UNIVERSITY OF MODENA AND REGGIO EMILIA

DOCTORATE SCHOOL IN ICT

XXXIII CYCLE

Ph.D. DISSERTATION

Big Data Management and Analytics: Open Challenges and Opportunities for Data Integration and Data Mining

Candidate: **Matteo Paganelli**

Advisor: Prof. Francesco Guerra

Director of the School: Prof. Sonia Bergamaschi

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

DOTTORATO DI RICERCA IN ICT

CICLO XXXIII

TESI DI DOTTORATO

Gestione ed Analisi dei Big Data: Sfide Aperte ed Opportunità per l'Integrazione dei Dati ed il Data Mining

Candidato: **Matteo Paganelli**

Tutor: Prof. Francesco Guerra

Coordinatore del Corso di Dottorato: Prof. Sonia Bergamaschi

Acknowledgements

First of all, I thank my family who represent my strength and who support me in all my choices. I thank my supervisor who represents for me a model of ethical and moral principles and who trained me in the scientific field. I thank Matteo Interlandi for involving me in interesting and challenging research projects. I also thank all the companions who shared this experience with me: Luca, Giovanni S., Paolo, Giovanni M., Francesco D.B., Chiara, Federica, Federico, and their tutors. Finally, I thank Angelo, Micaela and Alex.

Ringraziamenti

Ringrazio innanzitutto i miei familiari che rappresetano la mia forza e che mi supportano in tutte le mie scelte. Ringrazio il mio supervisor che rappresenta per me un modello di principi etici e morali e che mi ha formato in ambito scientifico. Ringrazio Matteo Interlandi per avermi coinvolto in progetti di ricerca interessanti e stimolanti. Ringrazio inoltre tutti i compagni che hanno condiviso con me questa esperienza: Luca, Giovanni S., Paolo, Giovanni M., Francesco D.B., Chiara, Federica, Federico, ed i relativi tutor. Infine ringrazio Angelo, Micaela ed Alex.

Abstract

In the Big Data era, the adequate management and analysis of data represents one of the most challenging activities. In the field of data management, a first reason is the constant growth in volumes, heterogeneity (in formats and quality) and dynamism of data. At the same time, numerous concerns about data security and privacy, and the traceability of the related management processes affect this discipline. On top of this practice, data analytics, conducted mainly through Machine Learning techniques, is instead subject to problems such as the availability of labeled data for their training, their complex configuration, the interpretation of their behavior, and their more mature and engineering adoption.

This thesis examines in detail the challenges and opportunities that characterize three hot topics: 1) data integration, 2) the cooperation between relational technology and Machine Learning and 3) data exploration.

Data integration is the task that combines multiple heterogeneous datasets in a unified knowledge representation, and one of its most critical aspects is its evaluation. It is currently achieved through a time-consuming try and error approach where expensive domain experts are required to manually evaluate the integration results. To solve this problem, an approach for its unsupervised evaluation is presented, which is based on the analysis of the word frequency distributions of the datasets involved in the process.

A second topic analyzed is the integration of ML capabilities within RDBMS for the implementation of inference tasks of ML pipelines. The main idea is to perform the inference phase directly in the database where the data resides, rather than extract and upload them to external and dedicated ML frameworks, with the consequent performance and safety disadvantages. To cope with this issue, MASQ (Machine Learning As Query) is presented, a library aiming at translating end-to-end ML pipelines into SQL queries to be used for solving ML inference tasks directly within relational databases.

Finally, this thesis examines the problem of data exploration and explanation, which nowadays requires the adoption of user-friendly and interactive tools for searching for relevant information within large repositories. In this context, an approach for the interactive generation of compact and informative descriptions of a dataset, able to satisfy task-specific and multi-faceted user needs, is presented.

Sommario

Nell'era dei Big Data, l'adeguata gestione ed analisi dei dati rappresenta un'attività complessa e sfidante. Nell'ambito della gestione dei dati, una prima ragione di tale criticità è legata alla costante crescita dei volumi, dell'eterogeneità (nei formati e nella qualità) e del dinamismo dei dati. Allo stesso tempo, tale disciplina è affetta da altre problematiche, quali la preservazione della sicurezza e privacy dei dati e la tracciabilità dei relativi processi di gestione. L'analisi dei dati, condotta principalmente attraverso tecniche di Machine Learning, è invece soggetta ad altre criticità, come la disponibilità di dati etichettati per il loro addestramento, la loro complessa configurazione, l'interpretazione della loro logica di funzionamento e la loro adozione più matura e ingegneristica.

Questa tesi esamina in dettaglio le sfide e le opportunità che caratterizzano tre temi di forte interesse per la comunità scientifica: 1) l'integrazione dei dati, 2) la cooperazione tra tecnologia relazionale e Machine Learning e 3) l'esplorazione dei dati.

L'integrazione dei dati è l'attività che combina molteplici sorgenti dati eterogenee in una singola rappresentazione unificata della conoscenza, ed uno dei suoi aspetti più critici è la sua valutazione. Attualmente essa è realizzata attraverso un costoso (sia in termini di tempo che di denaro) approccio iterivativo nel quale ad esperti del dominio è richiesto di valutare manualmente l'esito del processo d'integrazione. Per risolvere questo problema, in questa tesi si esamina un approccio per la sua valutazione senza ricorrere a supervisione umana, che si basa sull'analisi delle distribuzioni di frequenza delle parole delle sorgenti dati coinvolte nel processo.

Un secondo argomento analizzato è l'integrazione di funzionalità di Machine Learning (ML) all'interno di Relational Data Base Management System (RDBMS) per l'implementazione della fase di inferenza di modelli ML. L'idea principale è quella di eseguire tale attività direttamente nel database dove risiedono i dati, piuttosto che estrarli e caricarli in esterni e dedicati framework di Machine Learning, con i conseguenti svantaggi di performance e sicurezza. In risposta a questa esigenza, in questa tesi viene presentato MASQ (Machine Learning As Query), una libreria che è stata progettata per convertire pipeline ML in query SQL da utilizzare per eseguire all'interno del database l'attività di inferenza.

Infine, questa tesi esamina il problema dell'esplorazione e spiegazione dei dati, che oggigiorno richiede l'adozione di strumenti sempre più intuitivi ed interattivi per la ricerca di informazioni rilevanti all'interno di grandi archivi. In questo contesto, viene presentato un approccio per la generazione interattiva di descrizioni compatte ed informative di un dataset, in grado di soddisfare le variabili esigenze degli utenti in funzione anche del problema che intendono risolvere.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Nowadays, data acquisition and storage processes can be achieved at low cost due to significant advances in technology. Consequently, many aspects of everyday life are monitored, converted into digital form and the resulting raw data are stored for future processing. This trend has mainly been embraced in the enterprise, where the digitalization of business processes makes it possible to check their health status and identify their directions for improvement aimed at increasing business outcomes. However, obtaining these benefits is bound to the adequate use of data management and data analysis techniques, which represent the essential building blocks for an effective and profitable use of data. The first guarantees high levels of consistency, quality, updatability and security of the data, while the latter enables the extraction of insights to integrate into appropriate actions aimed at increasing the company business. In the Big Data era, these practices have to face multiple challenges. The main criticalities of data management concern the integration of multiple, high-dimensional and heterogeneous data sources, the interactive exploration of data at a glance, the preservation of data security and individual privacy, etc. On top of them, data analytics, mostly conducted through ML-based techniques, is instead subject to other problems, such as the availability of labeled data for their training, their complex configuration, the interpretation of their behavior, and their more mature and engineering adoption.

This thesis specifically analyzes 1) the problem of data integration, with related issues regarding its evaluation in business contexts, 2) the integration of ML capabilities within RDBMS, focusing on the inference of ML pipelines, and 3) the interactive exploration and explanation of datasets guided by task-specific and multi-faceted user needs.

Big Data Integration. Data integration is one of the most important data preprocessing steps before applying data analysis methods. As evidence of this, consider that its application, together with other data cleaning practices (missing values imputation, feature extraction,

etc.), occupies about 70-80 % of the time that data practitioners spend in preparing the data [47]. Driven by the intuition that data expresses greater value when integrated and linked together, it combines multiple heterogeneous datasets into a unified knowledge representation. Although it has been under study for several years, many challenges still afflict this discipline. First of all, it has to face the constant growth in volumes, heterogeneity and dynamism of data. Secondly, this task is still characterized by the active involvement of humans, which makes it an expensive and time-consuming process and limits its use in complex business scenarios. In this task, human intervention, mostly domain experts with limited technical skills, consists in verifying the correctness of the integration result, identifying conditions of systematic errors and proposing indications for improving the integration process. This work is not limited to isolated events, but is part of an iterative development cycle that generates an integration result through the application of repeated and progressive changes. This is motivating the data integration community to invest more and more efforts in drastically reducing human intervention within this workflow.

The main approaches to overcome such issue are based either on crowdsourcing techniques [153, 152, 151, 37] or on active/online learning methods [12, 17]. In the former case, many and very cheap users are exploited to quickly annotate large amounts of data but they are inexpert and pose the challenge of guaranteeing an expected quality. In the latter case, the effort required to domain experts is optimized by trying to focus it on the fewer cases that help to discriminate among data integration methods; however, training such learning methods is not trivial and there is still much room for improvement.

In Chapter 2 this topic is approached from a completely different point of view, looking for an unsupervised way to evaluate the quality of an integration process. Following this direction, a novel unsupervised measure, based on the comparison of the word frequency distributions of the involved datasets, is proposed. The expected advantages deriving from its adoption are: 1) the greater automation in the realization of effective integration results, with the consequent reduction of development times, and 2) the reduction of costs deriving from the use of expensive domain experts. This work is under review, but its preliminary experimentation is available in [107].

In-DBMS Machine Learning. Databases represent a popular, reliable and highly performing tool for managing business data. Based on a recent analysis conducted by the well-known Kaggle platform, 65% of data analysts rely on this technology for representing and managing their data. The reasons for its pervasive use are related to its proven ability to enable strict data governance, a requirement nowadays more and more pressing especially in the business environment. As evidence of this, consider for example the recent provisions on data privacy

that the European Community has promoted through the GDPR. Parallel to the consolidation of relational technology, numerous advances have been made in the field of data analytics, i.e. the set of practices focused on extracting knowledge from data. Mostly characterized by Machine Learning (ML) techniques, this discipline is transitioning from an R&D phase, for the exclusive use of specialized laboratories, to a mature phase where it can be adopted in business applications. However, this maturation process is still far from being completed. In fact, these practices are still unable to meet the strict data governance requirements of the industry: 1) most machine learning techniques are not directly interpretable by humans, limiting their debuggability and the confidence in their adoption, 2) this technology is affected by high management costs, the so-called *technical debt*, which make the production phase and the updating over time of ML models critical, and 3) ML techniques pay no attention to security and privacy concerns.

Based on these issues, various studies are being conducted to identify points of intersection between the strict data governance enabled by relational technology and the pervasive use of machine learning. Several works have already shown that by increasing the support of parallel/distributed database systems for recursive and iterative processing (adopted by gradient-descent-based approaches) and for the optimization of very large computation plans, it is possible to partially adapt these systems also for solving machine learning and data analytics tasks [132, 67, 86, 118]. This would provide significant advantages, such as the removal of expensive ETL cycles, needed for preparing database data for their injection into dedicated external analytic systems (and vice versa), the enabling of transparent scalable distributed computations integrated with recovery, traceability and debuggability features. At the same time, numerous challenges affect this problem, such as 1) understanding the most advantageous level of detail where to integrate analytics functionalities within a DBMS and 2) adapting the relational execution model, which relies on tuple-based operations, with the iterative, operator-centric and two-phase-based (i.e., train and test) ML workflows.

These works mostly focus on (1) the training aspect of ML, and (2) on optimizing specific workloads relying heavily on linear algebra. However, when referring specifically to ML analytics, there are several aspects of its workflow to consider, such as the implementation of a broader governance process aimed at monitoring ML effectiveness, performance and other aspects for its maintenance over time. This recently motivated the community to start to focus on the ML problems beyond just training. Examples are input data validation and cleaning [121, 23], model deployment [11] and technical depth remediation [22].

Following this direction, Chapter 3 proposes an approach for integrating ML capabilities within RDBMS, mainly focusing on the in-DBMS execution of ML pipeline inference tasks.

It represents a rather unexplored topic in the literature, and it is somehow surprising, in fact, consider that:

1. ML model deployment and prediction serving are responsible for 45-65 % of all running costs of an ML workflow [9];
2. models are often trained once and served many times (e.g., rendering of web pages based on users' profile, batch prediction of asset prices based on historical data), and this pattern appears quite amenable for in-DBMS execution;
3. applications where prediction serving will likely be used (e.g., websites, smart BI dashboards) are often backed by a DBMS;
4. when data already resides in a database, execution of in-DBMS predictions is a natural choice, whereas a different solution will require to pull the data out of the database. This not only is a path not always practicable, for instance, if for security reasons data cannot be moved outside the database, but it also causes performance cost and high maintenance costs.

These observations are further corroborated by the fact that commercial databases are starting to surface functionalities for expressing model predictions directly from SQL statements [94, 56, 10, 33]. Pushing the execution of predictions directly into the DBMS by translating ML pipelines end-to-end into SQL is therefore the natural next step.

To prove whether DBMSs are a good fit for ML inference, Chapter 3 presents MASQ, a library whereby trained ML pipelines are translated into standard SQL. This work is under review, but a demo and a technical report are available in [26] and [110] respectively.

Data exploration and explanation. The great availability of raw data poses also serious challenges in identifying and explaining the most relevant information for the realization of a certain task or to support business decisions. Consider for example the scenario where an analyst spends a significant amount of time searching meaningful data sources within the data lake of her organization or on the Web in order to conduct specific data-driven tasks. This problem is nowadays further aggravated by the fact that many data enthusiasts might not always be database-savvy. This has stimulated the creation of easy-to-use and interactive systems that can satisfy these requirements. A large plethora of approaches facilitates data management such as data debuggers [38, 72], data explanation systems [127, 159], exploratory search systems [129], outlier detectors [161], subgroup discovery systems [14, 61, 63] and so on.

Chapter 4 focuses on the subgroup discovery problem, a descriptive data mining field that aims at identifying interesting groups of individuals, where "interestingness is defined as distributional unusualness with respect to a certain property of interest" [14, 158]. More specifically, this chapter describes the application of a subgroup discovery technique for performing data explanation and exploration on structured datasets.

The proposed approach exploits the concept of *data description*, a compact, readable and insightful structure formed by predicates, and allows to make a (large) set of data understandable at a glance by a human user. The main idea behind this approach is to support task-specific and multi-faceted user needs, such as the summary understanding of the content of a dataset in data explanation tasks or the more targeted analysis of its specificities in outlier detection tasks. The proposed approach thus performs both data explanation since it builds *descriptions* that provide users with an explanation of the content of a dataset, and data exploration since it allows users to interactively customize the ways the *descriptions* are built supporting different modalities for describing and visualizing the data. This work has been published in [113], and a Web prototype of the developed approach is available in [111].

Chapter 2

Data integration

Data integration is the task that integrates multiple heterogeneous data sources into a single clean dataset, and represents one of the most challenging and long-lasting research topics that the scientific community is confronted with for the last 30 years [55].

There are numerous application contexts for this task, such as the integration of different product catalogs in e-commerce systems, the combination of company data with information extracted from social networks for marketing analysis, etc. Testimony of this success is the generation of a rich and booming market, which produced in 2019 a growth of 6.8%, reaching nearly $ 3.1 billion [93].

Despite all this research work, data integration is still far from being a solved problem and it is even less mature when applied in a real production context. Apart from its intrinsic complexity, one of the barriers to fully empowering data integration is the human effort needed for evaluating and tuning data integration approaches. Indeed, you need to resort to controlled datasets, built on top of a manually created ground-truth, in order to compare your approach against this gold standard and score it accordingly. This is a long and economically demanding process, it presents serious challenges for scaling it up at the huge amounts of data needed in a real business scenario, and it is not able to keep pace with the quickly evolving data sources that you find in a real context and that call for a repeated over time and/or incremental integration process.

The main approaches to overcome such issues are based either on crowdsourcing techniques [153, 152, 151, 37] or on active/online learning methods [12, 17]. In the former case, many and very cheap users are exploited to quickly annotate large amounts of data but they are inexpert and pose the challenge of guaranteeing an expected quality. In the latter case, the effort required to domain experts is optimized by trying to focus it on the fewer cases that help to discriminate among data integration methods; however, training such learning methods is not trivial and there is still much room for improvement.

The expectation is that the data integration community will focus in the coming years to fill this gap through the creation of new techniques for its (semi-) automatic evaluation, which would have a positive impact on its more widespread and mature use.

This chapter examines this issue in detail. Section 2.1 provides an introduction on the implementation and evaluation of a generic data integration pipeline, in Section 2.2 an analysis of the state of the art is presented and in Section 2.3 a possible approach for the unsupervised evaluation of a data integration approach is proposed. This work is under review, but its preliminary experimentation is available in [107].

2.1 Preliminaries

This section provides the preliminary concepts for the implementation and evaluation of a data integration task, providing the basis for understanding the main critical issues and identifying promising directions for future contributions.

2.1.1 Data integration pipeline

From a technical perspective, the data integration process is implemented through a pipelined architecture, which consists of three major steps: *schema alignment*, *entity resolution*, and *data fusion* [41], and a representation of its workflow is provided in Figure 2.1.

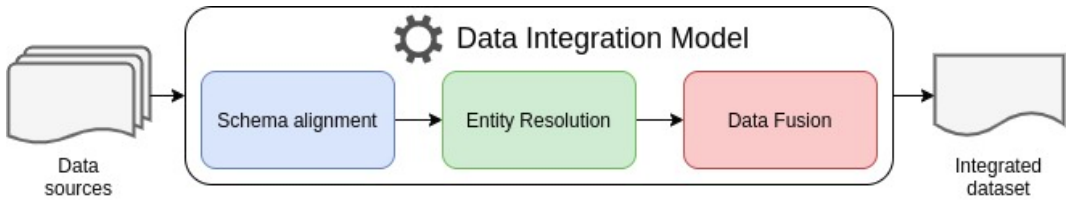


Fig. 2.1 Data integration pipeline.

Schema alignment

The first step aims to create a single and shared schema to represent the information of all the data sources to be integrated. This is necessary for several reasons: each data source in fact can 1) adopt different data format (structured, semi-structured, completely denormalized), 2) model differently the same semantic concepts (e.g. the name of a person can be modeled through a single "full name" attribute or through two distinct attributes "first name" and "last name") and 3) use different terminologies (e.g. the name of a product can be indicated both as "product" and "product name").

Entity resolution

Once a single representation of the data is provided, an entity resolution task is applied, which consists in identifying all the information that refers to the same real-world entity. The same entity can be described by the data sources in multiple variants, which mostly derive from syntactic variations (e.g. abbreviations, different order of tokens and / or different languages). It represents a complex task and is typically divided into 3 phases: *blocking*, *entity matching* and *clustering* (or transitive extension).

Blocking. A naive solution is to examine each pair of entities to verify that they describe the same real-world entity, however the dimensionality of the datasets makes this type of approach completely prohibitive in computational terms. The role of the blocking phase is to limit this search space by comparing the pairs of entities that have a higher probability of representing a match (i.e. referring to the same real-world entity). Assuming to integrate data about organizations, a possible blocking technique is to group all companies based in the same country.

Entity matching. Once the entities have been grouped according to a certain criterion, the entity matching phase identifies the pairs of entities that refer to the same real-world entity within each block. Numerous approaches have been proposed: from rule-based methods, which exploit combinations of similarity functions applied to each pair of attributes, to machine learning / deep learning models. The latter approach the problem as a binary classification task, in which a training dataset, consisting of a series of pairs of entities labeled as "match" or "non-match", is used to train a classifier that will acquire the ability to identify matching entities.

Clustering. The purpose of the clustering phase is to transitively extend the pairwise knowledge of entity correspondence, in order to group all the entities that refer to the same real-world entity. It is evident in fact that if an entity A corresponds to an entity B, which matches a third entity C, then A and C must also describe the same real-world entity. More formally, entities are assumed to refer to the same real-world entity when the matching elements form a clique [48]. The most common approach to carry out this operation is to represent the matching relationships between entities into a graph (where each node corresponds to an entity and the edges express match or non-match relationships) and calculate the connected components. Each connected component will correspond to a real-world entity.

Data fusion

The final step of the data integration pipeline is the data fusion, whose goal is to generate a single integrated representation for each real-world entity. In more detail, this operation

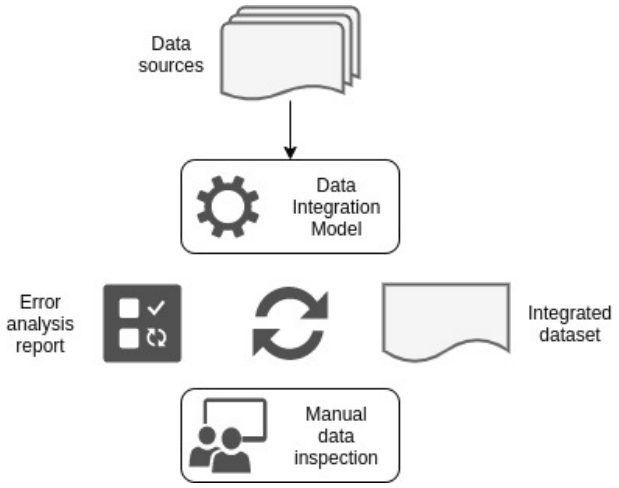


Fig. 2.2 *Try and error* approach to data integration.

applies some policy that merges the entities contained within the same cluster. In general terms, there is no standard way to accomplish this task, as it is typically driven by the user or domain knowledge. Supposing to assign to each data source a measure of reliability, the data fusion phase could, for example, simply select the entities deriving from the most relevant source. Other approaches could select the entities with the greatest information coverage, preferring those with few missing values.

2.1.2 Data integration evaluation

The data integration process is typically addressed via a *try and error* approach where a candidate integrated source is created by means of software applications and domain experts are then required to evaluate its quality and correctness. The evaluation, then, leads to modifications to the software and to the creation of a new improved release of the integrated source, that is subject to further evaluation. This process is iterated until the experts are satisfied with the result obtained, and its representation is provided in Figure 2.2.

The correctness of the result of an integration process can be measured by metrics such as accuracy, or error rate, which are used to evaluate the extent to which the integrated data source represents the original data sources. Nevertheless, the quality of the integrated source can only be assessed by an expert who has to perform the time-consuming task of manually inspecting the results of the integration process. This process becomes particularly heavy in real business scenarios, where the large amount of data makes checking all tuples infeasible. Moreover, in the case of evolving sources, where the content of the sources to integrate changes over time, and the integrated data source has to be kept aligned with them, or in case of an incremental integration process, where new sources are added over time, the evaluation

has to be performed over and over. Therefore, it is crucial to support users and domain experts with effective tools that facilitate and make the evaluation process less demanding. In Section 2.3 an approach to fill this need will be presented.

2.2 Related Work

Data Integration and Entity Resolution

Although numerous efforts have been carried out in the field of data integration, several challenges still afflict this discipline, making it a research area that is still growing rapidly. Among the most recent surveys that address this issue there are [96, 41]. They analyze the entire data integration pipeline, identifying its main challenges and opportunities. Most of the other works instead focus on its single constituent building blocks, such as data fusion [122, 40] and entity resolution [52, 103]. The latter in particular is the area in which the greatest efforts of the last 3+ decades have been concentrated. Several systems capable of scaling on large datasets and making use of advanced blocking techniques applicable also on schema-less data have been developed, such as [115, 140]. Several projects have also been financed in the industrial field, among these Magellan [74] is the main exponent. In parallel, a number of "integration functions" to discover and match the different structures that represent the same real-world entity have been proposed [154, 100, 141, 75]. Among these, rule-based [154, 141] and machine learning (ML) techniques are the most common ones. In the machine learning field, Deep Learning based techniques have recently proved particularly effective in solving this task. Some examples are DeepER [42], DeepMatcher [100], DITTO [84] and many others [166, 24]. Regardless of the use of ML or not, ER approaches require either careful manual configuration by domain experts or a large amount of labeled data [105, 83]. To cope with the first issue, methods have been proposed for the fine-tuning of parameters such as [109], but all proposals require some human supervision. Regarding to the second problem, many semi-supervised approaches in the field of active learning [12, 17] and crowd-sourcing [153, 152, 151, 37] have been introduced. The fundamental idea behind these techniques is to limit the validation intervention required by domain experts to a minimum or to resort to crowd-workers. However, these methods suffer from a poor quality control mechanism: indeed, the former approach focuses on optimizing recall while ensuring a user-specified precision level [29, 48], while crowd-based solutions are affected by uncertain labels provided by inexperienced workers [39, 122]. Other similar papers have analyzed this problem by setting up a more integrated human-machine cooperation [114, 157, 29, 65, 82].

Evaluating Data Integration and Entity Resolution

The effectiveness of ER and data integration processes is typically measured against ground truths, although some papers have analyzed its resolution in a completely unsupervised manner [160, 77]. The availability of labeled data is a problem in real scenarios, where experts have to manually assess the results obtained. This is also a problem for the evaluation of the approaches proposed by the research community since most of the techniques are evaluated against the same small number of sources (typically the benchmark made available by the Magellan tool[1]) with few hundreds of labeled data. This makes possible the development and promotion of approaches overfitting on those sources (which can have features really different from the ones in sources available in real scenarios). Only recently [87] addressed this issue, by proposing techniques for providing samples on datasets guaranteeing a fair evaluation.

2.3 Unsupervised evaluation of data integration processes

This section presents an approach for the unsupervised evaluation of an integration process. This will provide support to users and domain experts in understanding whether their integration approach needs to be revisited or tuned to be properly aligned with the targeted data sources or whether it is still working in a satisfactory way. The concept that will be used to measure the correctness of an integration result is that of *representativeness*, which evaluates how much one data source preserves the informative content of another data source. Intuitively, the more a dataset can be represented by an integrated source, the less there is a loss of information when the integrated source is considered in place of the original one; this is the *input representativeness*. Conversely, the more an integrated source can be represented by its datasets, the more it is consistent with them; this represents the *output representativeness*.

Besides being an unsupervised measure, which reduces the required human effort and is suitable also for highly iterative and/or incremental real business scenarios, the presented approach considers the integration process as a whole and evaluates its quality after the data fusion step, which is what practitioners and domain experts are confronted with in a real context. On the contrary, as analyzed in Section 2.2, most of the current literature [100, 141, 75] focuses on evaluating just the entity resolution step and, more specifically, on the entity matching by using measures like precision or recall. This type of evaluation overlooks what happens after the entity resolution step, e.g. clustering and data fusion, which can

[1] https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md

(a) D_1: the first data source.

entity id	**authors**	**title**	**venue**
freund1995a	yoav freund.	boosting a weak ...	in proceedings ...
haussler1994	haussler, d ...	rigorous learning ...	in proc. 7th ...
kearns1987	m. kearns, m. li ...	on the learnability ...	proceedings of ...
kearns1990	michael j. kearns.	the computational ...	
kearns1993b	m.j. kearns.	efficient noise-tolerant ...	in proc. 25th ...
schapire1996	r. e. schapire ...	learning sparse ...	j. of computer ...
kearns1994a	michael kearns, ...	on the learnability ...	proc. of the 26th ...
blum1994	avrim blum	weakly learning ...	in proceedings ...
freund1997a	yoav freund...	a decision-theoretic ...	journal of ...

(b) D_2: the second data source.

entity id	**authors**	**title**	**venue**
freund1995a	freund, y.	boosting a weak ...	in 'proceedings ...
haussler1994	haussler ...	rigorous learning ...	in proceedings ...
kearns1987	m. kearns ...	on the learn-ability ...	in proc. 19th stoc,
kearns1990	michael ...	the computational ...	
kearns1993b	m. kearns.	efficient noise-tolerant ...	in proceedings ...
haussler1994a	d. haussler, ...	bounds on the sample ...	machine learning,
kearns1988b	michael kearns.	thoughts on ...	(unpublished),
schapire1997	schapire, r.e ...	w.s.: boosting ...	proceedings of ...
rivest1989	r. l. rivest ...	inference of ...	in acm symposium ...

Table 2.1 Source datasets created from the "Cora Citation Matching" data.

dramatically change the final integration performance and is what they actually care about in a business context.

2.3.1 Motivating example

To better justify the need to use automatic techniques for evaluating a data integration process, an example of how this process works in practice is proposed below.

In this example the popular "Cora Citation Matching" data[2] is used to create two datasets of publications – D_1 and D_2 shown in Table 2.1 – where each publication is described by a unique identifier, authors, title, and venue. Table 2.2 shows some possible results from their integration. In particular, Table 2.2a shows I_P, the perfect integration according to the Cora ground truth. On the other hand, Table 2.2b shows I_C, a low-quality integration, obtained by just concatenating entities for the two sources. As a result, some merges are missing from it, i.e. some items from D_1 and D_2 are not recognized as referring to the same entity; for example, publication *haussler1994* is mapped to two separate entities – respectively, the second and the last entity– instead of the same one. Finally, Table 2.2c shows I_M, another

[2]https://people.cs.umass.edu/~mccallum/data.html

(a) I_P: the *P*erfect integrated dataset from D_1 and D_2, according to Cora.

id	entity id	authors	title	venue
1	*freund1995a*	yoav freund.	boosting a weak ...	in proceedings of the ...
2	*haussler1994*	haussler, d. ...	rigorous learning ...	in proc. 7th annu. ...
3	*kearns1987*	m. kearns, ...	on the learnability ...	proceedings of the ...
4	*kearns1990*	michael j. ...	the computational ...	
5	*kearns1993b*	m.j. kearns.	efficient noise-tolerant ...	in proc. 25th ...
6	*schapire1996*	r. e. schapire ...	learning sparse ...	j. of computer ...
7	*schapire1997*	schapire, r.e., ...	w.s.: boosting the ...	proceedings of ...
8	*blum1994*	avrim blum, ...	weakly learning dnf ...	in proceedings ...
9	*freund1997a*	yoav freund ...	a decision-theoretic ...	journal of ...
10	*haussler1994a*	d. haussler, ...	bounds on the sample ...	machine learning,
11	*kearns1988b*	michael kearns.	thoughts on ...	(unpublished),
12	*kearns1994a*	michael kearns, ...	on the learnability ...	proc. of the 26th ...
13	*rivest1989*	r. l. rivest and ...	inference of ...	in acm symposium ...

(b) I_C: low quality integrated dataset, obtained by just *C*oncatenating entities from D_1 and D_2.

id	entity id	authors	title	venue
1	*freund1995a*	yoav freund.	boosting a weak ...	in proceedings of the ...
2	*haussler1994*	haussler, d. ...	rigorous learning ...	in proc. 7th annu. ...
3	*kearns1987*	m. kearns, ...	on the learnability ...	proceedings of the ...
4	*kearns1990*	michael j. kearns.	the computational ...	
5	*kearns1993b*	m.j. kearns.	efficient noise-tolerant ...	in proc. 25th acm ...
6	*schapire1996*	r. e. schapire ...	learning sparse ...	j. of computer ...
7	*schapire1997*	schapire, r.e., ...	w.s.: boosting the ...	proceedings of ...
8	*blum1994*	avrim blum, ...	weakly learning ...	in proceedings ...
9	*freund1997a*	yoav freund and ...	a decision-theoretic ...	journal of computer ...
10	*haussler1994a*	d. haussler, m. ...	bounds on the ...	machine learning,
11	*kearns1988b*	michael kearns.	thoughts on ...	(unpublished),
12	*kearns1994a*	michael kearns, ...	on the learnability ...	proc. of the 26th ...
13	*rivest1989*	r. l. rivest and ...	inference of ...	in acm symposium ...
14	*kearns1987*	m. kearns, ...	on the learn-ability ...	in proc. 19th stoc,
15	*kearns1990*	michael kearns.	the computational ...	
16	*kearns1993b*	m. kearns.	efficient noise-tolerant ...	in proceedings of ...
17	*freund1995a*	freund, y.	boosting a weak ...	in 'proceedings ...
18	*haussler1994*	haussler ...	rigorous learning ...	in proceedings ...

(c) I_M: low quality integrated dataset, obtained by applying some *M*erging strategy to D_1 and D_2.

id	entity id	authors	title	venue
1	*freund1995a*	yoav freund.	boosting a weak ...	in proceedings of the ...
2	*haussler1994*	haussler, d. ...	rigorous learning ...	in proc. 7th annu. ...
3	*kearns1987*	m. kearns, ...	on the learnability ...	proceedings of the ...
4	*kearns1990*	michael j. ...	the computational ...	
5	*kearns1993b*	m.j. kearns.	efficient noise-tolerant ...	in proc. 25th ...
6	*schapire1996, schapire1997*	r. e. schapire ...	learning sparse ...	j. of computer ...
7	*blum1994, rivest1989*	avrim blum, ...	weakly learning dnf ...	in proceedings ...
8	*haussler1994a, freund1997a*	d. haussler, ...	bounds on the sample ...	machine learning,
9	*kearns1988b, kearns1994a*	michael kearns.	thoughts on ...	(unpublished),

Table 2.2 Three possible integrated datasets from sources D_1 and D_2 in Table 2.1.

low-quality integration, obtained merging each entity in D_1 with an entity in D_2. Five entities in I_M are the result of a correct integration process, since they are also in I_P. The remaining

(a) Errors in I_M

Id	Errors	Duplications	Wrong Merges
2	0	0	0
4	0	0	0
6	1	0	1
8	1	0	1
9	1	0	1
...	...	...	...
	44%	0%	44%

(b) Errors in I_C

Id	Errors	Duplications	Wrong Merges
6	0	0	0
7	0	0	0
9	0	0	0
15	1	1	0
5	1	1	0
...	...	...	...
	55%	55%	0%

Table 2.3 Error analysis conducted by a domain expert on the integrated datasets in Table 2.2.

4 entities (which were not merged in I_P) are here randomly integrated. For example, the last entity, that refers to the publication *kearns1988b*, contains also information from the pubblication *kearns1994a*, which is therefore not recognized as a distinct entity.

Let now see how a domain expert would manually assess the quality of I_C, and I_M. She/he would: 1) randomly sample (or based on "sentinel" elements defined a priori) a sufficient number of entities to check; 2) verify their correctness; and, 3) categorize erroneous outputs in a summary table, like Table 2.3, distinguishing between duplications (third column) – i.e. entities that are not merged even if they refer to the same real world entity – and wrong merges (fourth column) – i.e. entities merged even if they refer to different real world entities.

Let us start from the integrated dataset I_M in Table 2.2c whose error analysis is shown in Table 2.3a. For example, an expert may discover that the second entity has been correctly created while the sixth one contains an error since it merges two items referring to different real world entities, i.e. *schapire1996* in D_1 and *schapire1997* in D_2. Assuming the analysis is carried out for the whole dataset, the domain expert may conclude that I_M is mainly affected by incorrect matches: 44% of the entities present this type of error. Table 2.3b shows the result of a similar analysis performed on I_C. For example, I_C contains two separate entries for the entity *kearns1990* which actually refer to the same entity and therefore are a duplication. Extending the analysis to the whole dataset, the expert may discover that 55% of the entities are duplicated.

The effort required for performing the error analysis is very huge due to the large size of the datasets typically involved and the need for manually inspecting them. An accurate evaluation requires scanning the entire integrated dataset searching for duplicated and/or wrongly merged entities and a comparison with the input datasets to verify that every real-world entity has been included in the final result. Moreover, since the integrated dataset is obtained after several try and error iterations, the error analysis has to be repeated multiple times. Therefore, an automatic tool for analyzing the quality of an integration process would largely reduce the effort required for performing an integration task.

2.3.2 The approach

Motivated by this need, an approach for the automatic evaluation of a data integration process is presented below. The main idea behind this approach is the use of an unsupervised measure which quantifies the extent to which a source “represents" the content of another source. In more detail, the correspondence between the input data sources with respect to the result of the integration process (and viceversa) is considered. The proposed metric implements this intuition by considering the frequency distribution of the words in the sources and by quantifying how much a source is represented in another source in terms of how much their frequency distributions overlap.

The model

A dataset (or source) D is a collection of entities $D = \{e_1, \ldots, e_N\}$ whose attributes are defined over a common schema $R = \{A_1, \ldots, A_M\}$; each attribute represents a specific property of an entity. The integration of datasets is performed by means of an entity integration function, defined below.

Definition 1 (Entity Integration (EI) process) *EI is a process that creates an integrated dataset of entities $I = EI(\mathscr{D})$ from a collection of datasets $\mathscr{D} = \{D_1, \ldots, D_k\}$, which share a common schema R. The EI operator defines the logic for matching and merging the entities in the input dataset collection $\mathscr{D}$.*

The integration approaches are usually evaluated with controlled datasets, against a pre-existing ground truth. Accuracy, and, more frequently, due to the unbalanced datasets, recall, precision, and F-measure are the metrics used to measure the quality of the integration result.

In business environments, the absence of a ground truth imposes to define a different procedure for the evaluation. In these scenarios, the quality of the integration can be assessed through a *verification and validation process*. The verification process aims to check the formal correctness of the integrated dataset.

