Hasso Plattner Institute
Information Systems Group

Data Preparation and Domain-agnostic Duplicate Detection

Dissertation
zur Erlangung des akademischen Grades
"Doktor der Naturwissenschaften"
(Dr. rer. nat.)
in der Wissenschaftsdisziplin "Informationssysteme"

eingereicht an der
Fakultät Digital Engineering
der Universität Potsdam

von
Ioannis Koumarelas

Dissertation, Universität Potsdam, 2020

Reviewers

Prof. Dr. Felix Naumann
Hasso-Plattner-Institut, Universität Potsdam

Prof. Dr. Johann-Christoph Freytag
Humboldt-Universität zu Berlin

Prof. Dr. Norbert Ritter
Universität Hamburg

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-48913
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-489131

Abstract

Successfully completing any data science project demands careful consideration across its whole process. Although the focus is often put on later phases of the process, in practice, experts spend more time in earlier phases, preparing data, to make them consistent with the systems' requirements or to improve their models' accuracies. Duplicate detection is typically applied during the data cleaning phase, which is dedicated to removing data inconsistencies and improving the overall quality and usability of data. While data cleaning involves a plethora of approaches to perform specific operations, such as schema alignment and data normalization, the task of detecting and removing duplicate records is particularly challenging. Duplicates arise when multiple records representing the same entities exist in a database. Due to numerous reasons, spanning from simple typographical errors to different schemas and formats of integrated databases. Keeping a database free of duplicates is crucial for most use-cases, as their existence causes false negatives and false positives when matching queries against it. These two data quality issues have negative implications for tasks, such as hotel booking, where users may erroneously select a wrong hotel, or parcel delivery, where a parcel can get delivered to the wrong address. Identifying the variety of possible data issues to eliminate duplicates demands sophisticated approaches.

While research in duplicate detection is well-established and covers different aspects of both efficiency and effectiveness, our work in this thesis focuses on the latter. We propose novel approaches to improve data quality before duplicate detection takes place and apply the latter in datasets even when prior labeling is not available. Our experiments show that improving *data quality* upfront can increase duplicate classification results by up to 19%. To this end, we propose two novel pipelines that select and apply generic as well as address-specific *data preparation* steps with the purpose of maximizing the success of duplicate detection. Generic data preparation, such as the removal of special characters, can be applied to any relation with alphanumeric attributes. When applied, data preparation steps are selected only for attributes where there are positive effects on pair similarities, which indirectly affect classification, or on classification directly. Our work on addresses is twofold; first, we consider more domain-specific approaches to improve the quality of values, and, second, we experiment with known and modified versions of similarity measures to select the most appropriate per address attribute, e.g., city or country.

To facilitate duplicate detection in applications where gold standard annotations are not available and obtaining them is not possible or too expensive, we propose *MDedup*. MDedup is a novel, rule-based, and fully automatic duplicate detection approach that is based on *matching dependencies*. These dependencies can be used to detect duplicates and can be discovered using state-of-the-art algorithms efficiently and without any prior labeling. MD-edup uses two pipelines to first train on datasets with known labels, learning to identify useful matching dependencies, and then be applied on unseen datasets, regardless of any existing gold standard. Finally, our work is accompanied by open source code to enable repeatability of our research results and application of our approaches to other datasets.

Zusammenfassung

Die erfolgreiche Durchführung eines datenwissenschaftlichen Projekts erfordert eine Reihe sorgfältiger Abwägungen, die während des gesamten Prozessesverlaufs zu treffen sind. Obwohl sich der Schwerpunkt oft auf spätere Prozessphasen konzentriert, verbringen Experten in der Praxis jedoch einen Großteil ihrer Zeit in frühen Projektphasen in denen sie Daten aufbereiten, um sie mit den Anforderungen vorhandener Systeme in Einklang zu bringen oder die Genauigkeit ihrer Modelle zu verbessern. Die Duplikaterkennung wird üblicherweise während der Datenbereinigungsphase durchgeführt, sie dient der Beseitigung von Dateninkonsistenzen und somit der Verbesserung von Gesamtqualität und Benutzerfreundlichkeit der Daten. Während die Datenbereinigung eine Vielzahl von Ansätzen zur Durchführung spezifischer Operationen wie etwa dem Schema-Abgleich und der Datennormalisierung umfasst, stellt die Identifizierung und Entfernung doppelter Datensätze eine besondere Herausforderung dar. Dabei entstehen Duplikate, wenn mehrere Datensätze, welche die gleichen Entitäten repräsentieren, in einer Datenbank vorhanden sind. Die Gründe dafür sind vielfältig und reichen von einfachen Schreibfehlern bis hin zu unterschiedlichen Schemata und Formaten integrierter Datenbanken. Eine Datenbank duplikatfrei zu halten, ist für die meisten Anwendungsfälle von entscheidender Bedeutung, da ihre Existenz zu falschen Negativ- und Falsch-Positiv-Abfragen führt. So können sich derartige Datenqualitätsprobleme negativ auf Aufgaben wie beispielsweise Hotelbuchungen oder Paketzustellungen auswirken, was letztlich dazu führen kann, dass Benutzer ein falsches Hotel buchen, oder Pakete an eine falsche Adresse geliefert werden. Um ein breites Spektrum potenzieller Datenprobleme zu identifizieren, deren Lösung die Beseitigung von Duplikaten erleichtert, sind eine Reihe ausgefeilter Ansätze erforderlich.

Obgleich der Forschungsbereich der Duplikaterkennung mit der Untersuchung verschiedenster Effizienz und Effektivitätsaspekte bereits gut etabliert ist, konzentriert sich diese Arbeit auf letztgenannte Aspekte. Wir schlagen neue Ansätze zur Verbesserung der Datenqualität vor, die vor der Duplikaterkennung erfolgen, und wenden letztere auf Datensätze an, selbst wenn diese über keine im Vorfeld erstellten Annotationen verfügen. Unsere Experimente zeigen, dass durch eine im Vorfeld verbesserte Datenqualität die Ergebnisse der sich anschließenden Duplikatklassifizierung um bis zu 19% verbessert werden können. Zu diesem Zweck schlagen wir zwei neuartige Pipelines vor, die sowohl generische als auch adressspezifische Datenaufbereitungsschritte auswählen und anwenden, um den Erfolg der Duplikaterkennung zu maximieren. Die generische Datenaufbereitung, wie z.B. die Entfernung von Sonderzeichen, kann auf jede Relation mit alphanumerischen Attributen angewendet werden. Bei entsprechender Anwendung werden Datenaufbereitungsschritte nur für Attribute ausgewählt, bei denen sich positive Auswirkungen auf Paarähnlichkeiten ergeben, welche sich direkt oder indirekt auf die Klassifizierung auswirken. Unsere Arbeit an Adressen umfasst zwei Aspekte: erstens betrachten wir mehr domänenspezifische Ansätze zur Verbesserung der Adressqualität, zweitens experimentieren wir mit bekannten und modifizierten Versionen verschiedener Ähnlichkeitsmaße, um infolgedessen das am besten geeignete Ähnlichkeitsmaß für jedes Adressattribut, z.B. Stadt oder Land, zu bestimmen.

Um die Erkennung von Duplikaten bei Anwendungen zu erleichtern, in denen Goldstandard-Annotationen nicht zur Verfügung stehen und deren Beschaffung aus Kostengründen nicht möglich ist, schlagen wir MDedup vor. MDedup ist ein neuartiger, regelbasierter und vollautomatischer Ansatz zur Dublikaterkennung, der auf Matching Dependencies beruht. Diese Abhängigkeiten können zur Erkennung von Duplikaten genutzt und mit Hilfe modernster Algorithmen effizient ohne vorhergehenden Annotationsaufwand entdeckt werden. MDedup verwendet zwei Pipelines, um zunächst auf annotierten Datensätzen zu trainieren, wobei die Identifizierung nützlicher Matching-Abhängigkeiten erlernt wird, welche dann unabhängig von einem bestehenden Goldstandard auf ungesehenen Datensätzen angewendet werden können. Schließlich stellen wir den im Rahmen dieser Arbeit entstehenden Quellcode zur Verfügung, wodurch sowohl die Wiederholbarkeit unserer Forschungsergebnisse als auch die Anwendung unserer Ansätze auf anderen Datensätzen gewährleistet werden soll.

Acknowledgements

This PhD is a result of many influential people I was lucky enough to have in my life, who nurtured me with their positive energy and guidance. I want to begin by dedicating this work to my PhD advisor Prof. Felix Naumann who gave me this great opportunity to begin with, but also provided me with constant support, great amounts of positivity, and had the experience to make hard decisions at the right time. During these years and under his supervision, I learned new scientific concepts, improved on my discipline, and learned to pay attention on details even more. Thanks to him, I feel a lot more confident as a scientist and more optimistic about my next steps, whatever they are.

During all these years of university studies, I had the opportunity to meet exceptional people at the chair with whom I collaborated, supported each other, but also learned from. Some of them guided me in the beginning and during my PhD, to some others I wish I was helpful at the end of it, and with some, we walked this journey together. I hope you all have great careers and certainly wish to stay connected afterwards.

Finally, I want to thank my family for standing by me through all the difficult times we faced and for their constant love, especially at times when I was on the edge of collapsing. Following my dreams has not been an easy task and would certainly not be possible without your support and motivation.

Contents

1 Duplicate Detection

Computer applications have managed to improve the quality of human life tremendously over the last decades. In various application domains, such as health care and business analytics, making decisions in the 21st century heavily depends on acquiring and intelligently processing the right data. While improvements in computer hardware and software have greatly improved the performance of applications, the quality of the produced results still heavily depends on the quality of the provided input, i.e., data. To this end, data scientists have developed several techniques to identify common issues that occur in data and appropriate methods to repair them.

In general, these data quality issues can be loosely classified to two main categories: syntactical and semantical. *Syntactical* issues are typically easy to identify, and their resolutions are quite straightforward; for instance, improper file encoding (ASCII instead of UTF-8) or inclusion of redundant special characters due to poor parsing. These issues are usually resolved using *data preparation* techniques [Kandel et al., 2011; Yang et al., 2015]. *Semantical* inconsistencies, on the other hand, are much harder to identify and exist in multiple forms, such as alternative representations and schema mismatches. These representations can be synonyms or variations of a first name, or the existence of duplicate entries of the same entity. For these issues we employ *data cleaning* techniques, which usually involve more complex and sophisticated processes [Elmagarmid et al., 2007; Oliveira et al., 2005].

Although repairing syntactical issues can indirectly improve semantical problems as well, which we discuss later, in this thesis we focus on a particular semantic problem, that of *duplicate entries.* The existence of duplicate entries is a prevalent problem, especially in applications where multiple data sources are considered or merged into a single one, which happens, for instance, when integrating different departments of a company. The two main characteristics that make it a very challenging task is a typically large number of records that need to be compared and non-identical values in one or more of the records' fields across the duplicate entries.

Since duplicates exist in the first place due to inconsistencies in the recorded values, two main problematic situations can take place. First, if the system is not able to match a user's query to the appropriate record, the missed and correct record is characterized as a *false negative.* Second, if the system matches a user's query to a wrong record, the erroneous record is labeled as a *false positive.* Considering these issues in an example domain, that of lodging, a customer can be unsatisfied if she is not able to book a hotel she has in mind, although it exists in the system with a different representation (false negative), or the situation could become even more complicated if the system matches her query to a wrong hotel (false positive). Such issues exist in other domains too, such as parcel delivery, where parcels can be sent to wrong destinations. Companies lose much money due to such data quality issues, with the U.S. alone losing approximately $3.1 trillion per year, according to IBM's calculations [IBM Big Data Analytics Hub, 2016].

In the past, detecting duplicates was an easier task, since most datasets were smaller than what we face currently and were produced by fewer sources, which were more commonly conforming to predefined standards. However, with the increased amount of generated data, our job as data scientists becomes even more difficult. On the one hand, we must take into account that more diverse users and systems generate these data, which usually translates to poorer data quality. On the other hand, the amounts of data we have to process demand for more efficient solutions. Although adequate literature already exists for both these issues [Christen, 2012b; Chu et al., 2016; Elmagarmid et al., 2007], the problem is far from being solved.

To contribute to this research area, in this thesis we focus on improving the quality of record matching, i.e., less falsely identified pairs of duplicate entries as non-duplicate and vice versa. To this end, Chapter 3, based on generic data preparation operations, and Chapter 4, based on address normalization, focus on improving the quality of the provided data in a systematic way, which translates to improved quality in duplicate detection classification metrics. Finally, Chapter 5, based on matching dependencies, provides a novel pipeline for scenarios where no prior labeling of duplicate pairs is available. Having a set of true duplicates is essential to train a machine learning model to distinguish them from non-duplicates.

In this introductory chapter, we first provide an overview of typical data quality issues that may cause the existence of duplicates. Afterward, we discuss the challenges during duplicate detection and what are the commonly applied steps. We then introduce the research problems we consider in this thesis along with our solutions. Finally, we conclude with our concrete contributions and set the ground for the remaining of this thesis with the structure of the following chapters.

1.1 Data quality issues

In order to eliminate duplicate entries successfully, it is essential to begin by considering some common data quality issues that are the reasons duplicates emerge in the first place. The problem is well-known in research and taxonomies have already been proposed to describe the plethora of data quality problems [Oliveira et al., 2005; Rahm and Do, 2000]. However, although many of these data quality problems are met across multiple application domains and can have universal solutions, domain-specific issues are commonly present as well, which call for specialized techniques from domain experts. Typical data quality problems, which are relevant for duplicate detection, include:

- **Completeness**: How many attribute values are missing across records? If these attributes contain essential information, this can make matching duplicate records a challenging process.
- **Formatting**: Are different data sources using the same data formats? A typical example is often found in calendar dates, where if both the formats day-month-year (DMY) and month-day-year (MDY) are used across different data sources,

mistakes can be made by comparing days to months.

- **Spelling errors**: Are the values generated automatically by a system or entered manually by users? If values are provided by users, for instance during data entry processes, typographical mistakes in the expected input value of additional, modified, or removed alphanumeric characters, are very prone to take place.
- **Timeliness**: Are the records updated consistently across time? A common problem that causes duplicates is that information about entities changes across time, without being updated in the database. Therefore, when records of the same entity are inserted into the database in the future, the differences are quite significant, making record matching a more challenging task.
- **Validity**: Can we trust the stored values? This problem is related to *spelling errors*, but can be the result of more severe issues. Due to poor parsing or problematic user forms, values may end up in different attributes. For instance, a user might provide the full address, including the city and country, in the field about the street address of a place. In such cases, we need to parse and extract individual values, placing them in their respective attributes.
- **Data volume**: How many data have to be processed? This greatly dictates what kind of duplicate detection approaches can be used. For instance, if the number of records is large and their attributes contain large alphanumeric values, using edit-based similarity measures to compare values on characters, such as Levenshtein [Levenshtein, 1966], is not always the best choice semantically. Thus, token-based similarity measures have to be used instead, such as Jaccard [Jaccard, 1901].

The former problem categories are also discussed by Christen [Christen, 2012b], among other issues, and provide us with an idea of what we face when cleaning a database. Inability to identify and resolve these issues causes a number of problems for both customers and companies, having an actual massive cost for the latter ones, as we referred to previously. Therefore, the pressure to identify data quality issues and resolve them is quite high. To this end, in the next section, we focus on duplicate detection and consider the main challenges around it along with typical processes applied to resolve them.

1.2 Challenges of duplicate detection and existing solutions

Having seen some issues that cause duplicate records and a few examples, should provide us with a basic understanding of how challenging the problem is. Initially, the problem was defined by Newcombe et al. [Newcombe et al., 1959] in 1959 and Fellegi and Sunter [Fellegi and Sunter, 1969] formalized it in 1969. Since then, the problem has been the target of research from multiple domains and has been recognized with different names, including duplicate detection [Naumann and Herschel, 2010], record linkage [Christen, 2012b], entity resolution [Christen et al., 2009], and data matching [Christen et al., 2006], all of which are ironically *duplicates* of the same research area. Most approaches are similar for all these versions of the problem. Indeed, in certain setups, such as when we have two relations and record linkage is applied, specific properties of individual relations could be exploited. For instance, if no duplicates exist within individual relations, this could be taken into account if some record is matched with more than one record of the other relation. For simplicity, we decided to focus on *duplicate detection*, which is defined in the context of a single relation. Record linkage tasks can be solved by duplicate detection approaches by first merging the different relations into a single one, which of course causes some loss of information.

Over the years, many approaches have been developed to detect and resolve duplicate records [Christen, 2012b; Elmagarmid et al., 2007; Naumann and Herschel, 2010]. Because the challenges around duplicate detection are numerous many of these approaches focus on specific parts of the process. A typical pipeline of the whole process is provided by Christen [Christen, 2012b] and is shown in Figure 1.1. Every step provides a solution to a different challenge around duplicate detection. The challenges and solutions across these steps are the following:

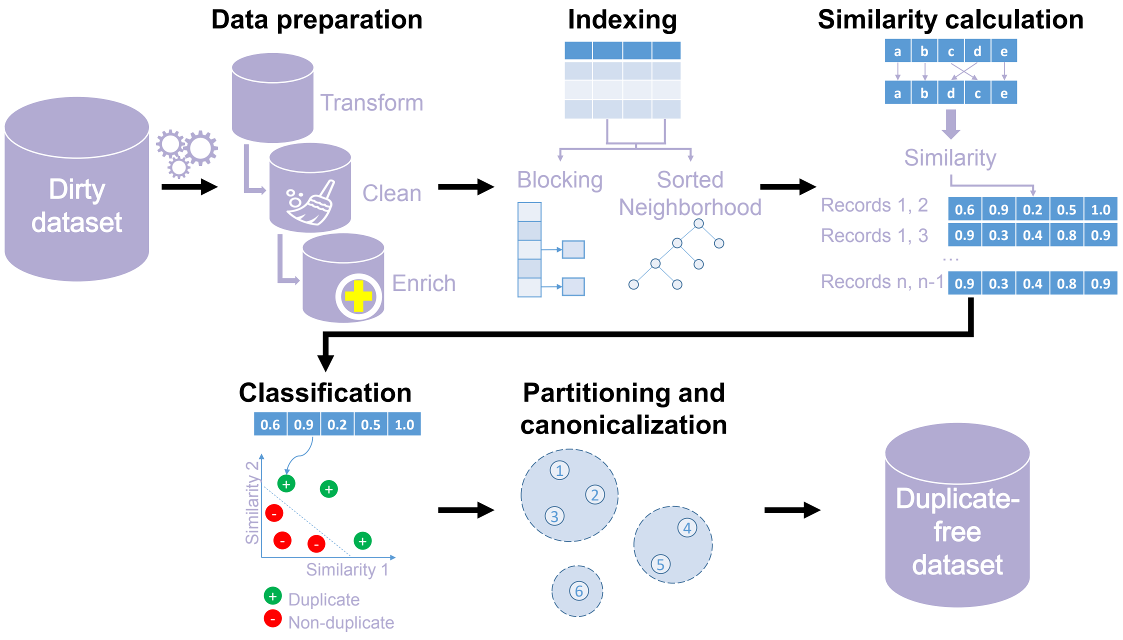


Figure 1.1: Commonly applied duplicate detection pipeline.

Poor data quality. First, similarly to how duplicate detection is applied before other processes start to improve data quality, certain steps could be applied to make duplicate detection feasible and more successful. These are part of the first step in Figure 1.1 and span from fixing capitalization issues by lower-casing characters to more fine-grained details, such as address normalization, to provide a more robust foundation for later parts of the process. In fact, as we show in Chapters 3 and 4, by focusing on this step alone and using simple approaches for the following steps of the process, we can improve the success of duplicate detection substantially, by up to 19% in F-measure.

Too many comparison candidates. Second, the duplicate detection process typically involves a single relation. Therefore, given a relation R of n records, deciding which pairs of records represent the same entity requires an exhaustive all-pair comparison, which has a complexity of $\mathcal{O}(\frac{n*(n-1)}{2}) = \mathcal{O}(n^2)$. For large relations, even if they include only a few thousand records, this number of comparisons could become prohibitive for executing within a reasonable time frame. The complexity is similarly high for the record linkage version of the problem, which involves two relations R and S of sizes n and m, with an $\mathcal{O}(n * m)$ complexity. To reduce the number of comparisons, data cleaning experts typically apply different indexing approaches, such as *blocking* [Li et al., 2013] and *sorted neighborhood* [Ramadan et al., 2015], which are based on hashtable and ordered tree data structures. The result of this process, which is the second step in Figure 1.1, provides us with a set of candidate duplicate pairs to be compared next.

Quantifying similarity between records. Third, before we can decide whether two records are duplicates, we need some features to base our decisions on. Typically, these features are generated based on attribute values of the two records. Based on these attribute values, similarity (dissimilarity) measures are used that provide us with arithmetic representations of their closeness (distance). Different measures can be used, depending on the attribute type; for instance, when comparing calendar dates, a higher distance should be returned between two values if they differ in years instead of days. Alphanumeric values are very typical, as they can express a wide range of information, and a plethora of different measures exist, such as Levenshtein and Jaccard [Jaccard, 1901; Levenshtein, 1966]. Various alphanumeric measures exist to quantify differences on multiple granularity levels, such as character or token. For instance, large alphanumeric values, such as descriptions, should usually focus on whether the two values share the same tokens instead of counting differences on characters that are more important for smaller values, such as cities' or persons' names.

Deciding if records are duplicates. Fourth, these numerical features can be utilized under different classification approaches to decide whether a given pair of records represents a duplicate or not. A wide range of classification approaches have been used in duplicate detection, spanning from simple models, such as the linear classifier, also known as the *Threshold-based* classifier, to more sophisticated ones, such as *Support Vector Machines* (SVMs) and more recently *Deep Neural Network* models, such as DeepER [Ebraheem et al., 2018] and DeepMatcher [Mudgal et al., 2018]. Although sophisticated models can usually achieve better results, simpler models are more useful as they can be explained and even fine-tuned easier by non-technical domain experts. Typically, when evaluating a duplicate detection process, the outcome of this phase is used directly.

Resolving duplications. Finally, after we decide which pairs represent duplicates, to eliminate the duplicates certain actions have to be made. These actions usually include producing *clusters* of records that represent duplicates and *merging* each of these clusters into new records that uniquely represent each entity. Clustering is performed using typical graph clustering techniques [Draisbach et al., 2019], where nodes are records and edges are pairs that were classified as duplicates in the previous step. After clustering is complete, for non-single element clusters, disagreements between records' attribute values have to be addressed and resolved so that a new record will be generated as the *canonical representation* of the entity [Culotta et al., 2007].

This section addressed general steps and respective approaches typically applied in a duplicate detection process. Next, we motivate the research problems that we considered of paramount importance during this thesis and briefly introduce our approaches to resolve them.

1.3 Research challenges and contributions

While the open research problems in the topic of duplicate detection are numerous, our choices of focus were made in collaboration with our industry partner *SAP Concur*[1]. SAP Concur provides business solutions to travel and expense issues. Part of their business pipeline involves matching user queries to different types of companies contained in their databases. Our collaboration focused on the area of lodging/hotels, where their databases contained records located worldwide. Although they had and have a successful deduplication process pipeline, they were looking for ways to improve it further.

During our collaboration, we experimented with various datasets, generated out of their different business processes. However, in this thesis, we focus our research on a single dataset, where we had enough gold standard information to evaluate our approaches. This dataset revealed a plethora of issues we could address. Therefore, we decided to apply general data preparation operations, as discussed in Chapter 3. After managing to improve and remove a number of different issues in most values, we identified addresses to be a core component of their data when comparing records. Any discussion regarding addresses is beneficial in multiple domains as they characterize a wide range of different use-cases. Although addresses are prepared and have their quality improved in the work of Chapter 3, our solution discussed in Chapter 4, evaluated further approaches on how to normalize and compare addresses, in more detail. Finally, being able to identify a set of duplicate pairs in unknown sources or even in different domains, lead us to a rule-based approach based on matching dependencies, discussed in Chapter 5. Matching dependencies provide us with a set of duplicate pairs, typically of high precision, while at the same time, their properties are more straightforward to comprehend than other machine learning solutions.

To better explain the previous points, let us consider Table 1.1, which includes modified records from the Restaurants dataset, which is a merged dataset of two Restaurant guides (see Chapter 3 for more details). The first two pairs of records are *duplicates*, whereas the latter two ones *non-duplicates*, but with high similarity. Several of the presented values include inconsistencies that need to be repaired. To this end, we suggest the following steps, which are the methodologies described in this thesis to repair these values and effectively match the duplicate records, avoiding the non-duplicate ones.

Data preparation. Several of the presented values include inconsistencies that specific data preparation operations, such as *lower-casing* and *removal of special characters*, could repair. Indeed, by applying these operations, we obtain Table 1.2, with values of higher quality. This is also reflected in similarities of both duplicate and non-duplicate pairs. Ideally, increased similarities should only happen for duplicates. However, when the effect takes place also in non-duplicates, it is sufficient for us that these similarity increases are smaller than for duplicates and, thus, do not cross classification borders so that non-duplicates are mistaken for duplicates.

Noticing the benefits of prepared values on their respective similarities, led to the research approach discussed in Chapter 3. Briefly, a *systematic data preparation* pipeline is suggested to identify the proper data preparation operators per attribute. Requiring a small sample of labeled pairs and using eleven selected data preparation operations, suitable for duplicate detection, our two-step heuristic approach identifies appropriate

[1] `https://www.concur.com/`

Table 1.1: *Original values, motivation for data preparation:* A sample of duplicate (<163,164> and <165,166>) and non-duplicate record pairs (<180,823> and <676,811>) from the Restaurants dataset, with some value modifications for the purposes of this example. The floating point numbers indicate value pair similarities.

id	name	phone	address	city	type
163	> georgia grille < /	404/352-3517	2290 peachtree rd. peachtree square shopping center	atlanta city	american
164	Georgia grille	404-352-3517	2290 Peachtree RD.	Atlanta	Southwestern
	0.68	0.92	0.29	0.5	0.17
165	hedgerose heights inn	404/233-7673	490 e. paces ferry rd.	atlanta	international
166	Hedgerose heights Inn the	404-233-7673	490 E. Paces Ferry RD. ne	Atlanta	Continental
	0.76	0.92	0.68	0.86	0.31
180	;ritz carlton cafe buckhead;	404/237-2700	3434 peachtree rd. ne	atlanta	american new
823	Ritz carlton cafe atlanta	404-659-0400	181 peachtree ST.	Atlanta	American new
	0.64	0.5	0.57	0.86	0.92
676	} johny rockets la {	213-651-3361	7507 melrose ave	la	american
811	} Johny rockets AT {	770-955-6068	2970 Cobb PKWY	Atlanta	American
	0.85	0.33	0.12	0.29	0.88

data preparations in a best-effort scenario. A heuristic approach is necessary, as considering all possible preparations in combination for all attributes cannot be calculated within a reasonable time-frame. Using our approach we managed to improve F-measure by up to 19%, according to our evaluation.

Address normalization. Although using the previous data preparation process can already improve the classification results remarkably, there is still room for further improvement, especially regarding addresses. By using the geocoding services of Nominatim[2], which is an online geographic information system, we obtain new and normalized values presented in Table 1.3. Apart from the values visible to this table, *geocoding*, which matches the user's query to the appropriate address, returns us extra information, such as the geolocation (latitude and longitude). Therefore, apart from *value normalization*, we also *enrich* our records with further information that can be exploited during the comparison of records or even used by a later application. For instance, duplicate detection could take advantage of geolocations and disconsider records that are located too far apart, depending on the use-case.

To this end, in Chapter 4 we present a pipeline that improves duplicate detection results of records that include addresses in two ways. First, we experiment with three different systems to improve the quality of address values. Second, for every address attribute, we experiment with all possible similarity measures and identify the most suitable one, resulting in the optimal duplicate detection classification. Applying our

[2]https://nominatim.openstreetmap.org/

Table 1.2: *Prepared values, motivation for address normalization:* Continuation of Table 1.1 with data preparation applied over attribute values. Although, similarities have improved a lot for duplicate pairs, there is still room for improvement, in particular for address-related attributes.

id	name	phone	address	city	type
163	georgia grille	404 352 3517	2290 peachtree rd peachtree square shopping center	atlanta city	american
164	georgia grille	404 352 3517	2290 peachtree rd	atlanta	southwestern
	1.0 (+0.32)	1.0 (+0.08)	0.34 (+0.05)	0.58 (+0.08)	0.17
165	hedgerose heights inn	404 233 7673	490 e paces ferry rd	atlanta	international
166	hedgerose heights inn the	404 233 7673	490 e paces ferry rd ne	atlanta	continental
	0.84 (+0.08)	1.0 (+0.08)	0.88(+0.2)	1.0 (+0.14)	0.31
180	ritz carlton cafe buckhead	404 237 2700	3434 peachtree rd ne	atlanta	american new
823	ritz carlton cafe atlanta	404 659 0400	181 peachtree st	atlanta	american new
	0.73 (+0.09)	0.58 (+0.08)	0.55 (−0.02)	1.0 (+0.14)	1.0 (+0.08)
676	johny rockets la	213 651 3361	7505 melrose ave	la	american
811	johny rockets at	770 955 6068	2970 cobb pkwy	atlanta	american
	0.88 (+0.03)	0.33	0.12	0.29	1.0 (+0.12)

pipeline led to an increase in F-measure up to 12%, as shown by our evaluations.

Matching dependency rules. Having prepared data with normalized and enriched addresses improves duplicate detection results substantially. However, even under this scenario, performing duplicate detection requires a set of labeled pairs to train a classification model and apply it to other unknown pairs. This is a fundamental issue for most machine learning problems.

For this purpose, in Chapter 5, we propose MDedup, a novel pipeline based on matching dependencies that can be applied to unknown datasets for the purpose of duplicate detection. In a nutshell, our approach utilizes the fact that matching dependencies can be discovered in any relational dataset, without any relevant gold standard. What constitutes our approach novel is that it learns characteristics over successful sets of matching dependencies in labeled datasets and applies them to unlabeled datasets. These characteristics can be calculated based on information that is available in both unlabeled and labeled datasets. The obtained results are characterized by high precision, which is particularly important in applications where false positives have to be avoided. In particular, our experimental evaluation shows that using our approach we can achieve up to 94% F-measure and 100% precision.

Across Chapters 3 to 5, we provide a set of experiments that in total involve nine different datasets, covering a range of different domains. One of these datasets is a real-world dataset provided by our industry partner SAP Concur, comprised of hotels, which is more challenging as it includes a lot more unconventional issues. This emphasizes the predominant flavor of this thesis, which is to provide practical improvements in the

Table 1.3: *Prepared and address normalized values, motivation for matching dependencies:* Continuation of Table 1.2 with data preparation and address normalization applied over attribute values. The quality of values has improved quite a lot and a rule-based method, such as ours based on matching dependencies, could identify duplicate pairs with less room for errors.

id	name	phone	address	city	type
163	georgia grille	404 352 3517	2290 peachtree rd	atlanta	american
164	georgia grille	404 352 3517	2290 peachtree rd	atlanta	southwestern
	1.0	1.0	1.0 (+0.66)	1.0 (+0.42)	0.17
165	hedgerose heights inn	404 233 7673	490 east paces ferry rd ne	atlanta	international
166	hedgerose heights inn the	404 233 7673	490 east paces ferry rd ne	atlanta	continental
	0.84	1.0	1.0 (+0.12)	1.0	0.31
180	ritz carlton cafe buckhead	404 237 2700	3434 peachtree rd ne	atlanta	american new
823	ritz carlton cafe atlanta	404 659 0400	181 peachtree st ne	atlanta	american new
	0.73	0.58	0.7 (+0.15)	1.0	1.0
676	johny rockets la	213 651 3361	7505 melrose ave	los angeles	american
811	johny rockets at	770 955 6068	2970 cobb pkwy se	atlanta	american
	0.88	0.33	0.12 (−0.01)	0.18(−0.1)	1.0

context of duplicate detection. Finally, although in this thesis we have mostly emphasized the impact in improving matching quality metrics, we ensure and validate that all our approaches execute in reasonable execution times.

