Embedding models for relational data analytics

Alexis Cvetkov-Iliev

université PARIS-SACLAY

Embedding models for relational data analytics

Modèles d'embedding pour l'analyse de données relationnelles

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique mathématique
Graduate School : Informatique et sciences du numérique. Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **Inria Saclay – Île-de-France (Université Paris-Saclay, Inria)**, sous la direction de **Gaël VAROQUAUX**, directeur de recherche à Inria Saclay – Île-de-France, et le co-encadrement de **Alexandre ALLAUZEN**, professeur à l'ESPCI.

Thèse soutenue à Paris-Saclay, le 25 janvier 2023, par

Alexis CVETKOV-ILIEV

Composition du jury

Membres du jury avec voix délibérative

Anne VILNAT Professeure, Université Paris-Saclay	Présidente
Felix BIESSMANN Professeur, Berliner Hochschule für Technik	Rapporteur & Examinateur
Adrien COULET Chargé de recherche (HDR), Inria Paris	Rapporteur & Examinateur
José HERNÁNDEZ-ORALLO Professeur, Universitat Politècnica de València	Examinateur

THESE DE DOCTORAT

NNT : 2023UPASG004

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Modèles d'embedding pour l'analyse de données relationnelles
Mots clés : apprentissage statistique, plongement vectoriel, analyse de données

Résumé : L'analyse de données, par exemple via des modèles d'apprentissage automatique, nécessite qu'elles soient regroupées dans une table unique, où les entités étudiées sont décrites par un nombre fixe d'attributs. En pratique cependant, les données sont souvent irrégulières et éparpillées à travers plusieurs sources (cf. bases de données relationnelles). Il est alors nécessaire de les assembler dans une seule table, ce qui requiert du temps et de l'expertise.

Pour faciliter l'analyse de données non-assemblées, nous étudions dans cette thèse le potentiel des modèles d'embedding, qui calculent une répresentation vectorielle pour chaque entité. Nous nous intéressons particulièrement aux deux problèmes suivants : **1)** l'appariement d'entités (par ex. rattacher "Paris" et "Paris, FR"), lorsque les sources de données ont différentes manières de représenter la même entité ; et **2)** l'agrégation et la jointure de données non-assemblées.

Au travers de cette thèse, nous montrons que les modèles d'embedding sont en effet des outils prometteurs pour l'analyse de données : **1)** utiliser des représentations vectorielles adaptées peut remplacer l'appariement manuel des entités, sans compromettre la qualité des analyses en aval ; et **2)** ces représentations peuvent être apprises directement sur de larges volumes de données non-assemblées pour agréger l'information disponible sur les entités dans des vecteurs facilement réutilisables dans de nombreuses applications.

Title : Embedding models for relational data analytics
Keywords : machine learning, vectorial embeddings, data analytics

Abstract : Data analysis, *e.g.* via machine learning models, requires data in a single table, where the entities under study are described by a fixed set of attributes. In practice however, data is often irregular and scattered across multiple sources. It must thus be assembled into a single table, which requires time and expertise from the analyst.

As an alternative, we investigate in this thesis the potential of embedding models to facilitate the analysis of unassembled data. We especially consider two problems : **1)** entity matching (*e.g.* linking "Paris" and "Paris, FR"), when data sources have different ways of representing the same entity ; and **2)** the aggregation and joining of unassembled data.

Throughout this thesis, we show that embedding models are indeed promising tools for data analytics : **1)** adapted vectorial representations of entities can replace manual entity matching without hindering the quality of subsequent analyses ; and **2)** these representations can be learned directly over large databases to summarize the available information on entities into vectors readily usable in various downstream tasks.

EMBEDDING MODELS FOR RELATIONAL DATA ANALYTICS

ALEXIS CVETKOV-ILIEV

Dissertation submitted in partial fulfillement of the requirements for the degree of Doctor of Philosophy.

Université Paris-Saclay
Sciences et technologies de l'information et de la communication

October 2019 – January 2023
Inria Saclay Île-de-France, Palaiseau, France

RÉSUMÉ DÉTAILLÉ

L'analyse de données, par exemple via des modèles d'apprentissage automatique, nécéssite qu'elles soient regroupées dans une table unique, où les entités étudiées sont décrites par un nombre fixe d'attributs. En pratique cependant, les données sont souvent irrégulières et éparpillées à travers plusieurs sources (cf. bases de données relationnelles). Il est alors nécessaire de les assembler dans une seule table, ce qui requiert du temps et de l'expertise.

Pour faciliter l'analyse de données non-assemblées, nous étudions dans cette thèse le potentiel des modèles d'embedding, qui calculent une répresentation vectorielle pour chaque entité. Nous nous intéressons particulièrement à deux problèmes : **1)** l'appariement d'entités, et **2)** la jointure de données non-assemblées.

Le premier problème survient lorsque les données analysées proviennent de sources non-normalisées. Dans ce cas, une entité donnée peut être représentée de différentes manières selon la source, par exemple "Paris" et "Paris, FR". Cela est problématique pour l'analyse des données, car les deux valeurs seront traitées indépendamment, ce qui va entacher la qualité des résultats. Pour corriger cela, une pratique courante est de rattacher à la main les deux valeurs, de sorte qu'elle aient la même orthographe. Cependant, un tel procédé demande généralement beaucoup de temps et d'effort de la part de l'analyste.

À la place, nous proposons ici de simplement remplacer les entités non-normalisées par des représentations vectorielles (ou *embeddings*) qui capturent les similarités entre entités, de sorte que deux valeurs qui désignent la même entité obtiennent des vecteurs proches. Au travers d'une analyse des salaires au Texas, nous montrons empiriquement que des embeddings adaptés, couplés avec de puissants modèles d'apprentissage statistique, peuvent se substituer à un nettoyage manuel des données, sans entacher la qualité de l'analyse en aval.

Le second problème survient quand il faut assembler les données de plusieurs tables en une seule pour les analyser. Pour cela, une opération courante est la *jointure*, qui permet de prendre des colonnes d'une table, pour les rajouter à une autre. Toutefois, une condition nécessaire à la jointure est d'avoir une relation 1-pour-1 entre les entités d'intérêt et les valeurs à joindre. Par exemple, en partant d'une table avec différentes villes, on peut facilement joindre leurs populations à partir d'une autre table, car chaque ville est associée à une seule valeur de population. En pratique cependant, les données contiennent souvent des relations 1-pour-plusieurs, où chaque entité (*i.e.* ville) est associée à plusieurs valeurs, par exemple les salaires de leurs habitants. Dans ce cas, il est nécessaire d'agréger au préalable ces valeurs en un unique indicateur (par ex. le salaire moyen par ville), afin de pouvoir faire la jointure.

Sur de grandes bases de données, cette étape d'agrégation avant jointure est difficile et coûteuse en temps. En effet, le nombre d'indicateurs que l'on peut créer explose rapidement, notamment lorsque l'on enchaîne les jointures à travers plusieurs tables. Même sans calculer tous les indicateurs possibles, identifier lesquels sont pertinents pour l'analyse en question est difficile.

Plutôt que de joindre les tables à la main, nous proposons d'utiliser un modèle d'embedding directement sur les données non-assemblées, afin d'obtenir un vecteur pour chaque entité. L'avantage de tels vecteurs est qu'ils agrègent implicitement toute l'information disponible sur les entités, et qu'ils peuvent être directement joints à de nouvelles tables, car il y a une relation 1-pour-1 entre une entité et son embedding. En parallèle, une contribution clé de notre approche est la prise en compte des valeurs numériques dans les modèle d'embeddings, qui sont par défaut limités aux entités catégorielles.

Enfin, nous montrons à travers une série d'expériences la qualité et la scalabilité de notre approche : les embeddings obtenus peuvent être facilement réutilisés pour enrichir de nombreuses applications avec de l'information externe.

REMERCIEMENTS

Je remercie tout d'abord mes encadrants, Gaël Varoquaux et Alexandre Allauzen, pour leur disponibilité et leur bienveillance tout au long de ma thèse. Ce fut un plaisir de travailler ensemble et d'apprendre toutes les choses essentielles à un jeune chercheur. Je remercie également les membres du jury: Anne Vilnat, Adrien Coulet, Felix Biessmann et José Hernández-Orallo, pour avoir évalué mes travaux de thèse, et pour leurs commentaires pertinents.

Je remercie aussi tous mes collègues des équipes Soda et Mind: Jovan, Lilian, Félix, Léo, Benoît, Cédric, Florent, Ambroise, Alexandre Blain et Perez, Apolline, Thomas, Julia; ainsi que tous les autres, trop nombreux pour les citer, pour les bons moments passés ensemble et les innombrables heures de ping-pong.

Enfin je remercie ma famille, mes amis, et toi Duya, pour ton soutien et ta bonne humeur constante.

CONTENTS

1 INTRODUCTION

The amount of information in the world is estimated to double every two years. Data is now ubiquitous and collected in various domains: business, industry, medicine, research; and under various forms: text, images, tables, graphs... Following this trend, *data analytics* (Runkler, 2020), or related fields such as *data mining* and *knowledge discovery*, have emerged as an ensemble of methods to generate *insights* from data. In this chapter, we introduce the notion of data analytics (Section 1.1), as well as the numerous data engineering challenges faced in practice by data analysts (Section 1.2). At last, we state the focus and objective of this thesis: leveraging embedding models to facilitate data analytics over relational data (Section 1.3), such as tables or knowledge graphs. Importantly, we take care in this chapter to connect different views of data, bridging those found in database research with those from statistics.

1.1 DATA ANALYTICS: DRAWING VALID CONCLUSIONS FROM IMPERFECT DATA

We introduce in this section what we mean by "data analytics". Starting from the classic database view, we move on to a more *statistical* vision of data analytics that better accounts for imperfections in real-world data. We then depart from typical analytic practices based on queries in databases to more advanced analytical models, such as statistical models used in machine learning.

1.1.1 *The database view: querying facts from perfect data*

The classic view for analytics in databases is to consider the data as a true, noiseless description of reality, and to query it to produce *factual* answers to analytical questions like *"What is the annual salary of a specific person?"*, or *"What is the average housing price in a given city?"* (see Figure 1.1). Importantly, the data must satisfy certain properties to ensure the validity of the results obtained by querying it. In particular, data must be:

1. **exact**, *i.e.* the values reported in the data (*e.g.* employee salaries or housing prices) are correct and do not exhibit uncertainties or errors.
2. **instance-complete**, meaning that *all* instances relevant to the question at hand are present in the data, *e.g.* the employee of interest or all houses in the considered city.
3. **attribute-complete**, meaning that *all* the attributes needed to select the relevant instances are available in the data, such as employee names or the cities in which houses are located.

Employees

Employee_ID	Name	Salary
1	John Doe	53,000$
2	Alice Pearce	61,000$
3	Bob Ross	39,000$

SELECT **Salary** FROM **Employees**
WHERE **Name** = "John Doe"

Houses

House_ID	City	Area	Price
1	New York	70 m²	610,000$
2	Los Angeles	38 m²	350,000$
3	New York	45 m²	420,000$

SELECT **Price** FROM **Houses**
WHERE **City** = "New York"

Figure 1.1 – **Example of two analytical questions:** *What is the salary of John Doe?* (left), and *What is the average housing price in New York?* (right). The classic "database" approach to answer these questions consists in data queries, illustrated here in SQL.

Obviously, whether or not a dataset satisfies these properties depends on the analytical question at hand. It is often easy to ensure for simple questions on individual instances (*e.g.* what is the salary of John Doe?), but these notions are harder to define for questions on large groups of instances (*e.g.* what is the average salary of a data scientist?).

1.1.2 *Imperfections in real-world data call for statistical estimation*

We adopt here a database perspective of analytics, as introduced above. However, while it is convenient to consider the data as perfect, it is seldom verified in practice: **due to many imperfections in real-world data, we generally obtain *estimates*, rather than *true* values for the analytical quantities of interest.** First, data can be *inexact* for various reasons: many quantities are hard to measure precisely (*e.g.* the daily number of COVID cases) and often come with error bars, as in experimental measurements. Even when quantities have a well defined value (*e.g.* employee salaries), human errors in the data management pipeline can lead to mistakes in the final values, such as typos or unit conversion errors. Data can also be *attribute-incomplete*: there may be no column "City" in our "Houses" table (Figure 1.1), or it may contain missing values, as resulting from a join between two tables.

Finally, the most important hurdle to data analysis is generally *instance-incompleteness*. Indeed, most of the interesting analytical questions strive for **generalization**, *i.e.* building insights that are valid outside of the data at hand. Therefore they often consider *populations* of instances rather than *individual* instances. In practice however, we only have access to a small fraction –potentially unrepresentative– of the population under study, *e.g.* employees of a certain company, rather than *all* employees.

Importantly, the notion of "instance-completeness" stems from the classic database stance which views data as *finite*. This departs from

the statistical view of data as *samples* of an underlying *distribution*, for which we can never have all the data.

For all these reasons, we move away from the view of analytics formulated in a database framing, and adopt instead a *statistical* perspective: data queries return *estimates*, rather than true results. In mathematical terms, an *estimator* f is a rule for calculating an estimate $\hat{\theta}$ of a given quantity θ^* (the *estimand* [1]), based on observed data $\mathcal{D} = \{\boldsymbol{X}_i\}_{1 \leqslant i \leqslant n} \in \mathcal{P}_n$:

$$\begin{aligned} f \colon \mathcal{P}_n &\longrightarrow \mathbb{R} \\ \{\boldsymbol{X}_i\}_{1 \leqslant i \leqslant n} &\longmapsto \hat{\theta} \end{aligned} \tag{1.1}$$

where $\boldsymbol{X}_i$ is a sample (*e.g.* a table row) describing the attributes of the i^{th} instance, and $\mathcal{P}_n$ is the set of all possible combinations of n samples from the population $\mathcal{P}$ of interest.

We can measure the estimation error in a statistical sense by viewing the dataset $\mathcal{D}$ as a random variable over $\mathcal{P}_n$, and computing the *mean squared error* (MSE). Importantly, the mean squared error of any estimator can be decomposed into a bias and variance term (Equation 1.2), with $\text{Bias}\left[\hat{\theta}\right] = \theta^* - \mathbb{E}_{\mathcal{D}}\left[\hat{\theta}\right]$ the deviation between the average estimate and the true value, and $\text{Var}\left[\hat{\theta}\right]$ the variance of these estimates.

$$\text{MSE} = \mathbb{E}_{\mathcal{D}}\left[\left(\theta^* - \hat{\theta}\right)^2\right] = \text{Bias}\left[\hat{\theta}\right]^2 + \text{Var}\left[\hat{\theta}\right] \tag{1.2}$$

1.1.3 *From binning to shrinkage estimators*

Statistical estimation in databases typically relies on **binning**, creating groups of instances from the population of interest via queries, followed by an **aggregation** procedure (*e.g.* averaging) to estimate the desired quantity. For example to study the distribution of ages in a population, we can count the number of individuals with age 18, 19, etc... Although unbiased [2], such estimators may exhibit high variance, especially when the number of samples in each bin is small. To reduce the variance of binning estimates, a common solution in statistics is to *smooth* them with *continuous* models, *e.g.* using a Gaussian kernel model to estimate the distribution of ages (Figure 1.2).

Following this idea, *shrinkage* estimators (Fourdrinier et al., 2018) go further by biasing estimates towards some *prior* value, *e.g.* $\hat{\theta}_{\text{shrinkage}} = (1-\alpha)\,\hat{\theta}_{\text{binning}} + \alpha\,\hat{\theta}_{\text{prior}}$, with $\alpha \in [0, 1]$. In our example of ages, the prior is that of continuity: estimates for similar ages should be close. The intuition is the following: after shrinkage, the distribution of estimates $\hat{\theta}_{\text{shrinkage}}$ (across randomly sampled datasets $\mathcal{D}$) is grouped around the prior, having thus less variance but exhibiting a bias. With a good prior and shrinkage factor α, the reduction in variance can overcome the increase in bias, and overall decrease the mean squared

1. In practice the estimand may be not a scalar $\theta^* \in \mathbb{R}$, but a vector $\boldsymbol{\theta}^* \in \mathbb{R}^p$. This is the case when estimating jointly multiple parameters, *e.g.* the average housing price in multiple cities.

2. Assuming no selection biases in the data.

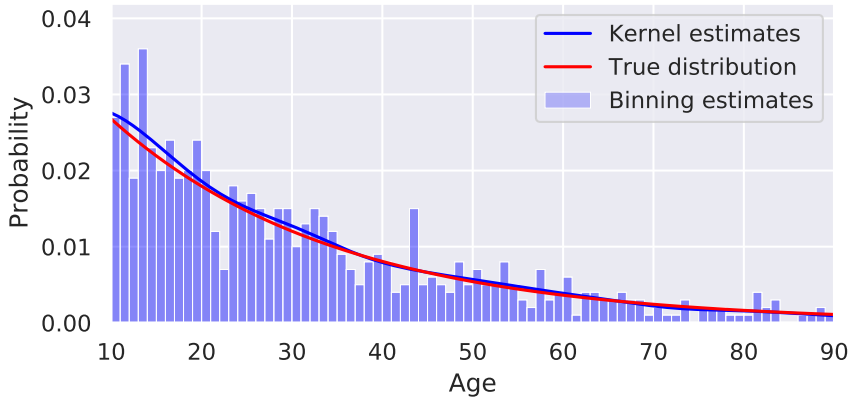

Figure 1.2 – **Smoothing binning estimates with kernel models:** When studying the age distribution from a small number of individuals, binning estimates tend to exhibit a lot of variance. Instead, kernel models can leverage samples in neighboring bins to provide less noisy and more accurate estimates.

error. A good example of this is the James-Stein estimator (James and Stein, 1992), which *always*[3] achieves a lower error than the empirical mean when estimating the mean value of several quantities simultaneously (*e.g.* the average housing price in multiple cities).

1.1.4 *Beyond shrinkage: machine learning for statistical estimation*

THE FRAMEWORK OF SUPERVISED LEARNING Supervised learning aims to learn to predict a target variable $Y \in \mathcal{Y}$ from inputs $\boldsymbol{X} \in \mathcal{X}$ (typically $\mathcal{X} = \mathbb{R}^p$), using pairs of examples $(\boldsymbol{X}, Y)$ sampled from a distribution P. Formally, we want to find a function $f\colon \mathcal{X} \to \mathcal{Y}$ that minimizes the expected risk $\mathbb{E}\left[l\left(f(\boldsymbol{X}), Y\right)\right]$ given a cost function $l\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, called the *loss*. The best possible prediction function is known as the *Bayes rule*, given by

$$f^* = \underset{f\colon \mathcal{X}\to\mathcal{Y}}{\operatorname{argmin}}\ \mathbb{E}\left[l\left(f(\boldsymbol{X}), Y\right)\right] \tag{1.3}$$

In practice, we do not know the distribution P and thus cannot minimize the expected risk to obtain the Bayes rule f^*. Instead, *learning* procedures such as random forests or neural networks minimize the *empirical* risk to construct a prediction function $\hat{f}_n$ from a set of *training* pairs $\mathcal{D}_{n,\text{train}} = \{(\boldsymbol{X_i}, Y_i), i = 1, ..., n\}$.

$$\hat{f}_n = \underset{f\colon \mathcal{X}\to\mathcal{Y}}{\operatorname{argmin}}\ \frac{1}{n}\sum_{i=1}^{n} l\left(f(\boldsymbol{X}_i, Y_i)\right) \tag{1.4}$$

Importantly, a learning procedure is *consistent* if it yields, given a infinite amount of data, a function that achieves the same expected risk as the Bayes rule f^*, *i.e.* $\mathbb{E}\left[l\left(\hat{f}_n(\boldsymbol{X}), Y\right)\right] \xrightarrow[n\to\infty]{} \mathbb{E}\left[l\left(f^*(\boldsymbol{X}), Y\right)\right]$.

3. Assuming that the data follows a multivariate Gaussian distribution.

MACHINE LEARNING FOR STATISTICAL ESTIMATION **Many of the quantities studied in data analyses can be viewed as conditional aggregates**, *e.g.* the average housing price in a city $\mathbb{E}[\text{Price}|\text{City}]$. Indeed, querying the data to select samples from the population of interest amounts to conditioning on the corresponding attributes, such as the city.

Interestingly, **conditional aggregates can be estimated by machine learning models if they are trained to minimize a well-chosen loss function**. For instance, with a squared error $l(Y_1, Y_2) = (Y_1 - Y_2)^2$, the Bayes rule f^* leads to conditional expectations $f^*(\boldsymbol{X}) = \mathbb{E}[Y|\boldsymbol{X}]$. Likewise, using a quantile (or pinball) loss leads to conditional quantiles $f^*(\boldsymbol{X}) = Q_\alpha(Y|\boldsymbol{X})$. With enough samples, *consistent* learning procedures will eventually converge towards these conditional aggregates.

LEVERAGING SIMILARITIES TO REFINE ESTIMATES A drawback of traditional estimators based on binning is that they only consider samples from the population under study, which may be rare in the data. Instead, machine learning models are able to leverage related samples outside the population of interest to refine their estimates. For instance, a simple K-nearest neighbors model can leverage data from a *similar* city (*e.g.* with close populations or locations) to better estimate housing prices in a city for which there are few samples. While this is akin to kernel estimates and shrinkage methods, powerful machine learning models go much further than that. They offer greater flexibility and can adapt to the specificities of the data, *e.g.* with an adaptative notion of *similarity* or *distance* between samples. For example, proximity to public transports may impact more housing prices in rural areas than in urban areas. Although these models come with more parameters, they can easily be tuned to minimize a *test* error (which serves as a measure for the estimation error) to provide accurate estimates and avoid overfitting. For all these reasons, machine learning models are particularly suited for data analytics, especially in complex questions where we only have a few representatives for the population of interest.

1.2 DATA ENGINEERING FOR DATA ANALYTICS

Although machine learning models are powerful tools for data analytics, **they typically require data in a single table** containing the features and target values $(\boldsymbol{X_i}, Y_i)$ for each sample. In practice however, data is often scattered across multiple databases, posing data integration challenges. Even in a single table, data quality issues may impair the validity of subsequent analyses. To address this, **many data engineering steps are generally performed by the analyst**. Yet they are time-consuming, taking up to **80%** of the effort in data analyses (CrowdFlower, 2016; Dasu and Johnson, 2003). We give in this section an overview of such data engineering problems.

1.2.1 *The data analysis pipeline*

Following Nazabal et al., 2020, we decompose a typical data analysis pipeline into the following steps:

1. **Data organization** aims to obtain a representation of the data that is best suited for the task at hand. This usually involves: identifying the structure of the raw data to read it properly (*data parsing*), understanding the contents of the data (*data dictionary*), and assembling data from multiple sources into a single table (*data integration*).
2. After organizing data into the desired format, **data quality** aims to fix imperfections that could hinder further analysis. It includes any process that modifies data values without changing the structure of the data, such as *standardization* or repairing *missing* and *anomalous* data.
3. Finally, **data analysis** is performed to extract insights from the clean data, for instance via visualization, regression, classification, clustering... An important part of the analysis then consists in making sense of the results, *e.g.* through model *interpretation* and *evaluation*.

Note that in practice, data analyses are often conducted iteratively rather than linearly, with many back-and-forths between data engineering and the actual analysis. In the following subsections, we describe the issues arising from data organization and data quality, that we summarize in Table 1.1.

1.2.2 *Data organization*

The initial step of a successful analysis is to organize the data into a structure well suited for the task of interest and the analytical method, *e.g.* a single table for machine-learning models.

1.2.2.1 *Data parsing*

Organizing raw data first requires to be able to **read** it, which poses multiple challenges. Data may come in a variety of formats (*e.g.* CSV, XML, RDF, JSON...), which must be loaded through different open or proprietary softwares. Even when data comes in a single format, it can be encoded in multiple ways. For instance, CSV files can exhibit various delimiters, quote and escape characters which must be carefully identified before loading the data. This led to several approaches for automating this tedious process (van den Burg et al., 2019).

1.2.2.2 *Data dictionary*

An essential step after loading the data is to build a *data dictionary* to **understand its content**, *e.g.* the meaning, type and unit of each attribute in a table. This can be difficult in practice: if such metadata

is not available or out of date, it must be inferred from the values and may require domain-expertise. Metadata may also come under different formats, such as text files, extra headers or additional CSV files. It is thus common for a data scientist to go back and forth between the data and the metadata to get an overview of its content.

The construction of such a data dictionary occurs at several levels: tables, features and feature values. Starting at the tables level, the goal is to get a global understanding of the data: how many tables? what are they about? are they linked (as in relational databases)? how many features and instances in each table?

The next step is to understand the meaning and type of each feature. This information is generally provided in the column names or in separate files, but may be missing or imprecise. For instance, medical data often contains acronyms or domain-specific terms, such as *DOB* for *Date Of Birth*. Looking at the values is also often necessary to feature understanding: a column *Annual Salary* may be categorical (*e.g.* with bins "40k-50k", "100k+") when we expected numerical values in the first place.

Finally, we must get an idea of the values contained in the features: what do they mean? what is the number of unique values? what is their range and unit? are there unexpected values? In some cases, understanding feature values is difficult and requires domain-expertise. For example, the U.S. bureau of labor statistics refers to a math teacher in higher education with the job code "25-1022".

1.2.2.3 *Data integration*

After loading and understanding the data, the analyst must **integrate it into a single structure** for analysis. This process is generally tedious, as data is often scattered across multiple sources, and may take various forms (*e.g.* tables, graphs, text, times series...). In tables, data integration mostly relies on two operations: **joins** to extend a table with new features from other tables, and **unions** to extend it with new rows (Figure 1.3)

TABLE UNIONING AND DEDUPLICATION Table unioning consists in aggregating together the rows of several tables. While it is in theory a straightforward operation (no record linkage involved, unlike joining), many imperfections in the data can hinder this process. Features may be ordered differently across tables, or labeled with different names that must be matched (a process known as *schema matching*). Similarly, features present in certain tables may be absent from others, leading to missing values if not removed. Finally, unioning may create duplicate records that must be identified and removed from the data, a process called *deduplication* (Elmagarmid et al., 2007).