Definition 2 (Verified Entity Integrated Dataset) *The Entity Integrated Dataset $I = EI(\mathscr{D})$, where EI is an entity integration function applied to a collection of datasets $\mathscr{D} = \{D_1, \ldots, D_k\}$, should be:*

- ***total**: each entity of every input dataset should be represented in I, i.e., $\forall e_i \in D_k, \exists\, e_j \in I, s.t.\ e_j$ and e_i refer to the same real-world entity;*

- ***minimal**: I should not contain duplicated entities, i.e., $\forall e_i, e_j \in I, e_i$ and e_j refer to different real-world entities.*

In the *validation process*, domain experts assess the correspondence of the informative content of the integrated dataset with the one of the input sources.

In the following, a black box technique for evaluating an EI process is considered. The evaluation is based on a *representativeness function* that scores how much a dataset D_1 can be represented by a second dataset D_2 by computing the loss of information in using D_2 instead of D_1.

The adopted *representativeness function* quantify the "similarity" of the datasets by analyzing their word frequency distributions.

Definition 3 (Word frequency distribution in datasets) *Given a dataset D, let V be its vocabulary of terms. The word frequency distribution $freq_D(w) : V \rightarrow \mathbb{N}_0$ of the dataset D is a function which associates each term $w \in V$ with its frequency in D.*

The vocabulary of terms V for a dataset is generated by applying a tokenization algorithm to the concatenation of all the tuples in D. Token splitting can be considered as a solved problem [142] and a large number of techniques are available in NLP code libraries. The notion of "representativeness" between two datasets is introduced below.

Definition 4 (Dataset representativeness score) *Given two datasets D_1 and D_2, the dataset representativeness $r_{D_1 \rightarrow D_2}$ quantifies the extent to which dataset D_1 represents D_2 by measuring how much the word frequency distribution $freq_{D_1}$ approximates $freq_{D_2}$.*

In the next section, a way to measure the approximation between two word frequency distributions in the context of a data integration process is proposed.

The representativeness score should provide users with an assessment of how much datasets are represented by integrated sources by showing if there is any loss of information; vice-versa, it should quantify how much integrated sources are represented by the original datasets by showing if there is any redundancy or irrelevant content.

Scoring representativeness

When assessing the quality of the integration process, it is needed to consider the two sides of the coin, i.e. how well a source D is represented by the integration I and, vice-versa, how well the integration I is represented by a source D.

In the former case, i.e. how well a source D is represented by the integration I, if the integration process is perfect, the content of D should be completely "covered" by the content

of I. This means that the vocabulary used in D should be included in the vocabulary used in I, and the word frequency distribution of words in D should be less than or equal to the one in I. The measure of the coverage of these word frequency distributions can provide a measure of the representativity of an integration source for a dataset. This measure, identified as $r_{D\to I}$ (*input representativeness*), is defined it in equation (2.1).

Definition 5 (Input representativeness) *Given two datasets D and I, where I is the integration of D according to some EI function, let V_D be the vocabulary of D and $freq_X(w)$ be the word frequency distribution of either D or I. We define the following representativeness score:*

$$r_{D\to I} = 1 - \frac{1}{|V_D|} \sum_{w \in V_D} \frac{freq_D(w) - min(freq_D(w), freq_I(w))}{max(freq_D(w), freq_I(w))} \tag{2.1}$$

In the latter case, i.e. how well the integration I is represented by a source D, an integrated dataset should contain more entities than an input dataset, due to the contribution of other datasets. Nevertheless, excluding stop words and other very generic words, it is possible to suppose that the distribution of frequencies of words belonging to the intersection of the vocabularies of I and D is close. By measuring this closeness, it is possible to evaluate how much the dataset can represent its integration for the shared words. This measure, identified as $r_{I\to D}$ (*output representativeness*), is defined in equation (2.2).

Definition 6 (Output representativeness) *Given two datasets D and I, where I is the integration of D according to some EI function, let V_D be the vocabulary of D and $freq_X(w)$ be the word frequency distribution of either D or I. We define the following representativeness score:*

$$r_{I\to D} = 1 - \frac{1}{|V_D|} \sum_{w \in V_D} \frac{freq_I(w) - min(freq_D(w), freq_I(w))}{max(freq_D(w), freq_I(w))} \tag{2.2}$$

Note that $r_{I\to D}$ is defined over the vocabulary V_D of the dataset D and not also on the vocabulary of the integration I. Indeed, there is an intrinsic asymmetry in the integration process and it is needed to keep the focus on the dataset D, either considering how much it is represented by the integration I, i.e. $r_{D\to I}$, or how much it represents the integration I, i.e. $r_{I\to D}$, but without skewing the scores by including all the terms of V_I. Indeed, considering the whole vocabulary V_I, and not just its overlap with V_D, would just bring in all the other sources than D, whose vocabulary may differ a lot from V_D, and, as a result, these additional (and possibly unrelated) terms would mask how much D and I represent each other.

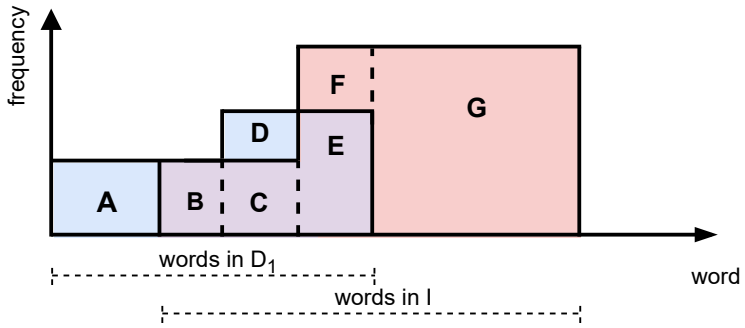


Fig. 2.3 Example of word distributions.

Example 1 *Figure 2.3 shows a simplified word frequency distribution for a dataset D_1 and its integration I. The x-axis represents the words found in the data sources and the y-axis their respective distribution. Note that, for sake of simplicity, the heights of the frequency histograms are approximated to three possible values and the actual words are not reported on the x-axis. The areas A,B,C,D,E represent the word frequency distribution for D_1 and the areas B,C,E,F,G the one of I. A and G represent words belonging only to the input dataset and integrated dataset respectively. The words in B,C,D,E,F are common to both the sources and: (1) those of B have the same frequency distribution; (2) those of C and D have frequency distribution equal to C in the integration and frequency distribution equal to $C+D$ in the input dataset; (3) those of E and F have frequency distribution equal to E in the input dataset and frequency distribution equal to $E+F$ in the integration. According to this figure, the representativeness scores are proportional to $r_{D\to I} \propto 1-(A+\frac{D}{C+D})$, and $r_{I\to D} \propto 1-(\frac{F}{E+F})$.*

Representativens as a support to verification

The representativeness score can be used to *verify* an entity integration process. In particular, the *input representativeness* score is able to measure the *totality* of the integrated dataset; the *output representativeness* score measures the *minimality* of the integrated dataset.

Let I be obtained by the integration of D_1 and D_2. The input representativeness of I with respect to the input datasets D_1 and D_2 is obtained by averaging their input representativeness scores (i.e. $r_{D_1\to I}$ and $r_{D_2\to I}$). This aggregated score provides a measure of the totality of the integration process, since the more I represents the sources D_1 and D_2, the more the entities of D_1 and D_2 are also in I. On the other side, the output representativeness of D_1 and D_2 with respect to I, obtained by averaging $r_{I\to D_1}$ and $r_{I\to D_2}$, is a measure of the minimality of the integration process. Indeed, if D_1 and D_2 have high output representativeness, it follows that I contains a few duplicated entities.

Representativeness as a support to validation

An integration process can be *validated* by plotting the representativeness scores in a two-dimensional Cartesian plane. The x-axis reports the *input representativeness* $r_{D\to I}$, i.e. the *totality*, and shows the values obtained by the datasets with respect to the integration; vice-versa, the y-axis reports the *output representativeness* $r_{I\to D}$, i.e. the *minimality*, and shows the behavior of the integration with respect to the input sources. Values closest to the point $(1,1)$ represent the best performance. The distance from (1, 1) is named as *representativeness distance* and it can provide a measure of the validation of an integration approach. Indeed, the more we depart from (1, 1), the more the correspondences between entities in the input and integrated datasets decreases. Note that only in ideal scenarios, where the entities are represented in the input datasets with the same property values, the combined representativeness score of a verified and validated integrated dataset is (1, 1). Often, data representing the same entities are not the same, due to updates, mismatches and mistakes. This affects the word frequency distributions of the corresponding datasets which will have small differences and make representativeness values departing from (1,1).

Example 2 *Figure 2.4 shows the values of the representativeness scores obtained for the I_P, I_C, and I_M integrated datasets, described in Section 2.3.1. As expected, I_P is the best integrated dataset, being the closest to the point (1,1). Note that I_C is the integration that better represents the input datasets since it has the highest values for the* input representativeness. *It is the concatenation of the input datasets, so the resulting input representativeness value is 1, since the input word frequency distribution is completely included in the integrated dataset. Nevertheless, I_C obtains the worst value of* output representativeness*, thus meaning that it contains duplicated entries. I_M shows the highest results for the* output representativeness. *I_M has been built minimizing duplicated items (all entries in the input datasets have been merged). The worst values obtained for the* input representativeness *score means that the integrated dataset does not completely represent the input datasets. This is due to the wrong entity-merges that have been introduced. Note that this analysis is consistent with the one performed in Section 2.3.1, where the error analysis tables, which are built using the ground truth, show entities erroneously merged in I_M and entities which are erroneously duplicated in I_C. Finally, note that input and output representativeness have to be jointly evaluated and the values assumed by the ground truth (I_P in the example) do not constitute an upper bound for the values that input and output representativeness can assume. In Figure 2.4, I_M and I_C are both located in the yellow area, which includes the elements with representativeness value greater than the one of the ground truth for at least one dimension. Nevertheless, even if I_M has a higher value of output representativeness, the quality of I_M (as the distance*

Scenario	input repr.	output repr.
$D_1 \rightarrow I_C$	1	0.626
$D_2 \rightarrow I_C$	1	0.538
$D_1 \rightarrow I_M$	0.712	0.953
$D_2 \rightarrow I_M$	0.633	0.918
$D_1 \rightarrow I_P$	1	0.889
$D_2 \rightarrow I_P$	0.917	0.849

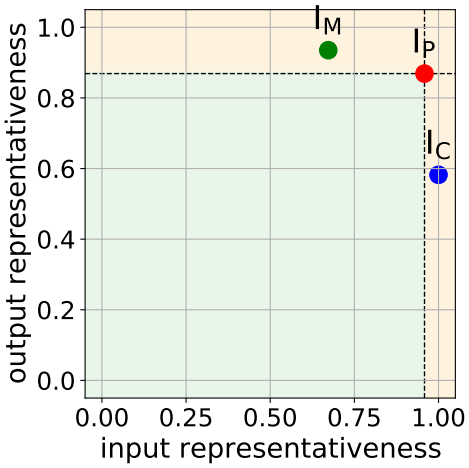


Fig. 2.4 Input and output representativeness for the sources of the motivating example.

from (1,1) shows) is worst than the one of I_P due to the lower input representativeness. The same happens for the quality of I_C, which is worst than the one of I_P due to the lower output representativeness.

2.3.3 Experimental evaluation

This section provides a quantitative (in Sections 2.3.3 to 2.3.3) and qualitative (in Sections 2.3.3 and 2.3.3) evaluation of the effectiveness of the measures proposed in the previous section. Finally, in Section 2.3.3, assesses their efficiency.

Experimental setup

For the experiments, 12 publicly available use cases (see Table 2.4) from the benchmark of the Magellan tool[3] are used. They are the main reference to evaluate entity matching approaches, and consist of two datasets describing entities and the ground truth which contains pairs of entities, one for each dataset, labelled as matching and non matching items. According to the literature [48], the entities refer to the same real-world entity when the matching elements form a clique. The third and fourth columns in Table 2.4 show the cardinalities of the input and integrated datasets.

The table also shows for each use case the ratio of shared entities (i.e., entities in the integrated dataset which are generated by merging more input entities) and unique entities (i.e., entities which come from one of the input sources only). The distribution of these

[3]https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md

Use Case	Name	Input Datasets	Integrated Dataset	Shared Entities (%)	Unique Entities (%)
U1	Textual Abt-Buy	$\|D_1\| = 949 - \|D_2\| = 920$	$\|I\| = 1174$	58.5	41.5
U2	Structured Amazon-Google	$\|D_1\| = 1171 - \|D_2\| = 1843$	$\|I\| = 2232$	32.7	67.3
U3	Structured Beer	$\|D_1\| = 237 - \|D_2\| = 233$	$\|I\| = 412$	14.1	85.9
U4	Structured Fodors-Zagats	$\|D_1\| = 89 - \|D_2\| = 238$	$\|I\| = 422$	24.9	75.1
U5	Dirty iTunes-Amazon	$\|D_1\| = 272 - \|D_2\| = 278$	$\|I\| = 450$	20.9	79.1
U6	Structured iTunes-Amazon	$\|D_1\| = 251 - \|D_2\| = 255$	$\|I\| = 410$	22.2	77.8
U7	Dirty DBLP-ACM	$\|D_1\| = 2419 - \|D_2\| = 2238$	$\|I\| = 2511$	85.5	14.5
U8	Structured DBLP-ACM	$\|D_1\| = 2406 - \|D_2\| = 2220$	$\|I\| = 2507$	84.5	15.5
U9	Dirty DBLP-GoogleScholar	$\|D_1\| = 2491 - \|D_2\| = 9877$	$\|I\| = 7959$	29.0	71.0
U10	Structured DBLP-GoogleScholar	$\|D_1\| = 2488 - \|D_2\| = 9286$	$\|I\| = 7865$	29.0	71.0
U11	Dirty Walmart-Amazon	$\|D_1\| = 1578 - \|D_2\| = 4297$	$\|I\| = 5080$	14.0	86.0
U12	Structured Walmart-Amazon	$\|D_1\| = 1524 - \|D_2\| = 4014$	$\|I\| = 4784$	14.0	86.0

Table 2.4 The use cases considered.

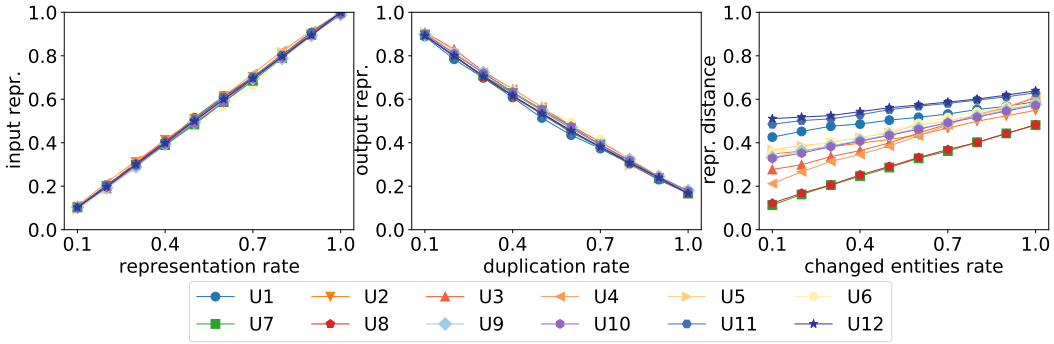


Fig. 2.5 Verification and validation of the representativeness measures.

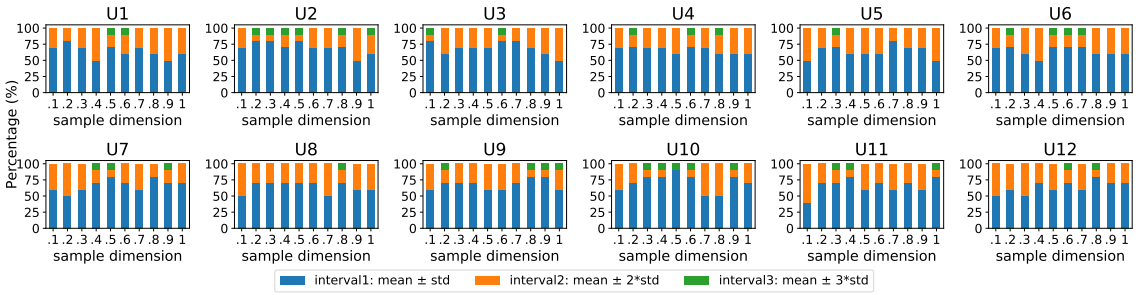
kinds of entities in the ground truth is typically unbalanced: only in U1 shared and unique entities have a similar distribution. In most of the use cases, the amount of unique entities largely overcomes the shared ones (all use cases expect U7 and U8). As will be clear in the following, this may have an impact on the performance.

All the experiments have been run on commodity hardware: a server with 4 virtual cores, 16GB of RAM, 256GB of local (SSD) storage and that runs Ubuntu version 20.04.

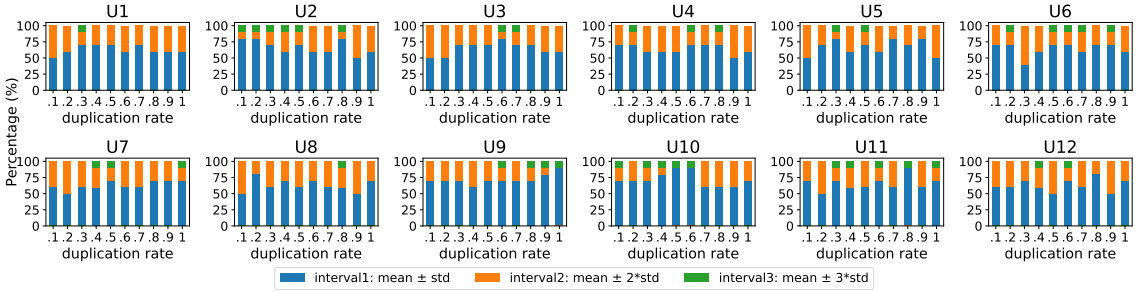
Verification and validation of an integration processes

This experiment evaluates the extent to which representativeness supports the verification and validation of an integration process: input representativeness for the totality, output representativeness for the minimality, and representativeness distance for the validation. In all the cases, the datasets of the different use cases have been deteriorated in a controlled way in order to check that the measures vary as expected to reflect these deteriorations.

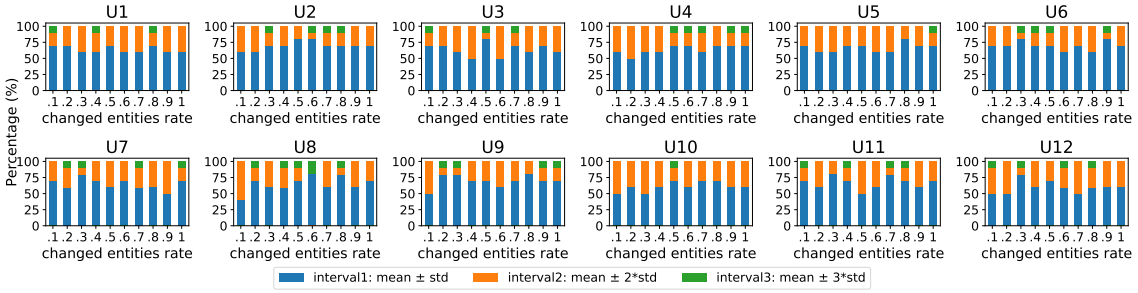
The first plot on the left of Figure 2.5 shows how the input representativeness scores vary when the integrated dataset does not include every entity of the input sources, i.e. it shows the totality of the integrated dataset. For each use case, a number of integrated datasets have



(a) Input representativeness score



(b) Output representativeness.



(c) Representativeness distance.

Fig. 2.6 Ratio of representativeness scores falling within one standard deviation (blue bar), two standard deviations (orange bar), and three standard deviations (green bar).

been created from the ground truth, by selecting an increasing percentage of ground truth entities, as specified in the x-axis. The input representativeness score computed with these reduced datasets is shown on the y-axis. As expected, the score increases with the number of entities included in the integrated dataset.

In a similar way, the second plot in the middle of Figure 2.5 shows on the x-axis the percentage of duplicated entities that have been introduced in the integrated dataset and, on the y-axis, the corresponding output representativeness score. As expected, the higher the number of duplicates, the lower the value of the output representativeness.

Finally, the third plot on the right of Figure 2.5 evaluates how well the representativeness distance measures the validation of an integration approach. The datasets have been modified

by removing and by duplicating the same percentage of entities; therefore, for example, a value of 10% on the x-axis means that 5% of the entities are duplicated and 5% are removed; the y-axis shows the corresponding value of the representativeness distance. As expected, the distance grows with the increase of duplicated and missing entities, providing an overall validation of the process. Note that the slope of the curves is less sharp than the previous ones. This is due to the joint contribution of the input and output representativeness in the definition of this measure. Indeed, an entity duplication generates both a reduction of the minimality and an increase of the totality.

Take-away: the input and output representativeness are effective implementations of the totality and minimality properties respectively, while the representativeness distance is a valuable validation measure for an integration process.

Quality of the representativeness scores

Robustness to randomness in the data. This experiment assesses to what extent randomness affects the proposed representativeness scores. To this end, for each representativeness score, the three experiments reported in the previous Section 2.3.3 have been repeated 100 times by randomly and uniformly sampling with replacement the data. In this way, it is possible to compute mean and standard deviations for each score and verify how often a given score falls in the expected range. Indeed, the more a score falls in the expected range using random and equivalent samples of the same data, the more robust is its predictions, and the less we would change our conclusions due to the observed sample.

Figure 2.6 shows the results of this experiment for each representativeness score and use case. Three ranges are considered: one standard deviation in blue; two standard deviations in orange; and, three standard deviations in green. Each bar in the stacked histograms indicates which ratio of the 100 scores falls in the blue, orange, or green interval. For example, in Figure 2.6a for use case U1 and a deterioration of 50% of the samples, i.e. 50% of the entities have been removed in this case, about 70% of the input representation scores fall in the one standard deviation range (blue bar); an additional 20% in the two standard deviations range (orange bar on top of the blue one); and, just a 10% (or less) in the three standard deviations range (tiny green bar on top of the orange one).

In the case of the input representativeness in Figure 2.6a the scores fall in the one standard deviation range in 50% to 75% of the cases, indicating a quite stable measure; almost all the other cases fall in the two standard deviations range, and just few of them in the three standard deviations range. A similar behaviour can also be observed for the output representativeness in Figure 2.6b and for the representativeness distance in Figure 2.6c.

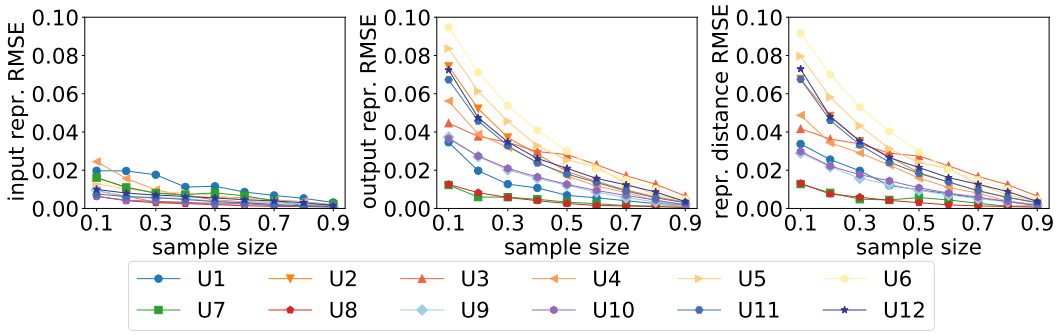


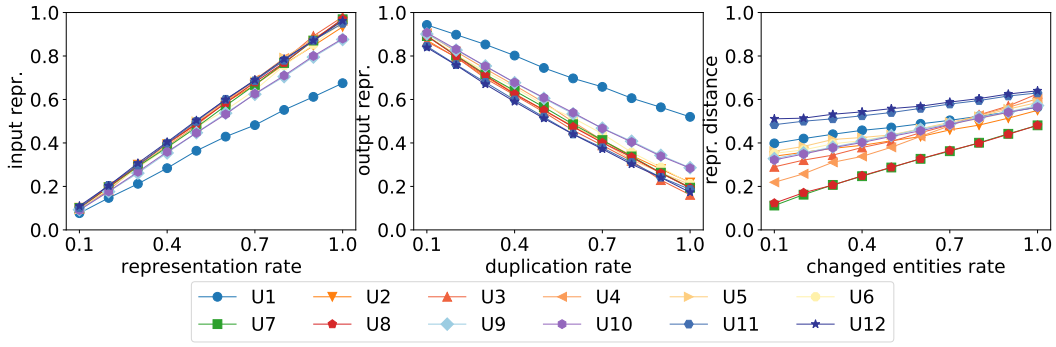
Fig. 2.7 RMSE between the mean representatives scores and the ideal score based on the ground truth.

Accuracy of the predicted score. This experiment evaluates the accuracy of the representativeness measures as the size of the considered datasets varies. This experiment provides a complementary assessment compared to previous experiments that focused on the variability of results. Samples of increasing size have been selected from the ground truth (equal to 10%, 20%, ..., 100%) and this sampling process 100 times has been repeated for each target size. For each type of representativeness score, Figure 2.7 shows the *Root Mean Square Error* (RMSE) between the score computed using the entire ground truth and the mean score computed over the samples related to a target sample dimension. The representativeness metrics do not show significant variations: only for use cases containing small datasets there are higher variations, although never greater than 0.1. This demonstrates their robustness even when significant changes in the size of the involved datasets are applied.

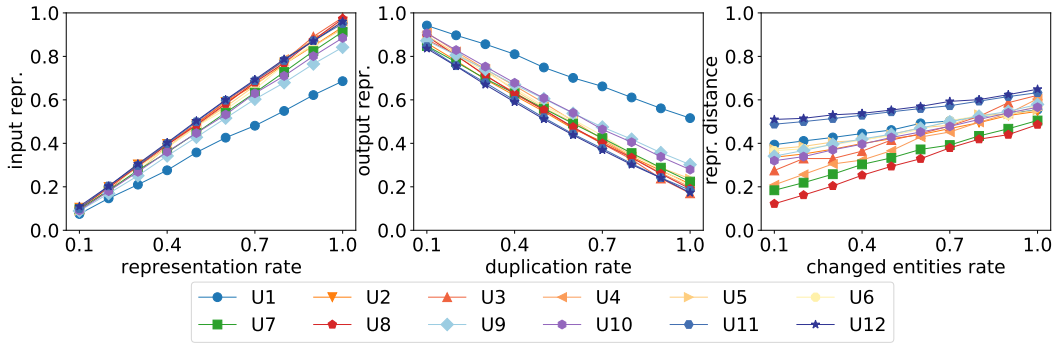
Robustness to randomness in the merging approach. This experiment analyzes how much the behaviour of the proposed measures depends on the actual merging strategy adopted to produce the integration. With the exception of small variations in the scores, the expectation is that the different data fusion techniques will have similar behaviours, otherwise, it could not be possible to reliably compare alternative integration processes.

To this end, the experiment of Section 2.3.3 has been repeated, but this time two different alternatives for merging are considered. Figure 2.8a shows the results for the first approach which randomly selects which entities to merge. Figure 2.8b shows the results for the second approach which randomly selects the values of the merged attributes. In both figures, it is possible to observe a trend which is consistent with all the previous experiments.

Take-away: the proposed representativeness scores are quite robust to different types of deterioration and randomness in the data, have a good predictive accuracy, and they are not biased by the considered data fusion techniques.



(a) Random Records.



(b) Random Values

Fig. 2.8 Verification and validation of the representativeness measures against random merging strategies.

Alternative techniques for measuring input representativeness

In Section 2.3.2 a specific way of computing representativeness based on word frequency distributions computed on the whole dataset has been proposed. These distributions can be inaccurate for describing entity similarities, computed at the tuple level. In this section, the following alternatives are considered for computing the input representativeness score: the jaccard-based similarity as a baseline, for its simplicity; the bleu-score as a reliable unsupervised measure for evaluating the quality of machine-translated text; finally, embeddings are largely used in NLP tasks to capture both syntactic and semantic similarity.

Jaccard-based representativeness. In this approach, the datasets are first tokenized and the Jaccard similarity is calculated between each pair of entities from two different datasets. We average the best Jaccard score for each entity to obtain a measure of representativeness for the entire dataset.

Bleu score-based [117] representativeness. In this approach, the same procedure of the the previous case is performed but applying the bleu score instead of the Jaccard similarity. This metric is traditionally used in the evaluation of machine translation systems and provides

a similarity score between the translation $\hat{y}$ generated by the system and a series of exact translations Y. Its formulation is given in equation 2.3. In a simplistic way, it compares the n-grams (from unigrams up to 4-grams) of the translation with respect to those of the reference translations and gives an indication of the precision of $\hat{y}$ to represent Y. With respect to a normal precision, this formula (second line of the equation) uses as numerator a count of the n-gram $Count_{clip}$ which is clipped by its maximum frequency of occurrence within the reference translations Y. The average of these modified precisions for each n-gram (with $n = 1, ..., 4$) is then calculated and a multiplicative factor BP (brevity penalty) is added to penalize short translations.

We adapt this metric to our scenario by considering the input entities as possible machine-generated translations and integrated entities as sets of correct reference translations. The presence of only one correct translation similar to the input data will allow to conclude that this entity is well represented in the integration result.

$$\begin{aligned} bleu\_score(\hat{y}, Y) &= BP * exp(\frac{1}{4} \sum_{n=1,...,4} p_n(\hat{y}, Y)) \\ p_n(\hat{y}, Y) &= \frac{\sum_{\text{n-gram} \in \hat{y}} Count_{clip}(\text{n-gram})}{\sum_{\text{n-gram} \in \hat{y}} Count(\text{n-gram})} \end{aligned} \tag{2.3}$$

Embedding-based representativeness. Three different techniques are applied (word2vec [95], fasttext [21], and glove [120]) for computing the embeddings of the tokenized entries of input and integrated entities. Pre-trained embeddings are considered and entity embeddings are computed by averaging the embeddings associated with their constituent tokens. Then, the similarity between input and integrated entities has been measured through the cosine similarity. For each input entity, the maximum value computed is considered and their mean provides the representativeness score for the entire dataset.

Alternatives for the representativeness distance. The same experiment as described in Section 2.3.3 has been performed for each of the alternative implementations of the representativeness distance. Figure 2.9 shows the difference of the representativeness distance computed with each alternative with respect to the ground truth. As expected, for all the alternatives, the difference from the ground truth increases as the deterioration of the datasets increases. Nevertheless, the measure defined in Equation 2.1 assumes the highest values in the majority of the scenarios, indicating that it better recognizes the errors in the integrated dataset.

Take-away: our measure outperforms alternative representativeness metrics based on syntactic and semantic similarities.

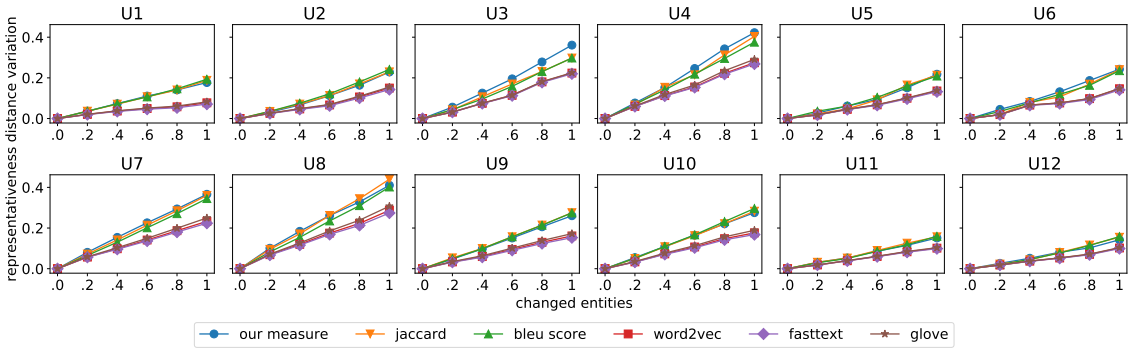
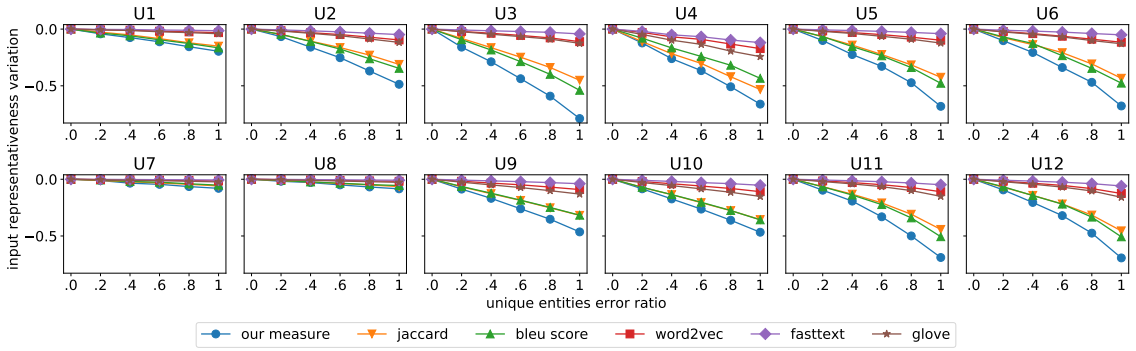


Fig. 2.9 Comparison among the measures introduced for computing the representativeness distance

Alternatives for input and output representativeness.

This experiment compares alternative measures for the input and output representativeness on the basis of how they react to possible errors in the integration process but also breaking this down by the kind of entities involved in the error, namely the *unique* and *shared* entities described in Table 2.4. In particular, two types of errors are considered: items in the input dataset which are merged even if they represent different entities and items referring to the same real-world entities which are not merged. Note that this experiment may resemble the one of Section 2.3.3 but here it operates directly on the input datasets and on the different categories of entities.

Below the "merge errors" are analyzed, and those entities in the input datasets which are not to be merged with other entities in the integration process are defined as *unique entities*. When unique entities are erroneously merged with other entities, the dimension of the integrated dataset decreases as well as its totality, since there are input entities which are not represented in the integrated dataset, i.e. the wrongly merged ones. As a consequence, this kind of error will affect the input representativeness. To evaluate the impact of these errors, different amounts of errors have been introduced in the ground truth of the use case datasets, and the difference of the input representativeness score measured with respect to the ground truth has been computed. The results of the experiments are shown in Figure 2.10a, where for each use case, a selected percentage of wrong merged entities (ranging from 0 to 100%) have been introduced. The input representativeness (independently from the approach used for its computation) decreases when the error increases in all use cases and with all the approaches. Nevertheless, it is possible to observe that the measure introduced in Equation 2.1 is able to better represent these mistakes, by showing larger variations. The other approaches perform worst: the measure based on Jaccard shows less marked variations, and the embedding-based measures do not show sensible variations in the values. Note that



(a) Variation of the input representativeness in presence of entities wrongly merged.

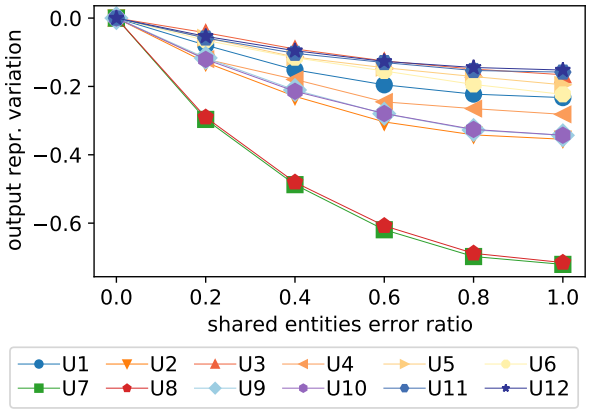
	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**	**U11**	**U12**
0.2	8.26	13.44	17.23	15.17	16	15.61	2.91	3.07	14.19	14.16	17.22	17.18
0.4	16.52	26.88	34.47	30.09	31.56	30.98	5.81	6.18	28.4	28.35	34.41	34.36
0.6	24.96	40.41	51.46	45.02	47.56	46.83	8.72	9.29	42.58	42.52	51.63	51.55
0.8	33.22	53.85	68.69	59.95	63.11	62.2	11.63	12.41	56.79	56.71	68.82	68.73
1	41.48	67.29	85.92	75.12	79.11	77.8	14.54	15.48	70.98	70.87	86.04	85.91

(b) Percentage of unique entities removed from the integrated dataset for each experiment

Fig. 2.10 Impact of wrongly merged entities on the input representativeness

Figure 2.10a shows the results on the overall dataset, not only on the portion of the dataset composed of unique entities. Moreover, the unbalanced distribution of unique entities (see Table 2.4) can introduce different amounts of wrong merges in the use cases. Table 2.10b shows the "real" impact of the perturbations introduced in the ground truth, by showing the percentage of missing unique entities for each experiment. Note that the variation in use cases U7 and U8 are less marked since the reduced number of wrong entities introduced. The plots describing U3, U11, and U12 are those with the largest variations, and this is consistent with the perturbed integrated entities.