1.4 Structure

The remainder of this thesis is structured as follows: In Chapter 2, we discuss related work in the context of duplicate detection. Then, we introduce our generic data preparation approach in Chapter 3 [Koumarelas et al., 2020a]. Data preparation was initially a working topic from Jiang and Naumann, while Koumarelas had the idea of combining it with duplicate detection. Both Jiang and Naumann provided suggestions through the whole project. In Chapter 4 we present our pipeline for normalizing and matching address-related records [Koumarelas et al., 2018]. Datasets and domain knowledge were provided by Mosley, prototyping and data analysis support by Kroschk, and suggestions and supervision by Naumann. Afterward, we present the domain-agnostic duplicate detection approach MDedup in Chapter 5 [Koumarelas et al., 2020b]. The idea was based on the work of Schirmer [Schirmer et al., 2020], which discovers matching dependencies (MDs). Koumarelas extended the line of work by using the discovered MDs to detect duplicates, by focusing on selecting those MDs that can bring the best results. Papenbrock and Naumann both provided essential insights and supervision to the project. Finally, we conclude in Chapter 6 with a synopsis of this thesis and open research questions for future work on the topic of duplicate detection.

2

Duplicate Detection Approaches

Data quality issues have been challenging researchers and practitioners over many years, which has resulted in contributions to improve different aspects of data, such as schema and values [Han et al., 2011; Lee et al., 2009; Wang et al., 2001]. Out of the many important data quality issues that exist, in this thesis, we consider the resolution of duplicates as the primary focus of our studies. However, certain quality issues are connected in a way that resolving some of them can have positive implications for the rest. This is evident in Chapters 3 and 4, where the preparation of values improves duplicate detection. Likewise, resolving duplicates can have positive implications for other data quality aspects, such as low information retrieval scores. Therefore, in this chapter, we discuss literature in the area of duplicate detection and use the commonly applied steps discussed in Figure 1.1 to group it. We omit literature of data preparation (first step in Figure 1.1), which is discussed in Chapter 3.

The remainder of this section is structured as follows: First, in Section 2.1, we present indexing approaches to reduce the number of record pairs being compared. Second, in Section 2.2, we discuss different ways of generating features to compare records, with similarity measures being the most common ones. Third, in Section 2.3, using the features generated from the previous step, we present various classification approaches to decide whether two records represent duplicates or not. Finally, Section 2.4 completes the typically applied procedure by discussing the last steps used to provide single representations for the detected duplicates.

2.1 Indexing

Deciding which pairs represent duplicates or not demands for a full-pair comparison of a dataset's records. However, processing this number of pairs has a complexity of $\mathcal{O}(n^2)$, where n is the number of records. For large datasets, this quadratic complexity makes the execution infeasible to complete within reasonable time constraints. Therefore, *indexing* solutions are used to reduce the pair comparisons only to those that are more probable to be actual duplicates, avoiding redundant comparisons. The most commonly applied indexing strategies [Christen, 2012b; Elmagarmid et al., 2007] are the following:

Blocking. This indexing method is based on *Hashtables* and is the most commonly applied due to its simplicity [Baxter et al., 2003; Karakasidis et al., 2015; Papadakis et al., 2013]. Analogous to hashing keys are the *blocking keys*, which define how to generate values from records that assign them to blocks, similar to Hashtable buckets. Records are assigned to blocks, with every block including multiple records. Within a block all records are compared in pairs in the next step (Section 2.2). Best practices suggest to use parts of the records that are discriminative and can set them apart from values of different entities, but at the same time flexible enough to allow variations of the values to be placed in the same block. For instance, a blocking key for Table 1.3 could be a concatenation of records' name and address values. Ideally, users should target producing high-quality buckets, with only a few mistakenly placed records (false positives). Multiple blocking keys can be introduced to address the problem of missed pairs (false negatives) by utilizing different attributes of the record. As an example, considering again Table 1.3 one blocking key could use a combination of the restaurant's name and the street address, whereas a second one could use the phone number and the city. Combining multiple blocking keys increases the probability that duplicates are placed in some block, regardless of inconsistencies in some records' values.

Another commonly applied technique, often considered as a separate category, is *canopy clustering* [McCallum et al., 2000]. Canopy clustering allows for a more relaxed form of blocks to be formed, which are usually overlapping and cheap to calculate. The motivation is to prune the vast majority of record pairs in this step and move on to a more robust indexing approach in the following second step. A typical approach used for canopy clustering is locality-sensitive hashing (LSH), which uses hashing functions, such as min-hashing that approximates the Jaccard similarity, to assign similarly enough values to the same buckets.

Sorted Neighbourhood. While blocking approaches have their advantages, such as being simple models with a linear runtime complexity, sorted neighbourhood method (SNM) approaches offer additional benefits by increasing, however, the runtime complexity to $\mathcal{O}(n \log n)$, where n is the number of records. In particular, the main benefit of SNM approaches is that values are ordered and thus alphabetically close values are considered as duplicate candidates [Hernández and Stolfo, 1998; Ramadan et al., 2015]. Similar to blocking keys, *sorting keys* are created from the record's attributes, producing a value that represents the record and should ideally be alphabetically close to other duplicate entries of the same entity. Values generated from sorting keys are first sorted, and then neighbouring values are considered as duplicate candidates, based on different strategies. These strategies are typically based on a sliding window, which can be of fixed size or have its size be decided based on similarity criteria. Typically, this window is slid from the left of the ordered list of values to the right. Records within a window form candidate pairs to be compared in the next step of value comparison.

An inherent assumption of SNM is that when sorting key values contain inconsistencies, these happen more at the end of a value and not at the beginning. Similar to blocking, SNM approaches can also apply multiple sorting keys to increase the probability of duplicate records being alphanumerically close by the values of one of them. Alternatively, different transformations on top of the generated values may help mitigate some of these problems, which are discussed followingly:

Improving Blocking and Sorted Neighbourhood. While using blocking and sorted neighbourhood can produce high-quality results already with normal blocking and sorting keys, there are several techniques to improve their success rate further.

- Phonetic encodings: Many spelling mistakes are the result of miscommunicating an orally transmitted phrase. The information entered in the database sounds acoustically similar but spelled incorrectly. To identify such cases, several *phonetic encoding* methods, such as Soundex [Odell and Russell, 1918] and Metaphone [Philips, 1990], have been developed that transform the values to an alphanumeric representation that is identical if the written values "sound" the same. For instance, given the values `Berlin` and `Bearlyn`, using the Metaphone encoding both would be transformed to `BRLN`. This means that if blocking would use Metaphone as blocking key, both values and thus records would end to the same block.

- N-grams: In case the inconsistencies are more complicated or the values are too large for phonetic encodings to be useful, using n-grams, producing *multiple sorting keys*, can be an alternative and quite effective approach [Gravano et al., 2001]. This approach works as follows: a window of size n is slid from the start to the end of the alphanumeric value, producing multiple tokens each of size n, each one of which is used as a separate sorting key. In particular, for an alphanumeric value that contains m characters, $m - n + 1$ tokens are produced, also referred to as n-grams. For instance, generating 3-grams for the value `Berlin` would result in the following tokens: ["Ber", "erl", "rli", "lin"]. Additionally, if padding is enabled the list would also include: ["␣␣B", "␣Be", "in␣", "n␣␣"], where '␣' represents a white space.

Please note that, as we discuss in Chapter 3, these approaches could be part of the data preparation step resulting in additional attributes that are already prepared and ready to be used by the indexing or even value comparison methods discussed next. However, following Christen's suggestion, we include them in the context of indexing [Christen, 2012b].

2.2 Value comparison

To decide which of the duplicate candidates provided by the indexing phase are actual duplicates, we need to first generate relevant features to take our decisions upon. In this step, for every candidate pair we apply comparison measures on records' attribute values to quantify their similarity (or distance). Depending on the attribute type, different measures can be applied. For instance, a similarity measure for calendar dates should return a higher similarity for dates that differ on just the day compared to dates that differ on the year. However, as discussed at the beginning of this chapter, for simplicity, we focus on alphanumeric values as a broad range of attributes are often encoded in this format. In case more detailed semantic information is available, such as the schema definition of an attribute that declares its specific type, such as calendar dates, specialized measures can be used instead.

Research and industry have established a wide array of similarity measures for alphanumeric attributes. In many cases, string-based measures are originally defined as distance measures, whose values are converted to a similarity by subtracting the normalized distance from 1. We briefly review different categories of similarity (distance) measures.

Edit-based. These string-based measures consider the whole two strings and count the edit operations (e.g., insertions, deletions, replacements, or swaps of characters) that are needed for one input to be converted to the other. Typically used measures of this category are the following:

- Exact Match returns 1.0 if the strings are exactly the same, and 0.0 otherwise.
- Hamming counts the positions at which characters are different [Hamming, 1950].
- Levenshtein counts the minimum number of edit operations to transform one string to the other [Levenshtein, 1966].
- Damerau-Levenshtein expands the Levenshtein measure with the transposition operation for two adjacent characters (a common typo) [Damerau, 1964].
- Jaro-Winkler is an extension of the Jaro distance [Jaro, 1989], which counts the transpositions of characters needed to transform one string to the other, as long as they happen within less than half the string's length. Jaro-Winkler then favors cases where the two input strings have a common prefix [Winkler and Thibaudeau, 1991].
- LongestCommonSubsequence is the longest sequence of characters that is common in the two strings [Chvatal and Sankoff, 1975], in contrast to LongestCommonSubstring [Gusfield, 1997], which demands the characters to be adjacent.

Token-based. These measures tokenize the strings and compare the sets of tokens:

- Jaccard, also known as Jaccard index or Jaccard similarity coefficient [Jaccard, 1901], represents how many tokens the two strings share.
- Jaccard n-gram is a variation of Jaccard where the tokens are produced using a sliding window of size n (i.e., n-grams).

Hybrid. Hybrid measures combine the advantages of both previous categories by using an internal edit-based similarity and applying it to combinations of tokens of the entire input values. The final similarity is then the average similarity of matched pairs.

- Monge-Elkan goes through all tokens of the first input string and finds the token of the second string with the maximum similarity, which can also be re-used for further matches [Monge and Elkan, 1996]. Monge-Elkan returns the mean similarity of the matched token pairs.
- Soft TF-IDF [Cohen et al., 2003] is based on the principles of Term Frequency (TF) and Inverse Document Frequency (IDF), typically used in Information Retrieval. TF-IDF values are calculated for the tokens of the compared alphanumerics, forming a vector for each alphanumeric, and finally compared using the cosine similarity.

The previous categories define the most commonly used string similarity measures. However, as we discussed in Section 2.1, phonetic encodings and n-grams can also be applied to values to facilitate the matching process. The prepared values could be compared with edit-based measures for phonetic encodings, and token or hybrid-based similarity measures for n-grams.

Even with the application of phonetic encodings and n-grams, predefined string similarity measures may not always provide adequate similarity measurements, depending on the dataset's idiosyncrasies. State-of-the-art approaches, though, such as DeepER [Ebraheem et al., 2018] and more recently DeepMatcher [Mudgal et al., 2018], circumvent these limitations by learning a tailored to the dataset similarity measure. These approaches are based on artificial neural networks that convert the provided input values to distributed value representations, also called *embeddings* of the values. These embeddings typically form vectors of normalized numerical values for every record's attribute. Lastly, these vectors are compared between record pairs using an energy function, such as the cosine similarity [Koch et al., 2015], resulting to a normalized similarity.

2.3 Classification

Having calculated similarities, we are ready to use them as features and make a decision of whether a pair of records refers to the same entity (duplicate) or not (non-duplicate). This decision is typically addressed using machine learning (ML) models, such as Support Vector Machines (SVM) [Christen, 2008a; Cortes and Vapnik, 1995], Random Forests (RF) [Ho, 1995], and recently deep-learning neural network models [Ebraheem et al., 2018; Mudgal et al., 2018]. To learn how to distinguish duplicates from non-duplicates, these ML models require some labeled pairs of duplicates and non-duplicates in the form of a gold standard, to train upon and properly configure their parameters. Afterward, they can be applied to new pairs, as they arrive from the duplicate detection process, and mark them as duplicates or not.

Some ML models may delegate the final decision to a human expert, in case a clear decision cannot be made, in a process referred to as *active learning* [Ngonga Ngomo and Lyko, 2012; Sarawagi and Bhamidipaty, 2002]. During active learning, the system decides which instances would be benefited the most from human annotation, typically the ones it is most uncertain of, and sends them to the expert annotators. After the annotation is complete, the system retrains itself, including these new annotations, to decide which are the next instances it is still most uncertain about, resulting in an iterative process. This iteration finishes either when the system is certain about all pairs or there are no available resources left, such as human annotators or time.

After decisions for duplicate and non-duplicate pairs are made, in a production environment, we typically move directly to the next step (Section 2.4), to return the best result to the user. However, during the development or testing phase, we usually need to evaluate our performance, which is normally achieved by calculating classification metrics. True or false positives and true or false negatives are usually calculated to consequently allow the calculation of precision, recall, and finally *F-measure*, which is the harmonic mean of the latter two and is typically reported. Most ML models require a threshold at the end of their process to make their final decision. In cases, though, where the ML model cannot select a stable threshold to perform its final classification, we may prefer to calculate the *area under the precision-recall curve (AUC-PR).* AUC-PR considers all possible thresholds and provides a more stable measurement of the model's effectiveness.

2.4 Partitioning and canonicalization

Since the final product of our pipeline needs to be a duplicate-free database, at this phase we replace the previously marked duplicates with new records to uniquely represent entities. To this end, we incorporate two commonly applied steps: Partitioning and producing canonical representations, in a process similar to data fusion.

Before *partitioning* is applied, first, all the provided edges are converted to an undirected graph. Then a number of different clustering algorithms [Draisbach et al., 2019; Hassanzadeh et al., 2009] can be applied to the directed graph to produce disconnected clusters, where each one of them should represent a different entity. As an example, a commonly applied partitioning strategy, due to its simplicity, is *transitive closure.* Transitive closure is based on the simple principle that if record a is considered duplicate to record b that in turn is considered duplicate to record c, then a duplicate relationship between records *a* and *c* could be automatically induced. It is important to note that possible false positives introduced by the classification phase could link clusters of different entities together, even if the incorrect decision was made only between two records. For this reason, more sophisticated clustering techniques are available, such as correlation clustering [Bansal et al., 2004] and maximum clique clustering [Bron and Kerbosch, 1973].

In case a cluster contains more than one node, a *canonical representation* has to be generated to replace the duplicate records. Different strategies can be applied to select attribute values for a canonical record from the duplicate records. These strategies may involve a simple majority voting, i.e., which attribute value is the most common among the records, or by considering other factors, such as the trustworthiness of individual records and their sources [Culotta et al., 2007]. The process is similar to data fusion [Bleiholder and Naumann, 2009], where the goal is to combine different sources to produce more complete information than the individual sources provide. The result of canonicalization is a clean dataset where no further duplicates are present.

Finally, all the approaches discussed in this chapter can be extended to be executed *incrementally*, where records can be dynamically inserted, deleted, or updated [Benjelloun et al., 2009; Gruenheid et al., 2014], and *distributed* for large-scale data [Chu et al., 2016; Das Sarma et al., 2014]. Various projects and products on duplicate detection, such as JedAI [Papadakis et al., 2018] and Febrl [Christen, 2008b], cover most of the steps discussed in this chapter and can be used for a broad variety of application scenarios. However, as more applications recognize the need for data cleaning and detection of duplicates, the need for projects with sophisticated solutions to handle larger volumes of more complex data will keep increasing.

3

Data Preparation for Duplicate Detection

In the previous chapters, we have introduced the reasons for applying duplicate detection as well as typically used approaches. While trying to apply these approaches to our datasets, which are described in Section 3.4.1, we realized that in many cases values contain inconsistencies that, if resolved before duplicate detection starts, they could benefit the final result. Therefore, improving data quality, by eliminating a number of different errors that may appear in data, prior to executing duplicate detection, can make the latter more successful. Typically, most of these errors are fixed with data preparation methods, such as whitespace removal. However, the particular error of duplicate records, where multiple records refer to the same entity, is usually eliminated independently with specialized techniques. Our work in this chapter is based on [Koumarelas et al., 2020a] and is the first to bring these two areas together by applying data preparation operations under a systematic approach, prior to performing duplicate detection. We show how data preparation steps are usually poorly specified in the literature, although as we show in our evaluation they are very important for later parts of an application, such as classification.

Our process workflow can be summarized as follows: It begins with the user providing as input a sample of the gold standard, the actual dataset, and optionally some constraints to domain-specific data preparations, such as address normalization. The preparation selection operates in two consecutive phases. First, to vastly reduce the search space by removing ineffective data preparations, decisions are made based on the improvement or worsening of pair similarities. Second, using the remaining data preparations an iterative leave-one-out classification process removes preparations one by one and determines the redundant preparations based on the achieved area under the precision-recall curve (AUC-PR). Using this workflow, we manage to improve the results of duplicate detection by up to 19% in AUC-PR. Our contributions can be summarized as:

- A collection of data preparation *examples in the literature.*
- A *systematic approach* to find the most suitable preparations for a given deduplication task.
- *Experiments* to verify our findings from different perspectives.

The rest of the chapter is structured as follows: We proceed with Section 3.1, which motivates the problem of data inconsistencies and briefly introduces our solution. Section 3.2, in which we have collected examples of data preparation throughout many papers on duplicate detection. Section 3.3 covers relevant background for the reader, in the areas of data quality, data preparation, and duplicate detection. Next, Section 3.4 introduces several real-world datasets, which were the inspiration for the selection of common data preparators presented threreafter. We then describe the core process of applying and selecting preparators in Section 3.5. Section 3.6 discusses the results of the process and its impact on duplicate detection. Section 3.7 concludes with our vision on the role of data preparation and how it could be incorporated in a variety of applications.

3.1 Data preparation

Difficult data cleaning tasks are typically treated with specialized algorithms. As an example, duplicated records, which refer to the same entity with variations in their values represent a common error in datasets and are dealt with by duplicate detection methods [Elmagarmid et al., 2007]. Depending on the application task, duplicate detection is referred in the literature under different names [Christen, 2012b], including entity resolution, when it involves matching records within a single relation, or record linkage, when it involves matching records across two or more relations. For this work we focus only on duplicate detection. To improve the performance of duplicate detection, researchers usually place their effort in improving individual parts of the method itself. In contrast, we decided to regard a sometimes overlooked aspect of duplicate detection, namely the preparation of the data before execution starts. We have found this latter issue to be prominent in the literature (refer to Section 3.2 for more details), where most systems are inconsistently preparing their input data and thus it is impossible to perform any accurate comparison of their results.

In this chapter, we look at typical issues with datasets before deduplication and fix them by applying relevant data preparation operators (*data preparators*), in a systematic way. In particular, given a list of n data preparation operations $P = \{P_1, P_2, \ldots, P_n\}$ and a relational schema $\mathrm{R} = \langle A_1, A_2, ..., A_m \rangle$ of m attributes, our objective is to systematically find the set of the most beneficial combinations $< P_i, A_j >$ for duplicate detection among all possible solutions. We envision, and explain further in future work (Section 3.7), that this discussion will result in tools that, depending on the use cases, can select and apply the appropriate data preparators in an automatic way. This could happen either with prior knowledge or in an empirical way, as we do in this chapter, by applying and evaluating the improvement afterwards. The process could be similar to the one shown in Figure 3.1, and depending on the end application will need some extra metadata, such as a labeled set of duplicates and non-duplicates for our use case of duplicate detection. Such functionalities are currently not available in any of the existing data preparation tools (Section 3.3). In particular, the currently available data preparation tools, first, provide only a very limited set of data preparators, which are too generalized and typically not really helpful for specific difficult tasks, such as duplicate detection. Second, they require labeled sets for the data preparation itself, including pairs of unprepared and prepared values, which is an additional time consuming step.

Having tools that can provide us consistently with the same data preparation steps are essential not just for the task of deduplication, but broadly for any kind of experiment,

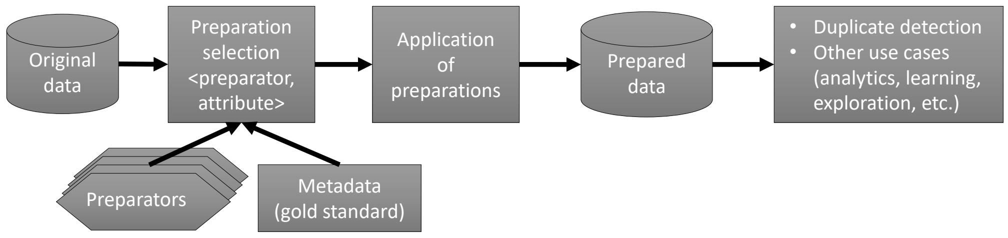

Original data
Preparation selection <preparator, attribute>
Application of preparations
Prepared data
• Duplicate detection
• Other use cases (analytics, learning, exploration, etc.)
Preparators
Metadata (gold standard)

Figure 3.1: Data preparation vision. A pool of preparators is available for many different operations, that test and select the best of them w.r.t. the process at hand. The prepared data are then exported.

Table 3.1: Examples of data preparation.

Preparator	Before	After
Capitalize characters	`NEW york`	`NEW YORK`
Remove special characters	<...> `Berlin` </...>	`Berlin`
Normalize address	`V AVE, NY`	`Fifth Avenue, New York`
Acronymize	`United States`	`US`

to enable *repeatability.* A fundamental aspect of research is the ability to repeat and compare experimental results from different approaches. In this way, when an approach manages to perform better than the previous state-of-the-art, we consider it a scientific achievement. However, in many cases experiments are performed on a different basis than the ones they are comparing themselves against. In particular, basing data cleaning and deduplication experiments on a consistent shared dataset with a ground truth has proven to be difficult. But also in many other areas of computer science, experimenters "massage" the data (usually in good faith) before applying their novel algorithms. We consider this to be a vital issue and have found many cases where this phenomenon takes place (see Section 3.2 for more details). Thus, not only are experimental results incomparable, but some simpler to understand and implement approaches would perform equally or even better had they been given the same input.

A number of different steps are performed in the early stages of any data-driven application, often described as an extract, transform, load (ETL) phase, including data preparation and duplicate detection [Christen, 2012b; Lee et al., 2009; Pyle, 1999]. Typical examples of data preparations are shown in Table 3.1, where character capitalization, special character removal, address normalization, and acronymization, transform the data into a more usable state. For more semantics-based issues, such as duplicate entries for a single entity or erroneous data, more sophisticated methods of *duplicate detection* and *data cleansing* have to be applied. Merely syntactic data preparators are not enough. Our goal is complementary to these duplicate detection methods as we try to improve their effectiveness by a systematic preprocessing phase.

Typically, duplicate detection can be improved by adapting special parts of its process: (1) using better indexing frameworks to retrieve similar candidates that are not completely identical; (2) similarity measures that can better quantify the similarity on each attribute for the previously retrieved candidates; and (3) classifiers that, given the previous similarities, make better decisions. To focus on the effects of data preparation on duplicate detection, we decided to use only basic duplicate detection methods; blocking [Baxter et al., 2003] for indexing (see Section 3.6.1), the hybrid similarity measure Monge-Elkan, using Levenshtein [Naumann and Herschel, 2010] as the token similarity measure (see Section 3.6.1), and a threshold-based classifier [Christen, 2012b] (see Section 3.6.1). We automatically determine those data preparators that convert the data to a more appropriate and clean state, before the actual process of deduplication takes place. In this way, we avoid the development of complicated methods afterwards, bloated with special cases. The overall procedure is not only more accurate but also faster than complicated approaches, which is a prerequisite for many scenarios, such as those of real-time applications. We believe that similarly to deduplication, application of an initial data preparation can improve other processes as well.

Our approach. In a nutshell, we prepare the data for deduplication as follows: First, we sample the set of duplicate and non-duplicate pairs. Second, we decide which combinations of preparators and attributes are valid that we call *preparations*, and apply them. Third, we keep only preparations that affect the similarities in a positive way, which indirectly suggests an improvement on classification, and call them *candidate* preparations. Fourth, the candidate preparations are reduced by applying leave-one-out optimization, leaving us with the *minimized* set of preparations. Fifth, we perform classification on the original and prepared data, and decide whether our process was helpful. Finally, the prepared data are exported so that the next phase of the complete duplicate detection and resolution can begin, which is not in the scope of this chapter. The process is explained in detailed in Section 3.5. The source code and the datasets used in the evaluation are available on our website[1].

3.2 Data preparation in the literature

The general issue of massaging data before evaluation has been identified in the past. Here, we focus on the domain of duplicate detection, also known as record linkage, entity resolution, etc.d [Christen, 2012b]. First, the issue of incomparable evaluation results in publications has been identified by Köpcke et al. [Köpcke et al., 2010], who point to publications that use different sizes of the Cora dataset, which we discuss in Section 3.4.1. Second, Vatsalan et al. recognize the importance of data preparation steps, prior to performing qualitative record linkage, such as enrichment, removal, and transformation of records' values [Vatsalan et al., 2017]. Finally, the inability to reproduce experiments of a given work is a major issue in the data cleansing field and has various causes. One such reason is that many authors do not reference the used datasets, or refer to them with ambiguous names. Another reason is that sometimes private datasets are used, which are not made publicly available with appropriate documentation. Indeed, we also consider such a private dataset, which we cannot disclose, but believe that the findings from it are sufficiently interesting. Finally, it is not always clear how the training and testing datasets are used, and whether parts of the latter are used for the training phase as well.

[1]`https://hpi.de/en/naumann/projects/repeatability/duplicate-detection/data-prep-for-duplicate-det.html`

To motivate the importance of data preparation in duplicate detection, we have selected quotes from papers in the general area of duplicate detection, in which the authors perform some form of preparation steps on the data. In particular, we also included literature where these preparation steps are not clearly specified, which constitutes reproducibility non-existent. While we focus on duplicate detection, massaging data is a common practice for all of data science. The two most common preparation steps done on the datasets are (1) selecting a subset of the dataset and (2) altering the content of values. Our focus is on the latter type and begin with cases of well-explained data preparation steps. The extent of this selection is also intended to show the wide variety of data preparation steps.
[Li et al., 2011]: "*. . . we thus increased the hardness by deriving a data set with only first name and last name initial for each inventor. We call the original data set the full set and the derived one the partial set . . .*"
[Wang et al., 2011]: "*We concatenated attribute values of each citations to a string . . .*"
[Mann et al., 2016]: "*A set is a publication; tokens are character q-grams of the concatenated title and author strings (q = 2, case insensitive) . . .*"
[Kunft et al., 2017]: "*In order to obtain the full feature set, we join the comments and authors CSV input files. To create a vector representation, we apply feature hashing to the authors name and the comments text. We split the name by camel case, white space, and other delimiters and hash the words to a fixed size feature space. For the comments, we split the text into words and hash them as described before.*"
[Sood and Loguinov, 2011]: "*We process the collection by removing pages that have size less than 5 KB, contain no URLs, have an exact duplicate (identified using a standard hash function), or consist of non-English words, all of which shrinks D to a total of 70M pages. We parse each remaining page, removing stop-words and stemming all text outside of HTML tags. We then create feature vectors with weights $t_i(u)$ being the normalize TF-IDF score of each word i on page u . . .*"
[Aizawa and Oyama, 2005]: "*When preparing the data, the attribute values were first normalized using general substitution rules (for example, uppercase to lowercase conversion), and then segmented into tokens, either characters or words, depending on the attribute types and the description languages (for example, names written in Japanese characters were segmented into characters, and the titles were segmented into words) . . .*"
[Churches et al., 2002]: "*This paper describes an implementation of lexicon-based tokenisation with hidden Markov models for name and address standardisation . . .*"
[Churches et al., 2002]: "*Training of HMMs for residential address standardisation was performed by a process of iterative refinement . . .*"

While the mentioned publications do explain their preparation steps, they remain ambiguous in some cases. The following are selected examples of more unclear preparation procedures:
[Weis et al., 2008]: "*Prior to actual duplicate detection, we performed data standardization, which consisted in removing special characters from string attributes.*"
[Chandel et al., 2007]: "*For the company names dataset, we also inject domain specific abbreviation errors, e.g., replacing Inc. with Incorporated and vice versa.*"
[Churches et al., 2002]: "*Name standardisation. To assess the accuracy of name standardisation, a subset of 10,000 records with non-empty name components was selected . . .*"

We split the literature w.r.t. the level of explanation given. Most probably though, it is expected that people's opinions differ on what is a good explanation or not. In any case, we believe that there is some space for improvement, and in particular for a systematic approach that makes the process of deciding data preparation steps easier to explain and justify.

3.3 Related work

There are many and various approaches to resolve *data quality* issues that exist in real-world data. These attempts include, among others, data transformation, in case of different file formats, encodings, attribute names and types, data standardization (or normalization), so that the values conform to the uniformly agreed rules, and data integration, in order to merge different sources of data. *Data preparation* is a general term we use to refer to all these different tasks, as most of them are not always easily distinguishable.

Data quality. Data quality has been broadly studied over the years [Han et al., 2011; Lee et al., 2009; Wang et al., 2001]. Since it represents a core part of business intelligence (BI) processes, there has also been work that explains typically applied steps to improve it from that perspective [Redman, 2001]. Taxonomies of the different quality issues that appear in data have been proposed in the past [Oliveira et al., 2005; Rahm and Do, 2000], with multi-source data integration causing a number of additional data inconsistencies over single-source datasets.

Data preparation. A clever selection and application of *data preparators* on data, can transform the latter to a cleaner state. For this reason data preparators are an essential part of this chapter. Authors of [Kandel et al., 2011] under the term data wrangling consider different actions to *transform* data, *manage* these transformations, and even *scale* them in the cloud. A large part of that work also considers interactive visualization, which they consider to be a vital part of this process.

Yang et al. introduce the extract transform load (ETL) framework Lens and the concept of lenses, which are data processing components [Yang et al., 2015]. An example lens is domain constraint repair, where constraints are applied to prepare the data. For instance, when the "NOT NULL" constraint repair is applied on an attribute, for records that contain null values on that attribute a trained machine learning model is used to infer the missing value among a distribution of possible values.

In cases where user input is possible by providing input/output examples of unprepared/prepared alphanumeric values, the authors of [Gulwani, 2011; Jin et al., 2017] have proposed methods for synthesizing functions, which learn how to perform these transformations and apply them to the remaining data. Both approaches solve the issue of searching through a large space of possible solutions by incorporating heuristics and pruning rules, which form functions comprised of string operations. These string operations, examples of which are *substring*, *concatenate*, and *split*, extract and then transform the input to the desired output format. In the same spirit, with the user providing input/output examples, the work of [He et al., 2018] utilizes over 50K functions that transform an input value into the desired format from GitHub and Stackoverflow along with examples from Wikipedia tables to synthesize its programs. Using a number of representative examples for each data type, such as dates and names, in combination with the extracted functions, it finds those functions that fulfill the criteria of the provided input-output examples, and finally synthesizes them into a program that performs this transformation. For instance, consider the task of date normalization, given a set of input-output examples, where all output examples are dates following the U.S. format, while the input examples follow a variety of date formats, it finds a function that parses the input date and transforms them into the U.S. format.

DataXFormer is similarly based on user-provided input/output examples, but expects as input a relational table with the goal of transforming it into a format denoted by the examples [Abedjan et al., 2015]. This is done by utilizing Web Tables[2] to find the most similar tables and rows, which share a number of attributes with the input table. Thus, for every row of the input table, a number of rows from Web Tables provide attribute values of better quality.

Finally, in case a human-in-the-loop is possible, the work of [Ao and Chirkova, 2019] provides a framework that suggests to domain experts the next most useful records to be manually cleaned, which can improve entity matching results the most and in an active learning principle.

Before any repair can be performed, the errors have to first be identified. The authors of [Abedjan et al., 2016] experiment with a number of different tools, such as OpenRefine[3] and Trifacta[4], on the basis of recognizing specific types of errors on a number of different datasets. Focusing on student enrollment data, Pullen et al. [Pullen et al., 2013] examine typically occurring problems in student identifiers and attributes, such as the existence of special characters or nicknames in first names. Their solution involves manual curation and specialized similarity measures that can exploit this information during data matching.

Duplicate detection. Research on identifying and resolving duplicates has been present for decades, as the problem appears in many applications. This has led to some commonly applied steps [Christen, 2012b; Elmagarmid et al., 2007; Herzog et al., 2007; Naumann and Herschel, 2010] that are used to identify such entries: (1) prepare the data, (2) index and retrieve candidates of possibly matching records, (3) compare these retrieved candidates, (4) classify them as matches or not, and (5) evaluate the entire process. In this chapter, we are focusing on the first part, *data preparation*, suggesting that the latter steps are easier and more successful after data preparation. For the rest of the steps (2-5) we just use standard deduplication techniques (see Section 3.6.1). Most of the current literature focuses on the latter steps with different techniques, having accepted the fact that input data is not in good shape.

