Doctoral theses at NTNU, 2024:173

Nils Barlaug

Performance and Interpretability of Entity Matching with Deep Learning

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of Science and Technology

Nils Barlaug

Performance and Interpretability of Entity Matching with Deep Learning

Thesis for the Degree of Philosophiae Doctor

Trondheim, May 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Computer Science

ISBN 978-82-326-7938-6 (printed ver.)
ISBN 978-82-326-7937-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2024:173

Printed by NTNU Grafisk senter

Abstract

Entity matching is the problem of identifying which records refer to the same real-world entity. It is a key data integration task and, despite decades of research, is still challenging. In recent years, deep learning has emerged as the new state-of-the-art paradigm to tackle entity matching. This new paradigm brings about new strengths, weaknesses, trade-offs, and characteristics compared to classical methods.

In this thesis, we explore the use of deep learning for entity matching with the goal of gaining insight into what these new methods contribute to the task, how they differ from classical methods, and what their current limitations are. We put special focus on interpretability and blocking because these are, in our opinion, aspects that highlight the contrasts the most.

Through a combination of literature analysis and experimental work this thesis provides three main contributions:

1. Insight and overview of how new deep learning methods compare to classical methods for entity matching.
2. A state-of-the-art model-agnostic explainability method tailored to entity matching.
3. A state-of-the-art blocking method based on set similarity joins.

We hope that these contributions are valuable to practitioners and the research community and further the development of deep learning for entity matching.

Preface

This thesis is being submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfilment of the requirements for the degree of Philosophiae Doctor (PhD).

This doctoral work has been performed at the Department of Computer Science (IDI), NTNU, Trondheim, with Professor Jon Atle Gulla (IDI) as the main supervisor and with Professor Kjetil Nørvåg (IDI), Trygve Karper (Cognite), and Alexander Gleim (Cognite) as co-supervisors.

The thesis has been supported by the Research Council of Norway and Cognite under project number 298998.

Acknowledgements

First of all, I would like to thank the Research Council of Norway, Cognite and Jon Atle Gulla for providing me with the opportunity to pursue a PhD. I am grateful for the valuable feedback, guidance on how to navigate a PhD, and fruitful discussions with my supervisor Jon Atle and my co-supervisors Kjetil, Trygve, and Alexander.

I extend my heartfelt appreciation to my amazing colleagues at IDI, whose support and shared joy have made this academic journey truly memorable. Beyond the confines of research and academia, it has been the shared laughter, camaraderie, and moments of respite that made me look forward to going to work every day. The countless lunches, ice cream breaks, random interactions in the kitchen, and shared laughs have not only lightened the academic load, but have made these years truly enjoyable. The correct thing to do would probably be to list names, but I am too afraid to forget someone.

Thank you to my parents, Hanne and Øivind, for getting me where I am today. To my girlfriend Elise, I am forever grateful that you shared this journey with me and supported me every day. I could not have done it without you.

Contents

Chapter 1

Introduction

This chapter will provide a brief overview of the thesis. We first motivate the topic before we introduce the research questions, contributions, and publications. Lastly, we give an overview over the thesis structure.

1.1 Motivation

Entity Matching (EM) is a key data integration task that determines which records across or within data sources refer to the same real-world entity when we have no unique and common identifiers [25, 34]. It is ubiquitous and emerges in practically all domains that exercise data management. Typical real-world use cases are comparing product catalogs across stores, merging of customer databases, or deduplicating inventory lists. It is tempting to think that one can avoid this altogether by simply doing a better job of data management and standardization. While this is partly true, there are two main underlying reasons outside our control we have these problems, and the need for this task, that are hard to avoid and unlikely to go away anytime soon:

1. **Imperfect Data Sources**: There are often factors outside our control that negatively affect the quality of the data we receive. Human input contains typos or other mistakes. Same for automatic data collection such as OCR on documents. Furthermore, we might not be in control of keeping our data up to date with reality — i.e., a customer can move or change name without notifying us.

2. **Decentralized and Uncoordinated Data Management**: Decisions about data procedures and formats are often taken locally without the knowledge of which other data sources it needs to integrate with in the future. Unique and common identifiers across two data sources require upfront coordination and consensus. Unfortunately, it is hard to know with certainty which data sources one will need to integrate in the future. One solution is to agree on standards, but it is simply infeasible to have ubiquitous and indefinitely forward compatible standards for everything at all times that everyone agrees upon. When a company models its product catalog or inventory list it is often not possible to follow a standard identification scheme and data format that will guarantee interoperability with all potential companies it might merge with years or decades from now. Importantly, the company can not enforce that everyone else follows the same (or any) standard.

Entity matching has been studied for decades and despite multiple rounds of new paradigms over the years it remains a non-trivial task. The previous paradigm shift was the movement towards machine learning-based methods. These methods reduced the manual tailoring of algorithms to different datasets, but required significant manual feature engineering. Similarity features have, up until recently, almost exclusively been based upon classical string similarity measures (i.e., edit distance, jaccard overlap between tokens, etc). Not only do they mostly rely on surface-level syntactical similarity, they also often need to be manually crafted, tuned, and curated for each unique use case.

Recently, deep learning has emerged as a new paradigm for performing entity matching [11]. Since deep learning has a strong ability to automatically learn features it has the potential to significantly lower the dataset-specific manual efforts that are typically needed for more traditional machine learning approaches. On the other side, deep learning methods come with their own set of challenges — like high computational requirements and lack of interpretability.

The challenges involved in Entity Matching overlap that of multiple research fields, such as information retrieval, databases, and natural language processing. Natural language processing in particular has been an important source of methods and techniques in recent time and plays a central role in this thesis. However, as we will see in Chapter 2, it is important to note that Entity Matching poses unique challenges that are not adequately addressed alone by the methods and techniques that these research fields offer.

1.2 Research Questions

In this thesis, we take a closer look at deep learning for entity matching. The overall goal is to gain insight into what these new methods contribute to the entity matching task, how they differ from classical methods, and what their current limitations are. We pay particular interest to interpretability and blocking since these are, in our opinion, aspects that highlight the contrasts the most. Using standard benchmark datasets, we hope to provide novel insight into the strengths and weaknesses of deep learning in regards to interpretability and blocking and how this differs across different types of data. We will now briefly go through the research questions.

RQ1 What are the contributions from deep learning to entity matching?

While deep learning has had a large impact on a variety of tasks, we are interested in what the unique capabilities of deep learning contribute to specifically entity matching compared to earlier methods based on traditional string similarity measures, information retrieval, database techniques, and machine learning.

RQ2 How does modern deep learning approaches compare to classical approaches for blocking?

One of the core challenges of entity matching is that high precision requires explicit comparison of record pairs but the number of pairs is quadratic in the number of records. Therefore, entity matching is typically done in two steps. First, we perform a high-recall step, called blocking, that implicitly prune away obvious non-matches and generate explicit record pairs that are good candidates for being matches. Then we perform a high-precision step, called record pair classification, that compare explicit pairs and classify them as matches or non-matches. The blocking step have unique challenges in terms of scalability and recall. It is hard to effectively prune the quadratic candidate space while also consistently picking out the high-similarity pairs from the noise. We seek out to understand how deep learning approaches compare to more classical ones in this critical step.

RQ3 Can we explain predictions from deep learning matchers and classical matchers in the same frame of interpretation, and if so, how?

One drawback of deep learning methods is the lack of interpretability. This makes it harder to compare to classical methods. However, in recent years

there has been considerable research on explainability techniques and methods. We investigate whether we can exploit explainability methods to understand the behavior of both deep learning and classical matchers in the same way.

RQ4 How does explainbility methods adapted to entity matching improve interpretation of matchers?

There have been proposed many explainability techniques, but relatively few have been adapted to entity matching. We are interested in finding out which benefits there is to doing so.

1.3 Approach

We mainly address RQ1 through literature analysis, while R2, R3, and R4 are mainly addressed through experimental analysis. Our experimental work on both blocking and explainability uses and extends existing state-of-the-art work and methods from the field. For blocking, we extend and improve existing classical approaches in order to compare them more fairly to modern deep learning-based approaches, while for explainable entity matching we extend and specialize existing general explainability methods to entity matching in order to meet the unique challenges it has.

Throughout the thesis, we will make use of mostly the same standard, and widely used, datasets from Magellan Data Repository [30]. This makes the experimental results easier to compare between different parts of the thesis and against existing work.

When evaluating, we focus on computational cost, time, interpretability, and practicality in real-world use. As far as possible, we try to represent the trade-offs when balancing precision and recall instead of assuming a desired balance. Most experiments are purely quantative and are performed by custom software implementation. However, due to the subjective nature of interpretability, we also ground our work on explainability in user studies.

1.4 Contributions

The main contributions of the this thesis are:

C1 **Insight and overview of how new deep learning methods compare to classical methods for entity matching**. We provide a substantial

survey of the use of neural networks in entity matching and go into detail about how the new generation of deep learning methods differ from previous methods.

C2 **A state-of-the-art model-agnostic explainability method tailored to entity matching**. The method we propose enables us to do effective analysis on predictions from both classical and deep learning methods.

C3 **A state-of-the-art blocking method based on set similarity joins**. We show results suggesting that classical methods can still outperform deep learning-based methods for blocking by significant margins in widely used benchmarks.

1.5 Publications

This thesis is a collection of three papers:

Paper A **Neural Networks for Entity Matching: A Survey** [11]
Nils Barlaug, Jon Atle Gulla
Published in ACM *Transactions on Knowledge Discovery from Data* (TKDD) in 2021

Paper B **LEMON: Explainable Entity Matching** [9, 8]
Nils Barlaug
Published in IEEE *Transactions on Knowledge and Data Engineering* (TKDE) in 2022, with extended version (as demanded and approved by reviewers) available on arXiv.

Paper C **ShallowBlocker: Improving Set Similarity Joins for Blocking**
Nils Barlaug
Being prepared for submission

In addition, the following were published during the thesis work but is not directly related to the thesis:

Paper D **Tailoring Entity Matching for Industrial Settings** [10]
Nils Barlaug
Published in ACM *Proceedings of the International Conference on Information and Knowledge Management* (CIKM) in 2019

Paper E **Balancing Multi-Domain Corpora Learning for Open-Domain Response Generation** [146]
Yujie Xing, Jinglun Cai, Nils Barlaug, Peng Liu, Jon Atle Gulla
Published in ACL *Findings of the Association for Computational Linguistics* (NAACL) in 2022.

Paper	RQ1	RQ2	RQ3	RQ4	RQ5	C1	C2	C3
A	•	•	•			•		
B				•	•		•	
C		•						•

Table 1.1: Overview of how the different papers covers the different research questions and main contributions.

Table 1.1 provide an overview of which research questions and main contributions each publication addresses. The three following chapters are adapted from the papers.

1.6 Thesis Structure

Chapter 1 **Introduction**: Motivates the thesis, presents research questions and main contributions, and provide an overview of the thesis.

Chapter 2 **Neural Networks for Entity Matching (Paper A)**: A detailed survey of the use of neural networks for entity matching.

Chapter 3 **Approach-Agnostic Explainability for Entity Matching (Paper B)**: Overview of existing work on explainability for entity matching, introduction of a new state-of-the-art method, and extensive experimental results.

Chapter 4 **Strong Blocking Baseline for Deep Learning (Paper C)**: A closer look into the blocking step, introduction of a new state-of-the-art method based on classical string similarity measures and set similarity joins, and extensive experimental results comparing the proposed method with existing deep learning and classical methods.

Chapter 5 Conclusion: Discussion of the research questions and the consequences of the contributions, as well as some thoughts about future research.

Chapter 2

Neural Networks for Entity Matching

Paper A

Original title: Neural Networks for Entity Matching: A Survey
Authors: Nils Barlaug, Jon Atle Gulla
Published in: ACM *Transactions on Knowledge Discovery from Data* (TKDD) 2021

Entity matching is the problem of identifying which records refer to the same real-world entity. It has been actively researched for decades, and a variety of different approaches have been developed. Even today, it remains a challenging problem, and there is still generous room for improvement. In recent years we have seen new methods based upon deep learning techniques for natural language processing emerge.

In this survey, we present how neural networks have been used for entity matching. Specifically, we identify which steps of the entity matching process existing work have targeted using neural networks, and provide an overview of the different techniques used at each step. We also discuss contributions from deep learning in entity matching compared to traditional methods, and propose a taxonomy of deep neural networks for entity matching.

2.1 Introduction

Our world is becoming increasingly digitalized. While this opens up a number of new, exciting opportunities, it also introduces challenges along the way. A substantial amount of the value to be harvested from increased digitalization depends on integrating different data sources. Unfortunately, many of the existing data sources one wishes to integrate do not share a common frame of reference. For example, let us say a factory wants to use statistics from equipment maintenance logs to decide which equipment to prioritize for upgrades. Currently, at this factory, equipment inventory is kept in one system, while maintenance logs are kept in a separate system. Sadly, these two systems do not refer to equipment in the same way – i.e., there are no common identifiers or names across the two systems. While it is possible for a human to identify which maintenance logs belong to which equipment in the inventory system, there is no simple, automatic way to tie the maintenance logs to the inventory records.

Entity matching is the field of research dedicated to solving the problem of identifying which records refer to the same real-world entity. It is an important data integration task that often arises when data originate from different sources. The records are usually assumed to either be from two different data sources without duplicates or from the same data source with duplicates. It is not a new problem. A group of similar problems has been studied for a long time in a variety of fields under different names (see Section 2.2). Despite having been researched for decades, entity matching remains a challenging problem in

practice. There are several factors that make it difficult in general:

- **Poor data quality** : Real-world data is seldom completely clean, structured, and homogeneous. Data originating from manual insertion can contain typos, alternative spellings, or fail to comply with the schema (e.g., mixing first and last name). Automatic processes extracting information from unstructured sources might not always be accurate on the scope of attributes (e.g., {`firstName: "John Smith", lastName: "Programmer"`}). Furthermore, some data might simply be missing. Data in entity matching is often assumed to be structured in records. However, it is not unusual that these records are in practice semi-structured because of certain unstructured string attributes – opening up a world of possible inconsistencies – for example, a `name` attribute (`"John Smith"`, `"Smith, John"`, `"John R. Smith"`, `"John Richard Smith"`) or an `adress` attribute. In addition, we cannot always expect different data sources to follow the same schema, format, and syntactic conventions.
- **The large number of possible matches**: Given $|A|$ records from one data source and $|B| \in \Theta(|A|)$ from another, there are $\Theta(|A|^2)$ possible matches. We would normally expect the number of positive matches to be $O(|A|)$. This has two important implications. First, it is infeasible to explicitly compare all possible pairs for any nontrivial number of records. Second, there is an extreme imbalance between positive and negative matches; more specifically, there are $\Omega(|A|)$ times as many negative as positive matches. The potential for false positives is inherently greater. If one wants to use a learning-based approach, it can be difficult to label enough positive examples, since they occur in an ocean of negative examples.
- **Dependency on external human knowledge and interaction**: The space of potential entity matching problem instances is unbounded and offers great variety. While a substantial part of the instances can of course, in theory, be solved automatically, in many real-world instances, it is either unrealistic or impossible to perform matching as an automatic, isolated process, as the data sources simply do not contain all necessary information. Moreover, to perform matching, our solution has to interact with human experts and make use of their knowledge. Human interaction is in itself a complex domain.

Deep learning has in recent years become an essential part of multiple research fields, most notably in fields such as computer vision and natural language

processing, which are concerned with unstructured data. Its most prominent advantage over earlier approaches is its ability to learn features instead of relying on carefully handcrafted features [74]. Researchers have already realized the potential advantage of deep learning for entity matching [e.g., 39, 91]. In this survey, we aim to summarize the work done so far in the use of neural networks for entity matching.

2.1.1 Research questions

One of the challenges of comparing how neural networks are used in entity matching is that published methods often do not address the exact same problem. They tend to cover somewhat different aspects of entity matching. With this is in mind, we formulate the following research questions:

- How do methods using neural networks for entity matching differ in what they solve, and how do the methods that address the same aspects differ in their approaches?
- What benefits and opportunities does deep learning provide for entity matching, and what challenges does it pose?
- How can we categorize the different deep neural networks used for entity matching?

2.1.2 Main contributions

To answer our research questions, we provide the following main contributions:

- We use a reference model of the traditional entity matching process to identify which steps of process that existing work has targeted using neural networks and provide an overview of the different techniques that are used for each step.
- We discuss the contributions of deep learning to entity matching compared to traditional approaches using a proposed reference model for a deep learning-based entity matching process.
- We propose a taxonomy of deep neural networks for entity matching.
- We discuss challenges and propose potential future work for deep learning in entity matching understood in the context of our reference entity matching process and deep network taxonomy.

2.1.3 Outline

First, as necessary background information, Section 2.2 will introduce the problem definition and give a brief introduction to neural networks. Section 2.3 mentions related work — both publications that survey or summarize similar topics and problems that are similar to entity matching. We then provide an overview of the surveyed methods using a reference model of the entity matching process as a framework in Section 2.4, before we in Section 2.5 take a step back and discuss contributions from deep learning to entity matching compared to more traditional approaches. With those contributions in mind, we introduce a taxonomy of deep neural networks for entity matching in Section 2.6. Section 2.7 provide a brief overview of how evaluation is performed and reported comparative evaluations between deep learning approaches and traditional methods. Finally, we discuss challenges and opportunities for deep learning in entity matching in Section 2.8.

2.2 Background

This section introduces the entity matching problem definition and its many names and variations. What follows is a brief introduction to neural networks and deep learning and how they are used with text.

2.2.1 Problem definition

Let A and B be two data sources. A has the attributes $(A_1, A_2, ..., A_n)$, and we denote records as $a = (a_1, a_2, ..., a_n) \in A$. Similarly, B has the attributes $(B_1, B_2, ..., B_m)$, and we denote records as $b = (b_1, b_2, ..., b_m) \in B$. A data source is a set of records, and a record is a tuple following a specific schema of attributes. An attribute is defined by the intended semantics of its values. So $A_i = B_j$ if and only if values a_i of A_i are intended to carry the same information as values b_j of B_j, and the specific syntactics of the attribute values are irrelevant. Attributes can also have metadata (like a name) associated with them, but this does not affect the equality between them. We call the tuple of attributes $(A_1, A_2, ..., A_n)$ the schema of data source A, and correspondingly for B.

The goal of entity matching is to find the largest possible binary relation $M \subseteq A \times B$ such that a and b refer to the same entity for all $(a, b) \in M$. In other words, we would like to find all record pairs across data sources that refer

to the same entity. We define an entity to be something of unique existence[1]. Attribute values are often assumed to be strings, but that is not always the case. It is important to note that the two record sets need not necessarily have the same schema.

Aspects beyond what the surveyed methods cover have been intentionally left out. For example, we make no matches within each data source, only across the two. Which is not to say there cannot be duplicates within a data source. However, in this problem definition, we assume that we are not interested in finding them. In practice, it is quite common to assume no duplicates within the data sources. If we are explicitly interested in finding duplicates within a single data source, we can, as will be mentioned below, address duplicates in this formulation of the problem by simply having $A = B$.

In addition, there is also a more subtle assumption in this problem definition: The record sets A and B are assumed to operate with the same taxonomic granularity. This is not necessarily always the case. One data source might refer to households; the other, to individuals, or two data sources could refer to street-level addresses and postal code areas, respectively. In many cases, it would still make sense to match records that do not strictly refer to the same entity, but rather refer to entities with some defined taxonomic closeness. We leave this out of the definition for simplicity, as it does not affect our analysis of the surveyed methods.

Somewhat ironically, as often pointed out, entity matching itself suffers from the problem of being referenced by many different names, some referring to the exact same problem, while others are slight variations, specializations, or generalizations. In addition, the names are not used completely consistently. Table 2.1 lists a selection of these names. We will comment on a few.

Table 2.1: Some of the many names that are used for entity matching or similar variations of it.

Entity matching	Entity resolution	Record linkage
Data matching	Data linkage	Reference reconciliation
String matching	Approximate string matching	Fuzzy matching
Fuzzy join	Similarity join	Deduplication
Duplicate detection	Merge-purge	Object identification
Re-identification		

[1] An entity does not have to be a physical object, but can also be abstract or conceptual – e.g., a company or an event.

Entity resolution, *record linkage*, and *data matching* are frequently used for more or less the same problem as we defined above. It is not unusual that A and B are assumed to have the same schema — either because the schemas are, in fact, equal, or because some kind of schema matching has already been performed as a separate step. Sometimes, fusing the matching pairs to one representation is considered a final step of the problem. If we also have duplicates within each data source, it might be necessary to cluster and fuse more than two records at a time. In this article, we will stick to the more narrow definition laid out above. *Deduplication* or *duplicate detection* is the problem of identifying which records in the same data source refer to the same entity, and can be seen as the special case $A = B$. *String matching* attempts to find strings that refer to the same entity and can be regarded as the special case $n = m = 1$, if strings are interpreted as single-attribute records.

2.2.2 Neural networks and deep learning

We provide a brief and simplified description of neural networks and deep learning, followed by a short introduction to how deep learning is used in natural language processing. A comprehensive introduction to these topics is outside the scope of this survey. See instead, for example, Goodfellow et al. [48], from which we will adapt some of our notation in the following paragraphs, for a general introduction to deep learning, and Goldberg [47] and Jurafsky and Martin [63] for introductions to deep learning for natural language processing.

A *neural network* is a machine learning model. We wish to approximate some unknown function $f^*(\boldsymbol{x}) = \boldsymbol{y}$ that can map from some interesting input $\boldsymbol{x}$ to some desired output $\boldsymbol{y}$. Usually, we will have some examples $D = \{(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)}) | 1 \leq j \leq m\}$, which are known to be such that $f^*(\boldsymbol{x}^{(j)}) \approx \boldsymbol{y}^{(j)}$ for all j, to help guide us. To approximate f^* we define a function $f(\boldsymbol{x}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, and then try to learn what $\boldsymbol{\theta}$ should be using the examples D. This function f is the neural network.

Even though there are no strict requirements for what constitutes a neural network, they usually follow a common recipe. Generally, we let f consist of one or more nested functions $f(\boldsymbol{x}) = f_L(f_{L-1}(...f_1(\boldsymbol{x})))$. Each such function f_l would normally be a linear operation, like matrix multiplication, using the parameters $\boldsymbol{\theta}$ and then nested by a nonlinear element-wise operation. For example, $f_l(\boldsymbol{x}) = \max(0, W\boldsymbol{x} + \boldsymbol{b})$, where both W and $\boldsymbol{b}$ are part of $\boldsymbol{\theta}$ and max is element-wise. We call these nested functions *layers*, and L is the *depth* of the network. When a neural network has several layers (no clear threshold), we call it a deep neural network.

Given a suitable network architecture f, we try to find parameters $\boldsymbol{\theta}$ that will make it behave close to the examples D. We first define a loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ quantifying how wrong a prediction $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$ is compared to the correct $\boldsymbol{y}$. Then we randomly initialize $\boldsymbol{\theta}$ and perform some variant or descendant of stochastic gradient descent (SGD) with mini-batches:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{1}{|\widetilde{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \widetilde{D}} \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))$$

where α is the learning rate, and $\widetilde{D} \subset D$ is a random mini-batch. The stopping criterion and other details vary between methods. This procedure is expensive, because it needs to evaluate f and differentiate $\mathcal{L}$ with respect to $\boldsymbol{\theta}$. To make it efficient, we make sure to choose $|\widetilde{D}| \ll |D|$ and also differentiate with the backpropagation algorithm. Generally, we can interpret f as a directional acyclic computational graph. The backpropagation algorithm simply applies dynamic programming using the chain rule over this computational graph.

The real strength of deep learning is its ability to do hierarchical representation learning. With modern techniques, multilayered networks are able to learn useful features from relatively unstructured input data [74]. This is especially valuable for data such as images and text, which are notoriously hard to extract good features from with manually crafted procedures.

Deep learning for natural language processing

Many state-of-the-art methods for natural language processing are deep learning models [e.g., 132, 32, 147]. Central to all these methods is how text is transformed to a numerical format suitable for a neural network. This is done through embeddings, which are translations from textual units to a vector space – traditionally available in a lookup table. The textual units will usually be characters or words. An embeddings lookup table can be seen as parameters to the network and be learned together with the rest of the network end-to-end. That way the network is able to learn good distributed character or word representations for the task at hand. The words used in a data set are often not unique to that data set, but rather just typical words from some language. Therefore one can often get a head start by using pretrained word embeddings like word2vec [88], GloVe [104] or fastText [16], which have been trained on enormous general corpora. Following a rather recent trend, large pretrained networks that can produce contextualized word embeddings that take into account the surrounding words are also available [105, 32, 109].

Text is naturally interpreted as a sequence. It is therefore perhaps not so surprising that neural networks designed for sequences are often used. One way to model sequences is to use Convolutional Neural Networks (CNNs) — first popularized by computer vision applications — which has received considerable attention within the natural language processing community [66, 27]. However, a more prominent sequence model has been Recurrent Neural Networks (RNN) [41] and their variants. RNNs are constructed by repeating the same layer multiple times. Each layer takes both the output from the previous layer as well as some part of an input sequence. So assuming the input to be a sequence $\boldsymbol{x_1}, ..., \boldsymbol{x_L}$, we nest layers recursively as $\boldsymbol{h_l} = f_l(\boldsymbol{h_{l-1}}, \boldsymbol{x_l}; \boldsymbol{\theta})$, where $\boldsymbol{h}_l$ is called the hidden state. Layers share the same parameters, and the number of layers can therefore be dynamically adjusted to the length of the input sequence. The last hidden state will, in theory, contain information about the whole input sequence. Additional layers can be appended to further process this feature vector and produce some desired output. Output sequences can be generated in a number of ways by setting the initial hidden state and then extracting the hidden state from different layers. RNNs themselves consist of a (dynamic) number of layers, but it is also possible to nest several RNNs. We then get what is called stacked RNNs.

RNNs are relatively deep networks and are therefore prone to what is called vanishing gradients. The gradients from the early layers become so small that they are ineffective in gradient descent. In other words, the first parts of the input sequence have too little influence over the end result. Therefore, variants of RNNs such as Long Short-Term Memory (LSTM) [57] and Gated Recurrent Units (GRU) [24] are often used in practice. They make sure that hidden states are more easily able to flow through the subsequent layers undisturbed, so that gradients will remain strong when backpropagated through many layers. Despite this improvement, the networks will still tend to be influenced more by the end of the input sequence then the beginning. It has become quite common to have bidirectional RNNs [117, 52], which can be seen as combining two RNNs, where one of them processes the input sequence backward.

Another popular way to face the issue of skew in influence for sequences is to use attention mechanisms [6]. The idea is to let the network itself choose what parts of the input to focus on, potentially for several iterations. This is typically achieved in a network by producing some normalized attention vector that is multiplied with the vector of interest.

While initially used as an enhancement to RNNs, networks based almost solely on attention [132] have recently started to proliferate [32, 109, 147] and are currently considered state-of-the-art for many, if not most, natural language

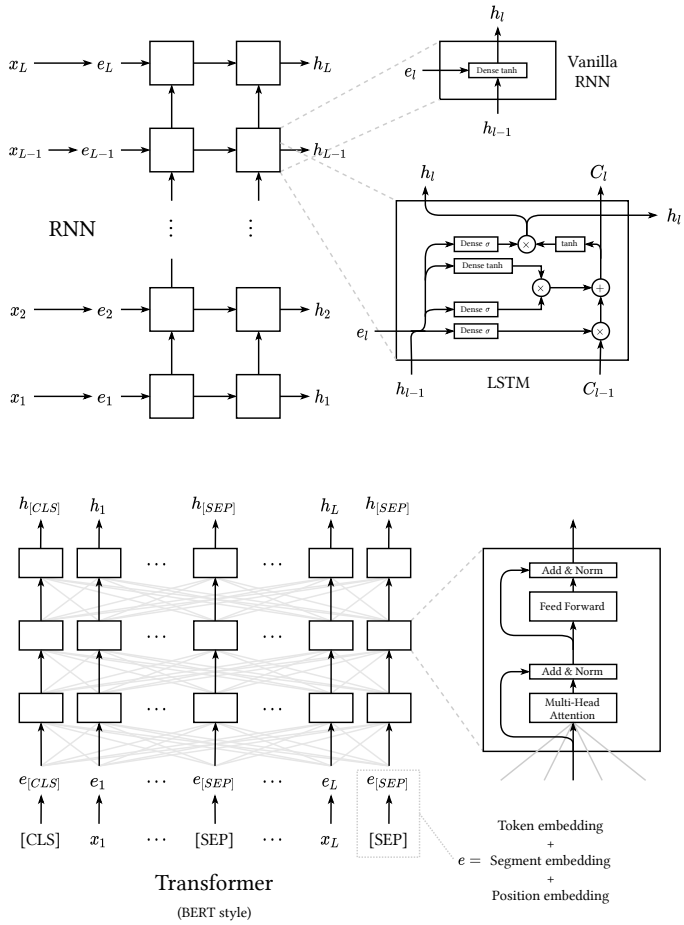


Figure 2.1: Illustration of the architecture for a two-stack uni-directional RNN encoder and a three-layer BERT-style [32] encoder for natural language processing. Let $(x_1, x_2, \ldots, x_l, \ldots, x_L)$ be the input sequence, and e_l be an embedding for x_l. Both a standard RNN and LSTM block are illustrated for the RNN architecture. Notice the additional *context* state C_l for LSTM, which can more easily carry gradients. Inspired by illustrations in [98, 132, 32].

processing tasks. We call these Transformer-based networks — as originally named by [132] that targeted machine translation. In contrast to RNN-based networks, they are not sequential with respect to the input sequence. See Figure 2.1 for an illustration of an RNN and Transformer encoder. This makes them more parallel, which again makes it easier to leverage modern, highly parallel hardware. In addition, one avoids prohibitive deep networks (due to vanishing gradients) for long input sequences. Each layer performs self-attention over the whole input sequence, effectively removing the long paths between cells of RNNs that makes it so hard to learn long-range dependencies. Since transformer networks are architecturally agnostic to the input sequence order, they are instead fed positional information through the input as positional embeddings.

One particular influential recent trend has been the ability to leverage huge pretrained models that have been trained unsupervised for language modeling on massive text corpora [105, 32, 109] — similar to what the computer vision community has done for a while. They produce contextualized word embeddings that take into account the surrounding words. The embeddings can be used as a much more powerful variant of the classical word embeddings, but as popularized by BERT [32], one can also fine-tune the network to the task at hand. Take BERT as an example. It is pretrained jointly on masked language modeling and next sentence prediction. Input during training is a special [CLS] token first, then the two sentences terminated by a special [SEP] token each. The [CLS] tokens output from the network is used to do the next sentence classification. Each token's embedding is augmented with a positional embedding and a segment embedding indicating which sentence it belongs to. This setup makes the network suitable for fine-tuning on both sequence labeling tasks as well as pair labeling tasks (such as question answering or entity matching).

2.3 Related work

2.3.1 Other surveys and extensive overviews

Given entity matching's long history, there is no surprise that it has been surveyed before in various ways, covering entity matching as a whole and more narrow aspects.

First, there are several books that provide an overview. Christen [25] is a dedicated and comprehensive source on entity matching, Naumann and Herschel [92] specifically cover the slightly more specialized problem of duplicate detection, and Batini and Scannapieco [12], Talburt [126], and Doan et al. [34]

all introduce entity matching in the context of data quality and integration. Second, the workshop tutorials by Getoor and Machanavajjhala [44] and Stefanidis et al. [121] serve as introductory summaries. Third, Elmagarmid et al. [40] present a literature analysis.

Other sources cover more narrow aspects of entity matching – such as specific techniques or subtasks. Quite early on, statisticians dominated the field of entity matching. Probabilistic methods were developed by Newcombe et al. [93] and given a solid theoretical framework by Fellegi and Sunter [42]. These probabilistic methods are summarized by Winkler [139, 140] and Herzog et al. [56]. Blocking, which is surveyed by Papadakis et al. [101], Christen [25], and Papadakis et al. [99], is considered an important subtask of entity matching, meant to tackle the quadratic complexity of potential matches. Christophides et al. [26] specifically review entity matching techniques in the context of big data. There has been an uptick in interest in both machine learning and crowdsourcing as a solution to entity matching in recent years. As part of a larger survey on crowdsourced data management, Li et al. [77] cover crowdsourced entity matching. Lu et al. [85] summarize the use of machine learning, while Gurajada et al. [54] present an overview of crowdsourcing, active learning, and deep learning for entity matching.

While earlier works mention or cover neural networks for entity matching to various degrees, we are to the best of our knowledge the first to present a dedicated, complete, and up-to-date survey.

2.3.2 Related problems

Entity matching can be seen as part of a larger group of tasks with roots in natural language processing that solve similar, but distinct, matching problems. Interestingly, but perhaps not surprisingly, deep learning-based methods have become state-of-the-art in all these tasks. We will briefly mention some of the most prominent ones.

- **Coreference resolution**: Given a text, find all mentions of entities and determine which mentions corefer. Two entity mentions corefer if they refer to the same entity [64]. In contrast, entity matching is concerned with more structured data with clearly distinct units of data (records). Importantly, entity matching does not have to take into account a larger textual context, which is necessary in coreference resolution to find coreferring mentions across multiple sentences. State-of-the-art methods are able to perform the whole task end-to-end using a deep network without

detecting and disambiguating mentions in two separate steps [75, 62].

- **Entity alignment**: Given two knowledge bases, find which entries across the two that refer to the same entity. Knowledge bases, in contrast to record sets in entity matching, have relations between entries. Leveraging these relations are central to the task. The way most neural-based methods do this is by producing so-called knowledge graph embeddings [155, 123, 23], embeddings of entries which incorporate information about their relationship to other entries.

  As a slightly specialized variant, user identity linkage is the problem of identifying which users across two social networks are the same [154].

- **Entity linking**: Given a text, find all mentions of entities and link them to entries in a knowledge base. One example of a heavily used knowledge base would be Wikipedia. In some ways, one can see entity linking as a hybrid between coreference resolution and entity alignment, and it differs from entity matching in the same ways. Neural-based methods are considered state-of-the-art [111, 68].

- **Paraphrase identification**: Given two texts, determine if they are semantically equivalent – i.e., if they carry the same meaning. This can be be seen as a generalization of string matching, if one interprets strings referring to the same entity as implicating that the strings are also semantically similar. Nonetheless, we still consider figuring out which texts convey the same meaning in general to be a distinct problem from entity matching. First, entity matching deals with more structured data. Second and most importantly, in entity matching, all records refer to an entity, and we are only concerned with which specific real-world entity a record is referring to. Any excess meaning carried by a record does not impact matching.

  Finally, there is also semantic textual similarity and textual entailment, which are closely related to paraphrase identification. Semantic textual similarity is concerned with the degree of how semantically similar two texts are, while textual entailment is about finding out whether one text semantically entails, contradicts, or is neutral to a second text. Additionally, in the case of multiple choice, question answering can also be seen as a matching problem.

  State-of-the-art for most of these matching problems rely on rather generic, but powerful, language understanding models [147, 32].

In a broader sense, similar problems are also studied in the context of information retrieval [89]. Neural networks not only provide effective techniques for retrieving unstructured text but also for data formats that have traditionally been less accessible such as images [82] — and even across modals [142].

2.4 The entity matching process

Traditionally, entity matching is often thought of as a process consisting of multiple steps, even though there is no generally agreed upon list of specific steps. It is useful to compare methods in light of how they relate to this abstract process. To this end, we introduce a high-level reference model describing the entity matching process as five distinct steps. These steps can also be viewed as a chain of the subtasks or subproblems that make up entity matching. Inspired by processes and figures such as those in [92, 25, 39, 26, 54], Figure 2.2 depicts this reference model of the traditional entity matching process. We will use the model to frame the discussion of different methods using neural networks.

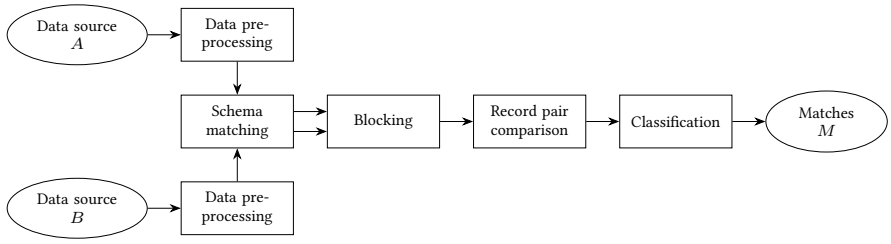


Figure 2.2: Illustration of the reference model for a traditional entity matching process and its five steps. Human-in-the-loop aspects are not considered.

The process adheres to the problem definition introduced above. It assumes two data sources as input. In theory, it could be generalized to multiple sources, but this is seldom done in the literature. A single source, as previously mentioned, can simply be seen as a special case. At the end of the process, the result is simply matches. Since this is an abstract process extracted from the literature, it is not necessarily followed step by step. The order might not be completely strict, and steps might be intermingled or skipped – as will be clear when we look at specific methods.

We also note that this process is machine-oriented and does not highlight

any iterative human interactions or feedback loops. Significant research has gone into both crowdsourcing [133, 135, 137, 50] and active learning [65, 5, 107]. Interestingly, Kasai et al. [65] use a deep neural network in their active learning approach. Such human-in-the-loop factors are often crucial for entity matching in practice [35]. We do not consider our proposed process to be in conflict with these aspects, but rather mostly orthogonal. Empirically, based on the surveyed methods, we do not find neural networks to be very tightly coupled to any human-in-the-loop techniques. We therefore focus on the machine-oriented aspects.

Data preprocessing

The first step in the process is data preprocessing, which is usually a crucial step in many data integration tasks [34]. The goal is to get both data sources into consistent and similar formats better suited for downstream tasks. Typical transformations may involve removing excess punctuation, lowercasing all letters, normalizing values, and tokenizing. Sometimes, one might also view this step as feature extraction, where records are transformed to a feature space. Preprocessing is, of course, very dependent on the domain and the specific data sources.

Schema matching

After preprocessing we perform schema matching, where the objective is to find out which attributes should be compared to one another, essentially identifying semantically related attributes. This will enable downstream steps to compare records across the two sources. Even though schema matching is often considered a separate problem to be solved before performing entity matching [e.g., 25], we choose to include it both because deep learning-based methods have the potential to perform it jointly with other steps (as a surveyed method shows [94]) and because it is frequently an unavoidable problem in real-world use cases for entity matching.

In practice, this step is often performed manually as part of the preprocessing step, simply making sure to transform both data sources into the same schema format. Traditional techniques for schema matching span a wide range of solutions. They can use both schema metadata and actual attribute values. Some are supervised learning methods, while others are unsupervised. Importantly, most of them are completely independent of downstream tasks in the process, though most techniques are actually not developed specifically for the

purpose of entity matching. For more in-depth coverage of schema matching, see Rahm and Bernstein [110], Bellahsene et al. [14], and Doan et al. [34].

Blocking

Since the number of potential matches grows quadratically, it is common to pick out candidate pairs $C \subseteq A \times B$ in a separate step before any records are compared directly. We call this step *blocking*, and the goal is to bring down the number of potential matches $|C| \ll |A \times B|$ to a level where record-to-record comparison is feasible. Note that in the literature, blocking is sometimes used as a name for only one of the possible strategies for avoiding the quadratic complexity [e.g., 26]. For simplicity, we refer to any effort to make record comparison feasible as blocking.