A similar analysis has been conducted for the second issue, i.e. duplicated entities. In this case, errors in the *shared entities*, i.e. those entities obtained from merging multiple input entities, result on items in the integrated dataset which are not merged and this will affect the output representativeness. As before, a controlled deterioration of the ground truth is considered, where 20%, 40%, ...100% of errors is applied to the shared entities. Figure 2.11a shows the difference of the output representativeness score with respect to the ground truth: the more the decrease, the more errors in shared entities are detected. Note that, as before, Table 2.11b is needed to support the analysis. It shows the percentage of new entities introduced with the perturbation: U7 and U8 show the largest amount of entity introduced. This is consistent with the results in the figure that show the largest variation.



(a) Output representativeness variation in presence of entities erroneously non-merged

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12
0.2	21.55	12.46	4.13	8.77	7.33	7.07	30.63	30.55	16.15	15.04	5.45	5.33
0.4	37.56	22.49	8.01	14.69	12.67	11.46	54.4	53.45	30.66	28.2	9.78	9.62
0.6	49.66	29.48	10.92	20.85	16.89	16.1	71.64	70.68	42.73	38.88	12.93	12.86
0.8	56.64	33.29	12.62	22.99	19.78	20.24	82.04	80.93	51.84	46.68	15.02	15.07
1	59.2	35.04	14.08	24.88	22.22	23.41	85.46	84.52	55.4	49.7	15.65	15.76

(b) Percentage of duplicated entities introduced in the integrated dataset for each experiment.

Fig. 2.11 Impact of errors on shared entities on the input representativeness

Joint effect of duplication and merging errors. This experiment analyzes variations of both the input and output representativeness scores resulting from the increase of wrong merged entities and wrong duplicated entities in the datasets. Figures 2.12 and 2.13 show the results of the experiments. Green arrows show the variations on the primary component (input representativeness in the first case and output representativeness in the second one). Red arrows show the secondary component. The longer the arrow, the higher the variation in the score. The scores range from -1 to 1. They represent the difference between the value assumed by the representativeness measured in the experiment and the one in the ground truth.

In Figure 2.12, green arrows are associated with the input representativeness, red to the output representativeness. The perturbations of the datasets are generated by introducing wrong merged entities. As already shown in Figure 2.10, the input representativeness decreases with the increase of the errors. This is shown by the green arrows which become longer and tend to -1 in correspondence of the largest perturbations. It is possible to note that the red arrows have an opposite behavior: they increase when the wrong merged entities increase. This is due to the fact that increasing the number of wrong merged entities increases the minimality of the integrated dataset.

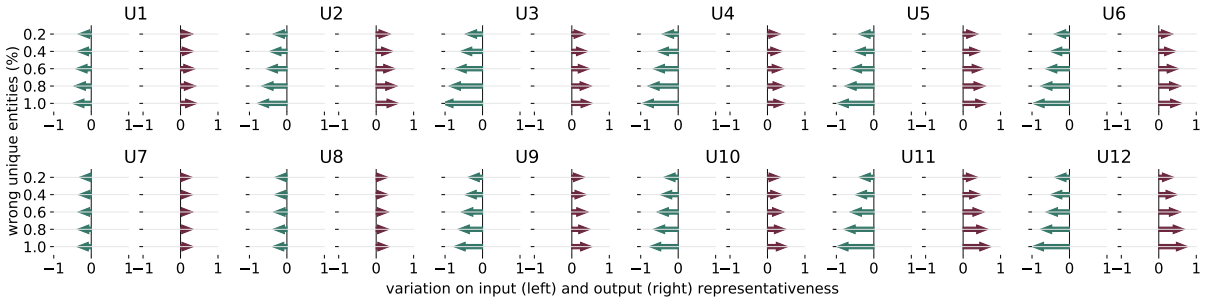


Fig. 2.12 Representativeness variations at different unique entity error rates.

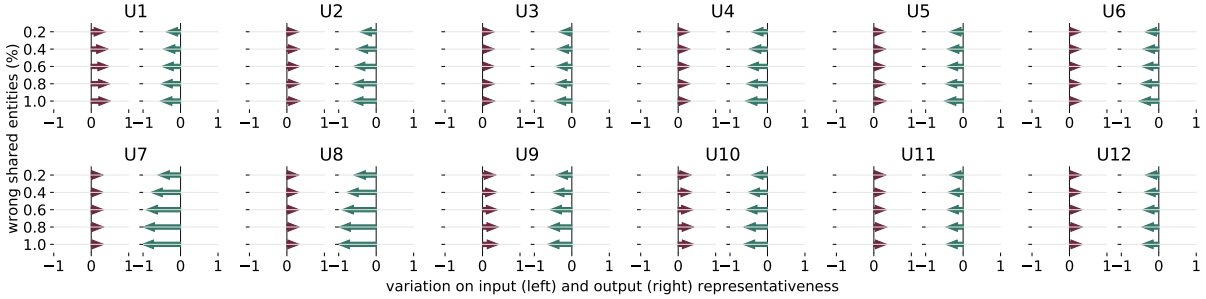


Fig. 2.13 Representativeness variations at different sharing entity error rates.

Figure 2.13 shows the results of the experiments with datasets where duplicated entities have been introduced. In this case, the green arrows show the output representativeness, which is the measure subject to the largest variations. As in Figure 2.11, in correspondence of the largest perturbations, the values assumed by the output representativeness are closer to -1. In this situation, even if less marked, it is possible to observe an increase of the input representativeness. This is due to the fact that an increase of duplicated entities in the integrated dataset increases also its ability to represent the input sources.

Take-away: examining the variations of input and output representativeness it is possible to understand the nature of the error affecting the integration task. A significant reduction of the input representativeness indicates that the ER model incorrectly merges non-matching entities (i.e. false positive items). On the other hand, an increase in false negative errors (i.e. matching entities incorrectly recognized as different entities) produces mainly a reduction of the output representativeness.

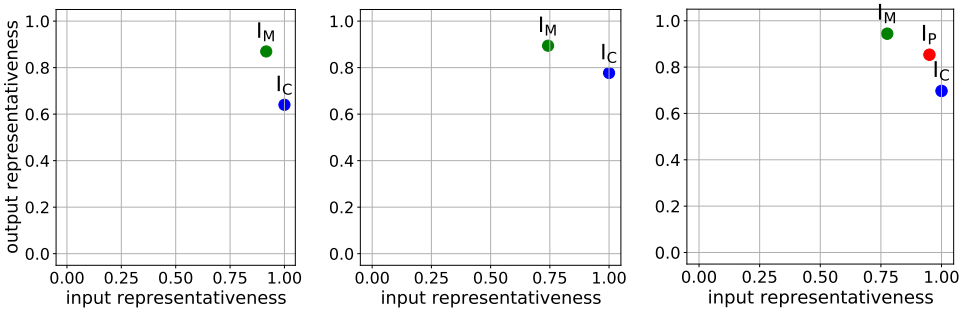
Controlled data integration scenarios

In this experiment a qualitatively assessment of the behavior of the representativeness scores in three controlled integration scenarios is analyzed. The results for U10 (the largest use case) are analyzed in detail, while the results for the other use cases are only reported.

Creating the datasets. For each use case four datasets have been generated, D_1, D_2, D_3 and D_4. D_1 has a cardinality double than D_2 which has the same cardinality as D_3. D_2 contains a subset of the entities of D_1 and, when integrated with D_1, can be used to simulate the integration of datasets describing the same entities. D_3 contains entities that are not in D_1 and can be used, when integrated with D_1, to create an integration scenario that involves datasets containing different entities. D_4 concatenates D_2 and D_3 and provides a mixed integration scenario when compared with D_1. Regarding the use case U10, the cardinality of D_1 is 2,000 entities (i.e. $|D_1| = 2000$), thus the cardinality of D_2 and D_3 is 1,000 entities. The cardinality of the vocabularies associated with the datasets is $|V_1| = 5,802, |V_2| = 3,905, |V_3| = 3,632, |V_4| = 5,939$.

Scenario 1: Datasets describing the same entities. In this scenario only the D_1 and D_2, which describe same entities, are considered. Since D_1 is a superset of D_2, it can be considered as a possible integration, called $I_M = D_1$ in Figure 2.14a. I_C is the integration obtained by a concatenation of the tuples in D_1 and D_2. In this case, the ground-truth is available and it is thus possible to compute the error rate, which is 0 for I_M, and 0.333 for I_C. The input representativeness shows that the concatenation I_C is the best integration scenario, since it does not generate any loss of information. This is clear in Figure 2.14a, where I_C assumes the maximum value of input representativeness on the x-axis. Nevertheless, concatenation introduces data duplication (D_1 is a superset of D_2) and this is the reason why in Figure I_C has an output representativeness value on the y-axis lower than I_M. The plot clearly shows that I_M is a better integration than I_C, as expected by analyzing the data sources.

Scenario 2: Datasets describing different entities. In this second scenario only the datasets D_1 and D_3, which describe different entities, are considered. As in the previous scenario, D_1 is used also as a possible integration result (I_M in Figure 2.14b). I_C is the integration obtained by the concatenation of D_1 and D_3, which does not contain duplicates in this case. In this scenario, I_C should be the best integration since all entities are included in this source. This is confirmed by the error rate, 0.5 for I_M and 0 for I_C. This is also clear by the representativeness scores, comparing the coordinates of I_C and I_M in the Figure. I_C has coordinates (1, 0.79). This means the maximum input representativeness value. I_M has coordinates (0.73,0.9). The output value is due to the low representativeness value for D_3 in I_M (0.46). Note that even if I_M does not contain the entities described in D_3 the representativeness is not zero since there is still a low number of words in D_3 which are contained in I_M anyway. The high level measured from the integration perspective is because I_M completely includes D_1 which has twice the cardinality of D_3.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 2.14 The scenarios used in the experiments

Scenario 3: Datasets describing common entities. The last scenario considers D_1 and D_4 which contain a half common and a half different entities. I_P in Figure 2.14c is generated by concatenating D_1 and D_3. This is a perfect integration since it includes all entities described by the D_1 and D_4 datasets. I_M, as in the previous scenarios, is D_1 only which, in this case, does not describe half of the entities in D_4. Finally, I_C is obtained by the concatenation of D_1 and D_4. This integration suffers from redundancy, generated by the duplicated entities of D_1 contained in D_4 and included twice in I_C. The error rates of these integrations are 0.5 for I_M and I_C, and no error rate for I_P. Figure 2.14c shows the representativeness scores and correctly reflects the datasets included in the integration, by showing the input representativeness values on the x-axis of I_P and I_M close, but not equal to 1, thus meaning that there is some loss of information in the integration. In I_C, the input representativeness values are equal to 1, since the datasets are completely represented, but the integration suffers from redundancy as shown by the lowest output representativeness value on the y-axis.

Extended evaluation. Table 2.5 summarizes the results of the experiments performed on the other datasets in the benchmark. The first column shows the names of the use cases and the cardinalities of the entities and vocabularies. The second column reports the scenarios, and the other columns outline the measures obtained by considering the I_M, I_C, and I_P integrations. The bold values are the best ones for each dataset in each scenario, i.e. the closest to the point (1,1). According to the previous discussion, the expectation is that I_M is the best integration in Scenario 1, I_C in Scenario 2, and I_P in Scenario 3. The measure performs correctly in almost all evaluations. Wrong best integrations reported in U1, U7 and U8 have all distance very close to the best one. Their mistakes are due to the sparse vocabularies (and the low cardinalities in the second dataset).

Take-away: the representativeness scores offer a fine-grained explanation on why an integration strategy can be preferred to another one.

Use case params	Sc.	I_M	I_C	I_P
U1 (|D1|=600, |D2|=300, |D3|=300, |D4|=600, |V1|=4776, |V2|=1431, |V3|=2258, |V4|=3092)	1	**(0.83, 0.83)**	(1.0, 0.71)	
	2	**(0.80, 0.89)**	(1.0, 0.73)	
	3	(0.7, 0.92)	(1.0, 0.73)	**(0.93, 0.77)**
U2 (|D1|=700, |D2|=350, |D3|=350, |D4|=700, |V1|=1664, |V2|=1139, |V3|=986, |V4|=1699)	1	**(0.91, 0.89)**	(1.0, 0.66)	
	2	(0.75, 0.91)	**(1.0, 0.76)**	
	3	(0.78, 0.95)	(1.0, 0.68)	**(0.92, 0.85)**
U3 (|D1|=50, |D2|=25, |D3|=25, |D4|=50, |V1|=208, |V2|=120, |V3|=136, |V4|=235)	1	**(0.9, 0.94)**	(1.0, 0.70)	
	2	(0.62, 0.97)	**(1.0, 0.89)**	
	3	(0.70, 0.97)	(1.0, 0.76)	**(0.92, 0.92)**
U4 (|D1|=100, |D2|=50, |D3|=50, |D4|=100, |V1|=375, |V2|=192, |V3|=192, |V4|=347)	1	**(0.98, 0.95)**	(1.0, 0.65)	
	2	(0.65, 0.96)	**(1.0, 0.88)**	
	3	(0.78, 0.98)	(1.0, 0.72)	**(0.98, 0.92)**
U5 (|D1|=90, |D2|=45, |D3|=45, |D4|=90, |V1|=697, |V2|=433, |V3|=462, |V4|=736)	1	**(0.92, 0.87)**	(1.0, 0.65)	
	2	(0.72, 0.93)	**(1.0, 0.79)**	
	3	(0.75, 0.94)	(1.0, 0.70)	**(0.95, 0.85)**
U6 (|D1|=90, |D2|=45, |D3|=45, |D4|=90, |V1|=503, |V2|=293, |V3|=335, |V4|=529)	1	**(0.95, 0.89)**	(1.0, 0.61)	
	2	(0.71, 0.93)	**(1.0, 0.79)**	
	3	(0.77, 0.95)	(1.0, 0.67)	**(0.98, 0.85)**
U7 (|D1|=2100, |D2|=1050, |D3|=365, |D4|=1415, |V1|=7359, |V2|=4854, |V3|=1790, |V4|=5460)	1	**(0.96, 0.87)**	(1.0, 0.59)	
	2	**(0.87, 0.79)**	(1.0, 0.7)	
	3	**(0.93, 0.91)**	(1.0, 0.61)	(0.97, 0.86)
U8 (|D1|=2100, |D2|=1050, |D3|=388, |D4|=1438, |V1|=7396, |V2|=4858, |V3|=1863, |V4|=5509)	1	**(0.98, 0.87)**	(1.0, 0.59)	
	2	**(0.87, 0.80)**	(1.0, 0.70)	
	3	**(0.93, 0.91)**	(1.0, 0.61)	(0.97, 0.86)
U9 (|D1|=2300, |D2|=1150, |D3|=1150, |D4|=2300, |V1|=5993, |V2|=4119, |V3|=3979, |V4|=6364)	1	**(0.92, 0.89)**	(1.0, 0.67)	
	2	(0.72, 0.91)	**(1.0, 0.8)**	
	3	(0.76, 0.95)	(1.0, 0.71)	**(0.93, 0.86)**
U11 (|D1|=700, |D2|=350, |D3|=350, |D4|=700, |V1|=2875, |V2|=2096, |V3|=1694, |V4|=3195)	1	**(0.86, 0.91)**	(1.0, 0.71)	
	2	(0.72, 0.93)	**(1.0, 0.80)**	
	3	(0.73, 0.96)	(1.0, 0.73)	**(0.88, 0.89)**
U12 (|D1|=600, |D2|=300, |D3|=300, |D4|=600, |V1|=2152, |V2|=1713, |V3|=1231, |V4|=2453)	1	**(0.88, 0.91)**	(1.0, 0.69)	
	2	(0.74, 0.96)	**(1.0, 0.79)**	
	3	(0.77, 0.95)	(1.0, 0.71)	**(0.88, 0.88)**

Table 2.5 The evaluation of the scenarios in other datasets

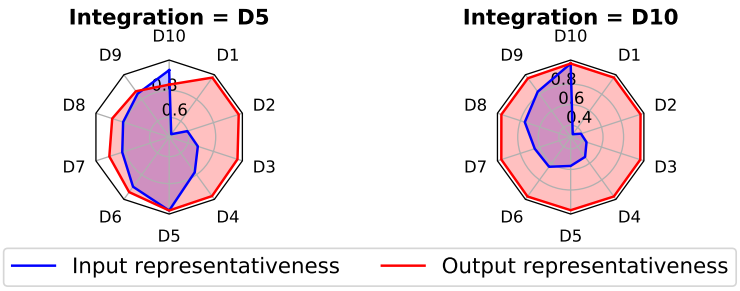


Fig. 2.15 Input and output representativeness variation on three multi-source integration scenarios.

Multi-source integration scenario

This experiment shows the behavior of the input and output representativeness in a multi-source integration scenario. Cora contains 112 clusters each one describing references to the same real-world publication. It is possible to create multiple sources from Cora by inserting the elements referring to the same publications in different datasets. It is possible to obtain 64 data sources, the largest including 112 entities and the smallest 1 entity. In this experiment the largest 10 datasets are considered, where, by construction and assuming to order the datasets by decreasing size, the entities of the first dataset are a superset of the entities included in the remaining datasets, i.e. the entities of the last dataset are included in all other datasets, while there are entities in the first dataset which are not included in any other dataset.

The goal of this experiment is to evaluate the diversified impact (in terms of input and output representativeness) that each source produces in the integration task. To carry out this type of analysis, 10 integration scenarios were created where in turn each data source was considered as the result of the integration task, and the other as the input datasets. The idea is to show that referring to the i-th integration scenario: (hp a) the dataset with the highest input representativeness is the i-th dataset, since it corresponds to the integration result; (hp b) the first $i-1$ datasets are a superset of the integration result, so they show a lower value for the input representativeness (i.e., their data is not completely covered by the integrated dataset). This simulates an integration scenario where there are entities – those belonging to the first $i-1$ sources – that have been mistakenly merged with other entities; (hp c) the output representativeness of the data sources is not affected by the choice of the integrated dataset since no duplication factor is applied to the data.

Figure 2.15 shows, for sake of simplicity, only the experiments where the fifth and the last datasets were considered as integration result. Each line represents a dataset. Blue values show the input representativeness, red values the output representativeness.

In the first scenario, where D5 is taken as the result of the integration process, datasets D1-D4 show an input representativeness value lower than D5 (hp b) which has the maximum value (hp a). The input representativeness value of datasets D5-D10 is still high since they hold data which is already in the integratd dataset. The output representativeness remains almost constant, i.e. it always assumes values higher than 0.8 which represents a high level of representativeness (hp c). Therefore the results show that there are entities in datasets D1-D4 which are incorrectly merged with other entities. This is consistent with the input datasets: there are entities in datasets D1-D4 which are not described in D5.

In the second scenario, where the smallest dataset is considered as the result of the integration process, a similar behavior as before can be observed, but this time all data sources (except D10, the result of the integration) are affected by a reduction in their input representativeness scores (hp b and hp a). No changes in the output representativeness scores are recorded (hp c).

Take-away: the developed approach is able to detect the provenance of certain error conditions. This facilitates the process of analyzing an integration result and supports the convergence of the integration development cycle.

Efficiency

This section analyzes the time required for computing the input and output representativeness scores, as defined in Equations 2.1 and 2.2, as well as the alternative input representativeness measures described in Section 2.3.3.

The time needed for computing the representativeness measures has been evaluated on integration processes involving datasets with increasing dimensionality (1K, 10K, 50K, 100K, 500K and 1M). These datasets have been obtained by applying sampling with replacement to the data contained in the use case U10 (the largest one). The experiment was repeated 5 times and Figure 4.3 shows the average times. Note that, if the time needed for loading pre-trained embeddings is not considered, all embedding-based approaches show the same performance. This is due to the same algorithm adopted to map the datasets in the vector space of the embeddings and to calculate their similarity using the cosine similarity function. The Figure also shows that there are measures that are not able to complete the task in all configurations due to memory and time limitations (a threshold of 48 hours on the execution time has been considered).

The proposed approach shows the best performance in all configurations: it takes less than 2 minutes to compute the representativeness of the largest dataset. The vectorized implementation of the cosine similarity makes the embedding-based approaches fast, but for

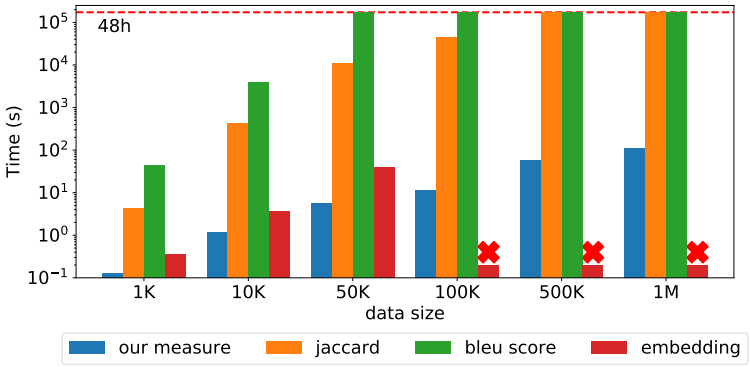


Fig. 2.16 Computing representativeness: efficiency

running on datasets larger than 100K entities it requires more memory than the one available in the considered system. The approach based on Jaccard similarity has a poor performance since it cannot be vectorized for performing the representativeness score. For this reason, the execution time grows quadratically with the size of the datasets. This is motivated by the pairwise comparison adopted to calculate the best similarity score between each input dataset entity and each entity of the integrated dataset.

Take-away: the developed approach is efficient in evaluating high dimensional data integration scenarios.

2.3.4 Conclusion and Future work

This section introduced the representativeness score, an unsupervised measure to evaluate the quality of an integration process by analyzing the word frequency distributions of the datasets involved. This measure is of extreme importance in real business scenarios, where the domain experts are needed to manually inspect the datasets to assess the results of an integration process. A deep experimental evaluation has showed that the representativeness is able to provide a means for verifying and validating an integration process.

The approach is conceived for textual datasets. Future work will deal with numeric datasets. A possible idea is to exploit functional dependencies (FD), i.e. when the values in one set of columns functionally determine the value of another column [2]. Tools, like Metanome [116], can easily retrieve the FDs existing in a dataset. The presented model based on word frequencies can be easily extended to a model based on FD frequencies.

Chapter 3

Re-purposing relational technology for advanced data analytics

Relational Data Base Management Systems (RDBMSs) have proved, over the last 4+ decades, to be a reliable and efficient tool for managing business data. Governance, security, access control, version management, data provenance tracking and performance are some of the reasons for their success.

In parallel with the consolidation of data management technologies, many efforts have also been made in the area of enterprise data analytics, i.e. the set of statistical practices, linear algebra (LA) operations and Machine Learning (ML) techniques used to extract knowledge and insight from data. In this context, particular attention has been directed toward their efficient execution in distributed environments, and as a result of this, many systems supporting large-scale LA/ML computations over-structured data have been implemented.

However, these two disciplines have followed parallel and rarely intersecting lines of development. Testimony of this fact is the use of independent systems for their implementation: database management systems on one side and dedicated analytics systems on the other side. In more detail, the management of these practices in business environment relies on multi-tier architectures, where the databases, located in the lowest tier, are typically used for data retrieval and accountability operations and feed dedicated analytics systems, on top of them, to generate insights or new knowledge from the data [148].

The adoption of such a business workflow produces high management and maintenance costs, which mainly derive from the execution of time and resource-consuming tasks for the transfer, updating and protection of the data exchanged between the systems involved in the process [118]. This therefore motivates the need for a unified system that 1) allows users to specify linear algebra and machine learning computations, 2) scales out as data increases, and 3) supports a series of features for secure and reliable governance of the entire pipeline [86].

In this chapter, in particular, a specific solution to this problem is analyzed, which consists in extending the relational technology in order to host data analytics functions. Several studies have already shown that by increasing the support of parallel / distributed database systems for recursive and iterative processing (adopted by gradient-descent-based approaches) and for the optimization of very large computation plans, it is possible to partially adapt these systems also for solving machine learning and data analytics tasks [132, 67, 86, 118].

Significant advantages could derive from their use, such as the removal of expensive ETL cycles and the enabling of transparent scalable distributed computations integrated with recovery, traceability and debuggability features.

At the same time, numerous challenges affect this problem, such as 1) understanding the most advantageous level of detail where to integrate analytics functionalities within a DBMS and 2) adapting the relational execution model, which relies on tuple-based operations, with the iterative, operator-centric and two-phase-based (i.e., train and test) ML workflows.

The remainder of this chapter analyzes this issue in detail, focusing in particular on 1) the main motivations behind this research area (Section 3.1), 2) on the analysis of the current systems available (Section 3.2), and 3) the presentation of a specific solution for performing in-DBMS ML pipeline inference tasks (Section 3.3).

This latter work is under review, but a demo and a technical report are available in [26] and [110] respectively.

3.1 Laying the foundations for the data management and data analytics integration

This section provides an analysis of the main issues affecting the current management of LA / ML analytics, laying the foundations for the use of relational technology at the rescue of these criticalities.

The success of Machine Learning is motivated by the fact that it enables the realization of tasks with a complex statistical nature that would not be directly expressible through traditional programming. The broadening adoption of machine learning in the enterprise, however, has placed new constraints on its use. The successful use of this technology in the industrial field is in fact bound to its conversion from an art and science discipline to a more mature technology, identified in [4] with the term *Enterprise Grade Machine Learning (EGML)*, and subject to the same safety, scrutiny, reliability and performance requirements that for years have been applied to previous technologies [70]. As a continuation of this view, [4] argues that an ML model should be conceived as a "data-derived software artifact",

and it should be managed in a dual perspective: on the one hand through standard software engineering techniques (*models-as-software perspective*), on the other hand through advanced data management practices (*models-as-data perspective*).

Although numerous frameworks, such as Spark [165, 92], have been introduced to enable the execution of efficient large-scale ML tasks via high-level interfaces, few efforts have been made to integrate these functionalities within full-fledged data management ecosystems. The reason for this is that there are numerous issues affecting this problem, which can be grouped into two main categories: 1) the inherent limitations of ML technology, and 2) the current use of ML differs significantly from other software artifacts.

3.1.1 ML intrinsic limits

Partially extending the categorization introduced in [136], the five main reasons for "technical debt" in ML technology, i.e. the features that increase its management costs, are: 1) system complexity, 2) data dependencies, 3) reliance on design anti-patterns, 4) complex monitoring and testing and 5) adoption of a two-step workflow.

System Complexity. The first reason for the high operating cost of an ML system is its inherent complexity, which derives from 3 main factors, identified in [136] as *entanglement*, *hidden feedback loops* and *undeclared consumers*. The term *entanglement* refers to the fact that the behavior of these systems can vary substantially even with limited and isolated changes to their inputs or configuration parameters. The second factor for their complexity is related to the knowledge acquisition mechanism adopted by them, which is part of a feedback loop that cannot be encoded in a predefined scheme and where hidden feedback may arise and produce unexpected behaviors in these systems. Finally, the absence of access control policies in the interaction between the multiple components of these systems can create unwanted producer-consumer communications (i.e., *undeclared consumers*) with consequent variations in the operating dynamics.

Data Dependencies. The management of ML systems is critical also due to different forms of data dependencies [136]. Maintaining an ML system in fact means 1) managing (implicit or explicit) unstable data dependencies over time, 2) checking dependencies deriving from underutilized features which continue to affect the behavior of the model, 3) dynamically monitor the evolution of these dependencies over time and 4) designing effectively the development of new models without applying cascade corrections to previously developed models that rely on shared data knowledge.

Reliance on design anti-patterns. Systems that incorporate ML functionalities are typically based on the joint work of different components, each of which implementing a specific ana-

lytics feature. The result of this organization is the existence of an ecosystem of independent general-purpose libraries or packages that communicate with each other only through "glue code", that is a software layer that enables the communication of these components based on the definition of standard API, the adoption of shared formats for exchanged data, and the unification of hyperparameter notations. Evidently this system management model, which does not refer to any well-known design pattern, significantly increases its maintenance cost and makes its configuration critical.

Complex monitoring and testing. Another critical aspect in managing ML systems concerns the constant monitoring and testing of their performance. As evidenced by the experiences shared by the main Big High-Tech companies, this process is typically structured in two steps [18]: an offline "health check" of the models, with the aim of verifying their ability to effectively solve the task for which they were designed, and their constant and periodic online monitoring to manage the occurrence of deviant or unexpected behaviors. In both of these phases, an assessment carried out exclusively using accuracy or label-based metrics proved to be ineffective and unreliable. In the offline phase this is motivated by the fact that it is difficult to obtain a correlation between model performance and business gains. This is caused by several reasons, including the *value performance saturation*, according to which the business profit produced by an ML model does not increase even with improvements in its performance, and phenomena such as *uncanny valley effect* and *proxy over-optimization*, according to which the too optimized functioning of an ML model can induce repulsion and disorientation in the users in appreciating its effectiveness [18]. Many issues deriving from the use of label-based evaluation metrics can also be found in the production phase. Some of them are the *incomplete and delayed feedbacks* cases, for which the verification of the correctness of the prediction generated by a model is masked or delayed when a target event occurs. With regard to the first scenario, consider the fact that a user can express interest in a product even without proceeding with its purchase (i.e., the label is not set to 1), while in the second scenario the confirmation of the correct prediction of a model can be delayed until the receipt of a review [18].

Two-step process. The typical ML workflow is organized in two distinct phases: on the one hand a training process is applied to identify the most effective transformations and models to solve the problem under examination, on the other hand the trained pipeline is used in the production phase to satisfy many inference requests. The joint management of these two phases is a complex task as they require significantly different requirements: the training phase involves the realization of long and repeated computations based on specialized hardware, while the inference phase is subject to low latency and high scalability constraints [35].

3.1.2 Unusual ML use

In addition to the intrinsic limitations of this technology, a further factor that complicates its management concerns its unusual use, which distinguishes it from other software artifacts. Today, in fact, as analyzed in [7], a trained ML model is mainly conceived as a building block to be embedded within a target application in order to provide it with inference functionalities. This execution model first of all differs from current engineering practices, according to which data science and software engineering functions are delegated to different processes or even companies. Secondly, the requirement that an ML model is deeply embedded within multiple application contexts, requires compliance with various runtime settings with related constraints, such as a reduced dependence on external libraries, the ability to manage data that doesn't fit in RAM, to scale according to the available hardware resources and to be portable on different platforms.

Comparing these limitations to the success of DBMS in the reliable and efficient management of corporate data even on a large scale, a question arises: can relational technology be used to limit these problems?

The advantages deriving from the use of this mature technology are innumerable. First of all, a centralized management of analytics within the DBMS would eliminate the high costs of extracting and transmitting data to separate and dedicated analytics systems. Secondly, it would benefit from the long-term experience of DBMS in the management and cost-based optimization of large-scale computations through a declarative interface (based on SQL) capable of masking unnecessary implementation details. [86, 67]. Furthermore, focusing on the management of ML workflows, the use of DBMS would allow to trace and scrutinize all the phases of their development cycle, promoting greater interpretability and debuggability. Even in the production phase, when an ML model is subjected to periodic updates, the use of a transactional logic would make it possible to guarantee its consistency and the tracking of its evolution over time, with consequent benefits in terms of reliability and security. [4]. Finally, the use of a well-known tool would prevent practitioners from learning another type of data processing system.

Despite these benefits, achieving this goal is a rather challenging and complex task. The main reason for this concerns the adaptation of the relational execution model with LA / ML computations. The operation of the DBMS is based on the use of operators, which adopt a tuple-oriented data updating paradigm and which are applied in succession in correspondence of diversified elaborations. A cost-based optimizer will then be responsible for identifying the optimal order of execution. On the contrary, LA / ML workflows rely on an operator / model-centric logic through which heavy vector computations iterate and transform data

several times [8]. Moreover, ML operators typically act on single-table data, unlike relational technologies which, due to the prior application of a denormalization process, store data in the form of multiple tables. This leads analysts to perform primary key-foreign key joins before learning on the integrated data layer, and therefore only the implementation of sophisticated join techniques, to reduce the cost of this data preparation step, would make the execution of ML analytics within a relational context advantageous [76]. Finally, in more general terms, it is necessary to identify the most advantageous form of integration between relational technology and data analytics functions. Numerous solutions are available, such as the use of user-defined functions (UDF), the creation of new SQL operators, the more intensified use of recursive computations already supported by some DBMS and the introduction of a new data modeling layer, however, each of them generates diversified impacts on the portability, maintainability and performance of the system [118]. A more detailed analysis of them is provided in the next section.

3.2 The state of the art on the data management and data analytics integration

This section provides an overview of systems that integrate data management capabilities with data analytics from a perspective more oriented towards the progress made by the database community.

The evolution of these systems has followed two parallel lines of development: on the one hand the relational databases have been used as platforms to host new LA features (DB-to-LA perspective), on the other hand subsequent and higher level extensions have incorporated the main low-level LA primitives (LA-to-DB perspective). A recent comparison of the systems belonging to both categories is provided in [148]. Only recently, the focus has shifted towards a more generic integration of the two disciplines. In the specific field of ML, for example, numerous efforts are being made towards a centralized and unified management, through DBMS, of the entire ML pipeline, including the critical production phase.

3.2.1 DB-to-LA perspective

The first forms of integration of RDBMS with data analytics functionality were introduced in the early 2000s, where a paradigm shift occurred in the use of DBs: from simple tools for data retrieval and accounting, to tools for the realization of predictive analytics based on statistical computations [104]. A conceptualization of this phenomenon is provided in [31], which underlines how data scientists require a Magnetic, Agile and Deep (MAD Skills) use

of DBs. These properties refer to the fact that data scientists 1) are magnetically attracted to the data, regardless of their diversified formats and / or information qualities and their prior preparation before ingestion into the DB (Magentic property), 2) apply iterative and progressive changes to the data (Agile property) and 3) analyze them through advanced and deep statistical techniques (Deep property). This new way of using DBs has prompted major vendors of DB technologies to equip their products with new operators. However, their integration was rather cumbersome and time-consuming, due to the diversified requirements (e.g. data access pattern, RAM consumption, etc.) that each (new) operator required, and for which a specific solver had to be designed [46]. These problems have therefore motivated the creation of more lasting and native forms of integration, and promoted a constant interest of the industrial sphere and the academic community towards this topic. Testimony of this fact is the realization, starting from 2010, of a great variety of systems capable of satisfying these requirements. A detailed analysis of them is proposed below.

By analyzing the solutions proposed in the literature, and partially modifying the classification provided in [118], 3 main forms of integration of relational technologies with analytics functionality have been recognized. They are distinguished according to the depth / level of integration and are:

- **Level 1**: integration realized through User-Defined Functions (UDFs)
- **Level 2**: extension of relational engines through new operators or modification of existing operators, as well as intensive use of recursive queries
- **Level 3**: re-definition of data models and application of low-level modifications to the physical layer of the DBMS

All the previous categories provide a solution to the expensive ETL process discussed in the previous section (with associated management costs), however, depending on the level of detail of integration, a series of critical issues afflict these solutions.

Level 1: UDF-based integration

The lighter forms of integration are those that perform data analytics within DBMSs using user-defined functions (UDFs). These are snippet codes written in a high-level language (eg. C ++ and Python) that allow interaction with databases using more sophisticated imperative logics than the dedicated imperative languages that DBMS already natively supports (eg Transact-SQL). While such solutions can be implemented quite easily, their main problem is performance [148]. On the one hand, in fact, they provide a high modularity in the realization of statistical computations incorporated in general-purpose building blocks and

take advantage of the greater user-friendliness of high-level languages, on the other hand they leave to the programmer the choice of the sequentiality of the operations to be performed in the DB. In other words, the DBs treat these functional blocks as black-boxes, for which no optimization can be applied by the relational engine [70, 124]. This problem is known as "impedance mismatch" and a partial solution to it is only provided in [124], which translates UDF into relational algebra expressions to be embedded in SQL queries and consequently optimizable by the relational engine.

MADlib [62]. The main exponent of this category is MADlib, which represents one of the first solutions ever created to integrate analytics functionality into DBMSs. It provides a direct implementation of MAD Skills [31], enunciated a few years earlier by the same creators of this system. This library operates on top of several DBMS (for now it only supports PostgreSQL and Greenplum) and provides analytical algorithms as user-defined functions written in C ++ and Python that are called from SQL queries and heavily uses data parallel query execution if provided by the underlying database system.

SAP HANA Predictive Analytics Library (PAL) [88]. This library represents a specific solution for the integration of LA functionalities applied exclusively to the SAP HANA main-memory database system. Like MADlib it takes advantage of UDFs written in C++, but it provides a more advanced form of integration, through which intermediate results can be stored inside relational tables and then exploited in subsequent computations.

Froid [124]. Froid converts UDFs (written in imperative language) into relational algebra expressions directly executable and optimizable by any DBMS. This allows improving the performance of UDF-based systems, which cannot be directly optimized by the relational engine.

Level 2: Relational engine extension

A deeper integration of DBMS with analytics functionality is the extension of recursive constructs, already available in some DBMS, to implement iterative statistical computations. A further integration step, on the other hand, concerns the creation of new operators who can better model such elaborations. A large variety of systems belonging to this category have been developed. In a first period, the proposed solutions tended to solve specific computations, such as the optimization of convex problems based on the gradient descent techniques, the implementation of generalized linear models and the adaptation of DB technologies to distributed patterns typical of scalable data analytics. In a second step, approaches oriented towards a unified management of generic statistical computations, through high-level (and possibly declarative) interfaces have been proposed.

