nous appuyons sur des méthodologies et des systèmes issus de différents domaines (de l'intégration de mots à l'intégration de graphes, en passant par les GNN et l'apprentissage multi-tâches) pour produire des solutions non supervisées capables de résoudre les problèmes d'intégration et d'imputation de données, même lorsque les données d'entrée sont de nature mixte. Comme nous l'avons mentionné, les systèmes de type "human-in-the-loop" peuvent être utilisés avec succès dans le domaine de la conservation des données. Cependant, le facteur humain est à la fois une bénédiction et une malédiction pour ces méthodes : les experts humains peuvent sélectionner le meilleur ensemble possible de règles, de contraintes et de corrections à appliquer aux données observées; d'un autre côté, il y a souvent beaucoup trop de données pour qu'un humain puisse les traiter correctement [14]. Un autre inconvénient est le fait que les experts en matière de conservation des données ne sont pas nécessairement experts dans la manière de mettre leurs recommandations dans le code, c'est-à-dire comment "informer le système" de ce qu'il devrait faire [6–8]. Pour ces raisons, nous choisissons de travailler avec des méthodes non supervisées : tous les systèmes dont nous parlerons dans cette thèse sont non supervisés, c'est-à-dire qu'ils sont formés exclusivement sur les données disponibles et n'ont pas besoin d'étiquettes, de disques d'or ou de règles définies par l'homme pour mener à bien leurs tâches. Bien que nous nous concentrions sur des solutions non supervisées, les encastrements qu'elles produisent peuvent toujours être utilisés par des méthodes supervisées et conduire à des améliorations du résultat final. Nous explorons ce point plus en détail dans notre chapitre sur les embeddings tabulaires, comment les générer et comment les utiliser. Nous avons développé nos systèmes de manière à ce qu'il soit possible de les appliquer sans avoir de grandes connaissances en codage, de sorte qu'un expert du domaine soit en mesure d'exécuter le système, puis de corriger le résultat si nécessaire. Nous publions tous les codes que nous avons développés pour que les praticiens puissent les utiliser.

Pour ce travail, nous nous concentrerons principalement sur les données relationnelles sous forme de tableaux. Les ensembles de données que nous considérons peuvent contenir des types de données catégoriques, numériques ou mixtes. Les données catégorielles ne peuvent prendre qu'un nombre limité de valeurs différentes qui peuvent être séparées en "catégories". Si les données catégorielles sont normalement textuelles, il n'est pas nécessaire qu'elles le soient : parfois, les attributs à valeur entière doivent être traités comme des attributs catégoriels (par exemple, lorsqu'on travaille avec des codes postaux ou des identifiants numériques). En raison de leur nature discrète, les valeurs catégorielles (c'est-à-dire les valeurs qui peuvent être assignées à une catégorie, comme les chaînes de caractères ou les identifiants numériques) sont particulièrement problématiques en apprentissage automatique, car la plupart des données sont stockées dans des bases de données. particulièrement problématiques en apprentissage automatique, car la plupart des modèles ont besoin de caractéristiques numériques pour fonctionner. L'absence de bonnes réponses à ce problème est ce qui nous a motivés à nous concentrer spécifiquement sur les ensembles de données qui comportent une majorité de données catégorielles : nous avons ensuite conçu un ensemble d'algorithmes et de systèmes qui seraient non seulement capables de traiter les valeurs catégorielles, mais qui "épouseraient" leur caractéristique discrète à travers une représentation discrète, à savoir un graphique. Pour appliquer les modèles ML aux données catégorielles, nous avons choisi d'utiliser des "modèles d'espace vectoriel" (VSM) pour attribuer des représentations vectorielles à ces dernières, afin qu'il soit possible d'employer des systèmes numériques pour les données catégorielles.

Les modèles d'espace vectoriel décrivent des modèles algébriques permettant de représenter des entités *vecteurs* (ou *embeddings*) dans un espace vectoriel à haute dimension [37]. Ces méthodes se sont avérées très efficaces pour coder toutes sortes d'entités, des modèles de langage [37, 67] aux graphes [85]. Dans les modèles d'espace vectoriel, chaque entité de l'espace se voit attribuer une position dans l'espace qui est décidée par rapport à toutes les autres entités de manière à ce que les entités corrélées soient placées très près les unes des autres dans l'espace vectoriel : ceci est modélisé comme un problème d'optimisation qui est résolu lorsque le modèle est entraîné. Il est essentiel que des propriétés géométriques s'appliquent à la relation entre ces vecteurs : les distances numériques entre les chaînes peuvent être mesurées et il est possible de "naviguer" dans l'espace vectoriel en effectuant des opérations vectorielles. Dans les premières applications, ces VSM ont été utilisés pour générer des représentations vectorielles de mots et de documents. L'introduction de nouveaux cadres tels que word2vec a toutefois ouvert la porte à leur utilisation pour une grande variété d'applications. En effet, un avantage fondamental des modèles d'incorporation est que tout type d'objet peut être transformé en un vecteur numérique, à condition que les entités puissent être représentées d'une manière appropriée au modèle d'apprentissage. Notre travail s'appuiera fortement sur cette caractéristique en utilisant des techniques d'incorporation pour coder des données tabulaires catégorielles, puis en agissant sur les vecteurs numériques résultants pour effectuer d'autres opérations qui ne seraient pas possibles sur les données originales. Les vecteurs d'incorporation permettent d'atténuer ce problème en convertissant ces valeurs catégorielles en vecteurs à hautes caractéristiques relativement bien gérés, ce qui permet de réduire le nombre total de dimensions requises par le modèle.

La deuxième raison qui nous pousse à utiliser les représentations VSM ("embeddings") est la façon dont elles peuvent être entraînées de manière non supervisée à partir d'un corpus d'entraînement correctement préparé. Cela permet de développer des outils de prétraitement appropriés qui, compte tenu d'un ensemble de données cible, peuvent produire une représentation d'entrée adéquate (qu'il s'agisse d'un corpus d'entraînement basé sur des phrases ou d'une représentation vectorielle enrichie en caractéristiques des valeurs de tables) que l'algorithme d'intégration doit "digérer". Nous répondons ainsi au deuxième problème sur lequel nous nous sommes concentrés : ces algorithmes peuvent être exécutés avec peu d'intervention d'experts humains, hormis une configuration et un prétraitement appropriés.

Alors que les embeddings sont réputés pour leur application générique, les données tabulaires sont connues pour être difficiles à modéliser à l'aide de modèles d'espace vectoriel : alors que le langage naturel est intrinsèquement redondant et structuré, les données tabulaires ne partagent pas la même structure et sont beaucoup moins redondantes, surtout si l'on ne considère qu'une seule table à la fois. Les données tabulaires contiennent également des structures syntaxiques qui sont absentes du langage naturel, à savoir des tuples, des attributs et un concept d'"appartenance" partagé par les tokens trouvés dans ces structures. De plus, bien qu'il soit possible d'atténuer certains des problèmes causés par les données catégorielles, les modèles basés sur les techniques d'apprentissage profond souffrent fortement de leurs caractéristiques : les domaines à grande cardinalité et les distributions de données fortement déséquilibrées sont très difficiles à modéliser, ce qui devient un problème majeur lorsque des problèmes de classification entrent en jeu. Nous explorons ce problème plus en détail dans la partie de ce travail consacrée à l'imputation des données, où nous introduisons l'apprentissage multi-tâches pour réduire l'impact des grands domaines catégoriels dans l'entraînement des modèles de DL.

Dans ce travail, nous proposerons une nouvelle procédure **modulaire** pour traiter la curation des données (spécifiquement **l'intégration des données** et **l'imputation des données**) qui s'appuie sur des représentations non traditionnelles des données tabulaires et des modèles basés sur le Deep Learning. Nous fournirons des exemples de la manière de représenter des tableaux relationnels **sous forme de graphe**, et de générer des représentations vectorielles pour les nœuds du graphe. Ces représentations de graphes sont construites de telle sorte que les nœuds de graphes incluent les tuples et les attributs des tables, permettant ainsi de générer organiquement des représentations vectorielles pour ces entités également : c'est un changement marqué par rapport aux travaux précédents, dans lesquels les représentations de colonnes et de lignes étaient générées en combinant les vecteurs des valeurs dans la colonne ou la ligne respective [70].