TABLE JOINS AND RECORD LINKAGE When joining two tables to merge their features, a first challenge is to **establish correspondences between their rows or entities**, which is the goal of *record linkage* (Elmagarmid et al., 2007; Fellegi and Sunter, 1969). Unlike dedupli-

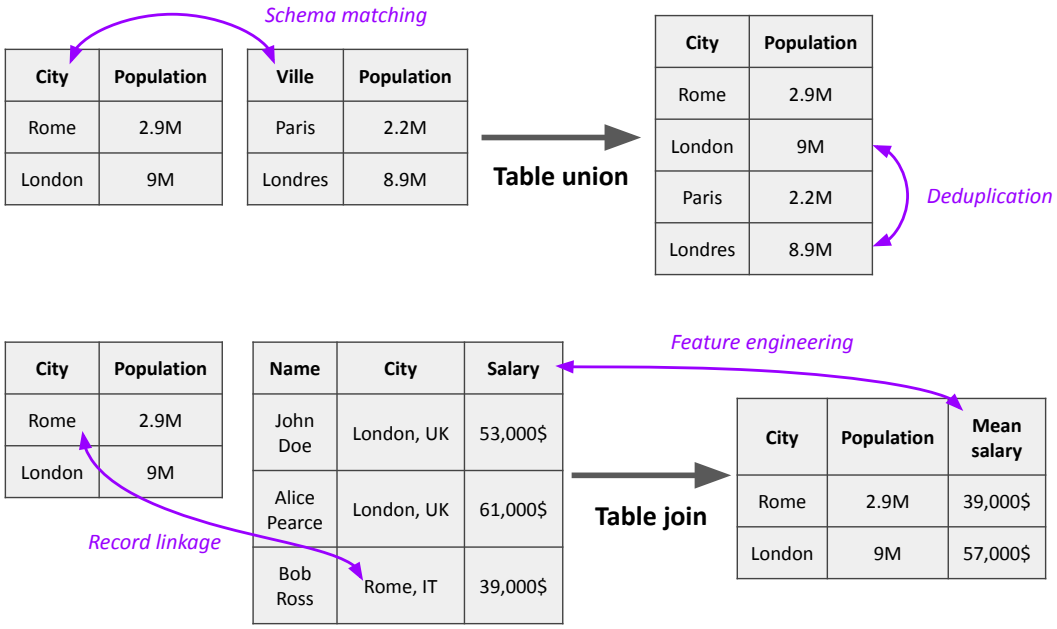

Figure 1.3 – **Two data assembling operations:** Table unions allow to concatenate rows from two tables, but may require schema matching or deduplication. Table joins allow to merge the columns of two tables, but may call for record linkage or feature engineering.

cation, record linkage focuses on matching instances *across*, rather than *within* tables, but relies on similar techniques. Although this step is straightforward when clean *primary keys* exist in the data, they often exhibit imperfections, as when dealing with data from non-normalized sources. For instance, there may be mismatches between the entries of the two tables, *e.g.* *"Paris"* and *"Paris, FR"*. Sometimes, the entries may be ambiguous: *"Paris"* can be the capital or France, or a city in Texas. To avoid wrongfully matching the two, we may need to look at other attributes, such as the country. Even with clean entries, the columns used for joining may have different names, *e.g.* *"City of residence"* and *"City"*.

TABLE JOINS AND FEATURE ENGINEERING Besides record linkage, **a more fundamental hurdle to table joining is the irregular nature of data**. Indeed, tabular data requires instances (*e.g.* cities) to be described by a *fixed* set of features. In practice however, many features come with a varying number of values across instances. For instance, we may know the salary of each inhabitant in different cities, but we cannot directly merge this information as each city comes with a different number of inhabitants. Instead, the typical practice is to **aggregate the irregular information into a *fixed* number of handcrafted features**, *e.g.* taking the *mean*, *median*, and *standard deviation* of salaries. Engineering the right features from irregular data is usually difficult, as it depends on the task at hand and may require domain-expertise. Further issues arise when dealing with different data structures (*e.g.* time series, text, graphs or images) that must be

transformed into a proper tabular format to enable information merging. Table joins are closely related to the concept of *denormalization* in databases, which aims to integrate data scattered in several tables into a single one to improve read performance.

1.2.3 *Data quality*

After organizing the raw data into a single table, the analyst must identify and fix imperfections that may hinder further analysis. Unlike data organization, data quality procedures focus on repairing data values, rather than changing the structure of the data. We give in this section an overview of common data quality issues, such as non-standardized, missing, or anomalous data.

1.2.3.1 *Data standardization*

Data standardization (or normalization) includes any procedure which **converts entries that can have more than one possible representation into a *standard* format**. This typically occurs when the data has been aggregated from sources with different data collection protocols, *e.g.* employee data across different employers and countries.

ENTITY MATCHING Starting at the entity-level, a first form of standardization is *entity matching*, which aims to **assign a common representation for each unique entity of a categorical feature**. For instance, employee data may contain various entries denoting the same job, such as *"postdoctoral student"* and *"postdoc"*. Although entity matching is closely related to deduplication and record linkage, it is different in that it focuses on matching *entries* in a feature (*e.g.* job titles), rather than whole *instances* (*e.g.* employees), which often requires to leverage multiple attributes (not only the job title, but also the hiring date or the name).

FEATURE STANDARDIZATION At the feature-level, standardization aims to represent all the values of a feature **with the same format**. For example, hiring dates in our employee dataset may be formatted in various ways: *"25 Mar 2020"*, *"25/03/20"*, *"2020-03-25"*, etc... This can often be solved via regular expressions that extract and reorganize relevant parts in the entries, but require time and skill from the analyst. Another common feature standardization step is to ensure values are in the same unit, *e.g.* converting employee salaries in dollars to euros.

DETECTING PROTOCOL CHANGES In many cases, data analysts are directly provided with a single dataset aggregated from multiple tables, with no knowledge of the original data sources. Yet this information is useful to data standardization, as we want to look for variations *across*, rather than *within* sources. For instance, missing values could be encoded with a "0" or a "NaN" depending on the source.

With only the aggregated data, we may wrongly think that these values carry different meanings. Methods detecting sudden changes in the data (*e.g.* no "NaN" values before and no "0" after a certain point) are thus helpful to retrieve the original sources and avoid this confusion.

In the end, the goal of data standardization is to obtain a table where all entities are represented uniquely, and all features follow a standard format.

1.2.3.2 *Missing data*

Missing values, emerging either from the raw data or as a side-product of data organization (*e.g.* an incomplete table join), are challenging for machine-learning models which require **complete** data. To address this, an analyst must typically go through the following steps:

— **Detection**: How are missing values encoded in the data? While entries such as "NULL", "NaN" or "NA" are relatively easy to identify, missing values can also be *disguised* as regular values, *e.g.* a "-1" in a feature describing ages.

— **Understanding**: What is the missingness mechanism generating the data? The seminal work from Little and D. B. Rubin, 1987 identifies three missingness patterns: 1) Missing Completely At Random (MCAR), 2) Missing At Random (MAR) and 3) Missing Not At Random (MNAR), based on whether missingness depends on observed or/and unobserved features, or none.

— **Handling**: How should we deal with missing values? Standard methods either remove or impute them with plausible values (Van Buuren and Groothuis-Oudshoorn, 2011). However, recent works suggest that flexible-enough supervised models can directly learn with missing values without imputation (Josse et al., 2020; Le Morvan et al., 2020).

1.2.3.3 *Anomalies*

Data is said anomalous when it does not follow a certain expected "normal" behavior. Anomalies can arise from a myriad of reasons, such as errors in the data collection pipeline or malicious activities (*e.g.* fraud). They are generally harder to detect than missing values, as they are not explicitly encoded in the data, and can be of various nature: *syntactical* (*e.g.* a value whose type is different from other values), *semantical* (*e.g.* a negative salary) or *statistical* (*e.g.* a value whose probability is too low given the feature distribution). In some cases, we may have to look at several features to identify anomalous values, such as a hiring date that is inconsistent with the age of an employee.

Anomalies are then typically handled through removal or imputation. When imputing, finding a meaningful value for replacement

can be difficult, as the mechanisms leading to anomalies are often complex. For example, fixing a typo in a zip code requires to look at the reported city or state to ensure there are no inconsistencies. However, as with missing values, we may be able to learn directly over anomalies with flexible-enough models.

We give in Table 1.1 a summary of all the data engineering issues described above. In the following section, we precise the focus and objective of this thesis: facilitating data assembling over relational data via entity embeddings.

Table 1.1 – Summary of data engineering problems (adapted from Nazabal et al., 2020). Highlighted issues are those that we tackle in this thesis.

Data Organization	
Data Parsing	What is the original data format? How can it be read properly?
Data Dictionary	
Table description	Are there several tables in the data?
	What are they about? What is their size?
Feature description	What is the type and meaning of each feature?
Value description	What is the meaning of feature values?
	What is their range and unit? Are there unexpected values?
Data Integration	
Table joining	How do we detect common entities across tables?
	How do we aggregate irregular data into a fixed set of features?
Table unioning	Are the features consistent across tables?
	How do we detect duplicate instances in the final table?
Data Quality	
Data standardization	
Entity matching	Do some values in a feature denote the same concept?
	How do we group them under a common representation?
Feature standardization	Are feature values encoded in different formats or units?
	How do we represent them with a standard format?
Protocol changes	Is the data assembled from different sources?
	Do the properties of the data change while exploring it sequentially?
Missing data	
Detection	Which values are used to represent missing data?
Understanding	What it the underlying mechanism: MCAR, MAR or MNAR?
Handling	Should we remove, impute, or leave missing values as such?
Anomalies	
Detection	How do we identify anomalous entries in the data?
Handling	Should we remove, impute, or leave anomalous values as such?

1.3 EMBEDDING MODELS FOR RELATIONAL DATA ANALYTICS

Our objective in this thesis is to leverage embedding models to facilitate data analysis over relational data. In the following sections, we first precise what relational data is and the data engineering challenges it raises that we aim to tackle. We then briefly introduce *statistical relational learning*, a parallel strand of research for learning over relational data, and explain why we depart from it. Finally, we expose how we can leverage entity embeddings to facilitate data assembling over relational data.

1.3.1 *Data assembling for relational data analysis*

1.3.1.1 *Relational versus propositional data*

We first highlight the main differences between *relational* and *propositional* data, as we use the former concept throughout this thesis. In the database literature, **propositional data consists of a single table describing entities of a certain type (*e.g.* cities) with the same features** or attributes. It is the standard input to most statistical learning methods.

On the other hand, **relational data describe entities of different types and how they are related to each other**. A good example are ***relational databases***, which use multiple tables to represent different entities and their relationships, *e.g* a table for cities, another for states, and a third table linking cities to the states they belong to. Another common representation are ***multi-relational graphs***[4] (or *knowledge graphs*), which consist of a set of triples (h, r, t). Each triple indicates a relationship r between entities (h, t), *e.g. (Paris, locatedIn, France)*.

An asset of relational data is that it allows to **inject more information about entities via their relationships**. For example, to better predict housing prices in cities we may want to use information about their states, taken from related tables or large knowledge graphs such as YAGO (Mahdisoltani et al., 2013; Pellissier Tanon et al., 2020) and DBPedia (Lehmann et al., 2015). In fact, **most datasets come as relational data**.

1.3.1.2 *Data assembling is key to successful analyses*

Although relational data contain rich information, it must first be assembled into a single table for analysis, a process sometimes known as *propositionalization* (Lachiche, 2017). In tables, propositionalization relies on two operations: table joins and unions (Section 1.2.2.3).

Our goal here is to facilitate such data assembling because it is often a time-consuming, yet crucial step for data analysis: **by extending the pool of information available to downstream analyses, data**

4. A common distinction between relational databases and knowledge graphs lies in the *closed* versus *open* world assumption. In relational databases, every missing fact is supposed to be false (closed world), whereas it is considered potentially true in knowledge graphs (open world), due to their incompleteness.

assembling is essential to the validity of their results. For example, unioning tables from different sources (which may exhibit biases when taken alone) leads to more diverse data from which we can draw more general conclusions. Similarly, joining new features to a table allows for more precise analyses, *e.g.* improving prediction performance, or measuring the influence of a specific feature on the target variable.

For all these reasons, we focus on relational data assembling, and identify below two common problems that we aim to address.

1.3.1.3 *Problem 1: entity matching in table unions*

The first data assembling problem we consider is **entity matching**, which typically occurs after a table union. Although table unioning is not difficult in itself (as long as features are consistent across tables), it often leads to data standardization issues when the tables come from different sources (Section 1.2.3.1). Among them, **entity matching is generally the most tedious one**, as it involves much more operations (*e.g.* one per entity) than those needed at the feature-level. Our goal is thus to avoid entity matching when doing data analysis over non-normalized sources, which is the focus of Part 1.

1.3.1.4 *Problem 2: feature engineering for table joins*

The second problem we consider is **feature engineering across relational data to enable table joins**. Indeed, joining several tables into a single one is often difficult due to the irregular nature of the data. For example a certain attribute (*e.g.* the population) may not be available for all cities. **A characteristic of relational data is also the presence of one-to-many and many-to-many relationships**, *e.g.* a city linked to all the people that live in. Joining features through such complex relationships calls for tedious feature engineering (Figure 1.2.2.3), which we address in Part 2.

1.3.2 *Beyond feature vectors: statistical relational learning*

While most statistical learning models expect data in the form of points in a high-dimensional space, the field of *statistical relational learning* aims to **learn over complex relational structures *directly***. We briefly introduce here this parallel strand of research, and explain why we depart from it.

1.3.2.1 *Learning with logic*

One of the first methods for learning over relational data is *inductive logic programming* (Cropper et al., 2022; Muggleton, 1991). Given background knowledge (*i.e.* known facts about entities) and a set of training examples, it learns a logic program (*i.e.* a combination of logical rules) that captures a certain relationship between entities. For instance, a model trained to determine whether a person A is the

daughter of person B (given the relations PARENT and WOMAN as background knowledge) will learn the following logic program:

$$\text{PARENT}(B, A) \wedge \text{WOMAN}(B) \implies \text{DAUGHTER}(A, B)$$

Compared to most machine-learning approaches, inductive logic programming offers some attractive features. The expressiveness of logic programs, alongside the possibility for the analyst to easily inject relational inductive biases (*e.g.* constraints) as background knowledge allows them to model complex relations that statistical methods would struggle to learn from few examples. In addition, the learned logic programs can be easily understood by humans. Unlike statistical models however, inductive logic programming cannot deal with uncertainty arising from noise or missing information in the data. Besides, it quickly becomes intractable on large datasets, due to its combinatorial nature.

1.3.2.2 *Dealing with uncertainty*

To bridge the expressiveness of relational logic (as in inductive logic programming) with probabilistic modeling (as in classic statistical learning), the field of statistical relational learning has emerged. It comprises many approaches which typically adopt a graph point of view or a probabilistic version of logic. (Kimmig et al., 2012; Koller et al., 2007; Sutton and McCallum, 2012).

A good example are Markov Logic Networks (Richardson and Domingos, 2006), which aim to describe observed data through a set of logical rules. Unlike traditional logic though, these rules act as soft constraints on the set of possible observations, rather than hard constraints: an observation that does not follow a logic rule is not impossible, but simply less likely. Each logical rule is associated to a weight which encodes the strength of the soft constraint, and is learned on the observed data. For instance, to describe cancer occurrence in a population, we may use the rule $\text{SMOKER}(X) \implies \text{CANCER}(X)$ with a high weight to encode the fact that smokers are more likely to develop cancer.

1.3.2.3 *Limitations of relational models*

Although these statistical models can be readily formulated on relational data, **their biggest limitation is the computational complexity of inference**, which is often non-polynomial with regards to the size of the database (Khosravi and Bina, 2010; Suciu et al., 2011). To avoid this, approximate inference is generally performed, for instance via Markov chain Monte Carlo methods in Markov Logic Networks. In addition, **statistical relational learning tools typically require tailored models**, whereas standard and efficient statistical models can be readily used when provided with a tabular representation of the relational data.

We thus depart from the statistical relational learning literature, and instead aim to learn vectorial representations from relational data, to serve as input to machine-learning models.

1.3.3 *Towards entity embeddings for relational data analytics*

We aim in this thesis to show that vectorial representations (*a.k.a.* embeddings) of entities can facilitate relational data analysis by alleviating the need for tedious data assembling. In the following paragraphs, we detail how entity embedding models can be helpful for the two data assembling problems we identified: *entity matching* across non-normalized sources, and *feature engineering* over relational data.

1.3.3.1 *Embeddings to replace entity matching*

When dealing with data assembled from non-normalized sources, the typical practice is to match entries denoting the same concept. Instead, we argue that **simply exposing similarities between entries is often enough for analysis**: given a vector representation for each entry, we want those denoting the same entity to be close, but not necessarily identical (as in exact matching). An analytical model can then leverage these similarities to implicitly account for matching. For instance, a nearest-neighbor model queried to estimate the salary of a "postdoc" will use the salary of the closest category in the data (*e.g.* a "postdoctoral student") without explicit matching.

The main challenge is then to obtain "good" vector representations that capture well similarities between entries. For this, embedding models seem to be a promising solution. In natural language processing, word embeddings which capture *semantic* or *morphological* similarities (*i.e.* two words with similar *meanings* or *spellings* have close vector representations) have led to breakthroughs in text analyses (Bojanowski et al., 2016; Mikolov et al., 2013a). Following this trend, we study in Part 1 whether embedding models are helpful for data analysis over non-normalized sources, and show that they can alleviate the need for manual entity matching.

1.3.3.2 *Embeddings to join information across relational data*

As we have seen, joining information over relational data often requires difficult feature engineering. Most existing methods to automate this process greedily explore tables to generate and join many features, as in a systematic denormalization (Kanter and Veeramachaneni, 2015; Lam et al., 2017). Yet their combinatorial nature raises scalability issues: the number of created features quickly explodes on large databases.

Instead, **embedding models can be directly trained on such unassembled data to provide compact, low-dimensional feature vectors for each entity**. Indeed, they typically input data in a format simple and expressive enough to integrate information from various sources. For example, word embedding models input pairs of words labeled positively if they co-occur in a sentence, and negatively otherwise. We can adapt this notion of co-occurrence to tabular data (*e.g.* two entities co-occur if they appear in the same row) to easily integrate

irregular information scattered across tables into a single list of entity pairs. Entity embeddings can then be trained in a self-supervised way to implicitly capture the information present in the data. Similarly, knowledge graphs use an entity-attribute-value representation to conveniently store information from heterogeneous sources, based on triples (h, r, t) indicating relationships between entities.

In Part 2, we investigate whether knowledge graph embedding models can be used to automatically join information across relational data, and show that they are efficient and scalable feature extractors.

Part 1

DATA ANALYTICS ACROSS NON-NORMALIZED SOURCES

Summary

Data analysis is increasingly performed over data assembled from uncontrolled sources, facing inconsistencies that call for data integration. In this part (based on our study from Cvetkov-Iliev et al., 2022a), we specifically consider the problem of *entity matching*, which strives to assign a unique representation to entries that denote the same underlying entity. We first present in Chapter 2 the typical practice for analytics over non-normalized sources: creating "clean" data via entity matching, and querying it to estimate analytical quantities of interest. To illustrate this approach, we also introduce an experimental data analysis setup: a socio-economic study of salaries across 14 employee databases. We then detail in Chapter 3 our proposal: rather than relying on tedious entity matching, leveraging machine-learning models trained on "good" vector representations (*i.e.* embeddings) of entries allows to estimate analytical quantities of interest without cleaning (see Figure 3.1 for an overview of the analytical pipeline). Finally, we evaluate in Chapter 4 whether analyses with less cleaning are trustworthy. We answer this question on our study of employee salaries, and compare the approaches based on machine learning and embeddings to manual entity matching. It reveals that the former improves results validity (*i.e.* smaller estimation errors) more than manual cleaning, with considerably less human labor. While machine learning is often combined with data management for the purpose of cleaning, our study suggests that using it directly for *analysis* is beneficial because it captures ambiguities hard to represent during curation.

2

STANDARD PRACTICE: CLEANING FOR ANALYTICS ACROSS SOURCES

2.1 INTRODUCTION

We introduce in this section the subject of this first part: data analysis over non-normalized sources, which is based on the study we conducted in Cvetkov-Iliev et al., 2022a. We start by presenting the data integration problems arising from non-normalized data, with a focus on entity matching. We then state our proposal: coupling advanced statistical models with entity embeddings to answer analytical questions without cleaning; and also present our experimental analysis setup for evaluation: a study of salaries across 14 employee databases. Finally, we give an outline of the different chapters of this part.

2.1.1 *Standard analytical practices call for tedious entity matching*

2.1.1.1 *Data standardization issues in non-normalized data*

Data analysis is increasingly performed across non-normalized sources, facing data standardization challenges at several levels (Section 1.2.3.1). Correspondences are first needed at the feature or **schema level**: similar columns may have different names or formats across tables, and information may be split differently across columns (Doan and Halevy, 2005). Bridging such mismatches is known as *schema matching*, and is often required as a data-preparation step. However, it tends to burden less the human operator as the number of operations to perform is relatively low, *e.g.* a few per feature. At the **entry level**, a much more challenging issue is *entity matching*, where entries representing the same concept are written differently across sources, *e.g.* "professor", "prof", or "professeur". Indeed, the number of matches to check (easily hundreds or even thousands) is generally much higher than that of features. We thus focus on entity matching in the remainder, though there are many other aspects to data quality (see Section 1.2, Ilyas and Chu, 2019; Nazabal et al., 2020).

2.1.1.2 *Entity matching: what is it and why is it needed?*

Entity matching aims to assign a common representation to all entries of a feature that denote the same underlying entity (see Figure 2.1). Importantly, this cleaning procedure is crucial to the validity of data analyses, as **standard analytic methodologies still heavily rely on entity matching**. To illustrate this, we list here a few examples from the literature: estimating product prices from web offers requires to match those referring to the same product (Agrawal

and Ieong, 2012); studying the influence of climate warming on plant species must overcome variability in plant names (Bjorkman et al., 2018; Nazabal et al., 2020); and early detection of acute kidney injuries faces the heterogeneous vocabulary of clinical notes (Li et al., 2018).

2.1.1.3 *Entity matching is difficult, can we do without?*

Even though this curation process can be partly automated (Christophides et al., 2015; Zhao and He, 2019), and despite the availability of dedicated data-integration softwares (Kandel et al., 2011; Stonebraker et al., 2013; Verborgh and De Wilde, 2013), entity matching remains a difficult task as it often involves domain expertise or faces the lack of clear correspondences in entities across sources. We thus aim to answer the following question: **is such matching necessary to the validity of the analysis, or can more complex analytical pipelines do without?**

2.1.1.4 *Our experimental data analysis setup*

To answer this question, we take as experimental setup a **socio-economic analysis of salaries across 14 employee databases**. We draw examples of analytical questions from studies of the determinants of salary, in data journalism (Texas Tribune, 2021) or academia (Blau and Kahn, 2017; Ciscel and Carroll, 1980; Xiao, 2002). In particular, we aim to answer three quantitative questions:

1. For a given job, how does salary evolve with experience?
2. For a given job, what is the 0.75-quantile of salaries?
3. What is the typical male-female pay gap?

Answering these questions from data assembled across different employers must overcome the lack of correspondences in job titles: as illustrated on Figure 2.1, the same job title appears with multiple variants, such as *senior research associate* and *sr research assoc*.

2.1.2 *Our claim: advanced statistical models can answer analytical questions without cleaning*

In this part, we show that **applying advanced statistical techniques directly to non-normalized data can avoid labor-intensive data cu-**

Job Title	Experience	Salary
0712 - postdoctoral fellow	1	65k
data scientist	3	90k
senior research associate	8	110k

Job Title	Experience	Salary
professor	5	72k
sr research assoc	4	100k
postdoctoral research associate	2	49k

Figure 2.1 – Entity matching across two employee databases.

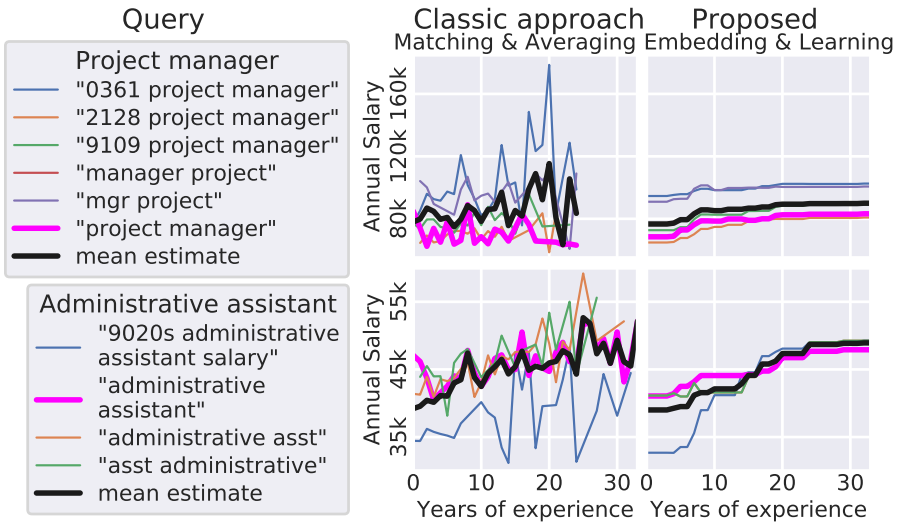

Figure 2.2 – **An analysis of salary as a function of experience for different job types**, considering variants of *administrative assistant* and *project manager*, computed by matching & averaging, or embedding & learning. Thick magenta lines are the most natural query.

ration for many analytical questions. On our socio-economic study of salaries across 14 employee databases, we benchmark whether relying more on machine learning and less on manual data cleaning compromises or not the validity of the analysis. To answer this important question, we formalize how many analyses boil down to estimating statistical quantities, and use various experiments that give an unbiased measure of the corresponding estimation error.

The quantities needed for the analytical questions can be estimated with machine-learning models applied to continuous embeddings of entries that represent ambiguities. A suitably trained model can be directly queried to give, for instance, the evolution of salary with experience for a given job, leading to a less noisy result than standard techniques after best-effort manual cleaning (see Figure 2.2). Machine learning is already increasingly used in data integration to create more uniform data warehouses (Dong and Rekatsinas, 2018; Stonebraker and Ilyas, 2018) or to clean their entries (Ilyas and Chu, 2019; Krishnan et al., 2016). Instead, our study applies it directly to the analytical question, as this can be easier than curating the data for fundamental reasons. First, the analytic task provides **supervision** (Berti-Equille, 2019; Krishnan et al., 2017, 2015), while cleaning needs examples of curated data. Second, **representing ambiguities in the analysis often leads to more accurate results**.

2.1.3 *Chapters outline*

We first present in Chapter 2 the standard practice for analytics over non-normalized entries: creating "clean" data via entity matching, and querying it to estimate analytical quantities of interest (Section 2.2). To illustrate this approach, we also introduce our experi-

mental data analysis setup: a socio-economic study of salaries across 14 employee databases. We then detail in Chapter 3 our proposal, and show how many data-science questions can be formulated in terms of machine learning on embeddings of entries (see Figure 3.1 for a overview). Finally, we study in Chapter 4 the validity of the results: on an analysis of wages across 14 data sources, we compare manual data cleaning to a simple machine-learning approach using embeddings of entries. Qualitatively and quantitatively, analyzing the non-normalized data gives better results. Finally, we discuss perspectives on adapting data-analysis practices to rely less on cleaning.

2.2 THE CLASSIC VIEW: CLEANING FOR ANALYTICS ACROSS SOURCES

We introduce in this section the classic view for data analysis across sources: **cleaning the data and querying it to estimate analytical quantities of interest**. Starting with cleaning, we explain why and how entity matching is done in practice, from manual to more sophisticated methods. We then present the standard estimation technique: "matching & averaging", and its limitations.

2.2.1 *Entity matching: why and how?*

2.2.1.1 *Answering conditional questions on non-normalized categories requires matching*

Fundamentally, the motivation to unite data sources is to **establish a more general result**: in our example analysis, salaries or the male-female pay gap may vary across employers. Yet, whether an analysis across sources requires entity matching or not depends on the analytic question at hand.

When answering questions ***conditionally*** to a non-normalized category, *i.e.* analyzing one quantity –such as the salary– while keeping another –the job title– constant, matching entries to uncover the category of interest is necessary to standard analytical procedures (Cerda et al., 2018). Conversely, computing ***marginal*** quantities, such as the overall distribution of salary, does not require entity matching as it relies on aggregates of *all* the data (assuming that there are no duplicates across the sources).

2.2.1.2 *Entity matching procedures*

Entity matching strives to match entries of a feature denoting the same entity. Closely related problems include the *deduplication* of duplicate records within a table (Elmagarmid et al., 2007), or *record linkage*, matching records describing the same entity across two tables (Winkler, 1999). **Entity matching techniques traditionally rely on an appropriate string similarity metric** (*e.g.* Levensthein, Jaro-Winkler, n-grams...) **and a threshold to assign entries to the same entity.** The issue is that both the similarity and threshold must be tailored to do-

main specificities and the resulting matches must often be manually reviewed. The process is thus labor intensive.