First, during the *preparation* step, typically applied techniques [Christen, 2012b; Elmagarmid et al., 2007] include data standardization and enrichment through external data sources, removal of unwanted characters, and segmentation of compound values into separate attributes. Regarding the second step, *indexing*, blocking and sorted neighborhood [Christen, 2012b] represent two of the most successful methods for retrieving candidates. Third, typically for the *comparison* of the retrieved records, similarity measures are used. For instance, for alphanumeric attributes Levenshtein, Jaccard, and Monge-Elkan [Christen, 2012b] are commonly applied as they represent three different

[2]`http://webdatacommons.org/webtables/`

[3]`https://github.com/OpenRefine/OpenRefine`

[4]`https://www.trifacta.com/start-wrangling`

categories of edit-based, token-based, and hybrid similarity measures. More recent works focus on machine-learnable similarity measures, where by using custom features or letting artificial neural networks learn the features themselves, and then further tailor the similarity measures to the data [Christen, 2008a; Mudgal et al., 2018].

Fourth, for *classification*, some threshold-based classifier is commonly used, as their usage is transparent and simple. Given a pair of records, and a list of its calculated attribute' similarities, it linearly combines them and if the sum is above a specified threshold, the comparison represents a match, otherwise not. Searching the parameter space on this classifier is equal to finding the threshold and optionally weights for each attribute's similarity (if not all attributes have the same impact). Other classifiers have also been used, such as Decision Trees [Cochinwala et al., 2001] and Support Vector Machines (SVM) [Christen, 2008a].

Finally, for *evaluation*, a classic calculation of true positives (TP), false positives (FP), and false negatives (FN) is enough to calculate precision, recall, and F-measure. In addition, area under the precision-recall curve (AUC-PR) has been proposed as an alternative to assess detection quality. It has been noted that the F-measure does not fairly balance the errors of the two classes, duplicates and non-duplicates [Hand and Christen, 2018]. In contrast, the AUC-PR, considers all possible thresholds and provides a more holistic evaluation. Using the precision-recall curve it is further possible to calculate the best combination of precision-recall and thus F-measure.

However in reality, any problems that are not fixed during the data preparation phase can and must be addressed in later steps. For instance, a complex similarity measure can address inconsistent formatting of phone numbers. With careful preparation, later steps are more easy to implement and are more effective. Therefore, this chapter focuses on identifying the appropriate preparations that need to be applied, to increase the effectiveness of the following phases, regardless of which methods are used.

3.4 Data quality and data preparation

The focus of this section is on how to identify and repair specific issues in datasets. First, we introduce a number of datasets that we used to find and resolve issues. This helps our discussion: Whenever possible we mention real examples from these datasets. Second, we propose preparators suitable for deduplication, along with some short description and examples.

3.4.1 Datasets

Identifying issues existing in real-world scenarios demands the usage of real-world datasets. For this reason we introduce such datasets with a brief description, and discuss the attributes we used for similarity calculation (see Section 3.6.1) and consequently as features for pair classification. The attributes' selection was based on completeness (percentage of records with a non-null value) and usefulness, which means we omit attributes we consider redundant or not relevant for data preparation. Some further information w.r.t. the gold standard that is needed for the experiments with the generation of non-duplicate (NDPL) pairs, is discussed in Section 3.6.1. Details about all datasets, as well as download information for four of them are available in our website[5]. Lastly, statistics about the number of records, duplicate (DPL) and non-duplicate (NDPL) pairs can be found in Table 3.2.

- CDDB contains entries of audio CDs with descriptive attributes, such as artist, title, tracks, genre, and year. From the eight available attributes we use artist, category, genre, title, tracks, and year. Tracks is a compound element, in which individual track values are concatenated using " | " as the separator.
- Census is based on real-world data, generated by the U.S. Census Bureau, and contains a single attribute with the record's value, called text. This dataset contains two relations `A` and `B`, and thus could be also used for record linkage, i.e., linking the two relations. However, for our purposes we treat it as a typical single-relation dataset and try to find the duplicates instead of the linkage matches. The single attribute text includes information about last_name, first_name, middle_name, zip_code, and address.
- Cora is comprised of bibliographic records about machine learning publications and includes relevant information in seventeen attributes, from which we use authors, journal, pages, title, and year. Moreover, contains large clusters of DPLs, which means many variations of entities.
- Hotels is provided by our industry partner, and contains information about hotels across the globe. We limited our experiments from twenty-seven attributes to hotel_name, street_address, zip_code, city, state, and country. Being a real-world industry dataset, it contains more difficult variations of DPLs, as hotel names and their addresses are provided in many different formats, with a lot of ambiguity. For instance, in hotel names sometimes only the chain is provided, whilst addresses are not always fully specified, with records missing important information, such as the city.
- Movies is the result of merging two different datasets, based on IMDB.org and filmdienst.de, such that real-world duplicates are available. The information provided is limited to the attributes of actors and movie title, which we both use.
- Restaurants is a merged dataset of two other relations, based on Fodor's and Zagat's restaurant guides. From the six available attributes, we use name, addr, city, phone, and type.

CDDB, Census and Hotels gold standards of DPLs were extended with some further pairs, obtained by transitive closure and exact match on the entire record.

3.4.2 Preparators

A preparator is a method that transforms a set of input values into a set of output values that are of higher quality or more useful for the use-case at hand. A preparator's complexity can vary from simple, such as upper-casing all strings, to complex, such as geocoding address fields. The number of input and output attributes can vary and be of any datatype, although in this work we focus on alphanumeric values.

Based on this definition, we present a number of preparators that we consider to be suitable for duplicate detection and have implemented. The presented preparators are meaningful for the sample of datasets we used and do not represent a list of all possible

[5]`https://hpi.de/en/naumann/projects/repeatability/duplicate-detection/data-prep-for-duplicate-det.html`

Table 3.2: Datasets statistics on number of records, duplicate (DPL), and non-duplicate (NDPL) pairs. NDPL pairs are generated according to the blocking process described in Section 3.6.1.

Dataset	Records	# DPL	# NDPL
CDDB	9,763	300	8,944
Census	841	376	1,631
Cora	1,879	64,578	176,285
Hotels	364,965	94,677	368,002
Movies	39,180	14,190	14,069
Restaurants	864	112	11,460

preparators for duplicate detection. Brief descriptions of the implemented preparators along with short examples are shown in Table 3.3. As we denote in Table 3.3, the first six preparators are *lossless*, whereas the last five are *lossy*. Lossy preparators are particularly useful for data matching tasks, such as duplicate detection, but lossy in the sense that important information is removed. We execute them after the others, creating new columns with the transformed values. In contrast, lossless preparators transform a given value to a value that is closer to the normalized version, removing only or mostly unnecessary information and retaining all the essential. Further information along with the description of their slightly different application is provided in Section 3.5. We proceed by providing an explanation of them and accompany them with examples from the datasets presented in the previous section.

Split attribute

While for some applications a coarser schema is appropriate, other applications need more fine-grained values. For instance, to prepare data, a name field might need to be split into first name and last name. The correct solution is to identify such cases, extract and move these data to their respective attributes. Census represents a good example of this category and is used in our experiments as it contains a single attribute (text) with a mixture of information. To allow for a more fine-grained comparison, we decomposed the attribute text into the new attributes last_name, first_name, middle_name, zip_code, and address. This is achieved by using metadata that specify the ranges of substrings required to extract these values. For instance, last_name is contained within the range of characters [0, 15), first_name follows in the range of [15, 28), and so forth. The retrieved values are trimmed, to remove excess whitespace before and after the values. Note that there can be more complicated situations, where the values are not easily separable, the attributes' order is not the same, or the phenomenon is only present on a subset of records. In such cases, more specialized approaches have to be applied, such as the use of regular expressions, to dynamically detect where each attribute's value starts and stops.

Geocode

Geo-coordinates (latitude, longitude) are useful for the deduplication process for data with geographic information. A pair of records whose geographic distance is greater

Table 3.3: Data preparation operators, sorted by their intended order of application. The first six preparators have a lossless effect on the applied values, whereas the last five are lossy. The latter ones are useful particularly in duplicate detection and when applied sacrifice a large portion of the initial values, in a way that is necessary to reveal duplicate values.

Preparator	Description	Example
Split attribute	Extract parts of an attribute, moving them into other attributes	`10117 Berlin, Germany` → ZIP-code: `10117`, City: `Berlin`, Country-name: `Germany`
Normalize address	Convert address to its commonly accepted form, fixing inconsistencies	`fredrich stret 43, berlin, German` → `Friedrichstrasse 43, 10117 Berlin, Germany`
Geocode	Convert address to its geolocation	`Friedrichstrasse 43, 10117 Berlin, Germany` → latitude: 52.5071081, longitude: 13.3896519
Remove special characters	Remove non-alphanumeric characters: [,.:;"^'/\!@#$%&*()_+= |¡¿?{}[]~-]	'/ > `@ Berlin!` <' → `Berlin`
Transliterate: UTF-8 to ASCII	Remove diacritics from words	`München` → `Munchen`
Merge attributes	Merge multiple attributes into a single one	(opposite of split attributes)
Acronymize	Keep the first character of all tokens	`Very Large Data Base` → `VLDB`
Capitalize characters	Convert all characters to upper case	`BERlin` → `BERLIN`
Syllabify	Split word to its syllables	`preparation` → [`prep`, `a`, `ra`, `tion`]
Phonetic encode	Convert value to its pronunciation representation	`Berlin` → `BRLN`
Stem	Reduce word to its base form	`programming` → `program`

than some threshold can be pruned, because they are too far away to represent the same entity. Such a threshold can range from a few meters to some kilometers, depending on the application. The process involves the usage of Geographic Information Systems and the operation of *geocoding*, which, given some address information, returns the geo-coordinates of the location. For our experiments we used the Nominatim system[6].

In the same context, another available preparator is *address normalization*, which given a noisy address retrieves the correct, standardized value. We considered *geocod-*

[6]`https://nominatim.openstreetmap.org`

ing and *address normalization* on Hotels, which includes detailed addresses improperly formatted. By focusing on street addresses, address normalization retrieves values' extended versions, for instance "585 PKWY Dr NE" normalizes to "585 parkway drive northeast" and "2361 W NW Hwy" to "2361 west northwest highway".

Remove special characters

Wrong parsing or conversion can create issues for the following steps of a process. Web scraping and extracting values with optical character recognition (OCR) are situations where irrelevant and redundant characters may be erroneously identified as part of the attribute's value. In such cases removing them leaves us with the pure intended value. For instance in Cora we observe several instances where redundant parentheses, full-stops, and hyphens are left-overs from parsing. For instance, in the attribute Year we meet values, such as "(1987).", pages with values similar to "(pp. 24-30).", and publisher with cases equivalent to "American Association for Arti-ficial Intelligence," where the hyphen probably meant a continuation to the following line.

Transliterate (UTF-8 to ASCII)

Transliteration is the process of converting a text from one format into another. Its purpose is to preserve the characters to the most similar ones in the destination language, which in our experiments is English. For our application, we aim at removing completely any diacritics, which convey a special meaning, but make the deduplication task harder. Examples here include a conversion of München to Munchen and Café to Cafe. Mappings of characters to their non-diacritic versions are provided through a transliteration library[7].

Merge attributes

During the data entry phase, errors due to improper placement of values across the attributes might be introduced. In those cases we merge all the participating attributes into a single one using a whitespace character in between the merged values, effectively treating it as a single long string, with the goal of alleviating a number of errors during duplicate detection. Alternatively, we could instead merge only a subset of the attributes. For instance, in Hotels we could merge the attributes that describe the address into a new single attribute address.

The following descriptions concern *lossy* preparators, as explained in the beginning of this section (Section 3.4.2).

Acronymize

An *acronym* is a type of abbreviation or, in other words, shorter form of a phrase, which is typically performed by keeping the first character of all tokens. For instance, `Data Preparation` gets converted to `DP`. By applying this preparator we aim at identifying those cases where people were not consistent in submitting the full version of the word or the acronymized one. In contrast, *abbreviations* are a superset of acronyms and can

[7]`https://github.com/gcardone/junidecode`

include cases where the short form is selected with a different reasoning, such as first and last characters, phonetic encoding, or something else. An example abbreviation is `Str.` from `Street`. Abbreviations are domain-specific and thus a careful consideration has to be taken when expanding an abbreviated value into its full form. The preparator of abbreviations would be useful in cases where some people might know a place with its short form or consider it too large for a field. Given the multitude domains of our datasets' attributes, abbreviations are currently not supported as a preparator.

Capitalize characters

Different *capitalization styles*[8] can be used, mainly the *sentence case*, *title case*, and *full capitalization* (all-caps), to denote a specific meaning. To overcome such inconsistencies in capitalization, which can be also the result of a data entry error, we choose to follow the full capitalization style, by upper-casing and lose information whilst increasing the similarity. For instance, in research titles of Cora one might come across versions, such as "Small disjuncts in action: Learning to diagnose errors in the telephone network local loop," and "Small Disjuncts in Action: Learning to Diagnose Errors in the Local Loop of the Telephone Network". By upper-casing both values the differences in capitalization would be eliminated and thus the similarity between these two values would be increased.

Syllabify

Syllabification is a type of word segmentation, where a word is split into its syllables, usually for visual reasons, such as in fields that depend on resizeable user interfaces. Such interfaces introduce a continuation of words in the following lines, particularly useful in larger values, such as the address, which in some countries can be truly long[9]. As an example, `Friedrichstrasse 43, 10117 Berlin, Germany` is syllabified to `Fried rich stras se 43, 10117, Ber lin, Ger many`. For our experiments we used English syllabification patterns. Alternatively, if support of more languages is desired, a combination of language detection with specialized patterns for each language is possible. Multilingual approaches are beyond the scope of this work.

Phonetic encode

Phonetic encodings aim at converting words into a representation that mimics their pronunciation. Different encodings have been implemented throughout the years, including Soundex and Metaphone [Christen, 2012b], which we use in our experiments, and represent a successful step towards deduplication. For instance, in CDDB, using Metaphone for "Metallica" we obtain "MTLK", which is also the same for "meettaaliica". Strictly speaking, such lossy preparations are very specific to the problem of duplicate detection, where they perform well, as they are able to abstract from the original values.

Stem

Stemming helps overcome inconsistencies in word endings. These inconsistencies can be the result of issues, such as different word representations, plural instead of a singular

[8]`https://en.wikipedia.org/wiki/Capitalization#Names_of_capitalization_styles`

[9]`https://en.wikipedia.org/wiki/List_of_long_place_names`

noun, wrong verb tense, etc. The stemming model attempts to retain only the base of a word, without incorporating any contextual information. Examples here include "discussing" and "discussion", which are stemmed to "discuss". For our experiments we used the English *Porter stemmer*[10]. Similarly to SYLLABIFY, multilingual support is possible, but is beyond the scope of this work.

In this section we described preparators that we considered to be the most impactful and that is why we implemented them for our system and experiments (see Sections 3.5 and 3.6). However, there are many more possibilities for other preparators, even in duplicate detection, which we did not implement, such as:

- Normalize person names, phone numbers, and dates: parsing and formatting values according to a specified format. For instance phone numbers could be represented in E.164[11], which is a worldwide standard that can universally represent phone numbers.
- Encode characters to UTF-8: Converting different character encodings to UTF-8 would allow for a more meaningful value storage and comparison.
- Generate character n-grams: having a list of n-grams from a given alphanumeric value could assist a fuzzy indexing, where similar, but not necessarily the same, values could be retrieved.
- Standardize NULL value representation: replace values that represent lack of a value, such as "n/a", ".", "-", and "not provided", with the empty string "" or some other predefined value.
- Remove stop words: such as "a", "the", "to", as these increase the similarity unnecessarily in non-duplicates.
- Convert categorical or ordinal to numerical: converting values to numerical using a predefined mapping or ordering. For instance, genders could be mapped to $\{0, 1\}$, whereas days of week could be represented with a mapping to $[0, 6]$.
- Normalize numerical values to [0, 1]: a normalization using the minimum and maximum values.
- Apply transformation rules: replacing or removing specific words. For instance, in the case of synonyms replacing values with one representative could help improve similarities among duplicates. Alternatively, values that have been erroneously entered can be completely removed.

Having all these preparators available, next we discuss a process to decide of these are the most useful for deduplicating a given dataset.

3.5 Deduplication preparation

In this section we describe our process for trying a number of available preparators on a dataset's attributes, and recommending the pairs of preparators and attributes that lead to improved results. We discover the most important preparations using a heuristic 2-step pruning process based on a (small) gold standard. A heuristic approach is required, because examining all possible preparation combinations is infeasible. Based on these assumptions our process works as follows: First, we apply a fast process based only on

[10] `https://tartarus.org/martin/PorterStemmer/index.html`
[11] `https://en.wikipedia.org/wiki/E.164`

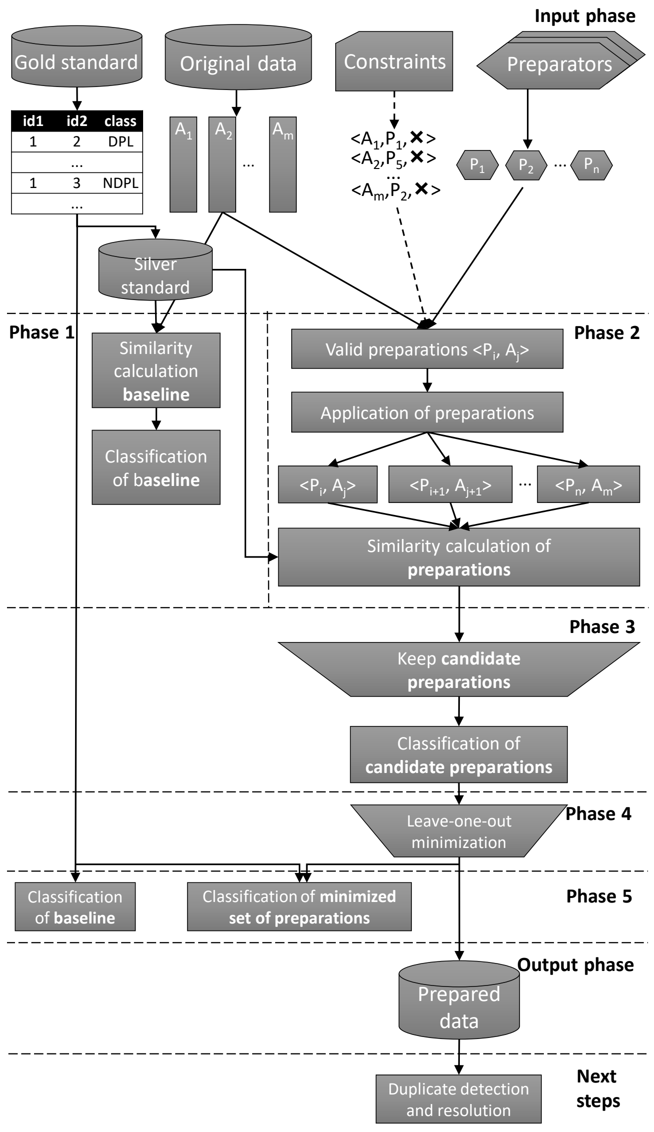

Input phase
Gold standard
Original data
Constraints
Preparators
id1 id2 class
1 2 DPL
1 3 NDPL
A1 A2 Am
<A1,P1,✕>
<A2,P5,✕>
<Am,P2,✕>
P1 P2 Pn
Silver standard
Phase 1
Phase 2
Similarity calculation baseline
Classification of baseline
Valid preparations <Pi, Aj>
Application of preparations
<Pi, Aj> <Pi+1, Aj+1> <Pn, Am>
Similarity calculation of preparations
Phase 3
Keep candidate preparations
Classification of candidate preparations
Leave-one-out minimization
Phase 4
Classification of baseline
Classification of minimized set of preparations
Phase 5
Output phase
Prepared data
Duplicate detection and resolution
Next steps

Figure 3.2: Process of identifying useful preparations, for duplicate detection.

similarities, which reduces the search space of preparator combinations drastically. Then, we follow with a careful process that bases its decisions on whether to keep preparations using duplicate detection classification on the prepared values, which is our final goal. An exhaustive attempt, trying all preparations could be possible, but is not feasible in real scenarios with large datasets. Our process is shown in Figure 3.2 and includes the following phases:

Input phase: The user provides data, optionally some constraints to specify which preparators are not valid for certain attributes, and a set of labeled pairs (gold standard), whereas the system provides its available preparators. In case a constraint was not provided for a preparation that is meaningless, such as ADDRESS NORMALIZE on a first_name the system would produce unhelpful results. By selecting a subset of 10% from the gold standard, we create a silver standard. These pairs can be chosen by following different strategies, such as to select them randomly or selecting those that are the most difficult to classify. We select the latter strategy and explain our approach further in Section 3.6.1. Selecting such a silver standard helps us represent a realistic scenario, as in practice before a larger and complete set of pairs (gold standard) can be collected, users typically start experimenting with smaller sets (silver standard) that include difficult cases. In domains where labeling is more expensive, users can further reduce this sampling ratio, assuming they still make sure to include challenging pairs. Moreover, since almost all of the process' steps are performed on sampled data, the execution time stays within reasonable limits.

Phase 1 – Baseline: In this phase we are interested in performing classification on the original data, so that we can later judge if the application of preparations helped or not. This classification is performed in two steps. First, we calculate the pair similarities for the silver standard, based on the data. Second, we perform classification using the pair similarities as features, evaluate the results, and determine the *AUC-PR*.

Phase 2 – Preparation application: Given the set of preparators $P = \{P_1, P_2, \ldots, P_n\}$ and attributes A_i of a relational schema $\mathrm{R} = \langle A_1, A_2, ..., A_m \rangle$ this step produces all valid combinations $< P_i, A_j >$, called *preparations*. We call a combination valid if the data type of the attribute is compatible to the data type that the preparator is expecting. For instance, it is pointless to apply PHONETIC ENCODE on a phone number. This information is provided as metadata at the input phase. Finally, for each preparation we calculate the pair similarities.

Phase 3 – Candidate preparations: In this step we aim to filter preparations w.r.t. their effect on similarities. This filtering approach is intuitive since classifiers use the similarities as their features. Given the N duplicate pairs and M non-duplicate pairs of the silver standard, we keep only those preparations that fulfill the following condition:

$$\frac{\sum_{i=1}^{N} \delta sim(\mathrm{DPL[i]})}{\mathrm{N}} - \frac{\mathrm{M}}{\mathrm{N}} \cdot \frac{\sum_{i=1}^{M} \delta sim(\mathrm{NDPL[i]})}{\mathrm{M}} > 0$$

$$\iff \sum_{i=1}^{N} \delta sim(\mathrm{DPL[i]}) - \sum_{i=1}^{M} \delta sim(\mathrm{NDPL[i]}) > 0$$

where δsim expresses the similarity delta of a pair before and after preparation ($sim_{after} - sim_{before}$) and becomes negative when the similarity drops. The weighting factor of $\frac{\mathrm{M}}{\mathrm{N}}$ in the NDPL pairs complies with the ratio of our class pairs, since more NDPL pairs exist than DPL, for reasons explained in Section 3.6.1. The mean similarity of DPL pairs has to increase, and in contrast, the mean similarity of NDPL pairs of records has to decrease or at least increase less than the mean similarity of DPL pairs. The second formula is a simplification of the first, where merely the summation of similarity deltas between DPL and NDPL pairs is enough to decide whether a preparation is useful.

After this filtering, we are left with the *candidate* preparations, with which we proceed to perform another classification.

This phase evaluates and decides which individual preparations are interesting to be considered as candidates. The order of application is an additional dimension that affects the results of the following two phases (Phases 4 and 5). We do not address this problem here but leave it for future work. Instead we proceed by applying all preparations using a *predefined* ordering, namely, as they appear in Table 3.3. As an extension of the order issue, some preparators are helpful for our domain specifically, but lossy in the sense that important information is removed. For instance, if the preparators NORMALIZE ADDRESS, TRANSLITERATE, and ACRONYMIZE are selected for the attribute city, we first prepare city using NORMALIZE ADDRESS and TRANSLITERATE, creating a respective column. ACRONYMIZE, which is a lossy preparator, is applied on top of the former column, creating an additional column with only the initials, having two columns in total. By not considering lossy preparators in combination we risk missing some corner cases where a combination of two such preparators would help reveal duplicates. In the majority of cases, however, applying one lossy preparator after another changes the value so drastically that it is likely useless for our task, introducing more matching mistakes.

Phase 4 – Leave one out optimization: Some preparations might have no effect after the application of other preparations. For instance, by considering the examples of Table 3.3, applying REMOVE SPECIAL CHARACTERS on '`/ > @ Berlin!  <`' returns the value `Berlin`. We would observe the same effect on the value if we had instead applied NORMALIZE ADDRESS. Thus applying both preparations is redundant. On the same principle, applying some preparations in combination can also worsen the result. Understanding the effect of a preparator on a pair of records is more complicated, as it is a combination of the pair's class (duplicate or non-duplicate), and its initial and final similarities. Thus, to make our final decisions and remove such preparations, we perform an optimization process: we remove preparations to avoid redundant preparations (similarity stays the same) or preparations that are in fact harmful (similarity increases after removing them). We iteratively leave out individual preparations and perform classification using the rest. For instance, given P candidate preparations, there will be P executions, where in each execution one preparation is removed, whilst the remaining $P - 1$ preparations are applied. This process could be also characterized as "backwards elimination with replacement". Removals that keep the AUC-PR constant or even improve it are made permanent, which leaves us with the *minimized* set of preparations, as no further preparations are going to be removed with our process.

Phase 5 – Final classification and comparison: After the pruning of the previous phase, where we decided the minimized set of preparations, we re-perform classification, but this time on the full set of pairs (gold standard), to acquire our final metrics. Finally, we compare these metrics to the baseline (also on gold standard).

Output phase: By comparing the AUC-PR results from our process and the baseline, we can determine whether our process was helpful. If this is the case, then we proceed by using the prepared data from Phase 4. Otherwise, we can ignore our whole process and proceed with the unchanged data.

Next steps: This is the main process that we performed the data preparation for. In our case, this should be equivalent to performing duplicate detection with a larger set of goals. For instance, using a larger set of pairs than the one provided in our process or resolving the duplications found by merging the duplicated records into a single one per entity. Resolving all issues around duplicate detection is not in the scope of this chapter.

3.6 Evaluation

We now examine the effect that different *preparators* have across *datasets* from two different perspectives, first by focusing on the preparators and second on the datasets.

Examples of these preparations across datasets are shown in Table 3.5 on page 38. We first explain some preliminaries in Section 3.6.1 to set the ground for the experiments described in Section 3.6.2. The findings are then summarized in Section 3.7.

3.6.1 Preliminaries

In the following sections we discuss critical decisions. Briefly, we begin by discussing the need for careful non-duplicate pair generation. Subsequently discuss the choices we made regarding the choice of non-duplicate pairs during the silver standard selection, the similarity measures, and classifiers.

Gold standard and non-duplicates generation

The gold standards that are available for many of the datasets in Section 3.4.1 each contain a list of record pairs that uniquely identify *duplicate record pairs*. In case this set is not transitively closed, we additionally create all transitive pairs and call this extended set *DPL*. To train and test a classifier, we also require a number of negative examples, i.e., non-duplicate pairs. When creating such non-duplicate pairs we should make sure that records are not paired up randomly, since such pairs are expected to be very different and therefore trivial to classify. More useful for training and for testing, however, is to expose the model to pairs that are harder to classify. In practice, to speed up duplicate detection, blocking methods are typically used to divide datasets into disjoint subsets, called blocks, according to a predefined partitioning key [Christen, 2012b; Elmagarmid et al., 2007]. To avoid false negatives, usually multiple partition keys are defined. It is important to select the partition key with great care, as it controls not only the size and number of blocks created, but also how similar the individual data records within each block are.

We perform a simple blocking by using a subset of the currently processed dataset attributes as partitioning keys. Record linkage datasets are treated as deduplication datasets, i.e., pairs of the same source relation can be positioned in the same block. In case an attribute is multi-valued, we first divide it into its individual values and use those for blocking. Very unique attributes, which would lead to many single record blocks, are split into word or character n-grams of size 2 to 10 characters, depending on the length of the attribute, and then used as partition keys. The complete blocking scheme is presented in Table 3.4 and is discussed next. As a result, we obtain several blocks, within which we create the cross product of all contained data records and thus form data record pairs, which are then combined into a common dataset. After removing known duplicate pairs in the resulting dataset, we refer to it as *NDPL*. The NDPL dataset is, therefore, the set of all record pairs that can be formed within all blocks and does not include duplicate pairs.

In Table 3.4 we present the *blocking scheme* used to generate the non-duplicate pairs. Across datasets, we selected attributes that have high uniqueness, which tends to characterize better the entities. This results in smaller blocks that usually contain only the really similar candidates. The following strategies are used: (1) “Complete value” that uses the whole value of the record’s attribute as a blocking key, (2) word n-grams that considers n consecutive words tokenized by white space of every record’s attribute value, and finally (3) character n-grams that, similarly to word n-grams, considers n consecutive characters from every record’s attribute value. In cases where the lengths of alphanumeric values are less than or equal to n the complete value is returned. Finally, the strategy and the size of n-grams (n) are selected taking into account the mean attribute's length, i.e., we prefer character n-grams for attributes with shorter lengths, whereas word n-grams for attributes with longer lengths.

Table 3.4: Blocking scheme, used to generate the non-duplicates (NDPL).

dataset	**attribute**	**blocking strategy**
cddb	artist	complete value
	title	word n-grams: n=6
	tracks	word n-grams: n=6
census	text	word n-grams: n=2
cora	authors	character n-grams: n=6
	title	character n-grams: n=6
hotels	hotel_name	character n-grams: n=10
	street_address1	character n-grams: n=10
movies	title	word n-grams: n=3
restaurants	name	character n-grams: n=6
	address	character n-grams: n=6
	phone	character n-grams: n=8

Selecting difficult pairs for silver standard

To better understand the impact of our preparations, instead of forming the silver standard of Section 3.5 by randomly sampling 10% from the gold standard, we decided to follow a different approach. First, having the sets of duplicate and non-duplicate pairs, we calculate similarities for the attributes discussed in the datasets section (Section 3.4.1), using the similarity measures discussed next. Afterward, we calculate the mean similarity for every pair, sort pairs based on this similarity, and select, similarly to the silver standard ratio, the 10% duplicates with the lowest similarity and the 10% non-duplicates with the highest similarity. These pairs should be closer to the boundaries of classifiers, thus harder to classify and more prone to classification mistakes. Although this approach does not guarantee that these are the hardest pairs for classification and thus provide us with the best possible insights, it still provides us pairs that are more challenging than randomly chosen ones.

Similarity measures

Choosing similarity measures can be a tricky task. In the ideal scenario, if we can identify the type of an attribute then we should select a similarity measure tailored for it. For instance, for an attribute that contains information about calendar dates, one could select a similarity that considers a difference in year a lot more important than a difference in days. To keep our process simple, however, we use alphanumeric similarity measures for all attribute types, with the exception of a geolocation distance. For the latter we use the Haversine distance [Inman, 1849] and instead of using a threshold to prune pairs with a larger distance we convert it into a similarity by using the formula similarity = $\frac{1}{\text{distance}^2+1}$, where distance is in meters. Using this formula smaller distances result in higher similarities. Therefore, pairs with large distances have a smaller similarity and less chances to be considered as duplicates, assuming always a successful prior application of GEOCODE.

In alphanumeric similarity measures, there are three main categories: edit-based, token-based, and hybrid similarity. Examples of these categories are: Levenshtein, Jaccard, and Monge-Elkan respectively [Naumann and Herschel, 2010]. We selected Monge-Elkan for our experiments, as it is a hybrid similarity that can provide a meaningful measurement in a variety of attribute formats. This measurement is calculated as follows: Monge-Elkan goes through all tokens of the first input string and finds the token of the second string with the maximum similarity, using Levenshtein in our case, which can also be re-used for further matches [Monge and Elkan, 1996]. Finally, it returns the mean similarity of the matched token pairs. Other categories of string similarity measures include phonetic similarity measures, such as SOUNDEX [Christen, 2012b], and feature-based ones, such as MinHashing and Locality Sensitive Hashing (LSH) [Leskovec et al., 2014], which offer an approximation of the Jaccard similarity coefficient [Christen, 2012b].