One can think of the blocking step as doing implicit comparison of records, while the comparison step described below is doing explicit comparison between pairs of records. There is often no way around performing at least some explicit pairwise comparison, since implicit comparison cannot offer the necessary precision. In certain cases, when the comparison lends itself to indexing, it is possible to fuse record pair comparison and blocking into one step. Usually, however, the explicit comparison is nontrivial, infeasible, or impossible to speed up by indexing – necessitating a need to prune away obvious nonmatches in a separate blocking step. This is possible because implicit comparison will typically have lower precision, but can be done more efficiently. At the same time, explicit comparison will typically have higher precision but has inherent quadratic complexity. By constructing a high-recall implicit comparison step to filter away obvious nonmatches first, we can make it feasible to use a more powerful high-precision explicit comparison afterward.

Typical techniques are based on hashing, sorting, or various ways of indexing. Some work completely independent from the downstream steps, while others are more coupled with the record pair comparison and classification steps. For example, if matches are decided based on thresholds of string similarity measures, it is often possible to specifically index attribute values to prune away according to that criteria [150]. Most techniques rely heavily on syntactic similarity, including those based on supervised machine learning. See Christen [25] and Christophides et al. [26] for extensive reviews on blocking techniques. In practice, it is not uncommon that blocking involves quite a bit of manual feature selection, picking out which attributes should be used and which technique to apply.

Record pair comparison

When the number of candidate pairs $|C|$ has been reduced to a manageable amount, we can compare individual records $(a, b) \in C$. The pairwise comparison results in a similarity vector S, consisting of one or more numerical values indicating how similar or dissimilar the two records are.

Traditionally, such comparisons have mostly been made using string similarity measures. These measures typically quantify a very specific syntactic similarity, and therefore differ in what clues for matching strings they are able to pick up. Some are, for example, good at adjusting for spelling errors or OCR errors. String similarity measures have been extensively covered before [40, 25, 85]. It is possible to incorporate domain knowledge in a string similarity measure to also perform semantic comparison instead of just syntactic [4, 119], but it is less common and introduces the extra challenge of acquiring such materialized domain knowledge.

String similarity measures are made to compare two strings and cannot be directly applied to a pair of records. Normally, one will compare those attributes which were found to be semantically similar in the schema matching step, thereby getting multiple measurements to include in S. Also, since the similarity measures are almost always static and only cover a narrow syntactic similarity, one can use multiple measures and offload the job of figuring out which ones to emphasize to the downstream classification step.

Classification

Lastly, the objective of the classification step is to classify each candidate pair as either match or nonmatch based on the similarity vector. In cases where $|S| = 1$, simple thresholding might be enough, while when $|S| > 1$, one needs more elaborate solutions.

Early efforts in entity matching were focused on unsupervised probabilistic methods for doing classification. Initially developed by Newcombe et al. [93] and later formalized by Fellegi and Sunter [42], the idea is that, given certain assumptions, one can calculate the optimal matching choice according to some bounds on false positives and negatives. It can be seen as very close to a näive Bayes classifier, classifying record pairs as either match, nonmatch, or uncertain – where the uncertain matches must go through manual review. The motivation is that common attribute values that agree (for example, a very common first name) are less significant than rare attribute values that agree. See Herzog et al. [56] for a complete introduction to probabilistic approaches.

Recently, methods based on rules or machine learning have been more prominent. Rules are predicates over the similarity vector S that flag them as match or nonmatch. They can be constructed manually, making them a powerful and highly explainable way of explicitly incorporating domain knowledge into the classification. Manually constructing rules requires a lot of expert labor, so significant work has been put into automatically learning rules from examples. Other efforts in leveraging learning have used off-the-shelf classification models such as decision trees and support vector machines. These machine learning models are then trained on examples of S for which it is known if they represent a matching or nonmatching record pair. Both rule-based and machine learning approaches are covered extensively in the literature [40, 34, 25].

Outline

Table 2.2 lists, to the best of our knowledge, all methods that use neural networks for entity matching and which steps of the process they tackle using neural networks. We will in the subsequent subsections take a closer look at each step and see how different methods use neural networks to handle them.

2.4.1 Data preprocessing

Deep neural networks are good at doing representation learning. As we will see, they can therefore effectively learn to do some of the data preprocessing we would traditionally do manually. When we explore how the different methods do this, we will focus on two aspects: How embeddings are used to get records in a suitable input format, and how the networks' hierarchical representations are structured.

Embeddings

Neural networks alone only work with numerical data, so an important enabling factor in letting networks learn representations is how textual records are transformed into a numerical format. In practice, this is done using embeddings, as explained in Section 2.2. Note that while some embedding models, like GloVe [104], are not neural networks, we still consider them a crucial component for neural networks and how they are able to replace manual feature extraction (see Section 2.5). They perform and enable powerful representation learning on text. Other embedding models, like word2vec, can be seen as a simple neural network. Even though the embeddings are later used in a lookup table, they

Table 2.2: Overview of which steps of the entity matching process reference model different methods tackle with neural networks.

Method	Data preprocessing	Schema matching	Blocking	Record pair comparison	Classification
SEMINT [79, 80]		•			
SMDD [149]		•			
Nin and Torra [95]		•		∼	
Pixton and Giraud-Carrier [106]					•
Wilson [138]					•
Tran et al. [130]					•
NNSM [151]		•			
Gottapu et al. [49]	•				•
Reyes-Galaviz et al. [113].					•
Kooli et al. [70]	•			•	•
DeepMatcher [91]	•			•	•
Wolcott et al. [141]	•			•	
DeepER [39]	•		•	•	∼
MPM [43]	•			•	•
Kasai et al. [65]	•			•	•
Seq2SeqMatcher [94]	•	•		•	•
Nozaki et al. [96]	•	•			
AutoBlock [96]	•		•		
Hi-EM [153]	•			•	•
Brunner and Stockinger [20]	•	•		•	•
Ditto [81]	•	•		•	•

were trained using this simple network. One interesting use of word2vec is that of Nozaki et al. [96]. They do not use the word embeddings as input to a neural network, but use them as is in a simple aggregation and comparison scheme to do schema matching (details in Section 2.4.2).

Granularity Embeddings can be used at different granularities, usually at word- or character-level. The second column of Table 2.3 shows which methods use which granularity. Word embeddings significantly reduce the length of the sequences to be processed compared to character embeddings but come at the expense of increasing the number of unique values that have to be represented by many orders of magnitude. This often makes solutions relying on word embeddings more vulnerable to out-of-vocabulary (OOV) words – i.e., words that were not present in the training data. Word-based embedding models usually handle unknown words by assigning the same embedding to all unknown words, making no distinction between them. When embeddings are pretrained on large

Table 2.3: Overview of how the surveyed methods use embeddings, specifically at what granularity, if they use pretrained embeddings, and whether they fine-tune embeddings. Surveyed methods not using embeddings at all are left out. $^{+}$Other options were tried, but this was found to be most preferential. *The method uses pretrained embeddings for the attribute value text, but standard embeddings trained from scratch for attribute labels.

Method	Embedding granularity	Pretrained embeddings	Fine-tuned embeddings
Gottapu et al. [49]	Word	No	-
Kooli et al. [70]	Word & Character N-gram	Yes	No
DeepMatcher [91]	Word & Character N-gram^{+}	Yes^{+}	No
Wolcott et al. [141]	Character	No	-
DeepER [39]	Word	Yes	Yes^{+}
MPM [43]	Word & Character N-gram	Yes	No
Kasai et al. [65]	Word & Character N-gram	Yes	No
Seq2SeqMatcher [94]	Word & Character N-gram	Both*	No
Nozaki et al. [96]	Word	Yes	No
AutoBlock [152]	Word & Character N-gram	Yes	No
Hi-EM [153]	Character	No	-
Brunner and Stockinger [20]	Character N-gram	Yes	Yes
Ditto [81]	Character N-gram	Yes	Yes

general corpora (as will be discussed next), but the data sources at hand contain domain-specific words that are otherwise rare, they will naturally tend to be less useful. In addition, the data sources at hand might have low data quality and contain typos or small spelling variations that are not common in the training data – thus effectively making those words out-of-vocabulary. Motivated by these concerns, the majority of the methods use fastText [16] embeddings (or similarly, Wordpiece/SentencePiece/Byte-Pair-Encoding [117, 72, 118] for the transformer networks). FastText combines embeddings for both the word itself and all character N-grams of certain lengths, often making it possible to find a suitable representation for an OOV word, since the word most likely has known N-grams. Using N-grams in this way is basically a way of approximately incorporating a morpheme granularity level to word-level embeddings [16].

Does the choice of embeddings matter? Mudgal et al. [91] compare fastText to (the purely word-based) GloVe and find fastText to have an edge when the

data sources contain domain-specific words that are OOV and otherwise comparable. Ebraheem et al. [39], meanwhile, compare fastText without N-grams, GloVe, and word2vec [88], reaching the conclusion that there is no significant difference. The combined results might indicate that the embedding granularity is more important than which particular embedding is used.

Pretraining One of the benefits of popular word embeddings models like word2vec, GloVe, and fastText is that you can get pretrained embeddings. They have been trained on enormous general corpora, have vocabularies of significant size, and are often available for different languages. Pretrained character embeddings are not nearly as common, though Zhao and He [153] are, in essence, pretraining the entity matching model on large amounts of training data and can in a way be thought of having pretrained character embeddings. The third column in Table 2.3 shows which methods use pretrained embeddings. Using pretrained embeddings is essentially a way of doing transfer learning for feature extraction. Since embeddings can be trained unsupervised, there is generally a substantial amount of training data available. This can be very helpful if it manages to reduce the necessary amount of labeled training data for the downstream entity matching task at hand. Mudgal et al. [91] found that learning embeddings from scratch instead of using pretrained embeddings can be favorable to highly specialized data sources, while for other data sources, pretrained embeddings either outperformed or were comparable to learning from scratch.

Fine-tuning Even when embeddings have been pretrained on some large text collection, one still has the opportunity to continue adjusting them when doing the task-specific training together with all other weights. We refer to this as *fine-tuning* the embeddings – the opposite of *freezing* them. The fourth column in Table 2.3 shows which methods fine-tune their embeddings, which currently is only Ebraheem et al. [39]. They found fine-tuning to help on hard data sets — i.e., those that are very challenging or impossible to get close to perfect F_1 score on.

Representation levels

Embeddings offer neural networks an initial mapping from the actual input to a suitable numeric representation. But as mentioned earlier, the strength of deep learning's use of neural networks is really its ability to do hierarchical representation learning, which is achieved using multiple layers, learning increasingly

abstract features [74]. The first layers of deep networks will typically be designed to enable building a good representation of the input, and then let the last layers focus on producing the desired output. It is nontrivial to figure out what each layer actually learns. When we compare the surveyed methods, we instead focus on explicit levels of representation.

Table 2.4 highlight how each method does representation learning by listing which explicit representation levels are used, and which techniques are used to build representation from the level below, where the explicit level units are character, word, attribute, and record. It is important to note that the table does not reflect the entire neural network of each method, but rather only the beginning layers that are to be considered as the feature extraction part of the network. We consider a representation level as used if you can simply pick out a vector representation for units of that level after some layer. A vector is considered to represent a unit if its calculations only rely on that unit or other input through an attention mechanism. Importantly, a vector that relies on two records through something else than an attention mechanism is not considered a representation, but rather a comparison (see Section 2.4.4). Figure 2.3 illustrates the difference in terms of computational graphs. Of course, with neural networks, the actual line between the initial feature extraction part and the rest is an artificial one and not necessarily indicative of how the networks actually learn and work. But they do reflect design decisions to a certain degree and help us compare them in that regard.

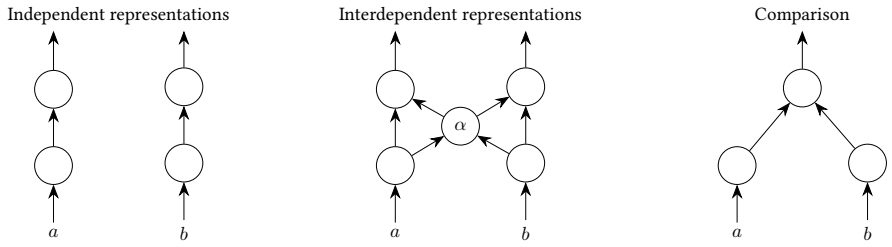


Figure 2.3: Illustration of what is considered a vector representation (independent or interdependent) and what is considered a comparison in terms of computational graphs. Here, a and b are records and α is an attention mechanism.

We see each method's first layer is (unsurprisingly) an embedding, providing initial character or word vectors. Some use a specific embedding model, like

Table 2.4: Overview of which explicit representation levels the surveyed methods make use of and which kinds of network layers are used to build representation from the level below. Methods upper half use independent record representations, and those in the bottom half use interdependent record representations. Self-attention is any attention mechanism that only uses elements within the same record, while cross-attention refers to any attention mechanism looking across two records. $^{+}$Other options were explored, but the representational power was similar or lower. *Multiple options in use at the same time.

Method	Character	Word	Attribute	Record
Gottapu et al. [49]		Standard embeddings		1 Convolutional
Wolcott et al. [141]	Standard embeddings			1 BiLSTM, 2 FC
DeepER [39]		GloVe^{+}		1 BiLSTM^{+}
MPM [43]	*	fastText*	*	
Kasai et al. [65]		fastText	1 BiGRU	
Nozaki et al. [96]		word2vec	Sum, average	
AutoBlock [152]		fastText	1 BiLSTM^{+}, self-attentation	Weighted sum
DeepMatcher [91]		fastText^{+}	Cross-attention, 1 BiGRU^{+}	
Seq2SeqMatcher [94]		Standard embeddings & fastText		Cross-attention
Hi-EM [153]	Standard embeddings	1 BiGRU, cross-attention, self-attention	1 BiGRU, cross-attention, self-attention	Concatenation
Brunner and Stockinger [20]	Byte-pair encoding^{+}, 12 transformer layers (self- and cross-attention)$^{+}$			
Ditto [81]	Byte-pair encoding^{+}, 12 transformer layers (self- and cross-attention)$^{+}$			

fastText, while others just use standard lookup table embeddings that they train themselves. Next, we note the popularity of RNN-based models among the methods, which is in line with the widespread use of such sequence-aware models in natural language processing [e.g., 51, 125, 73]. An interesting case is that of MPM [43], which actually combines two versions of DeepMatcher [91] as well as classical similarity measures in its architecture.

The methods can be naturally divided into two categories when it comes to representation learning: independent or interdependent representation. If the highest representation level relies on a record pair instead of a single record, we say it is an interdependent representation. Otherwise, it is an independent representation. See again Figure 2.3 for an illustration. The methods in Table 2.4 have independent and interdependent representation at the top and bottom, respectively. Interdependent representations are, in essence, a way to incorporate record pair comparison into the feature extraction. They have the benefit of being able to adapt based on what they will be compared to, while independent representations have the benefit of not relying on record pairs to be computed. The latter will be important when we discuss blocking in Section 2.4.3.

Independent representation There is significant variation among the methods with independent representation. Kooli et al. [70] and Nozaki et al. [96] mostly rely on word embeddings for the feature extraction part of the network. Kooli et al. simply concatenate them before the next layers do comparison, and Nozaki et al. aggregate them through summation and averaging. Wolcott et al. [141] use bidirectional LSTM on character embeddings followed by dense layers to produce record-level representations. As the only method, Wolcott et al. actually go straight from characters to record representation. Kasai et al. [65] use bidirectional GRU on word embeddings to get an attribute-level representation. As the only surveyed method, Gottapu et al. [49] apply a simple convolutional layer to word embeddings. Lastly, both DeepER [39] and AutoBlock [152] have networks solely aimed at finding good representations for use in blocking (see Section 2.4.3). DeepER uses bidirectional LSTM on word embeddings to get a record-level representation (but also show a simple averaging approach is competitive). Somewhat differently, AutoBlock applies bidirectional LSTM and self-attention on word embeddings to get an attribute representation and represents records as a weighted sum of attributes.

Interdependent representation DeepMatcher [91] explores several ways of building attribute representation from word embeddings. The one with the

highest representational power, Hybrid, uses a combination of bidirectional GRU and decomposable attention [103] across records. Unique among the surveyed methods, Seq2SeqMatcher [94] structures records as sequences of `(attribute, word)` pairs. The embedding of such a pair is a concatenation of a custom embedding for the attribute and a fastText embedding for the word itself. The record-level representation is produced through an attention technique between the sequences of two records. Brunner and Stockinger [20] treat a record pair as a sequence of attribute value sub-word tokens, while Ditto [81] model record pair as a sequence of alternating attribute name and value sub-word tokens. Both let each token keep its own representation throughout the representation building layers. These Transformer networks take interdependent representation to an extreme, as each token depends on all other in every Transformer layer. Finally, Hi-EM [153] is the only method which uses all the four explicit representation levels. It applies a combination of bidirectional LSTM, self-attention within the record, and attention across records — both from its standard character embeddings to word vectors and from its word vectors to its attribute vectors. For the record-level representation, it simply concatenate the attribute vectors.

2.4.2 Schema matching

Given two data sources A and B, we divide the ways in which the schemas can be related in three:

- **Aligned schemas**: Both data sources use the same schema. In other words, $\forall_{i \in \{1,2,\ldots,n\}}(A_i = B_i)$ and $n = m$.
- **Misaligned schemas**: Both data sources have the same attributes, but not in the same order. In other words, there exists a bijective relation $H \subset \{A_i\}_{i=1}^{n} \times \{B_j\}_{j=1}^{m}$ such that $\forall A_i B_j : ((A_i, B_j) \in H) \rightarrow (A_i = B_j)$.
- **Incompatible schemas**: There is no simple correspondence. In other words, there does not exist such a bijective relation as described above.

With aligned schemas, there is no need for schema matching. For misaligned schemas, finding a one-to-one correspondence between attributes is sufficient, while in the general case of incompatible schemas, more complex connections must be uncovered. Many schema matching techniques are concerned with the former case, misaligned schemas. For entity matching, the goal is usually to find out which attributes should be compared in the downstream task where records are compared; we want to find the pairs (A_i, B_j) of attributes that are

semantically related. So one attribute can be compared to several attributes from the other data source – implying incompatible schemas.

An additional challenge that might occur is dirty attribute values – values that should have been in another attribute [91, 94]. In such cases, we need to compare attributes that might not necessarily be semantically related in order to be robust to noise.

There are two sources of information when doing schema matching. There are the actual attribute values in records, and there is the attribute metadata. Attribute metadata will often simply be a name (e.g., `title`, `author`, etc).

When it comes to neural networks for schema matching, there are essentially four approaches in the surveyed methods:

Learn attribute matching from clusters

SEMINT [79, 80], SMDD [149], and NNSM [151] specifically target schema matching. They all first create training data by performing unsupervised clustering of attributes, and then use that data to train a multilayered perceptron[2] (MLP). SEMINT uses a Self-Organizing Map [67] to cluster the feature vectors of attributes in data source A into categories. The attribute features are handcrafted and are based on both schema metadata and attribute values. The category clusters are then used as labeled data to train an MLP with one hidden layer. Given an attribute feature vector, the network scores its similarity to these cluster categories, and this is used to match the attributes of data source B to the categories of A. SMDD follows a similar strategy, but uses the distribution of attribute values and a hierarchical clustering technique. Somewhat differently, NNSM clusters pairs of attributes into either being similar or dissimilar based on similarity scores of four traditional matchers. Next, they train an MLP with two hidden layers to classify a pair of attributes as either similar or dissimilar using the clusters as training data.

An interesting aspect of these schema matching methods is their lack of need for human-labeled training data. The methods that learn from clusters generate training data by using more traditional unsupervised manual methods. As Zhang et al. [151] explains it, they are essentially using neural networks as a way to combine several traditional methods.

[2] A multilayered perceptron is a simple feedforward network using only fully connected layers.

Learn schema mapping

Nin and Torra [95] translate records from source A to records following the schema of B. They train a network for each attribute in B, which can translate a record $a \in A$ to a record of B's schema. Working only on purely numeric data, they are able to simply use records from A as input and output values as the translated record from a neural network. The network effectively transforms incompatible schemas into aligned schemas. This approach can resolve the schema matching problem for downstream tasks but also does a lot of the heavy lifting of the record pair comparison by attempting to project records from A to corresponding records in B.

Compare attribute representations

Nozaki et al. [96] do schema matching by thresholding the cosine distance between attribute vectors. The attribute vectors are found by first summing up the pretrained word embeddings for each attribute in each record and then simply averaging per attribute across all records. Even though it relies on pretrained word embeddings, the method itself is unsupervised. The distance threshold was simply found experimentally.

Learn jointly with comparison and classification

While schema matching has traditionally been dealt with as a separate task, as with the methods above, Nie et al. [94], Brunner and Stockinger [20], and Li et al. [81] incorporate it as part of their deep learning approach for comparing and classifying record pairs.

As explained in the previous subsection, Seq2SeqMatcher [94] structure records as sequences of `(attribute, word)` tokens, and then solve entity matching as a sequence matching problem. The embedding of such a token is a concatenation of a custom embedding for the attribute and a fastText embedding for the word itself. Notice that no attribute metadata is used. Treating the input in this way enables the neural network to learn how to compare values across attributes. Specifically, the authors use a bidirectional attention mechanism between token embeddings from two records, and then use only the max k attention scores to get the soft-attended representation of a token. Using only the k largest attention scores, effectively setting the rest to zero, helps the model compare only relevant tokens and ignore irrelevant tokens.

Brunner and Stockinger [20] preserve no information about the attributes other than the order (which, of course, may differ across schemas), and simply

treat a record as a sequence of sub-word tokens. They rely entirely on the powerful attention mechanism in the Transformer network to do the schema matching using positional information provided through the input and whatever insight and correlation the attribute values provide. Ditto [81], on the other hand, explicitly incorporate attribute name sub-word tokens in the input sequence, which gives more information to the Transformer network to perform schema matching. In contrast to Seq2SeqMatcher, Ditto uses the attribute name instead of a randomly initialized embedding — enabling the network to exploit knowledge from its language modeling pretraining that the attribute name might signal.

2.4.3 Blocking

Few methods try to use neural networks for blocking, as seen in Table 2.2. The only two methods, DeepER [39] and AutoBlock [152], embed records into a high-dimensional metric space and then do nearest neighbor search to filter down the cartesian product $A \times B$ to a candidate pair set C. They both use cosine distance as a metric, and the networks are implicitly trained to produce record representations close to each other for matching records and far from each other for nonmatching records. Finding the nearest neighbors in high-dimensional spaces is computationally infeasible, so to make it more feasible, they perform approximate nearest neighbor search. Then there is no guarantee to find the nearest neighbors, but rather a high probability. Both methods do this using locality sensitive hashing (LSH) [58], which is a well-studied technique [136].

The two methods follow the same high-level strategy, but they have some important differences. The networks themselves that are responsible for building a good record representation are differentiated in Section 2.4.1. DeepER trains its network end-to-end with comparison and classification of record pairs. The record representations are compared using either elementwise subtraction or multiplication, and then a dense layer performs the classification. AutoBlock, in comparison, trains specifically for blocking with a custom loss applied directly to the cosine distance between records. For the actual nearest neighbor pruning, they use two different LSH methods. DeepER uses hyperplane LSH [21, 60], a well-studied method that is known to be easy to implement and often fast in practice. AutoBlock uses cross-polytope LSH [3], which has the benefit of theoretically optimal query running time while also being efficient in practice. Both use multiprobing with distance 1.

2.4.4 Record pair comparison

Central to matching is to assess the similarity of two records, both syntactically and semantically. The surveyed neural networks will generally produce some distributed representation at either attribute- or record-level and then compare the representations. We consider the layers in a network that are responsible for reducing from representations per record to representations across records appropriate for classification as record pair comparison layers. Or, to put it simply, those layers producing the similarity vector S from per-record representations. We will look at three central characteristics of how comparison is performed:

Table 2.5: Overview of how the surveyed methods perform record pair comparison. $^{+}$Other options were tried, but this was found to be preferential. *The most expressive model (BiLSTM-based) does non-attribute-aligned comparison, while the simpler averaging model is attribute-aligned.

Method	Attribute-aligned	Distributed similarity	Attention-based
Kooli et al. [70]	No	Yes	No
DeepMatcher [91]	Yes	Yes^{+}	Words^{+}
Wolcott et al. [141]	No	No	No
DeepER [39]	No*	Yes	No
MPM [43]	Yes	Both	(Partially) words
Kasai et al. [65]	Yes	Yes	No
Seq2SeqMatcher [94]	No	Yes	Words
Hi-EM [153]	Yes	Yes	Characters, words
Brunner and Stockinger [20]	No	Yes	Character N-grams
Ditto [81]	No	Yes	Character N-grams

Attribute-aligned comparison

If one assumes the two data sources A and B to have aligned schemas, one can compare attributes in a one-to-one fashion. We then say the comparison is attribute-aligned. The alternative is to perform comparison at record level, as one will be less dependent on the schemas to be aligned. The second column in Table 2.5 shows which of the surveyed methods do attribute-aligned comparison. DeepMatcher [91] and Kasai et al. [65] both compare attributes

one-to-one before they combine the similarity representation to record level. To handle cases where non-attribute-aligned comparison is necessary because the data is dirty and are partially put in wrong attributes, DeepMatcher merges all attributes to one long string attribute – essentially reducing the problem to a string matching problem. This is, of course, something most attribute-aligned methods can do to overcome this restriction, but then the information carried by the attribute separation is lost. Hi-EM [153] does actually not align the comparison of attribute representations, but the attribute representations have been produced by implicitly comparing characters and words through an attention mechanism across aligned attributes.

Distributed similarity

When two distributed representations are compared, one can either produce another distributed representation for the similarity of them or reduce the representations down to a nondistributed similarity representation – usually a scalar. The third column in Table 2.5 shows which of the surveyed methods make distributed similarity representations. As can be seen, the majority of the surveyed methods do. Typical ways of computing these similarities include vector difference, Hadamard product, or concatenation. The only with nondistributed similarities, Wolcott et al. [141], use cosine distance to compute the similarity.

Nondistributed similarities have the benefit of reducing complexity and training time, but at the cost of expressiveness. The increased expressive power of distributed similarities has to be matched by a classifier able to use it. Mudgal et al. [91] reported that distributed similarities outperform nondistributed. In addition, they found vector difference to be significantly better than concatenation when used after representation layers that do not use cross-attention. MPM [43] stands out since it combines both distributed and nondistributed similarity. It uses multiple classical similarity measures and two versions of DeepMatcher [91] in parallel and let the network effectively choose a similarity representation through a softmax.

Cross-record attention

As we saw in Section 2.4.1, some methods build distributed representations that are dependent on the record to be compared to through attention mechanisms. They are essentially peeking at what is to come, which enables them to focus on what is important for the comparison. The fourth column in Table 2.5 summarizes which methods use cross-attention and at which representation lev-

els. DeepMatcher [91] uses attention between words across records, while Hi-EM [153] uses attention between both character- and word-level representations across records. Both restrict their attention mechanism within each attribute, since the comparison is attribute-aligned. In contrast, Seq2SeqMatcher [94], Brunner and Stockinger [20], and Ditto [81], are able to use attention between all sub-words/words across the compared records. The Transformer networks take cross-attention all the way by relying almost exclusively on attention throughout the architecture and not making any distinction between self-attention and cross-attention.

2.4.5 Classification

Table 2.6: Overview of which neural network layers the surveyed methods use for classification. *The authors emphasize that any machine learning classifier can be used and do not explicitly favorize a neural network.

Method	Classification layers
Pixton and Giraud-Carrier [106]	Two linear layers with custom sparsity pattern, threshold on ratio of match and mismatch score
Wilson [138]	Single perceptron, threshold
Tran et al. [130]	MLP, softmax
Gottapu et al. [49]	Linear layer, softmax
Reyes-Galaviz et al. [113]	Two-layered MLP with custom sparsity pattern, threshold
Kooli et al. [70]	MLP, LSTM, or CNN
DeepMatcher [91]	Two-layered MLP with Highway-connections [120], softmax
DeepER [39]	Single dense layer*
MPM [43]	Two-layered MLP with Highway-connections [120], softmax
Kasai et al. [65]	Two-layered MLP with Highway-connections [120], softmax
Seq2SeqMatcher [94]	Two-layered MLP, softmax
Hi-EM [153]	Single dense layer
Brunner and Stockinger [20]	Single dense layer, softmax
Ditto [81]	Single dense layer, softmax

Compared to the other steps, there is less variance in how the surveyed methods perform classification. Generally, they take in a similarity vector S and do binary classification. Deeming a record pair matching or not. As an

exception to this approach, Gottapu et al. [49] classify records $a \in A$ directly to a corresponding record $b \in B$, treating matching the problem as a multiclass classification with $|B|$ classes.

S can be from a separate procedure, like string similarity measures, or upstream layers in the same neural network. The first was more common in earlier methods, while the former is common among newer deep learning methods. Nonetheless, the actual networks or layers used for classification are relatively similar. Table 2.6 shows how each method's classification network is built – either standalone or as the final layers of a larger network. We see that most are variations of the same theme of MLP with softmax at the end.

2.5 Contributions from deep learning

In the previous section we dissected the use of neural networks in all the surveyed methods using our process reference model. We now take a step back and summarize which contributions deep learning provide entity matching. Initially when neural networks were applied for entity matching, they were used merely as a classifier over feature vectors, either for schema matching or determining if pair of records matched or not. In the past few years, following the rise of deep learning, we have seen not only an increase in the use of neural networks, but also a broadening of the role they play in the entity matching process.

2.5.1 Learned feature extraction and comparison

Traditionally, as part of the preprocessing in entity matching, it is common to transform the data through a range of handcrafted procedures to a format more suitable for comparison. Important features are made more prominent and accessible to the steps downstream, while less important features are filtered away. Examples include phonetic encoding, removing punctuation, stemming, and expanding common domain-specific abbreviations. The problem is that these procedures are highly dependent on the data sources, and an expert has to decide which features are essential and how to extract them. Such customization makes it harder to scale a solution across data sources and use cases.

In contrast, driven by the advances of deep learning for natural language processing, more recent methods are able to learn feature extraction from less preprocessed records. As mentioned in Section 2.4.1, embeddings are used as a powerful gateway to letting neural networks work with text, while hierarchical representation learning and sequence models make it possible to make

semantically rich distributed representations of attribute values and records. This alleviates the need for much of the ad-hoc handcrafted feature extraction procedures, leaving mostly standardized feature extraction steps such as simple tokenization. This is in line with the development in other domains using deep learning, where deep neural networks have been increasingly able to replace complex handtuned feature engineering [74]. Of course, it does not replace all forms of preprocessing when data sources in different formats and of different origins are to be matched. One might, for example, need to extract records from some nonstandard XML or JSON format. While current deep learning methods do not remove the need for such handcrafted ad-hoc preprocessing, feature extraction can still be a very labor-intensive part of the preprocessing, and so deep learning has the potential to lighten the manual load drastically.

In addition to feature extraction, a point of significant handcrafted complexity is that of string similarity measures. Traditionally, one would use several string similarity measures between two records to produce a similarity feature vector and then use a neural network, some other classification model, or rules to classify the pair as matching or nonmatching based on this similarity vector. Doing comparison together with feature extraction in a deep neural network, we are able to effectively learn how to do comparison of records. The network will learn to extract features that are suitable for comparing records, either for static comparison or for downstream layers in the network. In cases where cross-attention is used, the network is even able to learn to extract features tailored to be suitable for comparison against a specific record. Compared to traditional approaches, learning comparison in this way can remove the need for these complex handcrafted similarity measures. Also, since deep networks can produce powerful distributed representations, this approach opens up a straightforward way to perform semantic comparison. This means one can depend less on syntactic similarity, which is hard to do with handcrafted methods.

Transfer learning has been a catalyst to being able to train more powerful deep learning models with fewer labeled examples for the task at hand in both natural language processing [115] and computer vision (where starting out with large networks [e.g., 55] pretrained on enormous data sets like ImageNet [31] is common). Deep learning methods for entity matching have also started to use such opportunities. Using some of the readily available pretrained word embeddings is popular, and is used as an efficient way of transfer learning from large general corpora. Thirumuruganathan et al. [129] outline different ways to transfer learn between entity matching task using weigted sums of word embeddings as distributed record representations. Some methods also use transfer learning beyond word embeddings and pretrain models specifically for entity

matching [65, 153]. Furthermore, most recently, we have also seen the use of large, powerful, general pretrained language models that are fine-tuned to entity matching [20, 81]. Compare this to traditional methods, which do not typically incorporate transfer learning. While one can argue that this is to be expected, since deep learning methods will usually require more training examples, the use of deep networks makes it possible to realize very powerful transfer learning scenarios, which are hard to pull off with traditional methods.

2.5.2 Coalescing the entity matching process

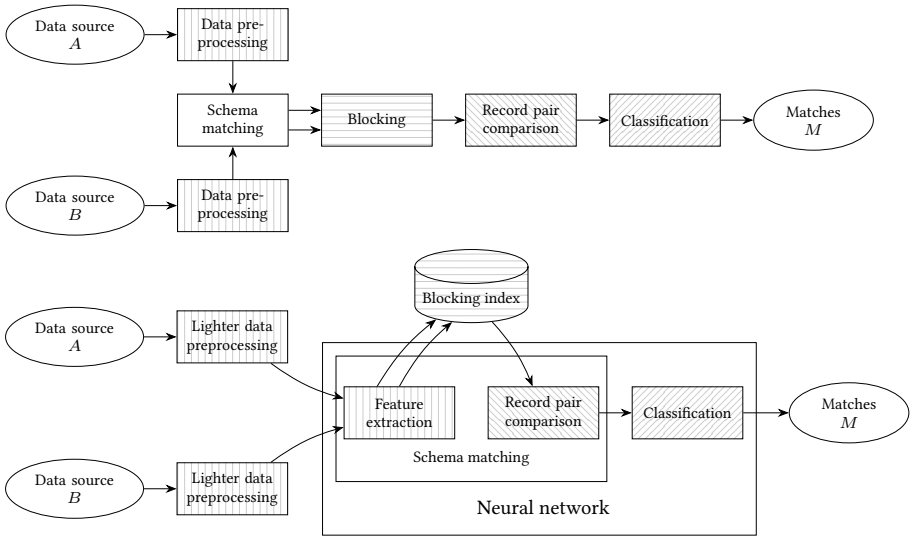


Figure 2.4: Illustration of the reference model for a deep learning entity matching process (bottom) together with the reference model for the traditional entity matching process (top) and how they relate to each other.

In Section 2.4, we introduced the reference model of the traditional entity matching process. To support our discussion, we introduce a reference model for a deep learning-based entity matching process. Figure 2.4 depicts this reference model together with the traditional variant while highlighting how steps correspond between them. One should note that none of the surveyed deep learning methods actually follow this exact process. It is the essence of all the methods

summed up in one view.

What all surveyed deep learning methods have in common is that they have fused together feature extraction, record pair comparison, and classification in a single step as a neural network. The need for data processing as a separate step still exists, but with less focus on feature extraction. In some ways, this development comes naturally. The steps are carried out in more or less the same order, and one can (to a certain degree) distinguish between them in the neural network architecture. But as we saw in Section 2.4, some methods have also explored using neural networks for schema matching and blocking.

Nie et al. [94] is the first deep learning method to incorporate schema matching in an end-to-end fashion together with record pair comparison and classification. They essentially construct the feature extraction and record pair comparison part of the network in such a manner that it is possible to learn schema matching – reducing the step to a built-in property of the network – in effect, jointly training a model for both matching schemas and records. Compared to solving schema matching as a separate task upfront, this has the potential benefit of being able to adapt how the schemas are matched to how they are used downstream. Other deep learning methods assume the schema to be the same for both data sources, but to what degree and how this assumption is manifested in the method differ (see Section 2.6).

The surveyed methods tackling blocking with deep learning [39, 152] rely on making a distributed representation of a record with a neural network, and then finding the candidate pairs with approximate nearest neighbor search. In other words, the network produces indexable feature vectors, and through the use of a suitable index, we find similar records in subquadratic time. While it does not remove the blocking step, it does reduce the core blocking mechanism to an indexing problem over some standard metric. The network will learn how to do blocking by learning how to represent similar records close to each other in the metric space, while the nearest neighbor search will always remain the same. If one uses the same distributed representations downstream for record pair comparison, it also serves as a good way to align how blocking and record pair comparison evaluate similarity. It is not unusual in traditional methods that these two steps, who both at their core do record comparison, have two considerably different ways of measuring similarity.

In computer vision and natural language processing, one has traditionally operated on large, complex, and heavily engineered pipelines. They are often divided into distinct steps that have been worked on separately. In contrast, deep learning methods in these fields have evolved to do what previously was done in several distinct steps in one go with a deep network [e.g., 125, 112],

making it possible to train on tasks end-to-end. In a similar fashion, we can see deep learning increasingly reshaping the entity matching process by coalescing traditional steps. This has at least two immediate benefits. First, it effectively reduces the number of steps in the process, and second, it makes it possible to train an increasing part of the process end-to-end. The main enabling factor is the representation learning in modern deep learning techniques for natural language processing, which is able to tie all of these together. As we have seen, powerful feature extraction can remove much of the need for manual feature engineering in schema matching, blocking, and record pair comparison. When in addition these steps are able to share the feature extraction and become part of an end-to-end network covering a large portion of the entity matching process, it can be translated to a potentially great reduction in complexity when building entity matching pipelines. Because the complexity of the process lies not only in the individual steps, but also in the interaction between them. Tuning and getting multiple heavily engineered steps to work smoothly together can be difficult.

2.6 Taxonomy of deep neural networks for entity matching

We have seen how neural networks serve different purposes in the entity matching process and vary in how they do so. Deep learning has led to an increase in the use of neural networks for entity matching in the past few years, making deep networks the norm. All of these deep learning methods have in common that they reduce the need for tedious handcrafting of features, but how the networks are structured at a high level differs in important ways, which impact their ability to interact with other steps in the entity matching process. The schema matching and blocking steps are especially interesting in this regard, since they have traditionally been solved with specialized methods separate from record pair comparison and classification. With this in mind, we propose a taxonomy of deep neural networks for entity matching consisting of four categories. There is no concrete definition of what constitutes a *deep* neural network, so for the purpose of this taxonomy we only consider networks that do feature extraction[3] (basically those who are marked for the data preprocessing step in Table 2.2). The categories are decided by two binary properties that were introduced in

[3]Note we also exclude Nozaki et al. [96], since the method do not represent a neural network itself.

Section 2.4: 1) Whether comparison is attribute-aligned or not, and 2) whether representations used for comparison are independent or interdependent. Figure 2.5 shows the taxonomy and which category each surveyed deep neural network falls into. In addition, it highlights how schema matching and blocking are related to the categories and which of the methods leverage their neural networks to tackle these steps. The two properties do not only reflect the high-level structural properties of the network, but also the assumptions of the problem to be solved. To help illustrate the taxonomy, we also show one representative deep learning network architecture for each category in Figure 2.6.

2.6.1 Attribute-aligned or non-attribute-aligned comparison

If the network is structured in a way that assumes records from the two data sources to have aligned schemas and constrain record comparison to one-to-one on attributes, we say it has attribute-aligned comparison. If it does not, we say it has non-attribute-aligned comparison. Notice that comparison does not necessarily need to be explicit record pair comparison, but can also be implicit comparison through, for example, indexing of distributed representations.

Attribute-aligned comparison is a powerful way to incorporate prior knowledge into a method, potentially making training easier and more efficient. At the same time, it prevents the possibility of performing schema matching. Non-attribute-aligned comparison does not necessarily imply the method is being used for schema matching, but that it is in its nature more compatible and should be easier to adapt to performing it.

2.6.2 Independent or interdependent representation

If the network relies on seeing a pair of records that are to be compared to produce representations of the records, we say it generates interdependent representations. Otherwise, we say it generates independent representations.