Bismark [46]. Bismark was developed in 2012 with the aim of creating a unified architecture for in-RDBMS analytics, providing a solution to the costly and time-consuming creation of dedicated solvers for the in-DBMS execution of specific statistical processing. More specifically, it provides a unified interface for performing convex optimizations based on gradient descent methods.

In-DBMS Generalized Linear Models [76]. System designed after Bismark (2015), which enables the execution of machine learning computations based on generalized linear models (and solved by gradient descent techniques) in an RDBMS setting. The main motivation behind this work is the need to effectively perform a series of joins to overcome the typical denormalization of DBMS information and therefore provide a unified view of the data on which to perform statistical processing.

LA on SimSQL [86] The main motivation behind this work is the evidence that distributed RDBMS represent the ideal starting platform for the realization of a scalable linear algebra system, discouraging any impulse towards the creation of new systems from scratch. This system extends SimSQL, a parallel / distributed DBMS, through: 1) the definition of new data types and 2) the introduction of new operators to manage vectorized computations.

SQL-Opeator-Centric Approach on Hyper [118]. The solution proposed in this paper exploits a multi-layered technique for integrating data analytics features into main-memory DBMS (such as Hyper) to satisfy variable analytics workloads. Two main additions are proposed: the first is based on the SQL language extension with a new iteration construct, the second exploits user-defined lambda expressions, i.e. anonymous SQL functions that can be specified inside SQL queries, to containerize ML operator computations (e.g. Kmeans, PageRank, etc).

SQML [147]. Although limited to the execution of generalized linear models, this system implements ML computations within DBMS through pure SQL (no ad-hoc extensions are applied and no UDF / UDA are exploited), favoring a more general applicability in any relational platform. This system therefore represents a general-purpose in-database machine learning system implemented entirely with pure SQL, whose main goal is the creation of a cloud-based database-as-a-service solution. In more detail, it integrates sophisticated model partitioning techniques across multiple cluster nodes to ensure scalability. A preliminary testing of its performance on Google cloud infrastructure has shown that it is able to provide better performance than Bismark.

LevelHeaded [3]. The main focus of this work is the development of efficient join algorithms (with related cost-based optimizers) to implement LA computations in SQL queries. The intuition behind this work is in fact the observation that the simple SQL translation of LA

operations via the pairwise join algorithms in standard RDBMSs (like simSQL) is orders of magnitude slower than using a LA package.

In-DBMS Gradient Descent [132] The focus of this work is to integrate a standard DBMS with gradient descent-based computations, which are the basis of many ML algorithms. To achieve this both a new data type and a new operator for the implementation of this optimization method have been developed.

MLearn & ML2SQL [131]. Starting from the observation that there is no unified solution for tensor and gradient descent based computations accessible through an integrated declarative interface, the goal of these papers is to fill this space. Following this direction, two main products have been created: *MLearn* and *ML2SQL*. The first is a multi-platform declarative language for writing ML-based computations. The second is an SQL compiler of this language, which integrated with a specific gradient descent optimizer and an algebra based on tensors, allows the execution of ML elaborations within RDBMS.

In-DBMS Declarative Recursive Computation [67]. This work argues that by applying minimal changes to a parallel RDBMS, such as greater support for recursive constructs and its optimization for very large compute plans, it is possible to make it compatible for distributed learning computations. The paper also underlines how DBMS 1) would enable model parallelism, a parallelization technique of ML computations that does not exclusively rely on the partitioning of data between different distributed computational units, for which it is required that the data fit the respective RAM, and 2) would make the management of computer clusters more transparent due to their declarative interface.

Level 3: Physical DBMS extension

The most advanced and lowest level adaptation of traditional relational technologies to support linear algebra is the definition of a new data abstraction layer based on vectors and tensors. The main solutions developed following this philosophy are the *arrays DBMSs*, whose main exponent is *SciDB*. Even more specific solutions optimize the most modern hardware capabilities, such as FPGAs, to define a more advanced physical layer to support complex LA / ML computations. An example of these systems is *DoppioDB*.

SciDB [143]. SciDB is the result of a current of thought that believes that traditional DBMSs are not compatible with numerical computation and therefore must undergo a complete redefinition. The main concept behind this DBMS is the array. This logical unit can be shared at low level between multiple physical nodes and enables the implementation of various vector computations. At a high level, this DBMS uses a SQL-like declarative language, in order to guarantee the logical and physical independence of the system.

DoppioDB [8]. This system is an extension of MonetDB (a popular columnar DBMS) that incorporates FPGA-based machine learning operators. The main motivation behind this work is the underuse of DBMS, which could be solved by exploiting more optimized hardware, such as FPGA technologies.

3.2.2 LA-to-DB perspective

Simultaneously with the expansion of relational technologies, statistical and machine learning analytics have also spread significantly.

Some of them can be expressed using LA operations that are iterative vectorized computations, which are available to analysts in the form of multiple packages / libraries / frameworks. The basis of these tools is represented by BLAS [1], a set of specifications that defines a set of low-level routines for performing common linear algebra operations. Many numerical software applications use BLAS-compatible libraries to do linear algebra computations, such as LAPACK[2], MATLAB[3], NumPy[4] and R[5].

Similarly, ML computations can be implemented through a varied ecosystem of libraries such as Scikit-Learn[6], ML.NET[7], H2O[8], as well as frameworks more oriented to the resolution of Deep Learning tasks such as Keras[9], TensorFlow[10], PyTorch[11], etc.

Although these tools are widely used in industry, several factors prevent their use in large and distributed scenarios. The reason for this lies in their inability to manage computations on large datasets that do not fit into a single-node RAM. To satisfy this need, a new type of system, the *scalable LA system*, has been developed. They allow users to scale LA-based algorithms to distributed memory-based or disk-based data without needing to manually handle data distribution and communication and by exploiting high-level APIs. They also support storage/retrieval of data to/from disk, buffering/caching of data, and automatic logical/physical optimizations of computations (automatic rewriting of queries, pipelining, etc.). Examples of these systems are Apache Spark [165, 92], SystemML [20], Apache Mahout Samsara [130], KeystoneML [145] and many others. Although these systems have

[1] http://www.netlib.org/blas/
[2] http://www.netlib.org/lapack/
[3] https://it.mathworks.com/products/matlab.html
[4] https://numpy.org/
[5] https://www.r-project.org/
[6] https://scikit-learn.org/stable/
[7] https://dot.net/ml
[8] https://www.h2o.ai/
[9] https://keras.io/
[10] https://www.tensorflow.org/
[11] https://pytorch.org/

been designed for different purposes than DBMSs, both technologies are converging towards the creation of unified systems for the joint integration of data management features and data analytics capabilities. As evidence of this, consider for example that Spark, through the integration of relational-style *DataFrames* and *DataSets* interfaces [13] in the implementation of ML computations, is increasingly assuming the connotations of a parallel RDBMSs [67].

In line with this development direction and anticipating the evolution of some of these technologies, [86] argued that:

> Our results also suggest that if scalable linear algebra is to be added to a modern dataflow platform such as Spark, they should be added on top of the system's more structured (relational) data abstractions, rather than being constructed directly on top of the system's raw dataflow operators.

3.2.3 ML is not only training

The previous sections examined the generic ability of a DBMS to incorporate data analytics functionality. However, when referring specifically to ML analytics, there are several aspects of its workflow to consider. First of all, as already pointed out, it consists of a training phase and an inference phase, which rely on very diversified functionalities. In addition, an appropriate governance process must be employed to monitor its effectiveness, performance and other aspects for its maintenance over time. The evaluation of the integration of ML functions within the DBMS must therefore consider all these constituent components [4]. In line with this consideration, [121] argues that:

> However, developing reliable, robust, and understandable ML models requires much more than a good training algorithm. Specifically, it is necessary to build the model using high-quality training data. Moreover, this training data needs to be translated into a set of features that can expose the underlying signal to the training algorithm. And finally, the data fed to the model at serving time must be similar in distribution (and in features) to the training data, otherwise the model's accuracy will decrease. Ensuring that each of these steps is done in a consistent manner becomes even more challenging in a setting where new training data arrives continuously and accordingly triggers the training and deployment of updated models.

This recently motivated the community to start to focus on the ML problems beyond just training. Examples are input data validation and cleaning [121, 23], model deployment [11] and technical depth remediation [22].

Following this perspective, several systems and libraries have been developed. One of the pioneering works in this field was *Clipper* [35], a general-purpose low-latency prediction serving system whose goal is to provide an efficient and unified interface for the realization of ML inference tasks through multiple DataFlow frameworks (like Spark, TensorFlow, Scikit-Learn). Other similar systems are Rafiki [155], Velox [34] and TensorFlow Serving [16] (although the latter is specifically designed to extend TensorFlow). A more sophisticated evolution of such systems is represented by *PRETZEL* [79], a prediction serving system for ML pipelines introducing a novel white box architecture enabling both end-to-end and multi-model optimizations. Unlike previous jobs, this system accesses the individual components of the pipeline to be served (which is considered a whitebox) in order to optimize its execution at serving time. Another system focused on accomplishing ML pipeline inference tasks is *Raven* [70], a framework under development that analyzes the problem from a completely new perspective compared to previous works. Its main focus is in fact to embed this ML pipeline inference capability directly into the DBMS by co-optimizing predictive pipelines and SQL queries.

A further step towards the full end-to-end management of the ML workflow is offered by *Mldp* [5] that provides a flexible data model for managing different kinds of data, an environment where to perform experiments that guarantees re-producibility and it is integrated with the most important ML frameworks.

3.3 DBMS as ML Prediction Serving System

This section, pursuing the direction provided by the most recent works in the integration of relational technology and Machine Learning, analyzes the in-DBMS prediction serving of end-to-end ML pipelines (i.e., pipelines composed of featurizers and ML models). It represents a rather unexplored topic in the literature, and this is somehow surprising:

1. in practice, ML models are seldomly deployed alone, whereas data featurizers are often required to transform data into the format that is understandable by ML models (e.g., in [123] pipelines can have up to hundreds of operators);
2. models are often trained once and served many times (e.g., rendering of web pages based on users' profile, batch prediction of asset prices based on historical data), and this pattern appears quite amenable for in-DBMS execution;
3. applications where prediction serving will likely be used (e.g., websites, smart BI dashboards) are often backed by a DBMS;

4. the top used operators in open source and practical data science over tabular data are not compute-heavy neural networks, but rather memory-intensive operations (such as one-hot encoding or tree ensemble methods [123]) which should benefit from in-DBMS execution;

5. when data already resides in a database, execution of in-DBMS predictions is a natural choice, whereas a different solution will require to pull the data out of the database. This not only is a path not always practicable, for instance, if for security reasons data cannot be moved outside the database, but it also causes performance cost.

These observations are further corroborated by the fact that commercial databases are starting to surface functionalities for expressing model predictions directly from SQL statements [94, 56, 10, 33]. Pushing the execution of predictions directly into the DBMS by translating ML pipelines end-to-end into SQL is therefore the natural next step.

To prove whether DBMSs are a good fit for ML inference, in this section is presented MASQ, a library whereby trained ML pipelines are translated into standard SQL. MASQ accepts as input pipelines trained in Scikit-Learn (Sklearn) [119] or ML.NET [7], and it supports a dozen among featurizers and models, purposely selected among the frequently used operators in open source projects and in the industry domain [123].

Unlike all the works described in Section 3.2, which mostly focus on (1) the training aspect of ML, and (2) on optimizing specific workloads relying heavily on linear algebra, the focus behind the creation of MASQ is to *understand whether off the shelf DBMSs are a good fit for serving ML predictive pipelines*.

Tidypredict [1] is the closest work to MASQ, although it works only in R, and for a small set of models (linear regression, generalized linear model, random forest, and decision tree). Another similar work is Raven [70], which co-optimizes predictive pipelines and SQL queries. Among the optimizations, Raven is able to generate SQL queries from single ML operators. MASQ instead end-to-end translates predictive pipelines, so it's the database optimizer's job to figure out how to best execute predictive pipelines and SQL queries.

To prove the effectiveness of MASQ only traditional ML pipelines are considered and a comparison of the DBMS execution performance against popular, single node, ML libraries, such as Sklearn [119] and ML.NET [7], is performed. Other alternative libraries include H2O [60], Weka [64], and Spark's MLlib [91] (for scale-out training). All these libraries provide a comprehensive set of ML models and algorithms, but they do not focus neither on "Enterprise-grade" features, nor on data management (in practice the assumption is that data resides on flat CSV files).

In the rest of this section this library is evaluated over 10 representative pipelines, spanning (1) binary classification, regression, and multiclass classification tasks; (2) a diverse set of

models (linear models, tree-ensembles) and featurizers (one-hot encoding, normalizers, etc.); and (3) over 7 different datasets: from large scale (tens of millions of records), to small ones with only few hundreds instances. The goal of these experiments is to 1) evaluate the inference performance of Sklearn and ML.NET pipelines against their SQL implementation (generated by MASQ) executed over MySQL and SQL Server, and 2) asses the performance of the library when using both different input / output modalities (flat CSV file or database), inference settings (batch of different sizes), and optimization and implementation strategies (e.g., w/ and w/o indexes, operator fusion, etc.). On the basis of the literature analysis carried out the Section 3.2, this experimentation represents the first attempt oriented to the empirical evaluation that ML pipelines can be run end-to-end in plain SQL, and that this can be achieved even for high-dimensional ML models and featurizers going beyond DBMSs limits.

3.3.1 Background: ML Workflow

Figure 3.1 depicts a typical ML workflow. Starting with some *input data*, a *data preparation* step is used for sanity checks, data validation, data cleaning (e.g., completion of missing values through imputers [133]), feature generation [135] and selection. Data preparation is commonly performed through a set of *data featurizers*. The *featurized data*, output of the data preparation step, is then passed to the *training* step, where a *learning* algorithm is used to fit a ML model through an iterative process. Once the model is trained, it can be represented as a *prediction function* transforming input features into a prediction score (e.g., 1 or 0 for binary classification). Finally, the trained ML model along with the data preparation operators constitute the *ML predictive pipeline* which is then *deployed* for *serving* prediction queries [121]. Wrapping data preparation and trained ML models into a unique artifact is common practice in ML systems [7] in order to avoid prediction skew [167]. At serving time, the new input data is pre-processed and featurized (using the same operators, and ideally code, as during training) and fed into the prediction function of the trained ML model for rendering the final score.[12]

The focus of this work is to study whether the prediction serving process can be pushed down and directly executed on DBMSs. The training process is kept as in the typical ML workflow and is not the focus of this work. Rather, once a model is trained, MASQ is used to generate SQL queries that perform the same data preparation and prediction logic as the original predictive pipeline. Standard SQL is purposely considered such that it is possible

[12]Note that this is an oversimplification of actual ML workflows, and it does not cover for example hyper-parameter tuning, model selection, or cases in which ML models are used as featurizers. It is however a fair summary of common use-cases.

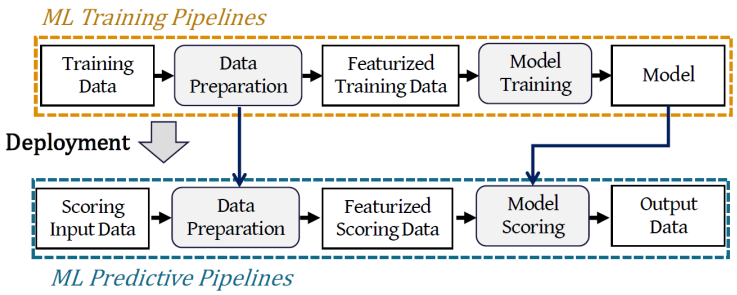


Fig. 3.1 A typical ML workflow. Rectangles are used to identify data artifacts (e.g., input data, or trained models); ellipses determine computations (e.g., data preparation and serving).

to (1) target different DBMSs; and (2) allow the optimizer to properly generate efficient end-to-end plans. Finally, the focus of this work is on models learned over relational data. Therefore, only pipelines composed of "traditional" ML operators are considered (i.e., no deep neural networks). Traditional methods are the state-of-the-art over structured data [36], and it is still the more widely-used type of ML [123]. Nevertheless, some tests related to the performance of shallow neural networks will be presented in Section 3.3.3.

3.3.2 The MASQ Library

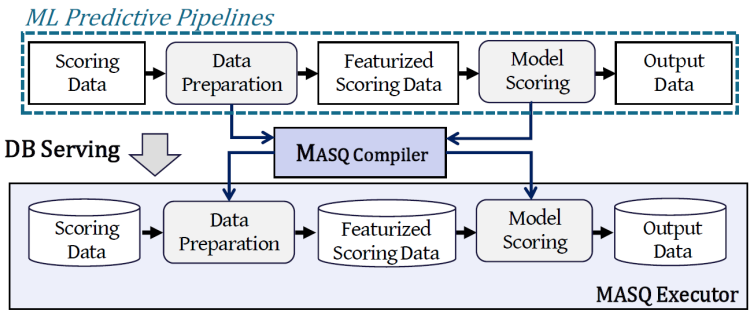


Fig. 3.2 MASQ applied to a ML predictive pipeline.

The MASQ library is built off two main components (Figure 3.2). The *Compiler* is responsible for the transformation of the predictive pipelines into SQL queries; the *Executor* instead connects and run the queries on the DBMS holding the data.

The Compiler

The *Compiler* job can be divided into three phases: during *parsing* the fitted parameters are extracted from the trained featurizers and models; parsed pipelines are then *analyzed and optimized* ; finally, a *conversion* phase generates the SQL implementations.

Parsing. Predictive pipelines are actually Direct Acyclic Graphs (DAGs) of *operators*, where each operator can be a data featurizer or a model. In the parsing phase, input predictive pipelines are parsed one operator at a time, and each operator is *wrapped* by a *container* object maintaining input/output relationship, as well as an *operator signature* and an *extractor function* used for extracting the fitted parameters. Operator signatures are initialized with the object types (e.g., the result of the `type` function applied over a Python operator object) and used for picking the correct extractor (and conversion) function for the given operator instance. MASQ compiler is extensible: extractor functions are registered at startup time into a hash table mapping operator signatures to the related extractor function. In its current implementation, MASQ provides wrappers for the Sklearn and ML.NET libraries, and extractors for linear and tree models, as well as a handful of featurizers (standard scaler, one-hot encoder, and label encoder). At the end of the parsing phase, the input pipeline is "logically" represented in MASQ as a DAG of containers storing all the information required for the successive analysis and conversion phase.

Example 1 (Parsing a Sklearn pipeline) *Let us suppose that a user provides a Sklearn pipeline composed of a* scaler *[134] followed by a* linear regression *model. Furthermore, let us suppose that, once trained, the pipeline is applied over the numeric columns of the* TaxiTable *dataset represented in Table 3.1, which collects information on taxi trips and can be used to predict their fares using regression techniques. Figure 3.3 depicts the trained pipeline with an excerpt of its parameters (top) and the result of parsing (bottom). During parsing* MASQ *generates (1) a container wrapping each operator, and containing the extractor function; and (2) wires the containers into a DAG following the input/output dependencies in the pipeline (in this specific example, the container DAG is a simple sequence).*

Analysis and Optimization. In this phase, the DAG of containers generated in the parsing phase is traversed in topological order in two passes. During the first traversal pass, for each operator MASQ extracts the operator's parameters by calling the referenced extractor function stored in the container. Extracted parameters are stored within the container. MASQ supports different to-SQL *converters* based on the operator characteristics. By default MASQ uses a mix of SELECT and CASE statements for converting ML operators into SQL, but sometimes

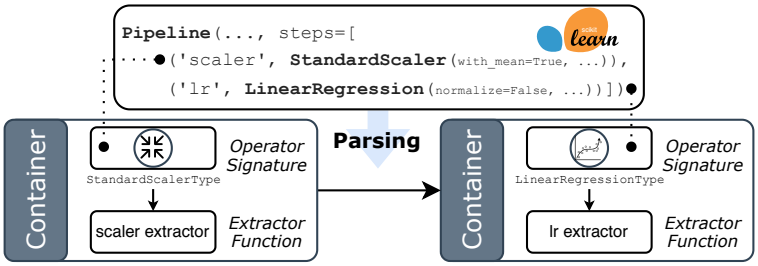


Fig. 3.3 Parsing of the pipeline of Example 1. The pipeline (top) is parsed on a container DAG (bottom). Each container stores a reference to the operator, its signature and extractor.

	Pc	Tts	Td	Pt	Vi	Label
t_1	1	1271	3.8	CRD	CMT	17.5
t_2	1	720	2.34	CRD	VTS	10.5
t_3	1	0	11.06	CSH	VTS	120
t_4	1	3	0	NOC	CMT	52
t_5	5	1560	19.97	UNK	VTS	52

Table 3.1 The TaxiTable used in the examples.

the number of features or structure of the operators is restricted by DBMSs' limits. In the latter case, in the first traversal pass, MASQ injects a *triplet-representation operator* (TRO). This operator is used to inform the compiler to transform the data from the default into a triplet format during the conversion phase, and to successively use the related triplet-based conversion function for each subsequent operator. During the second pass, MASQ tries to apply operator fusion optimizations. Operator fusion substitutes pairs of operators in the DAG with a merged operator where the signature is a concatenation of the signatures and containing the union of the original parameters.

Example 2 (Analysis of the Sklearn pipeline) *During analysis, the extractor functions of the parsed pipeline of Example 1 are triggered. Specifically, the parameters extracted from the scaler and linear model are shown in Tables 3.4a and 3.4b, respectively. In the* `StandardScaler` *case, the extractor pulls the* mean *and the* standard deviation *values for each column by calling* `mean_` *and* `scale_` *from the operator object, respectively. The extractor for the* `LinearRegression` *retrieves the weights and the bias by calling respectively* `operator.coef_` *and* `.intercept_`.

The discussion on triplet-based conversion is postponed, while next is presented the simplest case when no optimizations nor triplet-representation are triggered.

Conversion. During this last phase, the DAG of containers is again traversed in topological order and a conversion-to-SQL function triggered based on each operator signature. Each

	Pc	Tts	Td
mean	1.8	710.8	7.434
std	1.6	638.9	7.28

(a) Scaling

Findex	Weight
1	-20.33
2	-31.36
3	48.72
bias	50.4

(b) Regression

Fig. 3.4 Parameters extracted from the pipeline of Example 1.

conversion function receives as input the parameters (extracted during analysis, and stored directly into the container) and generates a string containing the SQL implementation. The SQL implementations of all operators are then merged into a unique query following the input/output dependencies expressed in the container DAG.

As for the extractors, MASQ stores a map of operator signatures / conversion functions. MASQ currently implements converters for the following operators (where each of them has a default and triplet-format version): standard scaler, one-hot encoder, label encoder, gradient boosting classifier / regressor (w/ and w/o tweedie loss), random forest, decision tree, linear regression with some variants (i.e., Poisson and SDCA), logistic regression classifier, PCA, and linear SVM classifier. In the default case, the above operators can be implemented using the following simple strategies.

Conversions via SELECT *statements.* The conversion into SQL is straightforward when the ML prediction function consists only of algebraic operations between the extracted parameters and the input features. Examples of methods implemented via SELECT statements are normalizers/scalers and linear models (by unrolling the linear algebra operations into the SELECT clause).

Example 3 (Pipeline conversion) *The conversion of the pipeline of Examples 1 and 2 leads to the queries in Figure 3.5, where the two* SELECT *clauses implement the scaler and the regressor, respectively. Note that the two queries, at conversion time, will be merged into a unique query.*

Conversions via CASE *statements.* SQL CASE statements can be used to implement rule-based learners such as decision trees, or data featurizers such as one-hot encoding (OHE). In the former case, each rule from the model is translated into a SQL CASE statement; rules are then nested, according to the model, by nesting the correspondent CASE statements.

For the latter, the CASE statements is used to encode input categorical values into a sequence of columns, one for each distinct value. For each input, only the column of that particular categorical value will store 1, all the other columns will be 0.

```
SELECT ((Pc - 1.8) / 1.6) AS Pc,
 ((Tts - 710.8) / 638.9) AS Tts,
 ((Td - 7.434) / 7.28) AS Td
FROM TaxiTable

SELECT (-20.33*Pc - 31.3*Tts + 48.72*Td + 50.4)
  AS Score
FROM NormalizedTable
```

Fig. 3.5 Scaling and linear model in SQL.

Example 4 (OHE) *Let us suppose to apply a one-hot encoder to the columns* Pt *and* Vi *of the data represented in Table 3.1. The result of this transformation is a new set of columns, one for each unique categorical value of the* Pt *and* Vi *columns. As we can see in the query of Figure 3.6, each column name is generated by concatenating the original categorical input name with each distinct value. Each column will store 1 only if the value is of the proper category.*

```
SELECT
 CASE WHEN Pt = 'CRD' THEN 1 ELSE 0 END AS Pt_CRD,
 CASE WHEN Pt = 'CSH' THEN 1 ELSE 0 END AS Pt_CSH,
 CASE WHEN Pt = 'NOC' THEN 1 ELSE 0 END AS Pt_NOC,
 CASE WHEN Pt = 'UNK' THEN 1 ELSE 0 END AS Pt_UNK,
 CASE WHEN Vi = 'CMT' THEN 1 ELSE 0 END AS Vi_CMT,
 CASE WHEN Vi = 'VTS' THEN 1 ELSE 0 END AS Vi_VTS
FROM TaxiTable
```

Fig. 3.6 One-hot Encoding in SQL.

Combining SELECT *and* CASE *statements.* Some model requires the combination of SELECT and CASE statements. This is, for example, the case for tree-ensembles models. Tree-ensemble methods construct a sequence of decision trees and adopt different strategies to select the outputting class (e.g., the mode class in classification tasks, the means of the resulting values in regression tasks). In the SQL implementation for this kind of methods the CASE-based queries of the decision trees are nested in a query that collects the results and computes the final output via a SELECT clause.

Escaping DBMSs' Limits. DBMSs are not designed for ML, and it is fairly easy to reach database limits with ML pipelines of reasonable complexity. During the analysis phase, MASQ detects when a certain limit is reached, and it automatically selects, at conversion time, the proper operator implementation. In the following a couple of problems, and related solutions, encountered while implementing MASQ are discussed.

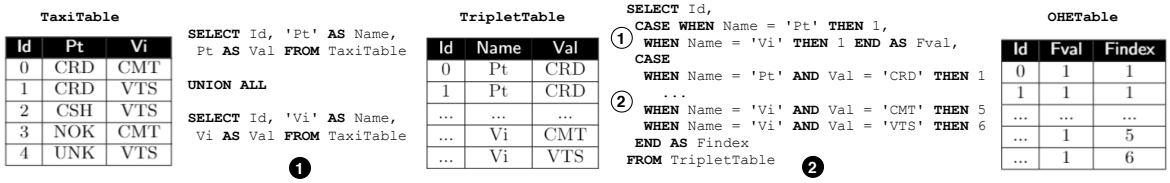


Fig. 3.7 SQL workflow for the one-hot encoding sparse implementation.

Limit on the number of columns. SQL Server wide (sparse) tables support 30k columns; 1024 in regular tables [137]. MySQL supports a maximum of 4096 columns per table [101]. Conversely, ML datasets and pipelines can easily reach several millions of features. Therefore, high dimensional data need to be stored using a different format.

MASQ **solution.** To overcome the above problem, a *triplet-based representation* is used, where each record is stored in the form `(identifier, attribute_name, attribute_value)`. As already discussed, MASQ injects TRO into the plan during analysis. As an example, next the compilation process for a pipeline that contains an OHE operator generating a large number of features is considered.

Example 5 (Pipeline with TRO and OHE) *Let us assume again to transform the columns* Pt *and* Vi *of Table 3.1 using OHE. This time, however, the total number of distinct values for these categorical columns is greater than the maximum number of columns supported by the database* [13]*. In this case, the compiler will inject a TRO operator before OHE. The following converter is then instructed to use the triplet-based conversion function for OHE, which uses a sparse implementation. Specifically, the converter in this case generates pairs in the form* `(1, index_value)` *instead of materializing the full dense vector like in Example 4. Figure 3.7 provides the SQL workflow implementing the pipeline.*

The SQL statement in the left-hand side of the Figure (❶) implements the TRO operator. This creates a TripletTable *where the first column is the identifier of the rows in the dataset, while the second and third columns store the attribute name and its values, respectively. In the SQL query on the right-hand side (❷), the first* CASE *statement (①) is used to select the attribute(s) to encode and sets 1 as the value for those attributes. The second* CASE *statement (②) provides the index of non-zero values. Note that indexes are sequential, even across categorical columns (the index for the* Vi *column starts at 5 instead of 1). This is because one-hot encoded columns are implicitly* concatenated *into a unique feature vector.*

Limits on SELECT **and** CASE **clauses.** High dimensional datasets introduce problems not only regarding the data representation, but also on how we implement operators in SQL. In

[13] This check is, for example, implemented for Sklearn as a condition on the total number of elements of the parameter extracted from `operator.categories_`.

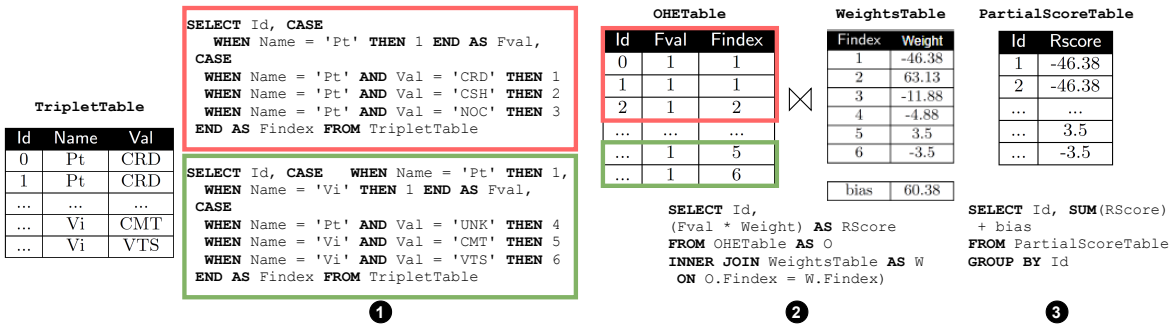


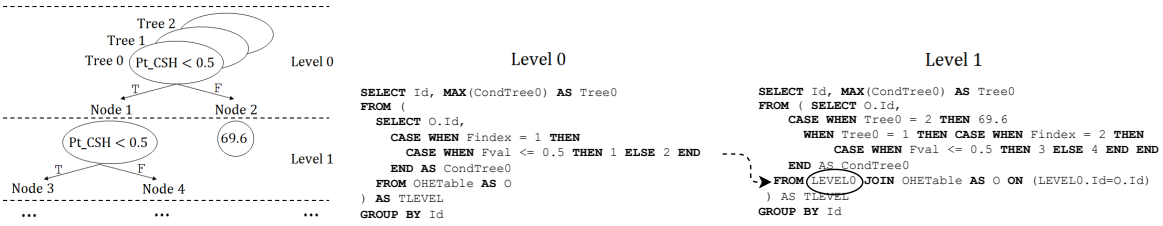
Fig. 3.8 Pipeline with OHE followed by a linear regression executed in MASQ with TRO and partitioning.

fact limits exist on the number of columns allowed in SELECT statements (e.g., 4096 for SQL Server), or the total number of conditions in CASE clauses (few thousands for SQL Server [138]).

MASQ **solution.** These two issues are addressed by injecting TROs, and partitioning large SELECT and CASE statements. Two pipelines made of an OHE plus a linear regression (Example 6) and a tree-ensemble (Example 7) are used to showcase how this strategy works.

Example 6 (OHE and linear regression) *Figure 3.8 depicts how* MASQ *translates this pipeline. The workflow directly starts from the* TripletTable *of Example 5, where a large number of features is generated after the application of OHE. Additionally, let us suppose that also the number of* CASE *statements in the OHE is too large, and therefore the query for the encoding needs to be partitioned (❶). Each partition is executed independently and generates a distinct* OHETable*. The* OHETable*s are then joined (❷) with the* WeightsTable *containing the linear regression's parameters (e.g., Table 3.4b). Over the output of the join, each feature value is then multiplied with the respective regression weight, and generate the partial sums which will then be aggregated by a final query (❸). Note that, differently than the unrolled version, the triplet representation prevents the overcoming of the limit of columns in the* SELECT *statements.*

Example 7 (OHE and tree-ensemble model) *The implementation of tree-ensemble models after OHE basically follows the same workflow as Example 6, with two important differences. First, while* WeightsTable *can be partitioned following the* OHETable *partitioning, for tree-ensembles each tree could potentially touch all input features. To solve this, tree-ensembles are partitioned into* batches *(up to the number allowed by DBMS constraints), and run each batch over the union of the* OHETable*s. Secondly,* CASE *statements cannot be directly used to implement trees on data in triplet-based representation. This is because each original (not triplet) row is split into several triplet rows, and* CASE *statements, to work, should now be*



(a) Snapshot of a tree-ensemble.

(b) How MASQ compiles the first 2 levels of *Tree0*.

Fig. 3.9 How tree-ensembles over triplet are translated in MASQ.

able to select multiple rows simultaneously. To overcome this limitation, MASQ *implements a technique whereby all the trees in the batch are traversed together, level by level, in a breath first search manner [146]. For each level, the triplets that match the conditions on the trees are selected, and the condition is used to select the next* CASE *statement in the next level. In Figure 3.9(a) a tree-ensemble model with 3 trees is reported and a detailed representation of the first 2 levels of* Tree0 *are shown, where the decision nodes use OHE features* Pt_CRD *and* Pt_CSH. *Figure 3.9(b) contains the queries for the first 2 levels of* Tree0. *The query for* Level 0 *(i.e., the root) of* Tree0 *contains two nested* CASE *statements: one for selecting the proper feature (i.e., feature* Pt_CRD *has* Findex = *1,* Pt_CSH *as* Findex = *2), and one for evaluating the condition of the feature. The result of the condition contains the index of the node which will then be used in the successive level. The final* GROUP BY *and* MAX *operations are used to return a unique not null record. In the* Level 1 *query the results of the* Level 0 *are used and three nested* CASE *statements are exploited: in the outermost statement there is one condition for each node, while for each node are considered, again, two* case *statements, one for selecting the proper feature, and one for evaluating the condition. The others levels follow a similar approach. With this technique it is possible to evaluate, for each level, batches of trees concurrently. The SQL query of Figure 3.9(b), for each level, will then actually contain different* CASE *statements for each tree. A padding logic to deal with trees with different number of levels is finally applied.*

The Executor

The *Executor* provides the functionalities necessary for the execution of the SQL queries generated by the *Compiler* in a relational database. The *Executor* makes use of a set of connectors (currently MASQ supports MySQL and SQL Server via Python and C# connectors). Finally, a small *driver* program manages the executions.

Datasets	#Rows	#Columns	Task
Iris	150	4 categorical	Multi-class
Heart Disease	303	13 numeric	Binary
Bike Sharing	17,379	12 numeric	Regression
Taxi Fare	200,000	3 numeric 3 categorical	Regression
Credit Card	284,897	30 numeric	Binary
Criteo	4,000,000	13 numeric 26 categorical	Binary
Flight Delay	21,604,865	23 numeric 3 categorical	Binary

Table 3.2 Datasets used in the experiments

3.3.3 Experimental evaluation

The experimental evaluation aims to provide an answer to the following question: *are databases a good fit for inference of ML pipelines?* In order to answer this question, 10 representative ML pipelines are implemented on two ML frameworks, namely Sklearn and ML.NET, and their execution performance is compared against MASQ-generated queries run on 2 DBMSs: MySQL and SQL Server. The tests will evaluate both the final accuracy (with the expectation of matching the same accuracy of the ML frameworks), and the pure throughput performance. Finally, the experiments will explore how SQL pipelines perform (1) without any optimization; (2) with database-specific optimizations, e.g., data indexing; (3) with "logical" optimizations such as operator fusion between OHE and GBDT; and (4) with different implementation variants based on DBMS limits. Finally, some negative results on text featurization and neural network models are reported.

Datasets. The majority of the experiments are conducted over the 7 datasets described in Table 3.2. On these datasets a wide range of tasks is executed: from binary and multi-class classification, to regression. Iris is the smallest one with 150 records, each described by 4 numeric columns. Criteo is the dataset with the largest number of features (39 columns). At inference time the input columns are transformed with OHE into around 2.5 million features. Flight Delay is the biggest dataset: it contains more than 21 million records and 26 initial columns which, during execution, they get expanded into approximately 700 features.