L'un des principaux avantages de cette méthodologie modulaire est qu'il devient possible de "brancher et de jouer" différentes méthodes : cela permet de tirer parti des recherches menées dans les différents domaines pour exploiter différentes méthodes de pointe et ainsi améliorer les performances du système. Nous en donnerons des exemples en utilisant différentes représentations graphiques des tableaux, ainsi que plusieurs méthodes de génération d'incorporations de nœuds de graphes (notamment basées sur DeepWalk [41, 42] et sur les réseaux neuronaux graphiques [43, 109]). Par exemple, en traitant les marches aléatoires comme des phrases, il devient possible de tirer parti d'algorithmes préexistants développés à l'origine pour des problèmes de traitement du langage naturel (NLP) dans un domaine totalement étranger. Les représentations vectorielles bénéficient de propriétés géométriques qui peuvent être exploitées pour effectuer un certain nombre d'opérations : ici, nous les utilisons pour effectuer la résolution d'entités, la correspondance de schémas et l'imputation de données. Enfin, une caractéristique clé des systèmes que nous proposons est qu'ils sont **unsupervised**, de sorte qu'il n'y a pas d'exigence stricte sur la participation humaine externe pour exécuter la formation.

Il est important de noter que l'apprentissage profond n'est pas une solution miracle pour toutes les applications, et qu'il souffre de certains inconvénients qui doivent être pris en considération lors de l'application de ces solutions, ou de toute autre solution d'apprentissage profond. Nous explorerons certaines de ces limites dans la dernière partie de ce manuscrit.

Ce travail se concentrera sur l'attaque du problème de la curation des données sous deux angles : **apprentissage non supervisé** et **modélisation de données catégorielles**. Nous utilisons le premier pour contourner le problème de l'humain dans la boucle : en concevant des modèles non supervisés, il devient possible d'effectuer de multiples tâches de curation de données sans avoir besoin de données fournies par l'humain autres que les ensembles de données de départ et la localisation des valeurs manquantes. Nous concevons ensuite nos modèles en fonction de la nécessité de modéliser les données catégorielles afin d'étendre l'utilisation de l'apprentissage profond dans la conservation des données : en effet, les modèles basés sur l'apprentissage profond sont plus sensibles aux données catégorielles, précisément parce que ces dernières sont plus difficiles à modéliser numériquement. Dans nos contributions, nous introduisons une nouvelle représentation des données relationnelles qui permet de générer élégamment des représentations numériques de données catégorielles : par la transformation d'une table relationnelle en un graphe , il devient possible de tirer parti des techniques d'intégration développées à l'origine pour le traitement du langage naturel afin de générer des représentations vectorielles des nœuds du graphe. Nous concevons nos systèmes de manière modulaire, afin qu'il soit possible d'appliquer différentes stratégies pour représenter les tableaux sous forme de graphes, pour générer les incorporations et pour travailler avec les incorporations résultantes pour différentes tâches.

Nous introduisons notre travail en explorant le contexte théorique et les travaux antérieurs dans les différents domaines que nous touchons. Parmi les technologies fondamentales, nous discutons des progrès réalisés en matière d'intégration de mots, de tableaux et de graphes. Nous décrivons les systèmes et les méthodologies utilisés pour effectuer les tâches d'intégration de données de résolution d'entités et de correspondance de schémas. Nous nous penchons sur le problème de l'imputation des données et sur l'état actuel des méthodes d'imputation des valeurs manquantes dans un ensemble de données. Enfin, nous présentons nos contributions à l'intégration des données et à l'imputation des données.

Intégration des données

Notre première contribution consiste à présenter EmbDI (**EMB**eddings for **D**ata **I**ntegration), une nouvelle architecture pour la restructuration des données tabulaires qui convertit les tableaux relationnels en une représentation graphique qui est ensuite utilisée pour générer des représentations vectorielles numériques distribuées du contenu du tableau : Ceci représente une méthode nouvelle et élégante pour générer des représentations distribuées de données tabulaires, un problème notoirement difficile [39, 55]. Une solution courante à ce problème consiste à s'appuyer sur des modèles d'enchâssement pré-entraînés pour générer des enchâssements pour les valeurs de la table, soit en transformant le vecteur lui-même [70, 71], soit en affinant le modèle [204]. Cependant, ces approches présentent deux inconvénients majeurs. Tout d'abord, le corpus d'entraînement peut ne pas contenir toutes les données présentes dans la table, ce qui conduit soit à l'échec de la génération d'enchâssements de tokens hors vocabulaire, soit à la construction de nouveaux enchâssements basés sur des combinaisons de tokens éventuellement non liés. Deuxièmement, le fait

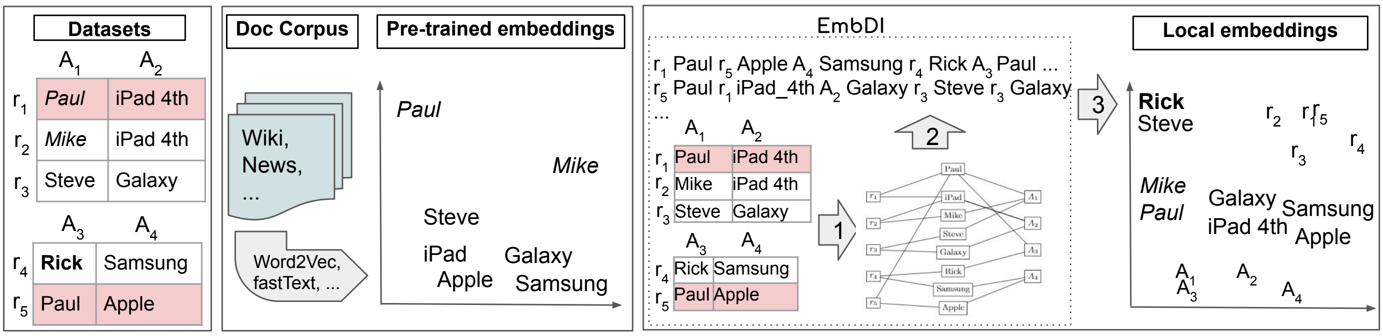


FIGURE 6.3 : Illustration d'un espace vectoriel simplifié appris à partir de texte (approches préalables) et de données (EMBDI).

qu'une valeur de la table puisse être trouvée dans le corpus d'entraînement peut encore conduire à des problèmes dans l'entraînement, parce que l'incorporation pré-entraînée pour cette valeur n'est pas consciente des corrélations qu'elle pourrait avoir avec d'autres tables, mais en même temps est biaisée par des valeurs qui pourraient ne pas apparaître du tout dans la table cible.

Nous répondons à ce problème en implémentant EMBDI : EMBDI est un système non supervisé pour la génération d'enchâssements locaux de données tabulaires qui est adapté au problème de la réalisation de tâches d'intégration de données. Avec EMBDI, nous introduisons une méthode pour générer des *représentations locales* des valeurs de la table, et en même temps nous produisons des embeddings pour les entités structurelles de la table, à savoir les tuples et les attributs : c'est un autre avantage de EMBDI par rapport aux modèles pré-entraînés car il permet de préparer organiquement les embeddings des lignes et des colonnes pendant le temps de formation et en relation avec les valeurs qui sont contenues dans ces lignes et colonnes. Avec les modèles pré-entraînés, les enchâssements de tuple et d'attribut doivent être générés en combinant les vecteurs des valeurs du tuple ou de l'attribut par addition, moyennage ou autre moyen.

EmbDI

Le pipeline de EMBDI (décrit dans la figure 6.3) consiste en trois blocs distincts avec des interfaces bien définies entre eux : (i) un **composant graphique**, qui est suivi d'un (ii) **composant de génération aléatoire de graphes** qui prépare le corpus d'entraînement pour le composant final, (iii) un **algorithme d'entraînement pour les incorporations de mots** tel que WORD2VEC [37]. Il s'agit d'une architecture modulaire dans laquelle les techniques spécifiques mises en œuvre dans les blocs peuvent être remplacées par d'autres algorithmes si nécessaire. Cette caractéristique est explorée dans la section expérimentale, dans laquelle nous testons les performances de deux algorithmes d'intégration de graphes (NODE2VEC [42] et HARP [99]) entraînés sur le graphe de table généré par EMBDI. En outre, cela permet à EMBDI d'être raisonnablement à l'épreuve du temps en autorisant l'utilisation de nouveaux algorithmes au lieu de ceux qui sont déjà disponibles dans l'implémentation que nous décrivons.