Interdependent representations have the benefit of being able to observe what they are being compared to. Intuitively, it is easier to compare two things if we can look at both at the same time. If we are only allowed to look at one thing at a time, then it is harder to know what to focus on. On the other side, by being dependent on seeing the record to be compared to produce a representation, one has essentially made it impossible to compare a large number of records in any subquadratic way. For independent representations, it is possible to generate a representation of each record once and implicitly

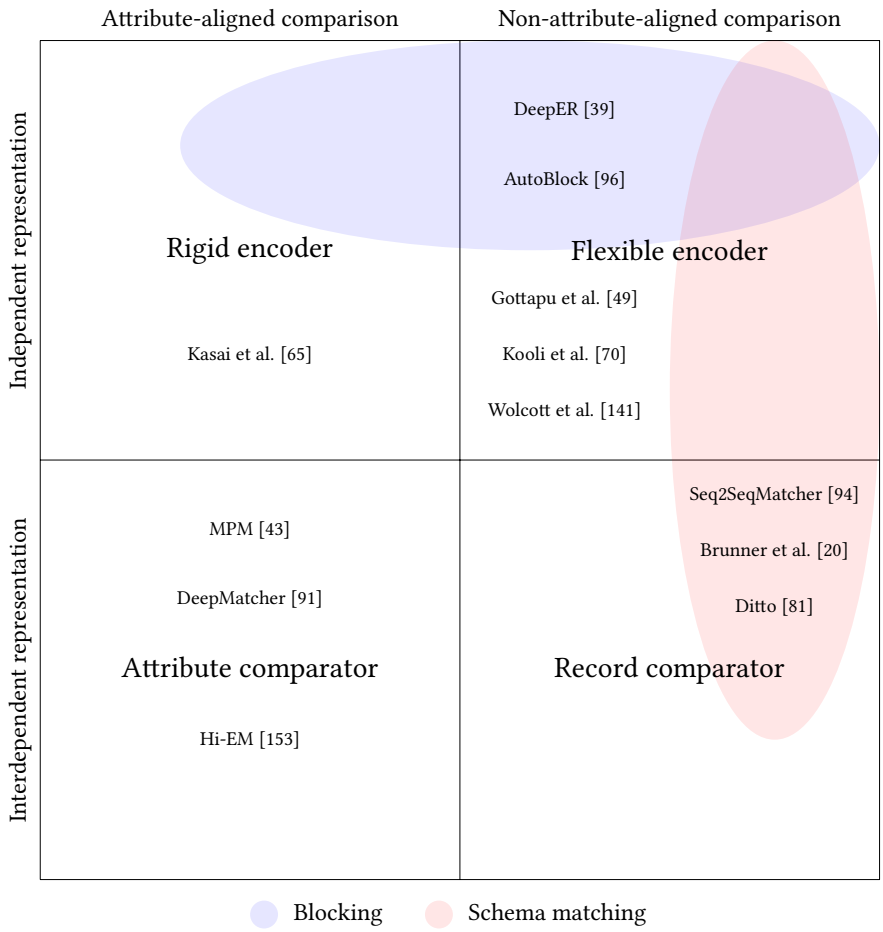
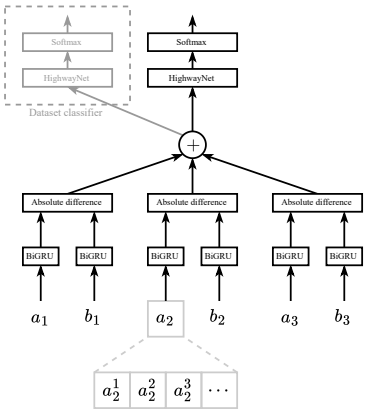
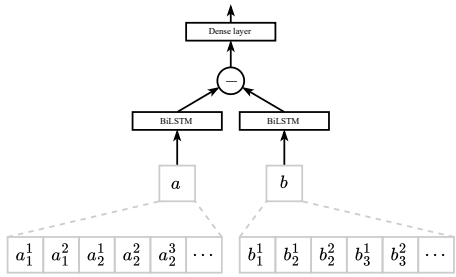


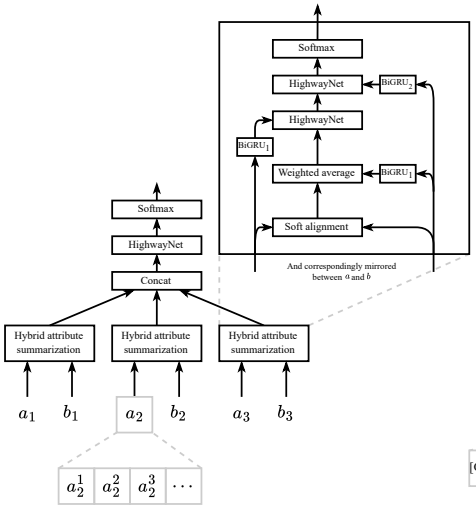
Figure 2.5: Taxonomy of deep neural networks for entity matching. Split into four categories along two axes that represent two binary properties. Methods within the blue and red colored areas are methods that address blocking and schema matching, respectively.



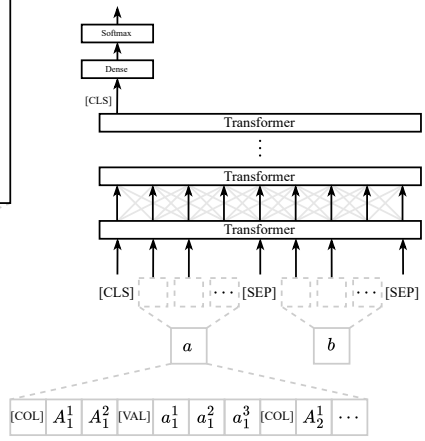
(a) Rigid encoder - Kasai et al. [65]



(b) Flexible encoder - DeepER [39]



(c) Attribute comparator - DeepMatcher [91]



(d) Record comparator - Ditto [81]

Figure 2.6: High-level illustration of four deep learning architectures — one for each category in the taxonomy. Note that the input is assumed to have already been replaced with its embedding.

compare them through a form of indexing, effectively opening up the possibility of doing blocking with the representations made from the network.

2.6.3 The four categories

- **Rigid encoders** are networks that produce independent attribute representations and then perform comparison. Even though none of the surveyed methods do, it is possible to use such independent attribute representations for blocking purposes.

- **Flexible encoders** are networks that produce independent record representations and then perform comparison. It is possible to perform blocking using these independent representations, as shown by DeepER [39] and AutoBlock [152]. If the representations can be built from records with two different schemas, one can also effectively perform schema matching, but none of the current methods do.

- **Attribute comparators** are networks that use cross-attention only within the same attribute. They are inherently focused on doing aligned attribute-to-attribute comparison. DeepMatcher [91] has attribute-to-attribute comparison explicitly designed into the network architecture. Hi-EM [153] takes it a step further by training individual networks per attribute. There is no easy way to perform blocking or schema matching with such a network architecture.

- **Record comparators** are networks that use cross-attention at record level without being constrained by the boundaries of attributes. They can learn how to compare across attributes, making it possible to overcome misaligned schemas and even incompatible schemas, depending on the network structure. Seq2SeqMatcher [94] was the first of the surveyed method in this category, and it is able to handle incompatible schemas. Transformer-based networks, which can naturally work as record comparators, have later shown to provide state-of-the-art performance [20, 81].

2.7 Evaluation

2.7.1 Metrics

An entity matching system can be evaluated in several ways. Since it is fundamentally a very skewed binary classification problem with few positives compared to negatives, it is natural to use precision/recall measures – popular metrics from information retrieval and machine learning classification. And while precision and recall (as well as accuracy) are sometimes reported, the most prominent is to report and evaluate the matches using the F_1 measure. Of course, while simple, this metric only measure one aspect of an entity matching system. Different authors have therefore used additional metrics. Kasai et al. [65] focus on the important issue of how many training examples a model needs, and measure F_1 for different amounts of provided training examples. Focused on scalability, Wolcott et al. [141] report wall-clock running time for different sizes of the data sources.

It is also possible to evaluate intermediate steps in isolation in addition to the end-to-end result. Some methods specifically target blocking, and so specifically measure the outcome of that step. One could evaluate blocking the same way as the end result, using F_1. But remember that for blocking we are mainly interesting in getting high recall, and less interesting in precision as long as the number of candidate pairs $|C|$ is sufficiently lower than the size of the Cartesian product $|A \times B|$. So the surveyed methods report both recall and some variant of reduction ratio from data sources to candidate set. Ebraheem et al. [39] report $RR = \frac{|C|}{|A \times B|}$, while Zhang et al. [152] report $P/E = \frac{|C|}{|A|}$.

2.7.2 Datasets

The early approaches to entity matching were mostly concerned with matching personal records (census data or medical records). Such datasets are usually not publicly available due to privacy concerns. Today, methods are evaluated on data from different domains. A substantial of amount of reported results are from closed datasets, but we're seeing increasingly more open datasets being used. Some of the most popular open datasets include domains such as publications, restaurants, products, songs, and companies [29, 37, 71]. See Table 2.7 for an overview of the most popular public datasets among the surveyed methods. In order to test certain scenarios, some authors also artificially construct new datasets based on existing ones, either by reducing the data quality [e.g., 91] or

constructing new records [e.g., 59].

Table 2.7: Overview of public datasets that have been used by at least two of the surveyed publications. # Records lists the number of records in each data source, # Matches denotes the number of matches between them, and # Pos / # Candidates denotes the number of positive examples among all the record pair candidates in an agreed-upon subset of the potential matches. The latter is used when blocking is not part of the experiment.

Dataset	Domain	# Records	# Attributes	# Matches	# Pos / # Candidates
Abt-Buy [71, 29]	Product	1081 - 1092	3	1096	1028/9575
Amazon-Google [71, 29]	Software	1363 - 3226	3	1300	1167/11460
Beer [29]	Beer	3274 - 4345	4		68/450
DBLP-ACM [71, 29]	Citation	2616 - 2294	4	2224	2220/12363
DBLP-Scholar [71, 29]	Citation	2616 - 64263	4	5347	5347/28707
Fodors-Zagats [37, 29]	Restaurant	533 - 331	6	112	110/946
iTunes-Amazon [29]	Music	4875 - 5619	8		132/539
Walmart-Amazon [29]	Electronics	2554 - 22074	5	1154	962/10242

2.7.3 Experimental results

Comparing surveyed methods

It has historically been hard to directly compare reported experimental results. Either because they do not target and measure the same aspect of the entity matching process, do not use the same data, or do not evaluate in the same way. For example, several methods have been evaluated on some of the same datasets but used completely different train-test splits (both the actual selection and train-test ratio). There is a general lack of widespread and agreed upon benchmarks that would make comparison across many methods straightforward — like we have seen in core computer vision and NLP tasks. To a large degree, on has relied on authors to reimplement and run other methods within their own evaluation setup. So for the majority of the surveyed methods, we can not make any comparison using reported experimental data.

It is, however, possible to partially compare some of the more recent methods. Figure 2.7 shows which surveyed methods that have experimental results that let you compare it to other methods. Mainly due to the adaptation of some of the experiments done by Mudgal et al. [91] as a benchmark, but also due to authors of a method running new experiments on earlier methods in their

evaluation setup. The Mudgal et al. [91] benchmark let us compare (at least partially) several methods at once. Table 2.8 presents an overview of all available experimental results on those specific benchmark experiments. We observe that the Transformer-based networks, especially Ditto [81], seems to be the current state-of-the-art. Ditto does use some domain-specific optimizations, but the authors report strong results for a baseline without those optimizations. Note that this benchmark only addresses certain aspects of the entity matching process. Comparable results for aspects such as training time, prediction latency, and label efficiency can not be collected in the same way for several methods — even though some results and comparison between pairs of surveyed methods have been reported. And neural methods addressing blocking, using active learning, or using transfer learning are not in a state where any significant overview can be provided.

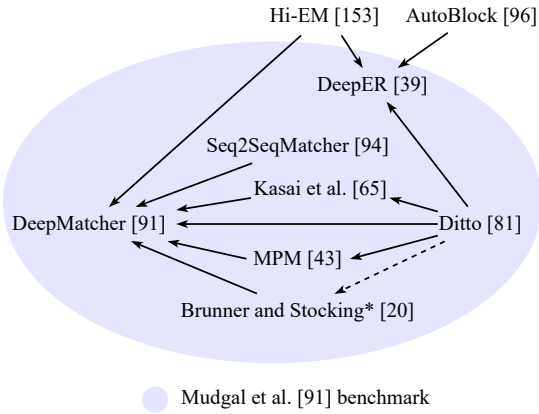


Figure 2.7: Overview of which of the surveyed methods has been compared experimentally to each other and which have at least partially been subject to the same benchmarks. Arrows indicate that the method at the head was compared to the method at the tail in the latter's publication. Note that they are not necessarily transitive, and they do not all test the same task (i.e., some compare blocking, others only matching after blocking). The blue area is methods that have been tested on at least a subset of the public benchmark from Mudgal et al. [91]. *As pointed out by Li et al. [81], the authors do model selection using the test set — effectively leaking from the test set and making the results slightly unfit for comparison with others.

Table 2.8: Overview of reported results from the surveyed methods on the public part of the benchmark from Mudgal et al. [91]. We have also included reported results [91] from Magellan (MG) as a state-of-the-art classical non-neural method. All experiments use the post-blocking record pair candidates described in Table 2.7. The *structured* versions are the unaltered datasets where attributes are aligned, while in the *dirty* variant other attributes are moved to the `title` attribute with a 50% probability. For the *textual* dataset, Abt-Buy, all attributes are long textual descriptions. $^{+}$We only report the Hybrid model, which is generally considered the strongest. †The results from Ebraheem et al. [39] are not comparable to the others due to different setup, so reproduced results from [81] are used instead. Note that those results do not involve the blocking method from [39]. *Not entirely fit for comparison since the authors do model selection using the test set.

Dataset	MG [69]	DM^{+} [91]	DeepER† [39]	MPM [43]	Kasai [65]	S2SM [94]	B&S* [20]	Ditto [81]
Structured								
Amazon-Google	49.1	69.3	56.08	70.7				75.58
Beer	78.8	72.7	50.00					94.37
DBLP-ACM	98.4	98.4	97.63		98.45	98.9		98.99
DBLP-Scholar	92.3	94.7	90.82		92.94	95.3		95.60
Fodors-Zagats	100.0	100	97.67					100.0
iTunes-Amazon	91.2	88.0	72.46					97.06
Walmart-Amazon	71.9	66.9	50.62	73.6		78.2		86.76
Dirty								
DBLP-ACM	91.9	98.1	89.62			98.4	98.9	99.03
DBLP-Scholar	82.5	93.8	86.07			94.1	95.6	95.75
iTunes-Amazon	46.8	74.5	67.80				94.2	95.65
Walmart-Amazon	37.4	46.0	36.44			68.3	85.5	85.69
Textual								
Abt-Buy	43.6	62.8	42.99				90.9	89.33

Deep learning vs. traditional methods

Several of the surveyed methods have evaluated deep learning approaches against more traditional approaches [152, 94, 65, 141, 70, 91, 39, 43, 20]. The results are generally promising for deep learning approaches, but traditional methods are often competitive. Most methods compare themselves to Magellan [69], which is considered a state-of-the-art traditional entity matching system, using F_1 scores. Mudgal et al. [91] report that DeepMatcher mostly outperforms Magellan with a few exceptions (see parts of the result in Table 2.8). The relative strength of DeepMatcher to the traditional method increases as the data quality decreases, and the same is reported about Seq2SeqMatcher [94]. AutoBlock [152] is found to be especially strong against traditional blocking techniques for dirty data.

This may suggest that deep learning approaches' strength is most clearly seen when the data is noisy and heterogeneous.

Kasai et al. [65] investigate the use of transfer learning and active learning for their deep learning model. They find Magellan to significantly outperform their model when given few labeled training examples, but they were able to make it substantially more competitive using transfer learning and active learning. The model still performs favorably in comparison to the traditional methods when given enough examples.

2.8 Future research

In Section 2.5, we saw which contributions deep learning have made to entity matching. We will now take a look at both the challenges and opportunities for deep neural networks in entity matching, both of which represent potential directions for future research.

2.8.1 Challenges

Explainability and ease of debugging

Entity matching, being a central data integration task, is constructing data models that are consumed by people or machines for downstream tasks. For many applications, it is crucial to trust the data source, and being able to understand why something does not work is key. Unfortunately, deep learning models are notoriously hard to interpret. As steps in the entity matching process increasingly coalesce into a large neural network, as illustrated in Figure 2.4, we get fewer checkpoints along the way in the process that can easily be inspected. We cannot see the output from each step in the same way anymore. Therefore, figuring out why two records where matched or not matched is usually nontrivial. There are a few techniques that are already used, such as looking at alignment scores, but we are still far away from a comprehensive way of debugging neural networks for entity matching.

Running time in interactive settings

Human interaction is considered an important factor in entity matching [35]. Users cannot generally be expected to wait for very long when they are supposed to do interactive work, limiting the potential applications of deep learning for entity matching in interactive settings, since the running time for both training

and prediction is high – especially compared to cleverly engineered traditional pipelines with good blocking.

Number of training examples

As more steps in the entity matching process are addressed by deep learning-based techniques, the more different steps rely on more training data. Deep learning models are hungry for training examples, and there is not always an abundance of them readily available. Even while the use of deep learning can potentially reduce the need for manual labor in the form of feature engineering, it might still be necessary to do expensive manual labeling. While transfer learning and active learning could help, there are not any pretrained entity matching models publicly available to train from.

2.8.2 Opportunities

End-to-end approach with schema matching and blocking

As we have seen, deep neural networks are increasingly able to take over steps in the entity matching process. But we have still not seen a method doing both schema matching and blocking together with record pair comparison and classification in an end-to-end solution with a neural network – in other words, an approach that implements the whole deep learning entity matching reference model in Figure 2.4. The network would presumably be a *flexible encoder*, belonging in the upper-right corner of our taxonomy. It would have independent representations to make efficient blocking possible and non-attribute-aligned comparison to make schema matching possible. This could be an important step toward tackling the entity matching problem in a streamlined end-to-end fashion.

More open datasets

One of the fundamental challenges when trying to develop an entity matching methods that will work across many datasets is the huge variation between domains and data sources. While several open datasets are available, we would like to see significantly more. It is especially important to increase the diversity of the domains represented. For example, many datasets are related to either research publications or consumer-focused products/services – there are no industrial datasets. This would not only enable more complete evaluations, but also provide data suitable for transfer learning.

Standardized benchmarks

It is not always easy to compare the different methods since they do not necessarily evaluate on the same data in the same way. We have seen how standard open datasets with corresponding standard evaluation metrics have been a catalyst for advances in machine learning for fields such as computer vision [116], recommender systems [15] and natural language processing [17]. We think there is great potential in similar agreed-upon benchmarks for entity matching. This need not only be restricted to traditional precision/recall measures for the resulting matches. It would also be of interest to standardize the evaluation of blocking techniques, transfer learning techniques, active learning techniques, and computational performance and efficiency.

Publicly available pretrained models

While there has been work on transfer learning [65, 153, 20, 81], there are no pretrained models publicly available specifically for entity matching. Having pretrained models to fine-tune could potentially speed up training and reduce the amount of necessary training examples. This is challenging, because, as mentioned above, there is a huge variation across domains and data sources. Each data source is different. Building pretrained models that can be fine-tuned for a broad number of data sets and making them publicly available would be of huge benefit to the field. This would, of course, benefit from the previous point about more open datasets.

An interesting approach for those networks with attribute-aligned comparison, explored by Zhao and He [153], is to have pretrained models for different types of attributes (e.g., name, addresses, organization) in addition to generic ones. Then one can mix and match models for each attribute one is about to match. This, of course, only works for aligned schemas. It can be even more interesting to look at networks with non-attribute-aligned comparison for such pretrained models, since they can potentially offer pretrained schema matching, which will be crucial to handle the large variety of data source schemas out there. For this, Transformer-based networks have considerable potential since they are already built on top of heavily pretrained language models.

Pretrained models can belong to any of the four categories of our taxonomy. Nonetheless, we find pretrained flexible encoders that can support both blocking and schema matching to be the most significant opportunity. Specifically, because it would enable transfer learning jointly on the largest possible portion of the entity matching process.

2.9 Conclusion

We have seen how existing work has used neural networks in entity matching. By using a reference model of the traditional entity matching process, we have shown how the surveyed methods address different steps of the process and which techniques have been used at each step.

More recently, approaches based on deep learning techniques for natural language processing have emerged. We looked at what such deep learning methods contribute to the entity matching task. The central contribution is powerful hierarchical representation learning from text. As we have seen, this can alleviate most of the handcrafted feature engineering necessary in the data preprocessing, which can ease the burden of manually tailored procedures in downstream steps such as record pair comparison. Furthermore, it is a driver in increasingly coalescing steps of the entity matching process into end-to-end neural networks performing several steps in one go, effectively reducing the number of steps and enabling end-to-end training. To give a clear view of how the entity matching process changes with such an increasingly coalesced deep learning approach, we propose a reference model for entity matching processes using deep learning.

To differentiate the deep neural networks used in the surveyed methods, we introduced a taxonomy of deep neural networks for entity matching. It focuses on two properties that are important in regard to how easy it is to support schema matching and blocking.

Lastly, we looked at potential directions for future research by discussing challenges and opportunities. The challenges being explainability, running time in interactive settings, and the large need for training examples, while for opportunities we think it would be interesting to develop a complete end-to-end approach with both schema matching and blocking, exploring a new part of the deep neural network taxonomy. We also see a lot of potential in trying to develop more open datasets, standardized benchmarks, and publicly available pretrained models for entity matching — which have been important for other fields.

Chapter 3

Approach-Agnostic Explainability for Entity Matching

Paper B

Original title:	LEMON: Explainable Entity Matching
Authors:	Nils Barlaug
Published in:	IEEE *Transactions on Knowledge and Data Engineering* (TKDE) in 2022

State-of-the-art entity matching (EM) methods are hard to interpret, and there is significant value in bringing explainable AI to EM. Unfortunately, most popular explainability methods do not work well out of the box for EM and need adaptation. In this paper, we identify three challenges of applying local post hoc feature attribution methods to entity matching: cross-record interaction effects, non-match explanations, and variation in sensitivity. We propose our novel model-agnostic and schema-flexible method LEMON that addresses all three challenges by (i) producing dual explanations to avoid cross-record interaction effects, (ii) introducing the novel concept of attribution potential to explain how two records could have matched, and (iii) automatically choosing explanation granularity to match the sensitivity of the matcher and record pair in question. Experiments on public datasets demonstrate that the proposed method is more faithful to the matcher and does a better job of helping users understand the decision boundary of the matcher than previous work. Furthermore, user studies show that the rate at which human subjects can construct counterfactual examples after seeing an explanation from our proposed method increases from 54% to 64% for matches and from 15% to 49% for non-matches compared to explanations from a standard adaptation of LIME.

3.1 Introduction

Entity matching is an essential task in data integration [34]. It is the task of identifying which records refer to the same real-world entity. Figure 3.1 shows an example. Machine learning has become a standard tool to tackle the variety of data and to avoid laborsome feature engineering from experts while still achieving high accuracy (e.g., [69, 91, 81]). Unfortunately, this is often at the cost of reduced transparency and interpretability. While it is possible to carefully select classical machine learning methods that are intrinsically interpretable and combine them with classical string similarity metrics, current state-of-the-art consists of large deep learning models [20, 39, 81, 91], which offer limited interpretability out of the box. The possible benefits of being able to explain black-box models are numerous. To mention some: 1) Researchers can gain new insight into their models and find ways to improve them 2) Practitioners will have a valuable tool for verifying that the models work as expected and debugging those which do not 3) Companies can gain the necessary transparency they need to trust such black-boxes for mission-critical data integration efforts

4) End-users can be reassured that models and their results can be trusted, or discover themselves that they should not be.

The challenge of explaining machine learning models and their potential benefits is not unique to entity matching. Therefore, explainable machine learning has in recent years received significant attention from the broader research community [1, 45, 53]. The result is a multitude of techniques and methods with different strengths and weaknesses. But as previous work has discussed, applying these techniques to entity matching is non-trivial. It is necessary to adapt and evolve them to tackle the unique characteristics of entity matching [33, 38, 128].

title	belkin shield micra for ipod touch tint
category	mp3 accessories
brand	belkin
modelno	f8z646ttc01
price	47.88

title	belkin ipod touch shield micra tint-royal purple
category	cases
brand	belkin
modelno	f8z646ttc02
price	12.49

Figure 3.1: Example of two records that need to be classified as either a match or a non-match from the Walmart-Amazon dataset. In this case, the records refer to almost the same product — the only definitive difference being the color.

Local post hoc feature attribution methods are perhaps the most popular type of explainability method in general, and the most studied so far for entity matching [38, 128, 33, 7]. Previous work on explainable entity matching base their work on LIME [114] — one of the most popular methods of that type. In this paper, we choose to focus mainly on LIME to be consistent with, and for ease of comparison to, earlier work. Our work is relevant beyond LIME, and we will reference and include other methods in our experiments, but we consider in-depth adaptation and treatment of other methods outside of the scope of this paper and hope to address them in future work.

Challenges. Unfortunately, standard local post hoc attribution methods do not work satisfactorily for entity matching out of the box. We identify three challenges of applying them:

1. **Cross-record interaction effects**: Since EM is a matching problem, features across a record pair will tend to have strong interaction effects, but linear surrogate models such as in LIME implicitly assume independent features. This can severely impair the accuracy of the surrogate model.

2. **Non-match explanations**: In essence, most feature attribution methods analyze the effect of removing features to determine their attribution. However, for record pairs for which the matcher is fairly confident that they do not match, so the match score is close to zero, it is unlikely that removal of features will make any significant difference on the match score. The result is that we have no significant attributions to explain why the records do not match. This is especially important since most record pairs do not match.

3. **Variation in sensitivity**: While some record pairs may only need to perturb a few features to trigger a significant change in the output of the matcher, others may contain a lot of redundant features, making it hard to substantially impact the matching score and provide meaningfully sized attributions. It can be hard to make the trade-off between token and attribute level feature granularity. One risk being either too fine-grained or unnecessarily course-grained, and it differs between specific record pairs in the same dataset, across datasets, and across matchers.

As we will outline in Section 3.2, earlier work has only partially addressed these challenges.

Proposed method. Our proposed method addresses all three challenges jointly by: 1) Using dual explanations to avoid cross-record interaction effects. 2) Introducing the novel concept of attribution potential, an improvement over the copy/append perturbation from previous work that is schema-flexible and more robust to dirty data and matchers sensitive to the order of tokens. 3) Choosing an interpretable representation granularity that optimizes the trade-off between counterfactual interpretation and the finest granularity possible. Our proposed method provides one unified frame of interpretation with the same single explanation format for all record pairs, has no dataset-specific hyperparameters that need tinkering, and does not require matched schemas. Source code is publicly available[1].

Evaluation of explainability methods is still an open problem, and there are no standard ways of evaluating explainable entity matching. Ideally, in broad terms, we would like to measure to what degree an explanation helps users understand how the model makes a matching decision. Inspired by the motivations behind counterfactual examples [134], we argue that a useful attribution explanation should help the user understand where the decision boundary is and what kind of difference in input would be necessary to sway the matcher. To

[1] `https://github.com/NilsBarlaug/lemon`

that end, we propose to ask users what they think would be a minimal change to a record pair to make the matcher change its prediction and then check if they are correct. We show how this can be done for simulated users as well as human subjects.

Finally, our results show that our proposed method is state-of-the-art, both in terms of faithfulness and helping users understand the matcher's behavior — though at the cost of higher runtime. Additionally, our user study shows great potential for real-world improvement in understanding by human subjects.

Contributions. In summary, our main contributions are:

- We propose a method that addresses three important challenges of applying local post hoc attribution methods to entity matching: 1) Cross-record interaction effects, 2) non-match explanation, and 3) variation in sensitivity. We show through experiments that this is indeed effective.
- To evaluate entity matching attribution explanations, we propose a novel evaluation method that aims to measure to what degree explanations help users understand the decision boundary of a matcher. We show how to perform experiments on both simulated users and human subjects.
- Through extensive experiments on public datasets we show that our proposed method is state-of-the-art both in terms of faithfulness and helping users understand the matcher's behavior. We verify the real-world applicability of our proposed method by performing an extensive user study. To the best of our knowledge, we are the first to conduct a user study for explainable entity matching.

Outline The rest of the paper is organized as follows. Section 2 briefly covers related work, Section 3 covers LIME and its adaptation to entity matching, and Section 4 goes into the details of our proposed method. We explain our experimental setup in Section 5, and then we walk through and discuss the experiments in Section 6 before we make our concluding remarks in Section 7.

3.2 Related work

Machine learning for EM. The immense variety in datasets makes machine learning a natural solution for entity matching. The traditional approach has been to handcraft string similarity metrics to produce similarity feature vectors and then utilize a classical off-the-shelf machine learning model such as SVM or random forest to classify them [25, 40, 69]. The two main drawbacks of this

approach are the necessary manual tinkering and poor performance on dirty data [91]. However, in the last few years, the research community has increasingly adopted deep learning [91, 39, 94, 153, 20, 81]. While early work focused on custom architectures and trained models from scratch, the current state of the art focuses on fine-tuning large natural language models such as BERT [32], which offers higher accuracy and decreases the need for training examples [81]. We refer to [11] for an extensive survey on deep learning for EM.

Explainable AI. There are many ways to explain machine learning models. Generally, explanations can be either global or local [36], in other words, explaining the model's behavior as a whole or explaining a single prediction. Furthermore, we often distinguish between intrinsically interpretable models and post hoc interpretation methods [90] (which can be model-agnostic or not). The former are models that are interpretable on their own, like linear regression or decision trees, while the latter are methods for explaining black-box models. We refer the reader to one of many extensive sources on the topic [1, 36, 45, 53, 90].

A particularly prominent group of approaches are local post hoc attribution methods (e.g., [114, 86, 124]), which aim to explain a prediction by communicating to what degree different parts of the input are to be attributed for the prediction. LIME [114] is one of the most prominent among such methods. It is a model-agnostic method, and works by randomly removing features of an input example and training a (usually linear) interpretable surrogate model to predict the model's output for these perturbations. Among other popular local post hoc attribution methods are the game-theoretic-based SHAP [86] and gradient-based methods (e.g., [124]).

Explainable EM. The use of explainability techniques for machine learning-based entity matching is still a young subject, and there has only been a limited amount of previous work. However, we note that rule-based methods have historically been used to make systems that can be interpreted by experts [40], and they represent an alternative way to make explainable matchers [108].

The authors in [38] demonstrate ExplainER, a tool for exploring explainable entity matching that provides multiple prominent explainability techniques such as LIME and association rules, while [128] discuss challenges and research opportunities. Further, there have been two significant adaptations of LIME for entity matching, which we will now describe and contrast to our work.

Mojito [33] introduces two versions of LIME for entity matching: LIME_DROP and LIME_COPY. The former is a straightforward application of LIME similar to how the original authors do text classification using token level feature

granularity[2], while in the latter they use attribute level representation and perturb by copying the entire attribute value to the corresponding attribute in the other record instead of removing tokens. LIME_COPY is an elegant way to address challenge (2), but leaves more to be wanted. Firstly, attribute level granularity is too coarse-grained for most cases with longer textual attributes (the extreme case being a single textual attribute). Secondly, since it is separate from LIME_DROP, it requires the user to interpret two different kinds of explanations. Finally, it relies on a matched schema with one-to-one attribute correspondence.

Recently, Landmark [7] was proposed as a two-part improvement over Mojito. Firstly, it makes two explanations, one per record, and avoids perturbing both records simultaneously, which effectively solves challenge (1). Secondly, for record pairs labeled as non-matches, instead of perturbing by randomly copying entire attributes, it appends every corresponding attribute value from the other record and performs regular token level exclusion perturbation (a technique named double-entity generation), effectively combining LIME_DROP and LIME_COPY. The authors demonstrate through experiments that their techniques are indeed effective and that Landmark is a substantial improvement over Mojito. One limitation of the approach is that tokens from the other record are only ever considered to be appended at the end of the corresponding attribute. This is unfortunate if the matcher is sensitive to the order of tokens (e.g., many products have the brand name first in the title), the schemas are not matched one-to-one, or the data is dirty. Similar to Mojito, Landmark also makes two different kinds of explanations.

While making important contributions, neither Mojito nor Landmark addresses all three challenges identified in Section 3.1. Only Landmark tackles challenge (1). Both propose a solution to challenge (2), but with important limitations, as we discussed above. Neither addresses challenge (3). Moreover, they do not provide a unified and coherent way of actually communicating or visualizing an explanation to the end-user in the same way the original authors of LIME do — something we aim to do.

3.3 Preliminaries

In this section, we first present the problem definition and then introduce LIME and describe how it can be adapted for entity matching.

[2]Tokens are typically words or singular values, and can be assumed to be for datasets and experiments in this paper, but does not necessarily have to be for the described methods.

3.3.1 Problem Definition

Entity Matching. Let A and B be two data sources. A data source is a collection of records following the same schema (they all have the same attributes), and a record $r = \{(\alpha_j, v_j)\}_j$ is an ordered set of attribute name-value pairs. The goal of entity matching is to find all pairs $(a, b) \in A \times B$ such that a and b refers to the same real-world entity. We call such pairs *matches* and all other pairs for *non-matches*. Since there is a quadratic number of pairs $O(|A||B|)$, inspecting all pairs in $A \times B$ is usually infeasible. Therefore, one will normally first perform a recall-focused step called blocking [100] to produce a set of candidates $C \subseteq A \times B$ such that $|C| \ll |A \times B|$ while still containing most matches with high probability. Then we classify every $(a, b) \in C$ as either match or non-match. In this paper, we focus on the record pair classification part of entity matching. Therefore, for our purposes, entity matching is a binary classification problem deciding whether a pair of records (a, b) refer to the same real-world entity or not (match or non-match).

Local Post Hoc Attribution for Entity Matching. The goal of a local post hoc attribution explainability method for entity matching is to explain a single prediction of a record pair classification from an arbitrary matcher by communicating the significance (in some shape or form) of the different parts of the two records to the user. Formally, let a matcher be a classifier $f(x)\colon A \times B \to \mathbb{R}$ that accepts a record pair $x = (a, b)$ such that $a \in A \wedge b \in B$ and outputs a prediction score between 0 and 1. Note that f is not restricted to supervised machine learning models but can be any procedure capable of classifying record pairs with a confidence score. A local post hoc attribution explainability method for entity matching provide two things. First, it provides a procedure $\lambda(f, x)$ that accepts a matcher f and a record pair x and outputs an explanation e_x. Secondly, it provides a framework of interpretation for the explanations. An explanation e_x attributes different parts of x to the prediction score $y = f(x)$ using real-valued attribution scores, and the explanation is communicated to the user either through numbers directly or some visualization (see for example [114]). How the attribution scores are to be interpreted and how it should be communicated to the user is up the method.

As stated in Section 3.1, evaluation of explainability methods is still an open problem and there are no standards for how to do it in entity matching. Therefore, as part of our contribution we propose ways to do this for attribution methods. We refer the reader to Section 3.6 for more on this.

3.3.2 LIME

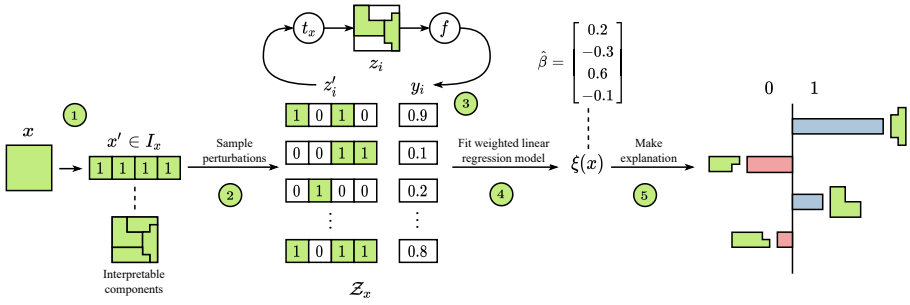


Figure 3.2: Illustration of LIME [114] for a binary classification problem. The main steps are: 1) Split the input x into interpretable components. 2) Sample neighbors z'_i of the interpretable representation $x' \in I_x$. 3) Convert each z'_i to a corresponding input domain representation $z = t_x(z'_i)$ and run inference to get $y_i = f(t_x(z'_i))$. 4) Fit a weighted linear regression model on the neighborhood dataset $\mathcal{Z}_x$ to get $\xi(x)$ — essentially the regression coefficients $\hat{\beta}$. 5) Present $\hat{\beta}$ in a user-friendly way relating them to the interpretable components.

The main idea of LIME [114] is to locally approximate a classifier around one instance with an interpretable model over an interpretable representation in a way that balances faithfulness and complexity, and then use the interpretable model as an explanation. The intuition is that while our problem is too complex for classical machine learning models that we regard as inherently interpretable (e.g., linear regression or decision trees) to be accurate enough, it might be possible to faithfully approximate the decision boundary for a black-box model locally around one input instance. In other words, the authors train an interpretable surrogate model using local data points sampled by perturbing an input instance and use it as an explanation of that particular instance. And while the input features of a black-box model might be unsuited for human interpretation (e.g., deep learning embeddings or convoluted string metrics), they define an alternative interpretable representation for the input instance we want to explain and use that to train the interpretable surrogate model.

Formally, let $f(x)\colon A \times B \to \mathbb{R}$ be the matcher we want to explain. Furthermore, for a single instance $x = (a, b)$ that we want to explain, let $I_x = \{0, 1\}^{d_x}$ be the interpretable domain and $x' \in I_x$ the interpretable representation of x.

Its elements represent the presence (or absence) of what is called *interpretable components* in [114], essentially non-overlapping parts of the input. E.g., for text, it could be the presence of different words. For each x there must exist a function $t_x(x')\colon I \to A \times B$ that can translate an interpretable representation to the input domain of the classifier f.

With the goal of approximating f local to x, the authors sample a new dataset $\mathcal{Z}_x = \{(z', y) \in I \times \mathbb{R}\}$ where $y = f(t_x(z'))$. Each z' is drawn by setting a uniformly random-sized subset of x' to zero. Let G be a class of interpretable models over I_x. Furthermore, let $\mathcal{L}(f, g, \pi_x)$ be how unfaithful $g \in G$ is to f in the neighborhood defined by the distance kernel $\pi_x(z')$. They want to find a $g \in G$ that is as faithful to f as possible, but since many interpretable models can be made more accurate by increasing the complexity, they need to balance the faithfulness with model complexity so g is simple enough to actually be interpretable for humans. To that end, let $\Omega(g)$ be a measure of complexity for g, and choose the following explanation for x:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \Big[\mathcal{L}(f, g, \pi_x) + \Omega(g) \Big] \tag{3.1}$$

In their work, the authors only present one concrete instance of their general framework[3]. They chose G to be weighted sparse linear regression models and $\mathcal{L}$ to be mean squared error weighted by π_x on $\mathcal{Z}_x$. Furthermore, $\Omega(g)$ is chosen to be the number of non-zero coefficients of g, and the trade-off between $\mathcal{L}$ and Ω is simplified by constraining $\Omega(g)$ to not be greater than a constant K known to be low enough. It is now simply a matter of fitting a regularized weighted least squares linear regression model on $\mathcal{Z}_x$. For a simplified overview of the whole process see Figure 3.2.

3.3.3 LIME for Entity Matching

Before we describe our proposed method in the next section, we will now go through the design decisions done within the LIME framework. A setup we then build upon and use as a baseline for our proposed method.

Let a record $r = \{(\alpha_j, v_j)\}$ be an ordered set of pairs with attribute name and value. Inspired by how [114] apply LIME for text classification, we define the interpretable representation I_x to be the absence of unique tokens in attribute

[3]We will, as is common in the literature, refer to both the general framework and the described concrete instance as LIME interchangeably.

names and values for both records in $x = (a, b)$ (i.e., $t_x(\mathbf{0}) = x$)[4]. Attribute values that are not strings are treated as single tokens, and their absence is their null/zero value.