ML Pipelines. Table 3.3 shows the pipelines used in the evaluation. 8 pipelines have been taken from ML.NET samples [97]; 2 of them (Criteo and Flight Delay) are pipelines commonly used to evaluate the scalability of ML frameworks [7]. Each pipeline was first implemented in ML.NET and then in Sklearn (note that for P3 the XGBoost [28] library is

Pipeline	Dataset	Featurizer	Model	Framework
P1	Iris	MapValueToKey	SDCAMaximumEntropy	ML.NET
		LabelEncoder	LogisticRegression	Sklearn
		SELECT Statement	SELECT Statement	MASQ
P2	Heart Disease	N/D	FastTreeClassifier	ML.NET
			GradientBoostingClassifier	Sklearn
			CASE and SELECT Statements	MASQ
P3	Bike Sharing	N/D	LBFGSPoissonRegression	ML.NET
			SGDRegression	Sklearn
			SELECT Statement	MASQ
P4			SDCARegression	ML.NET
			SDCARegression	Sklearn
			SELECT Statement	MASQ
P5			FastTreeRegression	ML.NET
			GradientBoostingRegression	Sklearn
			CASE and SELECT Statements	MASQ
P6			FastTreeTweedie	ML.NET
			XGBoost w/ Tweedie loss	Sklearn
			CASE and SELECT Statements	MASQ
P7	Taxi Fare	OneHotEncoding, NormalizeMeanVariance	SDCARegression	ML.NET
		OneHotEncoding, StandardScaler	SDCARegression	Sklearn
		CASE and SELECT Statements	SELECT Statement	MASQ
P8	Credit Card	DropColumns, NormalizeMeanVariance	FastTreeClassifier	ML.NET
		DropFeatures, StandardScaler	GradientBoostingClassifier	Sklearn
		SELECT Statement	CASE and SELECT Statements	MASQ
P9	Criteo	OneHotEncoding	FastTreeClassifier	ML.NET
		OneHotEncoding	GradientBoostingClassifier	Sklearn
		TRO + Partitioned Statements	CASE and SELECT Statements	MASQ
P10	Flight Delay	OneHotEncoding	FastTreeRegression	ML.NET
		OneHotEncoding	GradientBoostingRegression	Sklearn
		TRO + Partitioned Statements	CASE and SELECT Statements	MASQ

Table 3.3 ML pipelines used in the experimental evaluation.

used in order to match the Tweedie loss on ML.NET); finally both the implementations are converted into SQL queries with MASQ. For each pipeline, Table 3.3 contains the *Featurizer*s (when used) and the final *Model*. For each pipeline, the featurizers and models adopted are listed by *Framework*, and for MASQ is reported the technique used from Section 3.3.2, i.e.,

whether SELECT statements, CASE statements, both SELECT and CASE statements, TROs, or partitioned statements are adopted.

Setup. The experiments are executed on an Azure Standard D32 v3 machine with 32 virtual cores, 128GB of RAM and 256GB of local (SSD) storage. The machine runs Ubuntu version 18.04, Sklearn version 0.21.2, and ML.NET version 1.2. MASQ was evaluated on two RDBMSs: MySQL version 5.7.29 and SQL Server 2017 version 14.0.3223. All experiments have been performed 5 times, and the average results reported. For MASQ experiments the average query time as reported on the database catalog is considered; for Sklearn and ML.NET the time of the running process is measured. For GBDT models the default number of trees (100) is used.

Accuracy

The first step for evaluating whether DBMSs can be used as prediction serving systems is to check that the prediction outcomes match with the original ones generated by the ML frameworks. In Table 3.4 the errors between the outcomes generated by the baseline frameworks (Sklearn and ML.NET) and MASQare reported. They are computed as the mean of the absolute differences between the returned values (score of the predicted class) for regression (classification) tasks. As we can see from the table, using SQL queries for inference introduces negligible errors (e.g., between $1e-05$ and $1e-06$ in the general case; $1.49e-02$ in the worst case). The worst case is due to the *Compiler* which uses ML.NET tree-aggregation logic, while XGBoost apparently uses a specific aggregation function for Tweedie.

Pipeline	MASQ vs ML.NET	MASQ vs Sklearn
P1	$5.99e-08$	$1.97e-06$
P2	$2.47e-06$	$2.44e-06$
P3	$3.74e-05$	$2.49e-06$
P4	$2.38e-05$	$2.50e-06$
P5	$1.65e-05$	$2.52e-06$
P6	$2.69e-05$	$1.49e-02$
P7	$1.03e-05$	$2.50e-06$
P8	$5.17e-06$	$3.81e-06$
P9	$2.07e-06$	$2.13e-06$
P10	$7.46e-06$	$1.83e-06$

Table 3.4 Error (mean of the absolute difference) on the predictions generated by MASQ versus ML.NET and Sklearn.

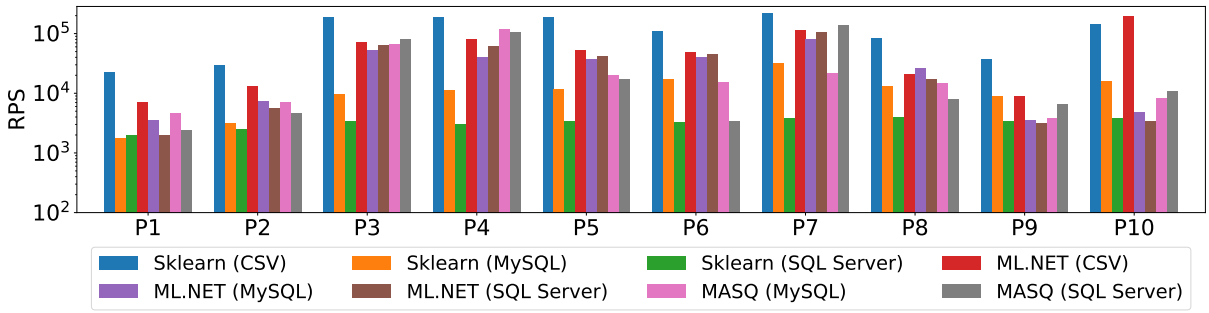


Fig. 3.10 Throughput for Sklearn, ML.NET (on CSV, MySQL and SQL Server) and MASQ (on MySQL and SQL Server).

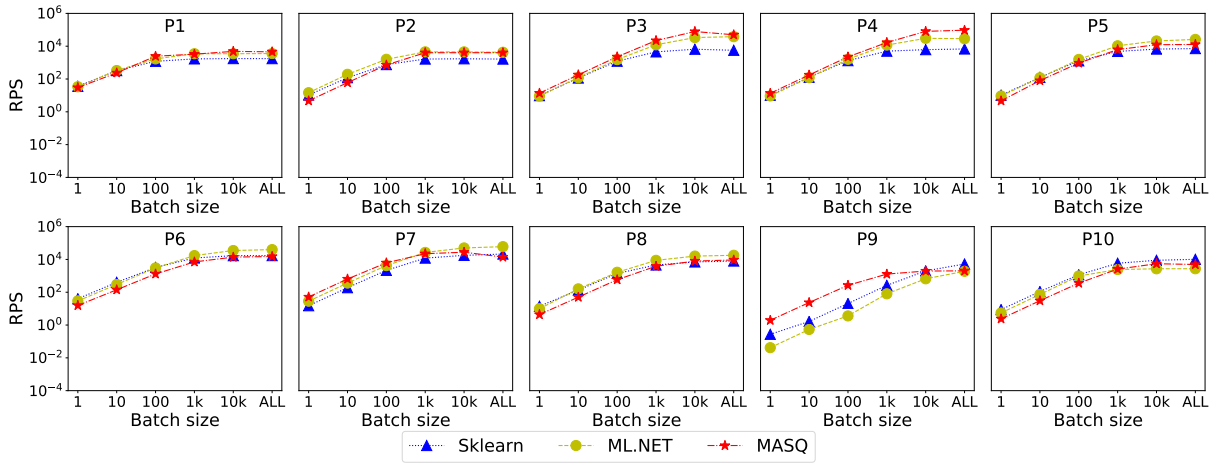
Throughput

The goal of this experiment is to compare the performance of each framework and on each pipeline in serving predictions over the full datasets. For Sklearn and ML.NET the performance when the data reside both over flat CSV files and in the databases are also tested. In the latter case, data has to be moved out of the database before the predictive pipeline can be executed. Since the data in the pipelines have different sizes, the average throughput in terms of *rows evaluated per second* (RPS) is calculated. Figure 3.10 shows the results.

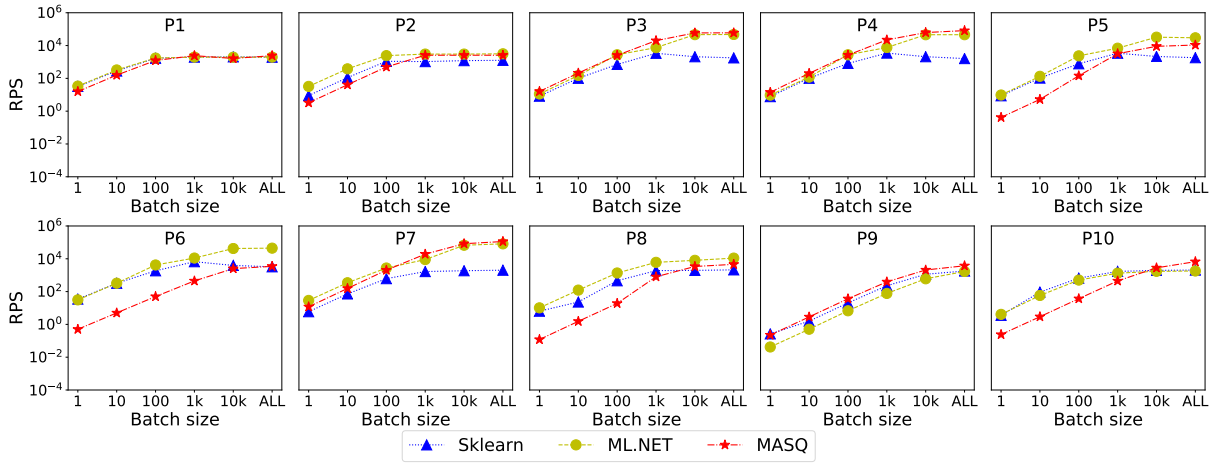
Discussion. There are several insights from this experiment: (1) except for P10 (Flight Delay) Sklearn on CSV has the higher throughput; (2) as expected, the throughput decreases for the ML frameworks when the data needs to be moved out the database, although it decreases considerably (around 10×) for Sklearn, less for ML.NET—this is probably due to the quality of connectors; (3) in general there is no clear winner between MySQL and SQL Server connectors for ML.NET, whereas for Sklearn, the SQL Server connector performs much worst than the MySQL one; (4) MASQ throughput is comparable to the database version of the ML frameworks for pipelines with linear models (P1, P3, P4, P7), while it is much slower when tree-ensemble models are used (P5, P6, P8); (5) MySQL and SQL Server implements different optimization strategies whereby the same query generated by MASQ can result in different performance. The 4th point is somehow surprising and invalidates the common knowledge that databases are not so performant over linear algebra. Conversely, tree-models performance varies based on the implementation and dataset. This aspect will be further explored in the following sections.

Scalability

This section studies how the throughput of the frameworks changes as the amount of data processed by each system is scaled. This scenario is implemented by splitting each dataset



(a) Scalability trend for MySQL



(b) Scalability trend for SQL Server

Fig. 3.11 Scalability of the different frameworks, over MySQL and SQL Server by changing the batch size.

into batches of various sizes, and the overall throughput is plotted. Batches of 1, 10, 100, 1K, and 10K rows are tested, plus the full dataset in one batch. The full dataset is used also in the cases where the batch size is greater than the total length. Figure 3.11 shows the results for MySQL (a) and SQL Server (b). For Sklearn and ML.NET the versions where the data resides in the database are considered.

Discussion. As we can see, all the pipeline present similar trends: as the batch size increases, the throughput increases as well, up to a saturation point (either the dataset size or the resources are saturated). As in Figure 3.10 we can see that Sklearn on SQL Server has worst performance than MySQL. Regarding MASQ: for pipelines P1, P3, P4, and P7, with linear models, MASQ shows the best performance in most of the settings. Conversely, for

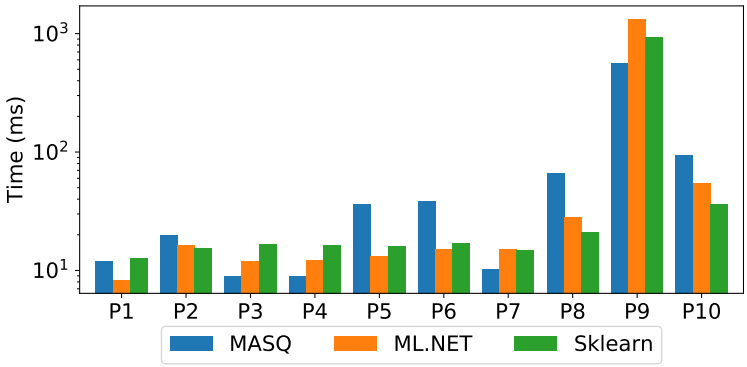


Fig. 3.12 Latency for Sklearn, ML.NET and MASQ on MySQL for a single record.

tree-based models we can see that in the majority of the settings (P2, P5, P6, P9) MASQ performance are either the best or in between Sklearn and ML.NET. For the remaining pipelines (P8, P10) MASQ trend is generally worst than the baseline frameworks, although in aggregate not by much. Tests performed on SQL Server provided similar results except for P6 and P8, where stricter limits on the number of nesting CASE statements forced MASQ to generate more simplified queries.

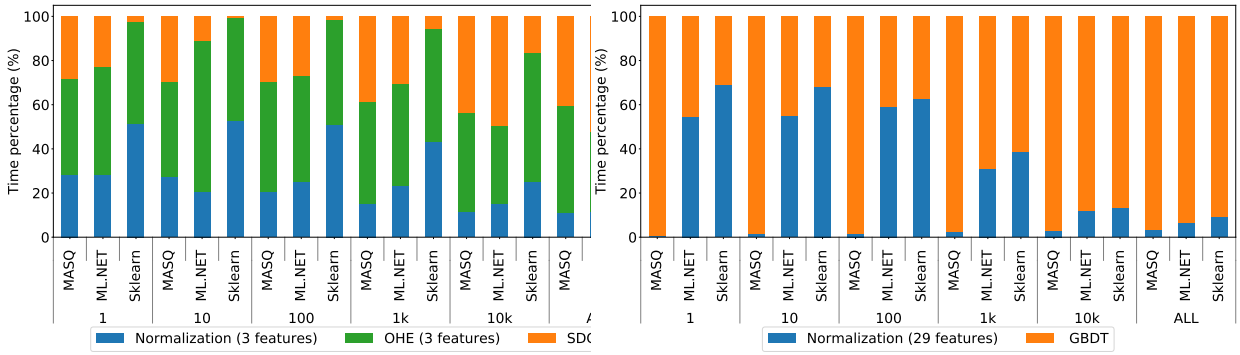
Latency

This experiment analyzes the latency performance for executing online (single record) predictions. Figure 3.12 shows the results computed over MySQL, where for ML.NET and Sklearn is also considered the time to pull the records out of the database.

Discussion. The latency numbers confirm that MASQ performs better (up to 3×) than the baseline frameworks for linear models (P1, P3, P4, P7) while tree-based models (P2, P5, P6, P8, P9, P10) can be up to almost 2× slower (P5, P6, P8). Even for the same dataset, we can notice the latency of tree ensemble models is worse than the linear ones (i.e., P4, P5). Next the trade-offs between linear and tree ensemble models will be studied more in detail by breaking down the performance for each single pipeline component.

Performance Breakdown

In this section the performance over few selected queries and for the large datasets is drill down. Firstly, the contribution of each pipeline operator on the final runtime for queries P7, P8, P9 and P10 is evaluated. Successively, the time spent between data loading, data write, and computation for all the above pipelines is analyzed.



(a) P7 breakdown (b) P8 breakdown

Fig. 3.13 Operator breakdown for P7 (Taxi Fare) and P8 (Credit Card).

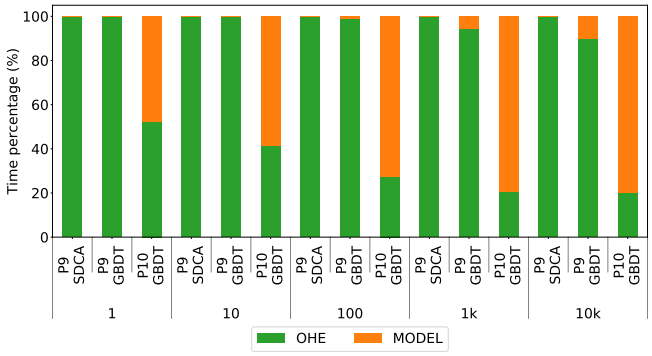


Fig. 3.14 Operator breakdown for P9 (Criteo) and P10 (Flight Delay). For P9, a GBDT model is also compared with a SDCA.

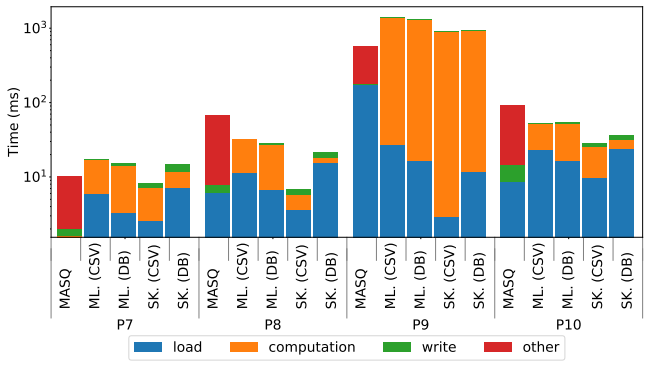


Fig. 3.15 Breakdown of the latency for MASQ ML.NET (ML.) and Sklearn (SK.) and for pipelines P7, P8, P9 and P10. The time spent is divided into four buckets: load, computation, write and other (which summarizes all the time spent in operations not related to the previous three components).

Operator Breakdown The runtime for each operator as percentage over the total runtime is plotted by batch size (where batch of 1 is online). For MASQ are reported the numbers over MySQL (similar results hold for SQL Server), and it is compared against Sklearn and ML.NET over CSV for P7 and P8 in Figure 3.13. In Figure 3.14 are instead reported the results for MASQ for P9 and P10, where for P9 two variants are considered: one with a tree model (GBDT, as described in Table 3.3) and one with a linear model (SDCA). Recall that Criteo is the largest dataset with 2.5M features (after OHE). By running these two variants it is possible to study, in the worst-case scenario, the tradeoffs between linear and tree ensemble models for MASQ.

Discussion. Starting with P7, it is possible to notice that: (1) data featurizers take the majority of the time; and (2) as the batch size increases, the time spent on normalization decreases. This second point is even more market on P8 where for Sklearn and ML.NET normalization surprisingly takes more than 50% for batches of 1, while it takes less than 10% when the entire dataset is scored at once. This behaviour can be explained by considering the benefits of vectorization which increases with the batch size. In P8, MASQ spends the majority of the time (>90%) on the evaluation of the GBDT model.

Concerning the evaluation of P9 and P10 in Figure 3.14, it is possible to note that: (1) the time required to complete the OHE operator is proportional, as expected, to the number of features generated rather than on the number of rows processed (i.e., the percentage of time spent on OHE is greater in P9 than in P10: the first generated 2.5M features over 4M rows, the second 700 over 21M rows); (2) as the batch size increases, the time spent on executing the GBDT model increases, up to reaching 80% in P10 for a batch size of 10K. The experiment performed on P9 with SDCA, instead, confirms that the time required to execute the linear model is irrelevant wrt the time for executing the featurizer or the GBDT.

Latency Breakdown This section analyzes the latency (single record) performance for the pipelines used in the previous section. The breakdown is performed by dividing the latency into four components: *data load*, *computation*, *write* and *other*. In *other* is aggregated all the time spent in miscellaneous operations that cannot be classified otherwise: examples are acquiring table locks, data structures disposal, etc. The results taken from MASQ running on MySQL are reported and compared against Sklearn and ML.NET latency times obtained by running them both over CSV files and over records loaded from MySQL.

Discussion. The breakdown in Figure 3.15 shows that the computation time is almost null for all pipelines executed with MASQ, while the majority of time is spent on data loading. For ML.NET and Sklearn the computation time is instead quite large, in particular in P9 where the OHE creates around 2 million features. Interestingly, the difference between data

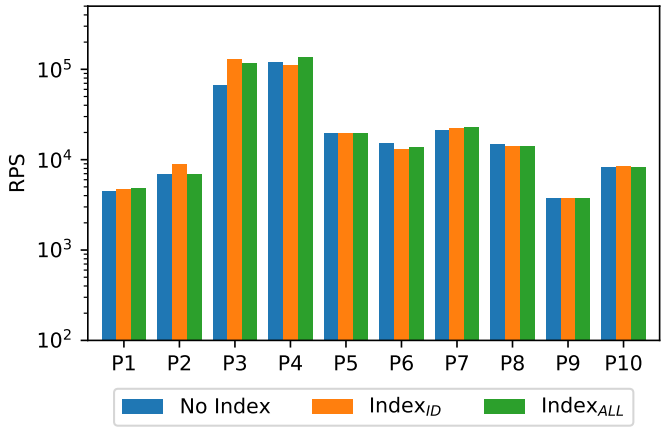


Fig. 3.16 Performance comparison with indexing (MySQL).

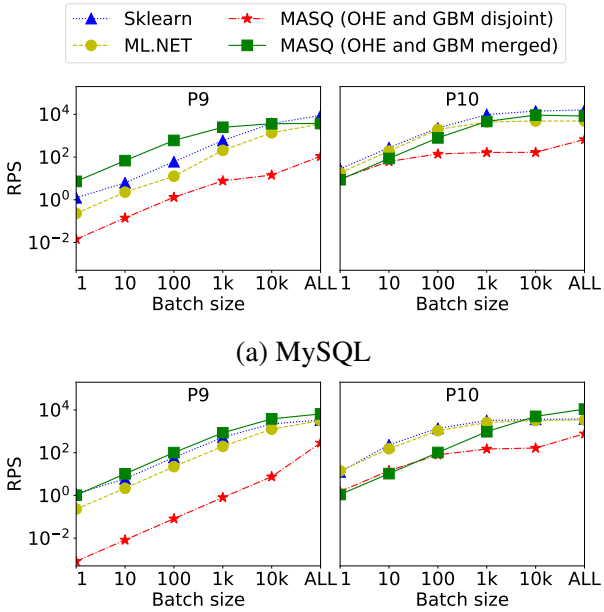
loading for the CSV and the DB is minimal for ML.NET while it is quite large for Sklearn. Again, the quality of the database connectors may have affected it.

Optimizations

This section explores two different types of optimizations: database-specific optimizations such as adding indexes, and "logical optimizations" at the operator level such as operator fusion.

Using Indexes. This experiment evaluates whether the performance over the DBMSs can be improved by applying indexes. Three settings are analyzed: in the first setting, referred to as *No Index*, a clustered index on the primary key is added. In the second setting, $Index_{ID}$, a non-clustered index is considered on the column identifier (*ID*). Finally, in the setting $Index_{ALL}$ to each column is applied a non-clustered index. The indexes have been added both to the input dataset, and to temporary tables when used (e.g., in P7, P9 and P10). Figure 3.16 contains the results for this experiment for MySQL. SQL Server results are similar.

Discussion. The results show that only for pipelines P3 there is a small benefit from indexing. This indeed is an unexpected behavior. No improvements for the pipelines where indexes are built also on temporary tables (P7, P9 and P10) are obtained. Note that in this latter case, the cost of building the index is counted into the final queries running time. In this experiment also a column store layout in SQL Server has been explored. Similarly to indexing, this technique does not introduce any significant improvement and sometimes even degrades performance. With small batches (i.e. 1 to 10k) the performance degraded of up to 3×. This is due to the overhead of reconstructing the per-row format of records. An increase in



(a) MySQL

(b) SQL Server

Fig. 3.17 Operator fusion (GBDT + OHE) for P9 and P10.

performance was recorded only in P8 with large batches (i.e. greater than 100k) and with a tree ensemble depth greater than 6. This is motivated by the fact that deep trees require repeated access to the features and this pattern is able to better exploit the columnar format.

Operator Fusion. In this experiment an optimization for pipelines P9 and P10 is analyzed, where the SQL queries implementing the tree-ensemble models are fused with the OHE featurizer. Specifically, the CASE statements evaluating the tree conditions on attributes targeted by the OHE featurizer are rewritten to compute both the featurization and the prediction in the same statement. Figure 3.17 shows the RPS for the optimized implementation on MySQL compared to the baseline where no optimization is used.

Discussion. The results show that when operator fusion is not used MASQ performance decrease substantially for P9 and P10. With operator fusion MASQ does more computations per single row (i.e., for each row the encoding is computed multiple times, one for every time the row is used by a tree), but since the number of features is large and not all of them used, the total number of encodings is less. A similar optimization was also applied to P8 where normalization was fused with GBDT. This last experiment introduced a 4× slowdown. This is because all features are used by the GBDT model. This result suggests that a cost-based optimizer is likely required for selecting the best compilation strategy when optimizations are enabled. Regarding latency, operator fusion improves P9 by 5×, and P10 by 2×.

Study of Operators Implementation

In this section few possible variants of the operator implementations discussed in Section 3.3.2 are studied, as well as how model characteristics affect the query performance.

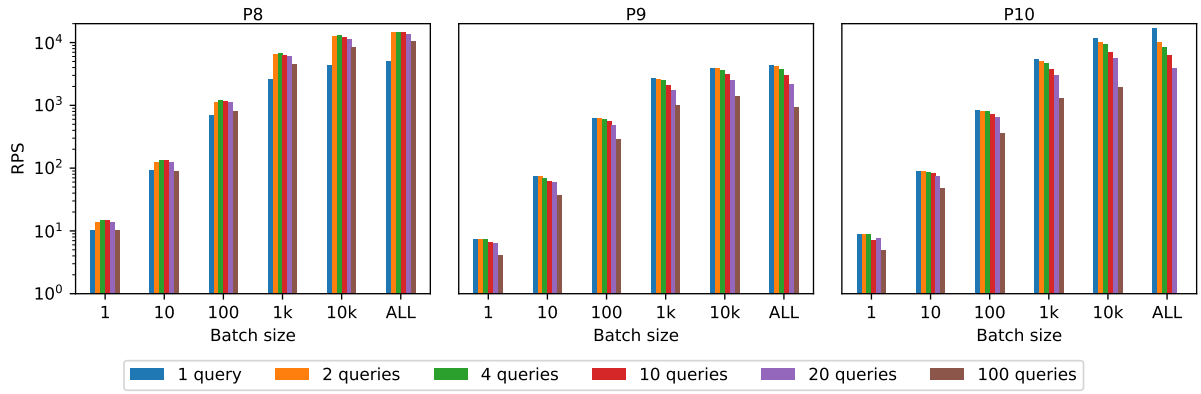


Fig. 3.18 Comparison of different tree implementation methods (MySQL).

Tree ensembles Implementation. Pipelines P2, P5, P6, P8, P9, and P10 make use of tree-ensemble algorithms whereby a certain number of trees (100 in our experiments) are executed, and their predictions combined. In this experiment, two different implementations for this operation are tested. In the first implementation, the queries representing each tree are a subquery of an outer query computing the final score over the partial results (this is the approach described in Example 4). The results obtained with this implementation are represented in Figure 3.18 as "1 query". In the second implementation, different set of trees (1, 5, 10, 25 and 50) are batched in multiple queries (respectively 100, 20, 10, 4, 2) which store the partial predictions into an intermediate table. A final query then computes the output by aggregating the results from the temporary table. In this experiment, pipelines P8, P9, and P10 are tested over different batch sizes size. Figure 3.18 plots the results for MySQL only.

Discussion. The experiment shows that no approach is clearly overcoming the other in all use cases. If we look at the largest datasets (P9 and P10), we see that the implementation with 100 queries obtains the worst results (for P10 over the entire dataset the system even crashed because of the size of the intermediate results). For these pipelines the best results are achieved with a single query. Interestingly, the same setting does not produce the best results in P8. This is due to the larger number of conditions evaluated by the trees of P8. The total number of conditions are 85,176 in P8, 18,166 in P9 and 11,837 in P10.

OHE followed by linear models When a OHE featurizer is followed by a linear model, a temporary table is built for storing the results of the featurization (the OHETable in Figure 3.8), and its content joined with the model parameters table (see Section 3.3.2 for details). In this experiment a possible alternative plan implementing the operation as a multi-way join is evaluated. Pipeline P9 with SDCA has been executed on the MySQL DBMS, and the OHETable has been partitioned in 300 tables. [14] Figure 3.19 plots the results of the experiment.

Discussion. As we can see the multi-way join implementation performs better over large batch sizes, whereas when the data to process is smaller, the single intermediate table implementation performs better. This is likely because for small bath sizes, less inserts to the intermediate table are executed concurrently.

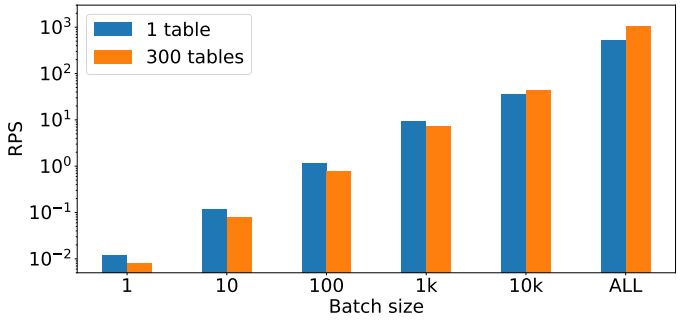


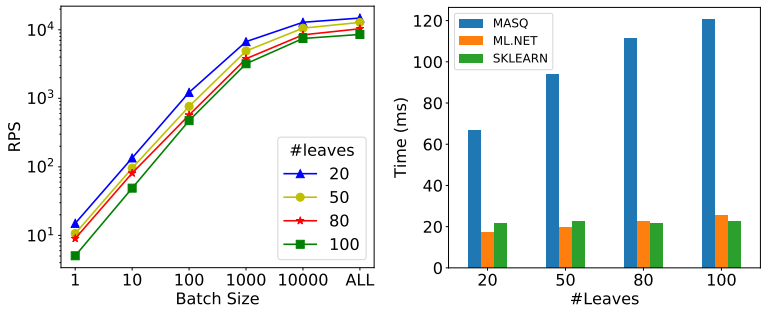
Fig. 3.19 Comparison of single intermediate data and multi-way join strategy for OHE + linear models.

Tree ensembles with variable number of leaves

In this experiment the performance of the tree ensemble implementations are studied when the number of leaves (i.e., the height) of the trees is increased. Figure 3.20a and 3.20b report the performance on MySQL of different P8 tree ensembles implementations obtained by varying the number of leaves.

Discussion. Figure 3.20a shows how performance varies, per batch size, as the number of leaves is increased. As we can see, the difference in performance is stable across the different batch sizes, and it is due to the fact that evaluating taller trees (with more leaves) requires the evaluation of more conditions. Looking at batch size of one, from Figure 3.20b it is possible to conclude that P8 latency is from 3× to 6× slower on MASQ compared to the baseline systems. Interestingly, Sklearn and ML.NET performance slightly increase with the increase

[14] This is the minimum number of tables required in order to meet SQL Server limits on case statements



(a) Batch evaluation (b) Latency evaluation

Fig. 3.20 Comparison of tree ensembles performance with variable number of leaves on P8.

of the number of leaves, while MASQ gets up 2× slower. This is likely due to the overhead of unrolling tree ensemble evaluation as a sequence of CASE statements.

Negative Results

In this set of experiments the performance of SQL implementations of textual data featurizations and neural network models is analyzed. They represent common building-blocks in ML pipelines, but they introduce computations that are hard to supported in databases with reasonable performance.

Managing textual data To study whether MASQ can support textual data, a pipeline over the Sentiment dataset [98] is considered. This dataset contains 40k records, with 7 numerical and 1 textual feature each. The ML pipeline is composed of a data featurizer (FeaturizeText in ML.NET, TfidFeaturizer in Sklearn) over the textual column, and a logistic regression model. After the application of the text featurizer, the number of features becomes around 210K. The text featurizer has been implemented in SQL using temporary tables and CASE statements, while the logistic regression is implemented as a simple SELECT statement. The left hand-side plot in Figure 3.21 shows the results against MySQL. Sklern and ML.NET are run over the data stored in the database. Similar results have been obtained for SQL Server.

Discussion. The experiment shows that MASQ performance is several order of magnitude off compared to the baseline frameworks. This is due to: (1) the large number of features generated; (2) the implementation of the text featurizer which mixes CASE statements and temporary table transformations; and (3) the heavy use of the string intrinsics functions provided by the database. This results suggests that probably text featurizers are better supported in databases with UDFs.

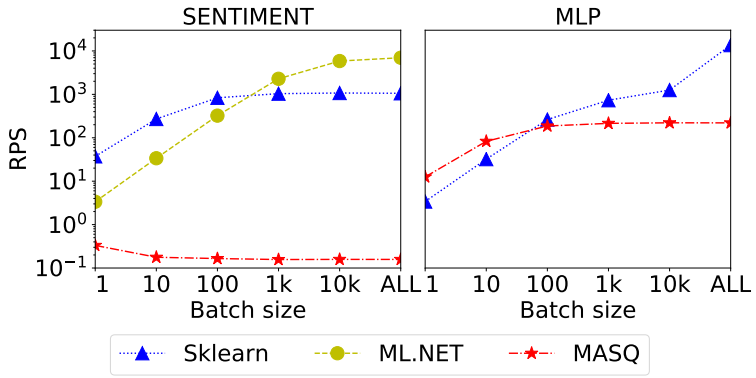


Fig. 3.21 Negative results: (left hand-side) Sentiment Analysis over textual features; and (right hand-side) a MLP model applied on Credit Card.

What About Neural Networks? For this experiment, a Multilayer Perceptron (MLP), composed of 3 hidden layers, each one with 5 nodes, has been implemented in SQL through SELECT statements. The dataset used for the evaluation is Credit Card, and MASQ is compared only with Sklearn, as ML.NET currently does not provide native support for MLP models. The results for MySQL are plotted on the right-hand side of Figure 3.21.

Discussion. As we can see from the results, MASQ performance is comparable to Sklearn only for small batch sizes, whereas for larger batch sizes Sklearn is able to better use the hardware than MySQL. This MLP model requires three matrix multiplications, and Sklearn uses BLAS libraries to efficiently compute them. Note that these results are over a very small MLP with only 3 layers and 5 neurons per layer. The same experiment applied to a larger MLP with few hundreds of neurons, has generated, as expected, worst results by several orders of magnitude.

3.3.4 Lesson Learned

From the implementation and evaluation of MASQ emerged several interesting insights. For example, linear models are not a bottleneck, while featurizers and tree-based models can be. Adding indexing is not helpful, while operator fusion sometimes is. Furthermore, several specific implementations and optimizations have been integrated in MASQ to address database limits, and these scenarios can be quite common in practice.

Another lesson learned is that although any ML operator can be translated into SQL, not all operators will have good performance (e.g., text featurization and neural networks). For their more efficient support, it is therefore more advantageous to rely on a UDF-based approach.

Finally, interesting compromises between optimizations and compilation strategies emerged. Examples are, when to use operator fusion, or when to change the operator implementation. This suggests that a cost-based optimizer is likely required to achieve the best performance. This is so more true, when hardware accelerators are also available [102].

3.3.5 Conclusion and Future work

Given the recent interest in commercial databases on providing native ML prediction features within SQL [10, 94, 56], in this section has been evaluated whether execution of predictive pipelines can be pushed inside databases. To do so MASQ has been implemented: a library translating end-to-end ML predictive pipelines into SQL. MASQ currently supports about a dozed of Sklearn and ML.NET operators. In order to evaluate the performance of the approach, MASQ has been tested over MySQL and SQL Server, and using 10 representative ML pipelines and 7 different datasets, and a comparison with Sklern and ML.NET has been provided. The results show that MASQ performance is comparable to Sklearn and ML.NET when the data is originally stored in the database. This outcome proves that predictive pipelines can be served without having to trade-off performance with the "Enterprise-grade" provided by DBMSs. Finally, several interesting compromises between optimizations and compilation strategies have been discovered. This suggests that a cost-based optimizer is likely required to achieve the best performance.

MASQ is currently actively maintained: new operators are being integrated (e.g., feature selection operators, imputers, K-means, missing linear and tree models). Additionally, support for accelerated hardware integration and optimizations such as constant folding, dead code elimination, and operators reordering [79] is being explored.

Chapter 4

Data exploration and explanation

Industry and research communities have always had the prerogative to develop easy-to-use and interactive mechanisms for data management and analysis. This problem is nowadays exacerbated to accommodate the many data enthusiasts who might not always be database-savvy and need to perform data-related activities. A large plethora of approaches facilitates data management such as data debuggers [38, 72], data explanation systems [127, 159], exploratory search systems [129], outlier detectors [161], subgroup discovery systems [14, 61, 63] and so on.

In this chapter, an approach for creating *data descriptions* (descriptions in short), i.e. a form of explanation that aims at making, from a human perspective, a (large) set of data more understandable at a glance, is presented. A description is a compact, readable and insightful structure formed by *predicates* that apply to the target dataset. Predicates are more informative than single tuples and, for this reason, are used extensively within explanation systems [127, 159, 164]. While existing data explanation tools and techniques are used to gain knowledge on specific tasks such as unexpected behaviors of systems [156, 164], query answers [78, 127, 159] and predictions [78, 44], data descriptions are generic inasmuch as they are able to concisely represent any arbitrary set of data tuples. The research described in this chapter has been published in [113], and a presentation of its Web prototype is available in [111].