Composant graphique
La structure de base de EmbDI s'articule autour de la construction d'un *graphe tripartite* qui unifie la structure d'un tableau avec son contenu : les lignes, les colonnes et les valeurs des cellules sont toutes affectées à des nœuds, avec des arêtes reliant un tuple à son contenu, et une colonne à toutes les valeurs de son domaine. Cette méthode est avantageuse pour deux raisons : premièrement, en parcourant le graphe à l'aide de marches aléatoires, il devient possible d'encoder les relations de même rang, de même attribut et de proximité dans les phrases qui sont ensuite utilisées pour l'apprentissage des mots incorporés ; deuxièmement, les rangées et les colonnes sont modélisées directement avec les valeurs de la table, sans qu'il soit nécessaire de les produire par une étape de post-traitement qui combine les incorporations préparées pour les valeurs de la table. Nous développons et testons plusieurs solutions différentes pour traiter les valeurs de cellules qui contiennent de longues chaînes de caractères afin d'exploiter les informations contenues dans la cellule et de maintenir une représentation adéquate de la chaîne de caractères complète.

Génération de structures intégrées avec des marches aléatoires
Le deuxième module est un algorithme de génération de marches aléatoires qui est exécuté sur le graphe de la table, qui traverse le graphe plusieurs fois et collecte tous les chemins résultants. Bien que de multiples stratégies de génération puissent être appliquées, de la sélection complètement aléatoire des nœuds à la sélection pondérée, en passant par le type node2vec, nous constatons que les marches aléatoires simples donnent des résultats comparables aux méthodes plus sophistiquées, tout en étant moins coûteuses à préparer. Pour générer la représentation distribuée de chaque nœud du graphe, nous produisons un grand nombre de marches aléatoires et les rassemblons dans un corpus d'entraînement où chaque marche aléatoire correspondra à une phrase. L'utilisation de graphes et de marches aléatoires nous permet d'avoir un ensemble de voisinages plus riche et plus diversifié que ce qui serait possible en codant un tuple comme une phrase unique. Notre approche est agnostique par rapport au type spécifique de marche aléatoire utilisé, différents choix produisant différents enchâssements. Nous utilisons des marches aléatoires uniformes dans la plupart de nos expériences pour garantir de bons temps d'exécution sur de grands ensembles de données, tout en fournissant des résultats de haute qualité. Nous comparons des marches aléatoires alternatives dans les expériences.

Un certain nombre d'approches antérieures, telles que DeepER [70] ou DeepMatcher [71], apprennent uniquement des enchâssements pour les tokens et obtiennent ensuite des enchâssements pour les tuples en les moyennant ou en les combinant à l'aide d'un RNN. en les moyennant ou en les combinant à l'aide d'un RNN. L'utilisation de nos marches aléatoires sous forme de phrases fournit des informations supplémentaires sur le voisinage de chaque nœud, qui ne seraient pas aussi facilement obtenues en utilisant uniquement le format de données structurées. Les phrases générées sont regroupées pour construire un corpus qui est utilisé pour entraîner l'algorithme d'intégration. Comme pour la génération de la marche aléatoire, il n'y a pas d'exigence stricte quant à l'algorithme d'intégration des mots à utiliser. Nous nous appuyons sur la pléthore d'algorithmes d'incorporation efficaces tels que word2vec, GloVe, fastText, etc. De manière générale, ces techniques peuvent être classées en deux catégories : celles basées sur les mots (comme word2vec) et celles basées sur les caractères (comme fastText).

Applications d'EmbDI

Applications de l'EmbDI en tableau unique
EmbDI peut être appliqué à des scénarios à une ou deux tables pour différentes raisons. Dans un scénario à table unique, EmbDI peut produire des incorporations de haute qualité pour les valeurs de la table et ses composants structurels (lignes et colonnes). Ces incorporations sont *local*, et encodent les informations de proximité des valeurs dans l'ensemble de données d'origine : cela en fait des remplacements appropriés pour les applications qui nécessiteraient autrement des modèles génériques pré-entraînés (par exemple [71]). Nous démontrons ces avantages dans la dernière partie du document, où nous utilisons les embeddings EmbDI pour initialiser notre système d'imputation de données. La génération d'embeddings permet de produire des représentations vectorielles numériques distribuées du contenu de la table. Par conséquent, la représentation discrète peut être utilisée directement pour représenter des types de données discrets et catégoriques. Nous montrons comment générer les embeddings sur le graphe et comment étudier leur qualité.

De la même façon que l'on peut utiliser les enchâssements de mots "réguliers" pour effectuer des opérations géométriques afin de trouver des similarités complexes entre les mots, les enchâssements de mots peuvent être utilisés pour sonder l'espace vectoriel afin de trouver les voisins d'un nœud donné, qu'il s'agisse d'un nœud de valeur ou d'un nœud de colonne/ligne. Cela permet, par exemple, de sonder l'espace des embeddings pour trouver des valeurs similaires à celles qui sont données, ou de traverser l'espace des embeddings en exécutant des opérations algébriques entre les vecteurs. . Nous effectuons des expériences supplémentaires pour montrer le potentiel d'utilisation de l'information stockée dans les encastrements pour de nouvelles applications, à savoir l'interrogation géométrique des encastrements et l'appariement de jetons. *Cette contribution relève le défi de la génération d'enchâssements tabulaires pour des données catégorielles.*

Applications de EmbDI à deux tableaux
Le scénario à deux tables peut être utilisé pour modéliser des problèmes d'intégration de données. Nous mettons en œuvre le scénario à deux tables en construisant un graphe au-dessus de la concaténation des deux tables. Ce faisant, les incorporations du graphe encodent des informations sur la structure des deux tables. Les propriétés géométriques de ces imbrications peuvent être exploitées une fois de plus, cette fois pour effectuer la **Résolution d'entité** (ER) et **Schema Matching** (SM). (SM). Nous développons EmbDI (Embeddings for Data Integration) pour mettre en œuvre ces contributions. Le problème de la résolution d'entités consiste à rechercher des sources de données avec des domaines similaires pour identifier les entités qui se trouvent dans les deux sources, mais qui sont représentées différemment. Pour le Schema Matching, nous cherchons plutôt une méthode qui, étant donné des tables qui partagent le domaine, peut identifier les attributs qui sont présents dans les deux tables et les faire correspondre. Dans ce travail, nous démontrons comment EmbDI peut réaliser ces deux tâches de manière élégante en s'appuyant sur les propriétés géométriques des incorporations mentionnées ci-dessus.

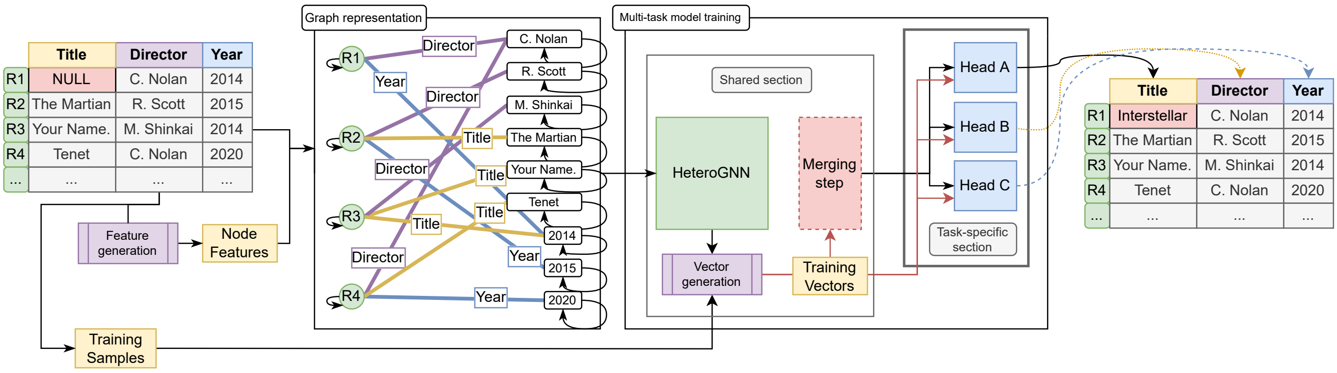


FIGURE 6.4 : Vue d'ensemble de l'architecture GRIMP.

Ceci est rendu possible par l'architecture de graphe de EmbDI, qui génère organiquement des embeddings pour les tuples et les attributs. Pour la tâche ER, nous développons un algorithme heuristique qui recherche les doublons d'un tuple donné en sondant l'espace vectoriel et en recherchant les tuples les plus proches selon une mesure de similarité numérique : cette heuristique est très précise et obtient des résultats de pointe pour les méthodes non supervisées. Dans le cas du SM, nous utilisons une méthode heuristique similaire pour faire correspondre les colonnes qui sont proches les unes des autres dans l'espace. Dans la section expérimentale, nous testons EmbDI et le comparons à deux autres algorithmes d'intégration de graphes (node2vec [42] et HARP [99]) dans les tâches ER et SM : cette approche démontre une fois de plus l'utilité de l'architecture de EmbDI, en vertu de la façon dont il devient possible d'employer des algorithmes qui sont conçus pour des tâches non liées au problème de la génération d'intégration de tableaux.