Classifier

We use a linear or threshold-based classifier (THR) [Hastie et al., 2009], because it is a simple and effective model, that is frequently used in duplicate detection [Benjelloun et al., 2009; Christen, 2008b]. The features we provide to the classifier are the similarities calculated for the attributes described in Section 3.4.1, using the similarity measures of Section 3.6.1. To keep our process simple, we choose the weights of the features to be equal. THR calculates a linear combination of the features and if the sum is above a threshold the classified label is 1 (match), otherwise 0 (no match).

Instead of selecting a single threshold to calculate the F-measure, which as discussed in Section 3.3 does not fairly balance the errors between duplicates and non-duplicates, we use area under precision-recall curve (AUC-PR) [Hand and Christen, 2018]. AUC-PR considers all possible thresholds, relevant to the evaluated pair similarities. Finally, since we do not choose any thresholds, and our THR model does not require training to tune any parameters anymore, we use the full set of DPL and NDPL pairs for testing.

3.6.2 Effect on similarity

In this section we examine how preparators affect the similarity of values in various attributes as described in Phase 2 of Section 3.5. Before we analyze the collective results across datasets, we motivate the discussion by presenting a few interesting cases of preparations for exemplary values across five of the datasets. In Table 3.5 we show selected value pairs from different datasets and their similarity for a true duplicate and for a true non-duplicate, each before and after preparation. In the case of hotels, ADDRESS NORMALIZE is using the address attributes of each record to decide what is the correct normalized address. Using the normalized address, attributes such as the city, can be corrected or filled in, resulting in an increased similarity. Most examples are successful cases, in terms of similarity being increased in duplicate pairs (DPL), whilst decreasing in non-duplicate pairs (NDPL). Exceptions to this are the examples of the NORMALIZE ADDRESS and ACRONYMIZE preparators, where DPL pairs' similarities decrease and NDPL pairs' similarities increase. First, regarding the NORMALIZE ADDRESS, this effect does not necessarily lead to worse overall results. The reason for this is that we consider the preparation beneficial if it is easier to separate the pairs afterwards, even if the NDPL pairs' similarity increases. Thus, the relative similarity of all DPL and NDPL pairs is more important, such as in the case of Hotels, where the DPL pair's similarity increases more than the NDPL pair's. The importance of relative similarity is considered in our candidate selection formula in Phase 3 of Section 3.5. Second, the examples of ACRONYMIZE represent cases that lead to worse results, for both DPL and NDPL pairs, and for this reason are not selected from our process.

Next, we discuss the aggregated results over *preparators*, whereas afterward we do the same from the *datasets'* viewpoint. By combining these perspectives we can obtain a better picture of the overall preparations' effect. We confirm their success in the final duplicate classification experiments, discussed in Section 3.6.3.

Effect of preparators

We define the positive effect ('+') in DPL pairs to take place when they become more similar, whereas for NDPL pairs when they become less similar. Vice versa, a negative effect ('−') exists when DPL pairs become less similar and NDPL pairs more similar. Following these two definitions of positive and negative effects, in Figure 3.3 we show the percentage of all pairs whose similarities are affected in a positive or negative way, across datasets.

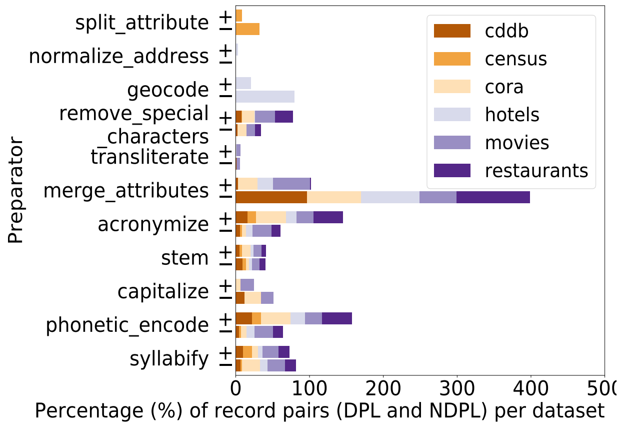


Figure 3.3: Percentage (%) of record pairs (DPL and NDPL) whose similarities improved (+) or worsened (−), across *preparators* for all datasets.

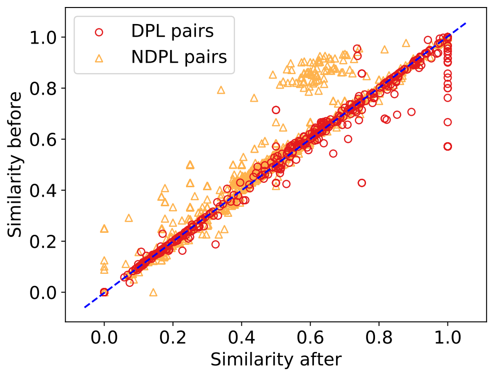


Figure 3.4: Similarity difference (before and after) of preparator REMOVE SPECIAL CHARACTERS.

Overall, most preparators either do not improve the similarities across all datasets or they do, but only marginally. More specifically, REMOVE SPECIAL CHARACTERS, ACRONYMIZE, and PHONETIC ENCODE are exceptions to the previous observation and clearly improve overall more records across datasets. The first is a general preparator, which can be applied to multiple use cases, but the last two are more tailored towards deduplication. On the other hand, there are preparators, such as MERGE ATTRIBUTES and CAPITALIZE, which in fact have worsened more pairs. However and as mentioned, this by itself is not enough to judge whether a preparator is indeed harmful or not.

Table 3.5: Preparation examples, across datasets. Before and After represent the values before and after the application of the preparation, respectively (with #1 and #2 representing the two records' values). Examining the similarities ("Sim"), using the Monge-Elkan similarity measure (explained in Section 3.6.1), we show selected successfully and **unsuccessfully** (marked with **bold** characters) prepared pairs; duplicates (DPL) and non-duplicates (NDPL).

Dataset	Preparator	Attribute	Class	Before #1	Before #2	Sim Before	After #1	After #2	Sim After
cddb	phonetic encode	artist	DPL	`VĂ¸mmĂ¸l Spellmannslag`	`V mm l Spellemannslag`	0.46	`FMLSPL`	`FMLSPL`	1.00
			NDPL	`The Beta Band`	`Raoul And The Big Time`	0.53	`OBTBNT`	`RLNTOB`	0.00
cora	remove special characters	authors	DPL	`Aha, D., Kibler, D., & Albert, M`	`D. Aha, D. Kibler, and M. Albert.`	0.67	`Aha D Kibler D Albert M`	`D Aha D Kibler and M Albert`	0.86
			NDPL	`Fahlman, S. E.,`	`Clouse, J., & Utgoff, P.`	0.35	`Fahlman S E`	`Clouse J Utgoff P`	0.00
restaurants	syllabify	address	DPL	`804 northpoint`	`804 north point st.`	0.63	`804 north point`	`804 north point st.`	0.89
			NDPL	`5937 geary blvd.`	`14928 ventura blvd.`	0.50	`5937 geary blvd.`	`14928 ven tu ra blvd.`	0.39
movies	remove special characters	title	DPL	`L'Intrus`	`Intrus, L'`	0.52	`LIntrus`	`Intrus L`	0.65
			NDPL	`Oh... Rosalinda!!`	`H.M.S. Defiant`	0.26	`Oh Rosalinda`	`HMS Defiant`	0.06
hotels [1]	normalize address	city	DPL	`CHICAGO`		0.00	`CHICAGO`	`CHICAGO`	1.00
			NDPL	`WILMINGTON`		0.00	`WILMINGTON`	`HERNDON`	**0.30**
movies	acronymize	title	DPL	`The Clinton Chronicles`	`Clinton Chronicles, The`	0.97	`TCC`	`CCT`	**0.33**
			NDPL	`Daredevils of the West`	`Desperadoes of the West`	0.82	`DOTW`	`DOTW`	**1.00**
hotels	acronymize	hotel_name	DPL	`HOTEL ADONIS`	`ADONIS HOTEL STRASBOURG`	0.86	`HA`	`AHS`	**0.33**
			NDPL	`INFUSION COFFEE SHOP`	`IP CASINO RESORT SPA`	0.23	`ICS`	`ICRS`	**0.75**

[1] ADDRESS NORMALIZE is using the rest address attributes of each record to decide what is the correct normalized address and thus city.

To validate the previous hypothesis and to obtain a better insight concerning the contribution of a preparator, i.e., whether it is indeed helpful or not, we show another type of visualization. In Figure 3.4 we consider the REMOVE SPECIAL CHARACTERS preparator, which has more of a positive effect, for visual reasons randomly downsampled to 5,000 pairs (DPL and NDPL) across all datasets and attributes. We can observe the effect on similarity, before and after preparation: pairs below the main diagonal have had their similarity increased, which we prefer for DPL pairs, and decreased above the axis, ideally mostly for NDPL pairs. In this preparator we see that a large proportion of NDPL pairs is above the main diagonal and, similarly, a noticeable subset of DPL pairs is below the main diagonal, which both align with the positive results in Figure 3.3.

Examining similar figures for SYLLABIFY and CAPITALIZE (not shown), which have more negative effects in similarities, we observe that although many NDPL pairs' similarity increases slightly, the simultaneous larger increase of DPL pairs' similarity, makes it easier to separate the two classes afterwards. This effect is examined in more detail in the next section for the Cora dataset. The situation is similar for other helpful preparators, such as NORMALIZE ADDRESS, with some small negative effect, but a predominant positive effect.

In summary, examining the preparators with an overall higher positive effect reveals an easier separation of the two classes, DPL and NDPL. However, this is more of an indicator and the actual positive effect of a preparator is decided in practice when it is combined with a particular dataset's attribute, as discussed in Phase 3 of Section 3.5.

Effect on datasets

We now examine the effect that all preparations together had on each dataset w.r.t. the percentage of affected pairs. In Figure 3.5 we see that in fact, with the exception of the Hotels dataset, most datasets experienced an overall positive effect in similarities in the majority of the pairs.

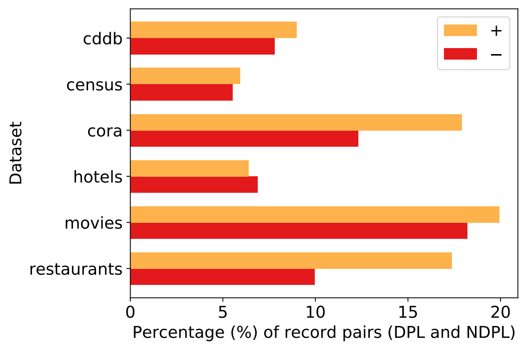


Figure 3.5: Percentage (%) of record pairs (DPL and NDPL) whose similarities improved (+) or worsened (−), across *datasets*.

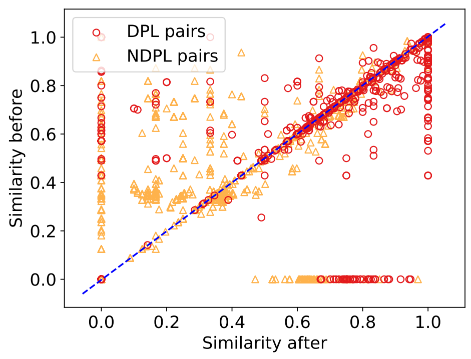


Figure 3.6: Similarity difference (before and after) of dataset Cora.

Cora is a case with a large improvement in its pair similarities and to understand the reason behind this, we show Figure 3.6 to review the actual changes on the similarity of pairs. One interesting remark is that most DPL pairs have a high similarity before and after preparation, although in most cases they still managed to improve by a large degree. At the same time, there is a clear difference between most DPL and NDPL pairs, which hints good classification results. The unusual vertically aligned pairs are caused mostly by the PHONETIC ENCODE and CAPITALIZE preparators, whereas the bottom line of pairs that had 0.0 similarity before and a higher similarity after preparation are caused by preparators that populate attributes with values that were initially empty, such as MERGE ATTRIBUTES or GEOCODE and NORMALIZE ADDRESS in Hotels.

3.6.3 Classification effect

Building on the previous section, we now examine the final classification results, as discussed in Phase 5 of Section 3.5. In Figure 3.7 we see the AUC-PR scores of all datasets on the full set of DPL and NDPL pairs (gold standard). The "No preparation" and "All preparations" bars describe two baselines of (i) using the original values and (ii) applying all available valid preparations respectively. "Minimized preparations" considers the minimized set of preparations produced by our pipeline. Comparing the set of "minimized" preparations to "all" preparations serves as a good indicator to show how important is to select high quality preparations among the set of possible preparations. In practice, only a small fraction of all the available preparations were selected by our process (Table 3.6), using up to six out of the eleven available preparators in total. Additionally, we present the best achievable F-measure over the precision-recall curve displayed in Figure 3.8. This represents the best-case scenario that a user would select to apply this process in practice, where the selected threshold of the precision-recall curve maximizes the F-measure.

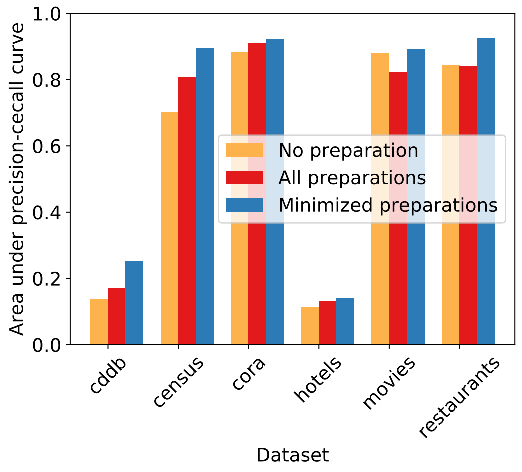


Figure 3.7: Classification results according to *AUC-PR* of *original* values (left bars), prepared with *all valid preparations* (middle bars), and prepared with *minimized* set of preparations (right bars).

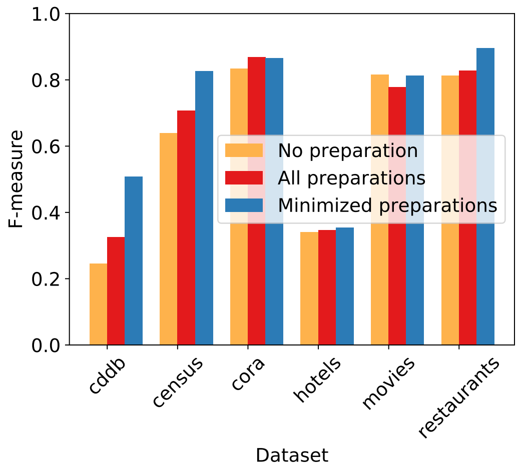


Figure 3.8: Classification results according to *best achievable F-measure* of *original* values (left bars), prepared with *all valid preparations* (middle bars), and prepared with *minimized* set of preparations (right bars).

Using "All preparations" does in fact improve the quality of duplicate detection in many of the datasets without the need to go through our process pipeline. An explanation for this is that a certain subset of preparators helps to improve duplicate detection in the majority of the attributes they are applied to. As an example, ACRONYMIZE and PHONETIC ENCODE are selected for many attributes by the "minimized" set of preparations, as shown in Table 3.6, with which we can derive their importance. However, to avoid a negative effect on the results, as is the case with AUC-PR in Movies and Restaurants, a careful selection of preparations needs to be done. The advantage of carefully selecting the preparations is verified by "Minimized preparations" bars: *all* datasets had an improvement in their classification results in AUC-PR, which is consistent with the similarity improvements (Figures 3.3 to 3.6). This is consistent as well with the leave-one-out optimization; as removing certain preparations in Phase 4 did help improve AUC-PR over the candidate preparations of Phase 3. Comparatively, with the exception of Cora in F-measure, using the minimized set of preparations achieved the best results, consistently outperforming both baselines.

In terms of improvement, CDDB, Census, and Restaurants had the largest improvements. Among them, Census achieves the largest improvement (19%), which can be explained by the discussion of Figure 3.8 in Section 3.6.3. In contrast, the other three datasets did not manage to achieve such levels of improvement: value inconsistencies in Cora and Hotels are more difficult and thus not repaired by our preparations to such a degree that would make the separation of DPL and NDPL pairs easier. Movies, on the other hand, does not contain enough information in the two provided attributes that could benefit substantially from preparations. In contrast, Census consists of five different types of information (Section 3.4.1) that are in a more normalized format, which is more adequate to distinguish two different entities. In most of the datasets AUC-PR is increased due to improvements in precision, except for Cora that bases its better results on recall. We observe that increases in AUC-PR are also reflected in the best achievable F-measures, with the exception of Movies. Although there is a positive effect in both similarities and AUC-PR, this does not result in a better F-measure.

Lastly, for completeness in Table 3.6 we present the minimized set of preparations, as they are produced by the leave-one-out minimization of Phase 4 in Section 3.5. Attributes and preparators not presented in the table are not part of the minimized set of preparations. The preparators PHONETIC ENCODE and ACRONYMIZE have been selected for many attributes. As explained in Section 3.5 they are both "lossy" preparators, in the sense that important information is lost. On the other hand, they do help improve duplicate detection results, which is why they are selected in many cases.

3.7 Conclusion

This chapter set out to analyze the impact of various data preparation activities on the success of duplicate detection. While some kind of preparation is usually vaguely reported on, it is consequential precisely which transformations were in fact executed. Our systematic analysis reveals the effects of various, selected data preparations and highlights the need for a more formal specification of data preparation steps and the benefit of a data preparation framework, which would free data experts to repeatedly build new scripts for old or new datasets. Ideally, these preparators should be offered by a library, such as those we discuss in Section 3.3. However, in our case the offering was not adequate and we implemented the preparators ourselves. Overall, such a library

Table 3.6: Minimized preparations of experiments in Section 3.6. Only preparations that are produced by our process are presented, i.e., for the attributes not presented no preparation is selected.

dataset	**attribute**	**split_attribute**	**normalize_address**	**remove_special_ characters**	**transliterate**	**acronymize**	**capitalize**	**phonetic_encode**	**syllabify**	**stem**
cddb	artist					X	X	X		
	title					X		X		
	tracks			X		X		X	X	
census	first_name	X								
	text							X		
cora	address			X		X	X	X	X	
	authors					X		X		
	date			X		X				
	title					X		X		
hotels	city		X							
	hotel_name							X		
	street_address1		X							
movies	actors			X						
	title			X	X		X		X	
restaurants	address			X		X		X	X	
	name					X		X		X

should help developers prepare their data faster for their use cases, without the need for deeper knowledge, often provided only by domain experts. Our implementation should already provide a start in this direction for any data cleaning practitioner who wants to get insights and recommendations on how to prepare her datasets. Similarly, our data preparation experiments reveal the importance of address normalization, which we explore further in the next chapter.

As possible future work directions, we propose examining the effect of a larger pool of preparators applied to more datasets. Another possible direction is to focus on different subsets of the dataset and not apply the preparator to all records, but only to those that can benefit from it – conditional preparations. Finally, finding the optimal order of preparations' application that we discuss in Phase 3 of Section 3.5, can be difficult if time efficiency is important, but may help improve the results even further.

4

Address Normalization and Matching

Given a database, duplicate detection is the problem of finding database records that represent the same real-world object. In the easiest scenario, database records are completely identical. However, in most cases problems do arise, for instance, as a result of data errors, data integrated from multiple sources or received from restrictive form fields. These problems are usually challenging, because they require a variety of actions, including field segmentation, decoding of values and similarity comparisons, each requiring some domain knowledge. In the previous chapter, we showed the benefits of applying generic data preparation actions to improve the success of duplicate detection.

In this chapter, our task similarly is to detect duplicates, but we focus on domains that contain *address information*, including attributes such as Street-address and City. To facilitate this matching process, we propose a domain-specific procedure [Koumarelas et al., 2018] to first enrich each record with a more complete representation of the address information through geocoding and reverse-geocoding, and second to select the best similarity measure per each address attribute, that will finally help the classifier to achieve the best F-measure. We report on our experience in selecting geocoding services and discovering similarity measures for a concrete but common industry use-case. More specifically, our contributions can be summarized as follows:

- A novel pipeline using three approaches to improve address quality. The first two approaches use frameworks to provide address geocoding and parsing, whilst the third uses conditional functional dependencies.
- Two new similarity measures for small to medium-sized attributes, which are modified versions of existing similarity measures.
- Experiments to verify the effectiveness of our pipeline under different configurations.

The rest of the chapter is organized as follows: In Section 4.1 we motivate the problem of address normalization and briefly introduce our approach. Relevant literature is discussed in Section 4.2. We study the operations allowed by GIS systems, and publicly available datasets for this reason, in Section 4.3. Then, we proceed with our process of finding the best similarity measure, that is comprised of three parts: First, in Section 4.4, preprocessing and enriching records' address information by applying geocoding and using conditional functional dependencies (CFDs) with the help of a reference address database. Second, in Section 4.5 we propose two similarity measures, which serve as extensions of Monge-Elkan [Monge and Elkan, 1996], and then we find the best similarity measure per attribute, among a number of typically used measures. Third, training and using a Random Forest (RF) classifier to determine whether the given similarities represent a duplicate or a non-duplicate pair of records, is explained in Section 4.6. Lastly, we conclude in Section 4.7 with our thoughts for future work, which includes modifications and extensions of this chapter's work.

4.1 Introduction

The problem of *record matching*, which in the literature has been referred to with many different names, including record linkage [Christen, 2012b], duplicate detection [Naumann and Herschel, 2010], entity resolution [Christen et al., 2009] and data matching [Christen et al., 2006], is a well-studied problem [Christen, 2012b; Elmagarmid et al., 2007]. As explained previously, two records match if they represent the same real-world object, such as a person, a product, or a company, despite their differences in their stored values. A typical method of classifying two records as duplicates is by determining the similarity or distance of their corresponding values. If the combined similarity is above a given threshold, the record pair is considered to be a match, and thus returned as a result.

In this chapter, we focus on matching records from a particular domain, i.e., records that contain address information. It is a notably difficult and important operation, which allows for geo-spatial analysis in a number of areas, including Health [Pickle et al., 2005], Traffic Accidents [Miler et al., 2016] and Natural Disasters [Guha-Sapir et al., 2015]. Address attributes, such as City or Street-address appear frequently in many real-world applications, and simultaneously have special (geographic) semantics that can be exploited to more effectively determine matching records. However, a system that is able to exploit these semantics must address several challenges.

Deduplicating addresses is particularly complex, when different countries or even areas within a country follow different *formats* and *rules*[1] for their addresses. Examples of such differences include building names, village and district names, house (building) and flat numbers, street types (street, highway etc.) and more. Additionally, address data does not always follow the proper format, as defined by the corresponding country's standards, most commonly by mixing up the order of attributes or simply missing some values. On top of that, ambiguous abbreviations, slang or synonyms make the problem even harder. For instance, "`St`" could abbreviate Street, Saint, State, Station, etc.[2]

We employ and enhance *geocoding*, which is an operation that transforms a given address to Latitude and Longitude geolocation coordinates. Internally, systems that perform such an operation apply a mixture of techniques to obtain their result, including a special case of record matching against a clean address reference database. Such reference databases include qualitative location records, that contain address information, such as Street-address and City, but also geolocation coordinates, that are used to determine the final result of *geocoding*. These systems are usually described as *geocoding frameworks*, since *geocoding* is their main operation, although other operations are also provided, such

[1]`https://en.wikipedia.org/wiki/Address_(geography)#Mailing_address_format_by_country`

[2]See `http://pe.usps.gov/text/pub28/28apc_002.htm` for more examples.

as parsing and address normalization (see Section 4.3.1). A popular example of references databases is OpenStreetMap (OSM)[3] (see Section 4.2 for more details). The best providers of such systems usually enforce limits on the number of allowed queries, on their free version, and such queries are not always guaranteed to have some result, especially in cases of typographical errors in some of the query's tokens. Access to such services is usually provided with a REST interface and there are tools providing cross-service access, such as the Python geocoder[4].

4.2 Related work

Record matching has been extensively studied over the years, under different names, as mentioned in the previous section. Several works include a thorough analysis of different methods that span across all the necessary steps to (1) normalize, (2) index and retrieve candidates of possibly matching records, (3) compare these retrieved candidates, (4) classify them as matches or not, and (5) evaluate the entire process [Christen, 2012a; Elmagarmid et al., 2007; Herzog et al., 2007]. In this chapter we contribute to the first step, for which we apply *geocoding* and conditional functional dependencies (CFDs) to normalize the addresses of our records, and to the third step by experimentally analyzing the most commonly used similarity measures, and finally to the fourth step to choose a classifier to make the final decisions. For Steps (2) and (5) we employ standard techniques from the literature.

The author of [Paull, 2003] describes abstractly the process of creating a clean reference database, described as G-NAF (Geocoded National Address File for Australia), by merging smaller datasets from 13 organisations, to be used for geocoding purposes, on Australian addresses. The cost for such a project was estimated to around \$12 million and ended up at \$2 million due to improved technology. This still indicates how difficult and complex the procedure is, where even matching 70% of any given records, is in many cases considered an acceptable result. Part of the process was also to resolve licensing problems when merging the smaller datasets into the single database.

Christen et al. [Christen et al., 2006] propose a Hidden Markov Model (HMM) that parses and separates an address into components. By matching these components using a rule-based engine, they identify the best matches from their reference database. Applying their methodology to a subset of G-NAF, they managed to match 94.94% of it on different levels (address, street, locality).

Finally, Miler et al. present a model for matching a traffic accident dataset against the OSM dataset, where the existence of Latitude and Longitude values is a vital part of the process, since the authors only consider pairs of records that are geospatially close [Miler et al., 2016]. In their application, they observed that many street names are based on persons' names. Thus, similarity measures that are suitable for people's names are suggested, and more specifically Jaro-Winkler [Winkler and Thibaudeau, 1991], for which they found the best threshold to be 68%.

[3]`http://wiki.openstreetmap.org/`

[4]`https://pypi.python.org/pypi/geocoder`

4.3 Geocoding

Geographic Information Systems (GISs) commonly provide operations similar to and including *geocoding*. They used to have primarily the form of a desktop application, but nowadays they are typically offered as a service. We first describe the typical set of provided operations. Most of the operators are based on some reference location datasets, which we describe thereafter.

4.3.1 Operations provided by GIS services

Operations that can be performed using these services include the following:

- *Parsing*: Split addresses into its constituent components, including House-number, Street-name, ZIP-code, City, County-name, State-code (or State-name), Country-code (or Country-name).
- *Verification*: Validate whether an address exists based on clean reference datasets, such as the OSM.
- *Geocoding*: Match a given query address to a geolocation of Latitudes and Longitudes, internally using records from clean reference datasets and by applying a mixture of operations, including record matching, interpolation and more. The matched geolocation is returned to the user. More often than not, these clean reference datasets contain entries with more information about a particular point. Thus, it can be also beneficial to include any kind of textual description along with or instead of the address. In our case Hotel-name could be used as complementary information, which could lead to improved results.
- *Reverse (or inverse) geocoding*: For given geo-coordinates, return the nearest location in the reference data, such as an element record in OSM. More specific criteria can be specified for the search results, such as to include only locations of a particular type, for instance, train stations or hotels.
- *Normalization (or standardization)*: Format and clean components of addresses to comply with the country's postal service standards, including special character removal, lower(or upper)-casing, zero padding number fields like ZIP-code, transposition of words, and encoding conversions.

Example providers of these services, to which the user sends his address as a REST request and the system sends back the geocoded coordinates, are ArcGIS[5], Google[6] and OpenStreetMap (OSM)[7]. The latter is the only service based on open source data, available for research. Further open source implementations based on OSM include Nominatim[8], Gisgraphy[9], and Pelias[10]. Out of these, the one that had the largest community support, based on github metrics (number of commits, stars and forks), and also seemed to produce the most reliable results in our case was *Nominatim* and for this reason we selected it for our experiments. In Section 4.4 we explain how we use which operation.

[5]`https://www.arcgis.com/`
[6]`https://developers.google.com/maps/documentation/geocoding/start`
[7]`https://www.openstreetmap.org`
[8]`https://nominatim.openstreetmap.org/`
[9]`http://www.gisgraphy.com/`
[10]`https://github.com/pelias/pelias`

4.3.2 Publicly available datasets of location information

The area of *geocoding* is very fragmented, where high quality datasets exist only for smaller areas, and in contrast when larger areas are covered, quality usually drops. Therefore, we present the most complete, publicly available location datasets we could find. Since a large fraction of our use case's domain considers hotels in US, we also include US-only datasets, apart from the global ones. The first four datasets of the following list contain addresses up to a house level accuracy.

- OpenStreetMap (OSM) is a worldwide volunteered geographic information (VGI) dataset, where people enter information in the form of primitives (nodes, ways, relations), that represent an element[11], and tags that contain metadata about the element. Downloading the latest compressed data for the whole world requires 87GB of disk space.
- OpenAddressesIo[12] is "a global collection of address data sources, open and free to use", which was started by OSM users and is based mostly on government datasets. The latest compressed data available for download requires 10GB of disk space.
- GeoNames[13] is another worldwide VGI dataset, which also allows for user modifications. This dataset is available in a compressed form that has a size of 347MB.
- TIGER US Census[14] is the acronym of topologically integrated geographic encoding and referencing system, and is provided at a national level by the US Census Bureau.The latest version is available in compressed files with a total size of 20GB.
- Country Mappings[15] was our source of combinations of country codes (2 and 3 character encoding) and the commonly used country's name. Since the number of countries is relatively small, acquiring the mapping in a compressed form requires 9KB of disk space.
- US-specific: We collected further data from various public data sources[16]. While all files contained basic attributes, such as ZIP-code and State, they differed widely in which other attributes they provide and in their quality and coverage. In total, the integrated file in a compressed form requires 1.4MB of disk space.

To make use of the US-specific datasets, we performed a merge operation to construct a single relation to include: Latitude, Longitude, ZIP-code, City, County-name, State-code (and State-name), Country-code (and Country-name). When pairs of records matched perfectly for all attributes except Latitude and Longitude, we merged the records and fused Latitude and Longitude to their average values.

4.4 Enriching addresses

Postal services of many countries issue recommendations for address formats, which are not always followed in practice, which in turn leads to different representations of the same address. These formats change across local areas, such as cities, states etc.,

[11] http://wiki.openstreetmap.org/wiki/Elements
[12] https://openaddresses.io/
[13] http://www.geonames.org/
[14] https://www.census.gov/geo/maps-data/data/tiger-line.html
[15] https://github.com/mledoze/countries
[16] http://federalgovernmentzipcodes.us/, http://simplemaps.com/resources/us-cities-data, http://www.sqldbpros.com/2011/11/free-zip-code-city-county-state-csv/, http://www.unitedstateszipcodes.org/zip-code-database/, https://www.bls.gov/cew/cewedr10.htm

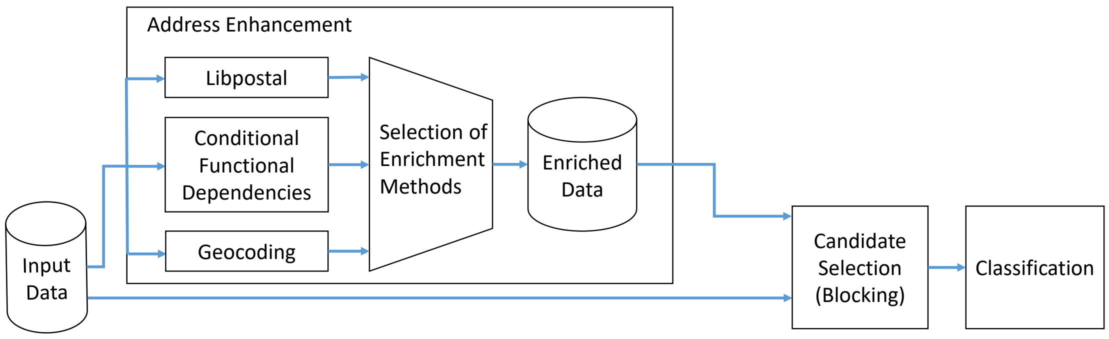

Address Enhancement
Libpostal
Conditional Functional Dependencies
Geocoding
Selection of Enrichment Methods
Enriched Data
Input Data
Candidate Selection (Blocking)
Classification

Figure 4.1: Address enrichment process workflow.

and more obviously across countries, which also use different languages. In addition, addresses may contain abbreviations, misspellings, mixed orders of attributes, or missing attributes. To handle the plethora of different formats and errors, we need to normalize and parse the addresses, to ultimately compare them on a per-attribute level.