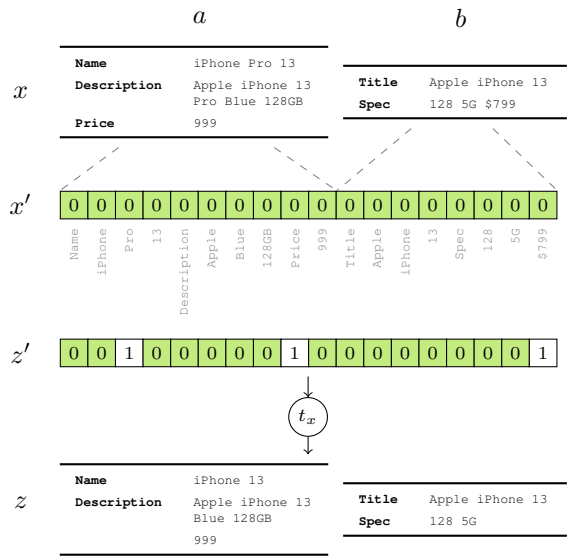


Figure 3.3: Example record pair x, its corresponding interpretable representation x' in LIME, and an example of a perturbed sample.

Example 1 *Figure 3.3 shows an example record pair x and its corresponding interpretable representation x'. In this example, the records are product descriptions of two similar (but different) phones. The record a refers to a "pro" version of the phone referred to by b. Furthermore, it is uncertain whether the phones have the same color since b does not specify its color while a is blue. The figure also shows an example of a perturbed interpretable representation z' and how it is translated with t_x into a perturbed record pair z. The token* `Pro` *(among others) is removed from record a and it is more likely that a reasonable matcher will consider the record pair z a better match then x.*

[4]The reader might also note that the choice of 0/1 semantics are flipped compared to the authors in [114] (see Section 3.3.2). This is simply to be more conceptually similar to our proposed method and is not critical to the approach.

We sample $\mathcal{Z}_x$ by setting random subsets of x' to one, where the size of the subsets are sampled uniformly from the interval $[0, D_{\max}]$, and we use the neighborhood distance kernel

$$\pi_x(z') = \exp(-2\frac{D(x', z')}{D_{\max}}) \tag{3.2}$$

where D is the Hamming distance. While the original authors simply used $D_{\max} = d_x$ for text classification, we empirically find this neighborhood too large due to entity matching generally being more sensitive to single tokens compared to standard text classification. This could, of course, be accounted for by narrowing π_x, but it is more sample efficient to also reduce the neighborhood we are sampling from. We use $D_{\max} = max(5, \lfloor \frac{d_x}{5} \rfloor)$, and let the number of samples $|\mathcal{Z}_x|$ be $\max(500, \min(30d_x, 3000))$. From our experience, the results are not sensitive to these parameters.

Finally, we let G be the set of weighted regression models without intercept. The loss then being

$$\mathcal{L}(f, g, \pi_x) = \sum_{(z', y) \in \mathcal{Z}_x} \pi_x(z')(y - g(z'))^2 \tag{3.3}$$

$\xi(x)$ is found using weighted least squares and forward selection (choosing K coefficients).

3.4 Method

We now describe how we address the three challenges described in Section 3.1 with three distinct, but coherent, techniques that together form our proposed method: Local explanations for Entities that Match Or Not (LEMON). As discussed earlier, we use LIME as the basis for our method, but the proposed ideas have wider applicability. The three following subsections respectively address and propose a solution to the three challenges (1) cross-record interaction effects, (2) non-match explanations, and (3) variation in sensitivity.

3.4.1 Dual Explanations

One shortcoming of LIME, when applied directly to entity matching, is that it does not take into account the inherent duality of the matching. No distinction is made between the two input records. This is problematic because our surrogate

linear regression model assumes independent features, but perturbations across two input records will naturally have strong interaction effects. In essence, the surrogate model g cannot sufficiently capture the behaviour of our matcher f even for small neighborhoods, which severely hurts the approximation accuracy.

The proposed solution is relatively straightforward but still effective. Equivalently to [7], we make two explanations, one for each record. We call such a pair (e_x^a, e_x^b) of complementary explanations for dual explanations. For each explanation, we let I_x represent only the absence of tokens in one record. In effect, we approximate attributions from only one record at a time while keeping the other constant. That way, we avoid the strong interaction effects across them. Intuitively, we explain why record a matches b or not and why record b matches a or not, separately. The two explanations can still be presented together as one joint explanation to the user.

3.4.2 Attribution Potential

Attribution methods such as the LIME implementation described above tell us which part of the input is the most influential. This is usually achieved using some kind of exclusion analysis where one looks at the difference between the absence and presence of input features (e.g., [114, 124, 86]). While that might be an effective approach for many machine learning problems, it is inherently unsuited for entity matching. The issue lies in explaining non-matches. Record pairs that a matcher classifies as a match can be explained subtractively because removing or zeroing out essential parts of the records will result in lower matching scores from most well-behaved matchers. But for record pairs where the matcher is convinced they do not match and provide a near-zero match score, it is unlikely that removal or zeroing out any part of the records will make a significant difference on the match score. For example, in the record pair from Example 1 a matcher's output might not change significantly by removing `Blue` from a because it correctly identifies that there is still a lack of matched color information. Seemingly, nothing influences the match score, thereby providing no useful signal of the contribution from different features. Notice that standard gradient-based methods are not able to escape this problem because f will, in these cases, be in a flat area and the gradients be rather uninformative. Intuitively, we can not explain why two records do not match by what they contain. A natural solution to this problem is instead to explain by what they do not contain.

Interpretable Representation. Let the interpretable representation $I_x = \{P, A, M\}^{d_x}$ be categorical instead of binary, where the values represent whether

the corresponding token is *Present*, *Absent*, or *Matched*. Unsurprisingly, $z'_i = A$ means $t_x(z')$ will exclude token i and if $z'_i = P$ it will be kept — much like before. On the other hand, if $z'_i = M$, we will copy and inject the token in the other record where it maximizes the match score $f(t_x(z'))$. For now, let us assume we have an accurate and efficient implementation of $t_x(z')$.

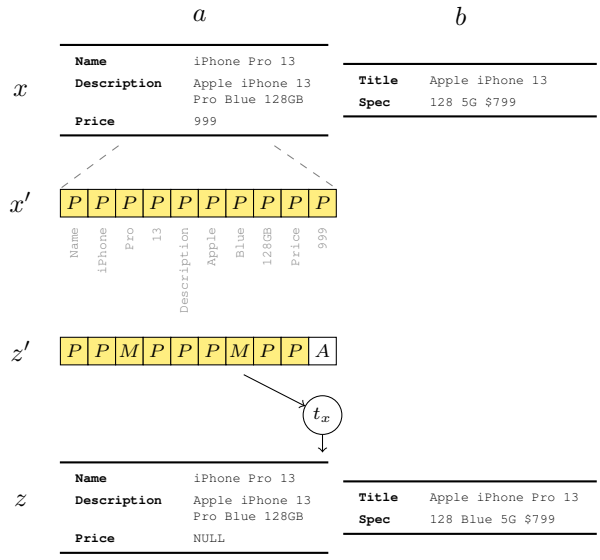


Figure 3.4: Example record pair x, its corresponding interpretable representation x' in LEMON for one of two dual explanations, and an example of a perturbed sample.

Example 2 *Reusing the record pair x from Example 1, Figure 3.4 shows the interpretable representation x' for one of two dual explanations in LEMON (explaining why a matches, or not, b). The representation would, of course, be similar for the other of the two explanations. In the figure, we also see an example of a perturbed interpretable representation z' and its equivalent record pair z provided by t_x. Notice how both* `Pro` *and* `Blue` *have been injected into b — making it more likely to be accepted as a match by a reasonable matcher.*

For the linear surrogate model, we dummy code I_x, using P as reference value. When we do forward selection, we select both dummy variables repre-

senting a categorical variable at once, so that we either pick the entire categorical variable or not. We will then have two estimated coefficients, $\hat{\beta}_i^A$ and $\hat{\beta}_i^M$, for each of the K selected interpretable features. Finally, we define the attribution for token i to be $w_i = -\hat{\beta}_i^A$, and the attribution potential to be $p_i = \hat{\beta}_i^M$. Intuitively, w_i is the contribution of token i, and is the same as in LIME, while p_i can be interpreted as the maximum additional contribution token i could have had if the other record matched better. Note that it is important to model this new attribution potential through a categorical variable instead of simply adding another binary variable to I_x, because exclusion and injection perturbations of the same token strongly interact with each other and should be mutually exclusive.

Approximating $\boldsymbol{t_x}$. In contrast to plain LIME as described in Section 3.3.3, t_x is now less straightforward to compute. The difficulty lies in where to inject tokens i for which $z'_i = M$ to maximize $f(t_x(z'_i))$. Since our method is model-agnostic, the best we can do is try all possible injections. That would be computationally prohibitive, not only because of the high number of possible injection targets but also because of the exponential growth of combinations when multiple tokens should be injected. Instead, we can approximate it by sampling L combinations of injection targets and picking the one that gives the highest match score.

The possible injection targets for a token in an attribute value are anywhere in the string attributes of the other record, but without splitting tokens in the target attribute value. If the token is a non-string value, it can also overwrite attribute values of the same type — e.g., a number attribute can replace a number attribute in the other record. Tokens from attribute names can only be injected to attribute names. When we sample injection targets, we first pick a target attribute uniformly at random and then a random position within that attribute. In addition, we employ a heuristic to incorporate information about matched schemas if available. In cases where the schemas are matched, we boost the probability of choosing the corresponding attribute as the target to 50%. This makes efficient use of prior knowledge about the schemas while still preserving robustness to dirty data. To pick the sample size L, we first sum the possible injection targets per token to be injected. We cap the number of targets to three per attribute and ten in total and then let L be the maximum of all tokens to be injected.

Neighborhood sampling. We sample the neighborhood $\mathcal{Z}_x$ much like before. Now, x' will be a vector of only P, and we let $z'_i = A$ instead of 1 for random subsets. But we additionally set random subsets of elements to M. The subset size is chosen uniformly at random from $[0, \max(3, \lfloor d_x/3 \rfloor)]$, except with a 50% chance of picking 0. The reason we sample M less than A is that injections

tend to have more dramatic effects on the match score than exclusions, and so we consider them to be larger perturbations and want to avoid drowning the exclusion effects.

3.4.3 Counterfactual Granularity

Depending on the dataset and matcher, influencing the match score significantly can sometimes require perturbing large parts of the input records. This is especially true for datasets where records contain many high-quality pieces of information because it provides the matcher with multiple redundant strong signals about whether they match or not. An example is the iTunes-Amazon dataset, where attributes such as song name, artist name, album name, and more might all agree or disagree at the same time. The problem is that we want to pick out K important features for the user to focus on, but in such cases, no single token is likely to be significantly important. While we could use attribute-level features, that would be unnecessarily coarse-grained for many cases. Instead, we propose an adaptive strategy where we automatically choose an appropriate explanation granularity. The idea is to exponentially decrease the granularity of the interpretable features until the attributions and attribution potentials are large enough in magnitude to explain the decision boundary.

Let $e_x = \{(w_i, p_i)\}_{i \in E_K}$ be the K pairs of attributions and attribution potentials for an explanation $\xi(x)$, where E_K is the K interpretable features chosen to be used in the regularized linear surrogate model. Further, let $\widehat{inc}_i = \max(-w_i, p_i)$ and $\widehat{dec}_i = w_i$. In other words, this is how much perturbation of token i could increase or decrease the match score according to w_i and p_i if you removed the token or injected the token in the other record. Then, to represent greedy actions increasing the match score, let INC be a vector of the elements in E_K with positive $\widehat{inc}_i$ and sorted by $\widehat{inc}_i$ in descending order, and similarly for DEC. We define the predicted counterfactual strength of k steps to be

$$\widehat{CFS}^k(e_x) = \begin{cases} p - \left[f(x) - \sum_{s=1}^{s=k} DEC_s\right], & f(x) > p \\ \left[f(x) + \sum_{s=1}^{s=k} INC_s\right] - p, & f(x) \leq p \end{cases} \tag{3.4}$$

Intuitively, this is to what degree one would assume to surpass the classification threshold p if one performs k greedy actions to change the matcher prediction. Note that most matchers, as well as those in our experiments, have a classification threshold p of 0.5 [69, 81]. Then let the greedy counterfactual strategy

$k_g(e_x)$ be the smallest number of steps predicted to be necessary to get a counterfactual strength of at least ϵ:

$$k_g(e_x) = \begin{cases} \min\{k : \widehat{CFS}^k(e_x) \geq \epsilon\}, & \text{if such } k \text{ exists} \\ |DEC|, & \text{otherwise if } f(x) > p \\ |INC|, & \text{otherwise if } f(x) \leq p \end{cases} \tag{3.5}$$

Finally, we define the predicted counterfactual strength of the explanation $\xi(x)$ to simply be $\widehat{CFS}(e_x) = \widehat{CFS}^{k_g(e_x)}(e_x)$, and the actual counterfactual strength to be:

$$CFS(e_x) = \begin{cases} p - \Big[f(x) - f\big(t_x(x'_g)\big)\Big], & f(x) > p \\ \Big[f(x) + f\big(t_x(x'_g)\big)\Big] - p, & f(x) \leq p \end{cases} \tag{3.6}$$

where x'_g is the perturbation of the interpretable representation x' corresponding to the greedy counterfactual strategy.

When an explanation's interpretable features represent (up to) n consecutive tokens, we say that explanation has a granularity of n tokens. To find our desired granularity, we start with a granularity of one token and then double until we find a granularity that satisfies $\widehat{CFS}(e_x) \geq \epsilon \wedge CFS(e_x) \geq \epsilon$ or no coarser granularity is possible (i.e., all features are whole attributes). When no granularity satisfies the requirement, we pick the granularity with the highest harmonic mean between $\widehat{CFS}(e_x)$ and $CFS(e_x)$, which will favor them to be large and similar.

We call the resulting approach for picking granularity counterfactual granularity. It will try to find explanations that explain the decision boundary while balancing maximal granularity and faithfulness. Note that the granularity is chosen independently for each of the two dual explanations. Decreasing the granularity exponentially avoids a large increase in runtime compared to fixed-step decrease by exploiting the fact that users are likely to be more sensitive to the same constant sized decrease at high granularities than low granularities. I.e., going from a granularity of one token to two tokens feels more substantial than going from a granularity of eight tokens to nine tokens.

3.4.4 Summary

All three extensions introduced above fit together in our proposed method. Finally, we choose K to be 5 and ϵ to be 0.1 for all examples. Figure 3.5 provides a simplified overview of the steps that make up LEMON, while Algorithm 1

Algorithm 1: LEMON

Input: Matcher f, record pair (a, b), number of features K (default: 5), counterfactual margin ϵ (default: 0.1), min and max number of samples $S_{\min}$, $S_{\max}$ (default: 500, 3000)

Output: Pair of dual explanations (e_x^a, e_x^b). Each explanation $e_x = \{(w_i, p_i)\}_i$ consists of attribution and attribution potential for K chosen interpretable features.

Function `Explain(`r, o`)`**:**
- $x \leftarrow (r, o)$
- $\overline{CFS}^* \leftarrow -\infty$
- $e_x^* \leftarrow$ null
- $n \leftarrow 1$
- $N \leftarrow$ max num. of tokens in any non-empty string in r or 1
- **while** $n < 2N$ **do**
  - $x' \leftarrow$ interpretable representation of x for r with granularity of n tokens
  - $\mathcal{Z}_x \leftarrow \{\}$
  - $S \leftarrow \max(S_{\min}, \min(30 d_x, S_{\max}))$
  - **for** $i \in \{1, 2, \ldots, S\}$ **do**
    - Sample perturbation z' of x'
    - $y \leftarrow f(t_x(z'))$
    - $\mathcal{Z}_x \leftarrow \mathcal{Z}_x \cup \{(z', y)\}$
  - $\{(\hat{\beta}_i^A, \hat{\beta}_i^M)\}_i \leftarrow$ linear regression on $\mathcal{Z}_x$ weighted by π_x, selecting only K features using forward selection
  - $e_x \leftarrow \{(w_i, p_i)\}_i$ calculated from $\{(\hat{\beta}_i^A, \hat{\beta}_i^M)\}_i$
  - Calculate $\widehat{CFS}(e_x)$ and $CFS(e_x)$
  - **if** $\widehat{CFS}(e_x) \geq \epsilon \wedge CFS(e_x) \geq \epsilon$ **then**
    - **return** e_x
  - $\overline{CFS} \leftarrow \frac{\widehat{CFS}(e_x) \cdot CFS(e_x)}{\widehat{CFS}(e_x) + CFS(e_x)}$
  - **if** $\overline{CFS} > \overline{CFS}^*$ **then**
    - $\overline{CFS}^* \leftarrow \overline{CFS}$
    - $e_x^* \leftarrow e_x$
  - $n \leftarrow 2n$
- **return** e_x^*

$e_x^a \leftarrow$ `Explain(`a, b`)`

$e_x^b \leftarrow$ `Explain(`b, a`)`

return (e_x^a, e_x^b)

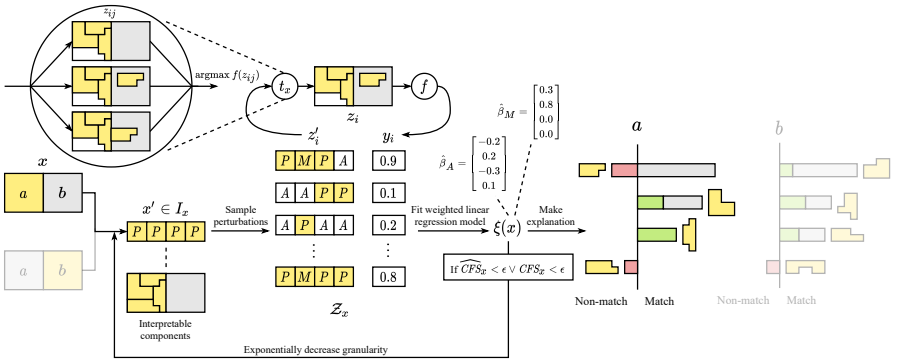


Figure 3.5: Illustration of how LEMON generates an explanation for a record pair $x = (a, b)$. The fundamental flow is similar to LIME in Figure 3.2, but have been significantly altered and expanded to support dual explanations, attribution potential and counterfactual granularity.

provides pseudocode. It is a model-agnostic and schema-flexible method without any hyperparameters that need tuning. One downside of LEMON is that, due to its extensions, it is more computationally demanding than LIME. The main reason is the increased number of matcher predictions made to estimate the attribution potential and finding the right granularity. However, in most cases it is still possible to generate an explanation within a few seconds.

Complexity. To analyze the runtime formally, we focus only on the number of predictions performed using the matcher f. This is reasonable because, for any non-trivial matcher, the runtime is completely dominated by the runtime of the matcher. Let F be the upper bound on the runtime of f for all possible perturbations of x. Furthermore, let N be the max number of tokens in any non-empty string in x or 1, S be the number of samples $|\mathcal{Z}_x|$ at 1 token granularity, and L be the max number of attribution potential samples for all perturbations of x. The time complexity of LEMON is then $O(FSL \log N)$. Technically, since S and L are bounded by constants, $O(F \log N)$ would also be accurate, but these factors are essential to understand the difference from similar methods. LIME [114], SHAP [86], and Landmark [7] (see Section 3.2) are all $O(FS)$, while gradient-based methods are typically $O(F)$. Ignoring that different methods have different strategies for choosing $|Z_x|$, the reason for LEMON's increased runtime compared to $LIME$ is the additional factor $L \log N$. Since L is low

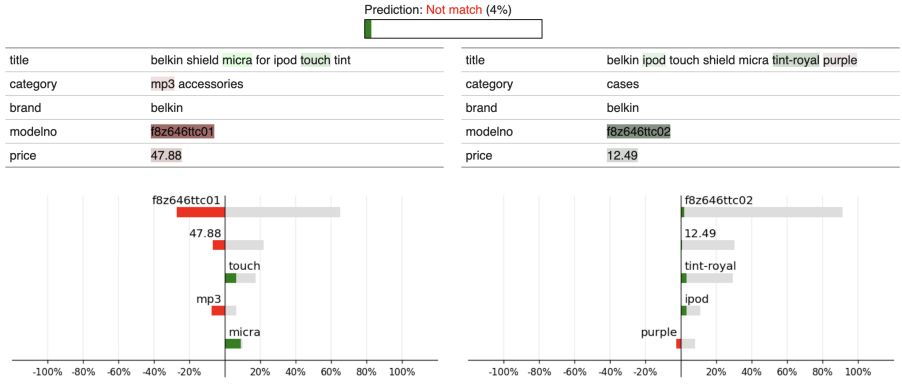


Figure 3.6: Example of how a LEMON explanation can be visualized and what we show users in our user study. This particular explanation is for the prediction of the BERT-Mini matcher used in our experiments on the record pair from Table 3.1.

and bounded we still get feasible runtime in practice. See Section 3.6.8 for an empirical evaluation. Note that an analysis of space complexity is less interesting since any non-trivial matcher and dataset will dominate the space requirements compared to the explanation method itself.

Explanations. One key advantage of LEMON over previous work is that it provides one type of explanation for all record pairs, whether the records match or not, with a clear and easy way to interpret and visualize. The attributions w_i are equivalent to those in LIME and can be interpreted in the same way. Its interpretation is to what degree interpretable feature i (some part of a record) contributes to the match score. If the corresponding part of the record is removed, we expect the match score to decrease by approximately w_i. For the same interpretable feature i, the interpretation of the attribution potential p_i is how much higher the attribution w_i could be if the other record matched the feature better.

While the explanation can be visualized in many ways, we propose a straightforward extension of the visualization proposed by the original LIME authors. Figure 3.6 shows an example. In addition to plotting a colored bar for each w_i, we also plot gray bars from w_i to $w_i + p_i$, outlining feature i's potential attribution.

3.5 Experimental Setup

3.5.1 Datasets

Table 3.1: The Public DeepMatcher [91] Benchmark Dataset and F_1 Score for the Magellan and BERT-Mini Matchers Used in the Experiments.

Type	Name	#Cand.	#Matches	Matcher F_1	
				MG	BM
Structured	Amazon-Google	11 460	1 167	0.52	0.67
	Beer	450	68	0.85	0.76
	DBLP-ACM	12 363	2 220	0.99	0.98
	DBLP-Scholar	28 707	5 347	0.94	0.93
	Fodors-Zagats	946	110	1.00	0.95
	iTunes-Amazon	539	132	0.90	0.93
	Walmart-Amazon	10 242	962	0.66	0.80
Dirty	DBLP-ACM	12 636	2 220	0.91	0.97
	DBLP-Scholar	28 707	5 347	0.83	0.94
	iTunes-Amazon	539	132	0.53	0.90
	Walmart-Amazon	10 242	962	0.41	0.79
Textual	Abt-Buy	9 575	1 028	0.51	0.81
	Company	112 632	28 200	0.57	0.90

All experiments are carried out on the 13 public datasets used in the evaluation of DeepMatcher [91] — originally from [71] and [29]. Table 3.1 lists them together with their number of candidates and number of matches. The datasets are divided into three types: structured, dirty, and textual. Structured datasets have nicely separated attributes. Dirty datasets are created from their structured counterpart by randomly injecting other attributes into the title attribute [91], and textual datasets generally consist of long textual attributes containing multiple pieces of information. For the company dataset, we truncate each record to max 256 space-separated words.

When we take a closer look at properties of the different explainability methods and the studied behavior is similar across all datasets we sometimes report only for a subset of the datasets or a single dataset (Abt-Buy) due to space restrictions. For those experiments one can assume the general behavior is similar on the other datasets.

3.5.2 Matchers

To show that our proposed method is versatile and model-agnostic, we perform our experiments for each dataset on both a matcher that uses classical machine learning with string metrics as features and on a deep learning based matcher. See Table 3.1 for their F_1 score on the benchmark datasets.

Magellan. For the classical approach, we train a Magellan [69] random forest matcher. We use the automatically suggested similarity features and the default random forest settings provided by the library. Furthermore, we do not downsample and train on the entire training dataset.

BERT-Mini. For the deep learning approach, we train a baseline Ditto [81] matcher using BERT-Mini [131]. While not quite achieving state-of-the-art accuracy, BERT-Mini provides a decent accuracy vs. cost trade-off while maintaining the main characteristics of state-of-the-art deep learning matchers and still significantly outperforming the classical matcher on dirty and textual data[5]. We use batch size 32, linearly decreasing learning rate from $3 \cdot 10^{-5}$ with 50 warmup steps, 16-bit precision optimization, and 1, 3, 5, 10, or 20 epochs depending on the dataset size. The final model is the one from the epoch with the highest F1 score on the validation dataset.

3.5.3 Baselines

We now introduce the baselines we use for comparison. For a fair comparison, we adopt dual explanations for all of them.

LIME. Since our work can be seen as a continuation of LIME [114], it is a natural baseline. We use LIME as described in Section 3.3.3.

Landmark. This is the most relevant work to ours (see Section 3.2). We use the source code provided by the authors[6] with default settings.

SHAP. Another popular approach for producing input attributions is SHAP [86]. It is based on the game-theoretic Shapley values [83, 122] and provides several methods for different types of models. For a fair comparison, we use their model-agnostic method, Kernel SHAP, which can be interpreted as using LIME to approximate Shapley values. Note that Kernel SHAP does not limit K and sets $\Omega(g) = 0$. We use default settings from the SHAP library.

[5]Since we perform an extensive set of experiments we want to be mindful of our usage of computational resources — both to reduce the energy footprint and keep the experiments as accessible as possible. For the purpose of this paper we consider this matcher to be sufficiently representative of state-of-the-art matchers. See Appendix 3.8 for results on the main experiments for a RoBERTa-based [84] DITTO matcher and DeepMatcher [91]

[6]`https://github.com/softlab-unimore/landmark`

Integrated gradients. Our proposed method is a perturbation-based attribution method. To compare against a gradient-based method, we use integrated gradients [124] as a baseline. This method is not truly model-agnostic as it requires gradients, so we can only apply it to our deep learning matcher.

Integrated gradients explain input x in reference to some baseline input x^* (some neutral input that gives a score close to zero). Let x be the embedding vector of a record pair. As the authors suggest for textual input, we let x^* be the zero embedding. The attribution for the ith element of x is then defined to be

$$IG_i(x_i) = (x_i - x_i^*) \times \int_{\alpha=0}^{1} \frac{\partial f(x^* + \alpha \times (x - x^*))}{\partial x_i} d\alpha \tag{3.7}$$

The integral is approximated by averaging the gradient of evenly spaced points from x^* to x. We use 50 points in our experiments. The raw attributions are for single elements of the embedded input, by no means interpretable for humans, so it is common to sum them for each embedding. To get attributions on the same representation level as our method, we combine attributions of Bert subword embeddings into whole words.

3.6 Experiments

We will now go through several experiments to evaluate LEMON and compare it to other methods. When we evaluate post hoc explainability, it is important to remember that we do not wish to measure the performance of the matchers, but rather what the explainability method can tell us about the matchers. Explanations should not be judged disconnected from the matcher on whether they provide the same rationale as users but to what degree they reflect the actual (correct or wrong) behavior of the matchers and to what degree they are effective at communicating this to users. Note that some experiments report only results for one or a few datasets when the results tend be similar, due to space constraints. Please see Appendix 3.9 for extensive results.

3.6.1 Counterfactual Interpretation

Explanations can sometimes provide enough information to the user to understand how the prediction could be different. The authors of [7] call this the "interest" of an explanation. We argue similarly that a useful explanation should implicitly reveal to the user some changes to the records that would flip the prediction outcome. But we further argue that we should help the user understand

Table 3.2: Counterfactual F_1 Score for All the Evaluated Explainability Methods Across All Datasets and the Two Matchers.

Model	Type	Method	Dataset													
			Structured							Dirty				Textual		
			AG	B	DA	DG	FZ	IA	WA	DA	DG	IA	WA	AB	C	Mean
Magellan	Match	LIME	0.96	0.83	**1.00**	0.87	0.77	0.98	0.82	0.50	0.79	0.90	0.86	0.95	0.47	0.82
		SHAP	0.95	**1.00**	**1.00**	**1.00**	0.95	**1.00**	0.83	0.65	0.96	0.68	0.81	0.98	0.68	0.88
		SHAP (w/ CFG)	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.98	0.99	**1.00**	0.91	**0.99**	**0.99**	**0.87**	**0.98**
		Landmark	0.92	0.89	**1.00**	0.96	0.73	0.87	0.95	0.75	0.74	0.88	0.90	0.95	0.28	0.83
		LEMON (w/o DE)	0.98	0.91	**1.00**	0.97	0.95	**1.00**	0.96	0.83	0.93	0.93	0.95	0.98	0.49	0.91
		LEMON (w/o AP)	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	0.98	0.83	**0.98**
		LEMON (w/o CFG)	0.96	0.89	**1.00**	0.88	0.91	0.98	0.81	0.52	0.81	0.85	0.83	0.92	0.41	0.83
		LEMON	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.98	**0.99**	0.80	**0.98**
	Non-match	LIME	0.02	0.11	0.02	0.01	0.02	0.14	0.03	0.09	0.10	0.17	0.10	0.13	0.11	0.08
		SHAP	0.00	0.03	0.00	0.01	0.00	0.05	0.02	0.07	0.06	0.10	0.12	0.06	0.13	0.05
		SHAP (w/ CFG)	0.02	0.22	0.01	0.02	0.00	0.05	0.02	0.09	0.09	0.23	0.21	0.08	**1.00**	0.16
		Landmark	0.14	**0.84**	**0.14**	0.20	0.23	0.21	**0.93**	0.04	0.43	0.39	0.03	0.79	0.09	0.34
		LEMON (w/o DE)	0.59	0.78	0.06	0.37	0.65	0.64	0.82	0.64	0.69	**0.87**	0.82	**0.88**	0.96	0.68
		LEMON (w/o AP)	0.04	0.42	0.02	0.03	0.02	0.14	0.05	0.17	0.21	0.26	0.36	0.17	0.68	0.20
		LEMON (w/o CFG)	0.40	0.46	0.08	0.13	0.03	0.24	0.73	0.23	0.49	0.69	0.63	0.78	0.13	0.38
		LEMON	**0.71**	0.50	0.12	**0.54**	**0.98**	**0.77**	0.76	**0.75**	**0.78**	**0.87**	**0.87**	0.87	0.96	**0.73**
BERT-Mini	Match	LIME	0.95	0.65	0.97	0.85	0.93	0.65	0.81	0.97	0.69	0.62	0.80	0.78	0.18	0.76
		SHAP	0.90	0.81	0.79	0.65	0.91	0.69	0.71	0.79	0.62	0.63	0.75	0.77	0.25	0.71
		SHAP (w/ CFG)	0.95	0.96	0.92	0.98	0.86	0.89	0.97	0.98	0.99	**1.00**	0.99	**1.00**	**0.39**	0.91
		IG	0.90	0.43	0.71	0.83	0.50	0.70	0.67	0.69	0.89	0.81	0.68	0.79	0.33	0.69
		IG (w/ CFG)	0.88	0.52	0.66	0.94	0.68	0.70	0.77	0.86	0.95	0.94	0.87	0.93	0.28	0.77
		Landmark	0.98	0.93	**1.00**	0.94	0.86	**0.94**	0.84	0.99	0.83	0.90	0.88	0.83	0.08	0.85
		LEMON (w/o DE)	0.99	0.69	**1.00**	0.97	0.98	0.88	0.83	**1.00**	0.94	0.89	0.84	0.86	0.25	0.85
		LEMON (w/o AP)	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.94**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	0.98	**0.39**	**0.95**
		LEMON (w/o CFG)	0.95	0.65	0.98	0.86	0.98	0.58	0.81	0.99	0.70	0.59	0.81	0.79	0.15	0.76
		LEMON	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.94**	0.98	**1.00**	0.99	**1.00**	**1.00**	0.97	0.37	0.94
	Non-match	LIME	0.13	0.06	0.01	0.04	0.04	0.13	0.08	0.02	0.04	0.23	0.07	0.05	0.03	0.07
		SHAP	0.14	0.16	0.01	0.04	0.01	0.29	0.08	0.02	0.05	0.33	0.14	0.15	0.18	0.12
		SHAP (w/ CFG)	0.14	0.26	0.02	0.04	0.01	0.34	0.11	0.02	0.05	0.49	0.17	0.37	0.26	0.18
		IG	0.07	0.00	0.00	0.03	0.01	0.00	0.02	0.01	0.01	0.00	0.03	0.07	0.02	0.02
		IG (w/ CFG)	0.08	0.03	0.00	0.04	0.01	0.00	0.03	0.02	0.02	0.02	0.06	0.08	0.03	0.03
		Landmark	0.40	0.70	0.05	0.17	0.50	0.63	0.64	0.07	0.35	0.50	0.74	0.67	0.01	0.42
		LEMON (w/o DE)	0.75	0.93	0.55	**0.68**	**0.86**	0.91	**0.93**	**0.74**	0.78	**0.88**	**0.97**	0.93	**0.97**	**0.84**
		LEMON (w/o AP)	0.18	0.08	0.03	0.08	0.05	0.67	0.14	0.04	0.09	0.56	0.16	0.23	0.26	0.20
		LEMON (w/o CFG)	0.50	0.92	0.02	0.19	0.85	0.94	0.89	0.04	0.18	0.76	0.95	0.90	0.96	0.62
		LEMON	**0.81**	**0.94**	**0.65**	**0.68**	**0.86**	**0.97**	0.90	0.50	**0.79**	0.87	0.95	**0.98**	**0.97**	**0.84**

a minimal number of such changes necessary, since that would mean the user has a greater understanding of where the decision boundary is.

To that end, we simulate users being shown an explanation for a record pair and then being asked what they think would be some minimal changes to the records that would flip the matcher's prediction. The simulated users will greedily try to make the smallest number of perturbations necessary according to the explanation, as described in Section 3.4.3. Of the two dual explanations, they pick the explanation with the lowest k_g if $\widehat{CFS}(e_x) \geq \epsilon$ or the one with the highest $\widehat{CFS}(e_x)$ otherwise. We extend the same greedy strategy to Landmark explanations but with their corresponding perturbations. Let the *counterfactual recall* of an attribution method be the fraction of explanations where at least one of the two dual explanations indicate how the matching prediction could be flipped ($\widehat{CFS}(e_x) \geq \epsilon$), and the *counterfactual precision* be the fraction of those where the greedy counterfactual strategy is actually successful ($CFS(e_x) > 0$). To unify them into a single metric, we report the counterfactual F_1 score. We formalize this in the following definition.

Definition 1 (Counterfactual Recall, Precision, and F_1) *Let λ be an entity matching attribution method that outputs dual explanations (e_x^a, e_x^b) and let $C \subseteq A \times B$ be a collection of pairs (a, b) (i.e., a dataset). The counterfactual recall of the method λ for the matcher f on the record pair collection C is*

$$CR(\lambda, f, C) = \mathbb{E}_{(a,b)\sim C}\Big[\max\big(\widehat{CFS}(e_x^a), \widehat{CFS}(e_x^b)\big) \geq \epsilon\Big]$$

where $[\dots]$ are Iverson brackets and we assume $(e_x^a, e_x^b) = \lambda(f, (a, b))$. Furthermore, let the recalled pairs in C be

$$C_r = \Big\{(a, b) | (a, b) \in C \wedge \max\big(\widehat{CFS}(e_x^a), \widehat{CFS}(e_x^b)\big) \geq \epsilon\Big\}$$

and let the greedy pick among the dual explanations be

$$e_x^g = \begin{cases} e_x^a, & \text{if } k_g(e_x^a) < k_g(e_x^b) \\ & \text{or } k_g(e_x^a) = k_g(e_x^b) \wedge \widehat{CFS}(e_x^a) \geq \widehat{CFS}(e_x^b) \\ e_x^b, & \text{otherwise} \end{cases}$$

The counterfactual precision of the method λ for the matcher f on the record pair collection C is

$$CP(\lambda, f, C) = \mathbb{E}_{(a,b)\sim C_r}\big[CFS(e_x^g) > 0\big]$$

Lastly, the counterfactual F_1 score is then simply

$$CF_1(\lambda, f, C) = \frac{2CR(\lambda, f, C) \cdot CP(\lambda, f, C)}{CR(\lambda, f, C) + CP(\lambda, f, C)}$$

We produce 500 explanations for both predicted matches and non-matches (or all when there are less than 500 available) for each explanation method per dataset[7]. Table 3.2 shows the counterfactual F_1 for the different explainability methods, matchers, and datasets[8].

We observe that LEMON performs best overall, with the highest or close to the highest F_1 score in most cases. It significantly outperforms all three baselines, where non-matches, as expected, have the most pronounced difference. Since all the baselines are fundamentally analyzing the prediction by observing what happens when features are removed, they suffer from the same issue of explaining non-matches as discussed in Section 3.4.2. Importantly, the low performance of all the baselines backs up the claim that standard local post hoc attribution methods do not work satisfactorily out of the box for entity matching. Our proposed method generally outperforms Landmark, with the exception of the three datasets for the Magellan matcher on non-matches. We note that LEMON has the biggest advantage over Landmark on datasets that typically would require more substantial perturbations to flip the prediction, such as matches in DBLP-GoogleScholar and Company. At the same time, it is clear that all methods struggle with non-matches on DBLP-ACM and DBLP-GoogleScholar (and Beer to a certain degree) more than other datasets — especially for Magellan. This is mainly because the datasets yield a binary classification problem with large margins for the decision boundary. The classification problem is too easy and the matchers too certain. The true matches contain many highly similar attributes, while true non-matches tend to have several significantly dissimilar attributes. Changing the matcher's prediction from non-match to match is hard because it requires many perturbations across most attributes. Therefore, the reason Landmark performs better in some cases with non-matches for Magellan is mainly because Landmark does not restrict the number of interpretable features to use in the explanation. This enables higher counterfactual recall at the expense of more complex and less specific explanations.

[7]The same pairs are used for the different explanation methods.
[8]See Appendix 3.9 for counterfactual precision and recall numbers.

3.6.2 Explanation Faithfulness

It is desirable that explanations are faithful to the matcher. All useful explanations provide some simplified view of the matcher's behavior, but we still want them to be indicative of how the matcher actually operates without being unnecessarily misleading. Inspired by [114], we make perturbations to a record pair and compare the resulting match score with what we would expect from the attributions and attribution potentials. Specifically, if we remove feature i, we expect the match score to decrease with w_i (remember that w_i can be negative). Ff we inject feature i into the other record, we expect the match score to increase with p_i. The same applies to Landmark, but with appending instead of injecting features. Since the baselines do not estimate attribution potentials, we ignore that perturbation for them. We perform 1, 2, and 3 random perturbations among the interpretable features for both dual explanations. We repeat for 500 explanations of matches and non-matches for each matcher and dataset (or all when there are less than 500 available). Let δ_l be the set of expected match score increases and decreases for experiment l out of L, and let the mean absolute error be

$$MAE = \frac{1}{L} \sum_{l} \left| f(z) - \left[f(x) + \sum_{c \in \delta_l} c \right] \right| \tag{3.8}$$

This error measure will favor conservative explanation methods that make small and insignificant claims, and punish methods like Landmark and LEMON that provide higher impact explanations because of the injected/appended features. Therefore, we define the perturbation error to be the mean absolute error by dividing by the average magnitude of the predicted change:

$$PE = \frac{MAE}{\frac{1}{L} \sum_{l} \sum_{c \in \delta_l} |c|} \tag{3.9}$$

Table 3.3 shows the perturbation error for all methods. No method achieves truly low error levels, which is expected given the simplified assumption of independent additative attributions. However, we observe that LEMON overall is the method with the smallest errors, with LIME performing very similarly. LEMON and LIME lie in the range of 0.25 to 0.75 in almost all cases, while SHAP, IG, and Landmark often exceed 1.0. Further, we see Landmark sometimes gets extremely high perturbation error, especially for Magellan on the dirty datasets like Dirty iTunes-Amazon. Upon closer inspection, we think this stems from a combination of sampling a too large neighborhood and the ineffec-

Table 3.3: Perturbation Error *PE* for All the Evaluated Explainability Methods Across All Datasets and the Two Matchers.