4.1 Motivating example

Consider the *Sensor* dataset of Figure 4.1a containing sensor measurements, and taken from [159]. Each measurement is a tuple, detected at a certain `Time` from a sensor device identified with a `SensorID`, which consists of `Voltage`, `Humidity` and `Temperature` measures. For the sake of brevity, and by following the order of appearance in Figure 4.1a, the

	Time	SensorID	Voltage	Humidity	Temp.
t_1	11AM	1	2.6	0.4	34
t_2	11AM	2	2.6	0.5	35
t_3	11AM	3	2.6	0.4	35
t_4	12AM	1	2.7	0.3	35
t_5	12AM	2	2.7	0.5	35
t_6	12AM	3	2.3	0.4	100
t_7	1PM	1	2.7	0.3	35
t_8	1PM	2	2.7	0.5	35
t_9	1PM	3	2.3	0.5	35

(a) The *Sensor* dataset.

D_1 : $\texttt{Ti} \in \{11\text{AM}, 12\text{AM}, 1\text{PM}\} \bigwedge \texttt{S} \in \{1,2,3\} \bigwedge \texttt{V} \in \{2.3, 2.6, 2.7\} \bigwedge \texttt{H} \in \{0.3, 0.4, 0.5\} \bigwedge \texttt{Te} \in \{34, 35, 100\}$

D_2 : $\{11\text{AM} \leq \texttt{Ti} \leq 1\text{PM}\} \bigwedge \{2.3 \leq \texttt{V} \leq 2.7\} \bigwedge \{0.3 \leq \texttt{H} \leq 0.5\} \bigwedge \{35 \leq \texttt{Te} \leq 100\} \bigwedge \{1 \leq \texttt{S} \leq 3\}$

D_3 : $\{11\text{AM} \leq \texttt{Ti} \leq 1\text{PM}\} \bigwedge \{0.3 \leq \texttt{H} \leq 0.5\} \bigwedge \{1 \leq \texttt{S} \leq 3\}$

D_4 : $\Big(\texttt{S} = 1 \bigwedge \texttt{H} \in \{0.3, 0.4\}\Big) \bigvee \Big(\texttt{S} = 2 \bigwedge \texttt{H} \in \{0.5\}\Big) \bigvee \Big(\texttt{S} = 3 \bigwedge \texttt{H} \in \{0.4, 0.5\}\Big)$

D_5 : $\Big(\texttt{H} = 0.3 \bigwedge \texttt{V} = 2.7\Big) \bigvee \Big(\texttt{H} = 0.4 \bigwedge \texttt{Te} \in \{34, 35, 100\}\Big) \bigvee \Big(\texttt{H} = 0.5 \bigwedge \{2 \leq \texttt{S} \leq 3\}\Big)$

D_6 : $\Big(\texttt{H} = 0.3 \bigwedge \texttt{Te} = 35\Big) \bigvee \Big(\texttt{H} = 0.4 \bigwedge \texttt{Te} \in \{34, 35, 100\}\Big) \bigvee \Big(\texttt{H} = 0.5 \bigwedge \texttt{Te} = 35\Big)$

D_7 : $\Big(\texttt{H} = 0.4 \bigwedge \texttt{Te} \in \{34, 100\}\Big) \bigvee \Big(\texttt{Te} = 35 \bigwedge \texttt{V} \in \{2.3, 2.6, 2.7\}\Big)$

(b) A subset of possible descriptions for *Sensor*.

Fig. 4.1 Data and descriptions of the *Sensor* running example.

attributes are identified with the short labels `Ti`, `S`, `V`, `H` and `Te`. Assume now a user who needs support for starting the data analysis on *Sensor*. A description is a suitable support for such type of user.

Example 1 *A naive description D_0 of* Sensor *consists of the set of tuples themselves. Each tuple in D_0 is a set of attribute-value pairs. For instance, a description for t_1 is $\{Ti = 11AM, S = 1, V = 2.6, H = 0.4, Te = 34\}$, whereby D_0 includes such description and all the analogously formed descriptions for the remaining tuples $t_2, \ldots, t_9$.*

D_0 is verbose and arguably does not help in understanding the content of *Sensor*. Desirably, when a user asks for a description, she wants an outline of the dataset that is both easy to read and understandable despite the loss of information due to summarization.

Example 2 *D_1 of Figure 4.1b is a description conjoining a list of predicates, each characterizing an attribute of the dataset with the related set of values. D_1 is more concise than D_0 since it removes duplicates at the expense of reconstructing the original tuples.*

When we have complex schemas and instances, a description like D_1 is too long and still difficult to read. A shorter description is preferable. In this case, the selection of the features and of the domain values for the predicates is a crucial task.

Example 3 *D_1 can be compacted by using other forms of predicates. For instance, D_2 of Figure 4.1b increases the readability of continuous attributes by using range predicates. D_2 can be additionally shortened by limiting the set of attributes being employed in the predicates composing the description. For instance, D_3 uses `Ti`, `H` and `S` attributes only.*

In other scenarios, users are interested in understanding the values assumed by specified attributes and how they vary with respect to the rest of the dataset. In this case, the attributes of interest constitute the *pivotal elements* of the descriptions.

Example 4 *D_4 and D_5 show examples of descriptions partitioned over pivotal elements. D_4 shows the values assumed by the attribute `H` for each sensor. In D_5, the user is interested in the values registered by the sensors when `H` changes.*

While generating a description per se is relatively easy, the problem of generating all descriptions is exponential whereby searching for the "best description" becomes a challenging task.

In this regard, the first challenge is to understand whether a description is relevant to a user. To solve this challenge a set of dimensions for qualifying different aspects of the descriptions will be introduced. They correspond to a set of tunable parameters that users can control to suggest their intent to the system.

Example 5 *Assume that a user indicates that she wants descriptions with many attributes. Among the descriptions in Figure 4.1b the preferable description is D_2. In other situations, a user might be interested in understanding what happens at different levels of* `Humidity`*. D_5 and D_6 best suit this purpose. If additionally the user wants a low number of distinct attributes, then D_6 should be preferred over D_5.*

Interestingly, some dimensions are somehow in contrast with each other, e.g., a short description is understandable at the expenses of losing a significant chunk of information. The proposed approach exploits a relevance function that balances all dimensions to solve this issue.

Another challenge is related to the large number of descriptions it is possible to choose from: for accommodating the considered exploratory use cases, good descriptions must be rendered at interactive speed. Clearly, a naïve solution that exhaustively evaluates all possible descriptions compromises the usability of the approach. To solve this second challenge, a dynamic programming approach with a set of heuristic rules, in concert with a mechanism for generating and ranking the top-k descriptions, is considered.

4.2 Data Descriptions

This section formalizes data descriptions and introduces the problem of generating the best descriptions. Let us consider a dataset I composed of tuples over a set of attributes $A = \{a_1, a_2, \ldots, a_n\}$, where each attribute a_i has associated a domain of atomic values. A tuple t is in the form $t = \langle v_1, v_2, \ldots, v_n \rangle$ where each value v_i is contained in the respective domain. Finally, given a set of tuples I, $adom_I(a_i)$ will denote the active domain of attribute a_i for I. Descriptions use predicates defined as follows.

Definition 1 (Predicate) *Given a dataset I over A, a predicate p is an atomic formula over A in one of the following forms:*

- SET PREDICATE $p : a_i \in \{v_1, v_2, \ldots, v_m\}$, *where* $a_i \in A$ *and each* $v_i \in adom_I(a_i)$.
- RANGE PREDICATE $p : (v_l \leq a_i \leq v_u)$, *where* $a_i \in A$ *and both* $v_l, v_u \in adom_I(a_i)$.

When a set predicate has only one value, it is considered as *atomic* and the notation $p : (a = v)$ will be used, e.g., $\mathtt{S} = 1$ of D_4 in Figure 4.1b.

Definition 2 (d-formula) *Given a set of tuples $I \neq \emptyset$ over A, a d-formula d for I is a conjunction of predicates over A, such that:*

1. *an attribute in A occurs in d at most once;*
2. *each predicate $p \in d$ is true in I according to the classic interpretation of atomic formulas in first order logic;*
3. *(*full coverage*)* $\forall t_i \in I$, *d is true for t_i.*

Descriptions are usually computed over (horizontal) *partitions* of a dataset I. A partition is a non-empty set of tuples P_i such that $I = \bigcup_i P_i$ and $\forall i, j$, $P_i \cap P_j = \emptyset$;

Definition 3 (Description) *Let $I \neq \emptyset$ be a dataset over A, and organized into ρ partitions, such that for each partition $i \in 1,...,\rho$ a d-formula d_i over A exists. A description D for I is a non-empty disjunctive-normal formula of ρ d-formulas $\{d_1,\ldots,d_\rho\}$.*

Given an input dataset I, the goal of this approach is to generate the *best* descriptions representing I. The quality of a description is supported by a *scoring function*.

Definition 4 (Scoring function) *Given a set of descriptions $\{D_1,\ldots,D_n\}$, a scoring function Cost is such that $\forall i$, $Cost(D_i) \geq 0$, and for each pair D_i, D_j, if $Cost(D_i) \leq Cost(D_j)$ then D_i is more relevant than D_j.*

In practice, the function *Cost* may take into consideration several factors, such as user preferences, and can be personalized depending on the domain. The list of the user preferences supported by the approach are listed in Section 4.3 and the implementation of the adopted scoring function is provided in Section 4.4.3. In the following, it is demonstrated that the general problem of generating the best description for a dataset I is NP-Hard.

Problem 4.2.1 (Finding the best description) *Given a dataset I organized into ρ partitions, and a PTIME-computable scoring function Cost, let $\mathscr{D} = \{D_1, D_2, ..., D_m\}$ be the set of all possible descriptions over I. The computation of the most relevant description in $\mathscr{D}$ aims at returning the description $D_i \in \mathscr{D}$ such that $Cost(D_i) \leq Cost(D_j), \forall D_j \in \mathscr{D}$.*

Theorem 1 *Problem 4.2.1 is NP-Hard.*

Proof. The demonstration is carried out by showing that there exists a polynomial-time reduction from the Weighted Set Cover (WSC) Problem to Problem 4.2.1. The WSC Problem, which will be recalled next, is a well-known NP-Hard problem [32].

Problem 4.2.2 (Weighted set cover problem) *Given a universe of n elements $\mathscr{U} = \{e_1,\ldots,e_n\}$, a set of m non-empty subsets of $\mathscr{U}$, $\mathscr{S} = \{s_1,\ldots,s_m\}$, and a PTIME cost function $c : \mathscr{S} \to \mathbb{R}^+$. The goal is to find the minimum cost subset $x \subseteq \mathscr{S}$ such that:*

1. *all elements are covered by x (i.e. $\forall e_i \in \mathscr{U}, e_i \in \bigcup_{s_j \in x} s_j$), and*
2. *the sum of the costs of the elements in x is minimized.*

Given an instance $(\mathscr{U}, \mathscr{S}, c)$ of Problem 4.2.2, let us define a mapping reduction r that creates a dataset $I = \{t_1, t_2, \ldots, t_n\}$ over a set of attributes $A = \{a_1, a_2, \ldots, a_m\}$:

- using a bijective function $g : \mathscr{U} \to I$ that associates each element e_i of the universe to a tuple t_i of I;
- using a bijective function $f : \mathscr{S} \to A$ that associates each s_j to an attribute a_j;
- each value $v_{i,j}$ for the i-th tuple and j-th attribute is s.t.:

$$v_{i,j} = \begin{cases} \top, & \text{if } g^{-1}(t_i) \in f^{-1}(a_j) \\ \bot, & \text{otherwise.} \end{cases}$$

It follows that each possible description $D_k \in \mathscr{D}$ over I is constructed out of boolean predicates in the form $p : (a_i = \top)$ or $p : (a_i = \bot)$. Additionally, r introduces a scoring function Cost$'$ that sums over the costs of each predicate p_i in a description D_k, i.e., Cost$'(D_k) = \sum_{p_i \in D_k}$Cost-p$(p_i)$, where:

$$\text{Cost-p}(p_i : (a_i = v)) = \begin{cases} c(f^{-1}(a_i)), & \text{if } v = \top \\ \infty, & \text{otherwise.} \end{cases}$$

Finally, a function h, that transforms a d-formula with boolean predicates computed over an instance created by r into a candidate solution to Problem 4.2.2, is also introduced: $h(D_k) = \{s_k : s_k = f^{-1}(a_j), \forall (a_j = \top) \in D_k\}$. It is straightforward to see that with r and h we have that Cost$'(D_k) = c(h(D_k))\ \forall D_k \in \mathscr{D}$, where $\mathscr{D}$ is the set of all possible descriptions over I.

Now it is possible to solve the Problem 4.2.1 over dataset I considering all the tuples grouped into the same partition[1] (i.e., $\rho = 1$). The solution is a description D formed by one d-formula d. By construction all predicates in d are in the form $p : (a_i = \top)$ and d (D) is true for each tuple $t_i \in I$ (full coverage). The next step is to show that if the solution D has a finite cost, then $X = h(D)$ is the optimal solution for Problem 4.2.2. Otherwise, there is no solution for Problem 4.2.2. This can be proved by considering the two clauses of Problem 4.2.2 separately. If a finite solution to Problem 4.2.1 exists, then the first clause is true since (1) all the predicates in d are positive (i.e. $a_i = \top$), and (2) $\nexists t_i \in I$ such that D is false for t_i. Consequently, all the tuples have at least one value that is true and it is captured by the description D. By construction of r, it follows that $\nexists e_i = g^{-1}(t_i)$ such that $e_i \in X$.

In order to show the clause of minimality, let us assume, by contradiction, that another solution X' is optimal for the instance $(\mathscr{U}, \mathscr{S}, c)$ of Problem 4.2.2 (i.e. $c(X') < c(X)$). Therefore, a description D' over I must exist such that $X' = h(D')$ and Cost$'(D') <$ Cost$'(D)$. This is impossible because it is assumed that D is the minimal solution to Problem 4.2.1. Hence X must be minimal.

[1] Note that this is a simplification: similar arguments hold for the cases where $\rho > 1$.

Since (1) Problem 4.2.2 is NP-Hard, (2) the reduction r is polynomial (i.e., $\mathscr{O}(|\mathscr{U}| \times |\mathscr{S}|)$), (3) Cost$'$ is in PTIME, and (4) the transformation h for generating D from X has linear time-complexity (i.e., $\mathscr{O}(|\mathscr{S}|)$), it is possible to conclude that generating the best description D is also NP-Hard. □

4.3 Generating Descriptions: Principles

Attributes of a dataset carry different degrees of relevance for users interested in gleaning insights from data. Some attributes have intrinsic value because, for example, they can identify entities in a domain (e.g., the attribute `SensorID` in the *Sensor* dataset). Sometimes, the relevant attributes are indicated by the users due to their domain knowledge or to a specific interest. For example, a user may be interested in understanding when a specific attribute, e.g., the `Temperature`, assumes certain values in a sensor. For this reason, it is more convenient to see the dataset *Sensor* partitioned by `SensorID` instead of a monolithic entity. *The first driving principle states that it is more explicative to think of a dataset as the composition of different groups of related tuples*. By following this principle, the implemented approach will have to let the user specify how the dataset has to be partitioned, and will have to generate a specific d-formula to describe each partition in the dataset. The set of d-formulas is the final description for the dataset.

Data explanation is conducted for different purposes. In some cases, users want an accurate and complete, yet readable, representation of the whole dataset. In other cases, a general profile of the dataset is enough for the user. The description represents, in these latter cases, an overview that ignores infrequent values. Often, instead, there are users who are interested in finding outliers. *The second driving principle is that multi-faceted goals of data explanation can be accommodated by relaxing the concept of full coverage in the descriptions (Definition 2)* [2]. By allowing users to interactively change the coverage of the expected descriptions (intended as the percentage of the total number of tuples that are true for the description) they are now capable of extracting proper insights from data. For example, let us imagine a user that wants to separately analyze each sensor in the *Sensor* dataset. Initially, she sets a high coverage to get a better picture and she gets that `Humidity` varies between 0.3 and 0.4 for `SensorID` 1, it is 0.5 in `SensorID` 2 and varies between 0.4 and 0.5 for `SensorID` 3. She then decides to dig deeper by lowering the coverage, and she finds that sensor 3 has `Temperature` equal to 100, which is an outlier.

Based on this considerations, it is possible to conclude it is impossible to define a single and global way to assess how good a description is since the quality is influenced by the

[2]Note that this relaxation does not invalidate the formalization of Section 4.2.

subjectivity of the end users, e.g., a prolix description can be suitable for a user but, at the same time, be poor for another one. *The third driving principle is to consider user preferences for qualifying descriptions.* This principle will have to be implemented by: (1) defining a series of dimensions that characterize the descriptions; (2) let the users indicate their preferential value for these dimensions; and (3) take into account the user's preferences in the scoring and ranking of the final descriptions (Section 4.4.3).

Specifically, three dimensions for characterizing a description D from a user perspective have been identified: *coverage*, *degree*, and *diversity*.

Definition 5 (Coverage) *Given a subset X of the dataset I for which a d-formula is true, the coverage of a d-formula is the fraction* $\frac{|X|}{|I_\rho|}$ *i.e., the percentage of I over the partition* ρ *for which the d-formula is true. Given a description D, the coverage of D, i.e.* $cov(D)$*, is the mean of the coverage of the d-formulas therein.*

Basically, broad dataset descriptions have a high coverage; descriptions of datasets' peculiarities (e.g., outliers) have low coverage. The proposed approach will use this definition of coverage to validate descriptions instead of the (more strict) one of Definition 2.

Definition 6 (Degree) *Given a description D over a dataset I with A attributes and composed by* ρ *partitions, let* $A_{d_i} \subseteq A$ *be the set of attributes in a d-formula* d_i *of D. The degree of a description is the average number of attributes in its constituting d-formulas:* $deg(D) = \frac{1}{\rho}\sum_{d_i \in D} \frac{|A_{d_i}|}{|A|}$.

Intuitively, a description with lower *degree* tends to be much easier to read.

Definition 7 (Diversity) *Given a description D, let* $||D||$ *be the overall number of predicates in D and* $A_D \subseteq A$ *be the set of attributes that appear in the predicates of D at least once. The diversity is measured as:* $div(D) = \frac{|A_D|}{||D||}$.

The *diversity* measures how often attributes are shared across d-formulas. While repetitions of a set of attributes in different partitions increase readability, they could reduce the fine-grained representation of partitions.

Example 6 *With reference to the descriptions reported in Figure 4.1b,* $deg(D_1) = 1$*,* $div(D_1) = 1$*;* $deg(D_3) = 3/5$*,* $div(D_3) = 1$*;* $deg(D_6) = 6/15$*, and* $div(D_6) = 2/6$ *All these descriptions have the maximal coverage (*$cov(D_1) = cov(D_3) = cov(D_6) = 1$*).*

These three dimensions will be considered as the minimal set of parameters that allows to both (1) extract enough information from users for enabling task-specific explorations; and (2) allow the system to return high-quality descriptions. This set of dimensions has been identified through experimental evaluation and by reviewing similar approaches (e.g., [68, 78]).

Parameter	Meaning
COV	The desired coverage. It is a float value between 0.0 and 1.0.
DEG	The desired degree. It is a float value between 0.0 and 1.0, where 1.0 stands for high-degree descriptions.
DIV	The desired diversity. It is a float value between 0.0 and 1.0, where 1.0 stands for high-diverse descriptions.
A_x	In the *user-driven* mode, the attributes of interest for the user, and $\|A_x\|$ the size of the description.
ρ	The expected number of partitions. In the *data-driven* mode, this is the number of clusters to be generated.
k	The maximum number of descriptions to be returned.

Table 4.1 Users' preferences used as input to the framework.

4.4 The Approach

The description generation process is interactive: users can specify their preference parameters (see Table 4.1) repeatedly until the explanation need is fulfilled.

The generation of the descriptions is performed over 3 phases, as shown in Algorithm 1. In the first phase (line 1) the input dataset is partitioned. Data partitioning is detailed in Section 4.4.1. The generation of the d-formulas happens during the second phase (line 2) [3]. For each partition, the number of possible d-formulas is exponential over the number of attributes' values. Generating all possible d-formulas is, therefore, prohibitively expensive. As explained in Section 4.4.2, a heuristic procedure that allows to prune d-formulas that are less relevant for the task at hand will be introduced. In the last phase (line 3), the actual descriptions are computed by combining d-formulas of different partitions. Intuitively, the top-k descriptions that minimize the scoring function are obtained by combining the generated d-formulas.

This problem can be solved with a dynamic programming approach: the proposed approach, as described in Section 4.4.3, will employ a variant of the Viterbi Algorithm called *LVA* [139] (a.k.a. *List Viterbi*), although any other algorithm in this class can be used. In

[3]Note that when numerical attributes exists in a dataset, a discretization algorithm is applied during the second phase to extract categorical features from numerical attributes (e.g., [51]). An inverse process is applied to the descriptions before returning them to the user. Descriptions with d-formulas based on categorical attributes generated by discretization, in a post-process phase are transformed by interpolation over range predicates (Definition 1).

this specific implementation, LVA takes as input (1) the set of d-formulas generated in the previous phase (DF in Algorithm 1), (2) the user provided values for degree and coverage, and (3) the desired value for the parameter k.

Algorithm 1: Generation of Descriptions

Input : The dataset I, A_x, COV, DEG, DIV, k, and ρ.
Output : The top-k descriptions D_k.

1 $P \leftarrow CreatePartitions(I, A_x, \rho)$;
2 $DF \leftarrow GetDFormulas(P, A, \text{COV}, \text{DEG})$;
3 $D_k \leftarrow ViterbiTopK(DF, \text{DEG}, \text{DIV}, k)$;
4 **return** D_k;

4.4.1 Building partitions

The first phase of the adopted technique splits the input dataset into partitions. Since partitions are described separately, it is desirable to create them such that similar tuples are grouped together. Users interact with the system in two different modes: *user-driven* and *data-driven*. The *user-driven* mode is useful when the task is to profile data over some feature of interest. The *data-driven* mode is instead more suited for cases where a user has little knowledge of the dataset, and the partitions are created in a fully-automated fashion via a clustering algorithm. In *user-driven* mode, the approach creates a partition for every distinct value of the projection of I over a non-empty set of user provided attributes $A_x \subseteq A$. In this case, the procedure applies a SQL's `group-by` operator over the pivotal attributes specified by the A_x parameter.

Example 7 *Looking at Figure 4.1b, D_4 is a user-driven description in which the user pinpointed* `SensorID`*; D_5 is another user-driven description but built around* `Humidity`*. For the latter the partitions are $P_1 = \{t_4, t_7\}$, $P_2 = \{t_1, t_3, t_6\}$, and $P_3 = \{t_2, t_5, t_8, t_9\}$.*

In *data-driven* mode, the partitioning logic ensues from the application of a clustering algorithm. The approach uses *k-means* and let the user specify the number of clusters (i.e. the ρ parameter). However, any clustering algorithm can be used.

Example 8 *In Figure 4.1b D_7 is a data-driven description composed of partitions: $P_1 = \{t_1, t_6\}$ and $P_2 = \{t_2, t_3, t_4, t_5, t_7, t_8, t_9\}$.*

4.4.2 Building d-formulas

In the second phase, the approach builds all feasible and relevant d-formulas for each partition. The number of candidate predicates for each partition depends on the size of the active

domain of the attributes in the partition. Therefore, the complexity of this phase is $\mathcal{O}\left(\rho \times \sum_{k=1\ldots AD_p} \binom{AD_p}{k}\right)$, where ρ is the number of partitions, AD_p is the cardinality of the union of the active domains of all attributes in partition p. In other words, $AD_p = |\bigcup_{a\in A} adom_p(a)|$, where A is the set of the attributes, and $adom_p(a)$ is the active domain of attribute a in p. Note that the complexity of this phase is exponential, since $\sum_{k=1\ldots AD_p} \binom{AD_p}{k} = 2^{AD_p}$.

Given the complexity for generating all possible d-formulas (which makes the generation of description at reasonable speed unfeasible), a heuristic process that generates the most relevant candidate d-formulas only is adopted. We consider two heuristics, one for pruning prolix d-formulas, and one for pruning d-formulas and predicates with undesirable selectivity. Section 4.5 (e.g., Table 4.7) will show that these heuristics allow the number of generated d-formulas to be decreased by up to 5 orders of magnitude (e.g., 28 d-formulas versus 1431480 for a coverage of 0.8).

Heuristic 1 - pruning prolix d-formulas A low degree is specified when the user prefers a small number of predicates and, conversely, high degree is for users who prefer descriptions with wide d-formulas. Given that a description with too many predicates is somehow hard to read, the degree intent of the users is pushed into the process of building d-formulas in order to early-prune d-formulas that do not meet such requirement. This limits the number of predicates to evaluate and consequently it improves the efficiency of the entire process.

Heuristic 1 is implemented through another parameter, called *conciseness* (or CONC) whose value dictates the maximum number of atomic predicates allowed in a d-formula. More precisely, $\text{CONC} = e^{\lambda * \text{DEG}}$. The exponential function models the perception of the user according to which small variations in the number of predicates of a low-degree description have a significant impact on its legibility. In the following experiments, $\lambda = 6$ in order to limit the number of atomic predicates in a d-formula to 400 when DEG is the highest. Thanks to this heuristic, the complexity of the phase becomes $\mathcal{O}\left(\rho \times \sum_{k=1\ldots\text{CONC}} \binom{AD_p}{k}\right) < \mathcal{O}\left(\rho \times \sum_{k=1\ldots AD_p} \binom{AD_p}{k}\right)$, when $\text{CONC} < AD_p$ (that is the expected usual scenario).

Heuristic 2 - pruning d-formulas and predicates with undesirable selectivity. The parameter COV indicates the desired percentage of tuples that make a d-formula true. Heuristic 2 transforms COV into an interval $[\text{COV}_l, \text{COV}_u]$ of admissible values of coverage. The width of the interval is proportional to the COV itself given that $\text{COV}_l = \text{COV} - \mathit{offset}(\text{COV})$, $\text{COV}_u = \text{COV} + \mathit{offset}(\text{COV})$ and $\mathit{offset}(\text{COV}) = \alpha * \text{COV}^2$. This supports different use cases,

e.g. outlier detection with small coverage or broad descriptions with high coverage when no much details is needed. The approach uses $\alpha = 0.14$ as it maximizes the quality metrics, as shown in the experimental evaluation (see Section 4.5).

The adopted approach exploits the above intuition so that it can early prune predicates (before they compose into d-formulas) if they do not meet the expected coverage. A naïve way to early identify admissible predicates is to check if their coverage is within the interval $[\text{COV}_l, \text{COV}_u]$. However, discarding predicates with coverage out of this interval limits the d-formulas computed to the combinations of those predicates with coverage close to the COV value only. This appears to be too selective and some data distributions can generate a low number of d-formulas (or even no d-formula). The interval width is thus enlarged by using $\text{COV}_p = \frac{\text{COV} - \mathit{offset}(\text{COV})}{\text{CONC}}$ instead of COV_l as left bound. This makes the coverage value depending on the conciseness, so that when CONC is high (wider descriptions are required), the interval width increases. By reducing the active domains of the attributes in the partitions, the impact of Heuristic 2 is a reduction of the numerator of the binomial function. The final complexity becomes $\mathcal{O}\left(\rho \times \sum_{k=1\ldots\text{CONC}} \binom{AD'_p}{k}\right)$, where AD'_p is the cardinality of the union of all active domains after the application of Heuristic 2 and $AD'_p < AD_p$.

Algorithm 2: `GetDFormulas`

Input : A list of partitions P, the dataset attributes A, COV and DEG.
Output : The d-formulas **d**.

```
1  d ← ∅;
2  CONC = e^λ *DEG;
3  COV_l ← COV − offset(COV);
4  COV_u ← COV + offset(COV);
5  COV_p ← COV_l / CONC;
6  foreach p ∈ P do
7      Θ ← ∅;
8      foreach a ∈ A do
9          foreach v ∈ adom_P(a) do
10             if |σ_{a=v}(p)| / |p| ∈ [COV_p, COV_u] then
11                 Θ ← Θ ∪ {a = v}
12     d ← d ∪ ValidDFormulas(Θ, CONC, COV_l, COV_u, p);
13 return d;
```

Generating d-formulas. Now it is described the routine for generating all possible and relevant d-formulas. Algorithm 2 takes in input the list of partitions (P), the list of dataset attributes (A), the coverage (COV) and the degree (DEG). The set **d** of the d-formulas (line 1) and, as per the adopted heuristics, the conciseness (line 2) and the desirable coverage

Algorithm 3: `ValidDFormulas`

Input : A set of predicates Θ, CONC, COV_l, COV_u, and a partition p.
Output : The d-formulas with coverage between COV_l and COV_u for partition p.

```
1  Θ* ← Θ, n ← 0;
2  repeat
3      Θ' ← ∅;
4      foreach θ_i ∈ Θ do
5          Θ_i ← GetDistinctPredicates(θ_i, Θ);
6          foreach θ_j ∈ Θ_i do
7              d ← θ_i ⊕ θ_j;
8              if |σ_d(p)| / |p| ∈ [COV_l, COV_u] then
9                  Θ' ← Θ' ∪ d;
10                 Θ* ← Θ* ∪ d;
11     Θ ← Θ';
12 until |Θ| = 0 or n = CONC;
13 return Θ*;
```

intervals (lines 3-5) are initialized. The d-formulas of each partition (i.e. Θ) are computed separately (lines 6-11), and the "atomic predicates" (i.e. $(a = v)$) for each distinct value v of a in the partition p are generated(line 11). Atomic predicates are created only if they are within the expected coverage (i.e., $[\text{COV}_p, \text{COV}_u]$) as defined by Heuristic 2 (line 10). Atomic predicates are combined together to generate conjunctions of predicates (line 12). The resulting d-formulas, for each partition, are then returned as output (line 13).

The `ValidDFormulas` procedure of Algorithm 3 combines together predicates and returns only those combinations (i.e., d-formulas) that are valid. The combination of the input predicates Θ is organized in a lattice. The main loop of lines 2-12 dynamically generates the lattice. In each iteration, one level of the lattice (the n-th) is generated by combining pairs of the previously created predicates (lines 4-10). One predicate might get combined with another predicate many times. For this reason, the function `GetDistinctPredicates` is used to efficiently selects a combinable predicate only once (line 5). Pairs of predicates are combined with the operator $\oplus$ (line 7). The combination is a d-formula that is either: (1) a new predicate created from the union of two predicates on the same attribute (e.g., $\{a = 4\} \oplus \{a = 5\} = \{a \in \{4,5\}\}$); or (2) a conjunction of predicates on different attributes (e.g., $\{a \in \{4,5\}\} \oplus \{b \in \{5,6\}\} = \{a \in \{4,5\} \wedge b \in \{5,6\}\}$). $\sigma_d(p)$ are the tuples covered by the d-formula d in partition p. Only combinations of predicates in the desired interval of coverage values $[\text{COV}_l, \text{COV}_u]$ are considered (line 8) for the next level of the lattice (line 9 and line 11) and in the list of final results Θ^* (line 10). The generation of the lattice ends when the threshold, as per Heuristic 1, is reached ($n = \text{CONC}$) or when it is no longer possible to generate predicates ($|\Theta| = 0$).

4.4.3 Building top-k descriptions

Selecting the best d-formulas in isolation does not necessarily lead to the best descriptions. It is needed to search for the optimal set of d-formulas across partitions that all together minimize the cost. Given the complexity of the problem, a dynamic programming technique is adopted. It relies on *LVA* [139], which is a generalization of Viterbi Algorithm [49] for top-k solutions. The complexity of *LVA* is $\mathcal{O}(\rho \times k \times d^2)$, where ρ is the number of partitions, k is the number of top elements to retrieve and d is the number of d-formulas. Viterbi Algorithm and its variants require to model the search space as a trellis. The trellis is built in the following way. A vertical slice for each partition is firstly created. The nodes in the slice are the d-formulas found in the previous phase. Nodes of a slice are connected to all the nodes of the next slice via weighted directed edges. A path represents a list of d-formulas, with at most one d-formula for partition. The algorithm is used to find the best full path, that is a path that starts in the first slice and terminates in the last slice of the trellis. Note that, for the final results, the order of the slices is irrelevant since all nodes of a slice are connected to all the nodes of the next slice.

Viterbi needs an objective function w to score a path, which intuitively represents the score of the (intermediate) descriptions we are computing. Given a trellis composed of N nodes and S slices, the score of a path terminating in a node j of a slice s of the trellis (i.e., $w_j(s)$) can be recursively computed. The best description D^* is the one minimizing the score of the node in the final slice S: $Cost(D^*) = \min_{1 \le i \le N} [w_i(S)]$.
The description D^* is constructed by backtracking the above computation, which determines the nodes in the corresponding path. *LVA* generalizes the approach to compute the top-k solutions. The function w used to (recursively) score a path is reported below:

$$w_j(s) = \begin{cases} \min\limits_{1 \le i \le N} [w_i(s-1) \ * L_{ij} * \Delta_{div}(i,j)] * \Delta_{deg}(j) * \ H(j), & \text{if } s > 1 \\ \Delta_{deg}(j) * \ H(j), & \text{if } s = 1 \end{cases} \tag{4.1}$$

L is the adjacency matrix of the trellis, where $L_{ij} = 1$ if there is an edge between node i and node j, otherwise $L_{ij} = +\infty$. $H(j)$ is a measure of *entropy* for evaluating how discriminative attributes are. It is computed as the normalized *Shannon entropy* value[4] of the attributes used by the d-formula associated to node j. Δ_{div} and Δ_{deg} score diversity and degree in terms of adherence to the preferences of the user DIV and DEG, respectively. More specifically, Δ_{deg} is defined as $\Delta_{deg}(d) = 1 - h(deg(d))$, where h is the Normal Distribution function centered on DEG and with variance equal to 0.2. The variance is chosen experimentally. In this way, a

[4]The standard definition of normalized Shannon entropy for a probabilistic variable X in a finite domain $\{x_i\}$ is used, that is $H(X) = \frac{-\sum_{i=1}^{N} P(x_i) * \log_2 P(x_i)}{\log_2 N}$.

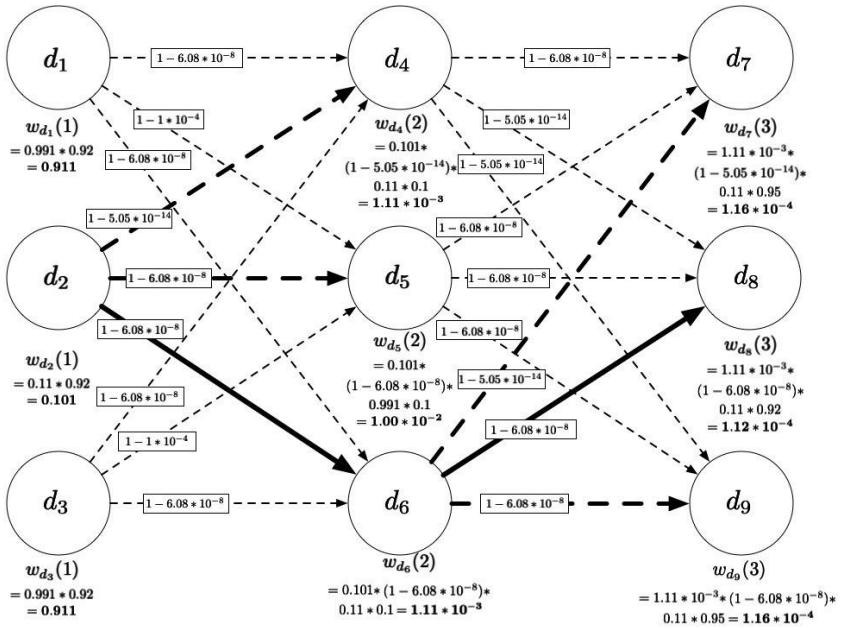


Fig. 4.2 The Viterbi Algorithm applied to *Sensor*.

smaller cost is assigned to descriptions whose degree is close to DEG. Instead, Δ_{div} considers a pair of d-formulas d_i and d_j, out of which the partial description D' that includes both of them is computed. Then, analogously to the degree factor, the diversity factor is defined as $\Delta_{div}(d_i, d_j) = 1 - g(div(D'))$, where g is the Normal Distribution function centered on DIV with variance equal to 0.2.

Basically, the scoring function measures (1) the adherence between the features of the description and the expectation of the user; and (2) the goodness of the chosen attribute in describing the partitions.

COV is omitted from Equation 4.1 since Heuristic 2 already makes sure that only descriptions with the desired coverage are computed.