In this section, we first define the different processes that we used to enrich addresses. Second, we describe our use-case dataset about hotels, and finally we evaluate the enrichment processes on the dataset. This process can be seen in Figure 4.1, where we show the executions over the original data and the enriched ones. The latter are a combination of the three proposed enrichment processes. Having either of the two data sets, in real-world applications we then perform the candidate selection, using blocking, to select the most promising candidate pairs for the next step, which in our case is duplicate classification. Keep in mind the lowest arrow presented in the figure denotes that in our experiments we perform the candidate selection before any enrichment takes place, on the original data, to ensure that the same set of candidates is selected with or without enrichment.

4.4.1 Enrichment processes

We describe three different processes to enrich addresses. Each can be used in isolation or in conjunction, as we show in Section 4.4.3.

Parsing

Since address parsing, i.e., labeling the individual parts of an address, is a difficult task in its own right[17], we decided to use the popular address *parser* and *normalizer* Libpostal[18]. Libpostal claims to have an accuracy of 99.45% and support for over 60 countries.

Libpostal first *expands* the address, eliminating abbreviations and converting numbers to a uniform representation. In a second step, each part of the address is labeled according to its meaning. As an example the address “`1317 e hwy sixty seven decatur 35601 alabama`” would be normalized to “`1317 east highway 67 decatur 35601 alabama`” and afterwards parsed to “House-number: `1317`, Road: `east highway 67`, City: `decatur`, ZIP-code: `35601`, State-name: `alabama`”. Thus, the parsing step can also be used to infer

[17]`https://www.mjt.me.uk/posts/falsehoods-programmers-believe-about-addresses/`
[18]`https://mapzen.com/blog/inside-libpostal/`

attribute values, which might exist in wrong positions; for instance the Street-address may also contain other attribute values, such as the city.

Enrichment through conditional functional dependencies

Filling missing attributes is an important task, which, apart from using Libpostal's parsing, may be facilitated using other approaches. If there exists sufficient attributes with values, we can use functional dependencies (FDs), to infer some of the missing values.

As an example consider the dependency between a ZIP-code and a city. If we know that the FD ZIP-code→City holds, and we know the ZIP-code of a record, then we can infer the city, by consulting a reference address database (such as those described in Section 4.3.2). Nonetheless, even for this example of ZIP-codes, the problem might be more complicated, since a ZIP-code can be either inside one city, span over a region of multiple cities (that can be in multiple counties or states), commonly found in military bases, and the same ZIP-code can exist in many countries across the globe.

General FDs are valid only if they apply across the entire dataset. *Conditional* functional dependencies (CFDs), on the other hand, are FDs that need to apply only to a subset of the records, as specified by some condition. As shown in [Ilyas et al., 2015], partial enrichment with CFDs is easy to apply, simple, and better than other methods. Moreover, it can enrich records by itself, resulting in improved results, or enhance the address normalization step (Section 4.4.1). For instance, the functional dependency ZIP-code→City might be true only for a specific country.

To make use of this concept, for each incoming record we examine the present non-null values. For each combination of these values we check with the reference database whether we can uniquely determine a value for one of the missing attribute values. Consider a record with (`98103`, `us`) as ZIP-code and Country-code but a missing value for City. We collect all records with the given values (`98103`, `us`) and count the number of distinct values for City. There are three possible outcomes: (i) If the original count is 1, we were successful and can infer the right hand side (RHS) attribute value. There is only one single city for the combination (`98103`, `us`), namely `seattle`. (ii) If the count is 0, we cannot directly infer a missing value. Thus, we relax this left hand side (LHS) condition by considering fewer attributes to infer the missing attribute value. In this case, we recursively count the number of distinct cities for the ZIP-code `98103` only and the number of distinct cities for the Country-name `us` only. (iii) If the count is higher than 1, we cannot uniquely infer the city, and there is no reason in examining more relaxed LHS conditions, as those will certainly have more results. Once a single value has been determined, we iteratively use this new value in the LHS to create further, more complex combinations, to check until no new values can be filled in. The correctness of this process is bound to the validity and completeness of the provided reference database. In particular, erroneous values and lack of records can lead to invalid CFDs being labeled as valid and vice versa.

For lack of a high-quality reference dataset, we implemented this process for US addresses only, and were able to infer information across the address attributes of ZIP-code, City, County-name, State-code, State-name, Country-code (2 or 3 character format), Country-name (common and formal names). In principle, the same procedure could be followed at a global level, for instance by parsing the entire OSM dataset and producing a relation with the above information.

Enrichment through geocoding

The previous steps could be used in isolation, but should also enable us to be more successful during the geocoding process. As previously mentioned, the goal of this step is to match the query's information with a record from a clean reference database (described in Section 4.3.2) to obtain its geolocation. Afterwards, by performing *reverse geocoding* we obtain a proper, formal representation of the record. In fact, most geo-coding systems return the reference record already during the *geocoding* step. The actual process of geocoding is more difficult than what has been already described in Section 4.2, including house number interpolation, in case the requested address is not available but neighboring house numbers are, synonyms in other languages, font encoding problems when not everything is in UTF-16 (usually UTF-8 is used), and more. We chose to use the open source system Nominatim, which we set up as a local server.

4.4.2 Use-case: Hotel dataset

To evaluate the different approaches for geocoding and matching, datasets a gold standard is needed. The problem with publicly available datasets, such as the North Carolina Voter Registration dataset[19], is that they do not contain significant variations among duplicates - as their quality is typically better than what we observe in our real-world queries. Therefore, we proceeded to use a real-world dataset that our industry partner Concur[20] provided, which includes information of *hotels* around the world, along with a gold standard of duplicate pairs. This dataset includes 364,965 records with 36 attributes and a size of 91.2MB. Along with the records, Concur reported 384,238 duplicate pairs, i.e., pairs of records that have been verified to represent the same hotel. Keep in mind that clusters of multiple records representing one hotel lead to many duplicate pairs. In fact, the largest cluster contained 54 records (accounting for 1,431 pairs), and 124,653 records had no annotated duplicates. The hotel dataset includes the attributes Hotel-name, Street-address, ZIP-code, City, County-name, State-code, Country-code, Latitude, and Longitude, which all contain real-world, human-made, errors, misspellings, and frequently, empty (null) values.

4.4.3 Experimental evaluation

Geocoding is the most important of the processes, because it enriches records with the most reliable address information. We used the first two of the processes as pre-enhancing steps for the last step, aiming for better geocoding results. In this section we report how many hotel records are in fact matched and enriched by these combinations of the processes. We focused on the eight countries (US, FR, DE, GB, IT, CA, CN, AT) that represent the largest fraction of our records, keeping 240,944 records and 301,155 pairs of the gold standard. All our experiments were conducted with an Intel Xeon CPU, 3.07 GHz, 8 Cores, a RAM of 24GB and a HDD of 2TB.

By examining Table 4.1, we can observe first the impact on the percentage of records that have been enriched, second the percentage of duplicate pairs from the gold standard

[19]`http://dl.ncsbe.gov/index.html?prefix=data/`

[20]`https://www.concur.com/`

for which *both* of the records have been enriched, and finally out of this the percentage of records where the geolocation distance, based on latitude and longitude, is zero. These statistics help us to make the decision among four choices, namely, (i) Nominatim, (ii) Libpostal + Nominatim, (iii) CFD + Nominatim, and (iv) CFD+Libpostal+Nominatim, each of which choice includes a combined enrichment processes, representing a set of consecutive steps. We chose the combination CFD + Nominatim for its simplicity and good results. A brief experimental comparison with an Ensemble approach that combines the results of all four choices is available in Section 4.6.2.

Table 4.1: Selecting the best enrichment process, based on gold standard of 240,944 records and 301,155 pairs.

Enrichment process	**% records**	**% pairs**	**% pairs with 0 distance**
Nominatim	24.29	16.62	73.83
Libpostal + Nominatim	20.68	14.15	75.70
CFD + Nominatim	37.21	29.07	78.73
CFD + Libpostal + Nominatim	33.42	26.88	80.17
Ensemble	43.65	34.89	77.31

Regarding the execution time per record, the mean values that we observed under a multi-threaded (Java 8 parallelStream) execution, were 17ms for CFDs, 96ms for Libpostal and 4,057ms for Nominatim. In total, the execution time of the classification process, not including the enrichment, was increased from 1,109s to 1,329s (+220s) due to the added information contained in the attributes and consequently in the similarities. For performing the GET requests on Nominatim, multiple Java libraries were tested and the execution times were always at similar levels. The final Java library that was used is unirest[21], and we set the timeout of a request to 20s. We assume that Nominatim's long execution time may happen because of long system retrieval times, e.g., for frequently occurring tokens. However, the use of a multi-threaded environment keeps the overall execution time over the whole dataset under a more reasonable time frame, which was approximately a day for our 240,944 records. Let us now examine a few experiences in using those systems:

Libpostal

By using Libpostal before calling Nominatim we try to achieve two things:

(1) Fill missing values, that might have been contained in other attributes. By *parsing* a concatenated string where we join all the information about the address attributes that we know about, we expect to improve our quality of attributes. However, several issues are faced here:

- Improper categorization of tokens. A ZIP-code could be identified as a House-number and vice versa, a Street-name as a City, etc. As an example, the result of the input "`7210 ga hwy 21 31407 port wentwort ga us`", incorrectly (and inexplicably) returns `31407` as the House-number and `7210` as the ZIP-code, swapping their respective categories.

[21]`http://unirest.io/java.html`

- Non-identification of certain tokens leads to the concatenation of tokens. Given the input "`4900 bryant irvin rd 76132 3616 fort worth tx us`" Libpostal returns the House-number 49003616.

These two types of problems appeared in more than 50% of the cases in total. This made the *parsing* part of this library unreliable and was thus avoided.

(2) *Expand* and normalize the address query that will be given to Nominatim, to eliminate abbreviations (i.e., `St` is converted to `Street`), represent numbers in a uniform format (i.e., `IX` is converted to `9`) and correctly position the tokens to reflect the correct pattern (i.e., house-number in front, then street-name etc.). This should increase the success rate of geocoding. Unfortunately, Libpostal returns multiple, unranked expansions as shown in Table 4.2. Therefore, having no straightforward solution to select the most probable and correct expansion, our strategy here is to select the largest expansion. Our intuition is that a larger value is in most cases more specific and thus has more chances to be correct.

Table 4.2: Libpostal unreliable expansions

"507 e main st 99328 dayton wa us"
"507 e main street 99328 dayton wa us"
"507 e main street 99328 dayton western australia us"
"507 e main street 99328 dayton washington us"
"507 e main saint 99328 dayton wa us"
"507 e main saint 99328 dayton western australia us"
"507 e main saint 99328 dayton washington us"
"507 east main street 99328 dayton wa us"
"507 east main saint 99328 dayton wa us"
"507 east main saint 99328 dayton western australia us"
"507 east main saint 99328 dayton washington us"

Table 4.3: CFDs example

attribute	original value	enriched value
City		marshal
ZIP-code	56258	56258
County-name		lyon
State-code	mn	mn
State-name		minnesota
Country-code	us	us
Country-name		united states

Conditional Functional Dependencies (CFDs)

The goal as stated in Section 4.4.1 is to enrich the record with information that can coexist only with our existing attributes' values. As an example in Table 4.3, we are able to obtain the City, State and the County-name as well. Overall, our entire process was able to enhance $\frac{131{,}391}{135{,}583} = 96.90\%$ of the US records with at least one additional attribute value. Using Nominatim alone yielded only 58,529 enhanced records (43.16%); using CFD + Nominatim led to 89,670 (66.13%) geocoded records.

Geocoding

When geocoding is successful, we apply reverse geocoding to obtain proper address details, as explained in the beginning of Section 4.4.1. The address information that is returned contains important additions and is useful for our later matching process. As an example consider the Table 4.4, where by providing the query of the original record to Nominatim, the returned result contains many more address attributes, which clarifies better the location of the hotel. In total, considering the ten address attributes presented in Table 4.4, before any geocoding, out of the 2,409,440 attribute values across the records 1,087,837 are filled (45.15%). Nominatim increases this value fulfillment to $\frac{1{,}380{,}476}{2{,}409{,}440} = 57.29\%$ and finally CFD+Nominatim increases it to $\frac{1{,}831{,}439}{2{,}409{,}440} = 76.00\%$.

Table 4.4: *Geocoding* (Nominatim) example

attribute	**1. original record**	**2. geocoded** to	**3. reverse geocoded to**
Latitude		42.6334525	42.6334525
Longitude		-88.6371988	-88.6371988
Street-address	511 e walworth ave		511 e walworth ave
ZIP-code	53115		53115
City	delevan		delevan
County-name			walworth county
State-code	wi		wi
State-name			wisconsin
Country-code	us		us
Country-name			united states
Query	511 e walworth ave 53115 delavan wi us		
Display-name			walworth avenue, delavan, walworth county, wisconsin, 53115, united states of america

4.5 Choosing similarity measures

Address information, which is the focus of this study, feature a wide variety of attributes and attribute types. The goal of this section is to recommend specific similarity measures for common attributes of addresses. To this end, we first introduce popular measures and then evaluate each of them for each of the attributes, recommending a measure for each.

4.5.1 Similarity measures

A similarity measure is a function that takes two input values, $v1$ and $v2$, and returns a value between 0.0 and 1.0, with 0.0 meaning they are completely different and 1.0 meaning they are exactly the same: $sim(v1, v2) \rightarrow [0.0, 1.0]$

The similarity measures we consider include the ones discussed in Chapter 2, for which we use the following abbreviations: Exatch Match (EM), Hamming (HM), Levenshtein (LS), Damerau-Levenshtein (DL), Jaro-Winkler (JW), LongestCommonSubsequence (LCS), Jaccard (JC), Jaccard n-gram (JCN), and Monge-Elkan (ME). Moreover, we introduce the following two hybrid similarity measures, which we found useful for our use-cases:

- Monge-Elkan Greedy Symmetric (MEG) modifies ME to avoid the re-use of matched tokens, giving both input values the same importance. We calculate the similarity of each token pair using the internal similarity measure and then greedily choose best matching pairs.
- Stable-Matching (SM) ensures that no token is more similar to another token than to the one it is matched to, while simultaneously that other token is also more similar than to its own match. We use the Gale Shapley algorithm [Gale and Shapley, 1962] to find the matching. As the algorithm is not symmetric, we repeat it with swapped input values and use the average similarity as the final result.

We combined the three hybrid measures ME, MEG, and SM with three different internal similarity measures, namely Levenshtein, DamerauLevenshtein and JaroWinkler, abbreviated as ME-LS, ME-DL, etc.

For the purpose of duplicate detection we define a *threshold* for each similarity measure and attribute: If the similarity is above the threshold, we call it a *match*, if it is below, it is a *non-match*. Having defined the different similarity measures, we proceed in the next section to identify what is the best mapping between them and the common address attributes, and which threshold to choose for each combination.

4.5.2 Experimental evaluation

We selected the seven attributes described in Section 4.4.2, which are available in most countries. In this section, we first describe the process to determine precision, recall, and F-measure. Second, we evaluate all the previously defined similarity measures across our selected record's attributes, and select combinations with the highest potential in terms of some metric, such as F-measure or execution time. For all experiments we perform a 10-fold cross-validation on our dataset and report the mean scores.

Selecting duplicate and non-duplicate pairs

The gold standard that is given to us by our industry partner, contains 130,428 pairs of records that are marked as *duplicates (DPL)*. Since we want to train a classifier for duplicate detection, we need to calculate the F-measure, based on precision and recall values, which in turn needs knowledge about false positives (FP) that are returned in a detection run. Identifying FPs needs *non-duplicate (NDPL)* pairs, and for this reason we created a process that produces them, in the same spirit as [Christen, 2007]. This process follows the same principles to the one discussed in Chapter 3, but here we follow a different blocking scheme, tailored to Hotels, which performed better for our experiments in this chapter. Since we already own a DPL set, we follow a different strategy, based only on blocking. It would be easy to choose many trivial NDPL pairs, simply by choosing many very dissimilar records. For a more realistic evaluation we describe how we generated more difficult NDPLs:

1. We insert all DPL into a UnionFind (UF) [Galler and Fisher, 1964] data structure, which provides us with transitive closure functionality. UF forms transitive closure groups, which include all different representations (records) of the same entity, that are connected in the gold standard through direct and indirect (transitive) connections.
2. We use the blocking technique to partition the records and thus save comparisons and reduce the number of trivial comparisons. We apply blocking individually to the attributes Hotel-name, Street-address, City, and Zip-code, using the entire value as the blocking key. These attributes have a high uniqueness, which means that the same value is not common across many records. Each bucket represents all records which have the bucket's value for that specific attribute. Finally, we go through all these attributes and their respective buckets, fetch all record combinations, and add them to a collective set of pairs, that contains both DPLs and NDPLs.
3. We remove the pairs that belong in the same transitive closure group, i.e., they are duplicates, using the UF structure.

Since pairs of NDPL have the same value in at least one attribute, this guarantees that their similarities have to be high, in contrast to randomly selected, where no guarantees are provided. Thus, distinguishing them from pairs of DPL is more challenging and, therefore, realistic. The two sets of pairs, DPL and NDPL, constitute our full gold standard (*Full-GS*), which we use for evaluation in the following steps.

Best similarity measure per attribute

To determine the best similarity measure for each attribute, we evaluate individually for each one all similarity measures and calculate the F-measure for 100 different thresholds ([0.01, 0.02, ..., 1.0]). We thus now know for each similarity measure the best F-measure, along with the corresponding thresholds and the overall execution time. We report the thresholds for which optimal results were achieved, but do not apply them in the classification step. We leave the selection of feature thresholds to the Random Forest classifier. With this information users can select the similarity measure that maximizes the F-measure, minimizes the execution time, or something in between. In our case we solely target on maximizing the F-measure. We discuss some exemplary results:

- Hotel-name: Figure 4.2a shows JCN to achieve the highest F-measure, while at the same time being one of the fastest measures as well. The best F-measure was achieved for the threshold of `0.29`.
- Street-address: MEG-JW is shown in Figure 4.2b to have the best F-measure, with the threshold of `0.95`. A Street-address is usually comprised of multiple tokens, explaining the good performance of this token-based measure.
- ZIP-code: The best choice as similarity measure is the SM-DL, achieved at the threshold of `0.9`, as shown in Figure 4.2c, since there are ZIP-codes that contain multiple tokens.
- City: Figure 4.2d shows that the best choice as similarity measure is the SM-DL, with `0.46` as the best performing threshold.

- County-name: For county names the selected similarity measure is the same as ZIP-code's and City's – SM-DL, with the threshold of `1.0`, as shown in Figure 4.3.
- State-code and Country-code: Since these codes have a length of two or three characters, we prefer to have an exact match. Therefore, we use the EM similarity measure.
- Latitude & Longitude: We select the well-established Haversine distance[22].

4.6 Classification for duplicate detection

Having enriched records and knowing the best similarity measure per attribute, from Sections 4.4 and 4.5, this section introduces a classifier to detect which of these pairs are duplicates or not. As features we provide the calculated similarities, for every pair of the *Full-GS* (discussed in Section 4.5.2), with the label `1` representing duplication and `0` the opposite. To this end, we have chosen Random Forest (RF) as the classifier for our experiments[23], for its scalability, ease in parametrization, and out-of-the-box applicability to most problems. This makes it easier to deploy in real-world applications. We have also experimented with Support Vector Machines (SVM), which needed excessive memory while performing only slightly better on our samples. An additional approach we have tried was to use a threshold-based classifier, where each attribute's similarity is weighted, and if it passes a given threshold the pair is a match. We also employed a Genetic Optimizer, since the problem of maximizing F-measure is non-convex [Nan et al., 2012], to find the optimal weights and threshold, but achieved only much lower F-measures. Another industry-relevant reason to choose RF is its ability to explain specific outcomes.

In the next sections, we describe how to first *train* and second *evaluate* the RF classifier on our domain's dataset.

4.6.1 Best parameter for random forest classifier

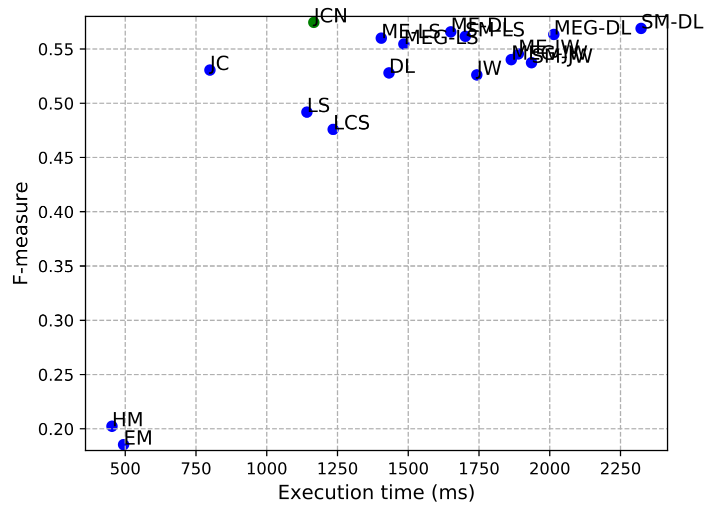
The RF classifier has a number of different parameters that can be configured, with the number of *trees* being typically the most important parameter to be examined, which we also do in our experiments. Further parameters include the number of *attributes* to be sampled upon at each node, and the criterion to make the binary *splits* in each node. For both these parameters, as well as all remaining ones, we keep to the library's default values[24], which are typically used in practice [Tibshirani et al., 2013]; for the former this is $\sqrt{m}$, with m being the number of attributes, and for the latter this is the GINI index. To decide the number of *trees*, we calculate all pair similarities using values of enriched records following the *CFD + Nominatim* enrichment process, which in Section 4.4.3 performed the best. Afterward, we perform experiments across countries using the selected number of *trees* to get further insights into the model's performance.

In Figure 4.4a we can see the differences in F-measure for the subset of records that have been enriched by *CFD + Nominatim* and in their original state. In other words, it helps us understand whether the enrichment helps (it does) and at the same time it

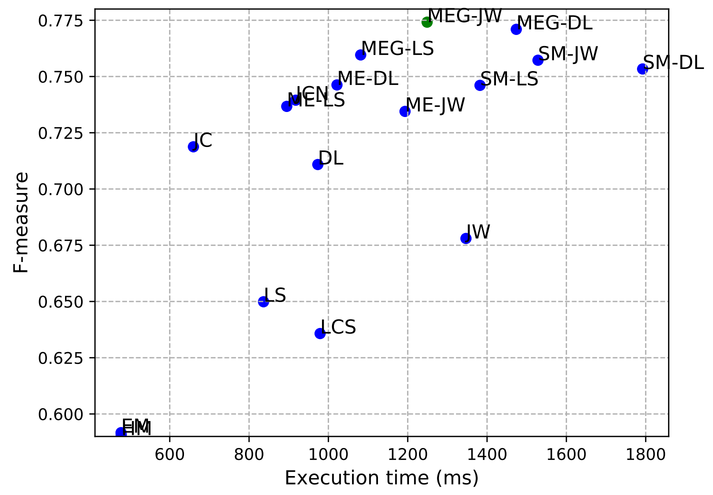
[22]`https://en.wikipedia.org/wiki/Haversine_formula`

[23]We used the Java library from `https://haifengl.github.io/smile/`.

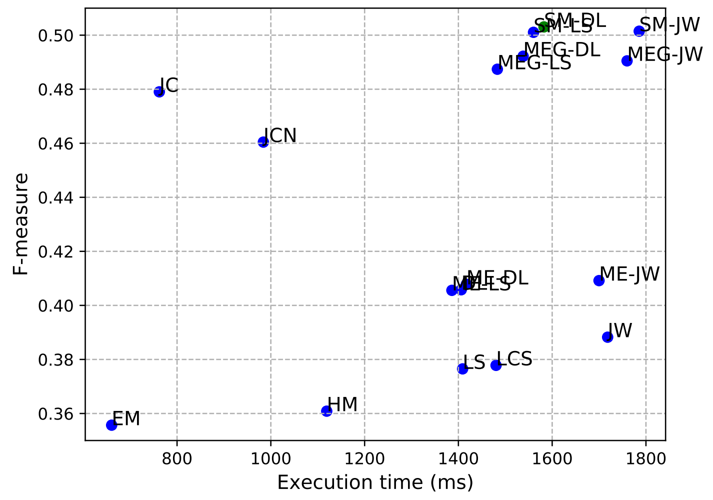
[24]`https://github.com/haifengl/smile/blob/master/core/src/main/java/smile/classification/RandomForest.java`



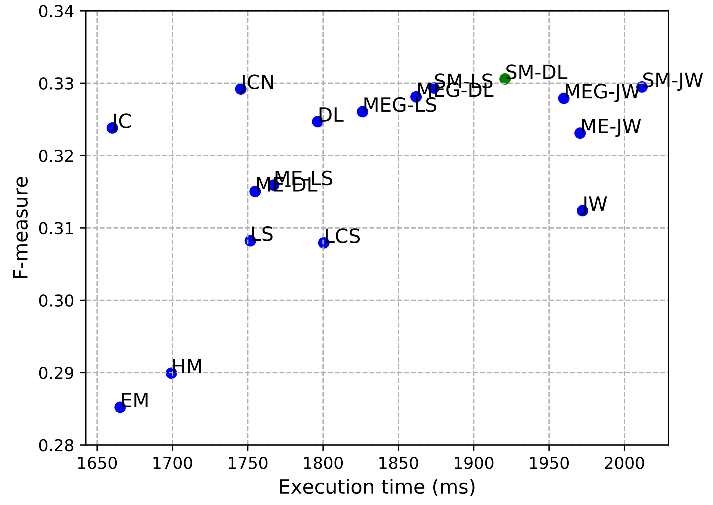
(a) Hotel-name



(b) Street-address



(c) ZIP-code



(d) City

Figure 4.2: F-measure for similarity measures per attribute.

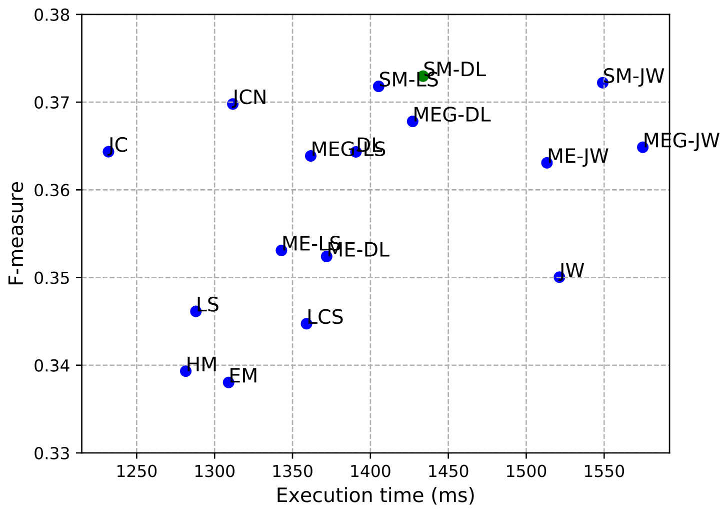
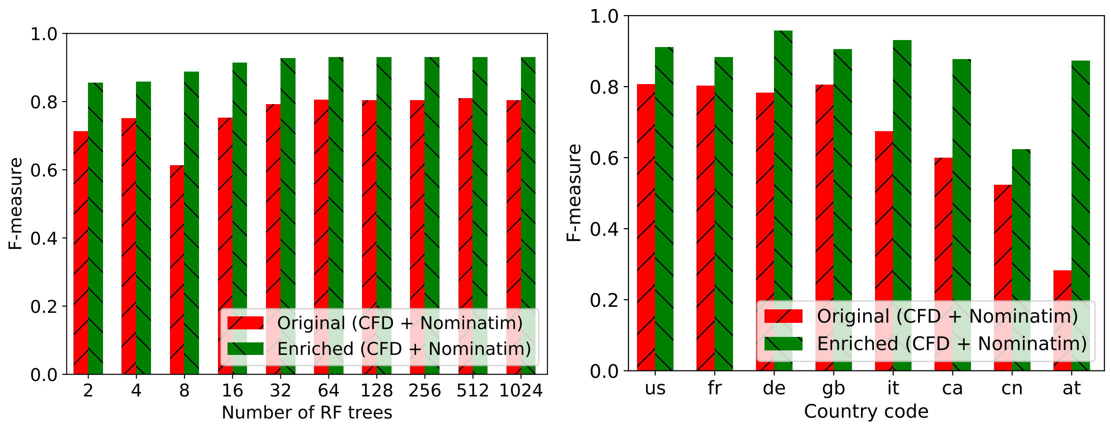


Figure 4.3: F-measure for similarity measures on County-name.



(a) F-measure across number of RF trees.

(b) F-measure across countries. (64 trees)

Figure 4.4: Effect of Random Forest (RF) on F-measure under different configurations.

shows us that for 64 trees we have a good compromise between the two data states, achieving an improvement of 12%, keeping in mind that also in future applications we cannot expect all records to be enriched. We continue our experiments with 64 trees.

In Figure 4.4b, we can observe the effect of enrichment across the eight countries, that were selected in Section 4.4.3. In “US” both *CFDs* and *Nominatim* contribute to the result, whereas in the rest countries only Nominatim can enrich the records. Overall, we observe that in all countries, the effect of the *CFD + Nominatim* enrichment is positive, with the improvement in “US” being 10% and “AT” having the most significant improvement with 38%.

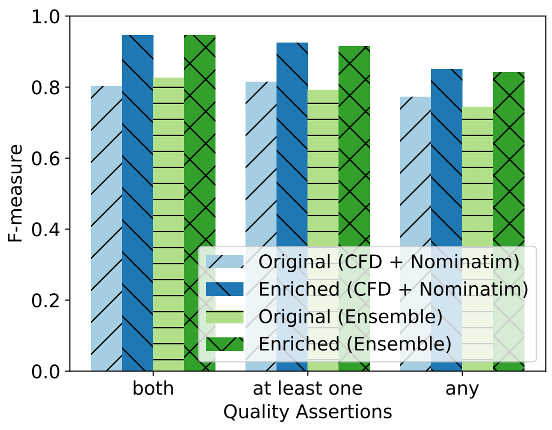


Figure 4.5: F-measure across GS usages

4.6.2 Evaluation with quality assertions

In search of understanding the importance of the enrichment, we performed three experiments by controlling the quality of the used record pairs. We created three datasets in which pairs from Full-GS were either "both" enriched, or "at least one" was enriched, or "any" on enrichment. Based on these three configurations, we can aim for a good intuition on how much enrichment contributes to the final matching quality. The previous experiments took place using the middle solution ("at least one") as the default configuration.

In Figure 4.5, we can see the difference among these three configurations with two things being clear: First, the quality of address values is better in the configurations where one or both of the records are enriched, which means that our hypothesis that enrichment does contribute to a better duplicate record classification holds true. Second, the cases that were enriched and performed better than the original, were higher quality ones, as the original records in the two left configurations have a better F-measure. Last, the improvement of using the *CFD + Nominatim* process is profound, although *Ensemble*, while being more expensive, manages to affect more records while achieving almost the same F-measure or even better.

4.7 Conclusion

In this chapter's work we showed a recipe for matching records when address information is present, in which case we can achieve better classification results. More specifically, we showed that by using different address enrichment processes, and by finding the best similarity measure for each address attribute, we were able to improve the F-measure results of the consequent Random Forest classifier over the non-enriched addresses. Regarding the address enrichment processes, Conditional Functional Dependencies along with a Geocoding framework, Nominatim in our case, achieved the best results. For most of our attributes, hybrid similarity measures performed the best. In particular, for Hotel-name Jaccard n-gram performs the best under the threshold of `0.29`, whereas for Street-address our improved version of Monge-Elkan with Jaro-Winkler as the token similarity measure was the best using the threshold of `0.95`. Finally, for the attributes

ZIP-code, City, and County-name, Stable Matching with Damerau-Levenshtein as the token similarity measure performed the best, with thresholds of `0.9`, `0.46`, and `1.0`, respectively.