Model	Type	Method	Dataset													
			Structured							Dirty				Textual		
			AG	B	DA	DG	FZ	IA	WA	DA	DG	IA	WA	AB	C	Mean
Magellan	Match	LIME	0.33	0.38	0.31	0.34	0.31	**0.25**	0.67	0.47	0.41	**0.35**	0.46	0.33	0.60	0.40
		SHAP	1.09	1.03	0.96	1.01	0.80	0.93	1.12	1.03	1.25	2.37	2.50	1.07	7.70	1.76
		SHAP (w/ CFG)	1.03	0.79	0.96	1.00	0.79	0.95	0.31	0.64	1.17	1.24	1.28	1.04	3.58	1.14
		Landmark	0.95	0.73	0.70	1.18	0.67	0.94	0.86	7.01	2.93	5.40	1.93	1.25	3.45	2.15
		LEMON (w/o DE)	0.36	0.38	**0.30**	0.36	0.33	**0.25**	0.35	0.45	0.40	0.40	0.37	0.33	0.68	0.38
		LEMON (w/o AP)	0.33	0.40	0.31	**0.33**	**0.28**	**0.25**	**0.23**	**0.42**	**0.39**	0.37	**0.35**	0.33	**0.53**	**0.35**
		LEMON (w/o CFG)	0.33	**0.36**	**0.30**	0.34	0.30	0.26	0.27	0.45	0.40	**0.35**	0.39	0.33	0.67	0.37
		LEMON	**0.32**	0.37	**0.30**	**0.33**	**0.28**	0.26	0.28	**0.42**	**0.39**	**0.35**	0.36	**0.32**	0.54	**0.35**
	Non-match	LIME	0.53	0.64	**0.42**	0.47	0.58	0.34	0.45	0.56	0.60	0.46	0.59	0.46	0.79	0.53
		SHAP	0.89	0.93	1.34	1.08	1.43	1.10	1.16	1.44	1.70	2.15	1.24	1.34	6.36	1.71
		SHAP (w/ CFG)	0.62	0.52	0.79	0.80	1.12	0.38	0.64	0.83	0.79	0.99	0.68	0.77	0.82	0.75
		Landmark	0.64	0.74	0.81	0.79	0.76	1.45	0.63	0.90	1.04	5.58	3.05	1.34	2.10	1.53
		LEMON (w/o DE)	0.44	0.52	0.63	0.50	0.46	0.37	**0.38**	0.59	0.51	**0.40**	0.46	**0.33**	**0.47**	0.47
		LEMON (w/o AP)	0.46	**0.35**	0.47	**0.46**	0.51	**0.21**	0.40	**0.49**	0.51	0.44	**0.41**	0.40	0.59	**0.44**
		LEMON (w/o CFG)	0.43	0.55	0.44	0.50	**0.43**	0.51	**0.38**	0.58	0.52	0.44	0.48	0.35	0.78	0.49
		LEMON	**0.42**	0.59	0.64	0.53	0.45	0.43	0.40	0.54	**0.49**	0.46	0.47	0.37	**0.47**	0.48
BERT-Mini	Match	LIME	**0.32**	0.47	0.39	**0.40**	**0.30**	0.44	0.61	**0.38**	**0.41**	0.48	0.62	0.77	1.20	0.52
		SHAP	0.67	0.50	1.48	0.87	0.65	0.61	1.14	1.37	0.85	0.81	1.08	0.82	2.18	1.00
		SHAP (w/ CFG)	0.64	0.57	1.03	0.78	0.61	0.71	0.74	0.91	0.71	0.71	0.71	0.60	0.60	0.72
		IG	1.20	0.76	1.64	1.15	1.01	0.80	1.18	1.65	1.05	1.16	1.17	0.89	1.93	1.20
		IG (w/ CFG)	1.16	0.72	1.45	1.06	0.87	0.93	1.11	1.28	0.92	0.89	0.95	0.80	0.84	1.00
		Landmark	0.92	1.00	1.17	0.81	0.55	0.74	0.94	1.20	0.78	1.01	1.14	0.79	0.91	0.92
		LEMON (w/o DE)	0.38	0.51	0.50	0.45	0.41	0.41	0.50	0.52	0.46	0.48	0.51	0.56	**0.40**	0.47
		LEMON (w/o AP)	**0.32**	**0.40**	0.45	0.43	0.33	**0.31**	**0.45**	0.40	**0.41**	**0.30**	**0.42**	0.49	0.61	**0.41**
		LEMON (w/o CFG)	0.33	0.42	**0.38**	**0.40**	0.31	0.43	0.51	**0.38**	**0.41**	0.44	0.54	0.60	0.68	0.45
		LEMON	0.33	0.45	0.45	0.44	0.40	0.43	0.49	0.42	0.44	0.39	0.46	**0.47**	0.53	0.44
	Non-match	LIME	0.50	0.51	0.68	**0.46**	0.61	0.50	0.61	0.58	0.52	0.53	0.67	0.63	0.61	0.57
		SHAP	0.67	0.82	0.75	0.82	0.87	0.79	0.89	0.81	0.75	1.04	0.79	0.86	1.32	0.86
		SHAP (w/ CFG)	0.65	0.80	0.77	0.75	0.79	0.77	0.75	0.76	0.68	0.78	0.71	0.80	0.62	0.74
		IG	0.95	1.01	1.05	1.02	1.71	1.80	1.27	0.73	0.88	2.73	1.03	0.88	1.31	1.26
		IG (w/ CFG)	0.94	1.06	1.19	0.98	2.31	5.08	1.29	0.90	0.86	4.38	1.14	1.08	2.34	1.81
		Landmark	0.76	0.83	0.84	0.79	0.72	0.65	0.72	0.85	0.83	0.83	0.74	1.03	0.80	0.80
		LEMON (w/o DE)	0.53	0.34	0.81	0.66	0.45	0.43	**0.39**	0.72	0.61	0.50	0.36	**0.33**	**0.42**	**0.50**
		LEMON (w/o AP)	**0.48**	0.49	**0.63**	0.54	0.58	0.47	0.53	**0.57**	**0.50**	**0.48**	0.58	0.63	0.62	0.55
		LEMON (w/o CFG)	0.51	**0.29**	0.98	0.70	0.45	**0.42**	**0.39**	0.81	0.71	**0.48**	**0.35**	**0.33**	**0.42**	0.53
		LEMON	0.49	0.40	0.87	0.58	**0.43**	0.48	0.46	0.77	0.54	**0.48**	0.38	**0.33**	**0.42**	0.51

tiveness of the double-entity generation strategy when the data does not follow the matched schemas (i.e., is dirty).

3.6.3 User Study

Table 3.4: Counterfactual Precision of Users After Being Shown an Explanation From LIME or LEMON.

Dataset	Method			
	LIME		LEMON	
	Match	Non-match	Match	Non-match
Structured				
Amazon-Google	0.71	0.22	**0.77**	**0.42**
Beer	0.47	0.16	**0.50**	**0.63**
DBLP-ACM	**0.82**	0.06	0.79	**0.23**
DBLP-GoogleScholar	0.53	0.08	**0.62**	**0.27**
Fodors-Zagats	0.59	0.14	**0.69**	**0.46**
iTunes-Amazon	0.45	0.18	**0.60**	**0.60**
Walmart-Amazon	0.55	0.24	**0.63**	**0.58**
Dirty				
DBLP-ACM	0.71	0.02	**0.81**	**0.40**
DBLP-GoogleScholar	0.45	0.08	**0.58**	**0.48**
iTunes-Amazon	0.53	0.22	**0.75**	**0.56**
Walmart-Amazon	0.53	0.24	**0.62**	**0.71**
Textual				
Abt-Buy	0.55	0.14	**0.62**	**0.73**
Company	0.14	0.16	**0.29**	**0.35**
Mean	0.54	0.15	**0.63**	**0.49**

To examine if explanations from LEMON improve human subjects understanding of a matcher compared to LIME, we adopt the experiment on counterfactual interpretation from Section 3.6.1 to human subjects. We recruit random test users from the research survey platform Prolific. Note that these users are laymen and do not have any experience with entity matching or a background in computer science. A user is shown an explanation for a record pair and then asked what they think would be a minimal change to the record pair that would make the matcher predict the opposite. Each user is shown one explanation for a match and a non-match for each dataset, and we use only the BERT-Mini matcher. Afterward, we check what fraction of them successfully gets the opposite prediction — i.e., the counterfactual precision. We conduct the experiment on 50 users for LIME and 50 different users for LEMON, and report the counterfactual precision in Table 3.4.

As expected, and in line with the experiments above, the greatest improvement is for non-matches. We see an average improvement in the counterfactual precision of 0.09 for matches and 0.34 for non-matches. The results are generally less pronounced than those of the simulated experiments. We suspect the lower maximum scores reflect the difficulty of the task for a layman, and that the higher minimum scores reflect human ability to use common sense to make up for weak explanations. Note that we cannot compare to Landmark [7] since the authors do not propose any way of presenting an actual explanation to a user. The combination of double-entity generation and not limiting $\Omega(g)$ (i.e., explaining using all features instead of limiting them to K) makes such a presentation non-trivial.

3.6.4 Ablation Study

Included in Table 3.2 and Table 3.3 is also an ablation study. We examine what happens when we remove each of the three main components of LEMON: 1) Dual Explanations. Instead of dual explanations, we produce one joint explanation for both records. We use $K = 10$ for a fair comparison. 2) Attribution Potential. We use the interpretable representation of the LIME baseline and do not estimate any attribution potential. 3) Counterfactual Granularity. We fix the granularity to be one token. In addition, we examine the effect of adding counterfactual granularity to the baselines SHAP and integrated gradients (there is no trivial way to do the same for attribution potential).

LEMON performs better across the board for matches with dual explanations, but the results more varied for non-matches. This makes sense since the problematic interaction effects mainly occur when two records match and have a lot of similar content. Unsurprisingly, since its primary goal is to explain how records could match better, attribution potential only significantly improves non-match explanations. Nevertheless, the improvement for non-matches is dramatic, demonstrating how effective attribution potential is for explaining record pairs that do not match. Finally, we observe that the effectiveness of counterfactual granularity varies greatly from dataset to dataset. It makes the most difference on datasets where we consider the records to have multiple high-quality pieces of information — either in the form of several high-quality attributes or long textual attributes with multiple high-quality keywords.

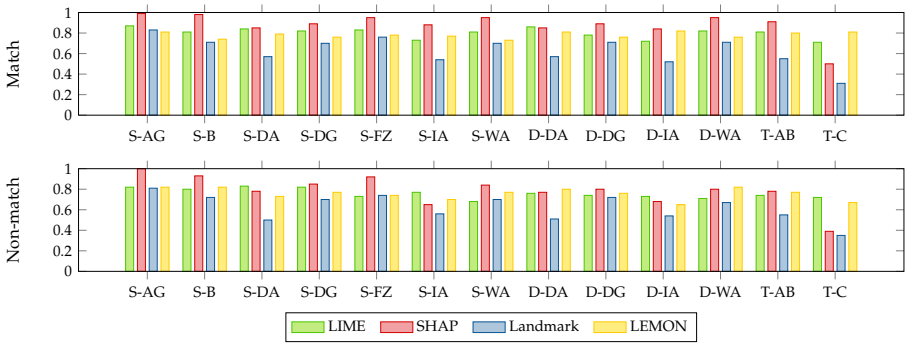


Figure 3.7: Stability of explainability methods based on neighborhood sampling for BERT-Mini on all datasets.

3.6.5 Stability

Several of the benchmarked methods, including LEMON itself, rely on random sampling of the neighborhood of x. Different initial random seeds will result in different explanations. However, with sufficient samples we would like a well-behaved method to generate similar explanations — i.e. explanations to be stable and not change much if different random seeds are used. Stability is a desirable trait from a trust perspective, but also especially useful when examining or debugging a matcher. If we make changes to a matcher, we want to be confident that the differences we observe in the explanations mostly reflect the matcher changes and not instability of the explanation method. LEMON not only relies on sampling the neighborhood of x in the interpretable domain I_x, but also on sampling to approximate t_x when translating from I_x. A natural question to ask is if this additional random sampling hurts stability.

Let $e_x^1 = \{(w_{i1}, p_{i2})\}_{i \in E_1}$ and $e_x^2 = \{(w_{i2}, p_{i2})\}_{i \in E_2}$ be two explanations for the same input x and matcher f with a different random seed. Let the the similarity between the two explanations $s(e_1, e_2) = \frac{e_1 \cap e_2}{e_1 \cup e_2}$ be the weighted Jaccard coefficient such that the intersection is

$$e_x^1 \cap e_x^2 = \sum_{i \in E_1 \cap E_2} \Big[(w_{i1} \dot{\cap} w_{i2}) + (p_{i1} \dot{\cap} p_{i2}) \Big] \tag{3.10}$$

and the union is

$$e_x^1 \cup e_x^2 = \sum_{i \in E_1 \cap E_2} \max(|w_{i1}|, |w_{i2}|) + \max(|p_{i1}|, |p_{i2}|) + \sum_{i \in E_1 \setminus E_2} (w_{i1} + p_{i1}) + \sum_{i \in E_2 \setminus E_1} (w_{i2} + p_{i2}) \tag{3.11}$$

where $\dot{\cap}$ is a shorthand for

$$r \dot{\cap} q = H(rq) \min(|r|, |q|) \tag{3.12}$$

and H is the unit step function. For LIME and SHAP, p_i is 0 for all i. To be able to compare explanations of different granularity, all explanations are normalized to single token interpretable features — i.e. if feature i is an n-token interpretable feature we split it into n features with attribution $\frac{w_i}{n}$ and attribution potential $\frac{p_i}{n}$. Finally, we define the stability of an explanation method as the expected similarity between two explanations $\mathbb{E}_x\left[s(e_x^1, e_x^2)\right]$. Note that this definition slightly favors methods such as SHAP and Landmark that uses all interpretable features in its explanations instead of only the K most important like LIME and LEMON. Picking the K most important interpretable features controls the explanation complexity at the cost of exposing the method to more instability because small changes in importance can change which features are within or outside top K.

Figure 3.7 shows the estimated stability of LIME, SHAP, Landmark, and LEMON for BERT-Mini on all datasets. For each dataset, we sample 100 predicted matches and non-matches uniformly at random, generate two explanations with different random seed for each example, and average the similarities. We see that LEMON is relatively stable and is similar to LIME in terms of stability. This is important because it shows LEMON does not degrade in stability despite the sampling-based approximation of t_x. SHAP is overall the most stable method, while Landmark is the least stable. To understand these differences in stability it is important to also take into account the sample size.

3.6.6 Neighborhood Sample Size

From our experience, the main concern when choosing the neighborhood sample size $|\mathcal{Z}_x|$ is stability. From Figure 3.8 we see that one achieves satisfactory counterfactual F_1 score and perturbation error with relatively few samples, but as we will see, it takes considerably more samples to get stable explanations.

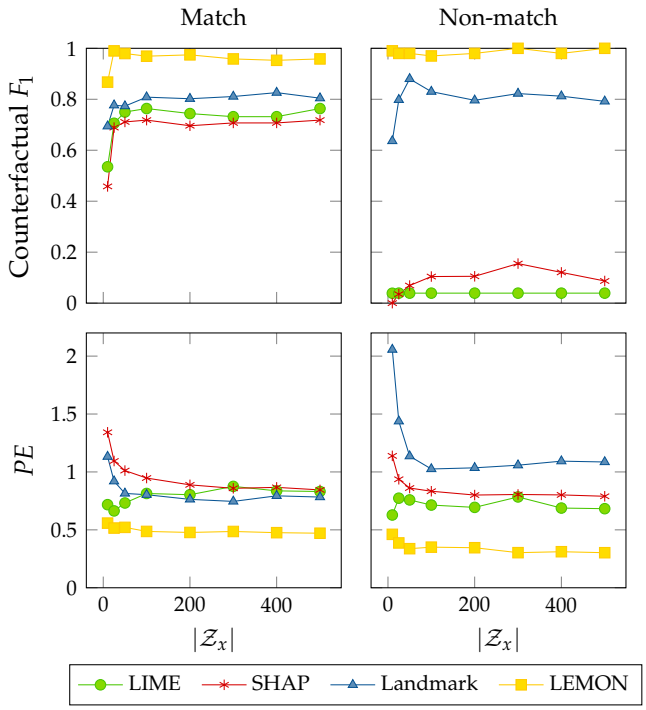


Figure 3.8: Counterfactual F_1 and perturbation error (PE) of explainability methods based on neighborhood sampling for BERT-Mini on Abt-Buy when varying the neighborhood sampling size $|\mathcal{Z}_x|$.

Thus, picking $|\mathcal{Z}_x|$ mostly boils down to a trade-off between stability and speed (see 3.6.8 for a discussion about runtime).

The different neighborhood sampling-based methods have different strategies for picking a sample size[9], so it is interesting to compare the stability at equal sample sizes. Figure 3.9 shows the stability of the explainability methods for BERT-Mini on the Abt-Buy dataset when we vary the neighborhood sample size. As before, we sample 100 predicted matches and non-matches, generate two explanations per example, and estimate the stability to be the average similarity

[9]SHAP defaults to $2d_x + 2048$, Landmark to 500, and our LIME baseline and LEMON to $\max(500, \min(30d_x, 3000))$.

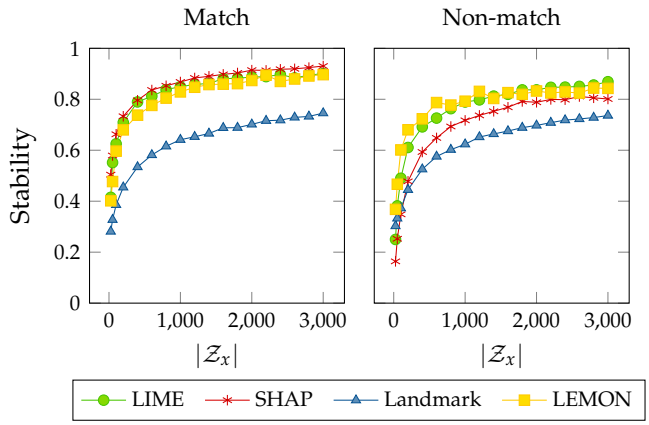


Figure 3.9: Stability of explainability methods based on neighborhood sampling for BERT-Mini on the Abt-Buy dataset when varying the neighborhood sampling size $|\mathcal{Z}_x|$.

between the explanation pairs. We observe that LEMON is close to or equally sample efficient as LIME and SHAP for matches, and slightly more for non-matches. This shows that the difference in stability between SHAP and LEMON is mainly a matter of difference in sample sizes. We deliberately use a less aggressive sampling scheme for LEMON than SHAP because we find the returns in terms of stability diminishing — especially given the higher computational footprint of LEMON. Landmark's instability, however, can not be attributed to the lower sampling size. It is clear from Figure 3.9 that the method is significantly less sampling efficient than the others. We suspect this is mostly due to the large neighborhood used when sampling.

3.6.7 Explanation Complexity

An important distinction between LEMON and Landmark is that LEMON, as LIME, limits the explanation complexity $\Omega(g)$ by constraining the number of interpretable features used in an explanation to $K < d_x$. This is important because we can not generally expect users to consume explanations with a large number of interpretable components. We consider $K = 5$ default for LEMON and have used this for all experiments, since we consider this a reasonable number of features for user consumption in practice. Furthermore, we argue that

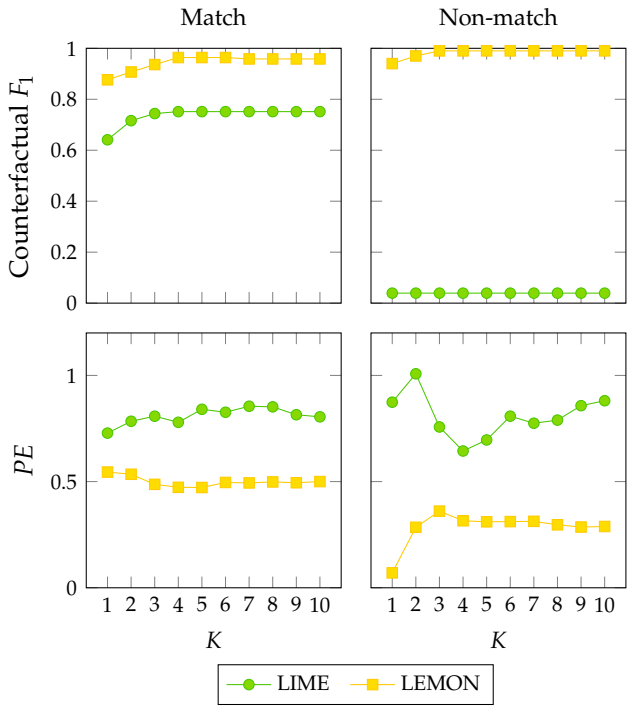


Figure 3.10: Counterfactual F_1 and perturbation error (PE) of explainability methods based on neighborhood sampling for BERT-Mini on Abt-Buy when varying K.

the choice of K is indeed mostly a matter of what is practical to the user. Figure 3.10 shows how the counterfactual F_1 score and perturbation error vary depending on the choice of K for LIME and LEMON (remember SHAP and Landmark use all interpretable features). We see that for $K \geq 3$ the counterfactual interpretation and explanation faithfulness is not affected much by the choice of K. For very low values of K we lose the necessary expressive power needed to capture the matcher's behavior — which makes it hard to produce counterfactually interpretable explanations.

3.6.8 Runtime

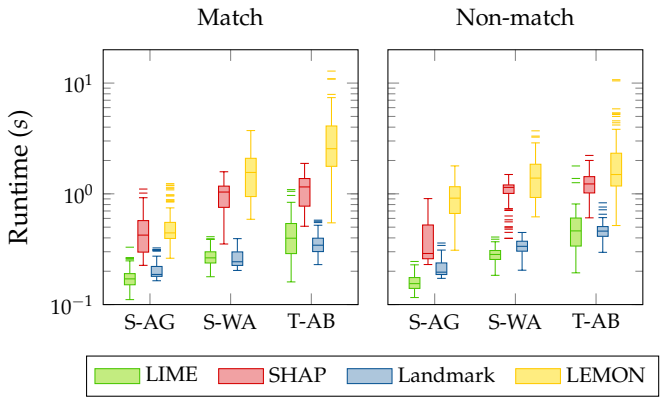


Figure 3.11: Runtime of explainability methods based on neighborhood sampling for BERT-Mini on three different datasets.

One of the main disadvantages of local post hoc neighborhood sampling methods are long runtimes. This is a result of having to do inference on a large number of sampled inputs. Of course, LEMON is more prone to this than existing work due to the approximation of t_x and counterfactual granularity. Figure 3.11 shows the time needed to make a single explanation of a BERT-Mini matcher prediction for three different datasets on a NVIDIA RTX 2080 Ti for the different neighborhood sampling-based methods. Each boxplot shows the distribution of 100 explanations. LEMON generally takes the longest time, with SHAP being most comparable. Note that the runtime varies significantly for every methods even on the same dataset. This is because inference time depends heavily on the input size, which varies between record pairs and depend on the random perturbation. LEMON's runtime varies more because of how the counterfactual granularity is found.

While the runtime is longer than in previous work, we argue it is still within reason for most applications on most datasets — especially taking into consideration the improvement in explanation quality seen in Section 3.6.1, 3.6.2, and 3.6.3. Moreover, it is possible to trade off some stability for shorter runtime if desired. As mentioned in Section 3.6.6, the choice of $|\mathcal{Z}_x|$ is essentially a trade-off between stability and speed. Figure 3.12 plots the stability against the median runtime for explaining BERT-Mini on the Abt-Buy dataset (one of the

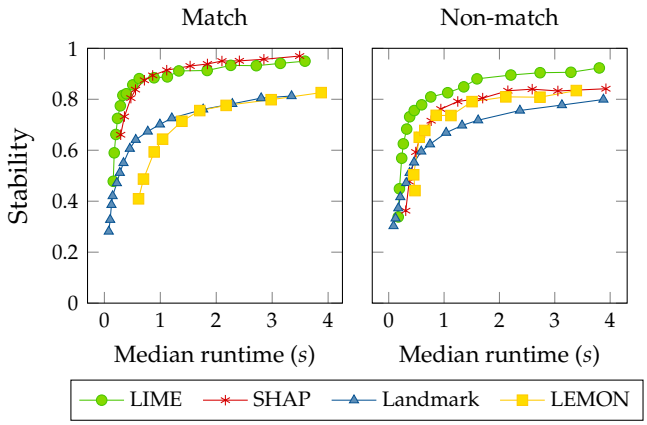


Figure 3.12: Stability of explainability methods based on neighborhood sampling for BERT-Mini on Abt-Buy when varying the median runtime (by changing $|\mathcal{Z}_x|$).

datasets with the longest runtime) when we vary $|\mathcal{Z}_x|$. We see that LEMON has a stability-runtime trade-off comparable to Landmark. By reducing the neighborhood sample size we can achieve more similar runtime to Landmark at the expense of also getting similar (low) stability as Landmark. To what degree depends on the dataset, but there is significant flexibility if lower runtime is critical.

3.7 Conclusion

Local post hoc feature attribution is a valuable and popular type of explainability method that can explain any classifier, but standard methods leave significant room for improvement when applied to entity matching. We have identified three challenges of applying such methods to entity matching and proposed LEMON, a model-agnostic and schema-flexible method that addresses all three challenges. Experiments and a novel evaluation method for explainable entity matching show that our proposed method is more faithful to the matcher and more effective in explaining to the user where the decision boundary is — especially for non-matches. Lastly, user studies support a real-world improvement in understanding for a layman seeing LEMON explanations compared to naive

LIME explanations.

There is still much to be done within explainable entity matching. A disadvantage of LEMON (and other perturbation-based methods like LIME, SHAP, and Landmark) is their running time. Even though it is possible to trade off significantly shorter running time for explanation stability, and trivial to parallelize the computational bottleneck (running inference of matcher f), depending on the hardware and matcher, it might still be infeasible in practice to do real-time explanation or generate explanations for all record pairs in large datasets, while still achieving satisfactory stability. Therefore, more efficient sampling strategies should be explored. Furthermore, there is more to be done on examining the adaptation of other explainability methods for entity matching in-depth, and on how to evaluate them. Our experiments and ablation study show that dual explanations and counterfactual granularity are easily applicable to SHAP and gradient-based methods, and that they are indeed effective for other methods than LIME. It is less clear how one would adapt the ideas of attribution potential to those methods, and we hope to address that in the future.

3.8 Other Matchers

In addition to Magellan and BERT-Mini, it is also interesting to evaluate LEMON on larger transformer models and other deep learning architectures. To that end we perform the experiments on counterfactual interpretation and explanation faithfulness from Section 3.6.1 and 3.6.2 on a RoBERTa-based [84] baseline DITTO model and DeepMatcher [91].

3.8.1 DeepMatcher

The authors [91] explore a range of different deep learning models for entity matching. We use their hybrid model since it performs the best overall. Each model is trained for 15 epochs with a batch size of 32 and a negative to positive sampling ratio of 3. The model is evaluated on the validation set after every epoch and the best model is kept. Note that we do not perform an exhaustive hyperparameter search like the authors and instead use default settings as provided by the publicly available implementation[10] from the authors — which gives performance reasonably close to what they report.

[10] `https://github.com/anhaidgroup/deepmatcher`

3.8.2 RoBERTa

The authors of DITTO [81] evaluated a number of prominent transformer-based natural language models and found RoBERTa [84] to generally perform the best. We train each baseline RoBERTa-based DITTO model the same way as described in Section 3.5.2 for BERT-Mini.

3.8.3 Results

Table 3.5 shows the performance of the DeepMatcher and RoBERTa models on every dataset. We have also repeated the performance of Magellan and BERT-Mini for easy comparison. As we see, DeepMatcher generally outperforms Magellan on the dirty and textual datasets while the results are more mixed on the structured datasets — which is in line with the DeepMatcher authors' reported results [91]. Furthermore, BERT-Mini performs better than DeepMatcher on most datasets while RoBERTa performs even better than BERT-Mini. This shows that even though bigger transformer models are better, a conservatively sized model is able to outperform the previous generation deep learning method.

Figure 3.6 shows the counterfactual F_1 score for LIME, SHAP, Landmark, and LEMON for DeepMatcher and RoBERTa across all datasets. We see that the results are similar to those of Magellan and BERT-Mini in Section 3.6.1, and the biggest improvements over the baselines are seen for non-matches. The results further strengthen the claim that LEMON is model-agnostic by showing that it is equally functional for other deep learning architectures and even bigger transformer models.

Furthermore, Figure 3.7 shows the perturbation error PE for the same explainability methods and matchers across all datasets. The general tendencies are the same as for Magellan and BERT-Mini in Section 3.6.2. LEMON is overall similar to LIME (but performs noticeably worse on some datasets for non-matches with DeepMatcher), while still being significantly more faithful than SHAP and Landmark. We note that even though Landmark has substantially higher perturbation error than LEMON and LIME for DeepMatcher and RoBERTa, it is still considerably better than for Magellan. We are uncertain why Magellan triggers particularly large errors, but we suspect it is because Magellan has a less forgiving decision boundary that changes more abruptly when multiple attributes are perturbed at the same time since it uses per-attribute string similarity metrics.

Table 3.5: F_1 Score for the Magellan, DeepMatcher, BERT-Mini, and RoBERTa Matchers Used in the Experiments on the Public DeepMatcher [91] Benchmark Dataset.

Type	**Name**	**Matcher** F_1			
		MG	DM	BM	RoBERTa
Structured	Amazon-Google	0.52	0.67	0.67	0.72
	Beer	0.85	0.69	0.76	0.90
	DBLP-ACM	0.99	0.98	0.98	0.99
	DBLP-Scholar	0.94	0.95	0.93	0.95
	Fodors-Zagats	1.00	0.91	0.95	1.00
	iTunes-Amazon	0.90	0.87	0.93	0.94
	Walmart-Amazon	0.66	0.66	0.80	0.87
Dirty	DBLP-ACM	0.91	0.96	0.97	0.99
	DBLP-Scholar	0.83	0.92	0.94	0.95
	iTunes-Amazon	0.53	0.65	0.90	0.96
	Walmart-Amazon	0.41	0.39	0.79	0.86
Textual	Abt-Buy	0.51	0.68	0.81	0.88
	Company	0.57	0.89	0.91	0.91

3.9 Extensive Results

3.9.1 Precision-Recall Trade-off

Table 3.2 from Section 3.6.1 reports counterfactual F_1 scores. For completeness we also present the counterfactual precision and recall for those same experiments in Table 3.8. The desired trade-off between precision and recall will depend on the use case, so we acknowledge that F_1 score will never be a perfect metric. One could argue that counterfactual precision is often more important than counterfactual recall because it is harder to trust explanations that convey false information than explanations that fail to convey anything useful. However, it would still be challenging to define exactly what the trade-off should be. Regardless, we see that all evaluated methods have relatively high precision and in general higher precision than recall.

As expected, we see that the main reason the baselines perform badly on non-matches is that the counterfactual recall is low. In other words, they simply struggle to generate explanations that could be interpreted counterfactually.

Table 3.6: Counterfactual F_1 Score for All the Evaluated Explainability Methods Across All Datasets and the Two Matchers.

Model	Type	Method	Dataset													
			Structured							Dirty				Textual		
			AG	B	DA	DG	FZ	IA	WA	DA	DG	IA	WA	AB	C	Mean
DeepMatcher	Match	LIME	0.98	0.76	0.86	0.68	0.74	0.54	0.84	0.77	0.85	0.59	0.91	0.93	0.22	0.74
		SHAP	0.96	0.76	**1.00**	0.81	0.98	**0.60**	0.86	0.99	0.97	**0.62**	0.92	0.96	**0.35**	0.83
		Landmark	0.98	0.76	0.95	0.90	0.96	0.57	0.92	0.96	0.95	0.52	0.85	0.90	0.12	0.79
		LEMON	**1.00**	**0.80**	**1.00**	**1.00**	**1.00**	0.57	**0.98**	**1.00**	**1.00**	0.54	**1.00**	**0.99**	0.21	**0.85**
	Non-match	LIME	0.18	0.26	0.02	0.03	0.00	0.22	0.10	0.02	0.07	0.11	0.14	0.54	0.03	0.13
		SHAP	0.22	0.30	0.00	0.01	0.01	0.60	0.17	0.00	0.02	0.35	0.13	0.51	0.08	0.18
		Landmark	0.25	0.74	**0.17**	0.14	0.64	0.37	0.15	0.10	0.21	0.54	0.57	**0.98**	0.04	0.38
		LEMON	**0.62**	**0.97**	0.15	**0.76**	**0.89**	**0.69**	**0.89**	**0.73**	**0.76**	**0.99**	**0.65**	**0.98**	**0.90**	**0.77**
RoBERTa	Match	LIME	0.99	0.80	**1.00**	0.89	0.67	0.87	0.77	0.99	0.77	0.84	0.71	0.79	0.19	0.79
		SHAP	0.99	0.83	**1.00**	0.99	0.73	**1.00**	0.63	**1.00**	0.98	0.87	0.68	0.81	**0.27**	0.83
		Landmark	0.99	**1.00**	**1.00**	0.88	0.82	0.92	0.82	0.99	0.80	0.86	0.82	0.94	0.06	0.84
		LEMON	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.95**	**1.00**	**1.00**	**0.96**	**0.96**	**0.99**	**0.27**	**0.93**
	Non-match	LIME	0.10	0.15	0.00	0.05	0.00	0.05	0.04	0.00	0.02	0.09	0.10	0.04	0.04	0.05
		SHAP	0.04	0.16	0.00	0.02	0.00	0.00	0.12	0.00	0.00	0.05	0.16	0.04	0.06	0.05
		Landmark	0.23	0.18	0.09	0.13	0.01	0.11	0.35	0.05	0.11	0.13	0.54	0.58	0.01	0.19
		LEMON	**0.73**	**0.58**	**0.18**	**0.71**	**0.83**	**0.87**	**0.79**	**0.53**	**0.77**	**0.75**	**0.92**	**0.97**	**0.94**	**0.74**

3.9.2 Magnitude of Changes in User Study

Table 3.9 reports the average edit distance for the record pair, before and after being altered by the users in the user study (see Section 3.6.3), after seeing an explanation from LIME or LEMON for all datasets. We observe that users tend to make bigger changes with LEMON, perhaps indicating that the users have a tendency to underestimate the changes necessary to sway the matcher when the explanations are less helpful and they need to rely more on their own intuition. Matches in the Company dataset are a good example. They require a surprising amount of perturbation to convince the matchers something is not a match because the record pairs contain so many redundant highly discriminative features.

3.9.3 Neighborhood Sample Size

Due to the space constraints, Figure 3.8 and 3.9 from Section 3.6.6 only report results from the Abt-Buy dataset. Figure 3.13 and 3.14 show the results for all datasets.

The key takeaway from Section 3.6.6 about neighborhood sampling size $\mathcal{Z}_x$ and performance is true for all datasets: it takes a relatively low number of samples to reach stationary levels of performance, and the F_1 score and perturbation error do not change much with more samples after that. We can observe that, unsurprisingly, datasets with larger records tend to need more samples to

Table 3.7: Perturbation Error *PE* for All the Evaluated Explainability Methods Across All Datasets and the Two Matchers.

Model	Type	Method	Dataset													
			Structured							Dirty				Textual		
			AG	B	DA	DG	FZ	IA	WA	DA	DG	IA	WA	AB	C	Mean
DeepMatcher	Match	LIME	**0.36**	**0.24**	**0.34**	0.46	0.29	**0.17**	0.55	0.55	0.46	**0.23**	0.42	**0.30**	1.28	**0.43**
		SHAP	0.70	0.34	0.82	0.96	0.41	0.40	0.94	0.93	1.09	0.34	0.70	0.46	1.35	0.73
		Landmark	0.82	1.38	0.65	0.91	0.38	0.68	0.84	1.01	1.13	2.95	1.75	1.17	1.11	1.14
		LEMON	0.38	0.69	0.35	**0.44**	**0.23**	0.28	**0.37**	**0.46**	**0.43**	0.57	**0.38**	0.41	**0.62**	**0.43**
	Non-match	LIME	**0.48**	**0.36**	**0.32**	**0.41**	**0.22**	**0.22**	**0.52**	**0.44**	0.50	**0.24**	**0.49**	**0.39**	0.70	**0.41**
		SHAP	0.75	0.65	0.83	0.73	0.44	0.49	0.79	0.92	0.89	0.57	0.83	0.75	0.92	0.74
		Landmark	0.80	1.02	0.85	0.82	0.44	0.63	0.82	0.83	0.80	1.42	0.90	2.21	1.28	0.99
		LEMON	0.56	0.91	0.69	0.55	0.30	0.51	0.53	0.49	**0.48**	0.78	0.58	0.47	**0.45**	0.56
RoBERTa	Match	LIME	**0.34**	**0.33**	**0.39**	0.51	0.60	**0.41**	0.67	**0.36**	0.54	0.54	0.72	0.93	1.34	0.59
		SHAP	0.88	1.02	1.27	1.15	0.91	1.05	1.32	1.12	1.07	1.27	1.37	0.94	1.62	1.15
		Landmark	0.82	0.73	0.79	0.88	0.84	0.78	0.86	0.80	0.89	0.93	0.90	0.68	1.33	0.86
		LEMON	0.36	0.43	**0.39**	**0.47**	**0.56**	0.44	**0.53**	**0.36**	**0.50**	**0.53**	**0.52**	**0.50**	**0.54**	**0.47**
	Non-match	LIME	0.55	**0.64**	**0.40**	**0.47**	0.57	**0.39**	0.91	0.92	0.61	0.74	0.79	0.79	0.49	0.64
		SHAP	0.88	1.05	1.09	1.31	**0.49**	0.89	0.96	**0.63**	0.89	1.71	0.96	0.82	1.03	0.98
		Landmark	0.81	0.85	0.81	0.80	0.83	0.79	0.78	0.81	0.79	0.77	0.78	0.81	1.00	0.82
		LEMON	**0.53**	0.69	1.05	0.56	0.74	0.61	**0.56**	0.74	**0.58**	**0.57**	**0.44**	**0.31**	**0.42**	**0.60**

reach this state.

Overall, the performance increases with more samples up to a certain point and is significantly hampered by a very low number of samples. This is not only because the low number of samples leads to erroneous modeling of the effect of perturbations, but also because there might not have been any interesting perturbations sampled. However, we note that in some instances the counterfactual F_1 score is higher for a lower number of samples. For example Landmark on the Company dataset. Upon inspection, we see this is because the low number of samples makes the surrogate model overfit and make overly confident claims. This turns out to be correct more often in a strictly counterfactual sense and pay off in terms of counterfactual F_1 score compared to a more faithful approach that fails to provide a counterfactually interpretable explanation. Unfortunately, this comes at the cost of unacceptably large perturbation errors and low faithfulness and does therefore not represent a viable option in practice.

In regards to stability, we see from Figure 3.14 that the behavior is similar on all datasets. The main difference is that datasets with bigger records tend to need more samples to reach similar levels of stability.