Automating the match of entities is challenging because it is an unsupervised-learning problem (Fellegi and Sunter, 1969), unless there are known matches for supervision (Bilenko and Mooney, 2003; McCallum et al., 2005; Mudgal et al., 2018). Such matches must typically be constructed manually by a user, though active learning can reduce human intervention (Krishnan et al., 2016; Sarawagi and Bhamidipaty, 2002). Dedicated data-integration softwares, such as OpenRefine (Verborgh and De Wilde, 2013), facilitate this process with a user interface. The software searches for potential matches and enables users to tune search parameters and match suggested entries in a semi-automated way.

Automation tools can use techniques capturing natural language semantics, which shares with entity matching the challenge of relating multiple forms that denote the same things. For instance, natural language processing tools such as `fastText` (Bojanowski et al., 2017) provide word embeddings resilient to morphological variations. Embeddings have led to many recent progresses in entity-matching pipelines (Ebraheem et al., 2018; Kasai et al., 2019; Mudgal et al., 2018; Zhao and He, 2019).

2.2.2 *The standard analytical practice: "matching and averaging"*

2.2.2.1 *Matching and averaging*

In general, analytic questions can be formalized as estimating a quantity y for a population or group of instances who share a set of attributes X, for instance the typical salary of a *project manager* with 3 years of experience. To that end, the standard procedure consists in ***matching & averaging***:

1. A **query** on X to **match** and select the relevant instances.
2. A procedure (typically a form of averaging) to **aggregate** the results and estimate y.

Even when the entries in the data are normalized, a successful analysis may require to **match them with the vocabulary used by the analyst**: for instance in some data the correct query for *"project manager"* may be *"mgr project"* (Figure 2.2).

2.2.2.2 *Beyond matching: leveraging similar entities for analysis*

The underlying problem behind matching & averaging is that of *statistical estimation*: **computing the quantity that represents best the complete population from the instances at hand**. If the entity matching is valid, matching & averaging estimates are unbiased from a statistical point of view. Yet they may exhibit high variance if the data contain a small number of representatives for the category of interest.

A paradox of statistics is that the most accurate way of estimating the mean of a population from a small sample may not be the sample average, but biasing estimates with other sources of information

Table 2.1 – A few example rows of the employee data.

Job Title	Hiring Date	Sex	Ethnicity	Salary ($)
Police Officer	17/03/2005	M	White	85 000
Security Manager	24/06/2017	F	Asian	70 000
Energy Analyst	04/11/1998	F	Black	105 000
Librarian	11/09/2011	M	Hispanic	50 000

(see Stein's paradox, Efron and Morris, 1977). For instance, estimating the typical salary of an *associate professor* may benefit from data from similar populations: *professor*, *lecturer*. Aggregating information across similar entities is related to the notion of semantic queries in databases (Bordawekar and Shmueli, 2017), as opposed to exact value matching, used in matching & averaging.

2.3 APPLICATION TO OUR EXAMPLE STUDY OF SALARIES

We illustrate here the analytical procedures described in the previous section on our example study of salaries (Section 2.1.1.4). After briefly introducing the dataset, we detail how we performed entity matching on the non-normalized job titles. We then show how the analytical questions we consider can be seen as estimating certain statistical quantities, and provide the corresponding matching & averaging estimators.

2.3.1 *The data: employee salaries across 14 employers*

To answer our questions on salary, we consider data from a study of salaries in Texas state institutions [1]. The data consist of the union of 14 tables from different employers, and describe the salaries, job titles, hiring dates, genders and ethnicities of 160 000 employees (Table 2.1). The Job Title information is particularly challenging: **without normalization there are about 14 000 different job positions**.

2.3.2 *Matching job titles for analysis*

We performed manual entity matching on the job titles using OpenRefine. The first step was to **clean common abbreviations** as they are the main hurdle to entity matching: string metrics struggle to capture their similarities. Typical examples included *sr*/*senior*, *asst*/*assistant* or *mgr*/*manager*, which we cleaned using complex regular expressions. An issue was that some abbreviations had multiple meanings. For instance *tech* was short for either *technology*, *technician*, or *technical* depending on the context. Similarly, some abbreviations had complex meanings, *e.g. CMC* for *Chemical, Manufacturing and Control*. Such manual cleaning is thus limited by the domain expertise of the operator.

1. Data available at https://dx.doi.org/10.21227/wfjs-ya22.

We then used OpenRefine to search and manually merge variants across sources. **Around 1000 job titles were paired in the process, which took about 3 days** (including the abbreviation cleaning). We believe that a more thorough entity matching, especially on rare job titles, could have been performed, but would have required intensive human labor to bring minor improvements.

2.3.3 *Analytic question 1: salary evolution with experience*

Our goal here is to estimate the mean salary as a function of work experience for a given job, which amounts to estimating the conditional expectation $\tau = \mathbb{E}[\text{Salary} \mid \text{Job, Experience}]$. For this, the matching & averaging procedure forms groups $G(j, e)$ of employees with job j and experience level e, and averages employee salaries y_i in each group.

$$\hat{\tau}_{\text{matching}}(j, e) = \frac{1}{|G(j, e)|} \sum_{\substack{1 \leqslant i \leqslant n \\ i \in G(j,e)}} y_i \tag{2.1}$$

2.3.4 *Analytic question 2: salary quantiles*

We are also interested in the distribution of salaries among employees with job j. More precisely, we aim to estimate the 0.75-quantile, *i.e.* the salary $\tau(j)$ so that 75% of employees with job j earn less than $\tau(j)$. To that end, matching & averaging groups employees by their jobs, and then computes the empirical 0.75-quantile of salaries $\hat{\tau}_{\text{matching}}(j)$ for each group.

2.3.5 *Analytic question 3: causal effect of gender on salary*

Many of the advanced analyses and visualizations of rich data sources pertain to understanding causes and effects. For instance, when studying salaries, measuring and understanding the causes of gender gap is a long-running question (Blau and Kahn, 2017; Blinder, 1973; Oaxaca, 1973).

Counterfactual analysis provides a good framework to address quantitatively the question of gender pay gap. A counterfactual is a thought experiment measuring the effect on the outcome of interest —the salary y_i— of changing only the feature of interest W_i —here the sex— for an individual i. Borrowing from clinical trials, W_i is called the "treatment" in the literature. The outcome y_i can take two potential values depending on W_i: $y_i(0) = y_i(W_i = 0)$ or $y_i(1) = y_i(W_i = 1)$ (1 for a man, 0 for a woman), though for each individual only one of these is observed in the data.

The analytical quantity of interest is the typical gender pay gap, known as the ***average treatment effect*** (ATE) $\tau = \mathbb{E}[y(1) - y(0)]$: the average difference in outcome for the same individual under scenario $W_i = 1$ and $W_i = 0$, *i.e.* that differ only by their sex (Imbens and D. B.

Rubin, 2015; D. Rubin, 1974). If we had access to $y_i(1)$ and $y_i(0)$ for each employee $i \in \{1, 2, ..., n\}$, we could easily estimate the ATE:

$$\widehat{\tau} = \frac{1}{n} \sum_{i=1}^{n} y_i(1) - y_i(0) \tag{2.2}$$

In practice though, we either observe $y_i = y_i(1)$ or $y_i = y_i(0)$ in the data, but not both at the same time. In a randomized trial, where the treatment W is assigned independently of the outcome y, we could simply subtract the average salary of men from that of women: $\mathbb{E}[y|W = 1] - \mathbb{E}[y|W = 0] = \hat{\tau}$. However, this is not applicable in general: *confounding factors* could make the two distributions (men and women salaries) not directly comparable and bias the results. For instance, due to maternity leave, women may have less work experience than men in general, and thus lower wages as a whole. Taking the difference in mean salaries would thus indicate an ATE > 0, even though wages between men and women with the *same* experience level could be identical (ATE $= 0$). To account for such confounding factors and isolate the treatment effect, the *potential outcome* framework (Table 2.2) uses *covariates*, extra information X on each individual, such as the job title or the experience level, allowing us to contrast salaries for men and women with the same covariates.

To infer the unobserved value $y_i^{\text{unobs}} = y_i(1)$ or $y_i(0)$ in Equation 2.2, we can then leverage data from employees with similar profiles, *i.e.* covariates, but of opposite sex. For this, matching & averaging considers the set O_i of employees with the same covariates as employee i but of opposite sex (which requires matched job titles), and take their average salary to estimate y_i^{unobs}. Note however that this is not always possible: some employees may have no counterpart of the opposite sex in the data. In practice, we thus consider only the set M of employees for which O_i is not empty, which may bias the ATE estimate or increase its variance:

$$\widehat{\tau}_{\text{matching}} = \frac{1}{|M|} \sum_{i \in M} W_i(y_i - \widehat{y_i}^{\text{unobs}}) + (1 - W_i)(\widehat{y_i}^{\text{unobs}} - y_i) \tag{2.3}$$

with $\widehat{y_i}^{\text{unobs}} = \frac{1}{|O_i|} \sum_{k \in O_i} y_k$

Table 2.2 – **Illustration of the potential outcome framework.** The data contains an observation of a male white firefighter with 15 years of experience, but not a matching female employee; likewise with an hispanic female post-doc with 2 years of experience. The challenge is to interpolate the missing data.

Covariates X			**Treatment**	**Outcome**	
Job Title	Experience	Ethnicity	**W** (sex)	**y(0)**	**y(1)**
Firefighter	15	White	1	NA	75000
Post-doc	2	Hispanic	0	60000	NA

3

ANSWERING ANALYTICAL QUESTIONS WITH MACHINE LEARNING

We describe in this chapter how machine learning models, when trained on "good" vector representations of the entries, can answer analytical questions without cleaning (see Figure 3.1 for an overview of the pipeline). First, we show that these models can be used to estimate various statistical quantities (Section 3.1). We then explain how embeddings capturing relevant similarities between entries can replace matching, and detail our practical implementation (Section 3.2). Finally, we briefly conclude on the potential of machine learning for data analysis and integration (Section 3.3).

3.1 MACHINE LEARNING TO ESTIMATE STATISTICAL QUANTITIES

3.1.1 *Salary evolution and quantiles*

As seen in section Section 2.3.3, matching & averaging methods study the evolution of salary with experience by grouping employees by job and experience level, and computing the mean salary in each group. This quantity is an estimate of the conditional expectation $\tau = \mathbb{E}[\text{Salary} \mid \text{Job, Experience}]$.

Instead of averaging on groups of employees, a machine-learning model trained to predict the salary given the job and experience level can estimate this quantity. Indeed, modeling the salary as a function $f_\theta(\text{Job, Experience})$ gives a *consistent* [1] estimate of the conditional expectation $\mathbb{E}[\text{Salary} \mid \text{Job, Experience}]$ if model parameters θ are optimized on the data to minimize the mean squared error on the salary (Bishop, 2006, section 1.5.5):

$$\hat{\theta} = \arg\min_{\theta} \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(j_i, e_i))^2 \right) \tag{3.1}$$

where y_i, j_i and e_i are the salary, job title and experience level of the i^{th} employee. Once trained, we can directly query the model to estimate the desired quantity: $\hat{\tau}_{\text{learning}}(j, e) = f_{\hat{\theta}}(j, e)$.

More generally, **a model can be trained to estimate different statistical quantities by choosing the measure of error (loss) that it minimizes** (Gneiting, 2011). For instance, a model $f_\theta(\text{Job})$ trained with a quantile loss estimates a quantile of the salary distribution for a given job (Koenker and Bassett Jr, 1978):

1. A *consistent* statistical procedure converges to the population values with increasing data size.

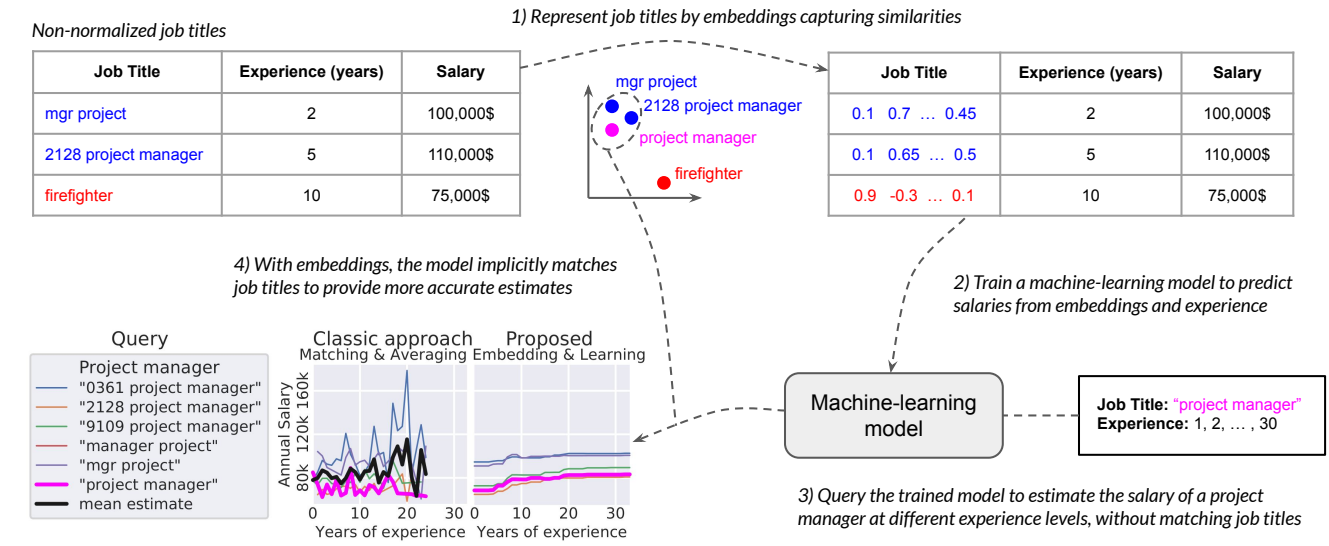

Figure 3.1 – Overview of our "embedding & learning" pipeline to answer analytical questions on non-normalized data.

$$\hat{\theta} = \arg\min_{\theta} \left(\frac{1}{n} \sum_{i=1}^{n} \rho_{\alpha}(y_i - f_{\theta}(j_i)) \right) \tag{3.2}$$

$$\text{where} \quad \rho_{\alpha}(x) = \begin{cases} -x\,(1-\alpha), & \text{if } x \leqslant 0 \\ \alpha\, x, & \text{otherwise} \end{cases} \quad \text{and } \alpha = 0.75$$

3.1.2 *Pay gap across sex: counterfactual analysis*

3.1.2.1 *Outcome regression methods*

To estimate the average treatment effect (ATE), modern causal inference techniques often rely on estimates of the outcome given the covariates and the treatment $\mathbb{E}[y|X, W]$, to be used in place of the unobserved value $y_i^{\text{unobs}} = y_i(1)$ or $y_i(0)$ in Equation 2.2. Instead of estimating this quantity with matching & averaging, we can use a machine-learning model $f_{\theta}^{(y)}$ trained to predict y_i from the employee covariates X_i with a mean squared error. The model outputs can then be plugged in Equation 2.2 to estimate the ATE:

$$\widehat{\tau} = \frac{1}{n} \sum_{i=1}^{n} \big[W_i (y_i - f_{\hat{\theta}}^{(y)}(X_i, W = 0)) + (1 - W_i)(f_{\hat{\theta}}^{(y)}(X_i, W = 1) - y_i) \big] \tag{3.3}$$

3.1.2.2 *Inverse propensity weighting*

Other approaches are based on *inverse propensity weighting*. They rely on estimates of the propensity score $e(X_i) = P(W_i = 1|X_i)$, *i.e.* the probability of an individual with covariates X_i to be a man. For this, we can use machine-learning models $f_{\theta}^{(w)}$, when they are *calibrated* (Niculescu-Mizil and Caruana, 2005), to predict the gender of

an employee from its covariates. We can then estimate the propensity score $f^{(w)}_{\hat{\theta}}(X_i) = \widehat{e}(X_i)$, and the ATE:

$$\widehat{\tau} = \frac{1}{n}\sum_{i=1}^{n} \frac{W_i\, y_i}{\widehat{e}(X_i)} - \frac{(1-W_i)\, y_i}{1-\widehat{e}(X_i)} \tag{3.4}$$

3.1.2.3 *Doubly-robust approaches*

Finally, powerful causal-inference tools combine both estimates for more robustness (Funk et al., 2011). State-of-the-art approaches already rely on machine-learning models to adapt to biases and noise in the input data (Blakely et al., 2019; Chernozhukov et al., 2018).

$$\begin{aligned}\widehat{\tau}_{\text{learning}} = \frac{1}{n}\sum_{i=1}^{n} \big[\widehat{y}_{i,1} - \widehat{y}_{i,0} \\ + \frac{W_i}{\widehat{e}_i}(y_i - \widehat{y}_{i,1}) - \frac{1-W_i}{1-\widehat{e}_i}(y_i - \widehat{y}_{i,0})\big]\end{aligned} \tag{3.5}$$

with salary estimates $\widehat{y}_{i,0/1} = f^{(y)}_{\hat{\theta}}(X_i, W = 0/1)$ and propensity-score estimates $\widehat{e}_i = f^{(w)}_{\hat{\theta}}(X_i)$.

A technical subtlety is that we use a cross-fitting procedure to estimate salaries and propensity-scores (Chernozhukov et al., 2018). Instead of fitting machine-learning models on all the data and then taking models output as estimates, we split samples in K folds and obtain estimates for each fold using models fitted on the $K-1$ remaining folds. The appendix (Section A.1.2) details the exact models used in our experiments to estimate the average treatment effect.

3.2 EMBEDDINGS TO REPLACE ENTITY MATCHING

3.2.1 *Embeddings capturing similarities implicitly account for matching*

Unlike averaging, casting the analytic question into a machine-learning task **does not require matching**. Rather than viewing each job title j as a discrete category, we can represent it by a vector $\mathbf{j} \in \mathbb{R}^p$, that will serve as features to train the machine-learning model f_θ.

The crucial point here is that these **vectors should capture similarities between entries**: related job titles (*e.g. administrative assistant* and *administrative asst*) should have close representations. This allows the machine-learning model to leverage salaries from related jobs to refine its estimates and implicitly account for matching.

3.2.2 *Implementation details*

To illustrate our approach in a way that is not specific to our study of salaries, but rather that could be applied in many applications, we choose in our experiments simple and widely available machine-learning tools.

For the embeddings, we leverage **pretrained fastText embeddings** (Bojanowski et al., 2017), which readily provide vector representations for strings that capture **semantic** and **morphological** similarities. Other approaches to encode string similarities into vectors could be used as well (Cerda and Varoquaux, 2019; Cerda et al., 2018). To estimate the analytic quantities of interest, a wide variety of machine-learning models can be used. For our experiments we rely on **gradient boosted tree models** from *scikit-learn* (Pedregosa et al., 2011), as they generally perform well in prediction tasks.

3.3 A PATTERN: MACHINE LEARNING

Because machine learning can capture complex links in complex data, it is increasingly used in data science to estimate quantities of interest to the analyst, whether they are intermediate quantities, as for counterfactual analysis (Section 3.1.2), or the direct answer to the question of interest, as for conditional links (Section 3.1.1).

For data integration, this evolution brings exciting new opportunities: machine-learning models do not need to rely on averaging, and hence do not need actual matching of entities across sources. Rather, they can use vector representations that express, even indirectly, relevant similarities between entities.

Besides avoiding tedious entity-matching, machine-learning models can form weakly-parametric estimators that are **resilient to noise and other imperfections in the data**. For instance, imperfect correspondences between schemas across the sources lead to missing values: some sources may not have all the information. Despite these missing values, supervised learning can give optimal estimates without relying on probabilistic modeling of the missing-data mechanism (Josse et al., 2020; Le Morvan et al., 2020).

4 LEARNING VERSUS CLEANING

Using machine learning can be less labor-intensive, as it does not require human-guided entity matching. But does it come at a cost to the validity of the results? To answer this important question, we conduct in this chapter an empirical benchmark of analytic methods based on cleaning or learning (Section 4.1). We then discuss in length the implications of these results for data analysis and data management practices (Section 4.2).

4.1 EMPIRICAL STUDY

In this section, we empirically compare learning and matching-based approaches on our three analytic questions of interest. [1]. Starting with a detailed description of our experimental setup, we then present qualitative and quantitative results on the estimation errors of the analytic methods under study.

4.1.1 *Experimental details*

MEASURING THE ESTIMATION ERROR How to compare estimators of a quantity such as conditional expectation of salary given job title? Even without entity-matching noise, the data at hand is limited and its mean is an imperfect estimate of the unknown population quantity y^*. We adapt a classic procedure of machine learning: we leave out a *test* fraction of the databases, and use the rest of the data to derive estimates $\hat{y}_{\text{train}}$. Applying a matching & averaging estimator on the *test* data provides another estimate $\hat{y}_{\text{test}}$, that is unbiased though noisy. Importantly, as it has been estimated from different data than $\hat{y}_{\text{train}}$, its estimation error is independent. We can thus use the difference between $\hat{y}_{\text{train}}$ and $\hat{y}_{\text{test}}$ over multiple splits (as in a cross-validation loop) to quantify the estimation error of the procedure that we use to compute $\hat{y}_{\text{train}}$.

ANALYTICAL APPROACHES STUDIED We compare several approaches to estimate the quantities relevant to our analytical questions:

1. **Matching & averaging**, as described in Section 2.2.2.
2. **Embedding & learning**: strategies of Chapter 3, relying only on standard machine-learning tools. Gradient boosted tree models from *scikit-learn* (Pedregosa et al., 2011) are used with pre-trained fastText embeddings (Bojanowski et al., 2017) to represent the job titles, capturing semantic and morphological similarities.

1. The code and data to reproduce our experiments is available on Code Ocean: https://codeocean.com/capsule/6435573/tree

3. **Embedding & fuzzy matching**: the notion of continuous similarities, as between embeddings, can also be exploited to define weighted averages. We modify the matching & averaging procedure to use fuzzy matches and weights defined with a cosine string similarity on the job title with an affine decay. See the appendix (Section A.1.1) for a more detailed description.

Parameters such as the affine decay, or the hyper-parameters of the machine-learning models are tuned in a nested cross-validation procedure (see appendix). To study the effect of entity matching, we apply these techniques on raw and manually matched entries.

COUNTERFACTUAL ANALYSIS To study the causal effect of gender on salary, we use the following covariates: job title, experience level, ethnicity and the type of employer (city, county, university, hospital). Including these features allows to compare salaries between similar employees and isolate the effect of gender.

Note that the ethnicity feature is also non-normalized: multiple variants for each ethnicity exist in the data (*e.g.* "Black", "BLK", "Black or African American"). When estimating the ATE with manual or fuzzy matching techniques, we thus had to group similar ethnicities into 7 categories. When using machine-learning models for estimation, we simply encoded ethnicities into vectors of dimension 10, using a Gamma-Poisson factorization [2] (Cerda and Varoquaux, 2019). Importantly, these vectors expose nuances that would have been lost in the matching process otherwise, for instance when grouping "Mexican" with "Hispanic or Latino".

ESTIMATION ON UNSEEN CATEGORIES In our experiments, we compute "ground-truth" values $\hat{y}_{\text{test}}$ for groups of employees in the test set, that we then compare to our estimates $\hat{y}_{\text{train}}$ based on the training data. However, some groups of employees in the test set may have no equivalent in the training data, which makes matching & averaging estimation impossible, unlike fuzzy matching and machine-learning pipelines which can leverage data from related groups to form estimates. In these cases, we compute matching & averaging estimates over all employees, regardless of their job titles.

4.1.2 *Qualitative results: dispersion across variants*

The curves of salary as a function of experience represented on Fig. 2.2 are computed either with a matching-based or a learning-based approach. Machine-learning estimates leverage job similarities and have low dispersion across variants of *project manager* or *administrative assistant*. This robustness reduces the need for manual matching: **querying the model on any variant provides reliable estimates** that are representative of the whole population. It is more convenient and reliable for an analyst to query the model for "project manager"

2. An implementation of this approach is available in the dirty-cat package: https://dirty-cat.github.io/stable/ (see GapEncoder)

Table 4.1 – **Cross-validated errors for salary, quantile, and propensity-score estimation.** To be more precise in our analysis, we also measure cross-validation errors on *seen* and *unseen* test entities only (*i.e.* test entities for which we have, or do not have matches in the training data), as matching & averaging estimation is done differently in those cases.
We also report here estimates of the propensity-score $P(W = 1|J)$ conditionally to the job title, rather than on all covariates, as matching-based estimates are very noisy in that case. RMSE = Root Mean Square error. MAE = Mean Absolute Error.

Estimation method	**Manual matching**	**Salary** (RMSE)	**Quantile** (MAE)	**Propensity** (Brier score)
Evaluation on *all* test entities				
Matching & averaging	Yes	55634	31802	0.231
Embedding & Fuzzy matching	No	52812	30955	0.195
Embedding & Fuzzy matching	Yes	51506	28851	0.192
Embedding & Learning	No	52683	28726	0.189
Embedding & Learning	Yes	50614	26713	0.184
Evaluation on *unseen* test entities				
Matching & averaging	Yes	49612	33031	0.259
Embedding & fuzzy matching	No	47232	28538	0.206
Embedding & fuzzy matching	Yes	45762	26533	0.201
Embedding & learning	No	44676	26909	0.197
Embedding & learning	Yes	43300	25719	0.194
Evaluation on *seen* test entities				
Matching & averaging	Yes	69158	29731	0.183
Embedding & fuzzy matching	No	65390	35028	0.176
Embedding & fuzzy matching	Yes	64350	32757	0.176
Embedding & learning	No	69653	31788	0.178
Embedding & learning	Yes	66266	28388	0.169

or "administrative assistant" (thick magenta curves), than to search the database for all variants and average them. Beyond the dispersion across variants, matching & averaging curves appear more noisy; in particular they fail to capture well the evolution of salary with experience. Finally, machine-learning estimates show plausible extrapolations for queries where there is no data with exact matches, such as project managers with more than 25 years of experience.

4.1.3 *Quantitative results: cross-validated errors*

To go beyond the face validity of Figure 2.2, we use cross-validation (as detailed in Section 4.1.1) to quantify which approach best estimates the population quantities. We consider here our three analytical tasks: **1)** salary given job and experience, **2)** 0.75-quantile of salaries for a given job, and **3)** a simplified version of the propensity-score $P(W = 1|J)$ (*i.e.* the proportion of men in a given job), used in

causal analysis (Section 3.1.2). The 14 databases are randomly split into two sets of 7 databases: one to compute estimates for salary, quantile, and propensity-score; and the other to measure their error, reported in Table 4.1. Results show that for all three quantities **embeddings notably reduce the error compared to exact matching and perform best when combined with learning**. Adding manual matching on top of embeddings improves further, but the benefit is smaller than that brought by embeddings & learning. The residual error is due to variance in individual salary that is not explained by the attributes of the employees present in the databases, such as the appreciation of the manager.

To push the analysis further, we distinguish in the rest of Table 4.1 cross-validated errors depending on whether they correspond to estimates on unseen categories (*i.e.* for which there are no matches in the training data) or not. As expected, machine-learning models based on embeddings significantly outperforms manual matching on unseen categories. Indeed, the latter approach is not able to leverage data from related jobs to come up with a reliable estimate. Machine-learning models are thus much more adapted for estimation outside of the original data. On seen categories however, we expect this gap to shrink: with cleaning and enough samples per category, matching & averaging estimates should converge to the true population values. Still, **embedding & learning remains overall competitive with manual cleaning**. Besides, the analytic tasks considered here are rather simple: we study the quantities of interest at a broad level, only conditioning on one or two attributes (*e.g.* job title and experience level). This results in large groups of employees on which matching & averaging provides good estimates. On more granular analyses (*e.g.* estimating the causal effect of sex on salary), or more generally in low-data regimes, we expect embedding & learning estimates to be much more reliable. We investigate this in the next section.