Several avenues of future work emerge: As reference databases grow ever larger (the complete OSM is already 803 GB), a better coverage can be achieved, but matching methods need to be distributed among multiple machines. Second, new geocoding frameworks are continuously emerging, so experimenting with them is a necessary part on all address-related tasks. Finally, achieving all the above in lower execution times demands for some distributed environment, such as Apache Spark[25], which is another interesting direction, where experimentation with distributed indexes, caches and more, could help.

Concluding this chapter provided us with even more evidence of the importance that improving data quality not only helps downstream applications but also duplicate detection. Experimenting with generic and address-specific data preparation steps showed us the potentials of emphasizing further on the start of the process, which is its data and improve their quality before we continue with the remaining steps of duplicate detection. While, as explained previously, we do have potential future work directions in mind, we prefer to turn our attention to other compelling areas of duplicate detection. The particular issue we address in the following chapter is, to some extent, even more fundamentally important than what we addressed here. If a gold standard exists, evaluating whether an approach is successfully detecting duplicates is straightforward. However, a gold standard is not always available, and thus in the next chapter, we consider an approach to detect duplicates even if no labeled pairs are available to guide machine learning approaches in creating a classifier.

[25]`https://spark.apache.org/`

5

Duplicate Detection with Matching Dependencies

Duplicate detection is an integral part of data cleaning and serves to identify multiple representations of same real-world entities in (relational) datasets. The previous chapters introduced approaches from the literature as well as ours to perform duplicate detection. While all of them are effective for detecting duplicates, many of them are also hard to parameterize or require pre-labeled training data, which is similarly true for our own data preparation approaches. Both parameterization and pre-labeling are at least domain-specific if not dataset-specific, which is a problem if a new dataset needs to be cleaned.

For this reason, in this chapter we propose a novel, rule-based and fully automatic duplicate detection approach [Koumarelas et al., 2020b] that is based on *matching dependencies (MDs).* Our system uses automatically discovered MDs, based on an efficient and state-of-the-art approach [Schirmer et al., 2020], various dataset features, and known gold standards to train a model that selects MDs as duplicate detection rules. Once trained, the model can select useful MDs for duplicate detection on any new dataset. To increase the generally low recall of MD-based data cleaning approaches, we propose an additional classification refinement step. Our experiments show that this approach performs well, with the exception of some datasets that unfortunately performed poorly. We reach up to 94% F-measure and 100% precision on our evaluation datasets, which are good numbers considering that the system does not require domain or target data-specific configuration. Exceptions to these satisfactory results are datasets where the discovered MDs cannot provide useful duplicate pair classification, and thus our method performs rather poorly. Our contributions are summarized as follows:

- **MDedup system**. A fully automatic end-to-end duplicate detection system that is based on discovered matching dependencies. The system is trained on a few datasets with a gold standard, but can then be applied to arbitrary unseen datasets. It adapts several well-known techniques and combines them into a novel, domain-agnostic duplicate detection solution.
- **Feature definition**. A set of novel features and heuristics that serve to train a model on how to distinguish accurate from inaccurate MDs for duplicate classification tasks.
- **MD selection**. An efficient algorithm to select the best subset of MDs for dupli-

cate detection either w.r.t. some gold standard or a pre-trained quality prediction ML model.

- **Evaluation**. Various experiments that demonstrate the effectiveness of *MDedup* on eight real-world datasets with different domains and sizes.

The rest of the chapter is organized as follows: Section 5.1 motivates the problem of unlabeled data and briefly introduces our solution. Afterwards, Section 5.2 provides a summary of related work in automatic duplicate classification and matching dependencies. Section 5.3 then introduces the most relevant concepts before Section 5.4 describes our *MDedup* system's training methodology. Then, Section 5.5 presents the application methodology, used to obtain duplicates in a new dataset. Section 5.6 provides the evaluation setup of our experiments. In Section 5.7, we evaluate *MDedup* and finally conclude in Section 5.8 with further remarks and future work.

5.1 MD-based duplicate detection

Data cleaning is a multivariate process that identifies and repairs various issues in given datasets. Duplicates, which specify multiple representations of same real-world entities in a database, are among the most addressed and harmful data quality issues. Hence, their detection plays an important role in data cleaning processes, with a plethora of approaches existing to effectively detect and merge duplicate records [Christen, 2012b; Elmagarmid et al., 2007]. A large portion of these approaches is based on machine learning (ML) techniques [Bilenko and Mooney, 2003; Elmagarmid et al., 2007]; several of them employ deep learning methods [Ebraheem et al., 2018; Mudgal et al., 2018]. Most other duplicate detection approaches are rule-based and follow systematic detection strategies.

Irrespective of whether ML is used or not, all known solutions require either careful manual configuration by domain experts and/or exhaustive pre-labeling of training data, which are both difficult requirements that we want to avoid. To this end, we use automatically discovered *matching dependencies (MDs)* as rules and devise a system called *MDedup*, which automatically selects the best discovered MDs for the duplicate detection process, without needing any special domain-specific configuration or training data for the dataset at hand. Other, non-MD-based rule languages for duplicate detection, such as [Singh et al., 2017; Wang et al., 2011], allow to formulate more expressive rules than MDs, but these rules are not discoverable without domain knowledge.

Most duplicate classifiers effectively distinguish true and false pairs, but many – especially ML-based ones – cannot provide a human understandable reasoning for their decision. Our approach, in contrast, does this by offering the rules, i.e., MDs that marked the pairs as true duplicates.

Intuitively, a matching dependency is a functional dependency $X \to A$ with a set of left-hand-side (LHS) attributes X and a right-hand-side (RHS) attribute A where attribute values do not need to be exactly equal but *similar* w.r.t. some attribute-specific similarity measure and some attribute- and dependency-specific similarity threshold (more details in Section 5.3.1). The similarities in these rules are expressed in the range of [0.0, 1.0] with 0.0 describing fully dissimilar values and 1.0 equal values.

Consider, for example, Table 5.1, which shows two example duplicates (pairs with ids 1 and 2) and two example non-duplicates (pairs with ids 3 and 4) from the real-

Table 5.1: A sample of duplicate (<163,164> and <165,166>) and non-duplicate record pairs (<180,823> and <676,811>) from the restaurants dataset. The floating point numbers indicate value pair similarities.

pair id	record id	name	phone	address	city	type
1	163	georgia grille	404 352 3517	2290 peachtree rd peachtree square shopping center	atlanta	american
	164	georgia grille	404 352 3517	2290 peachtree rd	atlanta	southwestern
		1.0	1.0	0.36	1.0	0.17
2	165	hedgerose heights inn	404 233 7673	490 e paces ferry rd	atlanta	international
	166	hedgerose heights inn the	404 233 7673	490 e paces ferry rd ne	atlanta	continental
		0.84	1.0	0.91	1.0	0.3
3	180	ritz carlton cafe buckhead	404 237 2700	3434 peachtree rd ne	atlanta	american new
	823	ritz carlton cafe atlanta	404 659 0400	181 peachtree st	atlanta	american new
		0.74	0.58	0.6	1.0	1.0
4	676	johnny rockets la	213 651 3361	7507 melrose ave	la	american
	811	johnny rockets at	770 955 6068	2970 cobb pkwy	atlanta	american
		0.89	0.33	0.18	0.29	1.0

world *restaurants* dataset (see Section 5.6.1). Given this dataset, our *MDedup* system detects (among others) the second pair of Table 5.1 as duplicate using the rule $\texttt{name}_{0.7}, \texttt{address}_{0.7} \rightarrow \texttt{phone}_{0.75}$ as an explanation for this decision. This rule is interpreted as follows: All record pairs in restaurants with a name similarity of at least 0.7 and an address similarity of at least 0.7 have a phone similarity of at least 0.75. This *if-this-then-that* rule is true for all record pairs, but the LHS matches only a few records, which are those that are considered to be duplicates. In our example, the LHS is true only for record pair 2 so the MD classifies it as a duplicate; the pairs 1, 3, and 4 also meet the MD, but they do not fulfill the LHS condition. Note that matching dependencies can be used also in other ways for data cleaning, e.g., to automatically correct RHS attribute values [Bahmani et al., 2012; Gardezi et al., 2012], but we focus on their duplicate detection ability in this chapter.

On the entire restaurants dataset, the matching dependency $\texttt{name}_{0.7}, \texttt{address}_{0.7} \rightarrow \texttt{phone}_{0.75}$ *alone* is able to achieve an F-measure of 72% while maintaining a perfect precision of 100%. To also capture record pair 1 as a duplicate, a second MD-rule, namely $\texttt{name}_{0.95}, \texttt{city}_{0.82} \rightarrow \texttt{phone}_{0.75}$, is necessary. For this reason, our duplicate detection system needs to identify good *sets* of MDs rather than a single MD. Both MDs together achieve an F-measure of 78% with still 100% precision, which is the best possible result that can be achieved with any combination of MD-rules on restaurants – the remaining duplicates cannot be described by automatically discoverable MDs. Because the set of discovered MDs might also contain many MDs that match non-duplicates, it is crucial to pick only those for duplicate detection that are appropriate *duplicate classifiers*. In this chapter, we propose a machine learning approach that learns to distinguish good from bad MD-rules based on several novel features.

More specifically, our *MDedup* duplicate detection system works in two phases: *training* and *application* (see Figure 5.1). The training phase first discovers all minimal MDs in possibly many pre-annotated datasets, i.e., ones with a gold standard of duplicates. The discovery itself is fully automatic and does not require the gold standard (Section 5.4.1). The gold standard is then used after the discovery to find the optimal subsets of discovered MDs, which are those that achieve the highest F-measure scores. With the scored MD combinations and certain characteristic features (Section 5.4.4), *MDedup* trains a regression model to predict effectiveness scores for arbitrary sets of MDs (Section 5.4.4).

The application phase then takes this general *prediction model* to predict the scores of MD combinations from a different, dirty dataset (without own gold standard). Afterwards, the best MD combination is used as a classifier for a first round of duplicate detection. Because the result usually has high precision but low recall, we propose a final *classification refinement* step, in which the high precision duplicates are used to train a binary classifier, which is a typically used Support Vector Machine (SVM) model, to find further duplicates (Section 5.5). This final classification refinement step clearly improves the system's recall in our experiments (Section 5.7.3).

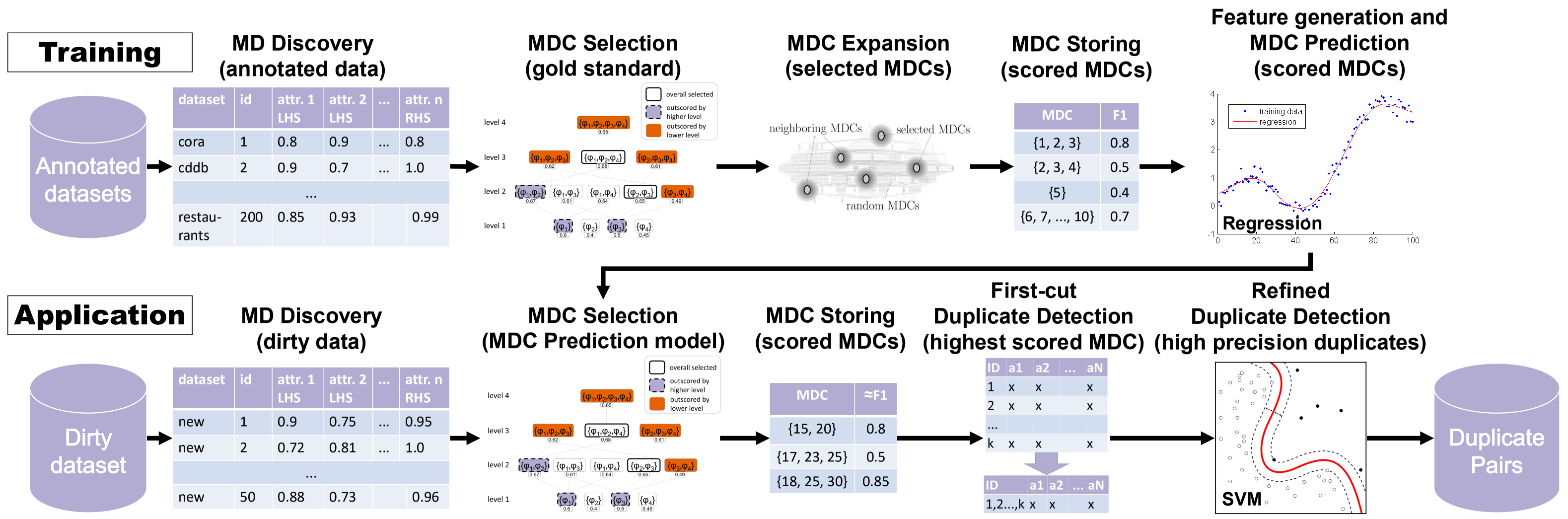
Intuitively, this chapter addresses the transfer learning problem in duplicate detection (*train on known datasets, apply on a new dataset*) with a non-transfer learning solution (*learn how to judge MDCs regardless of the domains of their datasets*) by solving the problem on a different, domain-agnostic level. We provide our MD combinations, the trained ML models, and the source code of *MDedup* online[1] to be reused in other cleaning projects.

5.2 Related work

We first discuss existing approaches for *automatic duplicate classification*, which is the focus of this research. Then, we concentrate on *matching dependencies* and their applications in data cleaning. In summary, we argue that our system is the first fully automatic, domain-agnostic system that is able to learn duplicate classification characteristics on datasets with a gold standard and use that knowledge on any other dataset.

Automatic duplicate classification. To date, no algorithm exists that *automatically* discovers duplicates without domain-specific parameterization and pre-labeled data (on the target dataset). In contrast, classification in duplicate detection is, in general, a well-researched area. Swoosh [Benjelloun et al., 2009] and Metablocking [Papadakis et al., 2013] are two example threshold-based approaches, but they are hard to parameterize without domain expertise. Similarly, rule-based systems, such as AJAX [Galhardas et al., 2000], require domain expert users to specify duplicate detection rules in a declarative way. Domain expertise is also required for unsupervised approaches, such as [Lehti and Fankhauser, 2006] and [Christen, 2008a]: In [Lehti and Fankhauser, 2006], the blocking strategy, the duplicate probability parameters, the similarity measures, and the configuration of the SVMs are all selected manually; similarly, [Christen, 2008a] needs to configure SVMs and requires thresholds to be set for the Threshold and Nearest-based

[1] `https://hpi.de/naumann/projects/repeatability/duplicate-detection/mdedup.html`


Training
Annotated datasets
MD Discovery (annotated data)
MDC Selection (gold standard)
MDC Expansion (selected MDCs)
neighboring MDCs
selected MDCs
random MDCs
MDC Storing (scored MDCs)
Feature generation and MDC Prediction (scored MDCs)
Regression
Application
Dirty dataset
MD Discovery (dirty data)
MDC Selection (MDC Prediction model)
MDC Storing (scored MDCs)
First-cut Duplicate Detection (highest scored MDC)
Refined Duplicate Detection (high precision duplicates)
SVM
Duplicate Pairs

MD: matching dependency MDC: matching dependency combination

Figure 5.1: Overview of our *MDedup* duplicate detection system; see larger figures in Figures 5.2 and 5.3.

approaches. Identically to our approach, however, both [Lehti and Fankhauser, 2006] and [Christen, 2008a] also use an SVM classifier in a final classification refinement step.

A variety of works focus on ML models using either custom features or having artificial neural networks learn their features [Christen, 2008a; Ebraheem et al., 2018; Mudgal et al., 2018]; the resulting models are always tailored to the data they were trained on, because they learn how to classify domain-specific record pairs rather than general data cleaning rules. Decision Trees [Cochinwala et al., 2001], Support Vector Machines (SVM) [Christen, 2008a], and Deep Learning [Ebraheem et al., 2018; Mudgal et al., 2018], are examples of such ML systems. They all require pre-labeled training data from the data that needs to be cleaned. All these approaches can be used in our duplicate detection system for the final classification refinement step. In our implementation, though, we have chosen an SVM approach, because it was also successfully used for classification refinement by [Lehti and Fankhauser, 2006] and [Christen, 2008a].

Matching dependency based approaches have also been used in duplicate detection [Bahmani et al., 2012, 2015; Gardezi et al., 2012]. However, they all start with some trusted, manually picked MDs and do not automatically select them – which is a major part of our contribution and more challenging to accomplish. These approaches also primarily aim at data correction rather than duplicate detection [Bahmani et al., 2012; Gardezi et al., 2012].

The Snorkel system [Ratner et al., 2017] is similar to our approach in that it also starts without pre-labeled data, but it requires some starting rules or functions whose definition, again, requires domain knowledge in the target dataset. The record linkage system of Negahban et al. [Negahban et al., 2012] is able to match records across datasets without needing example matches between these two datasets, which is similar to our setup. The system, however, assumes that the two datasets share the same domain and that gold standard matches are available that linked both datasets to same other datasets – two very restrictive assumptions that are hardly met in practice. In contrast, our system can detect duplicates without these prerequisites, but the results are not consistently as good.

Matching dependencies. Functional dependencies (FDs) are one of the most typically used types of data dependencies, also due to their capabilities in data cleaning when used in relaxed forms [Caruccio et al., 2016]. Matching dependencies (MDs) are one such relaxed form that was first introduced by Fan [Fan, 2008]. MDs extend FDs by also matching similar and not only strictly equal record values (see Section 5.3.1).

Song and Chen [Song and Chen, 2009, 2013] proposed the first discovery algorithm for MDs. The most recent algorithm for automatic MD discovery is HyMD [Schirmer et al., 2020]. To date this algorithm is the most efficient approach and we, therefore, use it in our duplicate detection system (see Section 5.4.1). However, any MD profiling algorithm can be used to serve the MDs.

Matching dependencies have been used in various works for data cleaning purposes. In [Bahmani et al., 2012; Gardezi et al., 2012], MDs are used in a query answering environment. There Minimally Resolved Instances (MRIs) define the final result set of records for a given question and they are produced by iteratively enforcing right-hand-sides of MDs in a repair step. Another approach by Bahmani et al. uses MDs for candidate blocking and to merge duplicate records [Bahmani et al., 2015]. For the actual duplicate classification, however, they use standard ML techniques, such as SVMs and K-Nearest Neighbors (KNN). In summary, all these approaches use MDs to identify duplicate candidates and to repair right-hand-side values. In our work, we focus on the detection of duplicates rather than their correction or resolution. In contrast to all MD-based previous works, we use MDs for the *classification* only and solve the problem of *selecting* proper MDs in unsupervised scenarios where *no gold standard* with labeled duplicate pairs is available.

5.3 Background and notations

In this section, we first give a more detailed definition of MDs and explain how we use them for duplicate detection (Section 5.3.1). We then introduce the concept of matching dependency combinations (Section 5.3.2).

5.3.1 Matching dependencies

Let R be a relational schema and r an instance of R. We identify the attributes of R by index, i.e., $\mathrm{R} = \langle A_1, A_2, ..., A_y \rangle$. A functional dependency (FD) on R is defined as $X \to A_i$ with $X \subseteq \mathrm{R}$ and $A_i \in \mathrm{R}$. An FD denotes that all pairs of records with same X values also have same A_i values. X and A_i are also known as LHS and RHS, respectively.

Matching dependencies, which we formally define afterward, are a relaxation of functional dependencies as they introduce three extensions: First, they relax the value comparisons, which are strictly equal (=) in FDs, by incorporating similarity metrics; this makes MDs useful for duplicate detection. Second, records can in theory be matched on different attributes, although this is not particularly useful for duplicate detection purposes. Third, they can match records across different relations, which is useful in the scenario of record linkage where the similarity join is between two relations ($\mathrm{R} \bowtie \mathrm{S}$); for the sake of simplicity and without loss of generality, we focus on single relation joins, i.e., self-joins ($\mathrm{R} \bowtie \mathrm{R}$).

The fuzzy matching of MDs is accomplished using a set of *similarity measures* ($\approx$), such as Levenshtein [Levenshtein, 1966] and Jaccard [Jaccard, 1901]. These similarity measures calculate similarities in the range of [0.0, 1.0]. To classify two values as match or non-match, MDs also specify a *decision boundary*, which is defined in [0.0, 1.0]. Similarities greater than or equal to this boundary are considered matches and lower similarities non-matches. The decision boundaries of m LHS attributes are denoted as $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ whereas the RHS has only one decision boundary ρ. Due to their strong connection, we call the combination of a similarity measure ($\approx_i$) and a decision boundary λ_i (or ρ) a *similarity classifier* $\approx_{i,\lambda_i}$ (or $\approx_{i,\rho}$). This leads us to the following definition [Schirmer et al., 2020]:

Definition 1 (Matching dependency). *Given a relational schema R with attributes $A_i, A_j \in R$, its instance r, and similarity classifiers $\approx_{i,\lambda_i}$ and $\approx_{i,\rho}$, a matching dependency (MD) φ is defined as follows:*

$$\forall r_s, r_t \in r : \left(\bigwedge_{i=1}^{w-1} r_s[A_i] \approx_{i,\lambda_i} r_t[A_i] \right) \to r_s[A_w] \approx_{w,\rho} r_t[A_w]$$

In other words, a matching dependency states that if two records r_s and r_t match in all their A_i values (attribute-specific similarity calculated by $\approx_i$ greater than or equal to

λ_i) then their A_w values need to be at least ρ-similar w.r.t. $\approx_w$. Note that in the broader definition of MDs the two A_i attributes and the two A_w attributes can be different attributes and even attributes from different relations, in which case we would have (A_{i_s}, A_{i_t}) and (A_{w_s}, A_{w_t}).

For practical reasons and because there is usually only one reasonable similarity measure per attribute, we usually use the following short notation to specify MDs:

$$\left(\bigwedge_{i=1}^{w-1} A_{i,\lambda_i}\right) \rightarrow A_{w,\rho}$$

The MD $\texttt{address}_{0.7}, \texttt{name}_{0.7}, \texttt{type}_{0.71} \rightarrow \texttt{phone}_{1.0}$, which is a true MD in the restaurants dataset (see example records in Table 5.1), follows this short notation. If for two records all LHS similarities match, they match the RHS similarity.

To use an MD for *duplicate detection*, we simply consider its LHS as a classifier: All record pairs that match the MD's LHS are labeled as *duplicate*. Intuitively, the LHS is the matching rule that we are looking for, and the presence of a valid RHS is an indicator (for rule discovery and scoring) that the LHS is relevant. For the majority of MDs, this inference leads to poor results. The MD $\texttt{name}_{0.0} \rightarrow \texttt{phone}_{0.0}$, for instance, is true on any instance of the restaurants dataset and it matches *all* record pairs. Hence, the challenge is to identify such MDs that are *useful* duplicate classifiers. The rule $\texttt{address}_{0.7}, \texttt{name}_{0.7}, \texttt{type}_{0.71} \rightarrow \texttt{phone}_{1.0}$, for example, yields an F-measure of 72%, which is relatively good. We discuss indicators that hint towards useful MDs for duplicate classification in Section 5.4.4.

5.3.2 Matching dependency combinations

As we illustrated in Section 5.1, one MD might not be able to capture all duplicates. Our approach, therefore, considers multiple MDs for the classification. We refer to these sets of MDs as matching dependency combinations:

Definition 2 (Matching dependency combination)**.** *Given the set of all MDs Φ, a matching dependency combination (MDC) χ is any selection of MDs with $\chi \subseteq \Phi$.*

An MDC χ can be used as a *duplicate classifier* by considering a record pair as duplicate, iff it matches the LHS of at least one MD $\varphi \in \chi$. Technically, an MDC-based duplicate classifier combines a set of MD-based duplicate classifiers via logical *or*-operations. In this way, different MDs can be used to classify different kinds of duplicates. The goal of MD-based duplicate detection in general is therefore to predict the best MDC for the duplicate classification step.

To illustrate, consider again our running example of Table 5.1, which offers 5 MDs. Our MDC selection strategy (see Section 5.4.2) then carefully selects 17 MDCs in total from the $2^5 - 1 = 31$ possible combinations. One of these MDCs consists of the two MDs $\texttt{name}_{0.7}, \texttt{address}_{0.7} \rightarrow \texttt{phone}_{0.75}$ and $\texttt{name}_{0.95}, \texttt{city}_{0.82} \rightarrow \texttt{phone}_{0.75}$, which offer the best possible F-measure of 78% (and a precision of 100%).

5.4 MDedup training

The MDedup process is comprised of two phases, namely training and application. We explain the training phase now and the application phase in Section 5.5. As shown in the MDedup overview of Figure 5.1, the training phase takes several datasets and their labeled duplicates as input and, then, runs five basic steps: At first, the pipeline *discovers* all minimal MDs (Section 5.4.1). Using our MDC selection algorithm, it then *selects* MDCs that produce possibly high quality results when used as duplicate classifiers (Section 5.4.2). In the third step, this set of high quality MDCs is enriched with further MDCs of varying quality via systematic *expansion* (Section 5.4.3). Afterwards, MDedup's training pipeline *stores* all scored MDCs in one training set that integrates the results calculated on different annotated datasets. The fifth and last step of the pipeline *generates* a set of features for the scored MDCs to then use both the MDCs and their features to *train* a machine learning (ML) model on how to *predict* the F-measure of a given MDC without a gold standard of duplicate pairs (Section 5.4.4).

5.4.1 MD discovery

The main assets of our system are the MDs that we use for duplicate classification, because they prescribe the optimal recall and precision that we can achieve. It is therefore essential to discover many and good MDs – in our implementation of MDedup, we discover *all minimal, non-trivial* MDs. Although any set of MDs can be used as input for our duplicate detection system, the complete set of minimal and non-trivial MDs is promising, because these MDs are by definition very close to core data patterns; it is also what most existing dependency profiling algorithms discover.

An MD φ is *minimal* if no other MD φ' with same LHS and RHS exists, such that only one LHS threshold is smaller (i.e., more general) or the RHS threshold is larger (i.e., more restrictive); given a valid MD, raising LHS thresholds or decreasing the RHS threshold will always generate a valid MD. Additionally, an MD φ is *trivial* if its RHS attribute is contained in its LHS attributes with either the same or a higher similarity threshold; if the LHS matches only those records with at least similarity x in attribute A_i, then these records are trivially also at least x (or less) similar in A_i.

MD profiling algorithms, such as those published by Song and Chen [Song and Chen, 2009, 2013], can effectively serve our pipeline with MDs. The implementation of MDedup that we developed for this chapter, however, uses our own algorithm *HyMD* [Schirmer et al., 2020] — the most efficient MD algorithm to date. It discovers all minimal, non-trivial MDs using several indexes to identify similar records, along with a number of pruning rules that effectively tame the exponentially large search space.

5.4.2 MDC selection

The goal of MDedup's matching dependency combination (MDC) selection component is to find MDCs that produce possibly high F-measure scores when used as duplicate classifiers. Identifying which combinations of MDs are the best is a challenging task, because the complexity of checking all possible combinations of a set of m discovered MDs is exponential. More specifically, there are $\sum_{k=1}^{m} \binom{m}{k} = 2^m$ candidates to be evaluated. The problem is particularly hard to solve, because our system often deals with thousands of MDs. For this reason, we propose a *greedy selection algorithm* that systematically searches the combination space for MDCs with high F-measures. The greediness of the algorithm decides which combinations are worth being further investigated, pruning all those MDCs on the way that offer sub-optimal F-measure scores. Because optimizing for F-measure is a non-convex problem [Nan et al., 2012] and we propose a linear, bottom-up greedy search, our approach does not guarantee optimality. It, however, always found the optimal combination in our experiments.

Before we discuss our lattice traversal-based search process, we first define certain preliminaries. To keep the search within reasonable time constraints, but at the same time consider a variety of possible solutions, we define a parameter k, which controls the fan-out of the process. In each level of the lattice, a maximum of *top k* MDCs produce further candidate MDCs to be considered in the next higher level of the lattice, with each candidate MDC being one MD larger than its predecessor. To consider more variations as we go up the lattice levels, we employ another trick. We combine every candidate with every MD of the first level, instead of just combining the top k candidates of each level with each other, which would produce a narrower fan-out. This grows the search space linearly, but at the same time allows us to consider some variation in our candidates. Nevertheless, the main focus is kept on the combinations that produce good scores. The process follows the beam search strategy and could also be characterized as "hill climbing with top-k". Although this approach is heuristic, an exhaustive, i.e., complete candidate testing approach produced the same results on smaller datasets in our experiments. We label the greedily selected MDCs as "selected MDCs" and define them as follows:

Definition 3 (Selected MDC)**.** *An MDC χ is defined as "selected" if $\exists$ MD φ where $\chi - \{\varphi\}$ reduces its score and $\nexists$ φ' where $\chi \cup \{\varphi'\}$ improves its score.*

Based on this definition, Algorithm 1 solves MDedup's *selection* process. The parameters of this process are the *HyMD* algorithm that provides all minimal, non-trivial MDs, the ORACLE algorithm that calculates the F-measure score given an MDC, and the search scope k of the greedy approach. For the training phase, the ORACLE algorithm uses the given duplicate gold standards; the calculations follow the principles that we described in Section 5.3.2. In the application phase, the ORACLE algorithm uses the trained MDC scoring model.

The MDC selection algorithm starts by calling HyMD to discover the MDs for this dataset (line 2). Then, it scores the initial MDCs, where every MDC contains exactly one MD (line 3). These MDCs are shown in Figure 5.2 at level 1. Subsequently, it filters out MDs with a score of zero to reduce the search space (line 4). Next, two sets are initialized (lines 5 and 6): the best MDCs for the currently examined level (*levelSelected*) and the overall best MDCs (*overallSelected*). None of the overall best MDCs may be outscored by another MDC of the *levelSelected* set. The algorithm then starts to iterate all search space levels from level one upwards as long as there are more non-outscored MDCs in the current level (line 7). For each level, the algorithm creates the respective candidate MDCs, based on the MDCs of the previous level (lines 8 to 11). For each generated set, the algorithm iterates over all its candidates, calculates their score, and if that score is higher than the scores of the MDCs that created them, they are added to the *levelSelected* set (lines 12 to 17). It proceeds by keeping only the k best MDCs (line 18), while these k MDCs (*levelSelected*) are added to the *overallSelected* set (line 19). Consequently, it removes the creator MDCs of the selected k from the *overallSelected* set (lines 20 and 21). We do not remove MDCs that create higher scored MDCs if these higher scored MDCs are not selected in the top k of their level to maintain a higher diversity in the final set. Finally, the *overallSelected* set containing all the best, non-outscored MDCs across all levels is returned (line 22).