Example 9 *Let us consider the dataset* Sensor *described in Table 4.1a and a user interested in descriptions explaining the behavior of each sensor (i.e., A_x = {`SensorID`}). The adopted partitioning mechanism (see Section 4.4.1) divides the dataset into three partitions, one for each sensor. Table 4.2 reports some d-formulas (see Section 4.4.2) along with their Δ_{deg} and entropy values that are used by* LVA *for computing the descriptions. Figure 4.2 shows the application of the Viterbi to the d-formulas in Table 4.2. The trellis is divided into three vertical slices corresponding, from the left-side, to the partitions for* `SensorID = 1`, `SensorID = 2` *and* `SensorID = 3`. *Each node represents one of the d-formulas. The weights assigned to the nodes have been computed according to Equation 4.1. Note that nodes in the first slice are computed following the formula of the initialization weight; while*

SensorID =1	SensorID =2	SensorID =3
d_1 = Te $\in \{35\} \wedge$ H $\in \{0.3\}$ $\Delta_{deg} = 0.991$ $H = 0.92$	d_4 = H $\in \{0.5\}$ $\Delta_{deg} = 0.11$ $H = 0.1$	d_7 = H $\in \{0.4\}$ $\Delta_{deg} = 0.11$ $H = 0.95$
d_2 = Te $\in \{35\}$ $\Delta_{deg} = 0.11$ $H = 0.92$	d_5 = Te $\in \{35\} \wedge$ H $\in \{0.5\}$ $\Delta_{deg} = 0.991$ $H = 0.1$	d_8 = Te $\in \{35\}$ $\Delta_{deg} = 0.11$ $H = 0.92$
d_3 = Te $\in \{34,35\} \wedge$ H $\in \{0.3,0.4\}$ $\Delta_{deg} = 0.991$ $H = 0.92$	d_6 = Te $\in \{35\}$ $\Delta_{deg} = 0.11$ $H = 0.1$	d_9 = Te $\in \{35, 100\}$ $\Delta_{deg} = 0.11$ $H = 0.95$

Table 4.2 A sample of d-formulas generated from *Sensor*.

COV	DEG	DIV	Description
0.2 (LOW)	0.1 (LOW)	0.1 (LOW)	({S = 1 $\wedge$ V $\in$ {2.6} (33.33%)}) $\vee$ ({S = 2 $\wedge$ V $\in$ {2.6} (33.33%)}) $\vee$ ({S = 3 $\wedge$ V $\in$ {2.6} (33.33%)})
0.2 (LOW)	0.1 (LOW)	0.9 (HIGH)	({S = 1 $\wedge$ H $\in$ {0.4} (33.33%)}) $\vee$ ({S = 2 $\wedge$ V $\in$ {2.6} (33.33%)}) $\vee$ ({S = 3 $\wedge$ Te $\in$ {100} (33.33%)})
0.2 (LOW)	0.9 (HIGH)	0.1/0.9 (LOW/ HIGH)	({S = 1 $\wedge$ Te $\in$ {34} $\wedge$ Ti $\in$ {11} $\wedge$ V $\in$ {2.6} $\wedge$ H $\in$ {0.4} (33.33%)}) $\vee$ ({S = 2 $\wedge$ Ti $\in$ {11} $\wedge$ V $\in$ {2.6} (33.33%)}) $\vee$ ({S = 3 $\wedge$ Te $\in$ {100} $\wedge$ Ti $\in$ {12} (33.33%)})
0.8 (HIGH)	0.1 (LOW)	0.1 (LOW)	({S = 1 $\wedge$ Te $\in$ {35} (66.67%)}) $\vee$ ({S = 2 $\wedge$ Te $\in$ {35} (100.0%)}) $\vee$ ({S = 3 $\wedge$ Te $\in$ {35} (66.67%)})
0.8 (HIGH)	0.9 (HIGH)	0.1/0.9 (LOW/ HIGH)	({S = 1 $\wedge$ H $\in$ {0.3} $\wedge$ Ti $\in$ {1, 12} $\wedge$ V $\in$ {2.7} $\wedge$ Te $\in$ {35} (66.67%)}) $\vee$ ({S = 2 $\wedge$ H $\in$ {0.5} $\wedge$ Ti $\in$ {1, 12} $\wedge$ V $\in$ {2.7} $\wedge$ Te $\in${35} (66.67%)}) $\vee$ ({S = 3 $\wedge$ H $\in$ {0.5,0.4} $\wedge$ Ti $\in$ {1} $\wedge$ V $\in$ {2.3, 2.6} $\wedge$ Te $\in$ {35} (66.67%)})
0.8 (HIGH)	0.1 (LOW)	0.9 (HIGH)	({S = 1 $\wedge$ V $\in$ {2.7} (66.67%)}) $\vee$ ({S = 2 $\wedge$ H $\in$ {0.5} (100.0%)}) $\vee$ ({S = 3 $\wedge$ Te $\in$ {35.0} (66.67%)})

Table 4.3 Descriptions with different users' preferences.

for each node in the other slices, only one incoming edge (the one minimizing the weight) is kept. Δ_{deg} *values (shown in Table 4.2) and* Δ_{div} *values (shown in Figure 4.2 on the edges) have been computed taking into account the users' preferences, that is for high coverage, low degree and low diversity. Once the trellis is completed, the last slice is analyzed and the top-k nodes minimizing the weight are selected. The highest ranked path is* $d_2 \rightarrow d_6 \rightarrow d_8$ *with a score of* $1.12E-4$.

Table 4.3 shows some descriptions generated by varying users' preferences. The first three descriptions describe low coverage settings. In the first case, the attribute `Voltage` has been used for all the d-formulas since the desired diversity level is low. In the second case, three different attributes are used for describing the sensors. In the third example, a description with a high degree is shown. Due to the low number of attributes in the dataset, the same description is obtained when the user selects low and high diversity. The other

Name	# Entries	# Features	# Partitions
Scenario 1 – SENSOR	2.3M	9	24 balanced
Scenario 2 – CRIME	76k	9	30 unbalanced
Scenario 3 – MUSHROOM	8k	23	2 balanced
Scenario 4 – SENSORRANDOM	2.3M	9	24 random

Table 4.4 Scenarios.

descriptions have high coverage and different levels of degree and diversity. The actual coverage of the d-formulas in the corresponding partition is enclosed between parentheses.

4.5 Experimental evaluation

In this section, the proposed approach has been evaluated over *efficiency* (Section 4.5.1), *objective effectiveness* using a list of quantitative measures (Section 4.5.2), and *subjective effectiveness* using the judgment of the human-in-the-loop (Section 4.5.3). A demonstration of the developed Python prototype is available at [112].

Configuration. All the experiments are performed on a machine running Ubuntu 12.04 with 16 processors, 128 GB of RAM and a 1 TB of storage.

Datasets. In the experiments, three datasets (*Sensor* [5], *Crime* [6], *Mushroom* [7]) with complementary characteristics are considered, as reported along with the test scenarios in Table 4.4.

Test Scenarios. The approach is evaluated in four test scenarios to highlight its behaviour in different use cases. By default, numeric attributes are transformed into categorical via data binning with 100 equal width bins.

Scenario 1 – SENSOR. This scenario evaluates the explanation for a large dataset (i.e. *Sensor*) with a small number of (numeric) features partitioned on a large number of equal-sized classes. The dataset has 24 equal-sized partitions.

Scenario 2 – CRIME. This scenario has the goal of evaluating the explanations in a dataset (*Crime*) with few features (both numeric and categorical) partitioned on a large number of unbalanced classes (average size of 2458, standard deviation is 4225).

Scenario 3 – MUSHROOM. This scenario aims at evaluating the ability of the developed approach to cope with a large number of features. The *Mushroom* dataset is partitioned in two equal-sized classes representing edible and poisonous mushrooms.

Scenario 4 – SENSORRANDOM. This scenario evaluates the robustness of the approach in a

[5] http://db.csail.mit.edu/labdata/labdata.html
[6] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
[7] http://archive.ics.uci.edu/ml/datasets/Mushroom

	SENSOR	CRIME	MUSHROOM	SENSOR RANDOM
Max Gini Index	0.86	0.86	0.63	0.83
Min Gini Index	0.75	0	0.62	0.80
Avg Gini Index	0.79	0.65	0.63	0.82
Std Deviation of Gini Index	0.025	0.159	0.001	0.005

Table 4.5 Scenario heterogeneity.

Coverage ($\alpha = 0.14$)	CRIME	MUSHROOM			SENSOR			SENSOR RANDOM
		$\alpha/2$	$3/4 \times \alpha$	α	B10	B100	B1000	
0.1 ± 0.0014	854	0	1	1	27958	29681	29899	29912
0.2 ± 0.0056	3500	2	4	5	939	982	3880	1569
0.3 ± 0.0126	446	7	10	12	50	52	194	24
0.4 ± 0.0224	45	12	15	20	45	94	34	70
0.5 ± 0.035	87	6	6	15	31	56	28	46
0.6 ± 0.0504	31	9	14	26	10	66	18	72
0.7 ± 0.0686	54	9	18	32	4	74	35	52
0.8 ± 0.0896	59	15	21	78	42	96	17	120
0.9 ± 0.1134	309	15	83	1439	359	90	72	72

Table 4.6 Number of d-formulas w.r.t. coverage.

data-driven mode with a bad partitioning. A synthetic attribute `Category` has been added to the *Sensor* dataset and used to generate 24 random partitions.

To understand whether the approach is robust in presence of data with high dispersion, the Gini Index is computed on each attribute of the datasets with respect to the class used in the partitioning (1 is max dispersion and 0 no dispersion). Table 4.5 reports some statistics of the Gini Indexes across the attributes. It shows that SENSORRANDOM has the most skewed distribution (due to the random partitioning), while MUSHROOM and CRIME are more homogeneous.

4.5.1 Efficiency

In this section, the efficiency of generating d-formulas and combining them into descriptions is evaluated.

First of all, the number of generated d-formulas with respect to the input coverage is reported in Table 4.6. All experiments are executed with a default of $\alpha = 0.14$ and with DEG $= 0.25$, which limits d-formulas to maximum 5 predicates. In MUSHROOM the sensitivity on different α-settings (half and three fourths of the default value) is also tested. The number of generated d-formulas is higher when the coverage is low for SENSOR, SENSORRANDOM and CRIME, and with high coverage for MUSHROOM. This is due to the

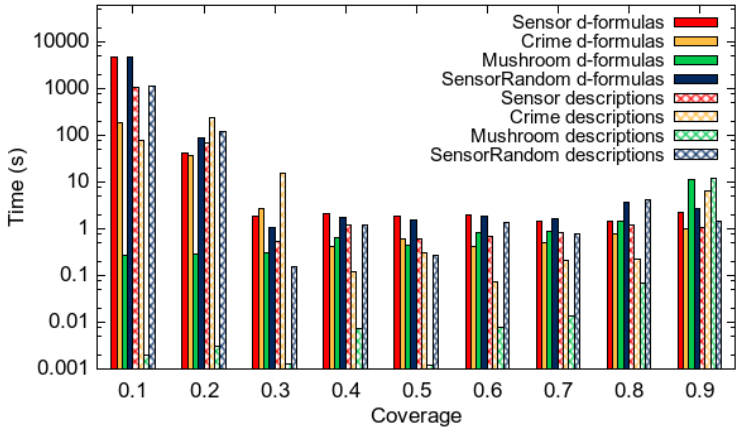


Fig. 4.3 Runtime performance.

Cov	No Heur.	Heur. 1							
		c=2	c=5	c=10	c=25	c=50	c=100	c=250	c=400
0.2	14570	6944	14497	14570	14570	14570	14570	14570	14570
0.4	115762	18710	107087	115757	115762	115762	115762	115762	115762
0.6	664226	39385	499940	664205	664208	664219	664219	664226	664226
0.8	1431480	51451	856218	1431460	1431480	1431480	1431480	1431480	1431480

Cov	Heur. 2	Heur. 1+2							
		c=2	c=5	c=10	c=25	c=50	c=100	c=250	c=400
0.2	52	2	5	11	26	42	51	52	52
0.4	258	8	20	37	124	167	249	256	258
0.6	636	6	26	59	138	368	489	624	630
0.8	10091	28	78	188	702	2021	4635	9759	10007

Table 4.7 Impact of heuristics on the number d-formulas.

fact that the frequency distribution of *Crime* and *Sensor* datasets' attributes follow the Zipf's rule: the majority of values appear only a few times. This results in many low-coverage descriptions. In the *Mushroom* dataset, there are many attributes with few values that repeat many times. Therefore, most of d-formulas are generated with these values at high coverage. The results of the MUSHROOM scenario also show that the parameter α does not affect the efficiency much. On SENSOR, the number of bins does not influence the number of generated d-formulas much.

Figure 4.3 shows the time required to generate d-formulas and to compute the final descriptions at different coverage settings. The final execution time is the sum of the two components. In general it is possible to notice that descriptions are produced at interactive time (few seconds) in most of the cases. The harder cases to process are with low coverage because the system has to sift throw several thousands of descriptions. It is also possible to note that the time for computing the descriptions is quadratic with the number of d-formulas, thus confirming the theoretical complexity of *LVA*.

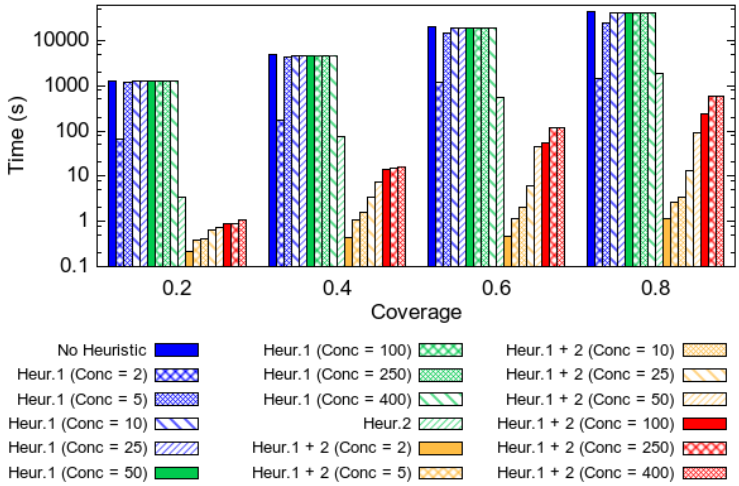


Fig. 4.4 Efficiency of heuristics (MUSHROOM scenario).

The efficiency of the heuristics, taken in isolation or together, compared against the exhaustive case (i.e., No Heuristic) is in Figure 4.4 (for MUSHROOM scenario). For Heuristic 1, different degree values are considered and the conciseness (from 2 to 400) derived from the degree variations from 0.25 to 1 is reported.

As we can see, the approach requires up to 30k seconds, whereas the combined use of the heuristics makes the approach usable in interactive applications.

The runtime performances are supported by the number of generated d-formulas in Table 4.7. The separate application of the heuristics is able to reduce the number of d-formulas up to 3 orders of magnitude, while the combined application reduces the number of d-formulas up to 5 orders of magnitude.

4.5.2 Quantitative evaluation of effectiveness

This experiment aims to evaluate the effectiveness of the proposed approach using quality metrics along three perspectives: (1) *structural quality*, i.e., characteristics of the descriptions independently of the use case; (2) *purity*, i.e., "how discriminative is a description?"; and (3) expressiveness, i.e., "how many details are captured by a description?".

Structural metrics. Two metrics, namely *overlapping* and *precision*, are used to evaluate the structural quality. *Overlapping* measures the percentage of tuples in the dataset for which more than one d-formula hold.

Generic descriptions have high percentage of *overlap*, whereas peculiar descriptions have small *overlapping*. The overlapping measure is $overlap(D) = \frac{2}{\rho*(\rho-1)} * \sum_{\substack{d_i,d_j \in D \\ i \leq j}} \frac{|I(d_i) \cap I(d_j)|}{|I|}$ where $I(d_i)$ represents the set of tuples of the dataset I that are true for d_i.

Orthogonally, the higher the number of values used in d-formula predicates, the higher the probability that a d-formula describes combinations of values that are not existing in the actual data. For instance, the d-formula $(a \in \{\alpha,\beta,\gamma\} \wedge b \in \{1,2\})$ describes three possible values for a and two for b. This is six combinations in total: $(a = \alpha \wedge b = 1)$, $(a = \alpha \wedge b = 2)$, etc. The *precision* checks how many combinations correspond to tuples that actually exist in the dataset. For instance, let us suppose that no tuple is true for $(a \in \{\alpha\} \wedge b \in \{1\})$. The *precision* highlights that one out of six unfolded formulas is extra.

The precision is defined as $prec(D) = \frac{1}{\rho} * \sum_{d \in D} \frac{\sum_{u \in C^{\times}(d)} eval(u, \sigma_d(I))}{|C^{\times}(d)|}$ where: $C^{\times}(d)$ returns all the combinations by computing the cartesian product between the values of the atomic predicates in d; and $eval(u, \sigma_d(I))$ is a boolean function that returns true if u holds in the data partition $\sigma_d(I)$ in which d holds, and false otherwise.

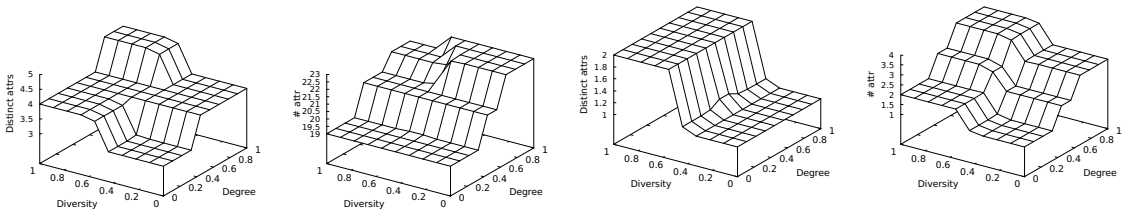
Purity metric. A description is "pure" when its predicates are discriminative. The *purity* measures the dispersion of the values of the attributes across the d-formulas and gets better (i.e., it tends to 1) when the most appropriate attributes to describe a set of tuples is selected. The purity of a description is the complement of the weighted average of the entropy of its d-formulas. More formally: $purity(D) = 1 - \left[\frac{1}{\rho} * \sum_{d \in D} \left(\frac{1}{|A_d|} * \sum_{a \in A_d} H_a(d)\right) * \frac{|I(d)|}{|I|}\right]$ with $H_a(d)$ being the Shannon entropy of the values of attribute a in the set of tuples described by the d-formula d.

Expressiveness metrics. The expressiveness of descriptions is evaluated with the *average number of predicates* per d-formula and with the *number of distinct attributes* per description. The *average number of predicates* provides a measure of the complexity of d-formulas: in fact, d-formulas composed of many attributes are clearly hard to read. This difficulty can be mitigated by using the same attributes across d-formulas. In fact, a low *number of distinct attributes* per description is preferable in terms of expressiveness.

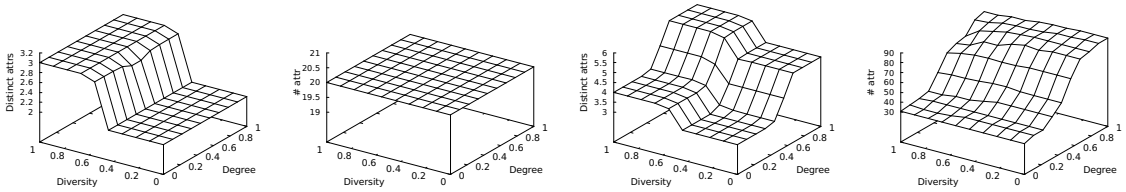
Benchmark Experiments

For each scenario, multiple combinations for the coverage, degree and diversity parameters are considered. For each combination, the top-25 descriptions are computed. In this way, a wide range of possible users' inputs are simulated. The result is the computation of 3025 descriptions per coverage level, which are evaluated according to the aforementioned metrics. Table 4.8 shows the average values (and the standard deviations between parentheses when not zero) of the descriptions of each combination, grouped by coverage.

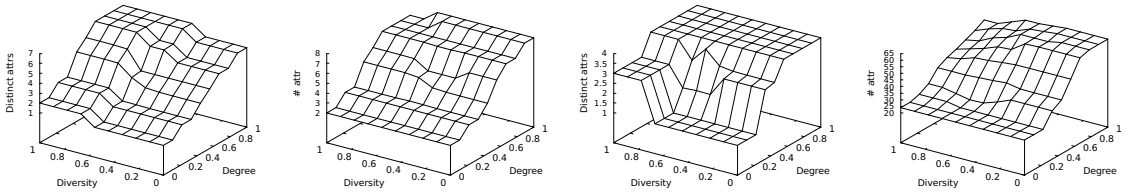
Analyzing the results, it is possible to observe that when the coverage increases, the purity increases as well. This is due to Heuristic 2 that avoids the combination of predicates with different coverage levels. A d-formula with low coverage is instead composed of predicates



(a) CRIME: low COV - # distinct attr. (b) CRIME: low COV - # attr. (c) MUSHROOM : low COV - # distinct attr. (d) MUSHROOM : low COV - # attr.



(e) SENSOR : low COV - # distinct attr. (f) SENSOR : low COV - # attr. (g) CRIME: high COV - # distinct attr. (h) CRIME: high COV - # attr.



(i) MUSHROOM : high COV - # distinct attr. (j) MUSHROOM : high COV - # attr. (k) SENSOR : high COV - # distinct attr. (l) SENSOR : high COV - # attr.

Fig. 4.5 Impact of the DEG and DIV parameters on CRIME's, MUSHROOM's and SENSOR's descriptions.

scenario	cov	purity	precision	overlap	# pred d-formula	# distinct attr
CRIME (30 part.)	0.10	0.21 (0.01)	1.00	0.01 (0.01)	1.00	3.333 (0.47)
	0.30	0.32 (0.09)	1.00	0.03 (0.01)	1.00	5.000
	0.50	0.38 (0.08)	1.00	0.05 (0.01)	1.09 (0.08)	4.000
	0.70	0.39 (0.04)	1.00	0.06 (0.02)	1.07 (0.07)	3.803 (0.79)
	0.90	0.59 (0.24)	1.00	0.26 (0.07)	1.91 (0.70)	4.384 (1.62)
MUSHROOM (2 part.)	0.10	0.03	1.00	0.01 (0.01)	1.00	1.833 (0.37)
	0.30	0.16 (0.17)	1.00	0.06 (0.05)	1.00	1.822 (0.38)
	0.50	0.17 (0.19)	1.00	0.24 (0.08)	1.00	1.844 (0.36)
	0.70	0.42 (0.26)	0.94 (0.07)	0.34 (0.17)	1.40 (0.40)	2.299 (0.87)
	0.90	0.28 (0.27)	0.53 (0.35)	0.57 (0.20)	6.23 (3.06)	6.990 (3.12)
SENSOR (24 part.)	0.10	0.10 (0.04)	1.00	0.04 (0.01)	1.00	4.000
	0.30	0.26 (0.02)	1.00	0.09 (0.01)	1.00	3.000
	0.50	0.29 (0.04)	1.00	0.22 (0.01)	1.00	2.444 (0.50)
	0.70	0.37	1.00	0.44 (0.06)	1.07 (0.07)	2.333 (0.47)
	0.90	0.58 (0.17)	0.98 (0.02)	0.72 (0.03)	1.78 (0.62)	2.679 (0.47)
SENSORRANDOM (24 part.)	0.10	0.01	1.00	0.04 (0.01)	1.00	1.00
	0.30	0.02	1.00	0.31 (0.01)	1.00	1.00
	0.50	0.19	1.00	0.47 (0.01)	1.00	1.00
	0.70	0.24	1.00	0.63 (0.02)	1.00	1.836 (0.37)
	0.90	0.38 (0.21)	1.00	0.83 (0.03)	1.00	2.360 (0.58)

Table 4.8 Effectiveness of the different scenarios.

with rather infrequent values, thus resulting in low purity. Furthermore, we can notice that the number of distinct attributes per description is typically low. This means that the descriptions are usually easy to read and are composed of d-formulas that can be compared with each other (because attributes are largely shared). The number of predicates, overlap, and precision also show a common trend. The overlap typically increases with the number of predicates while the precision decreases. The motivation is that with more predicates we increase the possibility to describe combinations of values that do not exist in the dataset. The partitions in the SENSORRANDOM scenario have been randomly generated, and contain data that is poorly related. This results in descriptions with high overlap and low purity with related to the SENSOR scenario where the data was partitioned over `hour`. Indeed this latter result shows that the approach is quite robust against random or "bad" partitioning.

Comparison with Other Approaches

The developed approach is compared against Decision Trees (DTC), Decision Sets (DS) [78] and rules generated through Subgroup Discovery Techniques (SD).

DTC. For each scenario, a tree is trained over the full dataset against the partitioning attribute. The rules obtained are translated into d-formulas of a single description. The DTC provided by the Scikit-learn platform is used with different parameters to find a configuration mini-

	rules/ part.	**degree**	**diversity**	**cov**	**purity**	**prec.**	**overlap**	**# pred d-formula**	**# distinct attr**
CRIME	161.97	0.77	0.25	99.95%	0.26	1.00	0.00	23.05	6.00
MUSHROOM	6.5	0.20	0.93	100%	0.19	1.00	0.00	5.00	9.00
SENSOR	2635.13	0.70	0.25	48.84%	0.13	1.00	0.00	20.32	6.00
SENSORRANDOM	3934.04	0.78	0.23	12.10%	0.13	1.00	0.00	24.71	6.00

Table 4.9 Comparison with Decision Trees. Coverage in this case is the final accuracy of the generated model.

mizing the number of rules, and using the entropy H (the same used by *LVA*) as optimization metric for the tree.

By analyzing the results reported in Table 4.9, it is possible to note that a decision tree is not a flexible choice since any customization, similar to the use of degree, diversity, and coverage parameters, is not applicable. Moreover, with a DTC it is not possible to constraint the generation of a single d-formula per partition. Only for MUSHROOM, the number of rules is reasonably limited. In other scenarios, DTCs generate thousand of rules, thus making impossible for a human to get insights from them. Note that when the partitions of the dataset are randomly generated, the number of rules explodes, and the coverage of the result is very low.

DS. Decision sets are set of independent if-then rules. Because each rule can be applied independently, decision sets are considered as simple, concise, and easily interpretable [78]. The authors propose a classification algorithm based on decision sets, which are obtained from the rules that maximize accuracy and interpretability among all the rules generated by the A-PRIORI algorithm [6].

In the experiment, the DS implementation provided by the authors themselves [8] is evaluated on the MUSHROOM scenario since the datasets used in their paper are not publicly available. Setting a support threshold to 0.8, the A-PRIORI algorithm returned 46 rules. The optimization process took more than 85 hours to return the best 14 rules, which have been then transformed into a single description for the dataset. Coverage, degree, and diversity of that description are in Table 4.10. Note that by reducing the support, more candidate rules to be analyzed by the optimization process are obtained. Nevertheless, the existing implementation does not converge when more rules are evaluated (the computation has been timeout after 300 hours). For the same reason experiments about other scenarios are not reported. Moreover, the optimization algorithm uses sampling techniques and therefore is not deterministic in time and results. Then, the developed system has been forced to generate a description with similar characteristics. The second line of Table 4.10 shows

[8] https://github.com/lvhimabindu/interpretable_decision_sets

that the description of the proposed approach is simpler and more interpretable than the one generated with DS since it has only two rules (one rule per partition), and has lower overlap while keeping a similar purity level. The rest of Table 4.10 shows that the best description is obtained with high degree and low diversity.

SD. Subgroup discovery is a data mining technique that builds interesting rules with respect to a target variable. The "interestingness" is evaluated according to some statistical measure of the data distribution (see related work in Section 4.6). Therefore, SD techniques can be directly used for generating descriptions of a dataset. The Python *pysubgroup* [80] library has been run on the MUSHROOM dataset. This library provides an implementation of the main approaches proposed in the SD literature, including BSD [81], BeamSearch [30], and a Deep First Search algorithm. Table 4.10 shows the results obtained with the application of the BSD algorithm, but, similar values (and in most cases the same values) are obtained with the other approaches. The algorithm has been tested with different depth values (i.e., the number of attributes that can jointly form a rule) and two "unusualness quality measures" typically adopted in these approaches for ranking the rules (i.e., the weighted relative accuracy – WRAcc , and the added value – AV [14]). The experiments show that changes in the algorithm depth do not provide any significant variation in the results. The WRAcc quality measure typically generates broad descriptions (the ones with coverage close to 1) and AV tends to return narrow and outlier descriptions (the ones with coverage close to 0). Note that Table 4.10 reports a coverage value for the rules ranked with the AV quality measure equal to 0.63 when depth is 1 and equal to 0.05 when depth is 5. The coverage values computed when depth is set to 2, 3 and 4 is 0.31, 0.08 and 0.04, respectively. Moreover, the SD approaches generate effective rules, with similar or better values for the dimensions under evaluation (purity, precision and overlap). Nevertheless, SD does not introduce any mechanism for customizing coverage, degree and diversity, which is one of the improvements of the proposed approach and a helpful support for the data exploration.

Degree/Diversity Impact on Descriptions

Finally, the evaluate of the effect of user-selected DEG and DIV on the generated descriptions is analyzed. In particular, multiple combinations of degree and diversity levels (DEG and DIV varying from 0.0 to 1.0 with an offset equal to 0.1) are executed and the total number of attributes and distinct attributes of the top-1 computed description is evaluated. Figure 4.5 shows the results of this evaluation. For each scenario, only one plot for low and high coverage levels is reported; the other settings show a similar trend. We see that the user's preferences drive the generation of descriptions with the desired features. When the DEG

	# rules	degree	diversity	cov	purity	precision	overlap	# pred d-formula	# distinct attr
Decision sets [78]	14	0.1	0.9	1	0.77	1.00	0.88	2.43	5.00
Our approach	2	0.1	0.9	1	0.75	1.00	0.77	2.00	4.00
		0.1	0.1	1	0.77	1.00	0.77	2.50	4.00
		0.9	0.1	1	0.38	1.00	0.61	3.50	5.00
		0.9	0.9	1	0.51	1.00	0.61	2.50	5.00
		0.8	0.6	0.05	0.49	1.00	0.001	1.00	2.00
SD (depth 1, WRAcc)	2	0.2	1	0.96	0.93	1.00	0.32	1.00	2.00
SD (depth 5, WRAcc)	2	0.2	1	0.96	0.93	1.00	0.32	1.50	3.00
SD (depth 10, WRAcc)	2	0.2	1	0.96	0.93	1.00	0.32	1.50	3.00
SD (depth 20, WRAcc)	2	0.2	1	0.96	0.93	1.00	0.32	1.50	3.00
SD (depth 1, AV)	2	0.2	1	0.63	0.71	1.00	0.16	1.00	2.00
SD (depth 5, AV)	2	0.8	0.63	0.05	0.84	1.00	0.001	4.00	5.00
SD (depth 10, AV)	2	0.8	0.57	0.05	0.84	1.00	0.001	3.50	4.00
SD (depth 20, AV)	2	0.8	0.57	0.05	0.84	1.00	0.001	3.50	4.00

Table 4.10 Comparison with Decision Sets over MUSHROOM.

increases, the number of attributes increases too. Low levels of DIV correspond to a low number of distinct attributes.

4.5.3 Qualitative evaluation of effectiveness

Section 4.5.2 performed a quantitative evaluation of the effectiveness of the proposed approach and compared it with Decision Trees, Decision Sets and rules generated through Subgroup Discovery Techniques. The experiments showed that, compared to proposed approach, they adopted strategies impeding the generation of outputs composed of one rule per partition and rules with user-specified levels of degree and diversity, i.e., the cornerstones of the presented solution. This Section qualitatively analyzes the effectiveness of the introduced approach with the aim of evaluating if tuning the values of degree and variety the user is able to generate interesting and useful descriptions. For this reason, the questionnaire shown in Figure 4.6 has been submitted to 21 PhD students of computer science. The questionnaire has 12 questions about 8 descriptions for the MUSHROOM dataset (4 in a low coverage setting and 4 with high coverage). In the first part of the questionnaire, 6 questions ask the users to select between two descriptions, the one they would have selected for solving a specific explanation task (i.e. a task that would have required either a broad description or an outlier detection). The descriptions proposed are conceived to force the users to select between a high and a low diversity description or between a high and a low degree description. This evaluation showed that the proposed measures are important to generate useful descriptions. Users generally prefer low diversity descriptions (61.90%). Only 19.05% of the population prefer high diversity, while the remaining 19.05% expressed no preference. The preferences

The Mushroom Data Set

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, and poisonous. The dataset is composed of 8124 instances. It is available at https://archive.ics.uci.edu/ml/datasets/mushroom

		POISONOUS	EDIBLE
D1	LOW COVERAGE	({class = poisonous AND cap-color ∈ {'cinnamon'}})	({class = edible AND cap-color ∈ {'purple'} }) }
D2		({class = poisonous AND ring-number ∈ {'none'} })	({class = edible AND ring-type ∈ {'flaring'} }) }
D3		({class = poisonous AND cap-color ∈ {''cinnamon' '} AND ring-number ∈ {'none '} AND population ∈ {'clustered'} AND odor ∈ {'musty'} AND ring-type ∈ {'none'}})	({class = edible AND cap-color∈ {'purple '} AND spore-print-color ∈ {'chocolate'} AND ring-type ∈ {'flaring'}})
D4		({class = poisonous AND cap-color ∈ {''cinnamon'} AND ring-number ∈ {'none '} AND population ∈ {'clustered'} AND odor ∈ {'musty'}})	({class = edible AND spore-print-color ∈ {'chocolate'} AND ring-type ∈ {'flaring'} })
D5	HIGH COVERAGE	{class = p AND spore-print-color ∈ {'chocolate', 'white'} })	({class = e AND spore-print-color ∈ {'brown', 'black'} }))
D6		({class = poisonous AND bruises ∈ {'no'} })	({class = edible AND odor ∈ {'none'} })
D7		({class = poisonous AND spore-print-color ∈ {'chocolate', 'white'} AND ring-number ∈ {'one'}}) OR	({class = edible AND spore-print-color ∈ {'brown', 'black'} AND cap-surface ∈ {'scaly', 'smooth', 'fibrous'} AND ring-number ∈ {'one'}}) }
D8		({class = poisonous AND ring-number ∈ {'one '} AND bruises ∈ {'no'} }) OR	({class = edible AND spore-print-color ∈ {'brown', 'black'} AND cap-surface ∈ {'scaly', "smooth', 'fibrous'} AND bruises ∈ {'no', 'yes'} AND ring-number ∈ {'one'} })

Let us suppose that you want to know some anomalies about the dataset.

1. Do you prefer D1 or D5?
2. Do you prefer D1 or D2?
3. Do you prefer D1 or D3?
4. Is there any real difference between the knowledge of the dataset content you obtain from D3 and D4? YES / NO

Let us suppose that you want to have a broad, generic description about the dataset.

1. Do you prefer D5 or D6?
2. Do you prefer D5 or D7?

Would you eat

1. a mushroom with a cap-color purple? YES / NO
2. a mushroom with a cap-color cinnamon? YES / NO
3. a mushroom with 1 ring? YES / NO
4. a mushroom with 1 ring and no bruises? YES / NO

Is it true that

1. an edible mushroom has no odor? YES / NO
2. an edible mushroom has spore-print-color brown or black? YES / NO

Fig. 4.6 Questionnaire administered to the users.

on the degree questions are less marked: 42.86% of the users prefer high degree descriptions, 23.80% low degree, and 33.34% have no preference. With the other 6 questions, it is evaluated how easy it is to gain insights via descriptions. The other 6 questions wanted to evaluate if the descriptions are able to provide users with insight of the dataset. Some questions about facts that can be inferred from data (i.e., "would you eat a mushroom with a purple cap-color?") have been then asked. Users answered correctly in 68.25% of the cases. Finally, the students rated their experience with our system with an average of 4.1/5.

Finally, the Fleiss' kappa is computed to assess the reliability of agreement between users. The result is 0.607, which is a significant agreement.

4.6 Related Work

In this chapter the application of a subgroup discovery technique for performing data explanation and exploration on structured datasets has been considered. Subgroup discovery is a descriptive data mining field that aims at identifying interesting groups of individuals, where "interestingness is defined as distributional unusualness with respect to a certain property of interest" [14, 158]. Explanation systems help users in gaining knowledge on the behavior of systems, experiments or query answers [90], and "black box" complex models [59]. Explanations typically assume the form of association rules [6], decision lists [73] and decision sets [78]. The proposed approach performs data explanation since it creates partitions from a dataset and builds rules that provide users with an explanation of their content. Data exploration is about efficiently extracting knowledge from data even if we do not know exactly what we are looking for [66]. It is typically based on descriptive statistical techniques and adopts visual exploration interface [150] for supporting users in the interactively discovery of data source contents. The introduced approach allows users to interactively customize the ways the partitions and rules are built (in terms of number, coverage, degree, and diversity) thus supporting different modalities for describing and visualizing the data, which is a data exploration task.

Subgroup Discovery Systems. Subgroup discovery was a very active data mining field in the early 2000s. A large number of subgroup discovery approaches have been proposed [14, 61] and experimented in real domains (such as the medical domain [50]). The approaches typically divide the task into three steps: the generation of the candidate rules, the adoption of a pruning scheme for selecting the significant ones, and the application of quality measures (see [126] for a list) to evaluates the results obtained. To classify the algorithms in the literature, [63] distinguishes between extensions of classification algorithms, extension of association algorithms [71] and evolutionary fuzzy systems. The main improvement to this

field promoted by the proposed approach concerns the definition of an objective function that scores patterns emerging from the dataset (e.g., the statistical distribution of the data values) with the specific analytic user needs (e.g., to obtain interpretable/readable rules, and/or rules that provide a precise description of the dataset, or of some anomalies only, ...). The function is optimized and a globally near-optimal model is found. Moreover, the approach has been designed for supporting users in interactively analyze a dataset, by allowing them tuning and testing the algorithm parameters and gathering different insight into the same dataset.