4.1.4 *Estimation of counterfactuals*

How do the differences in estimation errors reported in Table 4.1 impact complex end-user analytical questions? We investigate their impact on estimation of salary gap across sex. Figure 4.1 gives average treatment effects computed with statistical methods based on embedding & learning approaches, as well as manual matching and fuzzy-matching estimates. To force the need for analysis across the databases, we create a sex imbalance by dropping randomly a fraction of either men or women in each database, with 50/50 probability. As a result, the estimation relies on employees of opposite sex with matching covariates across databases. Importantly, embedding & learning estimates are consistent with those obtained after best-effort manual cleaning, but **exhibit much less variance** as fewer and fewer employees with similar covariates but of opposite sex can be matched. This low variance of machine-learning methods comes from their implicit interpolation, visible on Figure 2.2: if a given employee

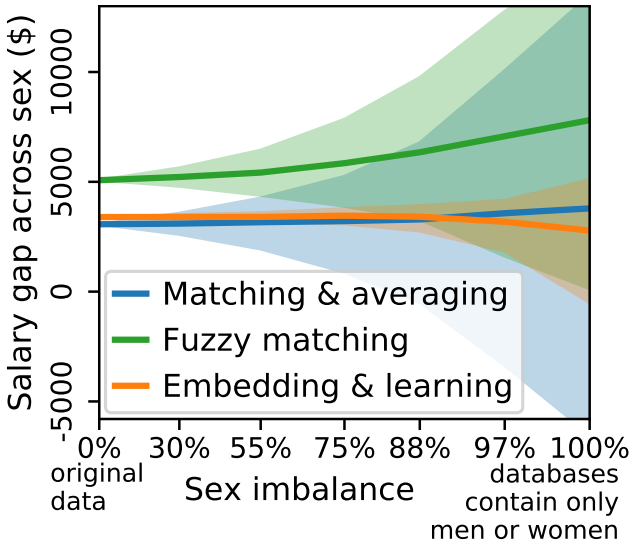

Figure 4.1 – **Salary gap:** Average treatment effect computed with added sex imbalance in individual databases, forcing the need for analysis across databases. The error bars give the quartiles across random deletion of men or women records.

lacks an opposite-sex with the same covariates, the model will use information from similar profiles. On the other hand, fuzzy matching introduces a sizeable bias: its estimates differs markedly from matching & averaging.

4.2 DISCUSSION: HOW MUCH CAN LEARNING REPLACE CLEANING?

On the data-integration problem that we have studied, relying more on learning rather than on cleaning facilitates the data analysis, and actually improves the validity of the results without manual labor. This result departs from classic data-management practices, and we now discuss its interpretation and impact for analytical practices.

4.2.1 *Cleaning is analysis*

Studying the salary gap showcases the importance of analysis across data sources: for the highest-payed positions, finding employees of opposite sex requires considering multiple companies. Matching entities faces the fundamental challenge that there might not be exact correspondences: not every institution has a *chief data officer* (CDO) and the nearest match may be *chief technology officer* (CTO). Omitting companies without CDO will bias the analysis by excluding large tech companies.

The notion of cleaning, to make data more uniform, carries in itself analytical choices which may bias the results (Boumans and Leonelli, 2020; Rawson and Muñoz, 2019). While *vinaigrette* is just French for *salad dressing*, its use on an American's restaurant menu signals upper-scale clientele. Merging the two will lead to loss of in-

formation. From an ontological point of view, the solution would be to create a new category, *posh salad dressing*. But maintaining a complete and consistent ontology, catering for all the edge cases, requires manual work each time new data is integrated. Should the necessity to merge entities be considered as a bug of analytic pipelines, rather than a feature? New tools that do not require exact matches can give more reliable analyses in the face of ambiguity, as illustrated by the estimation of salary gap across databases with sex imbalance (Figure 4.1).

Manually curating entity matching brings to the data a consistency that is good practice in production settings. Yet, as illustrated in our empirical study, favoring more advanced statistics down the line facilitates valid analysis. It can indeed be easier to pass on uncertainties to the statistical analysis tools than to resolve them in a relational store. The best data representation, clean or fuzzy, is tied to the analytic question.

4.2.2 *Supervision facilitates integrating data with ambiguities*

Representing uncertainty in relational systems helps tackling ambiguities (Bordawekar and Shmueli, 2017; Dong et al., 2009; Kimmig et al., 2018) or curating data (Rekatsinas et al., 2017). However, extending relational data management to a general probabilistic framework is intrinsically hard. Indeed, unlike with the relational algebra, queries in a probabilistic database can suffer non-polynomial complexity (Suciu et al., 2011). Approximate probabilities (Bach et al., 2017; Domingos and Lowd, 2009) or fuzzy logic and similarities (Petry, 2012) have better tractability. Yet how to weight similarities to best capture ambiguities is often a challenge in itself.

Using supervised learning to answer a given statistical question alleviates the need for probabilistic models. In particular, many recent success rely on *discriminative* modeling using empirical risk minimization, as with deep learning (Goodfellow et al., 2016). It is crucial to the success of our empirical study: optimizing the statistical models gives accurate estimates from non-probabilistic similarities –word representations that were not tailored to the question at hand. Such an approach goes much further than fuzzy matching (Figure 4.1), as supervised learning can be seen as implicitly tuning scaling factors and thresholds to combine information optimally while minimizing noise.

EMBEDDINGS TO CAPTURE AMBIGUITIES Entity embeddings are crucial to the success of our approach, to expose ambiguities to the analysis step. Our proof of principle purposely used a very simple implementation: a general-purpose machine-learning model applied on off-the-shelf word embeddings. Yet, it leads to analyses on the unaligned data more accurate than standard statistical approaches on data cleaned with three days of manual labor using a dedicated software (Table 4.1, Figure 4.1).

Importantly, the main requirement for our approach to work is that entries denoting the same entity, and more generally entries that have similar properties (*e.g.* jobs with similar salary distributions, or similar men/women proportions) should have close vector representations. While in our example these similarities are mostly morphological (two jobs with similar titles are more likely to be the same or to have similar properties) and can be captured with pretrained fastText embeddings, other analytical studies may require different embedding strategies. For instance data analytics across multi-lingual sources would require embeddings that group together words with similar meanings in different languages (X. Chen and Cardie, 2018). Likewise, domain-specific entities for which general-purpose embeddings (*e.g.* fastText) are not adapted may require training them from the data at hand to adapt to its specificities. Such training can be done via the string forms of entities (Cerda and Varoquaux, 2019) or their relations to other entities (Cappuzzo et al., 2020). This latter point will be the focus of Part 2.

4.2.3 *The road ahead: rethinking analytic pipelines*

MORE COMPLEX DATA-INTEGRATION PIPELINES The data-integration problem studied in Chapter 3 is very simple: it consists in analyzing the union of tables across sources. In relational algebra terms, the machine-learning models replace a `GroupBy` followed by aggregations. However, data integration often calls for joins across tables of different nature. Tackling these operations using machine learning on embeddings will require exploring new tools, for instance adapting similarity joins to merge information across tables (Silva et al., 2010; Yu et al., 2016), logic inferences on top of entity embeddings (Qu and Tang, 2019), or graph CNNs for relational data (Choi et al., 2019). We will explore the problem of joining information across data sources in Part 2.

BACK TO THE DATA SCIENTIST: OPENING UP BLACK BOXES Without explicitly merging variants into a small number of human-recognizable entities, data-analysis pipelines can be complicated to audit for the human analyst. And yet, such human inspection of pipelines is often important for validation and debugging. Understanding analytic pipelines based on machine learning rather than cleaning will need techniques from the growing field of black-box model explanation in AI (Molnar, 2020): counterfactual reasoning can be applied to understand how data-assembly pipeline transforms an input (Ghazimatin et al., 2020); permutation importance can gauge how a given attribute impacts the results by shuffling its values across instances (Altmann et al., 2010); finally, entity embeddings can be crafted to relate to human-comprehensible notions, for instance revealing latent categories (Cerda and Varoquaux, 2019).

4.2.4 *Cleaning or learning? Two complementary tools*

Replacing explicit cleaning by machine learning follows the trend from "schema on write" to "schema on read": it displaces the burden from the data producer to the data consumer (Terrizzano et al., 2015).

Cleaning is difficult, but it comes with the hope that the efforts will yield long-lasting benefits, useful for multiple usages of the data. These hopes are certainly well-grounded. Yet cleaning never ends; ambiguities in entity matching must be revisited given a new topic of analysis, or a new data source to integrate (Chessell et al., 2014). On the other hand, while variations may capture nuances –*vinaigrette* being posh for *salad dressing*–, expressing the exact same entity in two different ways is often an unnecessary hurdle to data integration. Standard vocabularies, as the universal resource identifier (URI) developed for *linked data* (Bizer et al., 2011), address these hurdles. They are complementary to a strategy based on embedding and learning, and can be priceless to bridge data sources, even if only a fraction of the entities can be expressed within the vocabulary. An analysis using machine learning to tackle ambiguities will be more successful if there are only few of these ambiguities. If data is normalized *enough*, data integration can leverage off-the-shelf embeddings, as `fastText` used in our proof of concept. These continuous embeddings are complementary to standard vocabularies.

4.3 CONCLUSION: LEARNING CUTS HUMAN LABOR BUT KEEPS VALID RESULTS

Ambiguities often arise when analyzing data, for instance if it comes from different sources with different conventions. The analysis then faces a fundamental challenge of validity: has the data been merged right, so as not to bias the results? The correct correspondence between entities across different data representations depends on the goal of the analysis: when integrating a "CDO" –chief data officer– into a employee directory that does not know such role, it could be legitimate to convert "CDO" to "executive officer" to study salary, or "data scientist" to study expertise.

The traditional view is that data cleaning is necessary to a valid analysis: carefully establish correspondences, typically combining automated approaches with manual supervision and quality assurance. Rather, our benchmark shows that valid answers to a given analytic question can be assembled by exposing ambiguities to a machine-learning pipeline. Indeed, many questions that do not explicitly call for machine learning can be formulated using such models as flexible estimators of the underlying quantities. Our empirical comparison of a simple machine-learning approach to a labor-intensive manual cleaning shows that learning improved the quality of the analysis as much, if not more, than the cleaning. We hope that it can provide a point of reference to future analysts, and justify saving time on manual cleaning.

Part II

ENRICHING DATA ANALYSES WITH BACKGROUND INFORMATION

Summary

For many machine-learning tasks, augmenting the data table at hand with features built from external sources is key to improving performance. For instance, estimating housing prices benefits from background information on the location, such as the population density or the average income. However, this information must often be assembled across many tables, requiring time and expertise from the data scientist.

In this part (based on our study from Cvetkov-Iliev et al., 2022b), we aim to facilitate such feature engineering to enable information joining over relational data. We first introduce in Chapter 5 the problematic under study, and describe in detail prior work on extracting features from relational data. Importantly, existing methods for automatic feature engineering are combinatorial and thus do not scale to large datasets. To overcome these limitations, we adopt in Chapter 6 a very different approach: replacing human-crafted features by vectorial representations of entities (*e.g.* cities) that capture the corresponding information. For this, we represent the relational data on the entities as a graph and adapt graph-embedding methods to create feature vectors for each entity. In particular, we show that two technical ingredients are crucial: modeling well the different relationships between entities, and capturing numerical attributes.

In Chapter 7, we thoroughly evaluate approaches to enrich features with background information on 7 prediction tasks. We show that a good embedding model coupled with KEN can perform better than manually handcrafted features, while requiring much less human effort. It is also competitive with combinatorial feature engineering methods, but much more scalable. Our approach can be applied to huge databases, creating general-purpose feature vectors reusable in various downstream tasks.

5

BACKGROUND: EXTRACTING FEATURES FROM RELATIONAL DATA

5.1 INTRODUCTION

We introduce in this section the subject of this part, which is based on the study we conducted in Cvetkov-Iliev et al., 2022b. We first present the problem we address here: feature engineering, although useful to enrich analyses with background information is difficult due to the irregular nature of relational data. Furthermore, existing methods to automate this process are combinatorial in nature and thus do not scale to large databases. As an alternative, we propose here to leverage embedding models to learn low-dimensional (and hence scalable) representations for entities directly from relational data. Importantly, as most embedding models are designed for discrete entities, we extend them to numerical attributes, which are often useful in downstream analyses.

5.1.1 *Joining new information to data analyses requires tedious feature engineering*

For machine learning on data tables, a data scientist may encounter columns with many different discrete entries or entities, for instance cities in a housing price prediction setting (Figure 5.1a). These city names can be encoded as a categorical variable, but generalizing to housing in a new city is then impossible. A good solution for such columns is often to **use external sources to bring in information**: the GPS coordinates of the cities, the population, the average income (Figure 5.1b)... From a data-science perspective, this requires **feature engineering** on relational data: merging and aggregating information across data sources to create an enriched table with extra features (Figure 5.1c). In practice however, such feature engineering is difficult and time consuming for the human analyst, because it requires a good understanding of both the different data sources and the application domain. For instance the number of wealthy people living in a city may be important, but estimating it may require crossing information across many tables to build a single somewhat abstract indicator. In fact, it is often recognized that data preparation is one of the biggest bottlenecks of data-science (CrowdFlower, 2016; Lam et al., 2021).

5.1.2 *A fundamental challenge: the irregular nature of data*

A specificity of learning across a complex relational structure is that **different entries come with very different information**. For instance,

when collecting information on local wealth in Wikipedia –querying DBpedia (Lehmann et al., 2015) or YAGO (Mahdisoltani et al., 2013)–, a data scientist will find for *San Francisco* the GDP as well as many known individuals and companies. But for the neighboring locality *Muir Beach*, none of this is available. The data scientist may then need to dig information at the county level, which has a different set of attributes. The root of the challenge is that the original relational information is fundamentally **irregular** and cannot be represented to a learning algorithm as a fixed set of "features".

5.1.3 *Current automatic feature engineering methods do not scale*

Our goal here is to make it very easy for the data scientist to enrich a feature with information from external data sources. Inspired by word embeddings (Mikolov et al., 2013b) which brought a breakthrough to text processing by their ease of use, **we strive to associate entities to general-purpose feature vectors that can be used in multiple downstream tasks**. This requires a feature extraction method that captures well entity attributes, and is **scalable** enough to be used on large databases. For instance, a general-purpose knowledge-base such as YAGO3 (Mahdisoltani et al., 2013) is a particularly use-

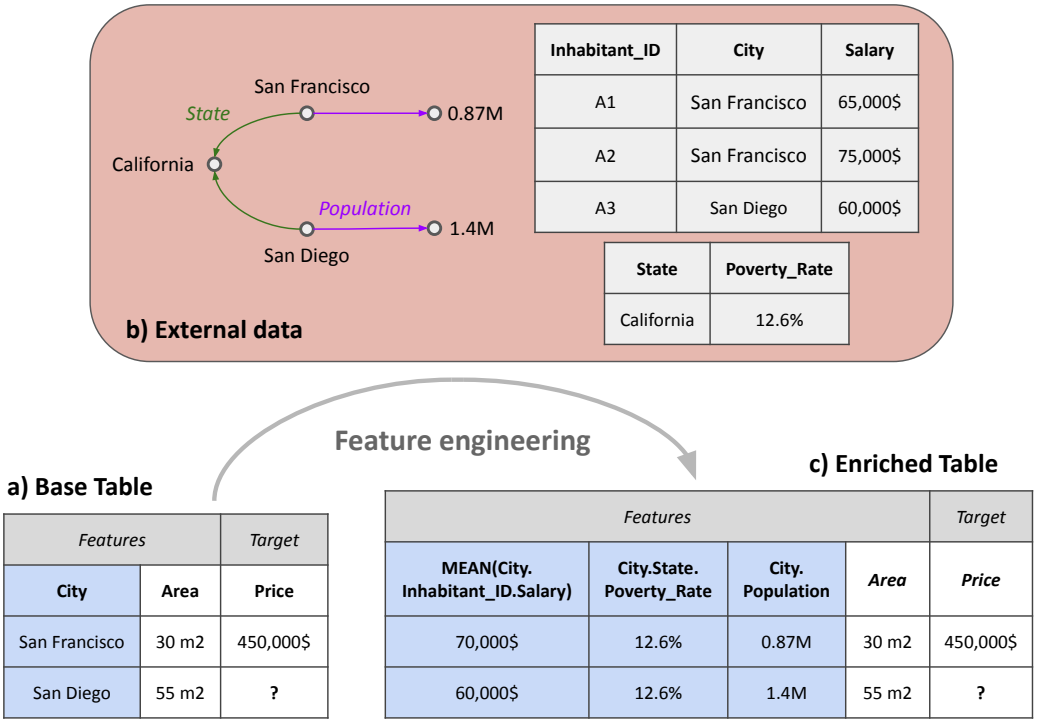

Figure 5.1 – **The classical pipeline of feature enrichment.** A base table **(a)** contains a target to predict and several features, including a categorical feature with discrete entities (here cities). To boost prediction performance, external data **(b)** about the entities of interest is incorporated into the base table –usually via tedious feature engineering– to obtain the enriched table **(c)**. The external data **(b)** can come under various formats, *e.g.* tables or multi-relational graphs.

ful source of data, with information on 75,000 cities; but it is huge: millions of entities and hundreds of attributes. Existing automatic feature engineering methods, such as Deep Feature Synthesis (DFS) (Kanter and Veeramachaneni, 2015), are **combinatorial**: they greedily join and aggregate entity attributes across tables to create feature vectors. Their combinatorial nature leads to tractability challenges: running DFS on YAGO3 produces very high dimensional vectors ($d \sim$ 10,000 – 140,000, see Table 7.2) which entail large storage costs and computational hurdles in downstream machine-learning tasks.

5.1.4 *Our proposal: embedding models that capture numbers*

Instead, we propose to use embedding models that **learn a static vector representation for each entity**. Indeed, they provide compact representations that can encode knowledge about various entities into a fixed, **low-dimensional** space (*e.g.* $d = 200$). We learn these vectors from the external data, and add them to the base table as new features to enhance prediction performance. A pioneering work in this direction is RDF2vec (Ristoski and Paulheim, 2016a) and its variants, which have been used to learn entity embeddings from multi-relational graphs for various downstream tasks (Egami et al., 2021; Ristoski et al., 2019; Saeed and Prasanna, 2018; Sousa et al., 2020). These works directly build on word-embedding tools developed for natural language –namely word2vec (Mikolov et al., 2013b). As such, they leverage *contextual* information: as *San Francisco* and *California* are connected in the graph they are related. However, they do not account for the nature of these relations, which requires modeling the *relational* information: Wikipedia specifies that *San Francisco* is in *California*, but *Sacramento* is the capital of *California*. We will see that capturing well this information is important to generate feature vectors for downstream analytic applications. Another, more general, drawback of embedding methods is that they are designed for **discrete** entities, and are not suited to capture numerical attributes. Yet these attributes are often useful for the end task. For instance, densely populated cities tend to exhibit high housing prices.

We propose here an approach that addresses these two limitations and provide high-performance embeddings. To capture relational information, we rely on **knowledge graph embedding models** (Q. Wang et al., 2017), widely used for graph completion but not studied for feature extraction purposes. In such models, embeddings are directly optimized to capture relationships between entities. We then introduce **KEN** (Knowledge Embedding with Numbers), a module that extends knowledge graph embedding models to **numerical attributes**. Finally, we conduct a thorough empirical evaluation of our approach, using entity embeddings to boost machine-learning performance in multiple tasks, and show that:

— Feature vectors obtained via knowledge graph embedding models perform much better than RDF2vec embeddings.

— Embeddings learned with KEN do capture numerical information, which greatly improves prediction performance in downstream tasks.

— A good embedding model coupled with KEN outperforms manually handcrafted features, while requiring much less human effort. It is also competitive with Deep Feature Synthesis, but is more scalable in terms of computation time, memory usage and size of the created features.

— Although designed for multi-relational graphs, simple heuristics allow our approach to be applied to tabular data, with good performance.

5.1.5 *Chapters outline*

The rest of this part follows as such: Section 5.2 goes into depth explaining related work on extracting features from relational data, Chapter 6 details our contributed approach, and Chapter 7 gives a thorough empirical study of approaches to create features from relational data.

5.2 RELATED WORK: EXTRACTING FEATURES FROM RELATIONAL DATA

We focus here on two common data structures for data-science: tabular data, as in relational databases, and multi-relational graphs (a.k.a. knowledge graphs), the backbone of Linked Open Data (Bauer and Kaltenböck, 2011). We broadly refer to both as *relational data*. In this section we give an overview of various lines of work related to creating vectors from relational data, drawing from a variety of scientific communities.

5.2.1 *The classic view: feature engineering*

MANUAL FEATURE ENGINEERING Feature engineering across multiple tables traditionally relies on a human analyst crafting SQL queries or dataframe operations, such as joins or aggregations, to build a single feature matrix. The problem is the same with Linked Open Data (Paulheim, 2013; Ristoski and Paulheim, 2016b): statistical studies require features extracted from the data, here coming as knowledge graphs rather than multiple tables. *Propositionalization* approaches (Kramer et al., 2001) tackle this by creating for each entity (node) of the graph a set of features, using statistical fingerprints and aggregates of its neighbourhood (Paulheim and Fümkranz, 2012; Ristoski and Paulheim, 2014). Here again, manual crafting is needed to capture specific information such as wealth.

Whether it is done on tables or knowledge graphs, feature engineering is a time-consuming task: studies show that data scientists spend **60%** or more of their time transforming the data for analysis (CrowdFlower, 2016). Indeed, designing the right features often re-

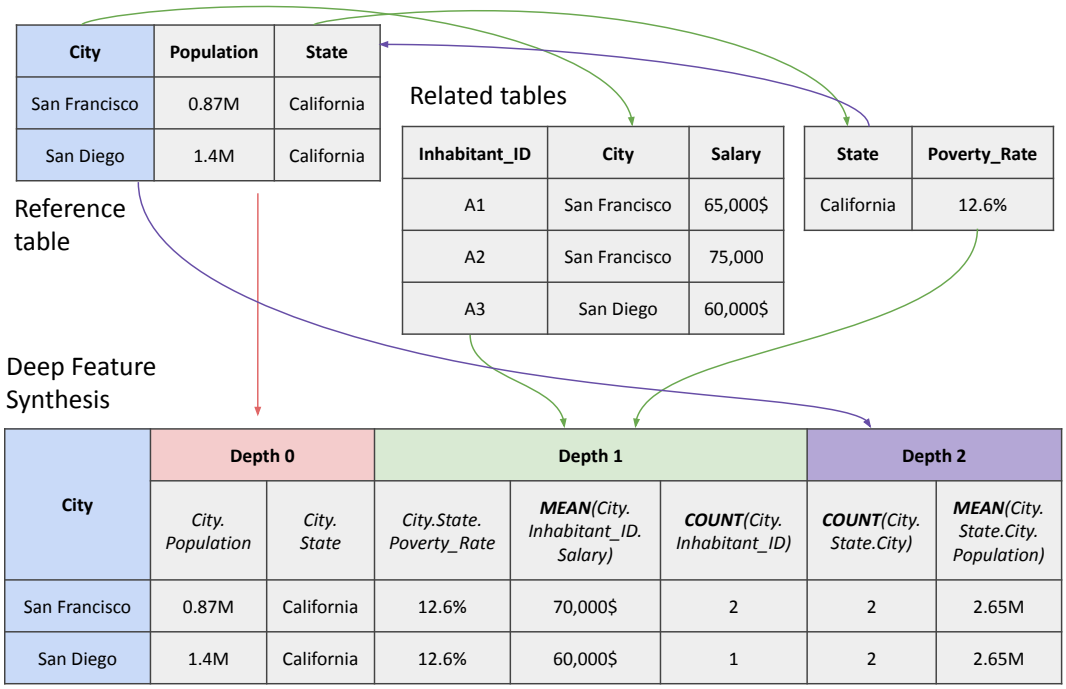

Figure 5.2 – **An example of Deep Feature Synthesis.** Starting from a reference table with entities of interest (here cities), new features are created by chaining joins to related tables, up to a certain depth = 2. To aggregate values from one-to-many relations (*e.g.* city inhabitants), we use the `MEAN` and `COUNT` operators, respectively for numerical and categorical features. Colored arrows indicate join paths across tables for each depth.

quires careful effort from the analyst: which information is relevant for the task at hand? How to query it? This is particularly difficult on large data sources. For instance, a knowledge graph representation of Wikipedia leads to hundreds of entity classes described by thousands of attributes in DBPedia (Lehmann et al., 2015). Exploring which joins are best for a given analysis is difficult even for an expert: how to assemble indirect signals that capture information on the question at hand, for instance estimating the distribution of wealth in a locality.

AUTOMATED FEATURE ENGINEERING A few approaches have been proposed to automate the construction of queries for feature engineering on relational databases. A fundamental challenge is that assembling such multi-table data transformations calls for discrete choices –*e.g.* to join, or not to join? (Kumar et al., 2016)– with combinatorial possibilities that explode on large databases. For instance, Deep Feature Synthesis (DFS) (Kanter and Veeramachaneni, 2015) is a greedy approach that denormalizes a database by chaining joins from one reference table to all related tables and aggregating one-to-many relations using combinations of a small base of functions (see Figure 5.2). Typical aggregation functions include `COUNT`, `MODE` (most common) for categorical features, and `MEAN`, `MIN`, `MAX`, `STD` for numerical features. A crucial parameter of DFS is the *depth*, which limits how many times joins can be chained to create new features. Higher depths capture

a wider range of information and usually improve performance, but quickly result in very large feature vectors and computation times, as the number of possible join paths grows exponentially. This often calls for post-processing techniques to remove unpredictive or redundant features.

Subsequent works have improved over DFS by adding aggregation functions for other types of data (text, sequences) (Lam et al., 2017), for instance via recurrent neural networks (Lam et al., 2019). Although powerful feature extractors, all these methods remain combinatorial in nature, and do not scale to large databases. Even with a limited depth, a large number of entities of different types leads to increasingly wide feature matrices with many missing values, as the different entities come with different sets of attributes. Finally, automated feature engineering methods present other drawbacks: the created features often contain categorical or missing values that must be encoded, and their interpretability (we can trace back the joins and aggregations needed to compute each feature) is challenged as their dimension quickly grows.

5.2.2 *Entity embeddings in relational data*

While entity embeddings come from a body of literature far from that of feature engineering, they also create feature vectors from relational data (Lavrac et al., 2020).

5.2.2.1 *Prelude: word embeddings*

Many embedding methods for relational data take inspiration from word embeddings. By injecting discrete entities (words) in vector spaces, word embeddings have boosted statistical analyses of text. They rely on the distributional semantics idea, which can be summarized by Firth's sentence: "a word is characterized by the company it keeps". A central model is Skip-Gram with Negative Sampling (SGNS), used in word2vec (Mikolov et al., 2013b). Each word w is associated to an embedding $\boldsymbol{w} \in \mathbb{R}^p$ [1]. SGNS learns these embeddings by optimizing similarities of pairs of words, using a *scoring function*:

Scoring function
$$f(w, w') = \boldsymbol{w} \cdot \boldsymbol{w'} \tag{5.1}$$

Given a text corpus, embeddings are optimized so that a word w is more similar to a word w' observed in the same context –*e.g.* the

1. To be precise, two embeddings $\boldsymbol{w_t}, \boldsymbol{w_c}$ are learned for each word. Which one is used in the scoring function depends if we view it as the context word or not. For instance if $w' \in \text{context}(w)$, then we maximize $f(w, w') = \boldsymbol{w_t} \cdot \boldsymbol{w'_c}$, and vice-versa.

same sentence–, than another word $w^\dagger$ not in the context; minimizing a cross-entropy loss [2]:

$$\text{SGNS} \quad L = - \sum_{\substack{w,\, w' \in \text{context}(w),\\ w^\dagger \notin \text{context}(w)}} \log(\sigma(f(w, w'))) + \log(1 - \sigma(f(w, w^\dagger))) \tag{5.2}$$

After training, word embeddings capture *contextual* similarities: words with the similar contexts (neighbors) end up close in the embedding space.