To better understand the selection process, consider the example in Figure 5.2 with

Algorithm 1: MDC selection

```
1  function MDC_selection($HyMD$, ORACLE, $k$)
       /* Discover, score, and filter MDs */
2      $mds \leftarrow HyMD.discoverMDs()$
3      $mds \leftarrow$ ORACLE.$score(mds)$
4      $mds \leftarrow mds.filter(lambda\ md : md.score > 0)$
       /* Initialize MD combinations sets */
5      $levelSelected \leftarrow topk(mds, k)$
       /* Copy */
6      $overallSelected \leftarrow \{levelSelected\}$
7      while $|levelSelected| > 0$ do
           /* Combine with single MDs */
8          $candidates \leftarrow \{\varnothing\}$
9          for $mdc \in levelSelected$ do
10             for $md \in mds \setminus mdc$ do
11                 $candidates.add(mdc \cup md)$
           /* Score MDCs and keep those with improved score */
12         $levelSelected \leftarrow \{\varnothing\}$
13         for $mdc \in candidates$ do
14             $mdc.score =$ ORACLE.$score(mdc)$
15             $creatorsScore = max(mdc.creators.scores)$
16             if $mdc.score > creatorsScore$ then
17                 $levelSelected.add(mdc)$
           /* Filter and save topk, and remove their creators */
18         $levelSelected \leftarrow topk(levelSelected, k)$
19         $overallSelected.addAll(levelSelected)$
20         for $mdc \in levelSelected$ do
21             $overallSelected.removeAll(mdc.creators)$
22     return $overallSelected$
```

$k = 2$. It shows a lattice of MDCs with the top MDC describing the combination of all MDs and the bottom holding all single-MD MDCs. The execution begins at the bottom by considering the single-MD MDCs, which are given by the MD discovery algorithm (see Section 5.4.1). As explained previously, we consider only the top k candidates with highest F-measure scores to reduce the search space. After selecting the top k (*levelSelected*), which are $\{\varphi_1\}$ and $\{\varphi_3\}$, from the first level, the algorithm combines them with every other possible MD to produce the candidates for the next level. MDCs that generate candidates for the next level are their *creators*; thus, MDCs $\{\varphi_1\}$ and $\{\varphi_3\}$ are creators of level 2's candidate MDCs. Moving to level 2, notice that the combination $\{\varphi_2, \varphi_4\}$ is not considered, as neither $\{\varphi_2\}$ nor $\{\varphi_4\}$ were selected in the previous *levelSelected*. By calculating the MDC scores, we find that all MDCs of size 2 have a better score than their creators, i.e., $\{\varphi_1\}$ and $\{\varphi_3\}$, except for one, which is the MDC $\{\varphi_3, \varphi_4\}$ with a worse score than $\{\varphi_3\}$; thus it is eliminated and not considered for the *levelSelected* selection. Because $\{\varphi_1\}$ and $\{\varphi_3\}$ created descendants with better scores, they are removed

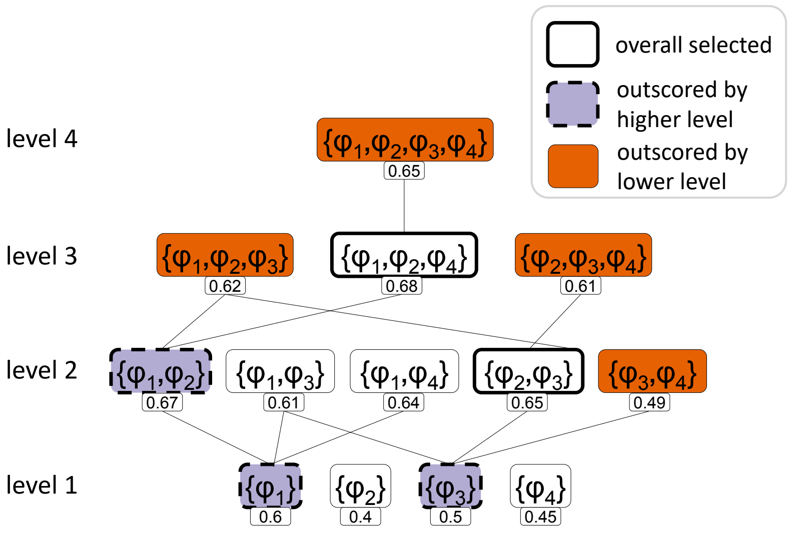


Figure 5.2: An example for the selection algorithm on an MDC lattice with four input MDs and $k = 2$. The finally selected MDCs are $\{\varphi_1, \varphi_2, \varphi_4\}$ and $\{\varphi_2, \varphi_3\}$.

from the final result set (*overallSelected*). The same process now repeats at level 2: We calculate the scores for each MDC and keep the top k, which are $\{\varphi_1, \varphi_2\}$ and $\{\varphi_2, \varphi_3\}$. Proceeding to level 3, we find only one MDC, which is $\{\varphi_1, \varphi_2, \varphi_4\}$, that is not outscored by its creators. In the end, we select $\{\varphi_1, \varphi_2, \varphi_4\}$ and $\{\varphi_2, \varphi_3\}$ in *overallSelected* as final result.

5.4.3 MDC expansion

The goal of MDedup's training phase is to create a model that can predict F-measure scores for MDCs. While most discovered MDCs in our experiments have a low F-measure score, the MDC selection step finds a set of k MDCs with high scores. This set is, therefore, biased towards top performing MDCs and it is rather small. Most ML models, however, behave better if their training data is more diverse and they are given more data [Banko and Brill, 2001]. So, to ensure that enough training instances with a large variety in their properties exist, we propose an MDC *expansion* process that consists of two strategies: neighbor expansion and random sampling.

Given the top k MDCs from the selection step, the *neighbor expansion* strategy considers neighboring MDCs, in terms of their lattice position, around these selected MDCs. This is achieved by an iterative process where we randomly select an input MDC and then *add*, *remove*, or *replace* some MDs using a Gaussian distribution, thus ensuring a focus on neighboring MDCs. Figure 5.3 visualizes this concept: Rectangular nodes represent the initially selected MDCs and their shaded neighborhoods show the Gaussian distribution on the lattice from which the additional MDCs are generated. This Gaussian distribution spans towards the bottom (MDCs with less MDs), the top (MDCs with more MDs), as well as left and right (same size MDCs, but with different MDs). By taking neighboring MDCs of top performing MDCs into the training set, we enable a machine learning algorithm to learn the precise characteristics of why certain MDCs perform well. The result of this strategy therefore produces a larger set of MDCs that describes the few top performing MDCs well.

Because the neighbor expansion strategy is still biased around top scored MDCs, the *random sampling* strategy injects MDCs of larger variety: It randomly creates (and scores) MDCs by considering the entire search space up to the maximum level reached by the MDC selection, disregarding selected and neighboring MDCs. In Figure 5.3's visualization, the random MDCs are scattered all over the surface. We control the overall size of the expansion set (relative to the number of selected MDCs), equally for both strategies, with the parameter *expansion_factor*.

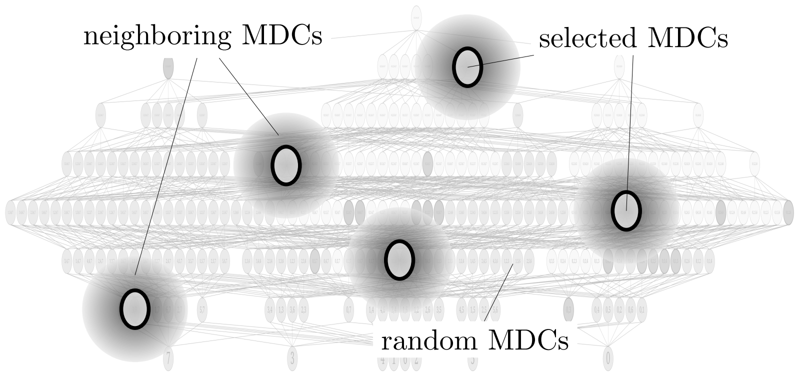


Figure 5.3: A visualization for two expansion strategies: neighbor expansion and random sampling.

5.4.4 Feature generation and MDC prediction

In this section, we describe the machine learning model that we train to predict F-measure scores for MDCs. MDedup needs this model to score the MDCs in the application phase where no gold standard is available to calculate the exact F-measure values (see Section 5.5). To build the ML model, we need to specify a set of features and the prediction strategy. For the features, we collected thirty-five metrics of MDC characteristics that are possibly relevant for the duplicate classification performance of the MDC instances, and discuss them next. Afterward, we propose a Gaussian regression process for the prediction strategy.

Assembling features for MDCs

To predict the F-measure an MDC would achieve when used as a duplicate classifier, we determine several features describing the MDC itself and the data for which it holds. These features need to be available on any relational input dataset and do not need a gold standard.

In total, we define thirty-five features, which are outlined in the taxonomy presented in Figure 5.4. Based on this taxonomy, we have the following main categories: instance statistics, match statistics, and MDC statistics. First, the *instance statistics* describe basic statistics of the relational input dataset, such as the completeness or uniqueness of attributes w.r.t. all attributes used in an MDC. Second, the *match statistics* describe how well an MDC is supported by the data. Finally, the *mdc statistics* describe the structure of an MDC and its MDs. In particular, these are metrics based on MDC characteristics, such as their size, and the similarity measures, attributes, and thresholds that are used

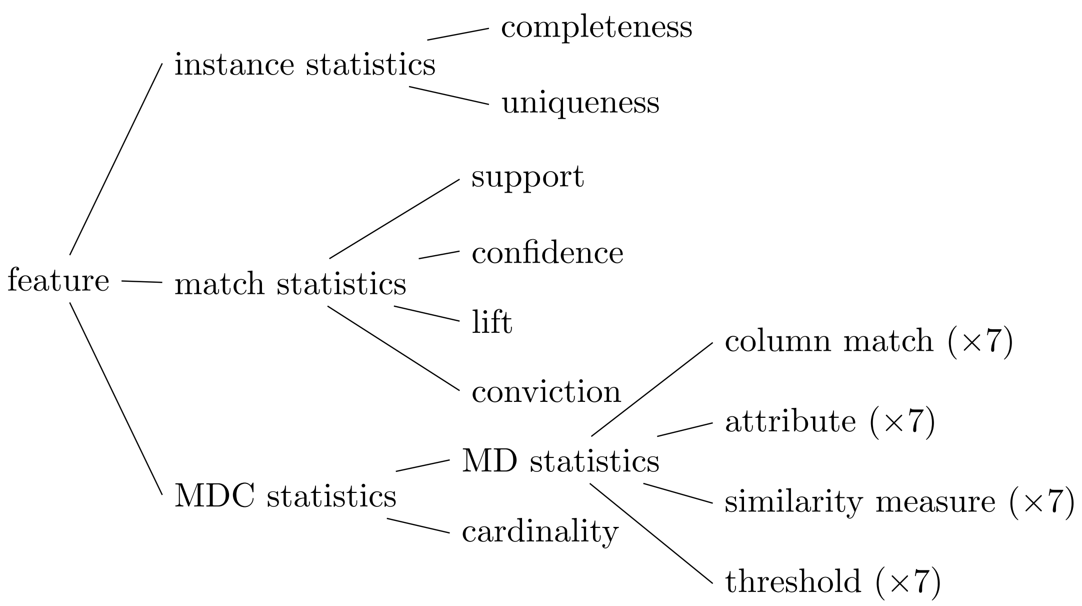


Figure 5.4: Taxonomy of the features based on the data required for their calculation.

in the MDs' LHSs and RHSs. Having explained the main feature categories, we now describe the individual features.

Completeness. Completeness is the percentage of records with no null values in the attributes of the MDC. Null values are not useful for record matching and, therefore, this feature can help to disregard MDCs that match on sparse or empty data. With '$\perp$' representing null values and $attr(\chi)$ the union of all attributes used by the MDC's MDs, the completeness feature is calculated as follows:

$$compl(\chi) = \frac{|\{r_i \in r \mid \forall A_j \in attr(\chi) \; : \; r_i[A_j] \neq \perp\}|}{|r|}$$

Uniqueness. The uniqueness of an MDC is the fraction of unique value combinations in the MDC's attributes, relative to the overall number of records. A high uniqueness is a good indication that the given set of attributes is suitable as a key, in that it can uniquely identify a record when comparing it with others. If we represent the projection of the relational instance r to all the attributes of the MDC χ as $r[attr(\chi)]$, the uniqueness of χ is defined as follows:

$$uniq(\chi) = \frac{|\{r_i \in r[attr(\chi)]\}|}{|r|}$$

Support. The support of an MDC is the percentage of record pairs that match at least one MD of the MDC on the LHS ($suppLHS(\chi)$), the RHS ($suppRHS(\chi)$), or both sides ($supp(\chi)$). In this way, the support features encode how strong the MDC is and what duplicate detection recall we can expect. For exact MDs, which are all MDs that we discover, $suppLHS(\chi) = supp(\chi)$. For definition purposes, let $p = r \times r$ be the set of all record pairs and $match(p_i, X)$ a function that returns true if the record pair p_i

matches the match conditions X, i.e., it matches $\bigwedge_{i=1}^{w-1} r_s[A_i] \approx_{i,\lambda_i} r_t[A_i]$ for LHSs and $r_s[A_w] \approx_{w,\rho} r_t[A_w]$ for RHSs. Then the support features are defined by the following three formulas:

$$supp(\chi) = \frac{|\{p_i \in p \mid \exists \varphi \in \chi : match(p_i, \varphi.LHS \cup \varphi.RHS)\}|}{|p|}$$

$$suppLHS(\chi) = \frac{|\{p_i \in p \mid \exists \varphi \in \chi : match(p_i, \varphi.LHS)\}|}{|p|}$$

$$suppRHS(\chi) = \frac{|\{p_i \in p \mid \exists \varphi \in \chi : match(p_i, \varphi.RHS)\}|}{|p|}$$

The formulas are a translation of $supp(X) = \frac{|\{t \in T; X \subseteq t|}{|T|}$ [Hastie et al., 2009]. Caching these scores after their calculation allows us to re-use them for the calculation of the following metrics.

Confidence. The confidence feature describes the portion of the data that satisfies at least one MD of the MDC. If all MDs are correct, i.e., no approximation was used during MD discovery, the confidence is always 100%; otherwise, the feature allows the algorithm to trust correct MDCs more than partially violated MDCs. Using our exact MD discovery algorithm, the confidence measure is basically irrelevant. However, it might be useful in alternative setups. Hence, the feature definition is as follows:

$$conf(\chi) = \frac{supp(\chi)}{suppLHS(\chi)}$$

Lift. The lift feature is a measure for correlation and represents the ratio of record pairs where both LHS and RHS are satisfied, divided by the product of independent percentages for LHS and RHS. A number larger than 1 indicates that the two sides tend to co-occur, meaning that if one appears the other will appear as well. On the other hand, a number smaller than one means the opposite. Similarly to confidence, the use of an exact MD discovery algorithm constrains the calculated number for lift, to values larger than or equal to 1. The calculation uses the following formula:

$$lift(\chi) = \frac{supp(\chi)}{suppLHS(\chi) \cdot suppRHS(\chi)}$$

Conviction. Conviction is a metric that can be interpreted as the fraction of the expected frequency that a LHS co-occurs with its RHS. Conviction values higher than 1 indicate to what extent the associations of the MDs' LHSs and RHSs are random. This should help the model to prefer meaningful dependencies over spurious ones. Since the use of an exact MD discovery algorithm constrains the value of confidence to 1, the denominator is 0 and the whole value becomes undefined. However, similarly to confidence, it might be useful in alternative setups. The feature is defined as follows:

$$conv(\chi) = \frac{1 - suppRHS(\chi)}{1 - conf(\chi)}$$

MD statistics. The MD statistics are set of features that describe significant characteristics of a given MDC by considering only its definition and not the data is was discovered from. The features use the following elements of the MDs: attributes, similarity measures, and thresholds, as well as their column matches, which is the composition of the previous three, i.e., A_i, $\approx_i$, and λ_i. Each MD in an MDC offers a set of these elements. We define a feature for each of these four element sets by calculating the following set operations:

- *Intersection size*: The number of elements that exist among all MDs (1).
- *Union size*: The number of elements that exist in at least one MD (2).
- *Jaccard distance*: The distance between common attributes and specific attributes, which is calculated by dividing the intersection size with the union size; as a result, we get a normalized value in the range of $[0.0, 1.0]$ (3).
- *Descriptive statistic*: Significant and extreme elements over all sets, calculated with the basic metrics of min (4), median (5), max (6), and stdev (7).

Cardinality. When increasing the size of an MDC by adding more MDs, its recall increases monotonically and its precision decreases monotonically. The cardinality of an MDC, which is simply defined as $card(\chi) = |\chi|$, is therefore an important feature for estimating the MDC's F-measure. Small MDCs would be preferable for human readability, whereas large MDCs can cover more special cases.

Regression for MDC score prediction

Using the feature definitions discussed above, we can now generate an instance set of scored MDCs with their MDC-specific features. With this instance set, we can train a machine learning model to predict the F-measure score for arbitrary MDs and their datasets. Although different types of models could be considered for this task, *regression models* seem to be a natural fit. They learn to estimate the relationship between a feature set that for us is MDC-specific, and a target variable, which in this case is the F-measure. Although many different regression models exist, we propose a Gaussian Process [Hastie et al., 2009] for two main reasons: First, it is capable of learning complex data patterns, which is necessary considering the many factors that influence the performance of an MD as a duplicate classifier. Second, it has a relatively small set of parameters, which is important, because our MDedup target system should be a largely automatic system and a small parameter space simplifies the hyperparameter tuning phase. We describe the configuration in further detail in Section 5.6.2.

5.5 MDedup application

This section describes the application phase of the MDedup duplicate detection system. This phase is shown as the lower pipeline in Figure 5.1 and starts with a dirty dataset for which no gold standard records exist. Instead of using pre-labeled data for the scoring of MDCs, the application phase takes the regression model that was constructed on other datasets in the training phase and is able to predict F-measure scores for MDCs w.r.t. their relational instance data (see Section 5.4). Because most of the application pipeline uses the same components as the training pipeline, in the following descriptions we focus on the differences.

At first, the application pipeline *discovers* all minimal MDs on the dirty input dataset with the same algorithm as for the training phase (Section 5.4.1). In the next step, the pipeline uses our MDC selection algorithm (Section 5.4.2) to greedily *select* the top k MDCs with the highest F-measure scores. Instead of scoring each MDC using the gold standard, the selection algorithm in the application phase uses the regression model provided by the training phase (Section 5.4.4). The results of the selection are then passed to the MDC store were MDedup *picks* only the highest scored MDC as the duplicate classifier. The next step of the application pipeline then *detects* all record pairs that this MDC classifier matches as duplicates. As a final step, MDedup uses the discovered duplicate pairs to *boost* the overall F-measure of the process: With the known duplicates, the pipeline trains a state-of-the-art SVM model to classify record pairs as duplicates or non-duplicates; this model is applied to the input data to refine and complement the duplicates.

MDC selection. A benefit of using a regression model for the MDC scoring is that we can easily convert our MDC selection approach for the application phase without any gold standard. Looking back at Algorithm 1, the score of every MDC is given by the ORACLE algorithm. In the absence of a gold standard, this ORACLE now needs to use the regression model for the F-measure calculation.

To predict the F-measure for an MDC, the ORACLE first generates a feature vector with the features discussed in Section 5.4.4. This feature vector is then provided to the regression model, which returns an estimate for the F-measure. The lattice traversal and top-k selection strategy in the application phase is identical to the training phase. The result of the MDC selection step is a list of MDC associated with their predicted F-measure scores.

First-cut duplicate detection. Given the highest scored MDC of the selection step, MDedup executes its first duplicate detection pass. This pass involves the three commonly applied steps of candidate indexing, candidate matching, and candidate classification. In the following, we briefly describe our implementations for these steps.

The *candidate indexing* method is responsible for the selection of record pairs that should be tested. The duplicate detection could, in theory, be applied to all pairs of records in the input relation, but this large number of candidates is, in practice, usually reduced by blocking methods [Christen, 2012b; Elmagarmid et al., 2007] that divide the quadratic candidate space into smaller subsets (i.e., blocks) according to a predefined partitioning key. To make sure that similar records fall into at least one common block and, hence, can get matched, MDedup takes every attribute as a partitioning key once; records with same values in that attribute are put into the same block. For large values, i.e., ones that are comprised of more than four words, the algorithm extracts word n-grams, with $n = 4$, from the value where each word n-gram defines a bucket; records with long values are, in this way, placed into multiple buckets. The result of indexing are several overlapping blocks of records.

For each block, the *candidate matching* step creates all pairs of records within the block. Because the blocks share large overlaps, MDedup places all pairs from all blocks into a common set that removes redundant pairs, reducing the matching effort significantly. All remaining candidate pairs are then compared by calculating their attribute-wise similarities, for all attributes and similarities specified in MDs. We finally store the candidate set to use it not only for this first-cut duplicate detection process, but also for the refined duplicate detection step later on.

For the *candidate classification*, MDedup applies the highest scored MDC as a duplicate classifier to every record pair, as described in Section 5.3.2. The result are two sets, a set of duplicates and a set of non-duplicates.

Refined duplicate detection. The evaluation of the first-cut duplicate detection results (in Section 5.7.2) shows that the discovered duplicate sets have high precision but rather low recall. For this reason, we propose a final *classification refinement* step to expand the result set.

Classification refinement is used to sequentially connect learning models in a way that each model can enhance the prediction of the previous models and reduce the overall bias and variance of the results. Because the first-cut duplicate detection pass provided us with a labeled set of record pairs and their similarities, we can use those to train a typical binary classification model. The overall rationale of this classification refinement approach is that MDCs identify duplicates based on very specific MD rules that cannot capture all kinds of similarity patterns. SVMs, in contrast, learn schema-specific similarity patterns and are, therefore, able to match records more precisely. Combining both approaches results in an overall schema-independent but still flexible classification approach.

Although many complex classification refinement approaches have been proposed, such as boosting, with AdaBoost [Collins et al., 2002] being a popular algorithm of this category, that combine multiple models on an instance level, we propose a simple Support Vector Machine (SVM) approach, because SVMs have already proven to be effective as a duplicate detection classification refinement step [Christen, 2008a; Lehti and Fankhauser, 2006]. Since training a binary classification model requires labels of both classes, the application pipeline takes the entire set of *duplicates* and a subset of the *non-duplicates* that is ten times larger than the duplicates set. With the record pairs, their similarities and duplicate labels, we finally train our SVM model. After training, we apply the model as a duplicate classifier to our candidate set, generated in the first-cut duplicate detection, to retrieve the final duplicates. We provide more details on the configuration of the classification refinement step in Section 5.6.2.

5.6 Evaluation setup

In this section, we present our evaluation setup of the experiments described in Section 5.7. We begin in Section 5.6.1 by describing our experimental setup and the data that we used for training and testing. Section 5.6.2 then proposes robust configuration parameters for MDedup based on several smaller experiments.

5.6.1 Preliminaries

In this section, we first introduce our experimental setup. Then, we discuss our datasets and how we complement their gold standards of duplicates with challenging non-duplicates.

Experimental configuration. All experiments were conducted on a machine with 4x Intel Xeon E7-8837 (2.67GHz, Octa-Core) and 256GB of RAM. All 16 hyperthreaded cores were used for parallelization. The system uses Ubuntu 18.04 LTS and Oracle Java 1.8. The source code, details and download information of the used datasets, and evaluation results are available on our website[2].

[2]`https://hpi.de/naumann/projects/repeatability/duplicate-detection/mdedup.html`

Datasets. To train and test our MDedup system, we use eight datasets from heterogeneous domains and their known gold standards, commonly used for evaluation purposes in the topic of duplicate detection by papers such as [Bilenko and Mooney, 2003; Mudgal et al., 2018; Vatsalan et al., 2013]: Amazon-Walmart (products), CDDB (audio), Census (census), Cora (bibliography), DBLP-Scholar (bibliography), Hotels (lodging), NCVoters (voter registrations), and Restaurants (restaurants). Table 5.2 lists the datasets' number of records, duplicates (DPL) and non-duplicates (NDPL). For all datasets, we select only those attributes for the MD discovery that have a completeness >10%, i.e., at least 10% of the values are non-null, because sparse columns make the discovery more difficult and are not useful for duplicate classification. We also exclude IDs, because they often encode the gold standard and, hence, make the task trivial. Only for NCVoters, we applied a stricter attribute filtering due to the schema size of that dataset. For more details and information about which attributes were used exactly, we refer to our website.

Table 5.2: Dataset statistics: number of records, attributes, duplicates (DPL), non-duplicates (NDPL), matching dependencies (MDs), and matching dependency combinations (MDCs).

Dataset	Records	Attr.	DPL	NDPL	MDs	MDCs	
						Selection	Prediction
Census	841	5	376	3,760	15	31	31
Restaurants	864	5	112	1,120	5	17	9
DBLP-Scholar	66,879	5	5,347	53,470	5	1	10
CDDB	9,763	6	300	3,000	11	37	11
Hotels	364,965	10	94,677	368,002	159	144	48
Cora	1,879	13	64,578	268,082	74	288	80
Amazon-Walmart	24,583	13	1,154	11,540	132,732	96	17
NCVoters	14,183	25	9,819	98,142	149,950	400	60

Table 5.3: Dataset statistics: execution times across phases.

Dataset	Execution time (hh:mm:ss)		
	HyMD	Selection	Prediction
Census	00:00:07	00:00:01	00:00:19
Restaurants	00:00:07	00:00:10	00:00:39
DBLP-Scholar	04:54:15	00:00:03	00:00:21
CDDB	04:21:44	00:00:01	00:00:15
Hotels	03:10:43	00:00:42	03:29:30
Cora	00:01:18	00:00:19	00:11:42
Amazon-Walmart	14:03:42	00:00:06	01:10:11
NCVoters	58:21:14	00:04:46	07:21:19

Gold standards. The evaluation datasets mentioned previously are accompanied by gold standards that contain lists of record pairs that uniquely identify *duplicates.* In case these lists are not transitively closed (CDDB, Census, and Hotels), we add all transitive pairs as duplicates to the gold standard in a pre-processing step.

To properly score and evaluate the MDCs during the training and testing phases produced by the steps of selection, expansion, and prediction, as well as the refined classification, we also require a number of negative examples, i.e., non-duplicate pairs.

When creating these pairs, we made sure that records are not paired up randomly, because most random pairs have a very low similarity and are, therefore, trivial to classify. More useful for training and more realistic for testing are non-duplicate pairs that are similar and, hence, harder to classify. To this end, following the same principles in as Chapters 3 and 4, we apply blocking as described in Section 5.5 on the input relation and pick non-duplicate record pairs from within the blocks; in this way, the non-duplicates are guaranteed to be similar in at least the blocking key. When selecting non-duplicates, we also enforce a ratio of 1:10 (DPLs to NDPLs) to not over-represent the class of non-duplicates. For Cora and Hotels, we took all non-duplicates form the blocking, which resulted in about the ratio 1:4 for both datasets.

5.6.2 MDedup configuration

In this section, we summarize MDedup's configuration per pipeline step and propose robust default values so that the system can be deployed without parameter tuning.

MD Discovery. For MD discovery, we use the HyMD algorithm and its default parameterization in both the training and application phase [Schirmer et al., 2020]. Regarding the similarity measures, we consider all attributes as alphanumeric and use the following principles: In general, all attributes are compared with the Levenshtein similarity measure [Levenshtein, 1966]; attributes with a mean value length of more than 100 characters, however, use the Jaccard similarity measure [Jaccard, 1901] that compares tokens, produced by splitting the value with whitespaces, instead of characters. Jaccard is beneficial for attributes with long values, because comparing long values on character basis is slow and, in general, in-effective: They often contain descriptions, comments, and other free-text fields that are less structured and words can be on different positions across two duplicate records.

For both measures, we set the minimum similarity threshold for discovered MDs to 0.7 – lower thresholds have not shown better results in our experiments, but the discovery time increases significantly with decreasing thresholds.

Tables 5.2 and 5.3 contain additional information about the execution of HyMD on our evaluation datasets. The datasets Amazon-Walmart and NCVoters produce a large number of MDs and, hence, also require the longest execution times.

Table 5.4: Impact of k on the average F-measure across all evaluated datasets.

k	1	2	4	8	16	32	64
F-measure	34.1%	39.5%	32.5%	47.3%	55.2%	55.4%	55.4%

MDC Selection. The MDC selection steps in both the training and application phase require the parameter k. This parameter specifies the number of MDCs with the highest score that are selected in each level of the lattice and passed to the following; it is also the number of MDCs that are returned in the end. To investigate the impact of k on MDedup's performance, we trained the system with different k values and measured the average F-measure scores across all datasets presented in Table 5.4. From this, we conclude that a certain set size is needed to learn how useful MDs look like, but at some point the set captures all relevant information and larger training sets do not improve the performance. Hence, MDedup is robust against large k value, i.e., overly large values impact its runtime performance but not its effectiveness. In our experiments, we set the parameter to 16.

Because the MDC selection step has exponential complexity in the number of MDs, its execution time can be very high. For this reason, we introduced a *maximum execution time* that, when exceeded, triggers MDedup to gracefully stop the lattice traversal: the current level is finished but the next level is not started. In our experiments, we set a maximum execution time of ten hours. Hence, this time limit is essentially irrelevant for the training phase, but the feature generation and predictions in the application phase are so expensive that our two largest datasets exceed a selection time of 10 hours. Tables 5.2 and 5.3 list the numbers of selected MDCs together with the actual selection times.

To evaluate the effectiveness of the thirty-five MDC score prediction features, we performed two experiments: The first experiment is a leave-one-out experiment, where we trained the system leaving out each feature once to see how this feature impacts the F-measure. In a second experiment, we trained the system once with every feature exclusively, i.e., with only that one feature to measure its performance. With the results of these two experiments, we could not identify a particularly important or harmful feature, because every feature increases the performance on at least one dataset and/or performs well on its own. For this reason, we use all proposed features in our system.

MDC Expansion. The parameter *expansion_factor* ensures that enough MDCs exist to capture the datasets' properties, especially for datasets with an inherently small number of MDCs. It also serves to represent some non-optimal MDCs, i.e., negative examples in the training set. Both the neighbor expansion and the random sampling are configured to enlarge the set of selected MDCs by a factor of 10 each. To evaluate the impact of the expansion, we trained MDedup once with an expansion factor of 0.0, which is expansion switched off, and once with a factor of 10. Without expansion, the F-measure for DLBP-Scholar increased from 0% to 83% (the one effective MD was selected now), but this effect appears to be incidental, because the F-measure decreases for all other datasets without expansion – on average by 2%. For this reason, we propose to use the expansion step. A default factor of 10 worked well in our experiments, because the F-measure did not improve for larger values.

MDC Prediction. The *Gaussian Process*, which is our regression model for predicting F-measures, requires a kernel and a regularization λ as parameters. For the kernel, we select a Gaussian kernel [Hastie et al., 2009] that in turn needs σ to control the width of Gaussian distributions. For σ, we set the value range $[2^{-3}, 2^{-1}, ..., 2^{7}]$ (6 values) and for λ the value range $[2^{-5}, 2^{-3}, ..., 2^{5}]$ (6 values). For the implementation of the Gaussian Process, we use the Smile library [Li et al.].

The experimental process for the prediction step as described in Section 5.7 follows a leave-one-out cross-validation approach on dataset level: For every individual dataset, we use all other datasets as the annotated training datasets (running the training pipeline on them) and the target dataset as the test dataset (running the application pipeline on this dataset). The best parameters, i.e., λ and σ, are chosen automatically by the models through an 80:20 train-validation split approach on the annotated datasets. The selected parameters are then applied to the tested dataset. Finally, results for both MDCs and execution time are given in Tables 5.2 and 5.3. Recall that datasets with execution times longer than ten hours are a result of expensive MDC score prediction; at ten hours, we terminate the selection process, returning the results obtained until that point.

Refined classification. Support Vector Machines (SVMs) are a widely accepted binary classifier and have already been used to refine and improve classification results in duplicate detection [Christen, 2008a; Lehti and Fankhauser, 2006]. We pair SVMs with a Gaussian kernel, which is a standard radial basis function (RBF) kernel, and configure two parameters: Gaussian kernel width σ and soft margin penalty Cs. Following the *loose grid search* recommendation [Hsu et al., 2003], we set the values $[2^{-3}, 2^{-1}, ..., 2^{7}]$ (6 values) for σ and the values $[2^{-5}, 2^{-3}, ..., 2^{15}]$ (11 values) for C. The results are reported upon an 80:20 split of train-test on the duplicates reported by the first-cut duplicate detection step on the new dataset with a 10-fold cross-validation on the train part to select the best parameters. The experiments on all datasets lasted from a few minutes to less than an hour at most.

5.7 MDedup performance evaluation

In this section, through a set of experiments, we show the potential of using MDCs as a duplicate detection classifier. Figures 5.5 to 5.7 serve as our evaluation summary and guidelines for the three experiments discussed afterwards. The figures show for each dataset four different effectiveness results that correspond to different steps in the MDedup algorithm. Each result is evaluated with the classification metrics recall, precision, and F-measure.

MDC Selection and *MDC Selection with classification refinement* are two variations of the training pipeline. Hence, they both utilize the gold standard to obtain their results. Because MDedup's training pipeline identifies the highest scored MDC for duplicate detection, the idea is the following: If we consider the selected MDC as a duplicate detection classifier, this classifier should provide us with the optimal F-measure *any* matching dependency combination (from the set of discovered MDs) can achieve; it, hence, represents an upper bound for our experiments and describes the capabilities of discoverable MDs in the domain of duplicate detection – we cannot predict any better results using MDs alone. To improve the recall of the best MDC, we can also apply the refinement step to its results. The version with classification refinement follows the principles we discussed for the application pipeline, i.e., the first-cut duplicate detection produces a set of duplicate pairs, which are then fed into an SVM classifier for training and a refined duplicate detection pass.

Analogous to the results of training pipeline, *MDC Prediction* and *MDC Prediction with classification refinement* present the results of the application pipeline. They describe MDedup's effectiveness when predicting good MDC classifiers rather than testing them on a gold standard. Training and testing of the regression model to predict the MDC scores is done in a leave-one-out cross-validation manner: Given one dataset, we train on all other datasets (and their gold standards) and test the prediction effectiveness on the given dataset (for which we omit the annotations).