3.9.4 Explanation Complexity

Figure 3.10 from Section 3.6.7 shows the effect of varying K for the Abt-Buy dataset. For completeness, Figure 3.15 shows the effect of varying K for all

Table 3.8: Counterfactual Precision and Recall for the Reported F_1 Scores in Table 3.2.

Model	Type	Method	Dataset																													
			Structured														Dirty								Textual							
			AG		B		DA		DG		FZ		IA		WA		DA		DG		IA		WA		AB		C		Mean	
			P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
Magellan	Match	LIME	0.96	0.96	0.88	0.79	1.00	1.00	0.96	0.80	0.77	0.77	1.00	0.97	1.00	0.70	0.72	0.38	0.88	0.72	1.00	0.82	0.94	0.80	0.96	0.95	0.67	0.36	0.90	0.77
		SHAP	0.95	0.95	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.95	1.00	1.00	0.99	0.71	0.65	0.65	0.96	0.96	0.68	0.68	0.89	0.75	0.98	0.98	0.79	0.60	0.91	0.86
		SHAP (w/ CFG)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.98	0.99	0.99	1.00	1.00	0.91	0.91	0.99	0.99	0.99	0.99	0.98	0.78	0.99	0.97
		Landmark	0.92	0.92	0.89	0.89	1.00	1.00	0.96	0.96	0.73	0.73	0.87	0.87	0.99	0.91	1.00	0.60	0.80	0.68	0.90	0.86	0.90	0.89	0.95	0.95	0.29	0.27	0.86	0.81
		LEMON (w/o DE)	0.99	0.97	1.00	0.84	1.00	1.00	1.00	0.95	0.95	0.95	1.00	1.00	1.00	0.93	0.99	0.71	0.99	0.88	1.00	0.86	0.97	0.93	0.98	0.98	0.89	0.34	0.98	0.87
		LEMON (w/o AP)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	0.99	0.99	0.98	0.98	0.98	0.73	1.00	0.98
		LEMON (w/o CFG)	0.96	0.96	0.94	0.84	1.00	1.00	0.97	0.80	0.91	0.91	1.00	0.97	1.00	0.68	0.79	0.38	0.91	0.72	0.94	0.77	0.93	0.75	0.93	0.92	0.78	0.28	0.93	0.77
		LEMON	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	0.99	0.96	0.99	0.99	0.99	0.67	1.00	0.97
	Non-match	LIME	0.80	0.01	1.00	0.06	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.08	0.47	0.02	1.00	0.05	0.84	0.05	0.56	0.10	0.42	0.05	0.70	0.07	0.35	0.06	0.78	0.04
		SHAP	0.50	0.00	1.00	0.01	1.00	0.00	1.00	0.01	0.00	0.00	1.00	0.03	1.00	0.01	1.00	0.04	0.94	0.03	0.38	0.06	0.28	0.07	0.94	0.03	0.13	0.13	0.71	0.03
		SHAP (w/ CFG)	1.00	0.01	1.00	0.12	1.00	0.00	1.00	0.01	0.00	0.00	1.00	0.03	1.00	0.01	1.00	0.05	1.00	0.05	0.67	0.14	0.79	0.12	1.00	0.04	1.00	1.00	0.88	0.12
		Landmark	0.16	0.13	0.85	0.83	0.51	0.08	0.45	0.13	0.36	0.17	0.22	0.19	0.96	0.90	0.04	0.04	0.44	0.41	0.43	0.36	0.04	0.03	0.81	0.77	0.09	0.09	0.41	0.32
		LEMON (w/o DE)	0.83	0.46	0.86	0.71	0.80	0.03	0.65	0.26	0.85	0.53	0.75	0.55	0.82	0.82	0.86	0.51	0.82	0.60	0.88	0.86	0.84	0.80	0.91	0.86	0.96	0.96	0.83	0.61
		LEMON (w/o AP)	1.00	0.02	1.00	0.26	1.00	0.01	1.00	0.02	1.00	0.01	0.86	0.08	0.59	0.03	0.98	0.10	0.98	0.12	0.74	0.16	0.78	0.24	0.86	0.10	0.97	0.52	0.90	0.13
		LEMON (w/o CFG)	0.74	0.27	0.79	0.32	0.80	0.04	0.60	0.07	0.50	0.02	0.79	0.14	0.74	0.71	0.70	0.14	0.76	0.36	0.75	0.63	0.71	0.57	0.82	0.74	0.13	0.12	0.68	0.32
		LEMON	0.74	0.68	0.50	0.50	0.70	0.07	0.64	0.47	0.98	0.98	0.77	0.77	0.76	0.76	0.79	0.72	0.78	0.77	0.87	0.87	0.87	0.87	0.87	0.87	0.96	0.96	0.79	0.71
BERT-Mini	Match	LIME	0.97	0.93	1.00	0.48	0.97	0.97	0.96	0.76	0.95	0.91	1.00	0.48	0.98	0.70	0.97	0.97	0.94	0.55	1.00	0.45	0.98	0.68	0.97	0.65	0.96	0.10	0.97	0.66
		SHAP	0.90	0.89	0.89	0.74	0.79	0.79	0.65	0.65	0.91	0.91	0.72	0.67	0.78	0.65	0.79	0.79	0.62	0.62	0.66	0.61	0.83	0.68	0.83	0.71	0.91	0.14	0.79	0.68
		SHAP (w/ CFG)	0.95	0.95	0.96	0.96	0.92	0.92	0.98	0.98	0.86	0.86	0.89	0.89	0.97	0.97	0.98	0.98	0.99	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00	0.25	0.96	0.90
		IG	0.90	0.90	0.43	0.43	0.73	0.70	0.83	0.83	0.50	0.50	0.70	0.70	0.68	0.66	0.70	0.68	0.89	0.89	0.81	0.81	0.69	0.67	0.79	0.79	0.33	0.33	0.69	0.68
		IG (w/ CFG)	0.88	0.88	0.52	0.52	0.68	0.65	0.94	0.94	0.68	0.68	0.70	0.70	0.78	0.75	0.87	0.85	0.95	0.95	0.94	0.94	0.88	0.86	0.93	0.93	0.28	0.28	0.77	0.76
		Landmark	0.98	0.98	1.00	0.87	1.00	1.00	0.94	0.94	0.86	0.86	0.96	0.93	0.89	0.80	0.99	0.99	0.83	0.83	0.90	0.90	0.91	0.85	0.92	0.76	0.09	0.08	0.87	0.83
		LEMON (w/o DE)	1.00	0.97	1.00	0.52	1.00	1.00	1.00	0.93	1.00	0.95	1.00	0.78	0.99	0.72	1.00	1.00	1.00	0.89	1.00	0.81	0.99	0.73	1.00	0.76	1.00	0.14	1.00	0.78
		LEMON (w/o AP)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.89	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95	1.00	0.25	1.00	0.93
		LEMON (w/o CFG)	0.97	0.92	1.00	0.48	0.98	0.98	0.99	0.76	1.00	0.95	1.00	0.41	0.99	0.68	0.99	0.98	0.97	0.55	1.00	0.42	0.99	0.68	0.99	0.65	1.00	0.08	0.99	0.66
		LEMON	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.89	0.99	0.97	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	0.95	1.00	0.22	1.00	0.92
	Non-match	LIME	0.97	0.07	1.00	0.03	1.00	0.00	1.00	0.02	1.00	0.02	0.86	0.07	0.91	0.04	1.00	0.01	1.00	0.02	1.00	0.13	0.90	0.04	0.81	0.03	0.64	0.01	0.93	0.04
		SHAP	0.90	0.07	0.86	0.09	1.00	0.01	1.00	0.02	1.00	0.01	1.00	0.17	0.78	0.04	1.00	0.01	1.00	0.02	0.84	0.21	0.70	0.08	0.59	0.08	0.76	0.10	0.88	0.07
		SHAP (w/ CFG)	0.95	0.08	1.00	0.15	1.00	0.01	1.00	0.02	1.00	0.01	1.00	0.21	1.00	0.06	1.00	0.01	1.00	0.03	1.00	0.32	0.96	0.09	0.92	0.23	1.00	0.15	0.99	0.10
		IG	0.40	0.04	0.00	0.00	0.33	0.00	0.32	0.02	0.33	0.01	0.00	0.00	0.13	0.01	0.18	0.01	0.10	0.01	0.00	0.00	0.25	0.02	0.42	0.04	0.07	0.01	0.19	0.01
		IG (w/ CFG)	0.67	0.04	1.00	0.01	0.33	0.00	0.48	0.02	0.33	0.01	0.00	0.00	0.44	0.02	0.29	0.01	0.21	0.01	0.17	0.01	0.70	0.03	0.65	0.04	0.62	0.02	0.45	0.02
		Landmark	0.42	0.39	0.71	0.69	0.06	0.05	0.18	0.17	0.50	0.50	0.63	0.63	0.65	0.64	0.07	0.06	0.36	0.35	0.51	0.49	0.74	0.74	0.68	0.66	0.01	0.01	0.42	0.41
		LEMON (w/o DE)	0.99	0.61	0.95	0.91	0.87	0.41	0.93	0.54	0.92	0.81	0.94	0.89	0.98	0.88	0.96	0.61	0.95	0.67	0.91	0.86	0.99	0.96	0.95	0.91	0.97	0.97	0.95	0.77
		LEMON (w/o AP)	1.00	0.10	1.00	0.04	1.00	0.02	1.00	0.04	1.00	0.02	1.00	0.50	1.00	0.07	1.00	0.02	1.00	0.05	1.00	0.38	1.00	0.09	1.00	0.13	1.00	0.15	1.00	0.12
		LEMON (w/o CFG)	1.00	0.33	1.00	0.85	1.00	0.01	0.96	0.11	0.93	0.79	1.00	0.89	0.97	0.82	1.00	0.02	0.98	0.10	0.88	0.67	0.98	0.92	0.96	0.84	0.96	0.96	0.97	0.56
		LEMON	0.82	0.81	0.94	0.94	0.80	0.55	0.69	0.67	0.86	0.86	0.98	0.96	0.90	0.90	0.52	0.49	0.79	0.79	0.87	0.87	0.95	0.95	0.98	0.98	0.97	0.97	0.85	0.83

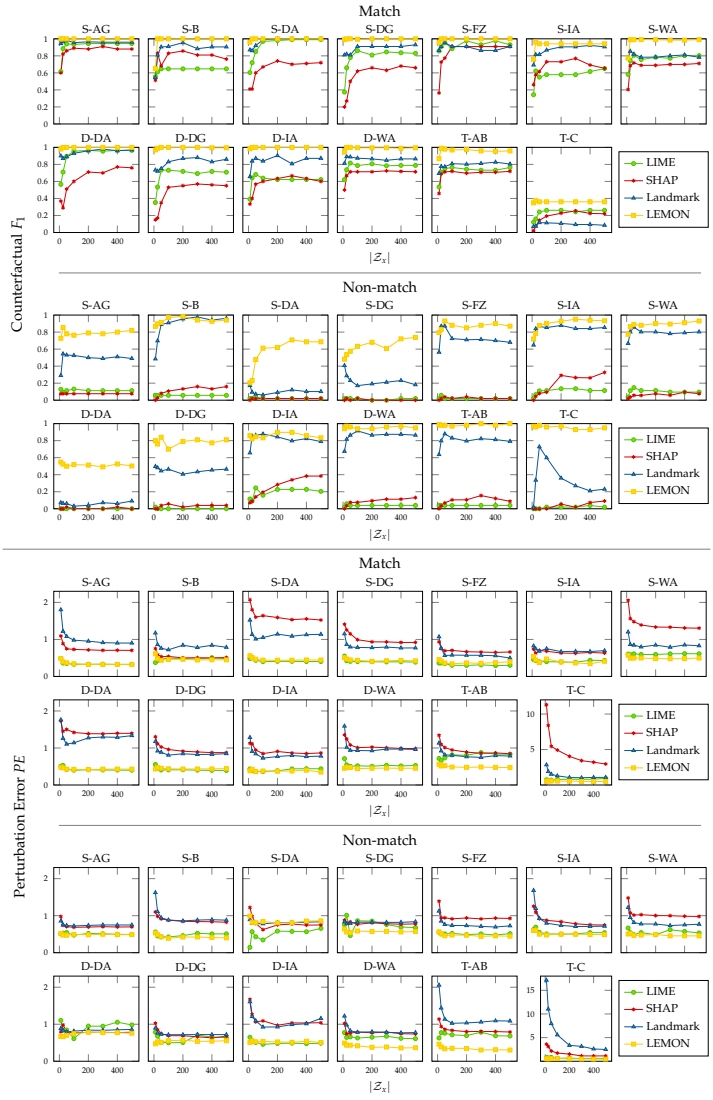


Figure 3.13: Counterfactual F_1 and perturbation error (PE) of explainability methods based on neighborhood sampling for BERT-Mini on all datasets when varying the neighborhood sampling size $|\mathcal{Z}_x|$.

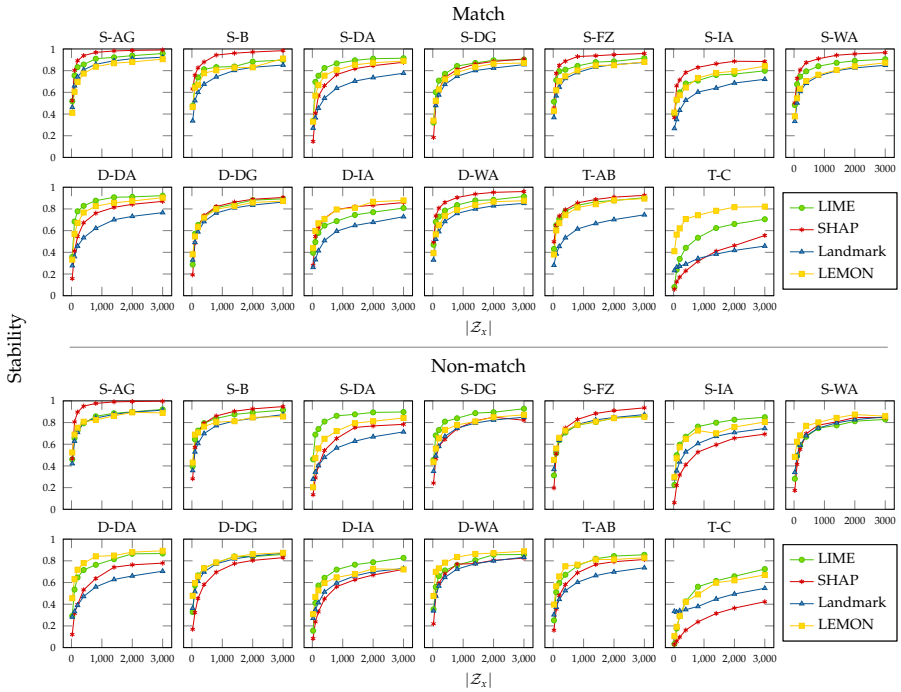


Figure 3.14: Stability of explainability methods based on neighborhood sampling for BERT-Mini on all datasets when varying the neighborhood sampling size $|\mathcal{Z}_x|$.

Table 3.9: Average Edit Distance of the Changes Made by the Participants in the User Study.

Dataset	Method			
	LIME		LEMON	
	Match	Non-match	Match	Non-match
Structured				
Amazon-Google	14	15	15	22
Beer	14	20	24	23
DBLP-ACM	24	35	43	64
DBLP-GoogleScholar	20	21	33	39
Fodors-Zagats	14	17	17	16
iTunes-Amazon	17	28	23	35
Walmart-Amazon	12	19	19	17
Dirty				
DBLP-ACM	25	25	49	82
DBLP-GoogleScholar	27	23	47	53
iTunes-Amazon	33	36	28	60
Walmart-Amazon	17	20	26	23
Textual				
Abt-Buy	14	35	36	36
Company	107	102	556	148

datasets. Experiments were performed as explained in Section 3.6.7.

Results for all datasets verify the claim that, for all but the lowest of Ks, the counterfactual interpretation and explanation faithfulness is not meaningfully affected by the choice of K. This is convenient because it lets us prioritize choosing a K that is suitable for user consumption.

3.9.5 Runtime

Figure 3.11 from Section 3.6.8 shows the runtime for three selected datasets. We report the equivalent results for all datasets in Figure 3.16. Furthermore, Figure 3.17 extends the results on stability-runtime trade-off in Figure 3.12 from Section 3.6.8 to all datasets.

In general, we observe that LEMON has higher runtime than the baselines across all datasets. The runtime is first and foremost determined by the neighborhood sampling size $\mathcal{Z}_x$. However, as discussed in Section 3.6.5 and 3.6.6, even small sample sizes yield satisfactory counterfactual interpretation and explanation faithfulness, and deciding $\mathcal{Z}_x$ in practice is mostly a matter of stability. Therefore, if low runtime is important, one has the option to trade off some stability to decrease the runtime. Figure 3.17 then tells a different story than

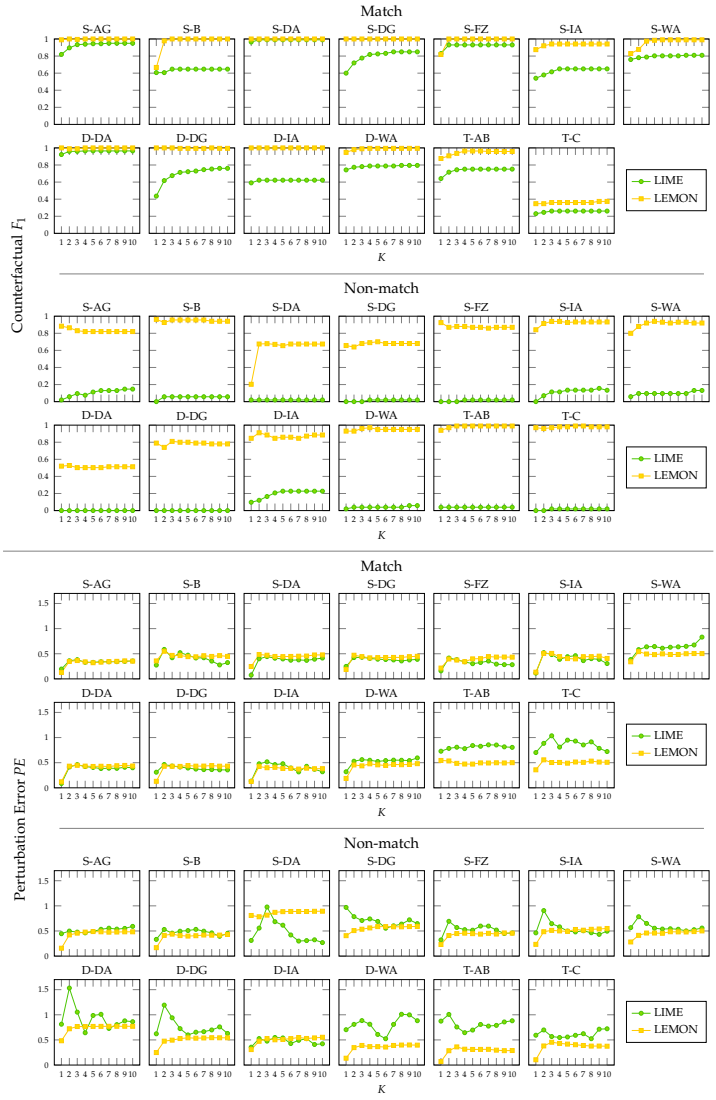


Figure 3.15: Counterfactual F_1 and perturbation error (PE) of explainability methods based on neighborhood sampling for BERT-Mini on all datasets when varying K.

Figure 3.16 because it shows that the trade-off between runtime and stability is less than the relative difference in runtime as seen in Figure 3.16 for most datasets. In other words, one can decrease the neighborhood sampling size of LEMON to get a more similar runtime as for example Landmark while still being equally stable and retaining the high level of counterfactual interpretation and explanation faithfulness. To what degree this trade-off is beneficial depends on the dataset.

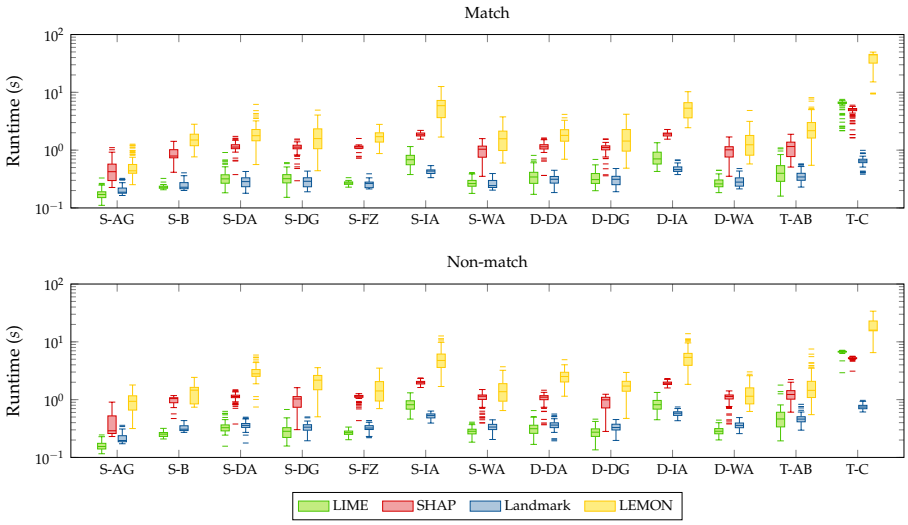


Figure 3.16: Runtime of explainability methods based on neighborhood sampling for BERT-Mini on all datasets.

3.10 Acknowledgement

This work is supported by Cognite and the Research Council of Norway under Project 298998. We thank the reviewers, Hassan Abedi Firouzjaei, Yanzhe Bekkemoen, Jon Atle Gulla, Dhruv Gupta, Benjamin Kille, Ludvig Killingberg, Kjetil Nørvåg, Mateja Stojanović, and Bjørnar Vassøy for valuable feedback.

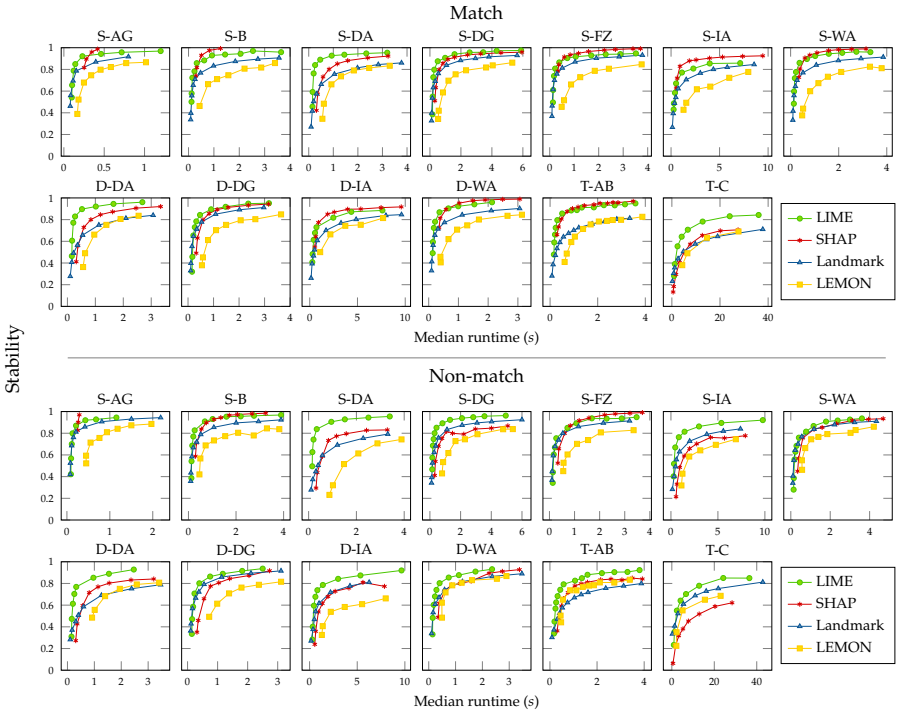


Figure 3.17: Stability of explainability methods based on neighborhood sampling for BERT-Mini on all datasets when varying the median runtime (by changing $|\mathcal{Z}_x|$).

Chapter 4

Strong Blocking Baseline for Deep Learning

Paper C

Original title:	ShallowBlocker: Improving Set Similarity Joins for Blocking
Authors:	Nils Barlaug
Status:	Being prepared for submission

Blocking is a crucial step in large-scale entity matching but often requires significant manual engineering from an expert for each new dataset. Recent work has show that deep learning is state-of-the-art and has great potential for achieving hands-off and accurate blocking compared to classical methods. However, in practice, such deep learning methods are often unstable, offers little interpretability, and require hyperparameter tuning and significant computational resources.

In this paper, we propose a hands-off blocking method based on classical string similarity measures: ShallowBlocker. It uses a novel hybrid set similarity join combining absolute similarity, relative similarity, and local cardinality conditions with a new effective pre-candidate filter replacing size filter. We show that the method achieves state-of-the-art pair effectiveness on both unsupervised and supervised blocking in a scalable way.

4.1 Introduction

Entity Matching (EM) is the task of identifying which records refer to the same real-world entity. It is a core data integration task, and is done either within one data source (i.e., deduplication) or across data sources [25, 34]. One of the task's key characteristics is its quadratic nature. The number of potential matches is quadratic in the number of records but the actual number of matches is typically linear in the number of records. This has two important consequences: 1) It is often computationally infeasible to explicitly compare all potential pairs. 2) Most record pairs do not match because the ratio of non-matches to matches increases linearly with the number of records. Therefore, the problem is often solved in two steps — blocking and then matching, where the blocking step generates a set of candidate record pairs and the matching step inspect the candidates to classify them as either match or non-match. The idea is that the blocking procedure will return a sub-quadratic number of pairs in sub-quadratic time (ideally linear for both) while still recalling most matches, paving the way for the matching procedure to achieve high precision using pairwise comparison in feasible time.

Blocking is a well-studied problem and there exists a multitude of techniques and methods [100]. Unfortunately, it is often hard to solve the task in practice without expert knowledge. One needs to be able to pick a suitable method and potentially hand-tune difficult to understand parameters for the data at hand. Recently, there has been substantial work on the use of deep learning

for blocking [39, 127, 152]. One of the potential benefits of such approaches is having a method that works satisfactorily across most datasets without expert handcrafting. Of course, deep learning is no silver bullet, and often entails less interpretability, longer runtime, and tweaking training hyperparameters per dataset. Additionally, recent work highlights that deep learning methods does not necessarily outcompete traditional methods in terms of effectiveness [102], despite earlier reports [127].

Proposed Method In this paper, we propose ShallowBlocker, a novel blocking method based on set similarity joins. The method relies on a new prefix filtering-based similarity join routine, `TTRKJoin`, that we introduce. The join routine combines constraints on absolute similarity, relative similarity, and local cardinality in order to exploit benefits of different similarity join types. Furthermore, we leverage parallelization and propose a new pre-candidate filtering technique to get an efficient implementation. To support hyperparameter selection, we perform analysis queries on randomly sampled records and known matches so that we can quickly estimate recall, number of retrieved pairs, and runtime for different hyperparameter configurations. Finally, through carefully selected early cutoff on searches in `TTRKJoin` we achieve approximate joins with quality guarantees.

ShallowBlocker consists of a strategy for choosing `TTRKJoin` hyperparameters in both an unsupervised and supervised setting. In the unsupervised setting the method tries to equally balance the pruning power of the three join constraints given a user-specified pair budget and degree of approximation. While in the supervised setting the method optimizes the join hyperparameters according to an arbitrary user-specified objective function expressing the desired trade-off between recall, number of returned pairs, and runtime.

Contribution Our main contributions are:

- We propose a new hands-off blocking method, ShallowBlocker, and show that it achieves state-of-the-art pair effectiveness for both unsupervised and supervised blocking in a scalable way. Importantly, it does not require elaborate dataset-specific tuning. For unsupervised use the user specifies pair budget and an optional approximation degree, while for supervised use the user can specify an arbitrary trade-off between recall, number of returned pairs, and runtime.

- We introduce a new expressive hybrid set similarity join primitive, (τ, τ_r, k)-join, suitable for blocking.
- We propose `TTRKJoin`, an efficient (τ, τ_r, k)-join algorithm with a novel pre-candidate filtering technique and strong bounds. Furthermore, we describe a framework for quickly estimating the effect of different hyperparameters on `TTRKJoin` and for performing approximate joins. ShallowBlocker introduces effective strategies for determining good hyperparameters for `TTRKJoin`.
- We demonstrate that classical string similarity blocking methods still outperform deep learning-based blocking methods, such as DeepBlocker, on widely used benchmark datasets.

Outline We start by covering related work (Section 4.2) and problem statement (Section 4.3). Then we introduce important set similarity join theory and techniques we will build upon (Section 4.4). In order to describe our proposed method we first discuss the new hybrid join primitive (Section 4.5) followed by the proposed algorithm for it (Section 4.6), a framework for estimating its behavior (Section 4.7), and how to do and interpret approximate joins (Section 4.8) — before we describe how everything goes together to form ShallowBlocker (Section 4.9). Finally, we describe the experimental setup (Section 4.10), go through the experiments with results (Section 4.11), and conclude (Section 4.12).

4.2 Related work

Researchers have pursued many approaches for doing blocking, and it is outside the scope of this paper to provide an extensive overview. We cover only the most relevant and refer the reader to Papadakis et al. [100] for a detailed survey.

4.2.1 Set Similarity Joins

If we convert records to token sets (e.g., q-grams or words) we can cast blocking as a set similarity join problem, where the goal is to find all pairs with similarity above some threshold. SSJoin [22] and AllPairs [13] proposed the highly effective prefix filtering technique for generating candidate pairs in addition to size filtering. PPJoin [145, 144] extends AllPairs with an additional candidate-time filter: the positional filter. L2AP [2] specializes prefix filtering for Cosine similarity measure and introduce tighter bounds based on the Cauchy–Schwarz

inequality. There is a multitude of methods with more elaborate and aggressive filtering techniques, both prefix-based and other [100]. However, Mann et al. [87] show that with efficient implementations the overhead of the filtering techniques often outweighs the benefits, and that AllPairs and PPJoin is the best performing methods and state-of-the-art.

4.2.2 Deep Learning

The research community have increasingly focused on the use of deep learning for entity matching the last few years [11], and some also target the blocking step. DeepER [39] uses pre-trained word embeddings and LSTMs to embed records. It generates candidate pairs using random hyperplane Multi-Probe LSH, and record embedding pairs are compared elementwise and then fed into a SVM to classify as match or no match. The network and SVM is trained using labeled data. AutoBlock [152] improves over DeepER by using a novel attention mechanism and cross-polytope LSH. DeepBlocker [127] is a state-of-the-art deep learning-based blocker. The authors explore a wide range of configurations, including different training strategies and different network types such as large pre-trained transformer models. Despite being self-supervised, the method is more effective than AutoBlock.

4.3 Problem Statement

Let A and B be two sets of records, and let $M \subseteq A \times B$ be all record pairs across A and B that refer to the same entity. In other words, for all $(a, b) \in M$ the records a and b are references to the same entity. The goal of entity matching/resolution is to determine M. For blocking in particular, the goal is to find a superset P of M such that $|P| \ll |A \times B|$ in subquadratic time and memory. Informally, the purpose is to remove large amounts of obvious non-matches so that a high-precision pair comparison downstream does not have to process a quadratic number of pairs. If A and B is the same set we call the problem deduplication.

When we evaluate blocking methods we are interested in three main performance characteristics:

1. **Recall**: To what degree the method is able to find true matches. We measure this with the classical recall measure $R = \frac{|P| \cap |M|}{|M|}$.

2. **Pruning Power**: To what degree the method is able to discard potential pairs. The most direct measure of this is $|P|$, but that is hard to interpret and compare across datasets. Reduction rate $(1 - \frac{|P|}{|A \times B|})$ is a popular measure, but since $|M|$ is generally expected to be linear in $|A|$ and $|B|$ it will increase quickly towards 1 for larger datasets if the blocking method is effective, which makes comparison across dataset sizes difficult. Therefore, in this paper, we will report the empirical cardinality $\widetilde{k} = \frac{|P|}{\min(|A|,|B|)}$.

3. **Efficiency**: How fast can it be done and with what computational resources. We measure this mainly through runtime, but memory consumption and special hardware requirements (e.g., GPU) is also of interest.

In practice, the former and the two latter are in conflict, so blocking methods will typically let the user adjust the trade-off between them.

4.4 Set Similarity Joins Using Prefix Filtering

Our method builds heavily on prefix filtering-based set similarity joins. Therefore, we will now describe existing set similarity join building blocks we exploit in our method, while the next sections will outline the join routine in our method.

We find it useful to introduce these concepts through the perspective of the popular similarity join method PPJoin [143, 144], as it offers a familiar frame of reference for the reader and let us introduce our join routine as an evolution of PPJoin. Algorithm 2 shows PPJoin similar to how it is presented by the authors. It performs self-join with jaccard similarity on unweighted sets. We will use this as a running example of the different building blocks as we go through them. At the same time we will also describe the same ideas extended to nonself-joins, other set similarity measures, and weighted sets — resulting in a more generic version of PPJoin at the end of the section that prepares us for the next sections.

4.4.1 Set Similarity Join

A set similarity join finds all set pairs with some set similarity measure above a user-provided threshold τ[1]. For the purpose of this paper a set contains tokens from some record. More formally, given two collections of token sets $\mathcal{A}$ and $\mathcal{B}$ we want to find all pairs $(a, b) \in \mathcal{A} \times \mathcal{B}$ that have similarity above some threshold

[1]We will see in Section 4.5 that there are other types of similarity joins.

Algorithm 2: PPJoin($\mathcal{A}, \tau_j$)

Input: $\mathcal{A}$ is a collection of token sets. Each token set is already sorted by $\mathcal{O}$. τ_j is the Jaccard threshold.
Output: Token set pairs $\{(a, a') \mid a \cap a' \geq \tau_j\}$

```
1  Sort A by increasing size;
2  I ← [∅]_t ;                                          // Inverted token indices
3  P ← ∅ ;                          // Pairs with Jaccard similarity at least τ_j
4  foreach a ∈ A do
5      O ← empty map, default value 0;
6      for i ← 1 to |a| − ⌈τ_o · |a|⌉ + 1 do                          // Prefix filter
7          t ← a[i];
8          I'_t ← {(a', j) ∈ I_t | τ_j · |a| ≤ |a'|} ;                 // Size filter
9          foreach (a', j) ∈ I'_t do
10             o* ← O[a'] + 1 + min(|a| − i, |a'| − j);
11             if o* ≥ (τ_j / (τ_j + 1)) (|a| + |b|) then              // Pos. filter
12                 O[a'] ← O[a'] + 1;
13             else
14                 Remove O[a'];
15         I_t ← I_t ∪ {(a', i)};
16     P ← P ∪ Verify(a, A, O, τ_j) ;
17 return P;
```

τ, i.e., find $C = \{(a, b) \in A \times B \mid \text{sim}(a, b) \geq \tau\}$. If $\mathcal{A} = \mathcal{B}$ we call it a self-join and usually ignore self-referring pairs (a, a).

In this paper, we consider four of the most used set similarity measures: Jaccard, Cosine, Dice, and Overlap. Table 4.1 list their definitions. Note that Jaccard, Cosine, and Dice are normalized and produce a similarity score between 0 and 1, while Overlap may be arbitrary large.

These similarity measures are for unweighted sets, but our method will rely on weighted sets. There are several ways to generalize these measures to weighted sets. We use a straightforward approach where the overlap is generalized to $\sum_k \min(a_k, b_k)$, where a_k is the weight of the kth globally ranked token in a according to some global ordering $\mathcal{O}$ if a contains this token or zero otherwise[2]. The weighted counterpart to the four similarity measures are listed in Table 4.1. For convenience we define $x[i]$ to be the ith token in the weighted set x according to $\mathcal{O}$, $x[i..j]$ to be the subset containing the ith to the jth (in-

[2]This is different from [144], where the weight for a given token is assumed to be global. Here the weight can be different for the same token in different sets — which is important for weighting schemes like TF-IDF.

	Measure	$sim(a,b)$	Prefix size $\pi(x,\tau)$	Size bounds $\lambda_l(a,\tau)$	Size bounds $\lambda_u(a,\tau)$	Equivalent overlap $\alpha(a,b,\tau)$
Unweighted	Jaccard	$\frac{\lvert a \cap b\rvert}{\lvert a \cup b\rvert}$	$\lvert x\rvert - \lceil \tau \cdot \lvert x\rvert \rceil + 1$	$\tau \cdot \lvert a\rvert$	$\frac{\lvert a\rvert}{\tau}$	$\frac{\tau}{\tau+1}(\lvert a\rvert + \lvert b\rvert)$
	Cosine	$\frac{\lvert a \cap b\rvert}{\sqrt{\lvert a\rvert \cdot \lvert b\rvert}}$	$\lvert x\rvert - \lceil \tau^2 \cdot \lvert x\rvert \rceil + 1$	$\tau^2 \cdot \lvert a\rvert$	$\frac{\lvert a\rvert}{\tau^2}$	$\tau\sqrt{\lvert a\rvert \cdot \lvert b\rvert}$
	Dice	$\frac{2 \cdot \lvert a \cap b\rvert}{\lvert a\rvert + \lvert b\rvert}$	$\lvert x\rvert - \left\lceil \frac{\tau \cdot \lvert x\rvert}{2-\tau} \right\rceil + 1$	$\frac{\tau \cdot \lvert a\rvert}{2-\tau}$	$\frac{(2-\tau) \cdot \lvert a\rvert}{\tau}$	$\frac{\tau(\lvert a\rvert + \lvert b\rvert)}{2}$
	Overlap	$\lvert a \cap b\rvert$	$\lvert x\rvert - \tau + 1$	τ	∞	τ
Weighted	Jaccard	$\frac{\sum_k \min(a_k, b_k)}{\sum_k \max(a_k, b_k)}$	$(1-\tau) \cdot \omega(x)$	$\tau \cdot \omega(a)$	$\frac{\omega(a)}{\tau}$	$\frac{\tau}{\tau+1}\big(\omega(a) + \omega(b)\big)$
	Cosine*	$\sum_k a_k \cdot b_k$	$\omega(x) - \tau$	τ	∞	τ
	Dice	$\frac{2 \cdot \sum_k \min(a_k, b_k)}{\lvert a\rvert + \lvert b\rvert}$	$(1 - \frac{\tau}{2-\tau}) \cdot \omega(x)$	$\frac{\tau}{2-\tau} \cdot \omega(a)$	$\frac{(2-\tau)}{\tau} \cdot \omega(a)$	$\frac{\tau}{2}\big(\omega(a) + \omega(b)\big)$
	Overlap	$\sum_k \min(a_k, b_k)$	$\omega(x) - \tau$	τ	∞	τ

Table 4.1: Definitions of the set similarity measures together with prefix size, size bounds, and equivalent overlap for both unweighted and weighted sets. *Requires every set x to be normalized such that $\|x\|_2 = 1$.

clusive) tokens of x, $t.w$ the weight of a token t, $t.k$ the global rank of a token t according to the total ordering $\mathcal{O}$, and the size $\omega(x) = \sum_{i=1}^{|x|} x[i].w$.

4.4.2 Inverted Token Index

Central to most set similarity joins (and PPJoin) are the use of inverted token indexes — i.e., indexes of which token set contains a certain token. A naive approach for finding all the token sets in $\mathcal{B}$ that are similar to some $a \in \mathcal{A}$ would be to look up all tokens of a to find all token sets in $\mathcal{B}$ that have at least one common token and check if $sim(a,b) \geq \tau$ for all of them. Finding all token set pairs above the threshold is then just a matter of repeating the procedure for all $a \in \mathcal{A}$[3].