Dans notre campagne expérimentale, nous présentons les performances de EmbDI dans les tâches ER et SM, à la fois dans le cas non supervisé par défaut et en tant que corpus pré-entraîné pour les méthodes supervisées : pour ce dernier scénario, nous utilisons DeepMatcher [71] pour comparer les incorporations de EmbDI aux incorporations fastText pré-entraînées et nous observons que les incorporations locales générées sur les données à disposition surpassent toujours les incorporations pré-entraînées. Nous enrichissons la section expérimentale avec une collection de figures qui démontrent comment les enchâssements peuvent coder différentes relations dans différentes dimensions, et comment cela peut expliquer l'efficacité démontrée dans les tâches d'intégration de données. Nous publions également le code de EmbDI pour que les praticiens puissent l'utiliser. Avec EmbDI, nous apportons une réponse au problème de l'intégration non supervisée de données sur des tables relationnelles de type mixte.

Imputation de données

Tout en restant dans le domaine de la curation des données, nous passons ensuite au problème de l'imputation des données, c'est-à-dire au remplacement des valeurs manquantes et erronées dans un tableau sale par des valeurs correctes, sur la base des données disponibles et (si elles sont disponibles) d'informations externes. Le problème de l'imputation de données consiste à corriger les valeurs manquantes ou erronées dans une table sale, avec l'objectif final d'éliminer (ou du moins de réduire) les biais que les entrées problématiques peuvent introduire dans les données. L'imputation de données est un problème difficile, car il tourne non seulement autour du problème de la détection des erreurs, mais il est encore compliqué par le fait que, une fois l'erreur trouvée, il faut faire un *choix* sur ce qu'il faut "imputer" pour combler la vacance. Si un attribut est numérique, ce problème s'apparente à l'échantillonnage d'une variable aléatoire numérique pour l'imputation correcte ; si l'attribut est catéogène, cependant, le problème doit être modélisé comme un "choix" de la valeur correcte à imputer parmi toutes les valeurs du domaine.

Comme nous le montrons dans la thèse, l'imputation de valeurs catégorielles est problématique, notamment lorsqu'on s'appuie sur des modèles basés sur l'apprentissage profond. En effet, l'imputation de données catégorielles peut être modélisée comme un problème de classification multi-classes, où chaque imputation devient le choix d'une seule valeur parmi toutes les valeurs du domaine de valeurs de la table. Ce domaine peut devenir extrêmement vaste, ce qui ajoute à la "malédiction de la dimensionnalité" à laquelle les méthodes d'imputation axées sur les données numériques sont déjà confrontées. En outre, si d'une part les problèmes déjà mentionnés de dépendance à l'égard des données étiquetées et de supervision humaine entrent en jeu, d'autre part des informations externes sont parfois disponibles, mais les modèles d'imputation ne peuvent pas les utiliser. Pour aborder ces questions, nous proposons deux solutions au problème. Tout d'abord, nous combinons l'architecture de graphes que nous avons développée pour EmbDI avec une architecture d'apprentissage multi-tâches en implémentant GRIMP (**G**raph embeddings for **R**elational data **IMP**utation), nous nous penchons ensuite sur le problème de la mise en œuvre d'informations externes dans l'apprentissage des modèles et améliorons l'algorithme d'imputation bien connu MissForest [45] en lui faisant prendre conscience de la présence de dépendances fonctionnelles au moment de l'apprentissage, en baptisant la version mise à jour FunForest (**FUN**nctional miss**Forest**).

GRIMP

GRIMP est un système d'imputation de données qui s'appuie sur une combinaison de techniques d'apprentissage profond pour imputer des données manquantes mixtes (numériques et catégorielles) dans des tables relationnelles. L'architecture d'GRIMP est basée sur deux modules. Tout d'abord, un module de prétraitement prépare la représentation graphique de la table d'entrée et le corpus d'entraînement requis par le second module. Ensuite, un composant DL basé sur l'apprentissage multi-tâches incorpore différents sous-modules spécifiques aux tâches, qui sont formés sur différents attributs de la table.

Encouragés par les résultats prometteurs obtenus avec EmbDI, nous nous appuyons à nouveau sur une représentation graphique (discrète) pour représenter le tableau sale d'entrée, qui peut contenir des données discrètes (catégorielles), ainsi que des valeurs numériques. Dans ce travail, nous tirons parti des progrès réalisés dans le domaine de l'incorporation des graphes par l'application de réseaux neuronaux graphiques (RCG), qui nous permettent de combiner des incorporations de nœuds connues avec la structure du graphe pour produire une représentation vectorielle améliorée des nœuds.

Ces représentations vectorielles améliorées sont ensuite transmises à la deuxième innovation majeure de ce travail : l'application de l'apprentissage multitâche [116, 117] au problème de l'imputation des données. Nous concevons et mettons en œuvre une architecture multitâche dans laquelle chaque sous-tâche (définie comme *head*) modélise le problème d'imputation pour un seul attribut de table, qu'il soit numérique ou catégorique : d'une part, cela permet de réaliser l'imputation en une seule fois, plutôt que d'entraîner le modèle de manière itérative et d'imputer sur une seule colonne à la fois, d'autre part, cela permet de modéliser de manière organique les attributs catégoriels, sans les conflits qui pourraient survenir entre les véritables variables numériques et les encodages numériques des variables catégorielles.

Un autre avantage de l'approche multi-tâches est qu'elle permet de répartir l'opération de classification sur plusieurs sous-tâches, chacune d'entre elles travaillant sur un domaine plus petit que ce qu'exigerait une architecture plus simple. Une architecture naïve qui utilise un GNN pour la classification pourrait être mise en œuvre en utilisant la sortie du GNN pour sélectionner une valeur unique dans l'ensemble du domaine de la table, sans distinction entre les différents attributs : cette approche est très problématique, car elle oblige le classificateur à sélectionner parmi une collection beaucoup plus grande de valeurs dans l'ensemble des données. Par conséquent, l'imputation de différents attributs en même temps prend du temps, et le modèle est beaucoup plus susceptible de prédire des valeurs qui apparaissent fréquemment dans l'ensemble de la table, même si elles n'appartiennent pas au domaine de la valeur qui est imputée à un moment donné. Un modèle multi-tâches permet de séparer la tâche de sélection afin que chaque sous-tâche ne regarde que son propre attribut à un moment donné : cela empêche le modèle de sélectionner des valeurs qui appartiennent à un domaine différent, ce qui améliore la précision de l'imputation et réduit le temps d'apprentissage. De plus, GRIMP est très robuste au bruit et peut exploiter toutes les valeurs non manquantes dans un ensemble de données sales en tirant parti, une fois de plus, de son architecture multi-tâches : grâce au fait que chaque tête est séparée de toutes les autres, il devient possible d'alimenter la même information à différentes têtes et d'obtenir des sorties multiples, différentes et valides de chacune d'elles.

GRIMP est totalement non supervisé : étant donné un ensemble de données comportant des valeurs manquantes, l'algorithme produira une version imputée de cet ensemble de données sans étiquettes ni supervision humaine. L'algorithme nécessite la préparation de l'intégration des nœuds sur l'ensemble de données sales cible afin que les nœuds du graphe aient des caractéristiques sur lesquelles le GNN peut opérer. Ces caractéristiques peuvent être générées par toute méthode appropriée, telle que EmbDI ou fastText. Cette approche évite le problème de la supervision humaine nécessaire à la préparation des données étiquetées. Cependant, il est toujours possible de fournir des informations externes à GRIMP. Cela peut être fait de deux manières : en utilisant des enchâssements pré-entraînés qui incorporent des informations externes (par exemple, les enchâssements fastText entraînés sur Common Crawl), ou en fournissant des informations au niveau des attributs telles que les dépendances fonctionnelles. Cette dernière solution est implémentée dans une couche d'attention qui combine des représentations vectorielles pré-entraînées des attributs et qui peut incorporer des informations externes pour corréler les attributs qui sont connectés via une dépendance fonctionnelle.