5.2.2.2 *Embedding entities in tables*

Word embedding methods, such as SGNS, can be extended to other data structures by defining a corresponding notion of context (Grohe, 2020). In tables, a common choice is to view rows as sentences: two entities are in one another's context if they appear in the same row. This was for instance applied to enable semantic queries over tables (Bordawekar and Shmueli, 2017) and for automatic table completion and retrieval (Zhang et al., 2019). More recent work integrates intra-row and intra-column information to learn richer representations. Cappuzzo et al., 2020 link entries of a table to the row and column nodes they belong to. Random walks through the resulting graph generate "sentences" of tokens, then fed to a SGNS model.

5.2.2.3 *Embedding entities in knowledge graphs*

Knowledge graphs use a more general representation of relational data than tables. They replace the notion of columns by that of *relations*, which enables a uniform representation over many tables, and helps assembling information from multiple sources of data. Each piece of information is encoded as a triple (h, r, t), indicating a certain *relation* r between the *head* and *tail* entities (h, t). Large knowledge graphs, such as YAGO3 (Mahdisoltani et al., 2013) or DBPedia (Lehmann et al., 2015) contain millions or even billions of triples – *e.g.* (San Francisco, HasState, California) – and cover millions of entities.

Knowledge graph embedding models typically learn a vector for each entity (node) and relation (edge) of the graph. They have been mostly developed for two purposes, leading to two distinct lines of research (Portisch et al., 2022):

1) **Predicting new triples** of the knowledge graph for completion purposes, which has been the main application of knowledge graph embeddings.
2) **Providing feature vectors for downstream tasks** outside the knowledge graph, which received much less attention in the literature, but is our focus here.

2. This is actually a simplified version of the loss optimized by word2vec, which does not account for multiple negative examples.

EMBEDDINGS FOR DOWNSTREAM TASKS RDF2vec (Ristoski and Paulheim, 2016a) is a central work applying knowledge graph embeddings in external downstream tasks. It has been used to incorporate background information in various tasks: geospatial data analysis (Egami et al., 2021), recommender systems (Ristoski et al., 2019; Saeed and Prasanna, 2018), or biomedical prediction tasks (Sousa et al., 2020). Given a knowledge graph, RDF2vec generates sequences of tokens by performing random walks on the graph, alternating between entities and relations (see Figure 5.3). These sequences are then fed to a SGNS model to obtain embeddings for entities and relations. An important parameter is the *depth*, which limits the number of hops in the random walk, and thus the range of information to capture. Assuming that the window size of the SGNS model is set accordingly, a depth of 1 captures relationships between entities and their nearest neighbors in the graph, and so on... Similarly to Deep Feature Synthesis, a challenge is that the number of possible walks increases exponentially with depth. To avoid this, walks are often computed for certain entities of interest only, with a limited number of walks for each entity.

Since RDF2vec, most research efforts focused on the creation of walks, for instance giving more weight to relations/entities based on their frequency, PageRank or degree, removing rare entities, or allowing teleportations between entities that share similar properties (Cochez et al., 2017; Vandewiele et al., 2020b).

EMBEDDINGS FOR GRAPH COMPLETION Knowledge graph embeddings have been widely used for graph completion, either through *link prediction* (predicting the missing entity in an incomplete triple $(h, r, ?)$) or *triple classification* (predicting if a triple is True of False). Similarly to SGNS, these models define a scoring function $f(h, r, t)$

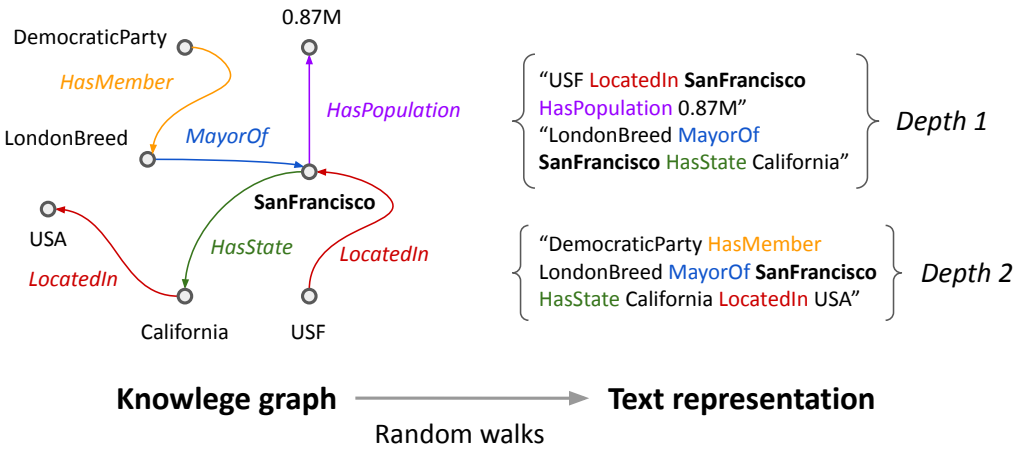

Figure 5.3 – **Graph to text representation in RDF2vec.** Random walks are performed on the knowledge graph to generate sentences of tokens. Often, walks are only computed for a subset of entities, here San Francisco. The depth parameter limits the number of hops in the random walk, either forward or backward.

that represent the plausibility of a given triple (h, r, t). Embeddings are then optimized so that observed triples obtain high scores, while negative ones (typically sampled by corrupting the head or tail entity in observed triples) obtain low scores.

Scoring functions typically model the different relations between entities as geometrical operations in the embedding space. For instance, the seminal TransE model (Bordes et al., 2013) represents a relation r as a translation vector $\boldsymbol{r} \in \mathbb{R}^p$ between entity embeddings $\boldsymbol{h}$ and $\boldsymbol{t}$:

$$\text{TransE} \qquad f(h, r, t) = -\|\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}\| \tag{5.3}$$

with $\|.\|$ a ℓ_1 or ℓ_2 norm. Given a knowledge graph $\mathcal{G}$, embeddings are trained to minimize a margin loss:

$$L = \sum_{\substack{(h,r,t)\in\mathcal{G},\\ (h',t') \text{ s.t.} (h',r,t')\notin\mathcal{G}\\ \text{with } h'=h \text{ or } t=t'}} [f(h', r, t') - f(h, r, t) + \gamma]_+ \tag{5.4}$$

Many models that improve upon TransE (Q. Wang et al., 2017) focus on better modeling of one-to-many relationships and certain relational patterns (*e.g.* symmetry/antisymmetry, inversion, composition) (Balazevic et al., 2019; Sun et al., 2019; Yang et al., 2015). For link prediction in knowledge bases, one of the best performing methods (Ali et al., 2020) is MuRE, Multi-Relational Poincare graph embeddings (Balazevic et al., 2019). The key component of the method is the model of the link between head and tail entity (homologous to (5.3) for TransE):

$$\text{MuRE} \qquad f(h, r, t) = -d(\boldsymbol{\rho}_r \odot \boldsymbol{h}, \boldsymbol{t} + \boldsymbol{r}_r)^2 + b_h + b_t \tag{5.5}$$

where $\odot$ is the element-wise multiplication, two vectors $\boldsymbol{\rho}_r, \boldsymbol{r}_r \in \mathbb{R}^p$ represent the relation r, and the head and tail entities are represented by vectors $\boldsymbol{h}, \boldsymbol{t} \in \mathbb{R}^p$ and biases $b_h, b_t \in \mathbb{R}$. d is the Euclidean distance[3]. The model is optimized by sampling positive and negative triples (as in (5.4), but using a logistic loss (5.2) instead).

STRUCTURE OF CONTEXTUAL VS RELATIONAL EMBEDDINGS Approaches based on SGNS such as RDF2vec only capture ***contextual* information**, while much progress in knowledge graph embedding has focused on modeling different types of relations separately. As a consequence they induce very different neighborhood structures on entity embeddings.

Contextual embeddings, as RDF2vec, are trained on "sentences" of tokens, where each entity is surrounded by the relations and entities it co-occurs with in triples (Figure 5.3). Two entities end up close in the embedding space if they have similar contexts: *1)* They may share a relation, but not necessarily with the same entity, *e.g.* (San Francisco, **LocatedIn**, California) and (Paris, **LocatedIn**, France). This tend to

3. MuRE can also use the Poincaré non-Euclidean geometry. However in practice (Balazevic et al., 2019) the Euclidean version is an excellent performer, as good as the non-Euclidean one for $p \geqslant 150$.

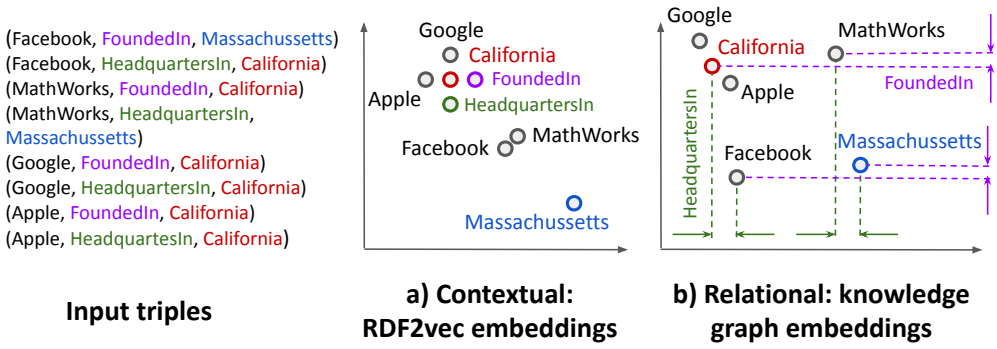

Figure 5.4 – **What drives entity neighborhoods in embedding space? a)** Contextual embeddings (as RDF2vec) ignore the nature of the relation: given information on states in which companies have been founded and have their headquarters, it cannot differentiate *Facebook* (born in Massachussetts, moved to California), from *MathWorks* (born in California, moved to Massachussetts). **b)** Knowledge graph embeddings models can give rise to different geometric constraints for these two relations, separating out the companies. For instance here a relation is encoded with a projection.

group entities of the same type, since entities of different nature, like people and cities, share few relations. *2)* They may share a connection to a common entity, but not necessarily via the same relation, *e.g.* (MathWorks, FoundedIn, **California**) and (Nevada, HasBorderWith, **California**). Figure 5.4a gives a paradigmatic example: such contextual information is blind to the difference between Facebook, founded in Massachussetts but headquartered in California, and MathWorks, founded in California but headquartered in Massachussetts.

Knowledge graph embeddings that use the relation type in the scoring function between two entities create a very different structure in the embedding space. As relations of different nature lead to different transformations of the embedding space, they each "pull" entities in different directions. In addition, modern models can learn transformations that are not one-to-one (*i.e.*non bijective), better suited to many-to-one relations, as when many cities are located in the same state. As a result the different relations can be encoded separately in entity embeddings, for instance along different coordinates (Figure 5.4b).

INTEGRATING NUMERICAL ATTRIBUTES IN EMBEDDINGS Numerical attributes, such as city populations, are poorly handled by most embedding methods. They are often simply dismissed, or at best binned and treated as discrete entities (Cappuzzo et al., 2020), which remains suboptimal as it does not capture the topology of numbers.

Recent knowledge graph embedding models address this issue (Gesese et al., 2021). TransEA (Wu and Z. Wang, 2018) adds a loss to reconstruct numerical values from embeddings with a linear model. LiteralE (Kristiadi et al., 2019) is a state-of-the-art approach where each entity i is represented by two vectors: $e_i \in \mathbb{R}^p$ representing the en-

tity itself, and $\boldsymbol{l}_i \in \mathbb{R}^q$, $\boldsymbol{l}_i$ containing each of its numerical attribute (0 if no value, and where q is the number of numerical relations in the KG). When used in the scoring function, embeddings $\boldsymbol{h}$ and $\boldsymbol{t}$ are constructed with a function g that combines the two vectors into a single one: $\boldsymbol{h} = g(\boldsymbol{e}_h, \boldsymbol{l}_h)$, and $\boldsymbol{t} = g(\boldsymbol{e}_t, \boldsymbol{l}_t)$, both in $\mathbb{R}^p$. LiteralE implements g as a learnable mechanism similar to gated recurrent units:

LiteralE
$$g(\boldsymbol{e}, \boldsymbol{l}) = \boldsymbol{z} \odot \boldsymbol{w} + (1 - \boldsymbol{z}) \odot \boldsymbol{e} \tag{5.6}$$

$$\text{with} \qquad \boldsymbol{z} = \sigma\left(\boldsymbol{W}_{ze}^T \boldsymbol{e} + \boldsymbol{W}_{zl}^T \boldsymbol{l} + \boldsymbol{b}\right), \qquad \boldsymbol{w} = \tanh\left(\boldsymbol{W}_h^T [\boldsymbol{e}, \boldsymbol{l}]\right),$$

and $\odot$ is the pointwise multiplication, $\boldsymbol{W}_h^T \in \mathbb{R}^{(p+q)\times p}$, $\boldsymbol{W}_{ze}^T \in \mathbb{R}^{p\times p}$, $\boldsymbol{W}_{zl}^T \in \mathbb{R}^{q\times p}$ and $\boldsymbol{b} \in \mathbb{R}^p$.

6

MULTI-RELATIONAL EMBEDDINGS THAT CAPTURE NUMBERS

We introduce in this chapter our approach to automatically extract information from relational data, creating feature vectors that can be used in downstream tasks. It relies on 3 key ingredients, that we describe in the following sections:

1) Using knowledge graph embedding models designed for graph completion, as opposed to RDF2vec, to capture well ***relational*** information.
2) KEN (Knowledge Embedding with Numbers), a module that extends knowledge graph embedding models to **numerical attributes**.
3) Representing tables as knowledge graphs, to leverage them in our approach.

Figure 6.1 summarizes our pipeline for automatic feature extraction from relational data.

6.1 RELATIONAL RATHER THAN CONTEXTUAL EMBEDDINGS TO ENCODE INFORMATION

With our goal of creating embeddings as features for downstream tasks, we motivate here the importance of using *relational* embeddings, originally designed for knowledge graph completion, rather than *contextual* RDF2vec-like models, traditionally used to extract features for downstream tasks.

From a big picture perspective, given an entity h of interest (*e.g.* a city), we would like an embedding $\boldsymbol{h}$ that encodes as well as possible the information related to h in the data. At the very least, it implies representing well the various relationships h has to other entities (*e.g.* its state), to make them available to the machine-learning model used in the downstream task. Representing not only the related entity t but also the nature of the relation r is often important: knowing whether a person A is the mother, the sister, or the daughter of a person B informs on the age difference.

In contextual embeddings such as RDF2vec, **the presence of a link between a entity h to another entity t is modeled somewhat independently from the nature r of the link**, *i.e.* the type of the relation. Indeed, the scoring function used in SNGS (Equation 5.1) is only applied to pairs (h, t), (h, r) and (r, t). Structure between h, r, and t is created indirectly as they appear in the same context.

In contrast, relational embeddings developed for knowledge graph completion use a scoring function involving h, r, and t **jointly**. As this scoring function is minimized for triples in the graph, it induces algebraic relations between the corresponding embeddings: for TransE

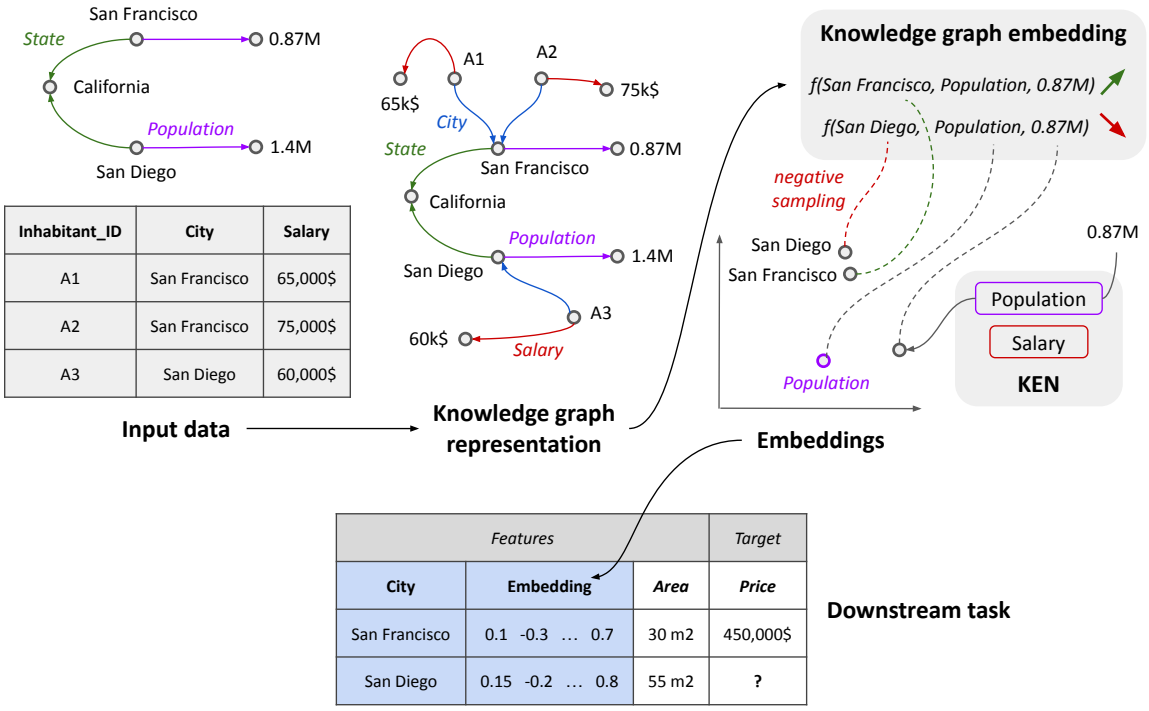

Figure 6.1 – **Our pipeline for automatic feature extraction from relational data.** 1) The input data, which may contain tables, is transformed into a knowledge graph. 2) We use a knowledge graph embedding model to learn a vector for each entity, and leverage numerical values by embedding them in the same space as other entities with KEN. 3) After training, entity embeddings can be easily added as new features in downstream tasks.

$\boldsymbol{t} \approx \boldsymbol{h} + \boldsymbol{r}$, or for MuRE $\boldsymbol{t} \approx \boldsymbol{\rho}_r \odot \boldsymbol{h} - \boldsymbol{r}_r$. These algebraic relations imply that $\boldsymbol{t}$ captures the link to $\boldsymbol{h}$ in a way that is specific to r and hence a downstream analysis model can recover this specific information, *e.g.* selecting on the mother, and not all relatives.

Figure 5.4 illustrates the specificity of the link: for RDF2vec, relations are encoded as vectors which lie in the middle of entity embeddings, while knowledge graph embedding models encode relations as transformations on entity embeddings (here, a projection). This allows the latter to express the different relations on different vector coordinates.

6.2 CAPTURING NUMERICAL ATTRIBUTES WITH KEN

6.2.1 *Numerical attributes are valuable to data analyses, but hard to capture in feature vectors*

Numerical attributes are omnipresent in relational data, and often contain precious information for downstream tasks, *e.g.* a city's wealth influences housing prices. While they are readily-available as numbers, the irregular nature of the information prevents from merely adding them as coordinates to the feature vectors. A first challenge is that different entities have different numerical attributes. A more serious one arises when aggregating numerical information

across many-to-one relations: there are many ways of doing so. For instance, to characterize wealth in a county from the GDP of its cities, the mean, the Gini index, the percentiles, *etc.* are all useful aggregates. As a result, Deep Feature Synthesis generates more than **2,000** features derived from numerical attributes for cities in YAGO3.

We aim for lower-dimensional representations, and thus strive to capture numerical information in entity embeddings. However, embedding methods are formulated in terms of discrete elements (Section 5.2.2): words, entities. A naive way to adapt them to numerical attributes would be to consider numbers as tokens and learn an independent embedding for each value. Yet doing so discards the topology underlying those numbers: close numerical values should have similar representations. Binning values before embedding reduces this effect, but remains suboptimal. To tackle this, we introduce here KEN (Knowledge Embedding with Numbers), a module that adapts embedding models to numerical attributes.

6.2.2 *The KEN module*

Entity-embedding approaches can be seen as relying on a linear encoder to associate an entity h with its vector representation $\boldsymbol{h} \in \mathbb{R}^p$. In this light, we propose to also **inject numerical values in the same vector space with an encoder**, learning a function $\boldsymbol{e} : \mathbb{R} \to \mathbb{R}^p$ that **maps numerical values to embeddings**.

We use as function a single-layer neural network with a ReLU activation to embed numerical values. To embed different types of attribute separately (e.g. city populations and GPS coordinates), we learn a function $\boldsymbol{e}_r$ for each attribute r:

$$\boldsymbol{e}_r(x) = \mathrm{ReLU}(x\,\boldsymbol{w}_r + \boldsymbol{b}_r) \tag{6.1}$$

with $x \in \mathbb{R}$ the numerical value to embed, and $\boldsymbol{w}_r, \boldsymbol{b}_r \in \mathbb{R}^p$ the weights and biases of the linear layer. Embeddings $\boldsymbol{e}_r(x)$ of numerical values can then be **used in place of tail embeddings $\boldsymbol{t}$** in the scoring function $f(h, r, t)$.

6.2.3 *Comparison with other methods capturing numerical attributes*

An asset of KEN is that it comes with **no hyper-parameters** to tune. This is unlike TransEA (Wu and Z. Wang, 2018), where the importance of numerical attributes must be controlled, with the danger that the optimal value might differ for each attribute. Another important difference with TransEA is that KEN can capture **non-linear** interactions between entities and numerical attributes, thanks to the ReLU activation. For instance, cities in California are associated to latitudes between 32° N and 41° N which cannot be expressed by a mere threshold on a linear representation.

Importantly, KEN uses numerical values x during training as new triples (h, r, x) whose scores must be maximized, which forces entity embeddings to capture these numerical attributes. This is different

from LiteralE (Kristiadi et al., 2019), where numerical values are incorporated to entity embeddings to better predict non-numerical triples (h, r, t). LiteralE therefore only captures the information in numerical values useful to triangulate other entities, and not the values in themselves. In particular non discriminant numerical attributes can be discarded by the gate mechanism. As an extreme example, an entity linked to numerical attributes but not to other entities will not be embedded in LiteralE, as there is no training data.

In contrast, KEN draws no major distinction between discrete entities and numerical values: they are embedded in the same space. Each type of numerical attribute is associated to a specific relation and thus embedded on a specific line segment via Equation 6.1. An analytic model for a downstream task can extract this information, proceeding in a similar way as with discrete information (Section 6.1). The numerical attributes that an entity has and its relations to other entities may contribute to create similar neighborhood structures: a city being *locatedIn* California is equivalent to its GPS coordinate taking specific value ranges.

6.2.4 *Making the architecture robust to attribute distribution*

One challenge of heterogeneous data is that **different numerical attributes have very different distributions**. We thus normalize numerical values $x \in \mathbb{R}$ to the interval $[0, 1]$ before embedding them. With neural networks, a common way to do so is "min-max" normalization: $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$. However it is problematic when dealing with heavy-tailed distributions, such as city populations. Indeed, after normalization, most values x' will be very close to zero and have similar representations $\boldsymbol{e}_r(x') \simeq \text{ReLU}(\boldsymbol{b}_r)$. This makes it difficult for instance to distinguish a village with 1 000 inhabitants from a medium-sized town of 10 000 people.

Ideally, we would like the values x' to be **evenly distributed** in $[0, 1]$, to separate as well as possible their embeddings. We achieve this with quantile normalization, which maps numerical values to their quantile in the attribute distribution, using an empirical estimate of the cumulative distribution function: $x' = \text{CDF}(x)$.

Figure 6.2 summarizes the complete picture of numerical value embedding with KEN.

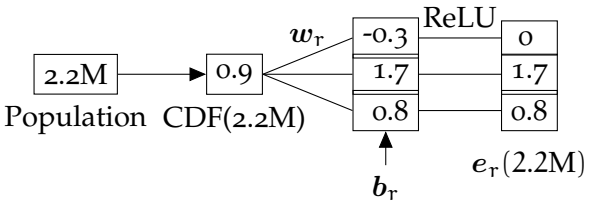

Figure 6.2 – Embedding numerical values with KEN.

6.3 REPRESENTING TABLES AS KNOWLEDGE GRAPHS

To create embeddings with rich semantics, the source data must contain as much detail as possible about the entities under study. This often requires to leverage data from different sources, for instance combining broad but shallow information (*e.g.* city populations) from large knowledge graphs with more granular data (*e.g.* recent house prices at the neighbourhood-level) from domain-specific tables. Although our approach inputs knowledge graphs, *i.e.* triples (h, r, t), **this representation is general enough to easily encode information from other data structures**. We focus here on tabular data, and explore a few strategies to represent tables as knowledge graphs.

The core idea to generate triples from tables is to link entities from the same rows with different relations. For instance, an exhaustive strategy consists in building all possible triples from the table, linking all discrete entries to other entities or numerical values from the same rows (Figure 6.3a). One asset of this method is that it produces good embeddings for all entities, as they are directly connected to their attributes in the graph. But it generates a large number of triples: $\mathcal{O}(n_{cols}^2 \, n_{rows})$, which increases the training time of embeddings. If we know beforehand the entities of interest, *i.e.* those used in the

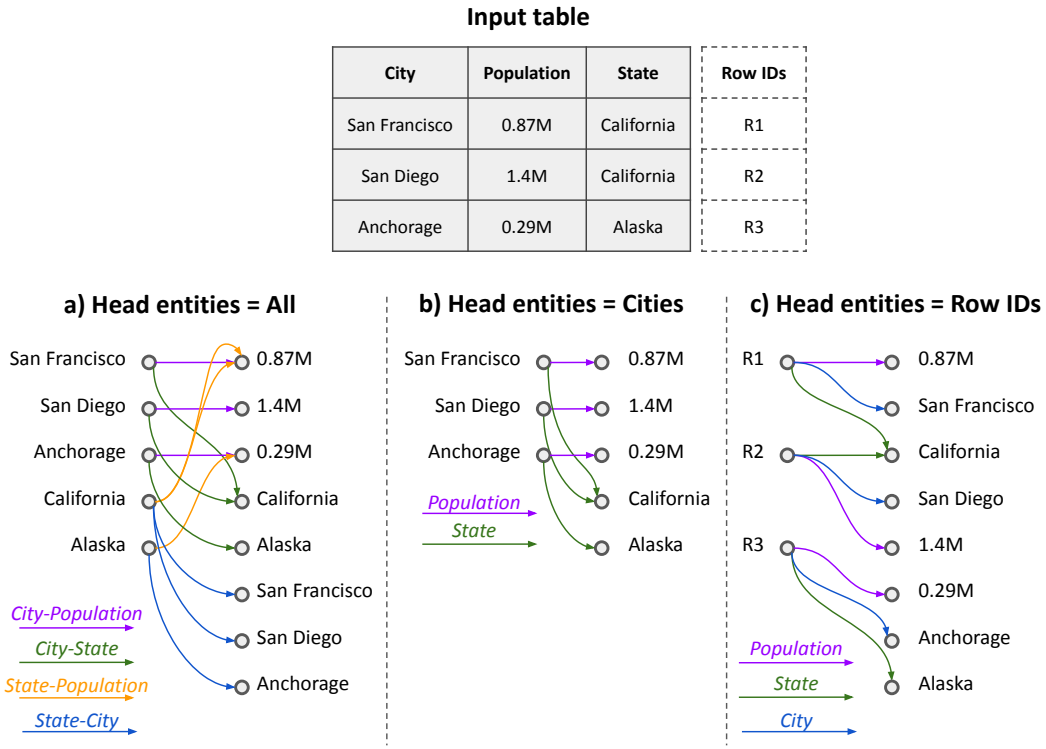

Figure 6.3 – **Representing tables with triples.** For each row of the table, we generate triples by linking its entries through different relations. The methods we present here differ on their choice of head entities when building triples: **a)** using all discrete entries as heads **b)** using only the entities of interest (generally from the same column) and **c)** introducing a "row id" entity for each row and using it as head entity.