This is the underlying idea of PPJoin, as seen in Algorithm 2. Note that since it performs self-join it employs an optimization where the sets are indexed

[3]In this case, sets in $\mathcal{A}$ are used to query against an index of $\mathcal{B}$. Obviously, one could also do it the other way around — and depending on the data it might be beneficial. However, for simplicity, but without loss of generality, we assume $\mathcal{A}$ to be the query collection and $\mathcal{B}$ the index collection throughout the paper.

only after having been queried (see line 15). The problem with this approach is tokens that occur in a large number of token sets lead to an excessive number of retrieved token sets, and therefore token set comparisons, for pairs that are not very similar. There are simply to many token set pairs that have at least one common token. It is possible to improve on this by efficiently merging the inverted lists and avoid redundant work (e.g., ScanCount [76]), but we still suffer from the same underlying problem. Therefore, to achieve feasible runtimes we need to reduce the total number of lookups and token set comparisons through multiple pruning techniques — also called filters.

4.4.3 Prefix Filtering

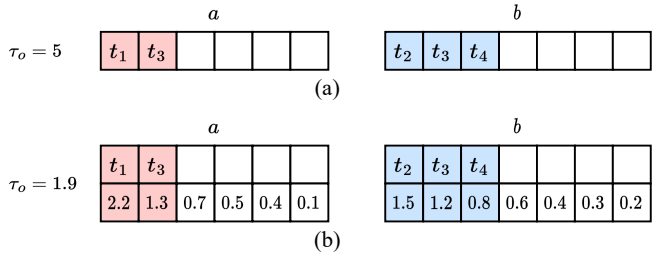


Figure 4.1: Example of the prefix filtering principle for (a) unweighted and (b) weighted token sets. The token sets are sorted by the total ordering $\mathcal{O}$. The highlighted tokens is the prefixes that must have at least one token common if the Overlap is at least τ_o.

The key insight that enables prefix filtering is that once one have scanned through π inverted lists without seeing some token set b we know the intersection can be at most $|a| - \pi$ because we know π of the tokens in a does not exist in b. In other words, we only need to probe $|a| - \tau_o + 1$ inverted lists to be certain we have found every $b \in \mathcal{B}$ with an overlap $|a \cap b|$ of at least τ_o, and can safely prune the remaining inverted token lists. We can provide an even stronger guarantee if we assume all token sets to be sorted in the same global total ordering $\mathcal{O}$. This is known as the *prefix filtering principle* [22] (adapted for this paper):

Lemma 1 (Unweighted Prefix Filtering Principle) *Let a and b be two token sets and assume some total token ordering $\mathcal{O}$. If $|a \cap b| \geq \tau_o$, then $\Big|a\big[1..(|a|-$*

$\tau_o + 1)\big] \cap b\big[1..(|b| - \tau_o + 1)\big]\Big| > 0.$

In other words, when denoting $x[1..\pi]$ as the π-prefix of x, if the overlap between a and b are at least τ_o, then the $(|a|-\tau_o+1)$-prefix of a and the $(|b|-\tau_o+1)$-prefix of b will have at least one token in common. See Figure 4.1a for an example. This not only means it is enough to probe the $(|a| - \tau_o + 1)$-prefix tokens of a to find the token sets with overlap at least τ_o in $\mathcal{B}$, but also that we only need to index the $(|b| - \tau_o + 1)$-prefix for all $b \in \mathcal{B}$. This further reduces the indexing and query time as well as the memory footprint of the inverted index. In the rest of the paper we will refer to such prefixes sufficient to find similar token sets of x above some threshold as simply the prefix of x.

Token Ordering The real power of the prefix filtering principle lies in our ability to freely choose the token ordering $\mathcal{O}$. We order by increasing frequency of occurrence among token sets — essentially ignoring the most frequent tokens. If τ_o is large enough and the distribution of tokens skewed enough we effectively avoid the problem of frequently occurring tokens.

Other Similarity Measures Utilizing this is straightforward when we have an Overlap similarity threshold. The trick to extending to other similarity measures is that we can bound the overlap $|a \cap b|$ from below using $|a|$ and the similarity measure at hand. Assume, for example, a Jaccard threshold τ_j. It can be shown easily that

$$\begin{gathered} \mathrm{sim}_j(a,b) = \frac{|a \cap b|}{|a \cup b|} \geq \tau_j \\ \Downarrow \\ |a \cap b| \geq \tau_j \cdot |a| \end{gathered}$$

Which means we can simply replace τ_o with $\lceil \tau_j \cdot |a| \rceil$ and only query the $(|a| + \lceil \tau \cdot |a| \rceil + 1)$-prefix for $a \in \mathcal{A}$ and only index the $(|b| + \lceil \tau \cdot |b| \rceil + 1)$-prefix of $b \in \mathcal{B}$. See line 6 in Algorithm 2. One can bound the overlap for the other similarity measures in a similar fashion. Table 4.1 lists $\pi(x, \tau)$, the prefix size for token set x with respect to some similarity threshold τ, for all four similarity measures.

Weighted Sets Prefix filtering can easily be extended to weighted sets where the overlap between a and b is defined as $\sum_k \min(a_k, b_k)$. We generalize the prefix filtering principle into a weighted version.

Lemma 2 (Weighted Prefix Filtering Principle) *Let a and b be two weighted token sets and assume some total token ordering $\mathcal{O}$. Furthermore, let $\phi(x,p) = \operatorname{argmin}_i \Big(\omega\big(x[1..i]\big) > p\Big)$. If $\sum_k \min(a_k, b_k) \geq \tau_o$ then $\sum_k \min\Big(a\big[1..\phi(a,\omega(a)-\tau_o)\big]_k, b\big[1..\phi(b,\omega(b)-\tau_o)\big]_k\Big) > 0$.*

So instead of having to only consider the $(|a| - \tau_o + 1)$-prefix to ensure we find every b with overlap $|a \cap b|$ of at least τ_o, we have to consider the smallest π-prefix where the total weight of the π tokens in the prefix are larger than $\omega(a) - \tau_o$ to ensure we find every b with overlap $\sum_k \min(a_k, b_k)$ of at least τ_o. The equivalent applies for which tokens of $b \in \mathcal{B}$ we need to index. See Figure 4.1b for an example.

We extend to other similarity measures the same way as for the unweighted case. Table 4.1 lists the prefix size $\pi(x,\tau)$ for all four similarity measures. The prefix size of a weighted set with respect to some similarity threshold is, instead of being the sufficient number of tokens, the total weight the prefix need to exceed.

4.4.4 Size Filter

A simple, yet effective, technique is to crop inverted lists using the size of the token sets. We note that if $|a \cap b| \geq \tau_o$, then $|b| \geq \tau_o$. In other words, τ_o is a lower bound on the size of b. Therefore, if we sort the inverted lists for the tokens of $\mathcal{B}$ by increasing size of b we can efficiently skip the beginning of the list until $b \geq \tau_o$.

Generalizing to other set similarity measures is a matter of bounding $|b|$ using τ and $|a|$. For example, for a Jaccard threshold τ_j we have that

$$\begin{gathered} \operatorname{sim}_j(a,b) = \frac{|a \cap b|}{|a \cup b|} \geq \tau_j \\ \Downarrow \\ |b| \geq \tau_j \cdot |a| \end{gathered}$$

We can see this in use at line 8 in Algorithm 2. For the normalized set similarity measures we can also identify an upper bound and stop traversal of the ordered inverted lists early, which is useless in the presence of the indexing trick for self-join in Algorithm 2, but will be useful for nonself-joins. Table 4.1 lists the upper and lower size bounds, $\lambda_u(a,\tau)$ and $\lambda_l(a,\tau)$ for the different similarity measures.

The extension to weighted sets when the overlap is defined as $\sum_k \min(a_k, b_k)$ is straightforward, follows the same derivation, and ends up with the same bounds (see Table 4.1). The only difference being the size bounds being on total weight instead of cardinality.

4.4.5 Positional Filter

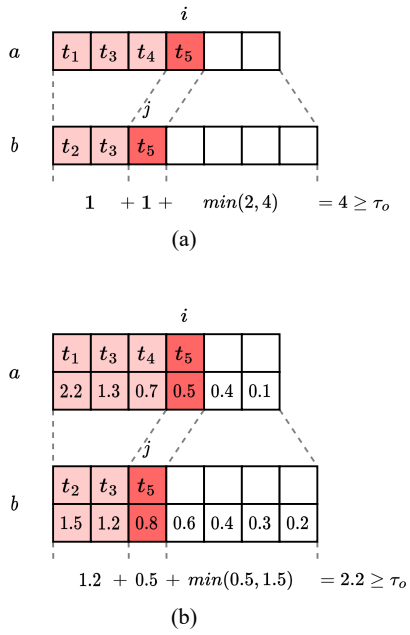


Figure 4.2: Example of the positional filtering principle for (a) unweighted and (b) weighted tokens sets. The token sets are sorted by the total ordering $\mathcal{O}$. The example show how we can use the fact that $a[i] = b[j]$ and that we know the overlap between $a[1..(i-1)]$ and $b[1..(j-1)]$ to calculate an upper bound on the overlap between a and b.

While prefix filtering enables us to reduce the number of tokens we need to probe, and size filtering let us ignore parts of the inverted lists, positional filtering let us dismiss many explicit pairs without calculating their similarity.

The key idea is that if we know a and b have a common token on position i and j (such that $a[i] = b[j]$), and we know the overlap between the prefixes $a\big[1..(i-1)\big]$ and $b\big[1..(j-1)\big]$, we can bound their total overlap because $a\big[(i+1)..|a|\big] \cap b\big[(j+1)..|b|\big] \leq \min(|a|-i, |b|-j)$. This is known as the *positional filtering principle* [144] (adapted for this paper):

Lemma 3 (Unweighted Positional Filtering Principle) *Let a and b be two token sets and $x[i..j]$ be the ith to the jth (inclusive) ranked tokens of a token set x according to some total ordering $\mathcal{O}$. If $|a \cap b| \geq \tau_o$ and $a[i] = b[j]$, then* $\Big|a\big[1..(i-1)\big] \cap b\big[1..(j-1)\big]\Big| + 1 + \min\big(|a|-i, |b|-j\big) \geq \tau_o$.

See Figure 4.2a for an example. We utilize this principle by keeping track of how many common tokens we have seen between a and all token sets in $\mathcal{B}$ as we traverse the inverted lists for a's prefix in order of $\mathcal{O}$. Additionally, we store the position j of tokens within the token sets in the inverted lists for $\mathcal{B}$. Thereby, when we encounter token set b as we probe the inverted list of token $a[i]$ and know the position of the token in b to be j we can simply check whether $\Big|a\big[1..(i-1)\big] \cap b\big[1..(j-1)\big]\Big| + 1 + \min(|a|-i, |b|-j) \geq \tau_o$ holds and discard the token set pair immediately if it does not.

Other Similarity Measures If we bound $|a \cap b|$ from below by some expression $\alpha(a, b, \tau)$ using $|a|$, $|b|$, and some similarity threshold τ we can use $\alpha(a, b, \tau)$ instead of τ_o in the inequality since $|a \cap b| \geq \alpha(a, b, \tau) \geq \tau_o$. We call $\alpha(a, b, \tau)$ an equivalent overlap. For Jaccard, one can derive $\alpha(a, b, \tau) = \frac{\tau}{\tau+1}(|a| + |b|)$, and we see this being used on line 11 in Algorithm 2. Table 4.1 lists equivalent overlaps for all similarity measures.

Weighted Sets The positional filtering principle can be extended to weighted sets:

Lemma 4 (Weighted Positional Filtering Principle) *Let a and b be two weighted token sets and assume some total token ordering $\mathcal{O}$. If $\sum_k \min(a_k, b_k) \geq \tau_o$ and $a[i] = b[j]$, then*

$$\sum_k \min\Big(a\big[1..(i-1)\big]_k, b\big[1..(j-1)\big]_k\Big) + \min(a[i], b[j])$$
$$+ \min\Big(\omega\Big(a\big[(i+1)..|a|\big]\Big), \omega\Big(b\big[(j+1)..|b|\big]\Big)\Big) \geq \tau_o$$

Figure 4.2b shows an example. This means we can utilize positional filtering the same way as before by using the accumulated overlap with different token sets $b \in \mathcal{B}$ as we traverse the prefix of a, but instead of storing the position j we store $\omega\Big(b\big[(j+1)..|b|\big]\Big)$. One problem with this naive extension of positional filtering is that in order to calculate $\min(a[i], b[j])$ we must either look up $b[j]$ or grow the index to include it. The PPJoin authors simply suggest to store $\omega\Big(b\big[j..|b|\big]\Big)$ and ignores the $\min(a[i], b[j])$ term [144]. This effectively exploits the fact that

$$\begin{aligned}&\min(a[i], b[j]) + \min\left(\omega\Big(a\big[(i+1)..|a|\big]\Big), \omega\Big(b\big[(j+1)..|b|\big]\Big)\right)\\ &\leq \min\left(\omega\Big(a\big[i..|a|\big]\Big), \omega\Big(b\big[j..|b|\big]\Big)\right)\end{aligned}$$

In our experience, this trade-off between the overhead of accessing/storing $b[j]$ and the tighter bound with $\min(a[i], b[j])$ favors their approach as long as the verification routine is fast. Therefore, we assume this form of positional filtering from here on. We can derive equivalent overlaps as in the unweighted case to support other similarity measures — see Table 4.1.

4.4.6 Generalized PPJoin

Algorithm 3: BuildIndex($\mathcal{B}, \tau$)

Input: $\mathcal{B}$ is a collection of weighted token sets already sorted by $\mathcal{O}$. τ is the similarity threshold.
Output: Inverted token indices I

```
1  I ← [∅]_t;
2  Sort B by increasing ω(b);
3  foreach b ∈ B do
4      j ← 1;
5      s_b ← ω(b);
6      while s_b ≥ σ(b, τ) do
7          I_t ← I_t ∪ {(b, s_b)};
8          s_b ← s_b − b[j].w;
9          j ← j + 1;
10 return I;
```

Figure 4.3 illustrates how the different filtering techniques come together. We call the prefix and size filter for pre-candidate filters. They prune the im-

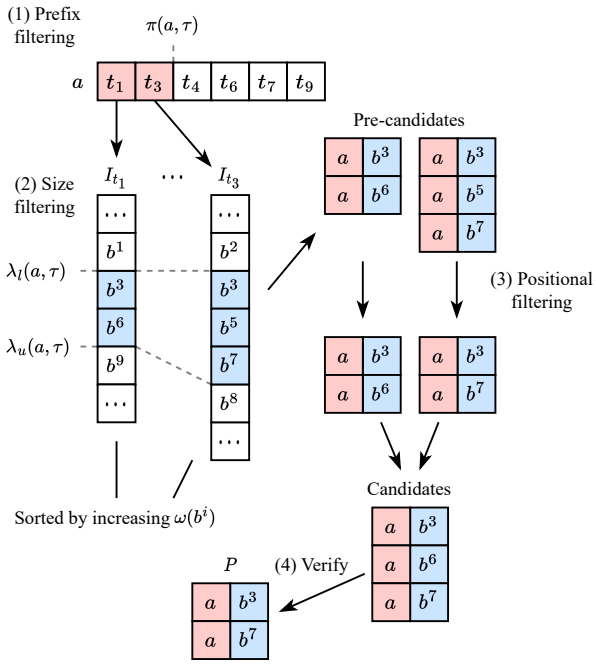


Figure 4.3: Illustration of the filtering steps in PPJoin. Inspired by Mann et al. [87].

plicit pair space and limit which explicit pairs are materialized. The positional filter is a candidate filter, which means it prunes explicit pairs before they are undergo the `Verify` routine to check if they are above the similarity threshold. While Algorithm 2 depicts PPJoin similar to how the authors do [143, 144], we also provide a fully generalized version for the reader in Algorithm 4 (and 3) that performs nonself-join for weighted sets with any supported set similarity measure using all the extensions described above. This will serve as a starting point for describing our method and contrasting it to existing methods.

4.4.7 Improved Prefix Filtering for Weighted Cosine

In Table 4.1 we presented the prefix for weighted cosine similarity one gets by following the PPJoin authors [144] instructions for extending to weighted

Algorithm 4: GPPJoin($\mathcal{A}, \mathcal{B}, \tau$)

Input: $\mathcal{A}$ and $\mathcal{B}$ are collections of weighted token sets already sorted by $\mathcal{O}$. τ is the similarity threshold.
Output: Token set pairs $\{(a, b) \mid sim(a, b) \geq \tau\}$

```
1  I ← BuildIndex(B, τ);                                  // Inverted token indices
2  P ← ∅ ;                                  // Pairs with similarity at least τ
3  foreach a ∈ A do
4      O ← empty map, default value 0;
5      i ← 1;
6      p_a ← 0;
7      while p_a ≤ π(a, τ) do                                   // Prefix filter
8          t, w ← a[i];
                                                                  // Size filter
9          I'_t ← {(b, s_b) ∈ I_t | λ_l(a, τ) ≤ ω(b) ≤ λ_u(a, τ)}
10         foreach (b, p_b) ∈ I'_t do
11             o* ← O[b] + min(ω(a) − p_a, s_b)
12             if o* ≥ α(a, b, τ) then                       // Positional filter
13                 P[b] ← O[b] + w
14             else
15                 Remove O[b]
16         i ← i + 1
17         p_a ← p_a + w
18     P ← P ∪ Verify(a, B, P, τ) ;
19 return P;
```

similarity — which is close to how prefix filtering is done in AllPairs [13]. The bound is a natural extension of the unweighted case[4]. Unfortunately, prefix and size filtering with these bounds are not very effective because the bounds loosen as the token sets get bigger. To see why, we will look at an example.

Example 3 *Assume a normalized weighted set x of n unique tokens with weight $\sqrt{\frac{1}{n}}$, which then has size $\omega(x) = \sum_n \sqrt{1/n} = \sqrt{n}$. The (relative) prefix size of x for weighted cosine similarity expressed in fraction of the tokens is*

$$\frac{\omega(x) - \tau}{\omega(x)} = 1 - \frac{\tau}{\omega(x)} = 1 - \frac{\tau}{\sqrt{n}}$$

We observe that the relative prefix size goes towards 1 as n increases. In other words, prefix filtering is less and less effective as the token set grow and will

[4]However, note that it is necessary to assume normalized (unit length) vectors in order to bound prefix/suffix and size when performing prefix filtering and size filtering. This simply means one must take care to normalize all token sets before performing the similarity join.

eventually have no effect. Contrast this to jaccard similarity, which has the relative prefix size

$$\frac{(1-\tau)\cdot\omega(x)}{\omega(x)} = 1-\tau$$

The relative prefix size is independent of the number of tokens. If $\tau = 0.5$ *then prefix filtering will will avoid lookup on half of the tokens no matter how many tokens the set contains. An equivalent example could be made for size filtering.*

L2AP [2] introduces tighter bounds for cosine similarity in AllPairs. The key is to utilize the Cauchy-Schwarz inequality:

$$\begin{aligned} \mathrm{dot}(a,b) &\leq \|a\|_2 \times \|b\|_2 \\ \sum_k a_k \cdot b_k &\leq \sqrt{\sum_k a_k^2} \times \sqrt{\sum_k b_k^2} \end{aligned} \tag{4.1}$$

We can then infer a suffix size bound:

$$\begin{aligned} \tau &\leq \mathrm{dot}(a,b) \\ &= \mathrm{dot}\Big(a\big[1..(i-1)\big], b\Big) + \mathrm{dot}\Big(a\big[i..|a|\big], b\Big) \\ &= \mathrm{dot}\Big(a\big[i..|a|\big], b\Big) \\ &\leq \big\|a\big[i..|a|\big]\big\|_2 \times \|b\|_2 \\ &\leq \big\|a\big[i..|a|\big]\big\|_2 \end{aligned} \tag{4.2}$$

Which is equivalent to a prefix size bound of

$$\big\|a\big[1..(i-1)\big]\big\|_2 \geq \sqrt{1-\tau^2} \tag{4.3}$$

Note that the bound is on the L^2 norm of the prefix instead of the L^1 norm as in PPJoin. Importantly, the bound is much more effective because it does not grow with the number of tokens.

4.5 Expressive Hybrid Join Primitive

This section will describe a new flexible join primitive that will be the core building block of our proposed methods. We first discuss strength and weaknesses of three types of similarity join conditions, and argue that they complement each others weaknesses and there is merit to combining them. Therefore, we propose a new hybrid join type and will follow up in the next section with an efficient algorithmic realization of such a join.

4.5.1 Similarity Join Conditions

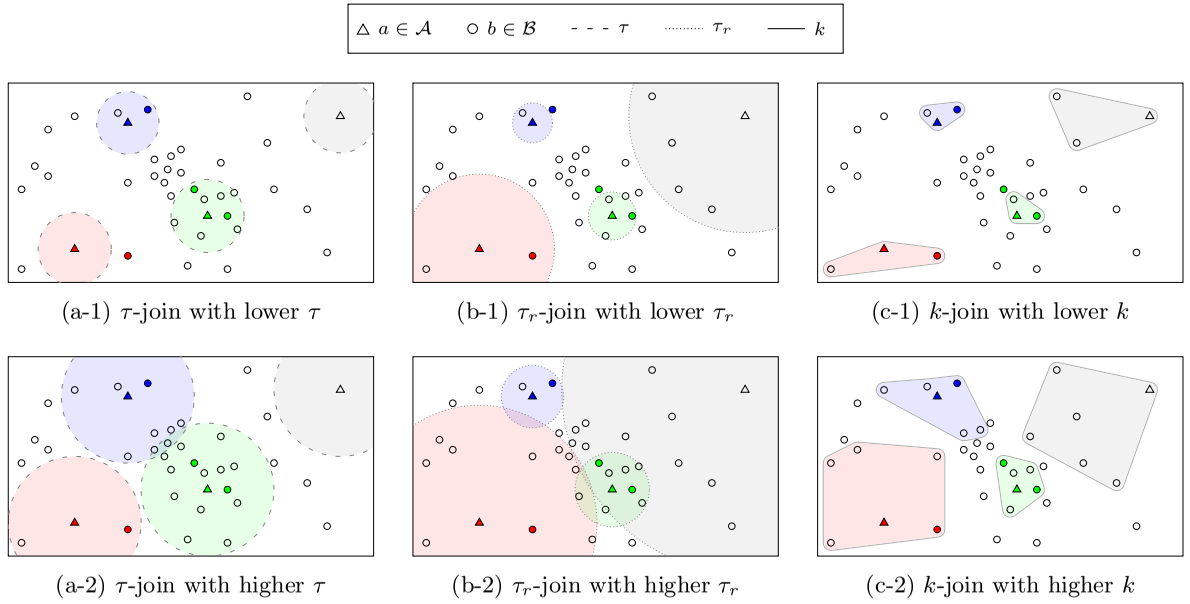
All similarity joins have an inclusion condition specifying which pairs should be returned or not. We will now briefly discuss three types of the similarity join conditions used for entity matching.

1. Absolute Similarity Threshold (τ-join) A τ-join returns all set pairs (a, b) across the set collections $\mathcal{A}$ and $\mathcal{B}$ such that $sim(a, b) \geq \tau$. This type of join for have received a substantial amount of work [e.g., 87], and PPJoin [143, 144] is an example of a τ-join. The main strength of absolute similarity thresholds is their ability to consistently filter out low similarity pairs. However, τ-joins do not handle varying similarity density well. Some sets in $\mathcal{A}$ may be quite similar to many sets in $\mathcal{B}$, while other sets in $\mathcal{A}$ may only be modestly similar to a few sets in $\mathcal{B}$. The problem is that even though a lower τ will avoid a substantial number of false positives from sets with high similarity density it will struggle to recall neighbors of sets with low similarity density. While a higher τ will have the exact opposite problem. See Figure 4.4a-1 and 4.4a-2 for an illustration of how there might not be any τ that provide an acceptable solution.

2. Relative Similarity Threshold (τ_r-join) A τ_r-join returns the pairs (a, b) for which $\text{sim}(a, b) \geq \tau_r S^*$, where $S^* = \max_{b' \in \mathcal{B}} sim(a, b')$. This type of join has not received much attention, but a very similar variant[5] was recently studied and motivated by Li et al. [78]. The strength of relative similarity thresholds is that they can exploit relative differences in similarity to filter more aggressively when there exists high-similarity matches. While normally one would be interested in b with a similarity to a of 0.6, one might want to dismiss it if there exists another b' with similarity 0.99. The main challenge is sets that are not highly similar to any other set. As S^* decreases the relative similarity of all other sets will converge towards each other. Therefore, setting τ_r to a value high enough to get acceptable recall might also lead to certain sets in $\mathcal{A}$ without any similar sets in $\mathcal{B}$ generate an unacceptable number of pairs. See Figure 4.4b-1 and 4.4b-2 for an illustration.

3. Local Cardinality Threshold (k-join) A k-join returns pairs (a, b) such that b is within the k most similar records to a. These join types have received increased attention recently for use with deep learning embeddings [e.g., 127]. Note that this is different from joins with global cardinality constraints such

[5] They formulated it with similarity distance instead of similarity.



(a-1) τ-join with lower τ (b-1) τ_r-join with lower τ_r (c-1) k-join with lower k

(a-2) τ-join with higher τ (b-2) τ_r-join with higher τ_r (c-2) k-join with higher k

Figure 4.4: Conceptual illustration of weaknesses of join conditions for τ-join, τ_r-join, and k-join. Triangle points are sets from $\mathcal{A}$, circle points are sets from $\mathcal{B}$, point color indicate matches, area color indicate inclusion according to the join condition, and euclidean proximity is similarity.

as top-k similarity join [145, 148], which return the k most similar pairs across all sets, and is outside the scope of this paper. The obvious benefit of a local cardinality threshold is that it bounds the number of returned pairs and is not prone to the same scenarios that would blow up the number of returned pairs like for τ-join and τ_r-join. It is also adaptive in the sense that it will be able to pick up both low and high similarity matches depending on the similarity density. The downside is that the number of returned pairs is static and it is not able to leverage the similarity values to reduce the number of pairs when the data suggests so. When all sets in $\mathcal{B}$ is very dissimilar to some set a it might be unnecessary to still insist on returning k pairs for a — maybe a have no matches. See Figure 4.4c-1 and 4.4c-2 for an illustration of how a k-join might get unsatisfactory recall when k is set low but might return too many pairs that are unlikely to match when k is set higher.

4.5.2 Hybrid Join Type

Join Type	Strength	Weakness
τ-join	Consistently filter out low similarity pairs	Can not adapt to varying similarity density
τ_r-join	Filters more aggressively when a set sticks out as more similar	Too forgiving when sets do not stick out with high similarity
k-join	Bounds the number of pairs and pick up both low and high similarity matches	Returns unnecessarily many pairs when all have low similarity

Table 4.2: Strength and weaknesses of the join condition for τ-join, τ_r-join, and k-join.

Table 4.2 summarizes the strengths and weaknesses of the three join conditions discussed in the previous subsection. We argue that the strengths of the three individual conditions complement the weaknesses of the others. Therefore, we propose a hybrid join type incorporating all three: (τ, τ_r, k)-join.

Definition 2 (**(τ, τ_r, k)-join**) *Let $\mathcal{A}$ and $\mathcal{B}$ be two collections of sets, and $sim(a, b)$ be some set similarity measure. Furthermore, let $\psi_a(b) : \mathcal{B} \to \{1, 2, \ldots, |\mathcal{B}|\}$ be a bijective function that order all $b \in \mathcal{B}$ by decreasing similarity to a so that*

$sim(a,b) > sim(a,b')$ if $\psi_a(b) < \psi(b')$. Finally let $S^*_a = \max_{b \in \mathcal{B}} sim(a,b)$. A (τ, τ_r, k)-join over $\mathcal{A}$ and $\mathcal{B}$ with respect to some similarity measure sim is a join that return all pairs (a,b) such that

$$sim(a,b) \geq \tau \wedge sim(a,b) \geq \tau_r S^*_a \wedge \psi_a(b) \leq k$$

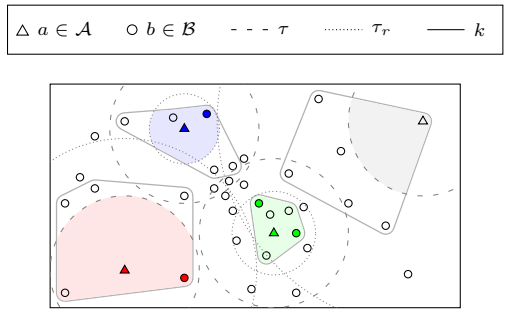


Figure 4.5: Conceptual illustration of (τ, τ_r, k)-join.

In other words, a (τ, τ_r, k)-join requires all three join condition to be met at once. Figure 4.5 illustrates how the different conditions can complement each other the same way Figure 4.4 illustrated weaknesses of the individual conditions. This join type is significantly more expressive, but the number of hyperparameters also increases, which makes it more challenging to choose hyperparameters. This will be a central part of our proposed method and will be addressed in later sections. For now, assume that they are provided.

4.6 Efficient (τ, τ_r, k)-join Algorithm

We will now propose an efficient algorithm, `TTRKJoin`, for performing (τ, τ_r, k)-joins with weighted Jaccard, Cosine, Dice, or Overlap similarity. The join routine is prefix filtering-based and is outlined in Algorithm 6. Even though `TTRKJoin` is significantly different than PPJoin, we will for the benefit of the reader motivate and describe `TTRKJoin` as a series of changes to PPJoin[6] — as this is a well-known and state-of-the-art method [87]. First, we describe how we handle the additional τ_r and k join conditions. Then, we describe how we incorporate a better cosine prefix bound from L2AP [2], before we propose a new

[6]Specifically, the generalized version, `GPPJoin`, presented in Section 4.4.6.

Algorithm 5: BuildTTRKIndex($\mathcal{B}, \tau; sim$)

Input: $\mathcal{B}$ is a collection of weighted token sets already sorted by $\mathcal{O}$. τ is the similarity threshold.

Output: Inverted token indices I^+, I^- for token sets with above and below average prefix weight.

$I \leftarrow [\emptyset]_t$;
$P \leftarrow [0]_t$;
foreach $b \in \mathcal{B}$ **do**
  $j \leftarrow 1$;
  $p_b \leftarrow 0$;
  while $s_b \geq \sigma(b, \tau)$ **do**
    $I_t \leftarrow I_t \cup \{(b, j, p_b)\}$;
    $P_t \leftarrow P_t + p_b$;
    $p_b \leftarrow p_b + b[j].w$;
    $j \leftarrow j + 1$;
$I^+ \leftarrow [\emptyset]_t$;
$I^- \leftarrow [\emptyset]_t$;
foreach $I_t \in I$ **do**
  $\overline{p}_b \leftarrow P_t / |I_t|$;
  I_t^+.minPrefix $\leftarrow \infty$;
  I_t^-.minPrefix $\leftarrow \infty$;
  foreach $(b, j, p_b) \in I_t$ **do**
    $s_b \leftarrow \omega(b) - p_b$;
    if $p_b \geq \overline{p}_b$ **then**
      $I_t^+ \leftarrow I_t^+ \cup \{(b, j, s_b)\}$;
      I_t^+.minPrefix $\leftarrow \min(I_t^+$.minPrefix$, p_b)$;
    else
      $I_t^- \leftarrow I_t^- \cup \{(b, j, s_b)\}$;
      I_t^-.minPrefix $\leftarrow \min(I_t^-$.minPrefix$, p_b)$;
  Sort I_t^+ and I_t^- by increasing s_b;
return I^+, I^-;

Algorithm 6: TTRKJoin$(\mathcal{A}, \mathcal{B}, \tau, \tau_r, k, \rho^*; sim)$

Input: $\mathcal{A}$ and $\mathcal{B}$ are collections of weighted token sets already sorted by $\mathcal{O}$. τ and τ_r are the similarity threshold and the relative similarity threshold. k is the maximum number of neighbors per token set. ρ^* is the maximum traversal rank.

Output: Results of a (τ, τ_r, k)-join with a maximum traversal rank of ρ^*

```
1  I⁺, I⁻ ← BuildTTRKIndex(B, τ);
2  P ← ∅;
3  foreach a ∈ A do
4      Q ← empty min heap;
5      V ← ∅;
6      τ̃ ← τ;
7      i ← 1;
8      ρ ← 0;
9      s_a ← ω(a);
10     while s_a ≥ σ(a, τ̃) do                                  // Prefix filter
11         t, w ← a[i];
12         foreach I_t ∈ [I_t⁻, I_t⁺] do
13             start ← binary search first I_t[start] ≥ λ_l(a, τ̃);
14             end ← binary search last I_t[end] ≤ λ_u(a, τ̃);
15             ρ ← ρ + (start − 1);
                                                                // PPS filter
16             foreach (b, j, s_b) ∈ I_t[start..end] do
17                 ρ ← ρ + 1
18                 if ρ > ρ* then break loop at line 10
                                                                // Positional filter
19                 if min(s_a, s_b) ≥ α(a, b, τ̃) then
20                     continue
21                 if b ∈ V then continue
22                 S ← PartialSim(a, b, i, j, s_a, s_b, τ̃)
23                 V ← V ∪ {b}
24                 if S ≥ τ̃ then
25                     Q.push(b, S)
26                     if |Q| > k then Q.pop()
27                     if |Q| = k then τ̃ ← max(τ̃, Q.top.sim)
28                     if τ_r · S > τ̃ then
29                         τ̃ ← τ_r · S
30                         while Q.top.sim < τ̃ do Q.pop()
31             ρ ← ρ + (|I_t| − end)
32         s_a ← s_a − w;
33         i ← i + 1;
34     P ← P ∪ Q;
35 return P;
```

Algorithm 7: PartialSim$(a, b, i, j, s_a, s_b, \tau; sim)$

Input:
Output:
if $a[|a|] > b[|b|]$ **then**
  swap(a, b); swap(i, j); swap(s_a, s_b);
$o \leftarrow \alpha(a, b, \widetilde{\tau})$;
intersection $\leftarrow 0$; // For $\sum_k \min(a_k, b_k)$
union $\leftarrow (\omega(a) - s_a) + (\omega(b) - s_b)$; // For $\sum_k \max(a_k, b_k)$
dotProduct $\leftarrow 0$;
while $i \leq |a|$ **do**
  while $a[i] > b[j]$ **do**
    $s_b \leftarrow s_b - b[j].w$;
    if $s_b < o$ **then return** 0;
    union $\leftarrow$ union $+ b[j].w$;
    $j \leftarrow j + 1$;
  if $a[i] = b[j]$ **then**
    $o \leftarrow o - \min(a[i].w, b[i].w)$;
    $s_a \leftarrow s_a - a[i].w$;
    $s_b \leftarrow s_b - b[i].w$;
    if $\min(s_a, s_b) < o$ **then return** 0;
    intersection $\leftarrow$ intersection $+ \min(a[i].w, b[i].w)$;
    union $\leftarrow$ union $+ \max(a[i].w, b[i].w)$;
    dotProduct $\leftarrow$ dotProduct $+ a[i].w \cdot b[i].w$;
    $j \leftarrow j + 1$;
  $i \leftarrow i + 1$;
while $j \leq |b|$ **do**
  $s_b \leftarrow s_b - b[j].w$;
  if $s_b < o$ **then return** 0;
  union $\leftarrow$ union $+ b[i].w$;
  $j \leftarrow j + 1$;
return $sim(intersection/union/dotproduct/\omega(a)/\omega(b))$;

effective filtering technique to replace size filtering. Following, we introduce an additional parameter to the algorithm that will be used to achieve approximate joins in Section 4.8. Lastly, we cover some implementation details.

4.6.1 Join Conditions

While it is possible to naively extend `GPPJoin` to handle the τ_r and k conditions by simply applying them to candidates from the `Verify` routine, this would not exploit the constraints put on the similarity to prune the search space. Therefore, we evolve `GPPJoin` in two main ways — one for k and one τ_r.

Local Cardinality Threshold k

We keep track of the k most similar token sets in a minimum priority queue Q as we probe the inverted lists for a. Instead of accumulating the overlap in O for all the candidates in the prefix of a, we eagerly compute the similarity (line 22) to each b on first encounter in order to maintain Q. We avoid redundant work by keeping track of already encountered token sets in V (line 21 and 23). Furthermore, let $\widetilde{\tau}$ be the similarity threshold used by prefix, size, and positional filtering. It is initially set to τ, but Q let us tighten $\widetilde{\tau}$ to $\max(\widetilde{\tau}, Q.top.sim)$ when Q have been updated and $|Q| = k$ because we know that we are not interested in token sets b with a similarity to a lower than those among the top k we have already found (line 27). Effectively, the three filters (prefix, size, positional) are now dynamic and tighten as we discover token sets with higher similarity.

Relative Similarity Threshold τ_r

Every time we compute a new similarity S to a set $b \in \mathcal{B}$ we try to tighten $\widetilde{\tau}$ with $\tau_r S$ (line 28-29). If it is tightened we also make sure to prune Q with the new threshold (line 30).

4.6.2 Incorporating L^2 Based Cosine Bounds

In Section 4.4.7 we saw that the bounds for weighted cosine from PPJoin [144] get more loose as the number of tokens increases. L2AP [2] offers a tighter bound on the prefix, but on the L^2 norm of the prefix instead of L^1. We want a unified algorithm that can handle all four similarity measures (Jaccard, Cosine, Dice, Overlap) while also exploiting this bound.

Measure	$sim(a,b)$	Norm l	Lower query suffix bound $\sigma(x,\tau)$	Index suffix bounds $\lambda_l(a,\tau)$	Index suffix bounds $\lambda_u(a,\tau)$	Equivalent overlap $\alpha(a,b,\tau)$
Jaccard	$\frac{\sum_k \min(a_k,b_k)}{\sum_k \max(a_k,b_k)}$	1	$\tau \cdot \omega(x)$	$\tau \cdot (\omega(a)+p_b)$	$\frac{s_a}{\tau} - p_a - p_b$	$\frac{\tau}{\tau+1}(\omega(a)+\omega(b))$
Cosine*	$\sum_k a_k \cdot b_k$	2	τ^2	$\frac{\tau^2}{s_a}$	∞	τ
Dice	$\frac{2 \cdot \sum_k \min(a_k,b_k)}{\lvert a\rvert + \lvert b\rvert}$	1	$\frac{\tau}{2-\tau} \cdot \omega(x)$	$\frac{\tau}{2-\tau} \cdot (\omega(a)+p_b)$	$\frac{(2-\tau)}{\tau} \cdot s_a - p_a - p_b$	$\frac{\tau}{2}(\omega(a)+\omega(b))$
Overlap	$\sum_k \min(a_k,b_k)$	1	τ	τ	∞	τ

Table 4.3: Definitions of the weighted set similarity measures together with suffix bounds and equivalent overlap we use in `TTRKJoin`. *Requires every set x to be normalized such that $\|x\|_2 = 1$.

We generalize the size function ω to be the l-norm sum:

$$\omega(x) = \|x\|_l^l = \sum_k x_k^l \tag{4.4}$$

Furthermore, let all prefixes and suffixes be l-norm sums. In this setup, Cosine has $l = 2$ while Jaccard, Dice, and Overlap have $l = 1$. The prefix size for cosine is then $1 - \tau^2$, while they remain the same for the other similarity measures. To simplify the bounds and the algorithm we operate with suffix bounds instead of prefix bounds. Table 4.3 lists the suffix bounds for the four similarity measures, as well as the norm to use. Note that we can not do size filtering for cosine with this definition of size because it will always be 1 for normalized weighted token sets. Luckily, we will not be using size filtering.