Le pipeline GRIMP

Le pipeline GRIMP est décrit dans la figure 6.4. Étant donné une table cible qui contient des valeurs manquantes (dans l'exemple, une table sur les films), GRIMP exécute deux opérations principales : le corpus d'entraînement (désigné par *échantillons d'entraînement*) est généré et la représentation graphique de la table est préparée. Lors de l'étape de génération du graphe, un graphe hétérogène bipartite encode les informations de la table en générant un nœud pour chaque ligne et chaque valeur, avec des arêtes typées reliant les nœuds de chaque côté. Dans le graphe, chaque enregistrement est associé à un nœud RID, chaque valeur unique de l'ensemble de données est associée à un nœud de cellule, puis les RID et les nœuds de cellule sont reliés par une arête typée. Le type d'arête est défini par l'attribut dans lequel se trouve la cellule, le nombre de types étant égal au nombre d'attributs dans la table. Cette architecture encode élégamment les caractéristiques discrètes d'une table relationnelle ; la structure hétérogène permet d'appliquer différents modules aux différents attributs, augmentant encore la granularité avec laquelle le modèle peut être adapté à un problème.

Composant multi-tâches

Le "cœur" de GRIMP est un module d'imputation qui utilise une architecture d'apprentissage multi-tâches (MTL) pour effectuer une opération de classification multi-classes sur chaque attribut de l'ensemble de données sale. L'apprentissage multi-tâches présente un certain nombre d'avantages que nous pouvons exploiter pour améliorer les résultats finaux de l'imputation. Il améliore l'efficacité de la formation en réduisant le temps de formation par rapport à un classificateur multi-classes normal et en exploitant toutes les données disponibles pour la formation, puisqu'il n'est pas nécessaire de supprimer les doublons des échantillons de formation. De plus, en ayant des têtes avec des domaines d'imputation distincts, le modèle MTL ne peut sélectionner que les valeurs trouvées dans le domaine d'une tête ; un classificateur multi-classes traditionnel sélectionnerait plutôt une valeur dans le domaine entier de la table, ce qui augmente la probabilité de sélectionner une valeur qui n'appartient pas au domaine de l'attribut : cela augmente la précision de l'imputation finale. Enfin, cette approche permet au modèle de traiter des variables de type mixte grâce au fait que chaque tête a une fonction de perte distincte : les valeurs numériques peuvent mesurer la perte à l'aide de MSE, tandis que les valeurs catégorielles utilisent plutôt l'entropie croisée. Cela facilite l'application de GRIMP à une plus grande variété de tableaux.

GRIMP utilise une architecture multi-tâches, *partage strict des paramètres* avec une *section partagée* où tous les paramètres sont partagés entre toutes les tâches, et une *section spécifique à la tâche* où chaque tâche est implémentée par une *tête* spécifique dont les paramètres sont uniques à cette tête, et ne sont pas partagés avec les autres. La taille du domaine de chaque tâche est beaucoup plus petite par rapport à la taille totale du domaine qui serait requise par un classificateur multi-classes entraîné avec le même objectif. Cela augmente la précision de l'imputation et réduit le temps de formation. La structure de chaque tête dans le sous-module spécifique à la tâche dépend de l'attribut auquel elle se rapporte : chaque tâche correspond à un attribut dans la table, et la sortie de chaque tête dépend du type de données de l'attribut (catégorique ou numérique) et de son domaine. Si l'attribut d'une tête est catégorique, alors la sortie de la tête sera un classificateur multi-classes avec autant de classes qu'il y a de valeurs distinctes dans le domaine de l'attribut ; si l'attribut est numérique, alors le classificateur a une seule sortie pour la valeur spécifique (nombre) qui devrait être utilisée pour l'échantillon d'entraînement donné.

Dans GRIMP, les têtes peuvent être implémentées à l'aide de couches linéaires qui s'appuient exclusivement sur les vecteurs d'entraînement, ou par le biais d'une couche d'attention qui combine des informations au niveau des attributs avec les vecteurs d'entraînement. Inspirés par les nombreux travaux sur les mécanismes d'attention et par les résultats très positifs obtenus dans un certain nombre de domaines par les systèmes qui les mettent en œuvre, nous ajoutons un mécanisme d'attention à l'architecture du classificateur de GRIMP, à la fois à la couche partagée et à chaque tête. Pour ce faire, nous adaptons et étendons le mécanisme d'attention employé dans [80] au problème multi-tâche.

GRIMP Couche d'attention

L'objectif de notre structure d'attention est de combiner les caractéristiques de niveau attribut et de niveau tuple afin d'exploiter les informations qu'elles contiennent. En utilisant des informations au niveau des attributs en plus des informations déjà disponibles au niveau des tuple, GRIMP améliore les performances d'imputation sur tous les ensembles de données testés. Avec le mécanisme d'attention, nous mettons en œuvre un certain nombre de stratégies différentes pour collationner l'information qui se trouve dans la représentation vectorielle des valeurs de la table (telle que préparée par le GNN) avec l'information qui se trouve dans les vecteurs pré-entraînés générés pour les attributs. La couche d'attention est la manière dont nous encodons les informations externes dans GRIMP : en modifiant la structure de certaines des structures de données d'attention, il devient possible de forcer le modèle à "s'intéresser" à des attributs différents de ceux qui seraient normalement choisis, le dirigeant ainsi vers des relations plus fortes entre les valeurs.

FunForest

L'idée de mettre en œuvre des informations externes dans la formation d'un modèle d'imputation de données est reprise dans notre deuxième contribution. À cette fin, nous concevons et implémentons FunForest (**Fun**ctional Miss**Forest**), une amélioration de l'algorithme d'imputation bien connu et très efficace MissForest [45] : avec FunForest, nous introduisons des modifications qui permettent à l'ensemble de forêts aléatoires employé par l'algorithme original d'utiliser des informations externes. Les modifications que nous introduisons sont simples et légères, et ne pénalisent pas le temps d'exécution de l'algorithme, ni ses performances standard ; toutefois, lorsque des données externes sont disponibles, nous montrons que FunForest surpasse MissForest par une marge importante.

Dans son implémentation originale, MissForest est basé sur des ensembles de forêts aléatoires, dans lesquels un grand nombre d'arbres de décision sont formés sur l'ensemble des données. L'entraînement est amorcé en utilisant une méthode d'imputation simple pour combler les vacances, puis l'algorithme affine progressivement les "suppositions" à chaque itération. L'apprentissage est effectué en entraînant itérativement la forêt aléatoire sur chaque attribut du tableau, jusqu'à ce qu'une condition de fin soit atteinte. Selon qu'un attribut est catégorique ou numérique, les arbres de décision sont des classificateurs ou des régresseurs. Le résultat final de l'imputation est obtenu en combinant les imputations de chaque arbre de décision, soit en calculant la moyenne de leurs imputations si la valeur est numérique, soit par vote majoritaire si l'attribut est catégorique. L'algorithme de base s'est avéré très efficace dans de nombreuses circonstances et, à ce titre, il a été utilisé comme référence pour d'autres ensembles de données d'imputation dans la littérature.

MissForest fonctionnelle
Avec notre contribution, nous travaillons à l'amélioration de MissForest en le rendant "FD-aware", c'est-à-dire capable d'exploiter des informations externes exprimées sous forme de dépendances fonctionnelles. Les FDs mettent en relation les différents attributs des tables : nous exploitons cette information en divisant le budget de l'estimateur (c'est-à-dire le nombre d'arbres de décision qui peuvent être utilisés) en une partie "générique" et une partie "dirigée". Les arbres de décision construits à partir de la première partie sont traités comme des arbres normaux et sont formés sur tous les attributs de la table ; les arbres de décision qui appartiennent à la partie "dirigée", cependant, sont formés exclusivement sur les colonnes qui sont connues pour être reliées entre elles par une dépendance fonctionnelle. En orientant les arbres de décision vers un sous-ensemble d'attributs de l'ensemble de données, plutôt que vers la table entière, il est possible de réduire le bruit introduit par les colonnes non liées et de se concentrer sur les valeurs trouvées dans les attributs les plus pertinents. Nous observons que, si l'affectation d'une partie du budget de l'estimateur aux arbres dirigés est bénéfique, l'absence d'arbres génériques pénalise le résultat final de l'imputation.

Nous concluons cette section en menant une campagne expérimentale approfondie pour étudier un certain nombre d'algorithmes d'imputation différents et leurs performances dans de nombreux scénarios, en faisant varier la table source, la distribution des erreurs et les hyperparamètres des algorithmes pour explorer l'effet qu'ils ont sur les performances d'imputation des données des différents systèmes. Nous nous concentrons également sur l'effet des dépendances fonctionnelles en testant le module d'attention dans GRIMP, ainsi que FunForest sur des ensembles de données qui ont des dépendances fonctionnelles connues et exactes. Enfin, nous tirons quelques conclusions sur les performances potentielles des systèmes d'imputation de données basés sur DL.