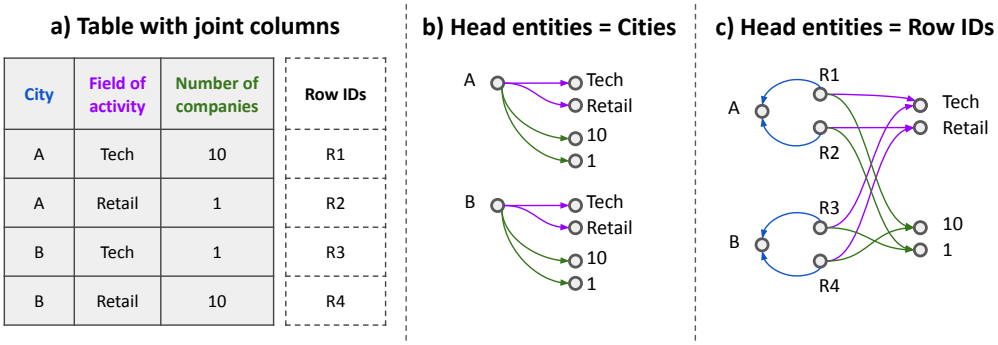

Figure 6.4 – **Capturing joint information across columns.** **a)** A table describing cities with two joint attributes that must be considered together to be meaningful. **b)** Using cities as head entities encodes the two attributes separately, hence we cannot differentiate them from their triples. **c)** Introducing row entities allows to capture all attributes jointly and distinguish the two cities.

end task (*e.g.* cities), we can instead build triples from these entities only (Figure 6.3b). This greatly reduces the number of triples to $(n_{cols} - 1)\, n_{rows}$ (these entities generally come from a single column) and returns embeddings tailored for the entities under study. However, this approach neglects other entities: they are not directly connected to the entries of the row and are thus likely to underperform in other applications. Finally, we consider a third heuristic that assigns a row id to each row of the table, treats this row id as an entity, and then links it to the various entries of the row (Figure 6.3c). This method combines benefits of the previous methods: it does not require any prior knowledge of the downstream application and generates a light graph with $n_{cols} \times n_{rows}$ triples. Yet learning an additional embedding for each row may raises scalability issues if there are much more rows than distinct entities to embed.

A desirable property of table-to-graph methods is their ability to represent **joint information across columns**. For instance Figure 6.4a considers two cities A, B with their number of companies in different fields of activity. Taken alone, the two columns are not very informative: what matters here is the number of companies in a certain field of activity, which requires to consider both columns jointly. Methods that build triples from table entries such as cities encode the attributes "field of activity" and "number of companies" independently, and thus cannot distinguish A and B from their triples (Figure 6.4b). In contrast, introducing row entities allows to capture row data jointly and differentiate the two cities (Figure 6.4c).

Finally, if missing data are present in the table, we encode them with specific entities (one for each column).

7 EMPIRICAL STUDY AND DISCUSSION

In this chapter, we investigate whether knowledge graph embedding models capturing numerical attributes can serve as efficient and scalable feature extractors. For this, we first conduct in Section 7.1 a thorough empirical evaluation of various feature engineering methods. We then discuss their performance and conclude on their potential in Section 7.2.

7.1 EMPIRICAL EVALUATION

We compare our approach with automatic feature extraction techniques, such as Deep Feature Synthesis (DFS) or RDF2vec, and focus on two criteria:

— the ***quality*** of the extracted features: how well do they improve performance in downstream tasks?

— the ***scalability*** of the approach: time and space complexity, size of the feature vectors

7.1.1 *Downstream tasks*

We evaluate our approach on 7 prediction tasks on various types of entities. In each task, we extract features for the entities of interest (*i.e.* target entities) from a *source* dataset, and add them to a *target* dataset containing the variable to predict. To showcase the versatility of our method, **we consider tables and knowledge graphs as source data**. More details about the downstream tasks and datasets are given in the appendix, Section A.2.1.

TABULAR DATA We first consider two classification tasks: **KDD14** (classification of educational crowdfunding projects) and **KDD15** (student dropout prediction in MOOCs). For these tasks the source data consists of multiple tables describing the target entities. To leverage this data in our approach, we represent it as a knowledge graph by using target entities as head entities and linking them to other entries from the same rows, similarly to Figure 6.3b.

KNOWLEDGE GRAPHS To support our claim that general-purpose embeddings can be learned from large databases and used in various end tasks, we consider a more challenging setup: **enriching several downstream tasks with background information from Wikipedia**. To that end, we leverage YAGO3, a knowledge graph representation of common knowledge, built from Wikipedia and other sources (Mahdisoltani et al., 2013).

Our version of YAGO3 contains 2.8 million entities, described by 7.2 million triples. We learn embeddings for various entities that are common in data science problems (counties, cities, people, companies, movies...) and use them in 5 regression tasks on socio-economic topics [1]:

— **Elections**: predict the number of votes per party in 3000 US counties.

— **Housing prices**: predict the average housing price in 23000 US cities.

— **Accidents**: predict the number of accidents in 8500 US cities.

— **Movie revenues**: predict the box-office revenues of 4900 movies.

— **Employees**: predict the number of employees in 3000 companies.

Note that there exists a more recent version of YAGO (Pellissier Tanon et al., 2020), with a much greater coverage of information: 64 million entities, with about 2 billion triples. However, we could not include it in our empirical study as the DFS baseline was intractable on such a large database.

7.1.2 *Approaches considered for evaluation*

We describe below the feature extraction approaches that we include in our empirical study.

OUR APPROACH We implement KEN on top of 3 embedding algorithms: TransE (Bordes et al., 2013), the seminal work that introduced relations as translations of embeddings, DistMult (Yang et al., 2015), with scoring function $f(h, r, t) = \boldsymbol{h} \cdot (\boldsymbol{r} \odot \boldsymbol{t})$, and MuRE (Balazevic et al., 2019) because it emerged as a top-performing method in link prediction (Ali et al., 2020). We learn 200-dimensional embeddings and keep all hyper-parameters constant, except for the number of epochs $\in [2, 4, 8, 16, 24, 32, 40]$ that we tune (see Section A.2.2 for the exact parameters used). We base our implementations on PyKEEN (Ali et al., 2021), a Python library for learning knowledge graph embeddings. In addition, PyKEEN implements a version of DistMult that leverages numerical values with LiteralE (Kristiadi et al., 2019), which allows for a comparison with KEN.

DEEP FEATURE SYNTHESIS We compare our embedding approach to Deep Feature Synthesis (DFS, see Figure 5.2). We use an implementation of DFS from the Python package *featuretools* and extract features at depths (0, 1, 2, 3) with the default aggregation functions: `MEAN`, `MIN`, `MAX`, `STD`, `SKEW`, `SUM` for numerical features, `MODE`, `NUM_UNIQUE` for categorical features and `COUNT` for both. Categorical features are one-hot encoded to their 10 most common categories. To apply DFS on YAGO3, we convert it to tabular format by creating a table with two columns (head, tail) for each forward/inverse relation.

1. Target entities for which we extract features from YAGO3 are underlined.

MANUAL FEATURE ENGINEERING Besides DFS, we include manual feature engineering to our empirical study. The objective is to estimate how well an analyst would perform given a time budget of 1-2 hours per dataset. Results obviously depend on the analyst and could be improved with more effort, but they provide a simple baseline for a time-constrained analysis. See appendix (Section A.2.2) for a description of the handcrafted features we used.

RDF2VEC Finally, we also compare our approach to RDF2vec, traditionally used to extract features for downstream tasks. For each entity under study, we generate all possible walks of depth 2, going through forward and backward relations (as in Figure 5.3). However, as the number of walks can be very high for certain entities (*e.g.* tens of millions), we cap this number to 10000, and checked empirically that this value is large enough to impact only a small fraction of entities. We then feed these sequences to a SGNS model with embedding dimension = 200, window size = 4 (which allows to capture 1-hop and 2-hop neighborhoods), and pick the epoch $\in$ [1, 5, 10, 20] that performs best. We used the pyRDF2Vec package (Vandewiele et al., 2020a) to run the experiments.

7.1.3 *Quality of the extracted features*

METHODOLOGY We first study how well feature vectors created from a source database can improve performance in data-science tasks. For this, we consider the prediction problems introduced in Section 7.1.1 and the feature extraction approaches presented in Section 7.1.2: TransE, DistMult and MuRE with and without KEN; Deep Feature Synthesis; manual feature engineering; and RDF2vec.

We measure performance with cross-validation scores, and only use entity representations to predict the target values [2]. For regression and classification, we use two analytic models from the *scikit-learn* library: k-nearest neighbors and gradient boosted trees, whose hyperparameters are tuned. We report in Table 7.1 5-fold cross-validation scores, averaged over multiple seeds for splitting the data in train-test and training the embedding models. See the appendix (Section A.2.3) for a more detailed description of the experimental setup.

RESULTS When using entity-embeddings as feature vectors, DistMult and **MuRE overall outperform RDF2vec by a wide margin** (except on the Employees dataset, where RDF2vec gets surprisingly good results), with MuRE appearing as the best approach. We explain this gap by their ability to capture well relational information. In particular, MuRE is more expressive than TransE and DistMult (their scoring functions can be seen as special cases of MuRE) and thus better model complex relations. In contrast, TransE does not model well many-to-one relationships: if we have (h, r, t) and (h', r, t), then h

2. Except in the Elections dataset, where we also include the political party when predicting the number of votes.

Table 7.1 – **Quality of the extracted features**: Cross-validation scores on target datasets using either embeddings, deep feature synthesis, or manually handcrafted vectors as features. The scoring metrics are: average precision (KDD14), AUC (KDD15) and R2 for the remaining datasets. Bold and underlined scores correspond to the first and second best-performing approaches. Grayed cells indicate when MuRE + KEN outperforms deep feature synthesis. Results with standard deviations are given in the appendix (Table A.5).

	Feature enrichment from domain-specific tables		*Feature enrichment from a general-purpose knowledge graph, YAGO3*				
Approach	**KDD14**	**KDD15**	**Elections**	**Housing prices**	**Accidents**	**Movie revenues**	**Employees**
Advanced analytic models: gradient boosted trees							
Feature vectors tailored for target entities							
Manual feature handcrafting	0.267	0.869	0.955	0.273	0.360	0.141	0.367
DFS, depth 0	0.158	0.584	0.836	0.165	0.162	0.016	0.126
DFS, depth 1	0.461	0.880	0.960	0.369	0.423	0.153	0.382
DFS, depth 2	0.463	0.880	0.964	0.605	0.570	0.163	0.384
DFS, depth 3	**0.499**	0.881	**0.969**	0.683	**0.590**	0.189	0.381
DFS, depth 3 + ontology			0.958	**0.686**	0.589	0.259	**0.390**
RDF2vec	0.173	0.849	0.873	0.355	0.236	0.074	0.380
General-purpose feature vectors							
TransE	0.242	0.854	0.899	0.321	0.256	0.092	0.003
TransE + KEN	0.334	0.875	0.939	0.447	0.381	0.095	0.214
DistMult	0.264	0.859	0.916	0.525	0.454	0.145	0.117
DistMult + LiteralE	0.286	0.870	0.841	0.484	0.443	0.110	0.227
DistMult + KEN	0.386	0.879	0.921	0.542	0.486	0.162	0.242
MuRE	0.287	0.863	0.945	0.571	0.461	0.165	0.109
MuRE + KEN	0.443	**0.883**	0.966	0.604	0.524	0.175	0.313
MuRE + KEN + ontology			0.957	0.602	0.541	**0.266**	0.345
Simple analytic models: K-Nearest Neighbors							
DFS, depth 0	0.078	0.504	0.742	0.004	0.130	-0.026	0.004
DFS, depth 1	0.110	0.821	0.715	0.297	0.320	0.121	0.144
DFS, depth 2	0.107	0.821	0.763	0.395	0.349	0.119	0.086
DFS, depth 3	0.142	0.816	0.618	0.503	0.361	0.043	0.025
MuRE + KEN	**0.205**	**0.830**	**0.936**	**0.536**	**0.488**	**0.136**	**0.273**

and h' are forced to have very close embeddings $\boldsymbol{h} = \boldsymbol{h'} = \boldsymbol{t} - \boldsymbol{r}$. Similarly, the scoring function of DistMult is symmetrical, i.e. $f(h, r, t) = f(t, r, h)$, which is not suited for non symmetrical relations like *locatedIn*. We can also see from Table 7.1 that leveraging numerical attributes with KEN *always* improves performance in TransE, DistMult and MuRE, and that it is superior to LiteralE in DistMult.

We now compare the performance of MuRE + KEN (the best embedding approach) to manual and automatic feature engineering methods. When using powerful prediction models (gradient boosted trees), MuRE + KEN does not consistently outperforms DFS, **but is often competitive** for depths $\leqslant 2$, and almost always **outperforms manual feature engineering**. However, when using simpler prediction models (K-Nearest Neighbors), MuRE + KEN significantly **outperforms DFS for all depths**. Indeed, embeddings tend to be well structured (as induced by the scoring function) and have homogeneous coefficients with similar distributions, which facilitates the downstream learning. In contrast, DFS creates a huge number of heterogeneous features, which even after scaling are hard to leverage by simple models.

We also study whether injecting taxonomic information into embedding models improves performance. Following d'Amato et al., 2021, we augment YAGO3 with triples describing its ontology, such as entity types and their relations (*subClassOf* and *disjointWith*). We apply MuRE + KEN on this augmented version of YAGO3 and observe that it generally improves prediction performance and reduces the gap with DFS.

CAPTURING ENTITY TYPES Finally, we investigate whether knowledge graph embeddings capture entity types, for instance differentiating cities from movies or counties. Such information can be useful in certain tasks that we did not consider in our previous experiments, *e.g.* clustering. To evaluate this, we take many entities of various types (cities, counties, movies, companies) from our previous tasks on YAGO3, and measure how well entity types can be predicted from their MuRE + KEN embeddings. We use a simple K-Nearest Neighbor model, whose number of neighbors is tuned and obtain a ROC AUC score of 0.996, showing that **knowledge graph embeddings indeed capture entity types**. We detail the experimental setup in the appendix (Section A.2.3).

7.1.4 *Scalability concerns*

Large databases, such as YAGO3, bear promises to provide general-purpose feature enrichment. For this, the scalability of features extraction methods is crucial. To that end, we compare in Table 7.2 the scalability of various approaches: Deep Feature Synthesis (for $0 \leqslant$ depth $\leqslant 3$), RDF2vec and MuRE (with and without KEN).

METHODOLOGY We quantify computational scalability with several metrics capturing:

1) the **scalability of feature extraction**: duration and RAM usage when computing the feature vectors.
2) the **scalability of feature usage**: dimension of the feature vectors, disk memory needed to store them, and duration of cross-validated evaluation in prediction tasks (using gradient boosted trees).

A benefit of knowledge graph embedding models is that they learn representations for **all entities at once** (*e.g.* cities, counties, movies in YAGO3). This is unlike DFS and RDF2vec which typically extracts feature vectors for target entities only. Given our objective to provide representations for many different entities, we thus benchmark DFS and RDF2vec when extracting features for all entities.

In some cases (KDD14 with depth 3 and YAGO3 with depth 2/3), **DFS breaks the RAM capacity of our machine (400 GB) and does not terminate**, even when splitting entities into 1000 chunks to lower the RAM usage. For these cases, we extrapolate the total duration based on the duration for a subset of entities, and the disk memory required to store features based on the memory it takes for a smaller number of features.

Similarly, we were not able to learn RDF2vec embeddings for all YAGO3 entities due to memory overflow. We tried limiting the number of walks to 100 per entity, and only generating them from the 1% most frequent ones, but we still could not compute them in less than a day, even with parallelization over 40 CPUs. We thus interrupted the process, and measured the duration and RAM usage just before stopping.

RESULTS We report in Table 7.2 the scalability metrics described above. As expected, **DFS quickly becomes intractable on large databases**: it requires huge amounts of time and RAM to run, and returns very high-dimensional feature vectors that need a lot of memory to be stored and a lot of time to be leveraged by machine-learning models. Interestingly, we saw in Table 7.1 that DFS must be computed at a depth of 2 or more to outperform MuRE + KEN (using powerful gradient boosted tree models). Yet based on this scalability study, this is already too deep to run DFS for all entities in YAGO3, due to memory issues. In the end, DFS produces high-performance features, but its usage is limited to small databases, or when the downstream task is known beforehand so as to extract features for a subset of entities only. Unlike knowledge graph embedding models, it cannot be used to create general-purpose feature vectors from large databases with millions of entities.

We observe similar trends with RDF2vec: feature extraction for all entities overall requires much more time and memory than MuRE. Actually, creating feature vectors for target entities (rather than all entities) with RDF2vec can take more time (*e.g.* 9300s for 23000 cities

Table 7.2 – **Scalability of feature extraction methods**: Computational scalability of embedding models versus deep feature synthesis. Grayed-out cells indicate models which are less tractable than MuRE + KEN. Red text indicates when DFS breaks the RAM capacity of our machine (400 GB).

Scalability metrics	Dataset	Deep Feature Synthesis *Depth 1*	*Depth 2*	*Depth 3*	MuRE	MuRE + KEN	RDF 2vec
Extracting feature vectors for all entities							
Duration (s)	KDD14	1014	11123	≃**110K**	2146	6708	52000
	KDD15	170	489	5107	3023	3566	1710
	YAGO3	690	≃**33K**	≃**8.5M**	1108	1762	⩾ **100K**
RAM usage (GB)	KDD14	10.5	48	⩾**400**	13.2	18.6	240
	KDD15	4.9	8.2	57.7	14.8	18.8	95
	YAGO3	40.1	⩾**400**	⩾**400**	15.9	16.1	⩾ **30**
Using feature vectors in downstream tasks							
Dimension of feature vectors	KDD14	372	2202	19379	200	200	200
	KDD15	163	277	1870	200	200	200
	YAGO3	271	10281	141K	200	200	200
Disk memory needed to store features (GB)	YAGO3	2.8	107	1471	2.1	2.1	2.1
Duration of cross-validated evaluation (s)	KDD14	48	91	1684	103	103	103
	KDD15	4	4	9.9	19	19	19
	Elections	100	276	8989	176	176	176
	Housing prices	89	330	11589	145	145	145
	Accidents	92	317	11496	146	146	146
	Movie revenues	56	356	14988	132	132	132
	Employees	72	449	15762	88	88	88

in Housing prices) than applying MuRE to all YAGO3 entities, and must be repeated for every new downstream task.

7.1.5 *KEN helps embeddings capture numerical attributes*

As visible on Figure 7.1, KEN provides embeddings that represent in a much simpler way the numerical information associated with entities. When embedding counties from YAGO3, the structure of KEN embeddings reflects well the population density, with a direction grouping together metropolitan areas such as Chicago (Cook county), Los Angeles (Orange County), Houston (Harris county), and Phoenix (Maricopa county), well separated from rural counties. On the other hand, this information is more diluted in standard MuRE embeddings.

METHODOLOGY To evaluate quantitatively the ability of embeddings to capture numerical information, we compare the performance of

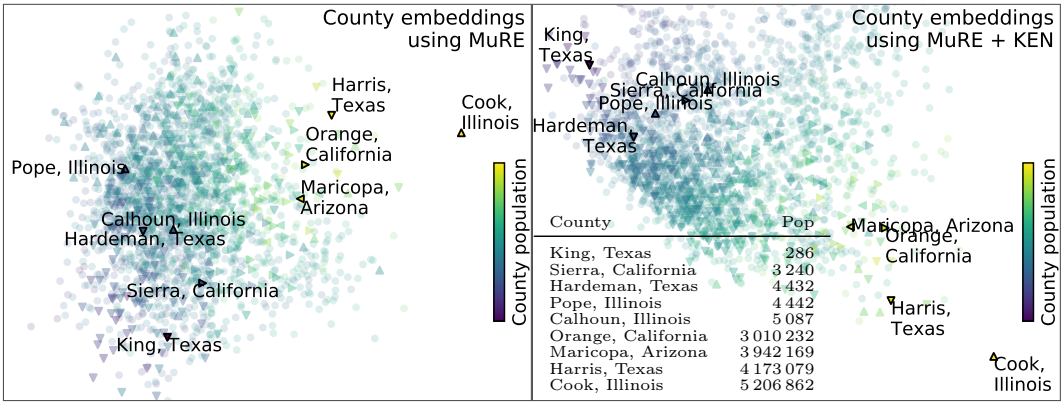

Figure 7.1 – **Embeddings of counties** using only categorical attributes (MuRE) or all attributes (KEN-E) from YAGO3: PCA projection of the 200-dimension embeddings in 2D. The color represents the county population and the symbols the state of the county. We randomly draw high and low population counties in the same state. Cook, Orange, Harris, and Maricopa counties correspond to major cities: Chicago, Los Angeles, Houston, and Phoenix. The global structure of MuRE + KEN embeddings better reflects the population of the counties, in particular separating the rural counties from those related to major cities. A simple linear projection of the MuRE + KEN embeddings suffices to roughly capture the rural-urban gradients, while it is less clear on MuRE embeddings.

simple supervised models to predict the numerical attributes of entities (*e.g.* county populations) from their embeddings. In practice we use K-Nearest Neighbors models (whose hyper-parameters are tuned) and aim to predict statistics about donations to projects in KDD14, students connections to MOOCs in KDD15 and county attributes in YAGO3. We measure performance with cross-validation scores. See the appendix (Section A.2.4) for the exact evaluation setup.

Table 7.3 – **Reconstructing numerical attributes -** Cross-validation scores (R2) of simple nearest-neighbour models predicting the numerical attributes associated to an entity from its embedding.

Target		DistMult	DistMult + LiteralE	DistMult + KEN	MuRE	MuRE + KEN
Donation amount (KDD14)	Mean	0.20±0.05	0.58±0.14	0.62±0.12	0.22±0.06	**0.66**±0.12
	1st quartile	0.34±0.05	0.46±0.05	0.67±0.10	0.34±0.06	**0.72**±0.12
	3rd quartile	0.33±0.05	0.48±0.05	0.57±0.10	0.33±0.05	**0.59**±0.09
Connection time (KDD15)	Mean	0.09±0.01	0.33±0.01	0.92±0.01	0.10±0.02	**0.97**±0.01
	1st quartile	0.15±0.01	0.27±0.01	0.78±0.01	0.15±0.01	**0.82**±0.01
	3rd quartile	0.39±0.02	0.45±0.01	0.74±0.01	0.39±0.02	**0.84**±0.01
County attributes (YAGO3)	Population	**0.73**±0.17	0.71±0.22	**0.73**±0.15	0.32±0.08	0.51±0.16
	Latitude	0.92±0.01	0.72±0.03	**0.93**±0.01	0.72±0.03	0.91±0.01
	Longitude	0.83±0.07	0.72±0.05	**0.90**±0.07	0.64±0.06	0.81±0.06

Table 7.4 – **Ablation study -** Drop in cross-validation scores of variants of MuRE + KEN and binning, relatively to the original MuRE + KEN. Scoring metrics are: average precision (KDD14), AUC (KDD15) and R2 for other datasets.

Dataset	**Binning**	**Variants of MuRE + KEN**	
		No quantile normalization	*No ReLU activation*
KDD14	-0.044	-0.068	-0.045
KDD15	-0.002	0	-0.001
Elections	-0.008	-0.020	-0.004
Housing prices	-0.091	-0.023	-0.021
Accidents	-0.063	-0.037	-0.010
Movie revenues	-0.015	-0.112	-0.030
Employees	-0.011	-0.007	0.002
Average across datasets	**-0.038**	**-0.047**	**-0.016**

RESULTS The scores reported in Table 7.3 confirms that **adding KEN significantly improves the ability to capture numerical information related to the entities**: in all settings adding KEN leads to better reconstruction of numerical attributes, and also outperforms LiteralE by a wide margin. In addition, results show that these embeddings capture to some extent the whole distribution of numerical attributes: their mean, but also their quantiles.

7.1.6 *Ablation study*

We study in this section the influence of two ingredients of KEN on the quality of entity-embeddings: 1) the quantile normalization of numerical values at the input, and 2) the presence of a ReLU activation function at the output (Figure 6.2).

METHODOLOGY We measure the drop in performance relative to the original MuRE + KEN when: 1) replacing the quantile normalization by a min-max normalization $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$ and 2) removing the ReLU activation. We also compare KEN to a standard binning practice, where numerical values are divided into bins and an embedding is learned for each bin. In practice we use 20 bins and split values evenly across bins to be robust to fat-tailed distributions: the first bin corresponds to values in the top 5%, the second bin to values in the range 5%-10%, and so on... We use gradient boosted tree models for prediction, and the same setup as in Table 7.1.

RESULTS Table 7.4 shows that **all ingredients of KEN are important**, especially the quantile normalization, and confirms that KEN leads to markedly better features than binning.

Table 7.5 – **Embedding can capture deep features**: Cross-validation scores (R2) of gradient boosted tree models using as features either embeddings trained on the full YAGO3 dataset, or on a subset of YAGO3 containing only the triples related to the target entities.

Dataset	YAGO3	TransE	TransE + KEN	MuRE	MuRE + KEN
Elections	subset	0.846	0.854	0.837	0.926
	full	0.899	0.939	0.945	0.966
Housing prices	subset	0.079	0.203	0.231	0.338
	full	0.321	0.447	0.571	0.604
Accidents	subset	0.117	0.170	0.243	0.345
	full	0.256	0.381	0.461	0.524
Movie revenues	subset	-0.003	-0.004	0.052	0.064
	full	0.092	0.095	0.165	0.175
Employees	subset	-0.015	0.071	0.087	0.297
	full	0.003	0.214	0.109	0.313

7.1.7 *Capturing deep features with embeddings*

METHODOLOGY We want to determine if embeddings can capture information deep in the knowledge graph, indirectly chaining relations as in Deep Feature Synthesis. For this purpose, we compare in Table 7.5 cross-validation scores of gradient boosted tree models with embeddings trained either on the full YAGO3 database, or on a subset of YAGO3 containing only the triples related to the target entities. For example, a subset with city-related triples would contain direct information about cities (*e.g.* the state in which they belong), but no information about the states themselves. Such "deep" information can however be helpful for analytical tasks, and should be captured by embeddings models. The evaluation setup is the same as in Table 7.1.

RESULTS Table 7.5 shows that adding triples indirectly related to the target entities improves the quality of their embeddings; hence **embedding models do capture deep information**.