Outlier Explanation Systems. Explanation techniques have typically been applied to data sources for explaining outliers. Traditional data lineage techniques present several limitations when applied to this field because users want to know which subsets of the lineage have a "bad" influence on the outlier result. Meliou et al. [89] pioneer this research area by studying causality, as opposed to correlation, identifying tuples, seen as potential causes, that are responsible for answers and non-answers to queries. They introduce the degree of responsibility of the tuples. The process of deleting candidate solutions and verifying if those solutions affected the outliers is called *intervention* and it is iteratively repeated in order to find the most influential groups of tuples, i.e., explanations [89, 127, 128, 159]. Scorpion [159] explains a (possibly outlier) query result by individuating responsible tuples in terms of influence on that result. Roy et al. mix intervention with causality to overcome the limitations of Scorpion in generating explanations [128]. To overcome the high complexity, online explanations have been proposed [127].

Data Explanation Systems. More general explanation techniques beyond outliers have been recently proposed. The *Smart drill-down* operator interactively computes the k most interesting rules (for a scoring function) that describe a portion of the dataset [68] with respect to attributes selected by the users. These drill-down rules are close to the descriptions presented in this chapter since both describe the content of a dataset. Unlike the proposed approach, this strategy focuses only on broad (i.e., high coverage) descriptions, and does not allow users identifying information with low coverage. Moreover, the drill down rules are automatically formulated after the selection of the target attributes without any possible user-customization. The presented approach allows users not only to select the attributes of interest, but also to customize the diversity and the degree of the rules. Techniques for explaining predictions of classifiers have been proposed [78, 125]. The work of Lakkaraju et al. [78] on using decision sets (i.e., rules) is comparable to the one considered in this chapter. The approach generates rules that explain the reason why a data instance belongs to a specific class. Rules are created by maximizing an objective function, where 7 quality measures are linear combined. Two measures concern the interpretability, by favouring decision sets with a smaller number of rules (the number of partitions in our approach) and fewer predicates.

Two measures favor the creation of decision sets with rules that do not overlap, one measure guarantees the existence of a rule for each class, and the remaining two rules guarantees the accuracy by measuring precision and recall of the rules. The descriptions are built by using similar measures, but are more flexible since implemented through a user-in-the-loop approach to select coverage and other quality features for the generated rules. Moreover the existing implementation does not scale on large dataset as evaluated in the experiments in Section 4.5.

Data Profiling and Summarization. Data profiling is the activity of determining metadata (such as statistics about the data or dependencies among columns) about a given dataset [2]. Conversely, the approach presented in this chapter generates metadata (descriptions) about horizontal partitions of the source. A type of data profilers is data summarization systems that aim at creating lossy compressed representation of database. These works however mostly focus on selecting the most important database tables [162] or compressing database schemas [163]. The goal of the presented approach, however, is to provide a tool allowing users to easily explore large datasets in order to fulfill their information need [112]. In data and knowledge-bases, summaries are typically used also for output presentation [54, 45], query or data access optimization through e.g., histograms [69, 99], ontology creation and exploration [149]. A recent and extensive study of semantic graph summarization techniques is also available in [27].

Data Exploration Systems. The discussed approach can support data exploration, since it allows users to interactively extract knowledge from data. [66] provides a survey whereby existing research is clustered in three groups according to their focus (user interactions, middleware and database layer). Nevertheless, the aim of data exploration systems in general differs from the one presented in this chapter, because their aim is of guiding the users toward an answer to their specific information needs [85]. In this field, approaches that allow exploratory and visual interaction with the database through choosing relevant tuples and attributes [129, 25] have been proposed. They have to consider the minimization of the costs of grouping and selection to guarantee interactivity with the systems [129]. In a similar way, there is a branch of research that extend these visual approaches to more analytical data manipulation scenarios [25]. Among these approaches: INDIANA [53] is a system supporting users in the exploration of transactional databases in an interactive way. This approach uses statistics for identifying the features that are relevant according to the users' exploration process. The description-based approach let the user discover what a database contains, based on features emerged from the dataset. QueRIE [43] suggests interesting exploratory SQL queries to the users by implementing a collaborative filtering approach. Finally, more recent research addresses issues related to the exploration of data and schema

changes. [19] introduce the notion of the change-cube data model and a set of primitives to support change exploration.

4.7 Conclusion

In this chapter, the problem of generating data descriptions, a compact and insightful sets of predicates allowing data enthusiasts to inspect datasets at a glance, has been introduced.

A set of heuristics to drive the generation of data descriptions in a computationally efficient manner while returning high-quality results has been proposed, and an extensive experimental evaluation over real-world datasets, has confirmed the goodness of the proposed approach both in a quantitative and qualitative perspective.

Chapter 5

Conclusion

In this thesis, three topics of great interest to both the academic community and the industrial world have been analyzed: 1) data integration, 2) in-RDBMS execution of Machine Learning capabilities and 3) data exploration and explanation. For each of them, the state of the art was analyzed and an approach was proposed for the resolution of its specific challenge.

In the field of data integration, the problem of evaluating data integration processes was analyzed in detail. It still requires an expensive and time-consuming involvement of domain experts, and an approach for its unsupervised evaluation was presented to provide a solution to this problem. The proposed approach relies on a series of "representativeness" metrics, which are based on the analysis of the word frequency distributions of the involved datasets, and have proved effective in solving this task. This is an important achievement as it allows to dramatically reduce the involvement of expensive domain experts and to accelerate the execution of an end-to-end and effective data integration pipeline. The main limitation of this approach is the adoption of a global and syntactic representativeness measure. This can generate locally inaccurate representativeness scores which are unable to detect semantic similarities. A possible solution to this problem is to integrate this approach with modern techniques of word/sentence embeddings, which would map the dataset entities in an embedding space where local relations of semantics and syntactic similarities are preserved.

A second topic addressed in this thesis was the integration of ML capabilities within RDBMS. In this context, the use of relational databases as a tool for serving ML pipelines was analyzed in detail. This activity is nowadays performed in two steps, where the data is first extracted from the database and then uploaded to a dedicated analytics system to apply the inference process. This produces obvious problems in terms of performance and safety. To solve this problem, MASQ (Machine Learning As Query) was presented, a library aiming at translating end-to-end ML pipelines into SQL queries to be used for

solving ML inference tasks directly within the databases where the data reside. An extensive experimental evaluation has shown that this approach provides comparable performance with well-known ML frameworks (e.g. Sklearn and ML.NET). This outcome proves that predictive pipelines can be served without having to trade-off performance with the "Enterprise-grade" provided by DBMSs. Although this study highlighted how a pure SQL-based approach can be sufficient to solve traditional machine learning pipelines, several limitations were encountered in solving deep learning techniques and text-based ML models. In such scenarios a user-defined functions (UDFs) approach would be preferable. A possible future work is also to test the MASQ-generated SQL queries in parallel computing platforms (such as Apache Spark), in order to analyze the impact of its integration also in such frameworks.

Finally, the problem of data exploration and explanation was analyzed, which increasingly requires the adoption of easy-to-use (even for non-expert users) and interactive tools for identifying and explaining relevant information in large repositories. In response to this need, an approach has been proposed for the interactive generation of task-oriented, compact and informative (data) descriptions of a dataset. An extensive experimental evaluation has shown that the proposed solution is effective in solving this task both quantitatively and qualitatively. An interesting future work is to extend this approach over specific scenarios beyond data exploration, such as query result explanation.

Other research topics not described in this thesis were also addressed during the Ph.D., such as the application of keyword search engines on top of relational databases based on a distributed implementation of the full disjunction operator [106], the alignment of Wikipedia data over time [144], the application of Automated Machine Learning (AutoML) techniques to EM models [108] and the interpretation of black-box EM models [15]. Finally, several publications in the field of data integration have been produced [58, 109, 57] as a result of participation in the European project Re-Search Alps [1], whose objective was to integrate multiple heterogeneous data sources concerning organizations in a single homogeneous version.

[1] http://researchalps.eu

References

[1] (2020). Tidypredict. https://tidypredict.netlify.com/.

[2] Abedjan, Z., Golab, L., Naumann, F., and Papenbrock, T. (2018). *Data Profiling*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[3] Aberger, C., Lamb, A., Olukotun, K., and Re, C. (2018). Levelheaded: A unified engine for business intelligence and linear algebra querying. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 449–460.

[4] Agrawal, A., Chatterjee, R., Curino, C., Floratou, A., Gowdal, N., Interlandi, M., Jindal, A., Karanasos, K., Krishnan, S., Kroth, B., Leeka, J., Park, K., Patel, H., Poppe, O., Psallidas, F., Ramakrishnan, R., Roy, A., Saur, K., Sen, R., Weimer, M., Wright, T., and Zhu, Y. (2019). Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML. *arXiv e-prints*, page arXiv:1909.00084.

[5] Agrawal, P., Arya, R., Bindal, A., Bhatia, S., Gagneja, A., Godlewski, J., Low, Y., Muss, T., Paliwal, M. M., Raman, S., Shah, V., Shen, B., Sugden, L., Zhao, K., and Wu, M. (2019). Data platform for machine learning. In Boncz, P. A., Manegold, S., Ailamaki, A., Deshpande, A., and Kraska, T., editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1803–1816. ACM.

[6] Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216. ACM Press.

[7] Ahmed, Z., Amizadeh, S., and et al., M. B. (2019). Machine learning at microsoft with ML.NET. In *KDD*, pages 2448–2458.

[8] Alonso, G., István, Z., Kara, K., Owaida, M., and Sidler, D. (2019). doppiodb 1.0: Machine learning inside a relational engine. *IEEE Data Eng. Bull.*, 42(2):19–31.

[9] Amazon (2020). The total cost of ownership (tco) of amazon sagemaker. https://pages.awscloud.com/rs/112-TZM-766/images/Amazon_SageMaker_TCO_uf.pdf.

[10] Amazon.com (2021). Redshift ml.

[11] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H. C., Kamar, E., Nagappan, N., Nushi, B., and Zimmermann, T. (2019). Software engineering for machine learning: a case study. In Sharp, H. and Whalen, M., editors, *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 291–300.

[12] Arasu, A., Götz, M., and Kaushik, R. (2010). On active learning of record matching packages. In *SIGMOD*. ACM.

[13] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., and Zaharia, M. (2015). Spark SQL: relational data processing in spark. In Sellis, T. K., Davidson, S. B., and Ives, Z. G., editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1383–1394. ACM.

[14] Atzmueller, M. (2015). Subgroup discovery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 5(1):35–49.

[15] Baraldi, A., Buono, F. D., Paganelli, M., and Guerra, F. (2021). Using Landmarks for Explaining Entity Matching Models. In *EDBT*.

[16] Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., and et al. (2017). Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1387–1395, New York, NY, USA. Association for Computing Machinery.

[17] Bellare, K., Iyengar, S., Parameswaran, A. G., and Rastogi, V. (2012). Active sampling for entity matching. In *The 18th ACM SIGKDD and KDD '12, Beijing, China, August 12-16, 2012*. ACM.

[18] Bernardi, L., Mavridis, T., and Estevez, P. (2019). 150 successful machine learning models: 6 lessons learned at booking.com. In Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., and Karypis, G., editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 1743–1751. ACM.

[19] Bleifuß, T., Bornemann, L., Kalashnikov, D. V., Naumann, F., and Srivastava, D. (2019). Dbchex: Interactive exploration of data and schema change. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org.

[20] Boehm, M., Dusenberry, M. W., Eriksson, D., Evfimievski, A. V., Manshadi, F. M., Pansare, N., Reinwald, B., Reiss, F. R., Sen, P., Surve, A. C., and et al. (2016). Systemml: Declarative machine learning on spark. *Proc. VLDB Endow.*, 9(13):1425–1436.

[21] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics*.

[22] Breck, E., Cai, S., Nielsen, E., Salib, M., and Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. In Nie, J., Obradovic, Z., Suzumura, T., Ghosh, R., Nambiar, R., Wang, C., Zang, H., Baeza-Yates, R. A., Hu, X., Kepner, J., Cuzzocrea, A., Tang, J., and Toyoda, M., editors, *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pages 1123–1132. IEEE Computer Society.

[23] Breck, E., Polyzotis, N., Roy, S., Whang, S. E., and Zinkevich, M. (2019). Data validation for machine learning. In *Proceedings of the 2nd SysML Conference, 2019, Palo Alto, CA, USA, March 31 - April 2, 2019*.

[24] Brunner, U. and Stockinger, K. (2020). Entity matching with transformer architectures - A step forward in data integration. In *EDBT*, pages 463–473. OpenProceedings.org.

[25] Buoncristiano, M., Mecca, G., Quintarelli, E., Roveri, M., Santoro, D., and Tanca, L. (2015). Database challenges for exploratory computing. *SIGMOD Record*.

[26] Buono, F. D., Paganelli, M., Sottovia, P., Interlandi, M., and Guerra, F. (2021). Transforming ML Predictive Pipelines into SQL with MASQ. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data, China, June 18–27, 2021*. ACM.

[27] Cebiric, S., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., and Zneika, M. (2019). Summarizing semantic graphs: a survey. *VLDB J.*, 28(3):295–327.

[28] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

[29] Chen, Z., Chen, Q., Fan, F., Wang, Y., Wang, Z., Nafa, Y., Li, Z., Liu, H., and Pan, W. (2018). Enabling quality control for entity resolution: A human and machine cooperation framework. In *ICDE*.

[30] Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283.

[31] Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J. M., and Welton, C. (2009). Mad skills: New analysis practices for big data. *Proc. VLDB Endow.*, 2(2):1481–1492.

[32] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

[33] Corporation, O. (2021). Oracle ml.

[34] Crankshaw, D., Bailis, P., Gonzalez, J. E., Li, H., Zhang, Z., Franklin, M. J., Ghodsi, A., and Jordan, M. I. (2015). The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org.

[35] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. (2017). Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA. USENIX Association.

[36] Criteo (2014). Kaggle challenge.

[37] Das, S., C., P. S. G., Doan, A., Naughton, J. F., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V., and Park, Y. (2017). Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD Conference*, pages 1431–1446. ACM.

[38] Dietrich, B. and Grust, T. (2015). A SQL debugger built from spare parts: Turning a SQL: 1999 database system into its own debugger. In *SIGMOD*, pages 865–870.

[39] Dolatshah, M., Teoh, M., Wang, J., and Pei, J. (2018). Cleaning crowdsourced labels using oracles for statistical classification. *Proc. VLDB Endow.*, 12.

[40] Dong, X. L., Berti-Équille, L., and Srivastava, D. (2015). Data fusion: Resolving conflicts from multiple sources. *CoRR*, abs/1503.00310.

[41] Dong, X. L. and Srivastava, D. (2015). *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[42] Ebraheem, M., Thirumuruganathan, S., Joty, S. R., Ouzzani, M., and Tang, N. (2018). Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.*, 11(11):1454–1467.

[43] Eirinaki, M., Abraham, S., Polyzotis, N., and Shaikh, N. (2014). QueRIE: Collaborative database exploration. *IEEE Trans. Knowl. Data Eng.*, 26(7):1778–1790.

[44] El Gebaly, K., Agrawal, P., Golab, L., Korn, F., and Srivastava, D. (2014). Interpretable and informative explanations of outcomes. *Proc. VLDB Endow.*, 8(1):61–72.

[45] Fakas, G. J., Cai, Y., Cai, Z., and Mamoulis, N. (2018). Thematic ranking of object summaries for keyword search. *Data Knowl. Eng.*, 113:1–17.

[46] Feng, X., Kumar, A., Recht, B., and Ré, C. (2012). Towards a unified architecture for in-rdbms analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, page 325–336, New York, NY, USA. Association for Computing Machinery.

[47] FigureEight (2016). Data Science Report. https://visit.figure-eight.com/data-science-report.html.

[48] Firmani, D., Saha, B., and Srivastava, D. (2016). Online entity resolution using an oracle. *Proc. VLDB Endow.*, 9.

[49] Forney, G.D., J. (1973). The Viterbi algorithm. *Proceedings of the IEEE*.

[50] Gamberger, D., Lavrac, N., and Krstacic, G. (2002). Confirmation rule induction and its applications to coronary heart disease diagnosis and risk group discovery. *Journal of Intelligent and Fuzzy Systems*, 12(1):35–48.

[51] García, S., Luengo, J., and et al. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *TKDE*, 25(4):734–750.

[52] Getoor, L. and Machanavajjhala, A. (2012). Entity resolution: Theory, practice & open challenges. *Proc. VLDB Endow.*, 5(12):2018–2019.

[53] Giuzio, A., Mecca, G., Quintarelli, E., Roveri, M., Santoro, D., and Tanca, L. (2019). Indiana: An interactive system for assisting database exploration. *Information Systems*, 83:40 – 56.

[54] Gkorgkas, O., Stefanidis, K., and Nørv*rule*(a)g, K. (2013). A framework for grouping and summarizing keyword search results. In *Advances in Databases and Information Systems - 17th East European Conference, ADBIS 2013, Genoa, Italy, September 1-4, 2013. Proceedings*, volume 8133 of *Lecture Notes in Computer Science*, pages 246–259. Springer.

[55] Golshan, B., Halevy, A. Y., Mihaila, G. A., and Tan, W. (2017). Data integration: After the teenage years. In *SIGMOD*.

[56] Google, I. (2021). Big query ml.

[57] Guerra, F., Russo, M., Fontana, M., Paganelli, M., Bancilhon, F., Frisch, C., Petit, L., Giorgi, A., and Zilio, E. (2017). The RE-SEARCH ALPS (research laboratories in the alpine area) project. In *HPCS*, pages 64–69. IEEE.

[58] Guerra, F., Sottovia, P., Paganelli, M., and Vincini, M. (2019). Big data integration of heterogeneous data sources: The re-search alps case study. In *BigData Congress*, pages 106–110. IEEE.

[59] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *CSUR*, 51(5):93.

[60] H2O (2020). H2O. https://www.h2o.ai/.

[61] Helal, S. (2016). Subgroup discovery algorithms: A survey and empirical evaluation. *J. Comput. Sci. Technol.*, 31(3):561–576.

[62] Hellerstein, J. M., Ré, C., Schoppmann, F., Wang, D. Z., Fratkin, E., Gorajek, A., Ng, K. S., Welton, C., Feng, X., Li, K., and et al. (2012). The madlib analytics library: Or mad skills, the sql. *Proc. VLDB Endow.*, 5(12):1700–1711.

[63] Herrera, F., Carmona, C. J., González, P., and del Jesús, M. J. (2011). An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.*, 29(3):495–525.

[64] Holmes, G., Donkin, A., and Witten, I. H. (1994). Weka: a machine learning workbench. In *Proceedings of ANZIIS '94 - Australian New Zealnd Intelligent Information Systems Conference*, pages 357–361.

[65] Hou, B., Chen, Q., Chen, Z., Nafa, Y., and Li, Z. (2020). r-humo: A risk-aware human-machine cooperation framework for entity resolution with quality guarantees. *IEEE Trans. Knowl. Data Eng.*, 32.

[66] Idreos, S., Papaemmanouil, O., and Chaudhuri, S. (2015). Overview of data exploration techniques. In *SIGMOD*, pages 277–281.

[67] Jankov, D., Luo, S., Yuan, B., Cai, Z., Zou, J., Jermaine, C., and Gao, Z. J. (2019). Declarative recursive computation on an rdbms: Or, why you should use a database for distributed machine learning. *Proc. VLDB Endow.*, 12(7):822–835.

[68] Joglekar, M., Garcia-Molina, H., and Parameswaran, A. G. (2019). Interactive data exploration with smart drill-down. *IEEE Trans. Knowl. Data Eng.*, 31(1):46–60.

[69] Kanne, C. and Moerkotte, G. (2010). Histograms reloaded: the merits of bucket diversity. In *SIGMOD*, pages 663–674.

[70] Karanasos, K., Interlandi, M., Psallidas, F., Sen, R., Park, K., Popivanov, I., Xin, D., Nakandala, S., Krishnan, S., Weimer, M., Yu, Y., Ramakrishnan, R., and Curino, C. (2020). Extending relational query processing with ML inference. In *CIDR 2020, 10th Conference on Innovative Data Systems Research, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org.

[71] Kavsek, B. and Lavrac, N. (2006). APRIORI-SD: adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence*, 20(7):543–583.

[72] Khoussainova, N., Balazinska, M., and Suciu, D. (2012). PerfXplain: Debugging MapReduce Job Performance. *PVLDB*, 5(7):598–609.

[73] Klivans, A. R. and Servedio, R. A. (2006). Toward attribute efficient learning of decision lists and parities. *Journal of Machine Learning Research*, 7:587–602.

[74] Konda, P., Das, S., C., P. S. G., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J. F., Prasad, S., Krishnan, G., Deep, R., and Raghavendra, V. (2016). Magellan: Toward building entity matching management systems. *Proc. VLDB Endow.*, 9(12):1197–1208.

[75] Konda, P., Seshadri, S. S., Segarra, E., Hueth, B., and Doan, A. (2019). Executing entity matching end to end: A case study. In *Proceedings of EDBT 2019, Lisbon, Portugal, March 26-29, 2019*. OpenProceedings.org.

[76] Kumar, A., Naughton, J., and Patel, J. M. (2015). Learning generalized linear models over normalized data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 1969–1984, New York, NY, USA. Association for Computing Machinery.

[77] Kwashie, S., Liu, J., Li, J., Liu, L., Stumptner, M., and Yang, L. (2019). Certus: An effective entity resolution approach with graph differential dependencies (gdds). *Proc. VLDB Endow.*, 12(6):653–666.

[78] Lakkaraju, H., Bach, S. H., and Leskovec, J. (2016). Interpretable decision sets: A joint framework for description and prediction. In *SIGKDD*, pages 1675–1684.

[79] Lee, Y., Scolari, A., Chun, B., Santambrogio, M. D., Weimer, M., and Interlandi, M. (2018). PRETZEL: opening the black box of machine learning prediction serving systems. In Arpaci-Dusseau, A. C. and Voelker, G., editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 611–626. USENIX Association.

[80] Lemmerich, F. and Becker, M. (2018). pysubgroup: Easy-to-use subgroup discovery in python. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part III*, volume 11053 of *Lecture Notes in Computer Science*, pages 658–662. Springer.

[81] Lemmerich, F., Rohlfs, M., and Atzmüller, M. (2010). Fast discovery of relevant subgroup patterns. In Guesgen, H. W. and Murray, R. C., editors, *Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference, May 19-21, 2010, Daytona Beach, Florida, USA*. AAAI Press.

[82] Li, G. (2017). Human-in-the-loop data integration. *Proc. VLDB Endow.*, 10(12):2006–2017.

[83] Li, L., Li, J., and Gao, H. (2015). Rule-based method for entity resolution. *IEEE Trans. Knowl. Data Eng.*

[84] Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W. (2020). Deep entity matching with pre-trained language models. *CoRR*, abs/2004.00584.

[85] Lissandrini, M., Mottin, D., Palpanas, T., and Velegrakis, Y. (2018). *Data Exploration Using Example-Based Methods*. Morgan and Claypool.

[86] Luo, S., Gao, Z. J., Gubanov, M., Perez, L. L., and Jermaine, C. (2017). Scalable linear algebra on a relational database system. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 523–534.

[87] Marchant, N. G. and Rubinstein, B. I. P. (2017). In search of an entity resolution OASIS: optimal asymptotic sequential importance sampling. *Proc. VLDB Endow.*, 10.

[88] May, N., Lehner, W., P., S. H., Maheshwari, N., Müller, C., Chowdhuri, S., and Goel, A. K. (2015). SAP HANA - from relational OLAP database to big data infrastructure. In Alonso, G., Geerts, F., Popa, L., Barceló, P., Teubner, J., Ugarte, M., den Bussche, J. V., and Paredaens, J., editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015*, pages 581–592. OpenProceedings.org.

[89] Meliou, A., Gatterbauer, W., Moore, K. F., and Suciu, D. (2010). The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45.

[90] Meliou, A., Roy, S., and Suciu, D. (2014). Causality and explanations in databases. *PVLDB*, 7(13):1715–1716.

[91] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., and et al. (2016a). Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.*, 17(1):1235–1241.

[92] Meng, X., Bradley, J. K., Yavuz, B., Sparks, E. R., Venkataraman, S., Liu, D., Freeman, J., Tsai, D. B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., and Talwalkar, A. (2016b). Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.*, 17:34:1–34:7.

[93] Menon, S. (2020). Market Share Analysis: Data Integration Tools, Worldwide, 2019. Gartner, https://www.gartner.com/doc/3987502/.

[94] Microsoft (2021). Predict in t-sql.

[95] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

[96] Miller, R. J. (2018). Open data integration. *Proc. VLDB Endow.*, 11(12):2130–2139.

[97] ML.NET (2020a). Samples. https://github.com/dotnet/machinelearning-samples.

[98] ML.NET (2020b). Sentiment analysis sample. https://github.com/dotnet/machinelearning-samples/tree/master/samples/csharp/getting-started/BinaryClassification_SentimentAnalysis.

[99] Moerkotte et al., G. (2014). Exploiting ordered dictionaries to efficiently construct histograms with q-error guarantees in SAP HANA. In *SIGMOD*.

[100] Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM.

[101] MySQL (2020). Limits on table column count and row size. https://dev.mysql.com/doc/mysql-reslimits-excerpt/5.7/en/column-count-limit.html.

[102] Nakandala, S., Saur, K., Yu, G.-I., Karanasos, K., Curino, C., Weimer, M., and Interlandi, M. (2020). A tensor compiler for unified machine learning prediction serving. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 899–917. USENIX Association.

[103] Naumann, F. and Herschel, M. (2010). *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[104] Netz, A., Bernhardt, J., Fayyad, U., and Chaudhuri, S. (2000). Integration of data mining and relational databases. In *Proceedings of the 26th International Conference on Very Large Databases*. Very Large Data Bases Endowment Inc.

[105] Ortona, S., Meduri, V. V., and Papotti, P. (2018). Robust discovery of positive and negative rules in knowledge bases. In *ICDE*.

[106] Paganelli, M., Beneventano, D., Guerra, F., and Sottovia, P. (2019a). Parallelizing computations of full disjunctions. *Big Data Res.*, 17:18–31.

[107] Paganelli, M., Buono, F. D., Guerra, F., and Ferro, N. (2020a). Unsupervised Evaluation of Data Integration Processes. Accepted at iiWAS 2020, but not yet published.

[108] Paganelli, M., Buono, F. D., Guerra, F., Pevarello, M., and Vincini, M. (2021). Automated Machine Learning for Entity Matching Tasks. In *EDBT*.

[109] Paganelli, M., Sottovia, P., Guerra, F., and Velegrakis, Y. (2019b). Tuner: Fine tuning of rule-based entity matchers. In *CIKM*.

[110] Paganelli, M., Sottovia, P., Interlandi, M., Guerra, F., and Park, K. (2020b). DBMS as ML Prediction Serving System. https://www.dropbox.com/s/6oqqyfhsb44ccsz/MASQ_Technical_report.pdf?dl=0. Technical Report.

[111] Paganelli, M., Sottovia, P., Maccioni, A., Interlandi, M., and Guerra, F. (2019c). Understanding data in the blink of an eye. In *CIKM*, pages 2885–2888. ACM.

[112] Paganelli, M., Sottovia, P., Maccioni, A., Interlandi, M., and Guerra, F. (2019d). Understanding data in the blink of an eye. In *CIKM 2019, 28th ACM International Conference on Information and Knowledge Management, 19, November 3–7, 2019, Beijing, China*.

[113] Paganelli, M., Sottovia, P., Maccioni, A., Interlandi, M., and Guerra, F. (2020c). Explaining data with descriptions. *Inf. Syst.*, 92:101549.

[114] Panahi, F., Wu, W., Doan, A., and Naughton, J. F. (2017). Towards interactive debugging of rule-based entity matching. In *EDBT*.

[115] Papadakis, G., Tsekouras, L., Thanos, E., Pittaras, N., Simonini, G., Skoutas, D., Isaris, P., Giannakopoulos, G., Palpanas, T., and Koubarakis, M. (2020). Jedai^3 : beyond batch, blocking-based entity resolution. In *EDBT*, pages 603–606. OpenProceedings.org.

[116] Papenbrock, T., Bergmann, T., Finke, M., Zwiener, J., and Naumann, F. (2015). Data profiling with metanome. *PVLDB*.

[117] Papineni, K., Roukos, S., Ward, T., and Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL.

[118] Passing, L., Then, M., Hubig, N., Lang, H., Schreier, M., Günnemann, S., Kemper, A., and Neumann, T. (2017). SQL- and operator-centric data analytics in relational main-memory databases. In Markl, V., Orlando, S., Mitschang, B., Andritsos, P., Sattler, K., and Breß, S., editors, *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, pages 84–95. OpenProceedings.org.

[119] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2012). Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490.

[120] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the EMNLP Conference, October 25-29, 2014, Doha, Qatar*. ACL.

[121] Polyzotis, N., Roy, S., Whang, S. E., and Zinkevich, M. (2018). Data lifecycle challenges in production machine learning: A survey. *SIGMOD Record*, 47(2):17–28.

[122] Pradhan, R., Bykau, S., and Prabhakar, S. (2017). Staging user feedback toward rapid conflict resolution in data fusion. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM.

[123] Psallidas, F., Zhu, Y., Karlas, B., Interlandi, M., Floratou, A., Karanasos, K., Wu, W., Zhang, C., Krishnan, S., Curino, C., and Weimer, M. (2019). Data science through the looking glass and what we found there. *CoRR*, abs/1912.09536.

[124] Ramachandra, K., Park, K., Emani, K. V., Halverson, A., Galindo-Legaria, C. A., and Cunningham, C. (2017). Froid: Optimization of imperative programs in a relational database. *Proc. VLDB Endow.*, 11(4):432–444.

[125] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144.

[126] Rodríguez, D., Ruiz, R., Riquelme, J. C., and Aguilar-Ruiz, J. S. (2012). Searching for rules to detect defective modules: A subgroup discovery approach. *Inf. Sci.*, 191:14–30.

[127] Roy, S., Orr, L., and Suciu, D. (2015). Explaining query answers with explanation-ready databases. *PVLDB*, 9(4):348–359.

[128] Roy, S. and Suciu, D. (2014). A formal approach to finding explanations for database queries. In *SIGMOD*, pages 1579–1590.

[129] Roy, S. B., Wang, H., Das, G., and et al. (2008). Minimum-effort driven dynamic faceted search in structured databases. In *CIKM, 2008*, pages 13–22.

[130] Schelter, S., Quinn, S., Marthi, S., and Musselman, A. (2016). Samsara: Declarative machine learning on distributed dataflow systems.

[131] Schüle, M., Bungeroth, M., Vorona, D., Kemper, A., Günnemann, S., and Neumann, T. (2019). ML2SQL - compiling a declarative machine learning language to SQL and python. In Herschel, M., Galhardas, H., Reinwald, B., Fundulaki, I., Binnig, C., and Kaoudi, Z., editors, *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 562–565. OpenProceedings.org.

[132] Schüle, M., Simonis, F., Heyenbrock, T., Kemper, A., Günnemann, S., and Neumann, T. (2019). In-database machine learning: Gradient descent and tensor algebra for main memory database systems. In Grust, T., Naumann, F., Böhm, A., Lehner, W., Härder, T., Rahm, E., Heuer, A., Klettke, M., and Meyer, H., editors, *BTW 2019*, pages 247–266. Gesellschaft für Informatik, Bonn.

[133] Scikit-learn (2019a). Impute. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.impute.

[134] Scikit-learn (2019b). Impute. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler.

[135] Scikit-learn (2019c). Pre-processing. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing.

[136] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J., and Dennison, D. (2015). Hidden technical debt in machine learning systems. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2503–2511.

[137] Server, M. S. (2020a). Maximum capacity specifications for sql server. https://docs.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server?redirectedfrom=MSDN&view=sql-server-ver15.

[138] Server, M. S. (2020b). Maximum number of "when then" lines in a case statement? https://www.sqlservercentral.com/forums/topic/maximum-number-of-when-then-lines-in-a-case-statement.

[139] Seshadri, N. and Sundberg, C.-E. W. (1994). List viterbi decoding algorithms with applications. *IEEE Transactions on Communications*, 42(234).

[140] Simonini, G., Bergamaschi, S., and Jagadish, H. V. (2016). BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *Proc. VLDB Endow.*, 9(12):1173–1184.

[141] Singh, R., Meduri, V. V., Elmagarmid, A. K., Madden, S., Papotti, P., Quiané-Ruiz, J., Solar-Lezama, A., and Tang, N. (2017). Synthesizing entity matching rules by examples. *Proc. VLDB Endow.*, 11.

[142] Smith, N. A. (2020). Contextual word representations: Putting words into computers. *Commun. ACM*, 63.

[143] Soroush, E., Balazinska, M., Krughoff, K. S., and Connolly, A. J. (2015). Efficient iterative processing in the scidb parallel array engine. *CoRR*, abs/1506.00307.

[144] Sottovia, P., Paganelli, M., Guerra, F., and Velegrakis, Y. (2019). Finding synonymous attributes in evolving wikipedia infoboxes. In *ADBIS*, volume 11695 of *Lecture Notes in Computer Science*, pages 169–185. Springer.

[145] Sparks, E. R., Venkataraman, S., Kaftan, T., Franklin, M. J., and Recht, B. (2017). Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 535–546.

[146] Supun Nakandala, Karla Saur, GyeongIn Yu, Konstantinos Karanasos, Carlo Curino, Markus Weimer, and Matteo Interland (2020). Taming Model Serving Complexity, Performance and Cost: A Compilation to Tensor Computations Approach. https://scnakandala.github.io/papers/TR_2020_Hummingbird.pdf. Technical Report.

[147] Syed, U. and Vassilvitskii, S. (2017). Sqml: Large-scale in-database machine learning with pure sql. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 659, New York, NY, USA. Association for Computing Machinery.

[148] Thomas, A. and Kumar, A. (2018). A comparative evaluation of systems for scalable linear algebra-based analytics. *Proc. VLDB Endow.*, 11(13):2168–2182.

[149] Troullinou, G., Kondylakis, H., Stefanidis, K., and Plexousakis, D. (2018). Exploring RDFS kbs using summaries. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, volume 11136 of *Lecture Notes in Computer Science*, pages 268–284. Springer.

[150] Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley series in behavioral science : quantitative methods. Addison-Wesley.

[151] Verroios, V., Garcia-Molina, H., and Papakonstantinou, Y. (2017). Waldo: An adaptive human interface for crowd entity resolution. In *SIGMOD Conference*, pages 1133–1148. ACM.

[152] Vesdapunt, N., Bellare, K., and Dalvi, N. N. (2014). Crowdsourcing algorithms for entity resolution. *Proc. VLDB Endow.*, 7.

[153] Wang, J., Kraska, T., Franklin, M. J., and Feng, J. (2012). Crowder: Crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5.

[154] Wang, J., Li, G., Yu, J. X., and Feng, J. (2011). Entity matching: How similar is similar. *PVLDB*.

[155] Wang, W., Gao, J., Zhang, M., Wang, S., Chen, G., Ng, T. K., Ooi, B. C., Shao, J., and Reyad, M. (2018). Rafiki: Machine learning as an analytics service system. *Proc. VLDB Endow.*, 12(2):128–140.

[156] Wang, X., Dong, X. L., and Meliou, A. (2015). Data x-ray: A diagnostic tool for data errors. In *SIGMOD 2015*, pages 1231–1245, New York, NY, USA. ACM.

[157] Whang, S. E. and Garcia-Molina, H. (2014). Incremental entity resolution on rules and data. *VLDB J.*

[158] Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In Komorowski, H. J. and Zytkow, J. M., editors, *Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, Trondheim, Norway, June 24-27, 1997, Proceedings*, volume 1263 of *Lecture Notes in Computer Science*, pages 78–87. Springer.

[159] Wu, E. and Madden, S. (2013). Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564.

[160] Wu, R., Chaba, S., Sawlani, S., Chu, X., and Thirumuruganathan, S. (2020). Zeroer: Entity resolution using zero labeled examples. In *SIGMOD Conference*, pages 1149–1164. ACM.

[161] Yan, Y., Zhang, J., Huang, B., Sun, X., Mu, J., Zhang, Z., and Moscibroda, T. (2015). Distributed outlier detection using compressive sensing. In *SIGMOD*, pages 3–16.

[162] Yang, X., Procopiuc, C. M., and Srivastava, D. (2009). Summarizing relational databases. *PVLDB*, 2(1):634–645.

[163] Yang, X., Procopiuc, C. M., and Srivastava, D. (2011). Summary graphs for relational database schemas. *PVLDB*, 4(11):899–910.

[164] Yoon, D. Y., Niu, N., and Mozafari, B. (2016). Dbsherlock: A performance diagnostic tool for transactional databases. In *SIGMOD*, pages 1599–1614.

[165] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65.

[166] Zhao, C. and He, Y. (2019). Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *WWW*, pages 2413–2424. ACM.

[167] Zinkevich, M. (2020). Rules of Machine Learning: Best Practices for ML Engineering. https://developers.google.com/machine-learning/guides/rules-of-ml.