Conclusions

Dans la dernière section, nous concluons et résumons les résultats obtenus dans les domaines de l'intégration de données et de l'imputation de données. Nous racontons comment les innovations que nous avons introduites peuvent mener à une discussion plus approfondie sur le sujet des incorporations tabulaires et de l'apprentissage profond appliqué au domaine du nettoyage des données. Nous tirons quelques conclusions sur les résultats obtenus, puis nous décrivons certaines directions de recherche envisagées et suggérons des opportunités de recherche futures dans les domaines de l'intégration des données et de l'imputation des données.

Bibliography

[1] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, "End-to-end entity resolution for big data: A survey," *arXiv preprint arXiv:1905.06397*, 2019.

[2] N. Swartz, "Gartner warns firms of'dirty data'," *Information Management*, vol. 41, no. 3, p. 6, 2007.

[3] 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?" *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 993–1004, 2016.

[4] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "Data cleaning: Overview and emerging challenges," in *Proceedings of the 2016 international conference on management of data*, 2016, pp. 2201–2206.

[5] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high interest credit card of technical debt," 2014.

[6] FigureEight, "Data science report," https://visit.figure-eight.com/data-science-report.html, 2016.

[7] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2917–2926, 2012.

[8] S. Lohr, "For big-data scientists,'janitor work'is key hurdle to insights," *New York Times*, vol. 17, p. B4, 2014.

[9] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan, "Data integration: After the teenage years," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, E. Sallinger, J. V. den Bussche, and F. Geerts, Eds. ACM, 2017, pp. 101–106. [Online]. Available: https://doi.org/10.1145/3034786.3056124

[10] T. Rattenbury, J. M. Hellerstein, J. Heer, S. Kandel, and C. Carreras, *Principles of data wrangling: Practical techniques for data preparation.* " O'Reilly Media, Inc.", 2017.

[11] "How The London Whale Debacle Is Partly The Result Of An Error Using Excel," Jan 2022, [Online; accessed 6. Jan. 2022]. [Online]. Available: https://www.businessinsider.in/finance/How-The-London-Whale-Debacle-Is-Partly-The-Result-Of-An-Error-Using-Excel/articleshow/21358120.cms

[12] R. Hart, "When artificial intelligence botches your medical diagnosis, who's to blame," 2017.

[13] IBM, "Data-driven healthcare organizations use big data analytics for big gains," White paper, http://www.ibmbigdatahub.com/whitepaper/data-driven-healthcare-organizations-use-big-data-analytics-big-gains.

[14] P. S. GC, C. Sun, H. Zhang, F. Yang, N. Rampalli, S. Prasad, E. Arcaute, G. Krishnan, R. Deep, V. Raghavendra *et al.*, "Why big data industrial systems need rules and what we can do about it," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 265–276.

[15] J. Liu, J. Li, C. Liu, and Y. Chen, "Discover dependencies from data—a review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 2, pp. 251–264, 2010.

[16] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "Tane: An efficient algorithm for discovering functional and approximate dependencies," *The computer journal*, vol. 42, no. 2, pp. 100–111, 1999.

[17] C. Wyss, C. Giannella, and E. Robertson, "Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract," in *International Conference on Data Warehousing and Knowledge Discovery.* Springer, 2001, pp. 101–110.

[18] F. Chiang and R. J. Miller, "Discovering data quality rules," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1166–1177, 2008.

[19] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 5, pp. 683–698, 2010.

[20] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 376–390, 2008.

[21] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *Proceedings of the VLDB Endowment*, vol. 6, no. 13, pp. 1498–1509, 2013.

[22] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1247–1261.

[23] J. Morcos, Z. Abedjan, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, "Dataxformer: An interactive data transformation tool," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 883–888.

[24] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 865–882.

[25] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "Holodetect: Few-shot learning for error detection," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 829–846.

[26] Z. Jin, Y. He, and S. Chauduri, "Auto-transform: learning-to-transform by patterns," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2368–2381, 2020.

[27] P. Bailis, J. M. Hellerstein, and M. Stonebraker, "Readings in database systems," *URL: http://www. redbook. io/all-chapters. html (26.09. 2017)*, 2015.

[28] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker, "Temporal rules discovery for web data cleaning," *Proceedings of the VLDB Endowment*, vol. 9, no. 4, pp. 336–347, 2015.

[29] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang, "Interactive and deterministic data cleaning," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 893–907. [Online]. Available: https://doi.org/10.1145/2882903.2915242

[30] V. V. Meduri and P. Papotti, "Towards user-aware rule discovery," in *International Workshop on Information Search, Integration, and Personalization.* Springer, 2016, pp. 3–17.

[31] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[32] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[33] J. You, X. Ma, D. Y. Ding, M. Kochenderfer, and J. Leskovec, "Handling missing data with graph representation learning," *arXiv preprint arXiv:2010.16418*, 2020.

[34] Z. Huang and Y. He, "Auto-detect: Data-driven error detection in tables," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1377–1392.

[35] J. Yoon, J. Jordon, and M. Schaar, "Gain: Missing data imputation using generative adversarial nets," in *International Conference on Machine Learning.* PMLR, 2018, pp. 5689–5698.

[36] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *J. Big Data*, vol. 7, no. 1, p. 28, 2020. [Online]. Available: https://doi.org/10.1186/s40537-020-00305-w

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

[38] K. Potdar, T. S. Pardawala, and C. D. Pai, "A comparative study of categorical variable encoding techniques for neural network classifiers," *International journal of computer applications*, vol. 175, no. 4, pp. 7–9, 2017.

[39] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *CoRR*, vol. abs/2106.03253, 2021. [Online]. Available: https://arxiv.org/abs/2106.03253

[40] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *CoRR*, vol. abs/2110.01889, 2021. [Online]. Available: https://arxiv.org/abs/2110.01889

[41] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[42] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD.* ACM, 2016, pp. 855–864.

[43] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[44] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *CoRR*, vol. abs/2106.11959, 2021. [Online]. Available: https://arxiv.org/abs/2106.11959

[45] D. J. Stekhoven and P. Bühlmann, "Missforest—non-parametric missing value imputation for mixed-type data," *Bioinformatics*, vol. 28, no. 1, pp. 112–118, 2012.

[46] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[47] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2013, pp. 746–751.

[48] A. Nazabal, P. M. Olmos, Z. Ghahramani, and I. Valera, "Handling incomplete heterogeneous data using vaes," *Pattern Recognition*, vol. 107, p. 107501, 2020.

[49] T. Halpin and T. Morgan, *Information modeling and relational databases.* Morgan Kaufmann, 2010.

[50] L. Caruccio, V. Deufemia, and G. Polese, "Relaxed functional dependencies—a survey of approaches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 147–165, 2016.

[51] 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.

[52] C. S. Jensen, R. T. Snodgrass, and M. D. Soo, "Extending existing dependency theory to temporal databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 4, pp. 563–582, 1996.

[53] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann, "Data profiling with metanome," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1860–1863, Aug. 2015. [Online]. Available: http://dx.doi.org/10.14778/2824032.2824086

[54] Z. Abedjan, P. Schulze, and F. Naumann, "Dfd: Efficient functional dependency discovery," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, pp. 949–958.

[55] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka, "Regularization is all you need: Simple neural nets can excel on tabular data," *CoRR*, vol. abs/2106.11189, 2021. [Online]. Available: https://arxiv.org/abs/2106.11189

[56] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein, "SAINT: improved neural networks for tabular data via row attention and contrastive pre-training," *CoRR*, vol. abs/2106.01342, 2021. [Online]. Available: https://arxiv.org/abs/2106.01342

[57] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshow, and R. L. Stevens, "Converting tabular data into images for deep learning with convolutional neural networks," *Scientific reports*, vol. 11, no. 1, pp. 1–11, 2021.

[58] Y. Mathov, E. Levy, Z. Katzir, A. Shabtai, and Y. Elovici, "Not all datasets are born equal: On heterogeneous data and adversarial examples," 2021.

[59] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[60] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[61] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: A core of semantic knowledge,” in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW ’07. New York, NY, USA: Association for Computing Machinery, 2007, p. 697–706. [Online]. Available: https://doi.org/10.1145/1242572.1242667

[62] D. Vrandecic and M. Krötzsch, “Wikidata: a free collaborative knowledgebase,” *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014. [Online]. Available: https://doi.org/10.1145/2629489

[63] G. Weikum, L. Dong, S. Razniewski, and F. Suchanek, “Machine knowledge: Creation and curation of comprehensive knowledge bases,” *arXiv preprint arXiv:2009.11564*, 2020.