7.1.8 *Influence of table representations*

METHODOLOGY When the source data consists of tables, it must be represented as a knowledge graph to be leveraged by our approach. We introduced in Section 6.3 three table-to-graph strategies, which differ on which entities are used as heads when generating triples (Figure 6.3). We either use: 1) all entities, 2) only target entities (which require some prior knowledge of the downstream application) or 3) row ids. We evaluate the performance of these strategies with cross-validation scores on KDD14 and KDD15, using gradient boosted tree models for prediction (as in Table 7.1). To show the importance of choosing well the column with the target entities in the second ap-

proach, we also evaluate a simple baseline taking entities from another column.

RESULTS Based on Table 7.6, the top performing table-to-graph strategy consists in generating triples from target entities. Indeed, the resulting graph directly connects them to their attributes, which facilitates the learning of embeddings. This intuition is confirmed when taking instead entities from another column, as we observe a sharp drop in performance. Interestingly, using all entities or row ids as head entities return embeddings that perform reasonably well without being tailored for the specific task at hand. These methods can provide general-purpose embeddings that perform well for various entities and applications. However, they either increase the number of triples (and thus the training time of embeddings) or the number of entities.

7.2 DISCUSSION

7.2.1 *Embeddings capturing numerical information can provide feature enrichment*

By relying on entity embeddings, our feature-synthesis pipeline departs strongly from the standard approach of feature engineering in databases. Our extensive experiments confirm that features created via knowledge graph embedding do capture the information needed for a statistical task. Embedding models coupled with KEN improve over manual feature engineering on almost all tasks.

We observe clear trends in the experimental results: Table 7.1 reveals the importance of capturing well *1)* the numerical attributes and *2)* relational, rather than contextual information. Indeed, across all analytic tasks and embedding methods explored, adding KEN leads to features that better capture numerical attributes and improve the

Table 7.6 – **Influence of table representations**: Cross-validation scores of different strategies to represent tables as a knowledge graph. Scoring metrics are average precision (KDD14) and AUC (KDD15). We also report the number of entities and triples (in millions) in the graph from each method.

Head entities in generated triples	**KDD14**		**KDD15**		**# triples** (KDD14, KDD15)	**# entities** (KDD14, KDD15)
	MuRE	MuRE + KEN	MuRE	MuRE + KEN		
Embeddings tailored for specific entities						
Target entities	0.287	**0.443**	0.863	**0.883**	44, 33	0.94, 0.27
Entities from another column	0.227	0.233	0.861	0.863	44, 33	0.94, 0.27
General-purpose embeddings						
All entities	**0.289**	0.406	**0.864**	**0.883**	155, 66	0.94, 0.27
Row IDs	0.282	0.409	0.856	0.878	51, 41	8.4, 8.5

downstream analytic task (Table 7.3 and Table 7.1). It also improves over binning and LiteralE by a large margin. The ingredients that we introduced in KEN, such as the quantile normalization to account for the distribution of numerical attributes significantly improves performance (Table 7.4). Improving models of relations makes a strong difference in how useful the resulting features are for downstream tasks: there are notable improvements from RDF2vec –no explicit model of the relation– to MuRE (Table 7.1).

7.2.2 *Deep Feature Synthesis cannot go so deep*

Automated feature-engineering methods like Deep Feature Synthesis greatly reduce the human cost of manually handcrafting features across tables, while achieving excellent results on all datasets. With deep-enough features, DFS performs consistently better than manual feature engineering and often slightly better that MuRE + KEN (Table 7.1).

However this ability to generate good features comes at the price of scalability. Since DFS combines aggregation functions and features at each depth, **the time and space complexity, as well as the number of created features grow exponentially** (Table 7.2). Even on relatively small databases like KDD14 or YAGO3, building features for all entities with DFS at a depth of 2 or 3 becomes intractable, with the memory requirements greatly exceeding our machine capacity (400 GB). Besides memory limitations, the number of features quickly reaches tens or hundreds of thousands, **making statistical models harder and slower to train** (*e.g.* 180x longer on Employees), and reducing feature interpretability.

Yet, the databases that we have explored are smaller than the latest repositories of general knowledge: YAGO3 is 50 times smaller than YAGO4 (Pellissier Tanon et al., 2020). Progress in linked open data is continuously increasing the amount of information available in a consistent representation: DBPedia (Lehmann et al., 2015) currently contains 900 millions triplets, and growth by a factor of 1.5 to 2 every two years (*DBPedia web page* n.d.). For instance, we could not run DFS, even with a depth of 1, on YAGO4. Even if it could run, it would provide a huge number of features, hard to leverage.

Embeddings, on the opposite, readily provide low-dimension representations ($p = 200$) which are able to capture "deep" information, indirectly chaining relations (Table 7.5). Finally, knowledge graph embedding methods are very scalable: embeddings are optimized with stochastic gradient descent ($\mathcal{O}(\#\text{triplets})$), and can be trained on huge amounts of data. Further optimizations can make embedding techniques $2-5\times$ faster than the implementations that we used (Zheng et al., 2020).

Knowledge graph embedding models are also naturally suited to capture complex relational patterns between discrete elements. This is unlike DFS, which struggles to encode categorical features: ensembles of discrete entities (e.g. the cities located in a county) are ag-

gregated by their most common element and then one-hot encoded, discarding a lot of information in the process.

7.2.3 *Current limitations call for further work*

INTERPRETABILITY The biggest drawback of automatic feature generation is that it leads to models harder to interpret. Indeed, features are often manually crafted to capture a quantity of interest, such as wealth of a locality. Data scientists can then reason about the role of the corresponding quantity, for instance the impact of local wealth on housing prices. A challenge to these interpretations is that the crafted feature must represent well the quantity, but for this the burden is on the analyst and not the tool. With automatically generated features, the quantities of interest must be identified from the features. This is typically hard: even in DFS where features are associated with descriptive labels, we may have to distinguish between many partly redundant features. This is even harder in embedding models, which are black-box and do not associate human-understandable labels to individual features.

MATCHING OUT-OF-VOCABULARY ENTITIES The target data may come with different naming conventions than the source, for instance county names in the Elections dataset are written differently than in YAGO3. In such case, **a form of matching must be performed** (e.g. *Cook County* → *Cook, Illinois*). This is often done manually using domain-knowledge. Further work should explore automated techniques, for instance using fuzzy or similarity joins (Mann et al., 2016; Silva et al., 2010), or adapting NLP techniques used to create **embeddings robust to out-of-vocabulary entities** (Bojanowski et al., 2016; L. Chen et al., 2022; Pinter et al., 2017).

SCALABILITY Even though knowledge graph embedding models are much more scalable than combinatorial feature engineering methods like DFS, training them on very large databases can still be challenging. Indeed, they typically learn a **separate** embedding for each entity, and may thus not fit in memory when there are a lot of them. For instance, storing embeddings for the 64 millions of entities in YAGO4 requires about 90GB of RAM, which is above the RAM capacity of most GPU devices. Several strategies have been explored to improve the scalability of knowledge graph embedding models. NodePiece (Galkin et al., 2022) combines embeddings from a small number of *anchor* entities to form representations for other entities, reducing the number of parameters to learn by 90-99%. Other approaches focus on graph partitioning, creating smaller, disjoint subgraphs whose entity embeddings can fit in memory and be trained separately (Kochsiek and Gemulla, 2021).

COMPLEX AND HETEROGENEOUS DATA So far, the knowledge graph embedding models used in our approach only capture relationships

between discrete entities, and numerical attributes (using KEN). Yet, knowledge graphs and tables may come with complex or heterogeneous information that is relevant for downstream tasks. To capture them, several approaches have been proposed in the knowledge graph embedding literature. For instance, diachronic embedding (Goel et al., 2019) accounts for the temporal nature of the data by representing each entity with a *dynamic*, rather than static embedding. Other methods consider not only the structure of the graph to learn entity embeddings, but also unstructured information that may come with it, such as text or images (Gesese et al., 2021).

7.3 CONCLUSION

We have shown how turn-key extraction of embeddings from relational data can distill valuable information from a database, synthesizing feature vectors for data enrichment in downstream analytic tasks. For these feature vectors to be most useful in the analytic tasks, experiments show that embedding methods must model well the different relations between entities, and capture their numerical attributes. For this, we proposed to use knowledge graph embedding models designed for link prediction, and extended them to numerical attribute with KEN. Our extensive experiments show that these embeddings improve markedly upon manual feature engineering and embedding methods traditionally used for feature extraction such as RDF2vec. They are also competitive with automatic feature engineering methods based on systematic denormalizations like Deep Feature Synthesis, but do not face the same scalability challenges.

A PIPELINE TO MINIMIZE HUMAN EFFORT Our pipeline is designed to facilitate data preparation. Not only does it circumvent the human labor of designing manual features, but also is minimizes data integration and wrangling challenges. Operating on a triple representation –sometimes automatically built from tables– removes many tedious aspects of data input. For instance it works well on tables in "long" or "wide" formats. It also allows to capture and mix information from various data structures: tables, knowledge graphs... Yet, richer representations may be useful in the long run to better capture complex relationships within the data, such as temporal dependencies (Arora and Bedathur, 2020).

TOWARDS GENERAL-PURPOSE FEATURE ENRICHMENT The scalability of our approach enabled to easily extract embeddings from YAGO3, capturing the corresponding information drawn from Wikipedia. These could readily be used as feature enrichment to improve statistical analysis on 5 different socio-economic datasets we investigated. Our work thus opens a path to capturing the large and complex stores of general information into feature vectors easy to integrate into any analysis. As such it contributes a major step towards facilitating data science with less manual data preparation.

8 CONCLUSION

8.1 MACHINE LEARNING ON EMBEDDINGS ENABLES DATA ANALYSIS WITHOUT CLEANING

In Part 1, we considered the problem of data analysis across non-normalized sources. Although leveraging data from various sources is useful to establish more general findings, standard analytic methods relying on data queries require prior entity matching to select all instances from the population of interest. Despite dedicated softwares and progress in automating entity matching, it remains a challenging task that often requires domain-expertise and manual supervision over a large number of entries.

As an alternative, we showed that machine learning models can be used to estimate various statistical quantities involved in data analyses. Importantly, these models do not rely on query aggregates, and thus do not require *exact* correspondences between entities. Instead, they can leverage vector representations that express *similarities* between entries: those which denote the same entity should have close representations.

Our experimental results support these claims: on a socio-economic study of salaries across 14 employee databases, approaches based on simple machine learning models and embeddings obtain similar or smaller estimation errors than those based on matching and queries, while requiring much less human effort. These results have practical implications for data analysis and management practices. Cleaning faces discrete challenges, *i.e.* to match or not to match, that is not suited to the ambiguities of real-world data and may thus bias analyses. Instead, representing these uncertainties in embeddings and passing them directly to a supervised model improves the quality of analysis.

Finally, recent progresses in embedding models present opportunities to better capture similarities between entities, such as state-of-the-art transformer-based models (Devlin et al., 2018), multi-lingual embeddings (X. Chen and Cardie, 2018), architectures robust to string variations and out-of-vocabulary entities (L. Chen et al., 2022; Pinter et al., 2017), or embedding models trained on the data at hand to adapt to its specificities (Cappuzzo et al., 2020).

8.2 EMBEDDINGS ENABLE JOINS OVER RELATIONAL DATA WITHOUT FEATURE ENGINEERING

We then addressed in Part 2 the problem of joining information across relational data. While joining new features to a table is often helpful in data analyses, it usually requires tedious feature engineer-

ing to assemble the information scattered across databases into a single feature matrix. Automatic methods for feature engineering have been proposed, but are generally combinatorial in nature and do not scale to large databases.

Instead, we showed that by representing the relational data as a knowledge graph, embedding models can be directly applied to the *unassembled* data to provide feature vectors for the entities under study. This approach offers multiple advantages: the triple representation used in knowledge graphs allows to easily incorporate information from other sources (*e.g.* tables), which is then captured in low-dimensional (hence scalable) embeddings. However, a major drawback of embedding models is that they are typically designed for discrete entities, but not for numerical values which are often informative. We thus introduced KEN, a module that extends knowledge graph embedding models to numerical attributes.

We thoroughly evaluated the performance and scalability of our approach on multiple analytical tasks that we enriched with background information from tables or knowledge graphs. Our results shows that it is competitive with state-of-the-art automatic feature engineering methods, while being much more scalable, and that it outperforms manual feature engineering. They also demonstrate that capturing numerical attributes with KEN leads to significant increases in performance. Our approach thus contributes a major step towards analytics with less data assembling, and paves the way for general-purpose entity embeddings to be used in various downstream applications. Finally, a remaining challenge to fully enable joins over relational data is to make embedding models robust to variations in the entity names.

8.3 EMBEDDING MODELS BRING NEW OPPORTUNITIES TO DATA ANALYSIS

Beyond the specific problems that we addressed in this thesis, embedding models bring many other opportunities to data analysis. Pretrained or data-specific embeddings have been successfully used in various tasks such as record linkage (Cappuzzo et al., 2020; Ebraheem et al., 2018; Zhao and He, 2019), schema matching (Koutras et al., 2020), table retrieval (Zhang et al., 2019) or more generally for data curation (Thirumuruganathan et al., 2020).

Nowadays, bigger and bigger embedding models are being trained on increasingly large amounts of data (Sharir et al., 2020). For example, multi-modal embedding models are able to integrate altogether data of various natures, such as tables, graphs, images or text (Pezeshkpour et al., 2018; Yin et al., 2020). In natural language processing, very large language models such as GPT-3 (Brown et al., 2020) or T5 (Raffel et al., 2019) now encapsulate huge amounts of human knowledge that can be harnessed in multiple tasks, such as automating data wrangling operations (Jaimovitch-López et al., 2023), or augmenting and curating knowledge bases (Razniewski et

al., 2021). Similar opportunities exist beyond language models: large knowledge bases such as YAGO4 (Pellissier Tanon et al., 2020) and huge corpuses of tables (Cafarella et al., 2008; Hulsebos et al., 2021) provide valuable information about millions of entities that has yet to be captured in rich representations and leveraged in numerous applications.

8.4 LESSONS I HAVE LEARNED

Throughout this thesis, a recurring and fundamental question that I faced was: *"How to learn rich, general-purpose representations of entities that could be used in a myriad of different tasks?"*. After three years of research, two key elements have appeared to me as crucial in this regard. The first ingredient towards good entity embeddings is the data from which we learn: to induce representations with rich semantics, data must be as exhaustive and diverse as possible, aggregating information about entities from various and large sources. This observation has practical implications: we need embedding models that can scale to large databases, and input information from diverse sources. For this, knowledge graph embedding models naturally appeared as a good solution.

The second ingredient towards good entity embeddings is the embedding model itself, which must *faithfully* encapsulate the information scattered in the data into the learned representations. Indeed, even with the best training data, a model that is not expressive enough to accurately represent it will result in poor embeddings (*e.g. contextual*, RDF2vec embeddings). However, the challenge is that there are often multiple information of different natures that co-exist in the data, and knowing which ones are relevant for a certain task or which ones are actually captured by the model is not straightforward. For example, knowledge graphs provide *relational* information about entities, *e.g.* knowing that two entities $\mathtt{h}, \mathtt{t}$ are linked through the relation $\mathtt{r}$. But they also contain information about the *types* of entities: an entity $\mathtt{h}$ that appears in a relation *livesIn* is likely to be a person. Yet it was unexpected to see that this information is captured as well by knowledge graph embedding models (Section 7.1.3), even though they were not primarily trained for that. Similarly, knowledge graphs inform us about the "importance" of entities via the number of triples in which they appear. And again, I was surprised to observe that embeddings implicitly capture the frequency of entities in the data.

In the end, I believe these observations call for more experiments to determine when and how embedding models are able or fail to capture certain patterns in the data, with the aim of learning better representations.

APPENDICES

APPENDIX

A.1 DATA ANALYTICS ACROSS NON-NORMALIZED SOURCES

We provide here the implementation details of the estimation methods presented in Part 1.3.3.2.

A.1.1 *Embedding and fuzzy matching*

Matching and averaging estimates can exhibit high variance, especially when the groups over which they are computed contain few employees. To obtain more reliable estimates, fuzzy matching averages manual matching estimates $\hat{\tau}_{\text{matching}}(j', e)$ over several jobs j', giving more weight to those that are similar to the job j of interest. For instance, when estimating the mean salary given the job j and experience level e, the fuzzy estimator is defined as follows:

$$\hat{\tau}_{\text{fuzzy}}(j, e) = \frac{\sum_{j' \in J} \hat{\tau}_{\text{matching}}(j', e) \cdot \text{sim}(j', j)}{\sum_{j' \in J} \text{sim}(j', j)} \tag{A.1}$$

with J the set of all job titles and $\text{sim}(j', j) \geqslant 0$ the string similarity between the job j' and the job j of interest.

To define the string similarity $\text{sim}(j_1, j_2)$ between job titles, we encode them into vectors $\mathbf{j_1}, \mathbf{j_2}$ using a pretrained fastText model [1] and compute their cosine similarity:

$$c(j_1, j_2) = \frac{\mathbf{j_1} \cdot \mathbf{j_2}}{||\mathbf{j_1}||\ ||\mathbf{j_2}||} \in [-1, 1]$$

We finally obtain the similarity score by rescaling the cosine similarity into $[0, 1]$, based on a threshold t that we tune to minimize cross-validation errors:

$$\text{sim}(j_1, j_2) = \begin{cases} \frac{c(j_1, j_2) - t}{1 - t} & \text{if } c(j_1, j_2) \geqslant t \\ 0 & \text{otherwise} \end{cases}$$

We follow the same procedure for all the tasks evaluated in Table 4.1, and tune the threshold $t \in \{0.9, 0.8, 0.7, 0.6, 0.5\}$ to minimize cross-validation scores.

We follow a slightly different strategy when studying the causal effect of gender on salary. When an employee i has counterparts of the opposite sex in the data (*i.e.* $O_i \neq \emptyset$), we keep the matching & averaging estimates. However, if $O_i = \emptyset$, we consider instead the sets $O_i^{(j)}$ of employees with the same covariates, **except for their job title**

1. The fastText model for english words can be downloaded here: https://fasttext.cc/docs/en/crawl-vectors.html.

$j \neq j_i$, and of opposite sex. For each set $O_i^{(j)}$ we compute the average salary $\bar{y}(O_i^{(j)})$, and finally estimate y_i^{unobs} as a weighted average over the different $\bar{y}(O_i^{(j)})$, based on the similarity between j and j_i.

$$\widehat{\tau}_{\text{fuzzy}} = \frac{1}{n}\sum_{i=1}^{n} W_i(y_i - \widehat{y_i}^{\text{unobs}}) + (1 - W_i)(\widehat{y_i}^{\text{unobs}} - y_i) \tag{A.2}$$

with

$$\widehat{y_i}^{\text{unobs}} = \begin{cases} \frac{1}{|O_i|}\sum\limits_{k \in O_i} y_k & \text{if } O_i \neq \emptyset \\ \\ \frac{\sum\limits_{j \in J} \bar{y}(O_i^{(j)})\,\text{sim}(j, j_i)}{\sum\limits_{j \in J} \text{sim}(j, j_i)} & \text{otherwise} \end{cases} \tag{A.3}$$

We use the same similarity score as before, with a threshold $t = 0.8$.

A.1.2 *Embedding and learning*

For regression and classification tasks, we use gradient boosted trees [2] as machine-learning model f_θ. We also use vector representations $\mathbf{j_i}$ of the job titles as features (obtained from a pretrained fastText model) to implicitly account for entity-matching.

For the tasks evaluated in Table 4.1, we tune the learning rate $\alpha \in \{0.01, 0.03, 0.1, 0.3\}$ of the models to minimize cross-validation errors. When doing causal analysis (Figure 4.1), we tune their learning rates $\in$ [0.1, 0.3, 0.5] and maximum depths $\in$ [8, 12, None].

2. See https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html and https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

A.2 ENRICHING DATA ANALYSES WITH BACKGROUND INFORMATION

A.2.1 *Downstream tasks*

TABULAR DATA

— The **KDD14** competition aims to predict "exciting" educational projects on a crowdfunding platform (binary target). The source data consists of three tables describing the projects, the donations they received, and the resources they need. The exact columns used in our experiments are described in Table A.1. Since embedding models with KEN are designed for discrete entities or numerical values, we perform minimal preprocessing on a few columns with different data types. For instance, we encode *donations_message* (free text) by their length. Temporal data, such as *donation_timestamp* are converted to a number of days after the project posting date. We also convert *date_posted* to a number of days after an arbitrary reference date. For a fair comparison, we use the same preprocessed features when running DFS.

— The **KDD15** challenge aims to predict student dropout prediction in MOOCs (binary target), using as source data 4 tables that contain information about the courses and how often students interacted with them (see Table A.2). To account for the temporal information in KDD15, we replace logs times (date) by numbers in $[0, 1]$, describing when they occur relatively to the courses start/end dates. We also replace the courses starting dates by a number of days after a reference date, and drop the ending dates as all courses have the same duration (29 days), making this feature uninformative.

DATASETS AUGMENTED WITH YAGO3 EMBEDDINGS

— **Elections -** We consider voting statistics in the 2020 presidential election, and aim to predict the number of votes per party in 3000 US counties. As the original data (Lab, 2018) come with no general information about counties, we enrich them with county embeddings learned on YAGO3

— **Housing prices -** We want to predict the typical housing price in 23000 US cities using their YAGO3 embeddings. We take target estimates from the Zillow group (Zillow, 2021).

— **Accidents -** We aim to predict the number of accidents in 8500 US cities between 2016 and 2020 using their YAGO3 embeddings. We use data described in (Moosavi et al., 2019).

— **Movie revenues -** We aim to predict the box-office revenues of 4900 movies using their YAGO3 embeddings. We used data from: https://www.kaggle.com/rounakbanik/the-movies-dataset.

— **Employees -** We aim to predict the number of employees in 3000 companies using their YAGO3 embeddings. We used data from: https://www.kaggle.com/peopledatalabssf/free-7-million-company-dataset

Since all these targets span over several orders of magnitude. We predict log(target) instead of the target in our experiments.

STATISTICS ON SOURCE DATASETS We give in Table A.3 the number of entities, relations and triples in the knowledge graph representations of the source data used to learn entity-embeddings.

A.2.2 *Approaches considered for evaluation*

OUR APPROACH When training embedding models (MuRE, DistMult and TransE), we do not tune hyper-parameters and use the following values in all experiments:

— embedding dimension = 200

— distance in scoring function: ℓ_2 norm for MuRE, ℓ_1 norm for TransE and DistMult

— batch size = 10^5

— optimizer: Adam with learning rate = 10^{-3}.

Table A.1 – Description of the KDD14 dataset. The *outcomes* table contains the target entities *project_id* for which we want to create feature vectors, and the binary value to predict *is_exciting*. We always use *project_id* as the head column when building the graph.

outcomes
project_id (str)
is_exciting (target)

donations
project_id (str)
donor_city (str)
donor_state (str)
is_teacher_acct (bool)
donation_timestamp (date)
donation_to_project (float)
donation_optional_support (float)
donation_message (text)

resources
project_id (str)
project_resource_type (str)
item_unit_price (float)
item_quantity (int)

projects
project_id (str)
teacher_id (str)
school_id (str)
school_city (str)
school_state (str)
primary_focus_subject (str)
primary_focus_area (str)
secondary_focus_subject (str)
secondary_focus_area (str)
resource_type (str)
poverty_level (str)
grade_level (str)
eligible_double_your_impact_match (bool)
eligible_almost_home_match (bool)
total_price_excluding_optional_support (float)
total_price_including_optional_support (float)
students_reached (float)
date_posted (date)

Table A.2 – Description of the KDD15 dataset. The *outcomes* table contains the target entities *enrollment_id* for which we want to create feature vectors, and the binary value to predict *dropout*. We use the first column of each table as head column when building the graph.

outcomes
enrollment_id (str)
dropout (target)

enrollments
enrollment_id (str)
student_id (str)
course_id (str)

dates
course_id (str)
from (date) — Course starting date
to (date) — Course ending date

objects (course modules)
module_id (str) — A module of a course
course_id (str)
category (str)

logs (student interactions with courses)
enrollment_id (str)
event (str) — Type of interaction
source (str) — Event source
object (str) — Module being interacted with
time (date) — Time of the event

— loss function: margin loss with $\gamma = 4$ in TransE, and a softplus loss for MuRE and DistMult

— negative sampling: for each positive triple (h, r, t), we generate 10 negative samples by replacing the head h by a random entity h' that co-occurs with the relation r. Doing so provides harder negative triples and improves the results.

We then train each model for 40 epochs, and pick the epoch $\in [2, 4, 8, 16, 24, 32, 40]$ that leads to the best cross-validation scores in downstream tasks.

A technical subtlety with MuRE is that we must define biases $b_t(x)$ for numerical values x. We do so by learning a constant bias b_r for each numerical attribute r: $\forall x, b_t(x) = b_r$.

MANUAL FEATURE ENGINEERING We describe below the typical feature engineering steps that we performed. See Table A.4 for the exact list of handcrafted features.

— identifying relevant features

— building features using joins and simple aggregation functions (mean, counts)

Table A.3 – **Statistics of the knowledge graphs representations** for the data used to train embeddings in our experiments. Numbers in parenthesis describe the part of numerical relations and triplets in the total.

Source data	Entities	Relations	Triples
KDD14	945k	27 (10)	44M (22.3M)
KDD15	227k	9 (2)	33M (8.2M)
YAGO3	2.8M	58 (21)	7.2M (1.6M)

Table A.4 – Manually handcrafted features for each dataset.

Dataset	**Handcrafted features**	
	Numerical	Categorical (one-hot encoded)
KDD14	– donation_to_project (mean, counts) – length_donation_message (mean) – students_reached – total_price_excluding_optional_support – total_price_including_optional_support – eligible_double_your_impact_match – eligible_almost_home_match	– primary_focus_subject – primary_focus_area – resource_type – poverty_level – grade_level
KDD15	– # of interactions (events) with courses – mean event time (relative to the course starting/ending dates) – course starting date – # of modules per course	– course_id
Elections	county population, latitude, longitude, area, population density	
Housing prices	city population, latitude, longitude, area, population density	
Accidents	city population, latitude, longitude, area, population density	
Movie revenues	– duration of the movie – number of actors, creators, editors, directors, music writers	– country of production
Employees	– mean value of all numerical attributes that exist for at least 5% of the companies – counts of all non-numerical attributes that exist for at least 5% of the companies	

— one-hot encoding of low-cardinality categorical features

— removing irrelevant, redundant, or hard to encode features (e.g. with high cardinality)

A.2.3 *Quality of the extracted features*

When using gradient boosted tree models (which offer native support for missing values), we use the default parameters from sklearn, except on the smaller datasets using YAGO3 embeddings. For these datasets, we tune the following model parameters with a cross-validated grid search: *max_depth* $\in$ [2, 4, 6, None] and *min_samples_leaf* $\in$ [4, 6, 10, 20].
When using KNNs, we tune the number of neighbors $\in$ [1, 3, 5, 10, 30], except on KDD14/15. We also impute missing values (common in DFS) with the median of each feature, and then normalize feature values between 0 and 1 with min-max scaling.

We report in Table 7.1 5-fold cross-validation scores, averaged across 5 random shuffles of the data (3 for KDD14/15) and over 3 different seeds for training the RDF2vec and knowledge graph embeddings (1 for KDD14/15). We also provide in Table the standard deviations across train-test splits associated to these scores.

To evaluate the ability of knowledge graph embedding models to capture entity types, we sample 1000 entities from the following datasets: Elections (counties), Housing prices (cities), Movie revenues (movies), Employees (companies), for a total of 4000 entities. When then measure with cross-validation how well MuRE + KEN embeddings predict entity types, using a simple KNN model whose number of neighbors $\in$ [1, 3, 5, 10, 30] is tuned. The cross-validation parameters are the same as above.