As an attempt to create a baseline, we first considered possible methods that can also exploit the knowledge gathered on annotated datasets to classify record pairs in another target dataset that lacks an annotation. Transfer learning, however, for typical binary classification is not trivial, as different datasets have different attribute domains. Thus, although some literature exists on transductive transfer learning using SVMs [Deng and

Wang, 2014], it is not clear how to perform such a schema matching across datasets in an automatic way. Thus, we experimentally enforced a schema mapping by splitting the same dataset into multiple parts and, then, ran both MDedup and [Negahban et al., 2012] on these perfectly matching parts. We did this for all our datasets and measured on average 51% F-measure for us and 68% for [Negahban et al., 2012]. So for same domain datasets with a transitive gold standard and a schema mapping, more effective approaches exist.

We then applied a traditional classification technique that directly uses the gold standards to train and apply SVM models. The results of training directly on high-quality and large sets of similarity pairs yielded, as expected, much higher precision and recall numbers. But such training approaches are incomparable to our approach and, in particular, inapplicable to our no-gold-standard scenario.

Subsequently, we discuss three experiments in more detail: First, in Section 5.7.1, we test the general effectiveness of (discovered) MDCs when being used for duplicate detection. Second, Section 5.7.2 compares the classification effectiveness of the predicted MDCs against best selected MDCs. Third and finally, Section 5.7.3 shows the F-measure scores that MDCs can achieve with classification refinement.

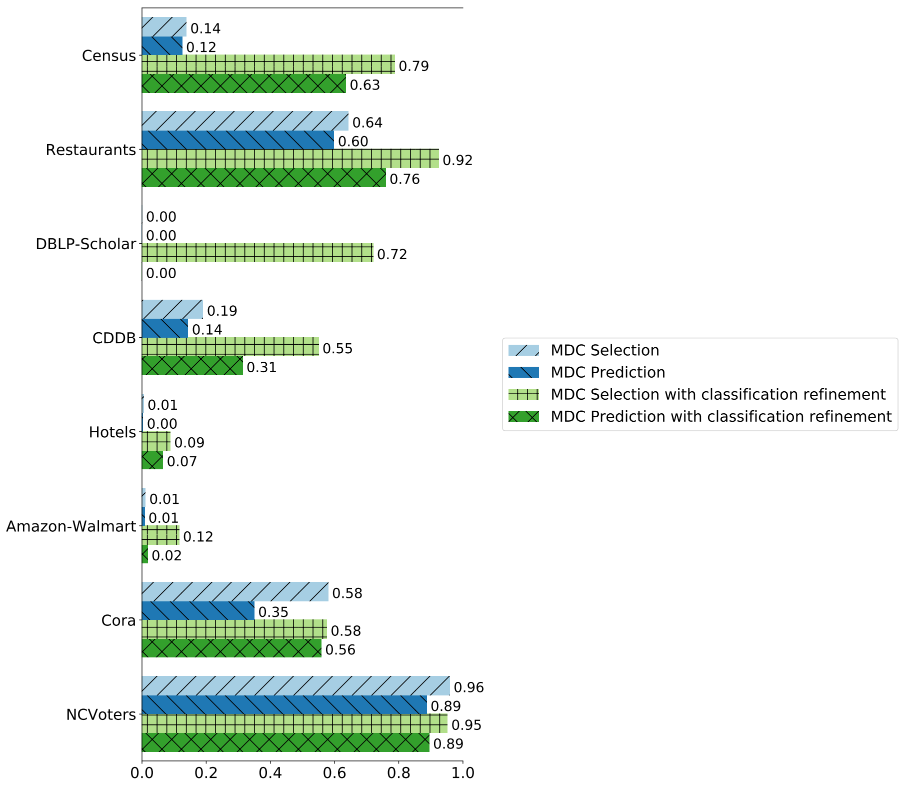


Figure 5.5: *Recall* of MD-based duplicate classifiers using the best MDC according to a gold standard (Selection) or a Gaussian process (Prediction); in both cases, no classification refinement and classification refinement are considered.

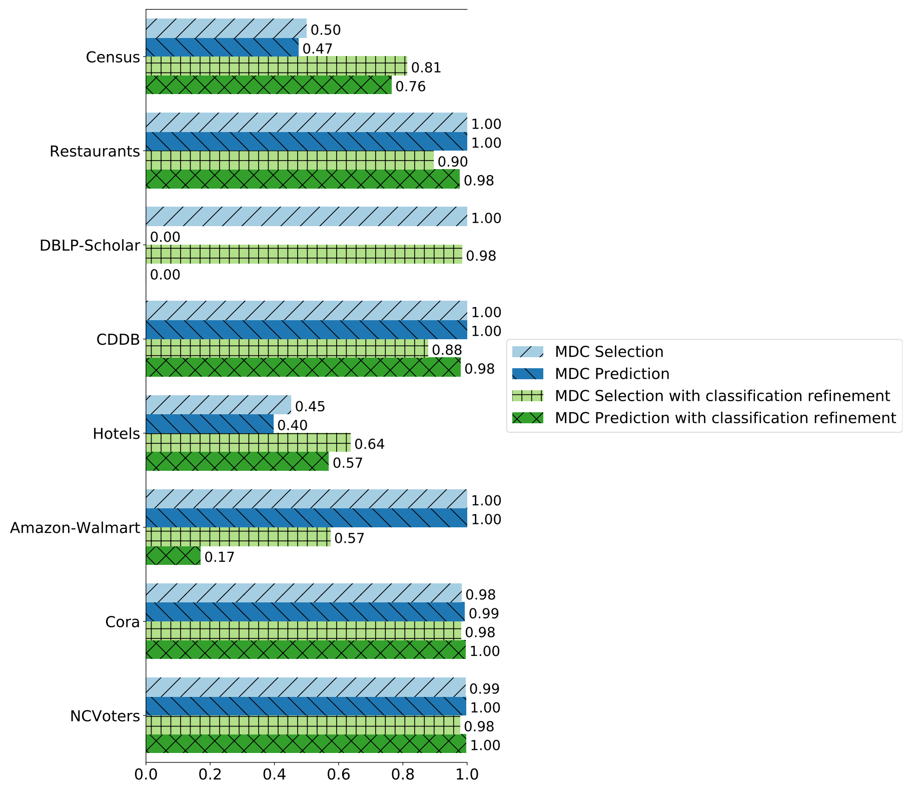


Figure 5.6: *Precision* of MD-based duplicate classifiers following using the best MDC according to a gold standard (Selection) or a Gaussian process (Prediction); in both cases, no classification refinement and classification refinement are considered.

5.7.1 MDC Selection

First, we evaluate the limits of MDs for classifying duplicates in the presence of a gold standard, using a top-k approximation. This corresponds to the training phase of our pipeline in Figure 5.1, which approaches the best-case scenario for a given dataset. More specifically, we discuss the results with the label *MDC Selection* shown in Figures 5.5 to 5.7.

Overall, the F-measure for most datasets is low. This is primarily due to poor recall – precision is near perfect on all datasets except Census and Hotels. The poor recall is mainly due to some error patterns that are not described by the discovered, minimal matching dependencies. However, the fact that the selection favors precision over recall has interesting implications: First, it shows that properties of good MDCs are learnable and transferable, because the majority of MDCs in general has a very poor precision. Second, a high precision is important for automatic approaches, because although they might not find all duplicates, their actual findings are reliable. The results are, finally, also good for classification refinement, because training subsequent models requires reliable test records (and not all records).

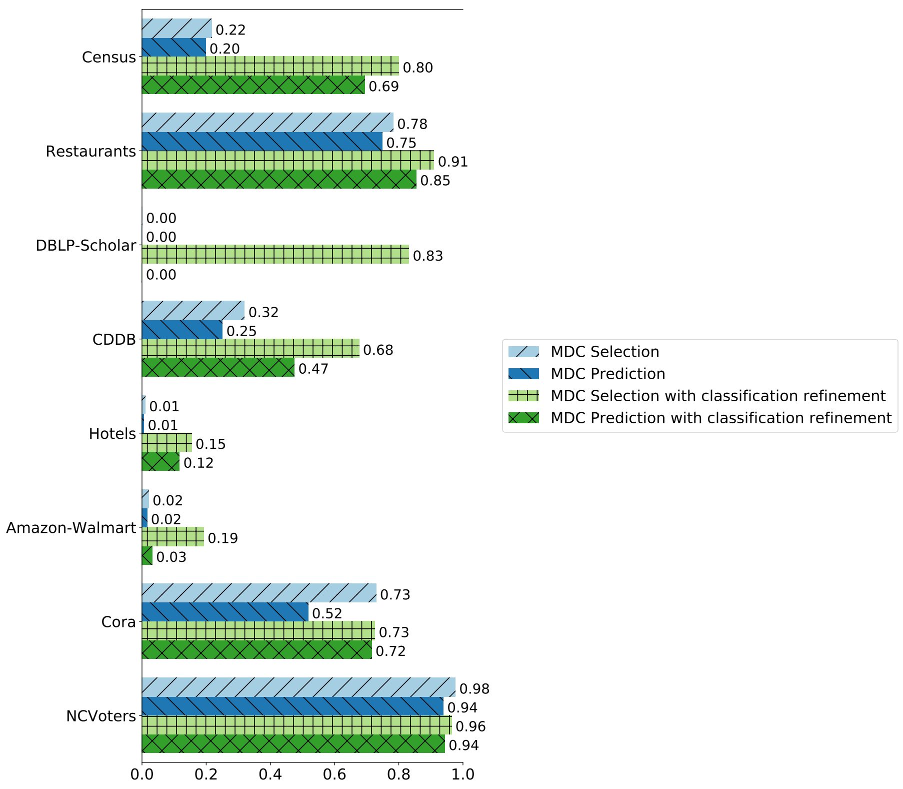


Figure 5.7: *F-measure* of MD-based duplicate classifiers using the best MDC according to a gold standard (Selection) or a Gaussian process (Prediction); in both cases, no classification refinement and classification refinement are considered.

5.7.2 MDC Prediction

Our second experiment evaluates how well MDedup can predict the MDC classifiers for duplicate detection, which is its ability to detect duplicates in new datasets. For this purpose, we discuss the measurements labeled as *MDC Prediction* in Figures 5.5 to 5.7 and compare them to the target values of *MDC Selection*, which are the best F-measure values the system can possibly achieve.

The measurements in Figures 5.5 to 5.7 show that the predicted MDCs do not achieve the F-measure scores of the selected MDCs; in particular, MDedup never predicted the best selected MDC. Considering the large search space and the fact that most MDCs perform poorly as duplicate classifiers, the results are still very good. The predicted MDCs have, in particular, high precision and are, therefore, like their selected counterparts well suited for *automatic* cleaning processes and *classification refinement*. Overall, we primarily lose recall when using the MDC scoring model instead of a gold standard.

Unfortunately, MDedup failed to predict a useful MDC for the DBLP-Scholar dataset, because the dataset offers only one useful MD. When selected and boosted, it achieves 83% F-measure. The MD does not look promising on its own and all interesting insights are derived from the correctly identified duplicates in the classification refinement. Hence, it is no surprise that our system cannot identify the MD reliably.

5.7.3 Classification refinement

In the last experiment, we evaluate the classification refinement step of MDedup. The classification refinement step aims to improve the overall F-measure with an additional SVM model. This model is trained on the high-precision duplicates that the system automatically generated with the selected/predicted MDCs and should be able to learn additional, domain-specific duplicate properties. To evaluate the refinement effect, we apply this step on top of both *MDC Selection* and *MDC Prediction*. The results, which are presented in Figures 5.5 to 5.7 with the *with classification refinement* suffix, show considerable improvements in F-measure for both selected and predicted MDCs; only Cora's and NCVoters' selected results lost one percent F-measure, which is due to slightly worse recall values. For this reason, classification refinement is a generally viable technique when using discovered MDs for duplicate detection.

Remarkably, datasets that performed poorly in the previous phases, have the largest improvements. More specifically, Amazon-Walmart, DBLP-Scholar, and Hotels were able to match only a few duplicate pairs beforehand. However, the combination of these duplicates, along with a few non-duplicates (as discussed in Section 5.5) was enough for an SVM-based approach to identify differences in the set of evaluated pairs. As expected *MDC Selection* provided SVMs with a noticeably better set of duplicates, which resulted in higher improvements and higher values of F-measure overall.

5.8 Conclusion

We proposed a system called MDedup that uses discovered matching dependencies for the *fully automatic* detection of duplicates, which means that no domain knowledge about or training data for a given target dataset is needed to detect duplicates in it. Because the duplicates discovered with only MDs usually have high precision but comparatively low recall, we proposed an SVM-based classification refinement ensemble step that in many cases greatly improves recall and, hence, the overall F-measure. Our experiments highlight the capabilities of discovered MDs when used for duplicate detection, thus closing a gap in research. They also show that an algorithm can, to a certain extent, domain-independently learn how to score MDs for being good duplicate classifiers.

The achieved F-measure scores are certainly not competitive with those produced by algorithms that can learn on pre-labeled data or gold standards, but they are very useful for scenarios where no training data is available. The domain-independent properties of MDCs allow us to recommend their application, independent of a gold standard. No other duplicate detection approach is able to process arbitrary datasets without a human providing domain-specific input, such as initial classifier rules, data labels, or similarity thresholds. The usage of MDs in our approach, finally, also allows the user to interpret the detected duplicates.

6

Conclusion and Future Work

In this thesis, we examined the problem of duplicate detection, where entities of a database are represented by multiple records each. To this end, we have developed three different approaches to *systematically prepare data* (Chapters 3 and 4), select the *best similarity measure per attribute* (Chapter 4), and obtain duplicate detection results in a *domain-agnostic principle* (Chapter 5). Preparing data and thus improving their quality should facilitate data scientists to accomplish a more successful duplicate detection, whereas acquiring a result set of duplicates, in cases of no existing labels, allows for answers to be given when none would be otherwise possible.

In particular, by addressing the *data preparation* issues, first, we confirmed that improving the quality of data impacts the resulting application, which for us is duplicate detection. Second, our two pipelines provide a good recipe to resolve data-quality related issues before resolving duplicates. Lastly, our discussion of which data preparators were successful for which attributes, along with similarity measures and frameworks for addresses, should provide data scientists valuable insights for their tasks.

Regarding *MDedup*, we believe there are a number of key points to keep as a synopsis.. First, it serves as an approach that provides human-understandable rules for duplicate detection. Second, finding high-precision results is often a requirement for many applications, where false positives have to be avoided, and as our experimental results showed is usually the case for MDedup. Lastly, the largest benefit of using MDedup is that it can be applied on new and label-less datasets, of even different domains, and still manages to return a result set of duplicates.

The ideas developed in Chapters 3 to 5 provide good evidences of the importance of the presented concepts. However, as is typical in research, hard lines had to be decided and drawn at the end to provide a reasonable product. Therefore, among the possible directions that could be followed for our projects, we suggest the following future work:

Data Preparation – (Chapter 3). Apart from the research directions outlined at the end of Chapter 3 there is still an open question we did not address in this thesis due to its complexity: Suggesting preparations on a statistical basis and not on an empirical one, as we do now. More specifically, this means correlating preparations with the resolution of specific data issues and thus not having to apply them and retroactively decide whether their application was helpful or not. However, this would require a deeper understanding of the data, along with the effect of the preparators on different data types and distributions.

Address Matching – (Chapter 4). Similarly, apart from the avenues for future work described in Chapter 4, an interesting direction is to incorporate contextual information about location in the data. This could allow for more sophisticated methods of data normalization and matching in the remaining attributes of the records. For instance, knowing district-specific slang and synonyms can be used while matching attributes such as person and company names. Such knowledge could be further extended with timeline information, in case that is available, as the context changes not only across different locations, but also across time.

MDedup – (Chapter 5). As the demand for duplicate detection on datasets keeps rising, MDedup can improve on multiple aspects. First, improvements on the discovery of matching dependencies (MDs) may include a smarter selection of similarity measures and minimum thresholds automatically for a dataset's attributes. Nonetheless, the focus should always be on high precision; thus, even if more relaxed similarity measures are selected this should be compensated by higher minimum thresholds. Second, expanding the current feature list should allow for a better understanding of MD combinations (MDCs) and therefore facilitate a better prediction of F-measure scores. Finally, better regression models with more diverse labeled datasets should close even further the gap between the predicted MDCs and the best available ones.

In a world where data keeps having more influence on our lives, knowing how to improve data's quality is of paramount importance. Acquiring such knowledge comes, however, with great responsibility. Data scientists, have access to vast amounts of many times private data of different domains, which know how to navigate, make more useful, and extract priceless knowledge from, giving them unprecedented power. Let this power be used wisely and data science be remembered as the science that bridged other sciences and enabled higher levels of prosperity to be reached.

References

Abedjan, Z., Morcos, J., Gubanov, M. N., Ilyas, I. F., Stonebraker, M., Papotti, P., and Ouzzani, M. DataXFormer: Leveraging the Web for Semantic Transformations. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2015.

Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., and Tang, N. Detecting Data Errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment (PVLDB)*, 9(12):993–1004, 2016.

Aizawa, A. and Oyama, K. A fast linkage detection scheme for multi-source information integration. In *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*, pages 30–39, 2005.

Ao, J. and Chirkova, R. Effective and Efficient Data Cleaning for Entity Matching. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA@SIGMOD)*, volume 2, pages 1–7. ACM, 2019.

Bahmani, Z., Bertossi, L. E., Kolahi, S., and Lakshmanan, L. V. Declarative Entity Resolution via Matching Dependencies and Answer Set Programs. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 380–390, 2012.

Bahmani, Z., Bertossi, L., and Vasiloglou, N. ERBlox: combining matching dependencies with machine learning for entity resolution. In *International Conference on Scalable Uncertainty Management (SUM)*, pages 399–414. Springer, 2015.

Banko, M. and Brill, E. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL)*, pages 26–33, 2001.

Bansal, N., Blum, A., and Chawla, S. Correlation clustering. *Machine learning*, 56(1-3): 89–113, 2004.

Baxter, R., Christen, P., and Churches, T. A Comparison of Fast Blocking Methods for Record Linkage. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, volume 3, pages 25–27, 2003.

Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S. E., and Widom, J. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.

REFERENCES

Bilenko, M. and Mooney, R. J. Adaptive Duplicate Detection Using Learnable String Similarity Measures. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 39–48, 2003.

Bleiholder, J. and Naumann, F. Data fusion. *ACM Computing Surveys*, 41(1):1–41, 2009.

Bron, C. and Kerbosch, J. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

Caruccio, L., Deufemia, V., and Polese, G. Relaxed functional dependencies: a survey of approaches. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 28 (1):147–165, 2016.

Chandel, A., Hassanzadeh, O., Koudas, N., Sadoghi, M., and Srivastava, D. Benchmarking declarative approximate selection predicates. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 353–364. ACM, 2007.

Christen, P. A two-step classification approach to unsupervised record linkage. In *Proceedings of the Australasian Conference on Data Mining and Analytics (AusDM)*, pages 111–119. Australian Computer Society, Inc., 2007.

Christen, P. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 151–159. ACM, 2008a.

Christen, P. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the Australasian Workshop on Health Data and Knowledge Management (HDKM)*, pages 17–25, 2008b.

Christen, P. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(9): 1537–1555, 2012a.

Christen, P. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer-Verlag Berlin Heidelberg, 2012b.

Christen, P., Willmore, A., and Churches, T. A probabilistic geocoding system utilising a parcel based address file. In *Data Mining*, pages 130–145. Springer, 2006.

Christen, P., Gayler, R., and Hawking, D. Similarity-aware indexing for real-time entity resolution. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1565 – 1568. ACM Press, 2009.

Chu, X., Ilyas, I. F., and Koutris, P. Distributed Data Deduplication. *Proceedings of the VLDB Endowment (PVLDB)*, 9(11):864–875, 2016.

Churches, T., Christen, P., Lim, K., and Zhu, J. X. Preparation of name and address data for record linkage using hidden Markov models. *BMC Medical Informatics and Decision Making*, 2(1):9, 2002.

Chvatal, V. and Sankoff, D. Longest common subsequences of two random sequences. *Journal of Applied Probability*, 12(2):306–315, 1975.

Cochinwala, M., Kurien, V., Lalk, G., and Shasha, D. Efficient data reconciliation. *Information Sciences*, 137(1):1–15, 2001.

Cohen, W. W., Ravikumar, P., Fienberg, S. E., et al. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of the International Workshop on Information Integration on the Web (IIWeb)*, pages 73–78, 2003.

Collins, M., Schapire, R. E., and Singer, Y. Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 48(1-3):253–285, 2002.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Culotta, A., Wick, M., Hall, R., Marzilli, M., and McCallum, A. Canonicalization of database records using adaptive similarity measures. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 201–209. ACM, 2007.

Damerau, F. J. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176, 1964.

Das Sarma, A., He, Y., and Chaudhuri, S. ClusterJoin: A Similarity Joins Framework Using Map-reduce. *Proceedings of the VLDB Endowment (PVLDB)*, 7(12):1059–1070, 2014.

Deng, Z. and Wang, S. Novel Inductive and Transductive Transfer Learning Approaches Based on Support Vector Learning. In *Support Vector Machines Applications*, pages 49–103. Springer, 2014.

Draisbach, U., Christen, P., and Naumann, F. Transforming Pairwise Duplicates to Entity Clusters for High-quality Duplicate Detection. *Journal of Data and Information Quality (JDIQ)*, 12(1):30, 2019.

Ebraheem, M., Thirumuruganathan, S., Joty, S., Ouzzani, M., and Tang, N. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment (PVLDB)*, 11(11):1454–1467, 2018.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1): 1–16, 2007.

Fan, W. Dependencies revisited for improving data quality. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 159–170, 2008.

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

Gale, D. and Shapley, L. S. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

REFERENCES

Galhardas, H., Florescu, D., Shasha, D., and Simon, E. AJAX: An Extensible Data Cleaning Tool. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, page 590, 2000.

Galler, B. A. and Fisher, M. J. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.

Gardezi, J., Bertossi, L., and Kiringa, I. Matching dependencies: semantics and query answering. *Frontiers of Computer Science*, 6(3):278–292, 2012.

Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., Srivastava, D., et al. Approximate string joins in a database (almost) for free. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 1, pages 491–500, 2001.

Gruenheid, A., Dong, X. L., and Srivastava, D. Incremental Record Linkage. *Proceedings of the VLDB Endowment (PVLDB)*, 7(9):697–708, 2014.

Guha-Sapir, D., Davis, R., Gall, M., Wallemacq, P., and Cutter, S. Exploring the potential of geocoding the impact of disasters: The experience of global and national databases. In *EGU General Assembly Conference Abstracts*, volume 17, 2015.

Gulwani, S. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the International Conference on Programming Languages (SIGPLAN)*, volume 46, pages 317–330, 2011.

Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

Hamming, R. W. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

Han, J., Kamber, M., and Pei, J. *Data mining: concepts and techniques.* Morgan Kaufmann, 3rd edition, 2011.

Hand, D. and Christen, P. A note on using the F-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28(3):539–547, 2018.

Hassanzadeh, O., Chiang, F., Lee, H. C., and Miller, R. J. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):1282–1293, 2009.

Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

He, Y., Chu, X., Ganjam, K., Zheng, Y., Narasayya, V., and Chaudhuri, S. Transform-data-by-example (TDE): an extensible search engine for data transformations. *Proceedings of the VLDB Endowment (PVLDB)*, 11(10):1165–1177, 2018.

Hernández, M. A. and Stolfo, S. J. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, 2(1):9–37, 1998.

Herzog, T. N., Scheuren, F. J., and Winkler, W. E. *Data Quality and Record Linkage Techniques*. Springer Publishing Company, Incorporated, 1st edition, 2007.

Ho, T. K. Random decision forests. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 278–282. IEEE, 1995.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. A practical guide to support vector classification. *National Taiwan University, Taipei*, 2003.

IBM Big Data Analytics Hub. Extracting business value from the 4 V's of big data, 2016.

Ilyas, I. F., Chu, X., et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

Inman, J. *Navigation and Nautical Astronomy, for the Use of British Seamen*. F. & J. Rivington, 1849.

Jaccard, P. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272, 1901.

Jaro, M. A. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406): 414–420, 1989.

Jin, Z., Anderson, M. R., Cafarella, M., and Jagadish, H. Foofah: Transforming data by example. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 683–698, 2017.

Kandel, S., Heer, J., Plaisant, C., Kennedy, J., van Ham, F., Riche, N. H., Weaver, C., Lee, B., Brodbeck, D., and Buono, P. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

Karakasidis, A., Koloniari, G., and Verykios, V. S. Scalable blocking for privacy preserving record linkage. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 527–536, 2015.

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In *International Conference on Machine Learning (ICML)*, volume 2, 2015.

Köpcke, H., Thor, A., and Rahm, E. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1-2): 484–493, 2010.

Koumarelas, I., Kroschk, A., Mosley, C., and Naumann, F. Experience: Enhancing Address Matching with Geocoding and Similarity Measure Selection. *Journal of Data and Information Quality (JDIQ)*, 10(2):8, 2018.

REFERENCES

Koumarelas, I., Jiang, L., and Naumann, F. Data Preparation for Duplicate Detection. *Journal of Data and Information Quality (JDIQ)*, 12(3):24, 2020a.

Koumarelas, I., Papenbrock, T., and Naumann, F. MDedup: Duplicate Detection with Matching Dependencies. *Proceedings of the VLDB Endowment (PVLDB)*, 13(5):712–725, 2020b.

Kunft, A., Katsifodimos, A., Schelter, S., Rabl, T., and Markl, V. Blockjoin: efficient matrix partitioning through joins. *Proceedings of the VLDB Endowment (PVLDB)*, 10(13):2061–2072, 2017.

Lee, Y. W., Pipino, L. L., Funk, J. D., and Wang, R. Y. *Journey to data quality.* The MIT Press, 2009.

Lehti, P. and Fankhauser, P. Unsupervised duplicate detection using sample non-duplicates. In *Journal of Data Semantics (JoDS)*, pages 136–164. Springer, 2006.

Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of massive datasets.* Cambridge university press, 2014.

Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Li, H. et al. Smile - Statistical Machine Intelligence and Learning Engine. `https://haifengl.github.io/index.html`. [Online; accessed 19-April-2020].

Li, P., Dong, X. L., Maurino, A., and Srivastava, D. Linking temporal records. *Proceedings of the VLDB Endowment (PVLDB)*, 4(11):956–967, 2011.

Li, S., Liang, H., and Ramadan, B. Two Stage Similarity-aware Indexing for Large-scale Real-time Entity Resolution. *Proceedings of the Australasian Conference on Data Mining and Analytics (AusDM)*, 146, 2013.

Mann, W., Augsten, N., and Bouros, P. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment (PVLDB)*, 9(9):636–647, 2016.

McCallum, A., Nigam, K., and Ungar, L. H. Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 169–178, 2000.

Miler, M., Todić, F., and Ševrović, M. Extracting accurate location information from a highly inaccurate traffic accident dataset: A methodology based on a string matching technique. *Transportation Research Part C: Emerging Technologies*, 68:185–193, 2016.

Monge, A. E. and Elkan, C. P. The Field Matching Problem: Algorithms and Applications. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 267–270, 1996.

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 19–34, 2018.

Nan, Y., Chai, K. M., Lee, W. S., and Chieu, H. L. Optimizing F-measure: A Tale of Two Approaches. In Langford, J. and Pineau, J., editors, *International Conference on Machine Learning (ICML)*, pages 289–296, 2012.

Naumann, F. and Herschel, M. *An Introduction to Duplicate Detection.* Morgan and Claypool Publishers, 2010.

Negahban, S., Rubinstein, B. I. P., and Gemmell, J. Scaling multiple-source entity resolution using statistically efficient transfer learning. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 2224–2228, 2012.

Newcombe, H. B., Kennedy, J. M., Axford, S., and James, A. P. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

Ngonga Ngomo, A.-C. and Lyko, K. EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming. In *The Semantic Web: Research and Applications*, pages 149–163. Springer, 2012.

Odell, M. and Russell, R. The soundex coding system. *US Patents*, 1261167, 1918.

Oliveira, P., Rodrigues, F., Henriques, P., and Galhardas, H. A taxonomy of data quality problems. In *International Workshop on Data and Information Quality*, pages 219–233, 2005.

Papadakis, G., Koutrika, G., Palpanas, T., and Nejdl, W. Meta-blocking: Taking entity resolution to the next level. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(8):1946–1960, 2013.

Papadakis, G., Tsekouras, L., Thanos, E., Giannakopoulos, G., Palpanas, T., and Koubarakis, M. The return of JedAI: end-to-end entity resolution for structured and semi-structured data. *Proceedings of the VLDB Endowment (PVLDB)*, 11(12):1950–1953, 2018.

Paull, D. A Geocoded National Address File for Australia: The G-NAF What, Why, Who and When. *PSMA Australia Limited, Griffith, ACT*, 2003.

Philips, L. Hanging on the metaphone. *Computer Language*, 7(12):39–43, 1990.

Pickle, L. W., Waller, L. A., and Lawson, A. B. Current practices in cancer spatial data analysis: a call for guidance. *International Journal of Health Governance (IJHG)*, 4 (1):3, 2005.

Pullen, D., Wang, P., Talburt, J., and Wu, N. Mitigating data quality impairment on entity resolution errors in student enrollment data. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*, pages 1–5, 2013.

Pyle, D. *Data preparation for data mining.* Morgan Kaufmann, 1999.

Rahm, E. and Do, H. H. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

REFERENCES

Ramadan, B., Christen, P., Liang, H., and Gayler, R. W. Dynamic Sorted Neighborhood Indexing for Real-Time Entity Resolution. *Journal of Data and Information Quality (JDIQ)*, 6(4):15:1–15:29, 2015.

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment (PVLDB)*, 11(3):269–282, 2017.

Redman, T. C. *Data Quality: the field guide.* Digital press, 2001.

Sarawagi, S. and Bhamidipaty, A. Interactive deduplication using active learning. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 269–278, 2002.

Schirmer, P., Papenbrock, T., Koumarelas, I., and Naumann, F. Efficient Discovery of Matching Dependencies. *ACM Transactions on Database Systems (TODS)*, 2020. Accepted for publication.

Singh, R., Meduri, V. V., Elmagarmid, A., Madden, S., Papotti, P., Quiané-Ruiz, J.-A., Solar-Lezama, A., and Tang, N. Synthesizing entity matching rules by examples. *Proceedings of the VLDB Endowment (PVLDB)*, 11(2):189–202, 2017.

Song, S. and Chen, L. Discovering matching dependencies. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1421–1424. ACM, 2009.

Song, S. and Chen, L. Efficient discovery of similarity constraints for matching dependencies. *Data and Knowledge Engineering (DKE)*, 87:146–166, 2013.

Sood, S. and Loguinov, D. Probabilistic near-duplicate detection using simhash. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1117–1126. ACM, 2011.

Tibshirani, R., James, G., Witten, D., and Hastie, T. *An introduction to statistical learning-with applications in R.* Springer New York, 2013.

Vatsalan, D., Christen, P., and Verykios, V. S. Efficient two-party private blocking based on sorted nearest neighborhood clustering. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1949–1958, 2013.

Vatsalan, D., Sehili, Z., Christen, P., and Rahm, E. Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges. In *Handbook of Big Data Technologies*, pages 851–895. Springer, 2017.

Wang, J., Li, G., Yu, J. X., and Feng, J. Entity matching: How similar is similar. *Proceedings of the VLDB Endowment (PVLDB)*, 4(10):622–633, 2011.

Wang, Y., Wang, R., Ziad, M., and Lee, Y. *Data Quality.* Advances in Database Systems. Springer US, 2001.

Weis, M., Naumann, F., Jehle, U., Lufter, J., and Schuster, H. Industry-scale duplicate detection. *Proceedings of the VLDB Endowment (PVLDB)*, 1(2):1253–1264, 2008.

Winkler, W. E. and Thibaudeau, Y. An application of the Fellegi-Sunter model of Record Linkage to the 1990 US decennial census. Technical report, US Bureau of the Census, 1991.

Yang, Y., Meneghetti, N., Fehling, R., Liu, Z. H., and Kennedy, O. Lenses: An on-demand approach to ETL. *Proceedings of the VLDB Endowment (PVLDB)*, 8(12): 1578–1589, 2015.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Doktorarbeit mit dem Thema:

Data Preparation and Domain-agnostic Duplicate Detection

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Potsdam, den 20. April 2020