[64] O. Lassila, R. R. Swick *et al.*, “Resource description framework (rdf) model and syntax specification,” 1998.

[65] 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.

[66] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[67] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[68] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[69] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, “Advances in pre-training distributed word representations,” in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[70] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, “Distributed representations of tuples for entity resolution,” *PVLDB*, vol. 11, no. 11, pp. 1454–1467, 2018.

[71] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *SIGMOD*, 2018.

[72] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[73] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[74] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 2227–2237. [Online]. Available: https://doi.org/10.18653/v1/n18-1202

[75] A. d. S. Correia and E. L. Colombini, "Attention, please! a survey of neural attention models in deep learning," *arXiv preprint arXiv:2103.16775*, 2021.

[76] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[77] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," *CoRR*, vol. abs/1508.01211, 2015. [Online]. Available: http://arxiv.org/abs/1508.01211

[78] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[79] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[80] R. Wu, A. Zhang, I. Ilyas, and T. Rekatsinas, "Attention-based learning for missing data imputation in holoclean," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 307–325, 2020.

[81] C. Joshi, "Transformers are graph neural networks," *The Gradient*, 2020.

[82] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[83] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[84] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[85] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[86] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[87] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[88] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[89] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[90] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 889–898.

[91] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[92] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[93] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[94] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 37–48.

[95] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering." in *Nips*, vol. 14, no. 14, 2001, pp. 585–591.

[96] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[97] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 891–900.

[98] B. Perozzi, V. Kulkarni, and S. Skiena, "Walklets: Multiscale graph embeddings for interpretable network classification," *arXiv preprint arXiv:1605.02115*, pp. 043 238–23, 2016.

[99] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[100] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2021.

[101] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[102] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, ser. Proceedings of Machine Learning Research, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, Eds., vol. 27. Bellevue, Washington, USA: PMLR, 02 Jul 2012, pp. 37–49. [Online]. Available: https://proceedings.mlr.press/v27/baldi12a.html

[103] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[104] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[105] A. Clauset, C. Moore, and M. E. Newman, "Hierarchical structure and the prediction of missing links in networks," *Nature*, vol. 453, no. 7191, pp. 98–101, 2008.

[106] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[107] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, pp. 3844–3852, 2016.

[108] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[109] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[110] Dsg, "A Review : Graph Convolutional Networks (GCN)," Jan 2022, [Online; accessed 31. Jan. 2022]. [Online]. Available: https://dsgiitr.com/blogs/gcn

[111] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[112] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[113] F. P. Such, S. Sah, M. A. Dominguez, S. Pillai, C. Zhang, A. Michael, N. D. Cahill, and R. Ptucha, "Robust spatial filtering with graph convolutional neural networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 884–896, 2017.

[114] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, 2019, pp. 2022–2032.

[115] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[116] Y. Zhang and Q. Yang, "A survey on multi-task learning," *arXiv preprint arXiv:1707.08114*, 2017.

[117] S. Vandenhende, S. Georgoulis, M. Proesmans, D. Dai, and L. Van Gool, "Revisiting multi-task learning in the deep learning era," *arXiv preprint arXiv:2004.13379*, vol. 2, 2020.

[118] R. J. Miller, "Big data curation." in *COMAD*, 2014, p. 4.

[119] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren, "Curated databases," in *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2008, pp. 1–12.

[120] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1090–1101, 2009.

[121] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti, "Extraction and integration of partially overlapping web sources," *Proc. VLDB Endow.*, vol. 6, no. 10, pp. 805–816, 2013. [Online]. Available: http://www.vldb.org/pvldb/vol6/p805-bronzi.pdf

[122] F. Naumann, A. Bilke, J. Bleiholder, and M. Weis, "Data fusion in three steps: Resolving schema, tuple, and value inconsistencies," *IEEE Data Eng. Bull.*, vol. 29, no. 2, pp. 21–31, 2006. [Online]. Available: http://sites.computer.org/debull/A06June/Hummer_DEBull06_v2.ps

[123] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis, "Clio: Schema mapping creation and data exchange," in *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, ser. Lecture Notes in Computer Science, A. Borgida, V. K. Chaudhri, P. Giorgini, and

E. S. K. Yu, Eds., vol. 5600. Springer, 2009, pp. 198–236. [Online]. Available: https://doi.org/10.1007/978-3-642-02463-4_12

[124] A. Bonifati, G. Mecca, P. Papotti, and Y. Velegrakis, "Discovery and correctness of schema mapping transformations," in *Schema Matching and Mapping*, ser. Data-Centric Systems and Applications, Z. Bellahsene, A. Bonifati, and E. Rahm, Eds. Springer, 2011, pp. 111–147. [Online]. Available: https://doi.org/10.1007/978-3-642-16518-4_5

[125] Z. Bellahsene, A. Bonifati, F. Duchateau, and Y. Velegrakis, "On evaluating schema matching and mapping," in *Schema matching and mapping.* Springer, 2011, pp. 253–291.

[126] N. F. Noy, "Ontology mapping," in *Handbook on ontologies.* Springer, 2009, pp. 573–590.

[127] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *the VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[128] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *Journal on data semantics IV.* Springer, 2005, pp. 146–171.

[129] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 695–701, 2011.

[130] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos, "Valentine: Evaluating matching techniques for dataset discovery," in *2021 IEEE 37th International Conference on Data Engineering (ICDE).* IEEE, 2021, pp. 468–479.

[131] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *vldb*, vol. 1. Citeseer, 2001, pp. 49–58.

[132] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with coma++," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 906–908.

[133] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to map between ontologies on the semantic web," in *Proceedings of the 11th international conference on World Wide Web*, 2002, pp. 662–673.

[134] M. Ehrig and S. Staab, "Qom–quick ontology mapping," in *International Semantic Web Conference.* Springer, 2004, pp. 683–697.

[135] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in *Proceedings 18th International Conference on Data Engineering.* IEEE, 2002, pp. 117–128.

[136] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *ICDE*, 2018.

[137] C. Koutras, M. Fragkoulis, A. Katsifodimos, and C. Lofi, "Rema: Graph embeddings-based relational schema matching," *SEA Data workshop*, 2020.

[138] V. Christophides, V. Efthymiou, and K. Stefanidis, "Entity resolution in the web of data," *Synthesis Lectures on the Semantic Web*, vol. 5, no. 3, pp. 1–122, 2015.

[139] E. Ioannou and Y. Velegrakis, "Embench^{++}: Data for a thorough benchmarking of matching-related methods," *Semantic Web*, vol. 10, no. 2, pp. 435–450, 2019. [Online]. Available: https://doi.org/10.3233/SW-180331

[140] F. Naumann and M. Herschel, *An Introduction to Duplicate Detection*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010. [Online]. Available: https://doi.org/10.2200/S00262ED1V01Y201003DTM003

[141] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl, "A blocking framework for entity resolution in highly heterogeneous information spaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 12, pp. 2665–2682, 2012.

[142] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, "Comparative analysis of approximate blocking techniques for entity resolution," *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 684–695, 2016.

[143] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *IEEE transactions on knowledge and data engineering*, vol. 24, no. 9, pp. 1537–1555, 2011.

[144] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas, "Blocking and filtering techniques for entity resolution: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–42, 2020.

[145] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller, "Framework for evaluating clustering algorithms in duplicate detection," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1282–1293, 2009.

[146] R. Singh, V. V. Meduri, A. Elmagarmid, S. Madden, P. Papotti, J.-A. Quiané-Ruiz, A. Solar-Lezama, and N. Tang, "Synthesizing entity matching rules by examples," *Proceedings of the VLDB Endowment*, vol. 11, no. 2, pp. 189–202, 2017.

[147] L. Chiticariu, Y. Li, and F. Reiss, "Rule-based information extraction is dead! long live rule-based information extraction systems!" in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 827–832.

[148] F. Panahi, W. Wu, A. Doan, and J. F. Naughton, "Towards interactive debugging of rule-based entity matching." in *EDBT*, 2017, pp. 354–365.

[149] M. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky, "Hil: a high-level scripting language for entity integration," in *Proceedings of the 16th international conference on extending database technology*, 2013, pp. 549–560.

[150] M. Paganelli, P. Sottovia, F. Guerra, and Y. Velegrakis, "Tuner: Fine tuning of rule-based entity matchers," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, W. Zhu, D. Tao, X. Cheng, P. Cui, E. A. Rundensteiner, D. Carmel, Q. He, and J. X. Yu, Eds. ACM, 2019, pp. 2945–2948. [Online]. Available: https://doi.org/10.1145/3357384.3357854

[151] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 39–48.