A.2.4 *KEN helps embeddings capture numerical attributes*

We obtain the results from Table 7.3 by predicting certain numerical attributes of entities from their embeddings, using simple K-Nearest Neighbors models. For the embeddings, we kept those from Table 7.1. We also tuned the hyper-parameters of nearest neighbors models to maximize prediction performance, using a cross-validated grid search over the following parameters:

— number of neighbors $\in$ [1, 2, 3, 4, 8, 16]

— distance: ℓ_1 or ℓ_2 norm

— weighting of the neighbors: uniform or proportional to the distance with the target entity

The final scores are then obtained with 5-fold cross-validation, averaged over 5 repeats.

Table A.5 – **Quality of the extracted features**: Cross-validation scores and standard deviations on target datasets using either embeddings, deep feature synthesis, or manually handcrafted vectors as features.

	Feature enrichment from domain-specific tables		*Feature enrichment from a general-purpose knowledge graph, YAGO3*				
Approach	**KDD14**	**KDD15**	**Elections**	**Housing prices**	**Accidents**	**Movie revenues**	**Employees**
Advanced analytic models: gradient boosted trees							
Feature vectors tailored for target entities							
Manual feature handcrafting	0.267 ± 0.004	0.869 ± 0.002	0.955 ± 0.003	0.273 ± 0.011	0.360 ± 0.021	0.141 ± 0.017	0.367 ± 0.035
DFS, depth 0	0.158 ± 0.002	0.584 ± 0.005	0.836 ± 0.007	0.165 ± 0.010	0.162 ± 0.016	0.016 ± 0.010	0.126 ± 0.036
DFS, depth 1	0.461 ± 0.006	0.880 ± 0.003	0.960 ± 0.003	0.369 ± 0.014	0.423 ± 0.020	0.153 ± 0.019	0.382 ± 0.035
DFS, depth 2	0.463 ± 0.006	0.880 ± 0.003	0.964 ± 0.003	0.605 ± 0.029	0.570 ± 0.016	0.163 ± 0.019	0.384 ± 0.035
DFS, depth 3	**0.499** ± 0.007	0.881 ± 0.003	**0.969** ± 0.002	0.683 ± 0.019	**0.590** ± 0.014	0.189 ± 0.023	0.381 ± 0.033
DFS, depth 3 + ontology			0.958 ± 0.005	**0.686** ± 0.019	0.589 ± 0.015	0.259 ± 0.023	**0.390** ± 0.031
RDF2vec	0.173 ± 0.003	0.849 ± 0.003	0.873 ± 0.009	0.355 ± 0.029	0.236 ± 0.019	0.074 ± 0.014	0.380 ± 0.047
General-purpose feature vectors							
TransE	0.242 ± 0.004	0.854 ± 0.003	0.899 ± 0.005	0.321 ± 0.046	0.256 ± 0.019	0.092 ± 0.016	0.003 ± 0.016
TransE + KEN	0.334 ± 0.004	0.875 ± 0.003	0.939 ± 0.006	0.447 ± 0.030	0.381 ± 0.020	0.095 ± 0.018	0.214 ± 0.031
DistMult	0.264 ± 0.004	0.859 ± 0.003	0.916 ± 0.043	0.525 ± 0.022	0.454 ± 0.023	0.145 ± 0.019	0.117 ± 0.032
DistMult + LiteralE	0.286 ± 0.005	0.870 ± 0.003	0.841 ± 0.038	0.484 ± 0.013	0.443 ± 0.022	0.110 ± 0.021	0.227 ± 0.029
DistMult + KEN	0.386 ± 0.007	0.879 ± 0.003	0.921 ± 0.053	0.542 ± 0.020	0.486 ± 0.022	0.162 ± 0.021	0.242 ± 0.034
MuRE	0.287 ± 0.005	0.863 ± 0.003	0.945 ± 0.015	0.571 ± 0.030	0.461 ± 0.021	0.165 ± 0.022	0.109 ± 0.038
MuRE + KEN	0.443 ± 0.006	**0.883** ± 0.003	0.966 ± 0.005	0.604 ± 0.020	0.524 ± 0.020	0.175 ± 0.025	0.313 ± 0.039
MuRE + KEN + ontology			0.957 ± 0.014	0.602 ± 0.028	0.541 ± 0.022	**0.266** ± 0.030	0.345 ± 0.040
Simple analytic models: K-Nearest Neighbors							
DFS, depth 0	0.078 ± 0.001	0.504 ± 0.030	0.742 ± 0.024	0.004 ± 0.027	0.130 ± 0.021	-0.026 ± 0.061	0.004 ± 0.140
DFS, depth 1	0.110 ± 0.002	0.821 ± 0.004	0.715 ± 0.019	0.297 ± 0.015	0.320 ± 0.023	0.121 ± 0.021	0.144 ± 0.020
DFS, depth 2	0.107 ± 0.001	0.821 ± 0.004	0.763 ± 0.015	0.395 ± 0.010	0.349 ± 0.020	0.119 ± 0.022	0.086 ± 0.028
DFS, depth 3	0.142 ± 0.002	0.816 ± 0.003	0.618 ± 0.050	0.503 ± 0.011	0.361 ± 0.021	0.043 ± 0.037	0.025 ± 0.022
MuRE + KEN	**0.205** ± 0.003	**0.830** ± 0.003	**0.936** ± 0.017	**0.536** ± 0.013	**0.488** ± 0.021	**0.136** ± 0.024	**0.273** ± 0.034

REFERENCES

Agrawal, R., & Ieong, S. (2012). Aggregating web offers to determine product prices. In *Proceedings of the 18th acm sigkdd international conference on knowledge discovery and data mining* (pp. 435–443).

Ali, M., Berrendorf, M., Hoyt, C. T., Vermue, L., Galkin, M., Sharifzadeh, S., … Lehmann, J. (2020). Bringing light into the dark: a large-scale evaluation of knowledge graph embedding models under a unified framework. *arXiv preprint arXiv:2006.13365*.

Ali, M., Berrendorf, M., Hoyt, C. T., Vermue, L., Sharifzadeh, S., Tresp, V., & Lehmann, J. (2021). Pykeen 1.0: a python library for training and evaluating knowledge graph embeddings. *Journal of Machine Learning Research*, *22*(82), 1–6.

Altmann, A., Tolosi, L., Sander, O., & Lengauer, T. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics*, *26*, 1340.

Arora, S., & Bedathur, S. (2020). On embeddings in relational databases. arXiv: 2005.06437 [cs.DB]

Bach, S. H., Broecheler, M., Huang, B., & Getoor, L. (2017). Hinge-loss markov random fields and probabilistic soft logic. *The Journal of Machine Learning Research*, *18*(1), 3846–3912.

Balazevic, I., Allen, C., & Hospedales, T. (2019). Multi-relational poincaré graph embeddings. *Neural Information Processing Systems*, *32*, 4463.

Bauer, F., & Kaltenböck, M. (2011). Linked open data: the essentials. *Edition mono/monochrom, Vienna*, *710*.

Berti-Equille, L. (2019). Learn2clean: optimizing the sequence of tasks for web data preparation. In *The world wide web conference* (pp. 2580–2586).

Bilenko, M., & Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Kdd* (p. 48).

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer New York, NY.

Bizer, C., Heath, T., & Berners-Lee, T. (2011). Linked data: the story so far. In *Semantic services, interoperability and web applications: emerging concepts* (pp. 205–227). IGI Global.

Bjorkman, A. D., Myers-Smith, I. H., Elmendorf, S. C., Normand, S., Rüger, N., Beck, P. S. A., … Forbes, B. C. (2018). Plant functional trait change across a warming tundra biome. *Nature*, *562*(7725), 57–62.

Blakely, T., Lynch, J., Simons, K., Bentley, R., & Rose, S. (2019). Reflection on modern methods: when worlds collide—prediction, machine learning and causal inference. *International Journal of Epidemiology*.

Blau, F. D., & Kahn, L. M. (2017). The gender wage gap: extent, trends, and explanations. *Journal of Economic Literature*, *55*, 789.

Blinder, A. S. (1973). Wage discrimination: reduced form and structural estimates. *The Journal of Human Resources*, *8*(4), 436–455.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. *CoRR*, *abs/1607.04606*. arXiv: 1607.04606

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135.

Bordawekar, R., & Shmueli, O. (2017). Using word embedding to enable semantic queries in relational databases. In *Proceedings of the 1st workshop on data management for end-to-end machine learning*. DEEM'17.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Neural information processing systems* (p. 2787).

Boumans, M., & Leonelli, S. (2020). From dirty data to tidy facts: clustering practices in plant phenomics and business cycle analysis. In *Data journeys in the sciences* (pp. 79–101). Springer, Cham (Switzerland).

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... Askell, A. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., & Zhang, Y. (2008). Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, *1*(1), 538–549.

Cappuzzo, R., Papotti, P., & Thirumuruganathan, S. (2020). Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Sigmod* (pp. 1335–1349).

Cerda, P., & Varoquaux, G. (2019). Encoding high-cardinality string categorical variables. *IEEE Trans. Knowl. Data Eng.*

Cerda, P., Varoquaux, G., & Kégl, B. (2018). Similarity encoding for learning with dirty categorical variables. *Machine Learning*, *107*(8), 1477.

Chen, L., Varoquaux, G., & Suchanek, F. (2022). Imputing out-of-vocabulary embeddings with love makes language models robust with little cost. In *Acl 2022-60th annual meeting of the association for computational linguistics*.

Chen, X., & Cardie, C. (2018). Unsupervised multilingual word embeddings. In *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 261–270).

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., & Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, *21*, C1–C68.

Chessell, M., Scheepers, F., Nguyen, N., van Kessel, R., & van der Starre, R. (2014). Governing and managing big data for analytics and decision makers. *IBM Redguides for Business Leaders*.

Choi, E., Xu, Z., Li, Y., Dusenberry, M. W., Flores, G., Xue, Y., & Dai, A. M. (2019). Graph convolutional transformer: learning the graphical structure of electronic health records. In *AAAI*.

Christophides, V., Efthymiou, V., & Stefanidis, K. (2015). Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, *5*, 1–122.

Ciscel, D. H., & Carroll, T. M. (1980). The determinants of executive salaries: an econometric survey. *The Review of Economics and Statistics*, 7–13.

Cochez, M., Ristoski, P., Ponzetto, S. P., & Paulheim, H. (2017). Global rdf vector space embeddings. In *International semantic web conference* (pp. 190–207). Springer.

Cropper, A., Dumancic, S., Evans, R., & Muggleton, S. H. (2022). Inductive logic programming at 30. *Machine Learning*, *111*(1), 147–172.

CrowdFlower. (2016). Data science report.

Cvetkov-Iliev, A., Allauzen, A., & Varoquaux, G. (2022a). Analytics on non-normalized data sources: more learning, rather than more cleaning. *IEEE Access*, *10*, 42420–42431.

Cvetkov-Iliev, A., Allauzen, A., & Varoquaux, G. (2022b). Relational data embeddings for feature enrichment with background information. *Accepted in Machine Learning (Springer)*.

d'Amato, C., Quatraro, N. F., & Fanizzi, N. (2021). Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In *Eighteenth extended semantic web conference - research track*.

Dasu, T., & Johnson, T. (2003). *Exploratory data mining and data cleaning*. John Wiley & Sons.

DBPedia web page. (N.d.). https://www.dbpedia.org/resources/latest-core. Accessed: 2021-11-18.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *Bert: pre-training of deep bidirectional transformers for language understanding*.

Doan, A., & Halevy, A. Y. (2005). Semantic integration research in the database community: a brief survey. *AI magazine*, *26*(1), 83–83.

Domingos, P., & Lowd, D. (2009). Markov logic: an interface layer for artificial intelligence. (Vol. 3). Morgan & Claypool Publishers.

Dong, X. L., Halevy, A., & Yu, C. (2009). Data integration with uncertainty. *The VLDB Journal*, *18*(2), 469–500.

Dong, X. L., & Rekatsinas, T. (2018). Data integration and machine learning: a natural synergy. In *International conference on management of data* (p. 1645).

Ebraheem, M., Thirumuruganathan, S., Joty, S., Ouzzani, M., & Tang, N. (2018). Distributed representations of tuples for entity resolution. *VLDB*, *11*, 1454.

Efron, B., & Morris, C. (1977). Stein's paradox in statistics. *Scientific American - SCI AMER*, *236*, 119–127.

Egami, S., Nishimura, S., & Fukuda, K. (2021). A framework for constructing and augmenting knowledge graphs using virtual space: towards analysis of daily activities. In *2021 ieee 33rd international conference on tools with artificial intelligence (ictai)* (pp. 1226–1230).

Elmagarmid, A., Ipeirotis, P., & Verykios, V. (2007). Duplicate record detection: a survey. *IEEE Trans. Knowl. Data Eng.*, *19*, 1.

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, *64*(328), 1183–1210.

Fourdrinier, D., Strawderman, W. E., & Wells, M. T. (2018). *Shrinkage estimation*. Springer.

Funk, M. J., Westreich, D., Wiesen, C., Stürmer, T., Brookhart, M. A., & Davidian, M. (2011). Doubly robust estimation of causal effects. *American journal of epidemiology*, *173*, 761.

Galkin, M., Denis, E., Wu, J., & Hamilton, W. L. (2022). Nodepiece: compositional and parameter-efficient representations of large knowledge graphs. In *International conference on learning representations*.

Gesese, G. A., Biswas, R., Alam, M., & Sack, H. (2021). A survey on knowledge graph embeddings with literals: which model links better literal-ly? *Semantic Web*, 617.

Ghazimatin, A., Balalau, O., Saha Roy, R., & Weikum, G. (2020). Prince: provider-side interpretability with counterfactual explanations in recommender systems. In *International conference on web search and data mining* (pp. 196–204).

Gneiting, T. (2011). Making and evaluating point forecasts. *Journal of the American Statistical Association*, *106*(494), 746–762.

Goel, R., Kazemi, S. M., Brubaker, M., & Poupart, P. (2019). Diachronic embedding for temporal knowledge graph completion.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press Cambridge, MA, USA.

Grohe, M. (2020). Word2vec, node2vec, graph2vec, x2vec: towards a theory of vector embeddings of structured data. In *Proceedings of the 39th acm sigmod-sigact-sigai symposium on principles of database systems*. PODS'20.

Hulsebos, M., Demiralp, C., & Groth, P. (2021). Gittables: a large-scale corpus of relational tables.

Ilyas, I. F., & Chu, X. (2019). *Data cleaning*. Morgan & Claypool.

Imbens, G. W., & Rubin, D. B. (2015). *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press.

Jaimovitch-López, G., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., & Ramírez-Quintana, M. J. (2023). Can language models automate data wrangling? *Accepted in Machine Learning (Springer)*.

James, W., & Stein, C. (1992). Estimation with quadratic loss. (pp. 443–460). Springer.

Josse, J., Prost, N., Scornet, E., & Varoquaux, G. (2020). On the consistency of supervised learning with missing values. arXiv: 1902.06931 [stat.ML]

Kandel, S., Paepcke, A., Hellerstein, J., & Heer, J. (2011). Wrangler: interactive visual specification of data transformation scripts. In *Sigchi* (p. 3363).

Kanter, J. M., & Veeramachaneni, K. (2015). Deep feature synthesis: towards automating data science endeavors. In *Ieee international conference on data science and advanced analytics (dsaa)* (pp. 1–10).

Kasai, J., Qian, K., Gurajada, S., Li, Y., & Popa, L. (2019). Low-resource deep entity resolution with transfer and active learning. In *An-*

nual meeting of the association for computational linguistics (pp. 5851–5861).

Khosravi, H., & Bina, B. (2010). A survey on statistical relational learning. In *Advances in artificial intelligence* (pp. 256–268). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kimmig, A., Bach, S., Broecheler, M., Huang, B., & Getoor, L. (2012). A short introduction to probabilistic soft logic. In *Nips workshop on probabilistic programming: foundations and applications*.

Kimmig, A., Memory, A., Miller, R. J., & Getoor, L. (2018). A collective, probabilistic approach to schema mapping using diverse noisy evidence. *IEEE Trans. Knowl. Data Eng.*, *31*, 1426.

Kochsiek, A., & Gemulla, R. (2021). Parallel training of knowledge graph embedding models: a comparison of techniques. *Proceedings of the VLDB Endowment*, *15*(3), 633–645.

Koenker, R., & Bassett Jr, G. (1978). Regression quantiles. *Econometrica*, 33.

Koller, D., Friedman, N., Dzeroski, S., Sutton, C., McCallum, A., Pfeffer, A., ... Neville, J. et al. (2007). *Introduction to statistical relational learning*. MIT press.

Koutras, C., Fragkoulis, M., Katsifodimos, A., & Lofi, C. (2020). Rema: graph embeddings-based relational schema matching.

Kramer, S., Lavrac, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In *Relational data mining* (pp. 262–286). Berlin, Heidelberg: Springer-Verlag.

Krishnan, S., Franklin, M. J., Goldberg, K., & Wu, E. (2017). Boostclean: automated error detection and repair for machine learning. *arXiv:1711.01299*.

Krishnan, S., Wang, J., Franklin, M. J., Goldberg, K., Kraska, T., Milo, T., & Wu, E. (2015). Sampleclean: fast and reliable analytics on dirty data. *IEEE Data Eng. Bull.*, *38*(3), 59–75.

Krishnan, S., Wang, J., Wu, E., Franklin, M. J., & Goldberg, K. (2016). Activeclean: interactive data cleaning for statistical modeling. *VLDB*, *9*(12), 948–959.

Kristiadi, A., Khan, M. A., Lukovnikov, D., Lehmann, J., & Fischer, A. (2019). Incorporating literals into knowledge graph embeddings. arXiv: 1802.00934 [cs.AI]

Kumar, A., Naughton, J., Patel, J. M., & Zhu, X. (2016). To join or not to join? thinking twice about joins before feature selection. In *Proceedings of the 2016 international conference on management of data* (pp. 19–34). SIGMOD '16.

Lab, M. E. D. S. (2018). Version V9. Harvard Dataverse.

Lachiche, N. (2017). Propositionalization. Springer.

Lam, H. T., Buesser, B., Min, H., Minh, T. N., Wistuba, M., Khurana, U., ... Samulowitz, H. (2021). Automated data science for relational data. In *2021 ieee 37th international conference on data engineering (icde)* (pp. 2689–2692). IEEE.

Lam, H. T., Minh, T. N., Sinn, M., Buesser, B., & Wistuba, M. (2019). Neural feature learning from relational database. arXiv: 1801.05372 [cs.AI]

Lam, H. T., Thiebaut, J.-M., Sinn, M., Chen, B., Mai, T., & Alkan, O. (2017). One button machine for automating feature engineering in relational databases. arXiv: 1706.00327 [cs.DB]

Lavrac, N., Skrlj, B., & Robnik-Sikonja, M. (2020). Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, *109*(7), 1465–1507.

Le Morvan, M., Josse, J., Moreau, T., Scornet, E., & Varoquaux, G. (2020). Neumiss networks: differentiable programming for supervised learning with missing values. *Advances in Neural Information Processing Systems*, *33*.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., ... Auer, S. et al. (2015). Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, *6*, 167.

Li, Y., Yao, L., Mao, C., Srivastava, A., Jiang, X., & Luo, Y. (2018). Early prediction of acute kidney injury in critical care setting using clinical notes. In *2018 ieee international conference on bioinformatics and biomedicine (bibm)* (pp. 683–686). IEEE.

Little, R. J., & Rubin, D. B. (1987). *Statistical analysis with missing data*. John Wiley & Sons.

Mahdisoltani, F., Biega, J., & Suchanek, F. M. (2013). YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*, Asilomar, United States.

Mann, W., Augsten, N., & Bouros, P. (2016). An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment*, *9*, 636.

McCallum, A., Bellare, K., & Pereira, F. (2005). A conditional random field for discriminatively-trained finite-state string edit distance. *UAI*, 388.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (p. 3111).

Molnar, C. (2020). *Interpretable machine learning*. Lulu.com.

Moosavi, S., Samavatian, M. H., Parthasarathy, S., & Ramnath, R. (2019). A countrywide traffic accident dataset. arXiv: 1906.05409 [cs.DB]

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., ... Raghavendra, V. (2018). Deep learning for entity matching: a design space exploration. In *Proceedings of the 2018 international conference on management of data* (pp. 19–34).

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, *8*(4), 295–318.

Nazabal, A., Williams, C. K., Colavizza, G., Smith, C. R., & Williams, A. (2020). Data engineering for data analytics: a classification of the issues, and case studies. *arXiv preprint arXiv:2004.12929*.

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on machine learning* (pp. 625–632).

Oaxaca, R. (1973). Male-female wage differentials in urban labor markets. *International Economic Review*, *14*(3), 693–709.

Paulheim, H. (2013). Exploiting linked open data as background knowledge in data mining. In *Proceedings of the 2013 international conference on data mining on linked data* (pp. 1–10). DMoLD'13. Prague, Czech Republic.

Paulheim, H., & Fümkranz, J. (2012). Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*. WIMS '12.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, É. (2011). Scikit-learn: machine learning in python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.

Pellissier Tanon, T., Weikum, G., & Suchanek, F. (2020). Yago 4: a reason-able knowledge base. In A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, ...M. Cochez (Eds.), *The semantic web* (pp. 583–596). Springer International Publishing.

Petry, F. E. (2012). *Fuzzy databases: principles and applications*. Springer Science & Business Media (Germany).

Pezeshkpour, P., Chen, L., & Singh, S. (2018). Embedding multimodal relational data for knowledge base completion. In *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 3208–3218).

Pinter, Y., Guthrie, R., & Eisenstein, J. (2017). Mimicking word embeddings using subword rnns. arXiv: 1707.06961 [cs.CL]

Portisch, J., Heist, N., & Paulheim, H. (2022). Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction – two sides of the same coin? *Semantic Web*, 1–24.

Qu, M., & Tang, J. (2019). Probabilistic logic neural networks for reasoning. In *Advances in neural information processing systems* (p. 7712).

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

Rawson, K., & Muñoz, T. (2019). Against cleaning. In *Debates in the digital humanities* (pp. 279–292). University of Minnesota Press.

Razniewski, S., Yates, A., Kassner, N., & Weikum, G. (2021). Language models as or for knowledge bases.

Rekatsinas, T., Chu, X., Ilyas, I. F., & Ré, C. (2017). Holoclean: holistic data repairs with probabilistic inference. *VLDB*, *10*(11).

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine learning*, *62*(1-2), 107–136.

Ristoski, P., & Paulheim, H. (2014). A comparison of propositionalization strategies for creating features from linked open data. (Vol. 1232).

Ristoski, P., & Paulheim, H. (2016a). Rdf2vec: rdf graph embeddings for data mining. In *Semweb*.

Ristoski, P., & Paulheim, H. (2016b). Semantic web in data mining and knowledge discovery: a comprehensive survey. *J. Web Semant.*, *36*, 1–22.

Ristoski, P., Rosati, J., Noia, T. D., Leone, R. D., & Paulheim, H. (2019). Rdf2vec: rdf graph embeddings and their applications. *Semantic Web*, *10*, 721.

Rubin, D. (1974). Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, *66*(5), 688.

Runkler, T. A. (2020). *Data analytics*. Springer.

Saeed, M. R., & Prasanna, V. K. (2018). Extracting entity-specific substructures for rdf graph embedding. In *2018 ieee international conference on information reuse and integration (iri)* (pp. 378–385).

Sarawagi, S., & Bhamidipaty, A. (2002). Interactive deduplication using active learning. *KDD*.

Sharir, O., Peleg, B., & Shoham, Y. (2020). The cost of training nlp models: a concise overview. *arXiv preprint arXiv:2004.08900*.

Silva, Y. N., Aref, W. G., & Ali, M. H. (2010). The similarity join database operator. In *International conference on data engineering (icde)* (p. 892). IEEE.

Sousa, R., Silva, S., & Pesquita, C. (2020). Evolving knowledge graph similarity for supervised learning in complex biomedical domains. *BMC Bioinformatics*, *21*.

Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., … Xu, S. (2013). Data curation at scale: the data tamer system. In *CIDR* (Vol. 2013).

Stonebraker, M., & Ilyas, I. F. (2018). Data integration: the current status and the way forward. *IEEE Data Eng. Bull.*, *41*(2), 3–9.

Suciu, D., Olteanu, D., Ré, C., & Koch, C. (2011). Probabilistic databases. *Synthesis lectures on data management*, *3*, 1–180.

Sun, Z., Deng, Z.-H., Nie, J.-Y., & Tang, J. (2019). Rotate: knowledge graph embedding by relational rotation in complex space. arXiv: 1902.10197 [cs.LG]

Sutton, C., & McCallum, A. (2012). An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, *4*(4), 267–373.

Terrizzano, I. G., Schwarz, P. M., Roth, M., & Colino, J. E. (2015). Data wrangling: the challenging yourney from the wild to the lake. In *Cidr*.

Texas Tribune. (2021). Government salaries explorer.

Thirumuruganathan, S., Tang, N., Ouzzani, M., & Doan, A. (2020). Data curation with deep learning. (pp. 277–286).

Van Buuren, S., & Groothuis-Oudshoorn, K. (2011). Mice: multivariate imputation by chained equations in r. *Journal of statistical software*, *45*, 1–67.

van den Burg, Nazábal, A., & Sutton, C. (2019). Wrangling messy CSV files by detecting row and type patterns. *Data Mining and Knowledge Discovery*, *33*(6), 1799–1820.

Vandewiele, G., Steenwinckel, B., Agozzino, T., Weyns, M., Bonte, P., Ongenae, F., & Turck, F. D. (2020a). pyRDF2Vec: Python Implementation and Extension of RDF2Vec.

Vandewiele, G., Steenwinckel, B., Bonte, P., Weyns, M., Paulheim, H., Ristoski, P., ... Ongenae, F. (2020b). Walk extraction strategies for node embeddings with rdf2vec in knowledge graphs. arXiv: 2009.04404 [cs.LG]

Verborgh, R., & De Wilde, M. (2013). *Using openrefine*. Packt Publishing.

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: a survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*, 2724.

Winkler, W. E. (1999). The state of record linkage and current research problems. In *Statistical research division, us census bureau*. Citeseer.

Wu, Y., & Wang, Z. (2018). Knowledge graph embedding with numeric attributes of entities. In *Workshop on representation learning for NLP* (p. 132).

Xiao, J. (2002). Determinants of salary growth in shenzhen, china: an analysis of formal education, on-the-job training, and adult education with a three-level model. *Economics of Education Review*, *21*, 557.

Yang, B., Yih, W.-t., He, X., Gao, J., & Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. *ICLR*.

Yin, P., Neubig, G., Yih, W., & Riedel, S. (2020). Tabert: pretraining for joint understanding of textual and tabular data. *CoRR*, *abs/2005.08314*. arXiv: 2005.08314

Yu, M., Li, G., Deng, D., & Feng, J. (2016). String similarity search and join: a survey. *Frontiers of Computer Science*, *10*, 399.

Zhang, L., Zhang, S., & Balog, K. (2019). Table2vec: neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd international acm sigir conference on research and development in information retrieval* (p. 1029).

Zhao, C., & He, Y. (2019). Auto-em: end-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The world wide web conference* (pp. 2413–2424).

Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., ... Karypis, G. (2020). Dgl-ke: training knowledge graph embeddings at scale. In *Proceedings of the 43rd international acm sigir conference on research and development in information retrieval* (pp. 739–748).

Zillow. (2021). Home value index.