[152] C. Zhao and Y. He, "Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning," in *WWW*, 2019, pp. 2413–2424.

[153] Ö. Ö. Çakal, M. Mahdavi, and Z. Abedjan, "CLRL: feature engineering for cross-language record linkage," in *EDBT*, 2019, pp. 678–681.

[154] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa, "Low-resource deep entity resolution with transfer and active learning," *arXiv preprint arXiv:1906.08042*, 2019.

[155] S. Thirumuruganathan, S. A. P. Parambath, M. Ouzzani, N. Tang, and S. Joty, "Reuse and adaptation for entity resolution through transfer learning," *arXiv preprint arXiv:1809.11084*, 2018.

[156] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu, "Corleone: Hands-off crowdsourcing for entity matching," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 601–612.

[157] V. D. Cicco, D. Firmani, N. Koudas, P. Merialdo, and D. Srivastava, "Interpreting deep learning models for entity resolution: an experience report using LIME," in *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*, R. Bordawekar and O. Shmueli, Eds. ACM, 2019, pp. 8:1–8:4. [Online]. Available: https://doi.org/10.1145/3329859.3329878

[158] A. Farhangfar, L. Kurgan, and J. Dy, "Impact of imputation of missing values on classification error for discrete data," *Pattern Recognition*, vol. 41, no. 12, pp. 3692–3705, 2008.

[159] A. d. l. V. de León, B. Chen, and V. J. Gillet, "Effect of missing data on multitask prediction methods," *Journal of cheminformatics*, vol. 10, no. 1, pp. 1–12, 2018.

[160] D. A. Bennett, “How can i deal with missing data in my study?” *Australian and New Zealand journal of public health*, vol. 25, no. 5, pp. 464–469, 2001.

[161] D. B. Rubin, “Inference and missing data,” *Biometrika*, vol. 63, no. 3, pp. 581–592, 1976.

[162] S. Fielding, P. M. Fayers, A. McDonald, G. McPherson, and M. K. Campbell, “Simple imputation methods were inadequate for missing not at random (mnar) quality of life data,” *Health and Quality of Life Outcomes*, vol. 6, no. 1, pp. 1–9, 2008.

[163] N. Resseguier, R. Giorgi, and X. Paoletti, “Sensitivity analysis when data are missing not-at-random,” *Epidemiology*, vol. 22, no. 2, p. 282, 2011.

[164] S. Van Buuren and K. Groothuis-Oudshoorn, “mice: Multivariate imputation by chained equations in r,” *Journal of statistical software*, vol. 45, pp. 1–67, 2011.

[165] J. Luengo, S. García, and F. Herrera, “On the choice of the best imputation methods for missing values considering three groups of classification methods,” *Knowledge and information systems*, vol. 32, no. 1, pp. 77–108, 2012.

[166] J. W. Grzymala-Busse, L. K. Goodwin, W. J. Grzymala-Busse, and X. Zheng, “Handling missing attribute values in preterm birth data sets,” in *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing.* Springer, 2005, pp. 342–351.

[167] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, “Missing value estimation methods for dna microarrays,” *Bioinformatics*, vol. 17, no. 6, pp. 520–525, 2001.

[168] W. W. Cohen, “Fast effective rule induction,” in *Machine learning proceedings 1995.* Elsevier, 1995, pp. 115–123.

[169] P. Clark and T. Niblett, “The cn2 induction algorithm,” *Machine learning*, vol. 3, no. 4, pp. 261–283, 1989.

[170] L. Gondara and K. Wang, “Mida: Multiple imputation using denoising autoencoders,” in *Pacific-Asia conference on knowledge discovery and data mining.* Springer, 2018, pp. 260–272.

[171] F. Honghai, C. Guoshun, Y. Cheng, Y. Bingru, and C. Yumei, “A svm regression based approach to filling in missing values,” in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems.* Springer, 2005, pp. 581–587.

[172] P.-A. Mattei and J. Frellsen, “Miwae: Deep generative modelling and imputation of incomplete data sets,” in *International conference on machine learning.* PMLR, 2019, pp. 4413–4423.

[173] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, “Creating embeddings of heterogeneous relational datasets for data integration tasks,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1335–1349. [Online]. Available: https://doi.org/10.1145/3318464.3389742

[174] J. Turian, L. Ratinov, and Y. Bengio, “Word representations: a simple and general method for semi-supervised learning,” in *ACL*. ACL, 2010, pp. 384–394.

[175] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *CoRR*, vol. abs/1607.04606, 2016. [Online]. Available: http://arxiv.org/abs/1607.04606

[176] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[177] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, “Deep contextualized word representations,” *CoRR*, vol. abs/1802.05365, 2018.

[178] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, “HARP: hierarchical representation learning for networks,” *CoRR*, vol. abs/1706.07845, 2017. [Online]. Available: http://arxiv.org/abs/1706.07845

[179] R. Bordawekar and O. Shmueli, “Using word embedding to enable semantic queries in relational databases,” in *DEEM Workshop*. ACM, 2017, p. 5.

[180] ——, “Exploiting latent information in relational databases via word embedding and application to degrees of disclosure.” in *CIDR*, 2019.

[181] R. Bordawekar, B. Bandyopadhyay, and O. Shmueli, “Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities,” *arXiv preprint arXiv:1712.07199*, 2017.

[182] R. C. Fernandez and S. Madden, “Termite: a system for tunneling through heterogeneous data,” *arXiv preprint arXiv:1903.05008*, 2019.

[183] M. Günther, “Freddy: Fast word embeddings in database systems,” in *SIGMOD*. ACM, 2018, pp. 1817–1819.

[184] M. Günther, M. Thiele, E. Nikulski, and W. Lehner, “Retrolive: Analysis of relational retrofitted word embeddings,” *EDBT*, 2020.

[185] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu, “Corleone: hands-off crowdsourcing for entity matching,” in *SIGMOD*, 2014.

[186] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park, "Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services," in *SIGMOD*, 2017.

[187] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *TACL*, vol. 5, pp. 135–146, 2017.

[188] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014, pp. 1532–1543.

[189] R. J. Miller, F. Nargesian, E. Zhu, C. Christodoulakis, K. Q. Pu, and P. Andritsos, "Making open data transparent: Data discovery on open data." *IEEE Data Eng. Bull.*, vol. 41, no. 2, pp. 59–70, 2018.

[190] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, c. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *SIGKDD*. ACM, 2019.

[191] S. Thirumuruganathan, N. Tang, M. Ouzzani, and A. Doan, "Data curation with deep learning," *EDBT*, 2020.

[192] P. Suganthan, A. Ardalan, A. Doan, and A. Akella, "Smurf: Self-service string matching using random forests," *PVLDB*, vol. 12, no. 3, pp. 278–291, 2018.

[193] E. Zhu, Y. He, and S. Chaudhuri, "Auto-join: Joining tables by leveraging transformations," *PVLDB*, vol. 10, no. 10, pp. 1034–1045, 2017.

[194] R. Hull and M. Yoshikawa, "ILOG: declarative creation and manipulation of object identifiers," in *VLDB*, 1990, pp. 455–468.

[195] X. Chu and I. F. Ilyas, *Data Cleaning.* ACM, 2019.

[196] 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?" *PVLDB*, vol. 9, no. 12, pp. 993–1004, 2016.

[197] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro, "Messing up with BART: error generation for evaluating data-cleaning algorithms," *PVLDB*, vol. 9, no. 2, pp. 36–47, 2015.

[198] A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou, "Word translation without parallel data," *arXiv preprint arXiv:1710.04087*, 2017.

[199] S. Maßmann, S. Raunich, D. Aumüller, P. Arnold, and E. Rahm, "Evolution of the COMA match system," in *International Workshop on Ontology Matching*, 2011.

[200] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro, "++Spicy: an opensource tool for second-generation schema mapping and data exchange," *PVLDB*, vol. 4, no. 12, pp. 1438–1441, 2011.

[201] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[202] F. Biessmann, T. Rukat, P. Schmidt, P. Naidu, S. Schelter, A. Taptunov, D. Lange, and D. Salinas, "Datawig: Missing value imputation for tables." *J. Mach. Learn. Res.*, vol. 20, pp. 175–1, 2019.

[203] N. Ahmadi, H. Sand, and P. Papotti, "Unsupervised matching of data and text," in *ICDE*, 2022.

[204] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "Turl: Table understanding through representation learning," *arXiv preprint arXiv:2006.14806*, 2020.