4.6.3 Prefix-Partitioned Suffix Filtering

PPJoin (and many other set similarity joins) accumulates overlap for candidates over all tokens in the prefix before computing all similarities. We compute similarities eagerly on first occurrence of a candidate in an inverted list and simply ignore the same candidate if it shows up in subsequent inverted lists. This allows us to make use of the local cardinality threshold k and relative threshold τ_r to prune more aggressively by increasing the dynamic threshold $\widetilde{\tau}$ during the search. Moreover, it also opens up alternative ways than size filtering to trim the inverted lists. We will now describe our proposed pre-candidate filter, the Prefix-Partitioned Suffix (PPS) Filter, in three steps.

Suffix Instead of Size

The key observation is that when we encounter token set b for the first time when looking up a's ith token and that token has position j in b we know that there is no overlap between $a[1..(i-1)]$ and $b[1..(j-1)]$. To simplify notation, let the prefix and suffix of a and b be $p_a = \omega(a[1..(i-1)])$, $s_a = \omega(a[i..|a|])$, $p_b = \omega(b[1..(i-1)])$, and $p_b = \omega(b[j..|b|])$.

The overlap between a and b is upper bounded by the suffix s_b (and s_a). Therefore, we can sort by and threshold s_b instead of $w(b)$ in size filtering with the same bounds. We call this a suffix filter[7] — in contrast to size filter. It makes intuitively sense to filter on the suffix of sets in $b \in \mathcal{B}$ because the suffix of b reflects the remaining tokens a can still overlap with. In effect, the suffix filter moves part of the pruning power of the positional filter from candidate filtering to pre-candidate filtering. It has significantly tighter lower bounds than the size filter because $s_b \leq \omega(b)$, but also has looser upper bounds. However, reusing the bounds from size filtering is naive, and we can do better with the extra positional information we have available.

Position-Enhanced Index Suffix Bounds

Tighter bounds can be inferred by using the fact that $a[1..(i-1)]$ and $b[1..(j-1)]$ will not have any overlap. Take Jaccard as an example (with threshold τ_j):

$$\begin{aligned} \frac{\sum_k \min(a_k, b_k)}{\sum_k \max(a_k, b_k)} &\geq \tau_j \\ \frac{\sum_k \min(a_k, b_k)}{\omega(a) + \omega(b) - \sum_k \min(a_k, b_k)} &\geq \tau_j \\ (\tau_j + 1) \sum_k \min(a_k, b_k) &\geq \tau_j(\omega(a) + \omega(b)) \\ (\tau_j + 1) \sum_{k \geq k_s} \min(a_k, b_k) &\geq \tau_j(\omega(a) + \omega(b)) \end{aligned} \tag{4.5}$$

[7] Not to be confused with the suffix filter in PPJoin+ [144], which performs candidate (not pre-candidate) filtering.

where $k_s = \max(a[i].k, b[j].k)$. We can use the fact that $\sum_{k \geq k_s} \min(a_k, b_k)$ is upper bounded by s_b and s_a to get a lower and upper bound of s_b:

$$\begin{aligned}
(\tau_j + 1) \sum_{k \geq k_s} \min(a_k, b_k) &\geq \tau_j(\omega(a) + \omega(b)) \\
(\tau_j + 1)s_b &\geq \tau_j(\omega(a) + \omega(b)) \\
(\tau_j + 1)s_b &\geq \tau_j(\omega(a) + p_b + s_b) \\
s_b &\geq \tau_j(\omega(a) + p_b)
\end{aligned} \tag{4.6}$$

$$\begin{aligned}
(\tau_j + 1) \sum_{k \geq k_s} \min(a_k, b_k) &\geq \tau_j(\omega(a) + \omega(b)) \\
(\tau_j + 1)s_a &\geq \tau_j(\omega(a) + \omega(b)) \\
(\tau_j + 1)s_a &\geq \tau_j(p_a + s_a + p_b + s_b) \\
s_b &\leq \frac{s_a}{\tau_j} - p_a - p_b
\end{aligned} \tag{4.7}$$

Bounds for the other similarity measures can be inferred in similar fashion[8]. They are listed in Table 4.3.

The bounds rely on p_a, s_a, and p_b. The attentive reader might have realized that, while p_a and s_a is known when applying the suffix filter, we do not actually know p_b because it might (and most likely will) differ between the different entries in the inverted list I_t. We must replace it with a lower bound. Instead of simply using zero we can use the minimum p_b in I_t — denoted I_t.minPrefix.

Prefix-Partitioned Index

For Jaccard and Dice, where the bounds rely on p_b, higher I_t.minPrefix results in tighter suffix bounds. This makes intuitive sense since a larger prefix of index sets $b \in I_t$ that does not overlap with the query a means a smaller fraction of each b's size can overlap with a. Obviously, with large inverted lists it is likely that I_t.minPrefix will be low.

In order to prune sets with large prefixes more aggressively we partition each I_t into two inverted lists I_t^+ and I_t^-, containing sets with prefixes above and below the average prefix size $\overline{p}_b$. Both I_t^- and I_t^+ are still sorted by s_b and we can crop them the same way we would for a suffix sorted I_t, but now I_t^+.minPrefix will be at least $\overline{p}_b$. The additional work at indexing time is negligible but it introduces some overhead at query time because twice as many lists need to be

[8] We use the Cauchy-Schwarz inequality for Cosine.

cropped. Assuming an appropriate spread in prefix sizes the increased pruning of I_t^+ can make up for it and more.

While other more intricate partitioning schemes are possible, we find this to be a reasonable approach that exploits additional positional information without severely increasing the complexity or overhead compared to a standard sorted inverted list. We deem further exploration of alternative approaches outside the scope of this paper and hope to address it in future work.

Putting it Together

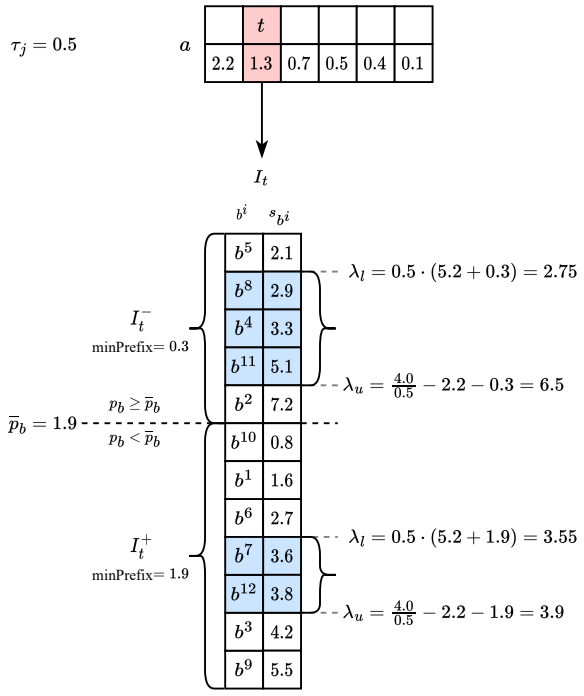


Figure 4.6: An example of performing Prefix-Partitioned Suffix Filtering when looking up token t of token set a and looking for matches with a Jaccard similarity of at least 0.5.

Algorithm 5 outlines the construction of the index with prefix-partitioned

inverted lists sorted by suffix size. We first construct inverted lists for all tokens and then partition them on prefix and sort by suffix afterwards. The actual implementation partitions I_t in-place and reuses its memory for I_t^- and I_t^+. The runtime complexity is the same as for building `GPPJoin` index (Algorithm 3): $O(T + |\mathcal{T}|\mathrm{df}^* \log \mathrm{df}^*)$, where df^* is the maximum frequency of any token among the token sets in $\mathcal{B}$. We use the partitioned lists at line 12 in Algorithm 6. Note that we crop and traverse I_t^- first because it is (heuristically) more likely to contain high-similarity sets. Figure 4.6 show an example of performing Prefix-Partitioned Suffix Filtering.

4.6.4 Early Traversal Rank Cutoff (ρ^*)

The filtering techniques employed allow us to prune the search space aggressively by skipping and cropping the inverted lists and avoid full similarity comparison, but sometimes this is not enough. This is especially true when the thresholds are set conservatively (low τ and τ_r, high k). Therefore, we introduce an additional parameter to stop the search for each $a \in \mathcal{A}$ early.

Definition 3 *Assume we are querying a and $I^-_{a[1]}, I^+_{a[1]}, \ldots, I^-_{a[|a|]}, I^+_{a[|a|]}$ are all the inverted lists of the tokens in a. To simplify notation, let $I^-_{a[n]} = I_{2n-1}$ and $I^+_{a[n]} = I_{2n}$. The traversal rank ρ for the qth entry in I_n is $q + \sum_{i=1}^{n-1} |I_{a[i]}|$.*

The parameter ρ^* is the *maximum traversal rank* and Algorithm 6 will only consider entries of the inverted indices with $\rho \leq \rho^*$. Note that the traversal rank of an entry is stable across different values of τ, τ_r, and k. Thus, ρ^* will always prune the same entries. This requires some extra bookkeeping (line 15 and 31) to handle the PPS filter.

Setting $\rho^* = \infty$ yields a (τ, τ_r, k)-join, but lower values might yield different results. This parameter is the key to perform approximate (τ, τ_r, k)-joins. We postpone the discussion about how to interpret and automatically set this parameter to Section 4.8.

4.6.5 Exploiting Parallelization

State-of-the-art deep learning methods exploit modern GPUs to train and run their neural networks in a massively parallel fashion. While there is no trivial way to leverage GPUs for our join algorithm, it is relatively easy to make use of multiple CPU cores since the main loop on line 3 is embarrassingly parallelizable. Additionally, we can parallelize partitioning and sorting of the inverted lists in the `BuildTTRKIndex` routine (Algorithm 5).

4.6.6 Fast Verification

We adapt the verification routine from Mann et al. [87] to weighted sets and other similarity measures in order to form a fast similarity routine. See `PartialSim` in Algorithm 7. Since we compute the similarity when encountering first common token t and we know the position of t in a and b[9] as well as their suffixes s_a and s_b, we can skip the tokens $a[1..(i-1)]$ and $b[1..(j-1)]$ (remembering to take care of the Jaccard denominator). Furthermore, we can keep track over the remaining suffix weight of the token sets, as well as the remaining necessary overlap, to detect early that we can not meet the similarity threshold. Note that Algorithm 7 computes the terms for all similarity measures to give a complete picture, but in practice we can compute only the ones necessary for the measure at hand.

4.7 Join Behaviour Estimation Framework

`TTRKJoin` have four hyperparamters that are hard to set. No matter the techniques we employ to make the algorithm efficient, it will still be very resource intensive to explore different hyperparameter configurations by simply running the algorithm with a large number of different configurations. So central to the method we will propose is the ability to estimate its behaviour with different hyperparameters without running the full algorithm. We are interested in estimating the three key characteristics of blocking through their main measure: 1) Recall, 2) Number of retrieved pairs, and 3) Runtime. In this section, we will present a framework for estimating these properties through analysis of select or sampled queries.

4.7.1 Recall Conditions

The only reliable way to estimate recall (at least in the general case) is to evaluate against known matches. Assume we have a representative subset of known matches $\widetilde{M} \subseteq M$. Instead of running `TTRKJoin` with every hyperparamter configuration of interest to find the recall of $\widetilde{M}$, assume for each known match $m \in \widetilde{M}$ we know the maximum τ and τ_r and the minimum k and $\rho*$ that is necessary to recall it. We call those four values the recall conditions of m — denoted θ_m. Given a hyperparameter configuration $(\tau, \tau_r, k, \rho^*)$ we can answer

[9] Note that `BuildTTRKIndex` stores both the position j and the suffix size s_b of the token t in b in the inverted list.

whether a match m would be recalled by `TTRKJoin` in constant time by checking whether the hyperparameters respect the recall conditions θ_m. Furthermore, if we have the recall conditions $\Theta = \{\theta_m\}$ for all $m \in \widetilde{M}$ we can trivially estimate the recall in $O(|\widetilde{M}|)$ time. For this strategy to be effective we need to able to find the recall conditions efficiently.

Finding Recall Conditions

Algorithm 8: FindRecallCond($\mathcal{A}, I^{\pm}, \widetilde{M}, D, [\tau, \tau_r, k, \rho^*]; sim$)

Input: $\mathcal{A}$ is a collection of weighted token sets already sorted by $\mathcal{O}$. $I^{\pm}$ is a PPS index of $\mathcal{B}$. $\widetilde{M} \subseteq \mathcal{A} \times \mathcal{B}$ are known matches. D are regularization parameter values of interest. τ, τ_r, k, ρ^* are the most permissive values of the `TTRKJoin` parameters that the conditions will cover (defaults to $[0, 0, \infty, \infty]$).

Output: Θ^d, the strictest join conditions $(\tau, \tau_r, k, \rho^*)$ that would recall each match in $\widetilde{M}$ per regularization configuration $d \in D$.

```
/* TTRKJoin with the following changes:                              */
// 1. Use provided index
```
1 $I^+, I^- \leftarrow I^{\pm}$;
```
// 2. Initialize conditions
```
2 $\Theta \leftarrow \{\emptyset\}^d$;
```
...
// 3. Query A_M~ instead of A
A_M ← {a ∈ A | ∃b∈B[(a,b) ∈ M~]};
```
3 **foreach** $a \in \mathcal{A}_{\widetilde{M}}$ **do**
```
    ...
    // 4. Use matches to tighten threshold
```
6 $\widetilde{\tau} \leftarrow \max\left(\tau, (1 - \max D) \min_{b|(a,b)\in\widetilde{M}} sim(a,b)\right)$;
```
    ...
                    // 5. Store ρ for each candidate
```
25 Q.push($(b, \rho), S$);
```
    ...
    // 5. Recall conditions instead of candidates
```
34 $\Theta^d \leftarrow$ extract recall conditions from Q for each $d \in D$;

35 **return** Θ;

We can run a modified version of `TTRKJoin` once to determine the recall conditions of $\widetilde{M}$. Algorithm 8 outlines this modified version, `FindRecallCond`. The idea is to search for each a that occur in a match $(a, b) \in \widetilde{M}$, and then extract the recall conditions from Q. We make sure to store the traversal rank ρ of when b was discovered in Q (Line 25). The recall conditions are then trivially found by sorting in descending similarity order and traversing Q looking for

each b that match a. Given (b, ρ) at position i in sorted Q, the maximum τ is $Q[i].sim$, the maximum τ_r is $Q[i].sim/Q[1].sim$, the minimum k is i, and the minimum ρ^* is ρ.

We make two optimizations beyond those in `TTRKJoin`. First, we only query $a \in \mathcal{A}$ which are part of at least one match (Line 3). Second, we can tighten $\widetilde{\tau}$ up front (Line 6) since we know which $b \in \mathcal{B}$ we are interested in finding and token sets with lower similarity will not affect their recall conditions.

Regularization

If we use recall conditions for a limited sized $\widetilde{M}$ to decide hyperparameter configurations we risk overfitting. Therefore, we propose a simple, yet effective, regularization parameter applied on recall conditions to control overfitting: the similarity margin d. Given a match $m = (a, b)$, the recall conditions θ_m^d for m with similarity margin d is the recall conditions m would have if the similarity of a and b was $(1 - d)sim(a, b)$ and all other similarities remained unchanged. Higher values of d give more conservative recall conditions and stronger regularization.

`FindRecallCond` (Algorithm 8) accepts a set D of similarity margins and returns the recall conditions for all $m \in \widetilde{M}$ for each $d \in D$. The recall conditions for different d are extracted from the sorted Q by simulating the effect it would have if the similarity was $(1-d) \cdot sim(a, b)$. This is straightforward for maximum τ, maximum τ_r, and minimum k, but the effect on minimum ρ^* is not easily defined. As a heuristic, we set the minimum ρ^* to the maximum of the traversal rank of b and the candidate at the position in Q we would replace if we lowered the similarity of b to $(1-d) \cdot sim(a, b)$. Note that in order to be sure Q contains enough candidates to see the effect of each $d \in D$ we factor them in when tightening $\widetilde{\tau}$ with the matches on line 6.

4.7.2 Search Trajectories

One uncomplicated way to estimate the number of retrieved pairs and runtime for a hyperparameter configuration without running the full algorithm is to randomly sample a subset $\widetilde{\mathcal{A}} \subseteq \mathcal{A}$, run `TTRKJoin` on it, and then multiply the resulting number of pairs and runtime with $|\mathcal{A}|/|\widetilde{\mathcal{A}}|$. Assuming $|\widetilde{\mathcal{A}}|$ is large enough, it will provide accurate estimations with a $|\widetilde{\mathcal{A}}|/|\mathcal{A}|$ reduction in runtime. Unfortunately, that might still be too computationally expensive if we want to check many hyperparameter configurations.

Therefore, we propose an approach were we selectively record the state of the search and runtime as we query each $a \in \widetilde{\mathcal{A}}$ once, and then use this information to quickly simulate approximate runs of `TTRKJoin` on $\widetilde{\mathcal{A}}$. The result of the simulated runs will provide us with pessimistic estimations of the number of returned pairs and runtime. Additionally, we can estimate the sum of similarities for the returned pairs, which will be important when doing approximate joins.

Let a search trajectory checkpoint reflect the state of the search at some traversal rank ρ and consist of six parts:

1. CH_P: Cumulative histogram quantifying the number of pairs with similarity below $|CH_P|$ linearly spaced similarity levels between 0 and the highest possible similarity[10].
2. CH_S: Cumulative histogram quantifying the summed similarity for pairs with similarity below $|CH_S|$ linearly spaced similarity levels between 0 and the highest possible similarity.
3. $S^-_{\log k}$: Lower bound on similarities for the top k pairs for exponentially spaced values of k such that $k = \text{argmax}_{\hat{k}} \left[\lceil \log \hat{k} \rceil = \lceil \log k \rceil\right]$.
4. rt: Empirically measured runtime since the start of query for a.
5. S^*: Highest similarity among the candidates.
6. s_a: The current prefix of the query a we have traversed so far.

Let a search trajectory ψ for a token set a on index $I^{\pm}$ be a list of checkpoints for exponentially spaced values of ρ. We can record search trajectories for $\widetilde{\mathcal{A}}$ by running a modified version of `TTRKJoin`, as outlined in Algorithm 9. We will now briefly explain how we use the recorded trajectories to estimate the number of returned pairs, runtime, and similarity sum.

Estimating the Number of Pairs

In order to be more robust to overfitting we estimate an upper bound of the number of pairs instead of the number of pairs directly. We do this by first upper bounding the returned pairs from querying $a \in \widetilde{\mathcal{A}}$ and then multiplying the upper bound with $|\mathcal{A}|/|\widetilde{\mathcal{A}}|$ to get an estimated upper bound on the total number of pairs returned.

[10]The highest possible similarity is 1 for Jaccard, Cosine, and Dice — while for Overlap it is $\min(\omega(a), \max_{b \in \mathcal{B}} \omega(b))$.

Algorithm 9: RecordTrajectories($\widetilde{\mathcal{A}}, I^{\pm}, [\tau, \tau_r, k, \rho^*]; sim$)

Input: $\widetilde{\mathcal{A}}$ is a collection of weighted token sets already sorted by $\mathcal{O}$. $I^{\pm}$ is a PPS index of $\mathcal{B}$. τ, τ_r, k, ρ^* are the most permissive values of the `TTRKJoin` parameters that the trajectories will cover (defaults to $[0, 0, \infty, \infty]$).

Output: Ψ, search trajectory for each $a \in \widetilde{\mathcal{A}}$.

Constants: $|H_P| = |H_S| = |CH_P| = |CH_S| = 100$; $r_\psi = 1.1$; Base number of $\log k$: 1.1;

```
/* TTRKJoin with the following changes:                                   */
// 1. Use provided index
```
$I^+, I^- \leftarrow I^{\pm}$;
```
// 2. Initialize trajectories
```
$\Psi \leftarrow \emptyset$;

...

foreach $a \in \widetilde{\mathcal{A}}$ **do**

```
    // 3. Initialize trajectory bookkeeping
```
    $\psi \leftarrow$empty array, $\rho_\psi \leftarrow 1$, $S^* \leftarrow 0$;
    $H_P, H_S \leftarrow$ empty histograms, $start \leftarrow now()$;
    ...
        foreach $(b, j, s_b) \in I_t[start..end]$ **do**
```
            // 4. Store trajectory checkpoints
```
            if $\rho \geq \rho_\psi$ **then**
                $\rho_\psi \leftarrow \lceil r_\psi \cdot \rho_\psi \rceil$;
                $rt \leftarrow now() - start$;
                $CH_P, CH_S, S^-_{\log k} \leftarrow$ from H_P and H_S;
                $\psi.push((CH_P, CH_S, S^-_{\log k}, rt, S^*, s_a))$;
            ...
```
                // 5. Maintain H_P, H_S, and S*
```
                Q.push(b, S), $H.add(S)$;
                $S^* \leftarrow \max(S^*, S)$;
                if $|Q| > k$ **then**
                    $S_{pop} \leftarrow Q.pop().sim$;
                    Remove S_{pop} from H_P and H_S;
                ...
                    while $Q.top.sim < \widetilde{\tau}$ **do**
                        $S_{pop} \leftarrow Q.pop().sim$;
                        Remove S_{pop} from H_P and H_S;
    ...
```
    // 6. Last checkpoint
```
    $ts \leftarrow now() - start$;
    $CH_P, CH_S, S^-_{\log k} \leftarrow$ from H_P and H_S;
    $\psi.push((CH_P, CH_S, S^-_{\log k}, rt, S^*, s_a))$;
    $\mathcal{S} \leftarrow$ sorted descending similarities from Q;
    $C\mathcal{S} \leftarrow$ cumulative $\mathcal{S}$;
```
    // 7. Store trajectory
```
    $\Psi \leftarrow \Psi \cup \{(\psi, \mathcal{S}, C\mathcal{S})\}$;

return Ψ;

To get an upper bound on the number of pairs for a single query of a with hyperparameters $(\tau, \tau_r, k, \rho^*)$ using the corresponding search trajectory ψ, first lookup the checkpoint with lowest ρ such that $\rho \geq \rho^*$. Then lookup the lowest similarity bin of CH_P which interval lies below (or touches) $\max(\tau, \tau_r S^*)$ — denoted $CH_P[\max(\tau, \tau_r S^*)]$. Taking k into consideration, the lower bound is $\min(k, CH_P[\max(\tau, \tau_r S^*)])$. Sum the upper bounds for all $a \in \widetilde{\mathcal{A}}$ and multiple by $|\mathcal{A}|/|\widetilde{\mathcal{A}}|$ to get the estimate for (an upper bound of) the total number of returned pairs. This is done in $\Theta(|\widetilde{\mathcal{A}}|)$ time.

Estimating Runtime

To estimate an upper bound for the runtime of a single query a with hyperparameters $(\tau, \tau_r, k, \rho^*)$ using the corresponding search trajectory ψ, we binary search for the checkpoint with lowest ρ that satisfy $\rho \geq \rho^*$ and $s_a \geq \sigma\big(a, \max[\tau, \tau_r S^*, S^-_{\lceil \log k \rceil}]\big)$. The estimated upper bound for the runtime is then the empirical runtime rt for that checkpoint. Sum the estimates for all $a \in \widetilde{\mathcal{A}}$ and multiple by $|\mathcal{A}|/|\widetilde{\mathcal{A}}|$ to get the estimate for total runtime of the join[11]. Since the traversal rank is always bounded by the total number of tokens $\sum_{\mathcal{B}}$ in $\mathcal{B}$, and the checkpoints are exponentially spaced, estimating the runtime is $O(|\widetilde{\mathcal{A}}| \log \log \sum_{\mathcal{B}})$.

There are many sources of uncertainty when estimating the runtime like this. Measuring empirical runtime is in itself unreliable, the `RecordTrajectories` routine have higher overhead than `TTRKJoin`, and it does not effectively capture the effect of the PPS and positional filter (only prefix) when using different hyperparameters. However, we argue that it is sufficient as an heuristic to distinguish different orders of magnitude in runtime when automatically picking hyperparameters. The reported results of our method will back up this claim.

4.8 Approximate Joins

One can achieve great runtime performance for even large datasets with similarity joins by picking the similarity threshold aggressively enough [87]. Unfortunately, for datasets that require conservative thresholds (e.g., $\tau < 0.4$ for cosine, or equivalently picking k moderately high for a k-join approach) to get high recall it is challenging to avoid approaching quadratic runtime. The main reason

[11] In addition, divide by number of threads when running in parallel.

is because the effectiveness of prefix filtering degrades quickly once the similarity threshold is so low we do not prune away high frequency tokens anymore — we can end up checking most token sets in $\mathcal{B}$.

A popular approach to scalability problems when further impactful algorithmic improvements are difficult to pull off is to relax the requirements and allow approximations that are good enough for practical purposes. Note that for string similarity-based blocking in particular this is not unreasonable since the similarity measures and thresholds we use are already in reality approximations. In this section, we will propose a simple and effective way of performing approximate (τ, τ_r, k)-joins with `TTRKJoin`.

4.8.1 Definitions

Let us start by defining what constitute an approximate join and the quality of it.

Definition 4 (Approximate (τ, τ_r, k)-Join) *Let $\mathcal{A}$ and $\mathcal{B}$ be two collections of sets, and $sim(a, b)$ be some set similarity measure. An approximate (τ, τ_r, k)-join is any routine that will return a set of pairs P such that for all pairs $(a, b) \in P$*

$$sim(a, b) \geq \tau \wedge sim(a, b) \geq \tau_r \max_{(a', b') \in P | a' = a} \left[sim(a', b') \right]$$

and that for all $a \in \mathcal{A}$

$$\left| \{(a', b') \in P \mid a' = a\} \right| \leq k$$

In other words, while a (τ, τ_r, k)-join must return all pairs that are among the k most similar and within τ_r of the most similar for each $a \in \mathcal{A}$ in addition to having a similarity of at least τ, an approximate (τ, τ_r, k)-join must only return some set of pairs that is internally consistent. That is, there can not be more than k pairs for each $a \in \mathcal{A}$, two pairs (a, b) and (a, b') such that $sim(a, b) < \tau_r \cdot sim(a, b')$, or any pairs with similarity below τ. In order to quantify how well an approximate (τ, τ_r, k)-join approximates an exact (τ, τ_r, k)-join we define the quality of an approximate join.

Definition 5 (Quality of Approximate Join) *Let $\mathcal{B}_a$ be the token sets from $\mathcal{B}$ matched to some token set $a \in \mathcal{A}$ by a (τ, τ_r, k)-join and let $S_a^* = \max_{b \in \mathcal{B}_a} sim(a, b)$.*

Furthermore, let $\widehat{\mathcal{B}}_a$ be the token sets matched to a from an approximate (τ, τ_r, k)-join. The quality of this approximate (k, τ, τ_r)-join on $\mathcal{A}$ and $\mathcal{B}$ is defined as

$$\frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \frac{\sum_{b \in \widehat{\mathcal{B}}_a | sim(a,b) \geq \tau_r S_a^*} sim(a, b)}{\sum_{b \in \mathcal{B}_a} sim(a, b)}$$

The quality of an exact (k, τ, τ_r)-join is trivially always 1. The intuition is that approximations that miss high-similarity pairs that stick out and can not be replaced are of low quality, while approximations that miss some pairs but have found others with comparable similarity are of high quality. Note that we only include $b \in \widehat{\mathcal{B}}_a$ when the similarity is at least τ_r of the highest similarity that exist to a. This is so approximate joins can not be rewarded for not including the highest similarity pair for any $a \in \mathcal{A}$ (and achieving a quality above 1).

Example 4 *Assume $\tau = 0.2, \tau_r = 0.5, k = 4$ and let $\mathcal{A} = \{a_1, a_2\}$ and $\mathcal{B} = \{b_1, \ldots, b_6\}$. The similarity according to some similarity measure is*

	b_1	b_2	b_3	b_4	b_5	b_6
a_1	*0.5*	*0.8*	*0.1*	*0.4*	*0.6*	*0.6*
a_2	*0.4*	*0.3*	*0.5*	*0.9*	*0.2*	*0.4*

The result of a (τ, τ_r, k)-join would be

$$P = \{(a_1, b_1), (a_1, b_2), (a_1, b_5), (a_1, b_6), (a_2, b_3), (a_2, b_4)\}$$

Let P_1 and P_2 be the result of two different approximate joins:

$$\begin{aligned} P_1 &= \{(a_1, b_2), (a_1, b_4), (a_1, b_5), (a_1, b_6), (a_2, b_4)\} \\ P_2 &= \{(a_1, b_1), (a_1, b_4), (a_1, b_5), (a_1, b_6), (a_2, b_1), (a_2, b_2), \\ &\quad (a_2, b_3), (a_2, b_6)\} \end{aligned}$$

The corresponding quality is

$$\begin{aligned} q_1 &= \frac{1}{2}\left(\frac{0.8 + 0.4 + 0.6 + 0.6}{0.5 + 0.8 + 0.6 + 0.6} + \frac{0.9}{0.5 + 0.9}\right) = 0.80 \\ q_2 &= \frac{1}{2}\left(\frac{0.4 + 0.5 + 0.6 + 0.6}{0.5 + 0.8 + 0.6 + 0.6} + \frac{0.5}{0.5 + 0.9}\right) = 0.60 \end{aligned}$$

We are specifically interested in approximate joins that achieve a certain level of quality with some probability.

Definition 6 ((q, q_p)-Approximate (τ, τ_r, k)-Join) *A (q, q_p)-approximate (τ, τ_r, k)-join is an approximate (τ, τ_r, k)-join with quality of at least q with probability q_p.*

We will now look at how to perform a (q, q_p)-approximated join with `TTRKJoin`.

4.8.2 Approximation with `TTRKJoin`

A reasonable hypothesis for most real world data is that matching token sets will have at least one rare token in common with high probability [97]. Prefix filtering already exploits this for runtime performance but only within the constraints of still guaranteeing finding all pairs above some threshold. On the assumption that most matches have at least one rare token in common, prefix filtering with low thresholds must drudge through a disproportionately long tail of token sets that only have ubiquitous tokens in common compared to how likely it is to find any matches. Taking inspiration from information retrieval and O'hare et al. [97], a natural solution is to simply cut off the search early for each query $a \in \mathcal{A}$. Trading in some small degradation of the result for reduced runtime. Unfortunately, setting the cutoff condition is hard because its impact so heavily rely on the dataset and other hyperparameters. We do not have any control over how much quality we give up.

The key insight is that `TTRKJoin` always performs an approximate (τ, τ_r, k)-join, no matter the value of the traversal rank cutoff ρ^*, and that any approximation quality for a dataset $(\mathcal{A}, \mathcal{B})$ can be achieved by setting ρ^* high enough. In other words, for any dataset $(\mathcal{A}, \mathcal{B})$, join conditions (τ, τ_r, k), and approximation quality q there exist a finite minimum ρ^* that will make `TTRKJoin` do an approximate join with quality of at least q. Importantly, the converse is also true. That is, any `TTRKJoin` will yield an approximate join of some quality q. This opens up two main ways to utilize `TTRKJoin` for approximate joins. We can either target a specific quality level and try to determine the necessary ρ^*, or we can determine a suitable ρ^* by other means (e.g., validation/supervised learning) and then try to determine the approximation quality. We cover the former since it most relevant, but we note the latter is easily achieved in similar spirit.

4.8.3 Choosing ρ^* to achieve quality q

If we want to achieve a certain approximation quality by choosing a finite ρ^*, and without actually running a full exact join, we need to accept some level of

Algorithm 10: QualityToRank$(q, q_p, \tau, \tau_r, k, \Psi)$

Input: q and q_p is the desired approximation quality and probability of achieving at least that. τ, τ_r, k is the join conditions of a (τ, τ_r, k)-join. Ψ are a set of randomly sampled `TTRKJoin` search trajectories from the dataset of interest.

Output: ρ^*, a max traversal rank that will make `TTRKJoin` perform a (q, q_p)-approximated (τ, τ_r, k)-join.

Function `SimSumLB` *$(\tau, \tau_r, k, \rho^*, \psi)$*
  $CH_P, CH_S \leftarrow$ from the earliest checkpoint in ψ with $\rho \geq \rho^*$ (or last);
  $S^* \leftarrow$ from last checkpoint in ψ;
  $\widetilde{\tau} \leftarrow \max(\tau, \tau_r \cdot S^*)$;
  $i \leftarrow$ lowest bin of CH_P whose entire interval is at least $\widetilde{\tau}$;
  if $CH_P[i] \leq k$ **then** `// Tight similarity constraint`
    return $CH_S[i]$;
  else `// Tight cardinality constraint`
    $j \leftarrow$ lowest bin $j \geq i$ such that $CH_P[j] \geq k$;
    $\tau^+ \leftarrow$ upper similarity bound of bin j;
    return $CH_S[j] - \tau^+(CH_P[j] - k)$;

foreach $(\psi, \mathcal{S}, C\mathcal{S}) \in \Psi$ **do**
  $i \leftarrow$ binary search lowest $\mathcal{S}[i] \geq \max(\tau, \tau_r \mathcal{S}[1])$;
  $\Sigma_\psi \leftarrow C\mathcal{S}[\min(i, k)]$;
$R \leftarrow$ empty array;
foreach *bootstrap resample* $\widetilde{\Psi}$ *of* Ψ **do** `//` N_B `resamples`
  $\hat{\rho}^* \leftarrow$ binary search on log scale lowest such that
    $\frac{1}{|\widetilde{\Psi}|} \sum_{\psi \in \widetilde{\Psi}} \frac{\texttt{SimSumLB}(\tau, \tau_r, k, \rho^*, \psi)}{\Sigma_\psi} \geq q$;
  $R.push(\hat{\rho}^*)$;
Sort R ascending;
return $R\Big[\lceil q_p \cdot |R| \rceil\Big]$;

uncertainty. Therefore, we propose to use a sample of search trajectories to find a ρ^* that will yield a (q, q_p)-approximated join.

Assume we want to do a (q, q_p)-approximated (τ, τ_r, k)-join between $\mathcal{A}$ and $\mathcal{B}$. Let Ψ be a set of recorded search trajectories for a random subset $\widetilde{\mathcal{A}} \subseteq \mathcal{A}$. We can quickly determine an upper bound of the lowest ρ^* that is guaranteed to achieve quality q with `TTRKJoin` between $\widetilde{\mathcal{A}}$ and $\mathcal{B}$ using Ψ. It is mostly a matter of binary searching the exponentially spaced traversal ranks of the trajectories in Ψ because CH_S can be used to lower bound the quality. Furthermore, we can use this as an estimator for a ρ^* that achieve quality q on the full join with $\mathcal{A}$. Since we do not know the underlying distribution, we use bootstrapping to find the q_p quantile. Resample the already existing search trajectories Ψ N_B

times to produce N_B estimates and pick the empirical q_p quantile. The resulting ρ^* makes `TTRKJoin` do a (q, q_p)-approximated join. Algorithm 10 outlines the proposed approach.

4.9 ShallowBlocker

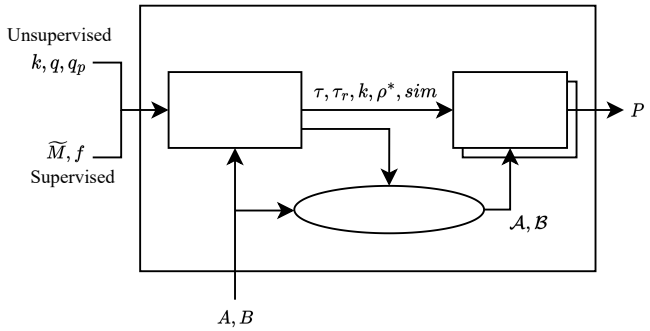


Figure 4.7: Illustration of the high-level conceptual flow of ShallowBlocker.

We are now ready to assemble everything and present our proposed method for blocking, ShallowBlocker. Fundamentally, our method performs one or more (τ, τ_r, k)-joins using `TTRKJoin`. Therefore, the focus in this section is mainly how ShallowBlocker chooses hyperparameters and applies `TTRKJoin`. Figure 4.7 illustrates the high-level flow consisting of two main steps. First, we perform hyperparameter selection — which not only includes hyperparameters to `TTRKJoin` and choice of set similarity measure, but also the token set model we use to go from strings to weighted token sets. Then, secondly, we run one or more configurations of `TTRKJoin`.

Note that we intentionally present the method in this fashion to highlight a key characteristic, namely its interpretability. At its core, the proposed method is a string similarity join with clearly defined conditions dictating what pairs to produce, and these join conditions can be inspected and interpreted to understand how the end result is produced. Specifically, since ShallowBlocker runs `TTRKJoin` it will always perform exact or approximated (τ, τ_r, k)-joins. In cases where ρ^* is finite we can infer some q and q_p for which it is true that the performed join is a (q, q_p)-approximate (τ, τ_r, k)-join (see Section 4.8).

We perform hyperparameter selection differently depending on whether examples of true matches are available or not. So we describe the unsupervised and supervised approach separately for ease of presentation.

4.9.1 Unsupervised Blocking

Algorithm 11: ShallowBlocker(A, B, k, q), Unsupervised

Input: A and B are collections of strings. k bounds to maximum number of pairs to $k \cdot \min(|\mathcal{A}|, |\mathcal{B}|)$, and q is the quality of underlying joins.
Output: Pairs $C \subseteq A \times B$ such that $|P| \leq k \cdot \min(|A|, |B|)$.

```
if |A| > |B| then
    swap(A, B);                                  // Also flip pairs in resulting C
k_ba ← ⌊ k|A| / 2|B| ⌋;
k_ab ← ⌊ (k|A| − k_ba|B|) / |A| ⌋ ;
Precompute TF-IDF weights and O for A and B using all tokenizers;
P_ab ← BalancedTTRKJoin(A, B, k_ab, q);
P_ba ← BalancedTTRKJoin(B, A, k_ba, q);
P ← P_ab ∪ flipPairs(P_ba);
return P;
```

Algorithm 11 outlines the unsupervised approach. It accepts two hyperparameters:

- k: Bounds the number of pairs to $|P| \leq k \cdot \min(|A|, |B|)$. This lets the user adjust trade-off between recall and number of returned pairs. A reasonable default is 10.

- q: The approximation quality of the joins performed (with q_p probability). This lets the user adjust trade-off between precision and runtime. This is a valuable parameter for quick iterations early in an entity matching development workflow. A reasonable default is 1 (no approximation).

While it is possible to construct unsupervised methods without any user-provided parameters [97], we find that unpractical in real-world use cases. No method is perfect on all datasets for all use cases and it is in practice necessary to be able to adjust the precision-recall and/or precision-runtime tradeoff. Existing blocking methods vary in the parameters they expose to let the user adjust this. We argue, based on our own experience, that a user-friendly way to adjust these

trade-offs is to control the pair budget and some level of approximation. Our reasoning is twofold: 1) The parameters are not very sensitive to small changes[12]. 2) The parameters are not very sensitive to dataset-specific characteristics, and therefore it is possible to provide reasonable defaults.

Since (τ, τ_r, k)-joins are directional it matters which way to perform `TTRKJoin`. For robust results we perform one join in each direction — denoted ab when indexing B and querying A and ba for the other way around. We divide the pair budget between them by fixing each joins k parameter, which guarantees we will not exceed the total budget. The budget is divided as even as possible (see line 1-4).

Algorithm 12: BalancedTTRKJoin(A, B, k, q)

```
   Constants: N_A = 1000; k_dp = 10; q_p = 0.95;
 1 Ã ← sample max N_A strings from A ;
 2 dp* ← −1;
 3 Ψ* ← ∅;
 4 foreach tokenizer tok do
 5     Ã ← ToTokenSets(Ã, tok, TF-IDF);
 6     B ← ToTokenSets(B, tok, TF-IDF);
 7     foreach similarity measure sim do
 8         I± ← BuildTTRKIndex(B, 0; sim);
 9         Ψ ← RecordTrajectories(I±, Ã, ∞);
10         if q = 1 then
11             ρ* ← ∞;
12         else
13             ρ* ← QualityToRank(q, q_p, 0, 0, k_dp, Ψ);
14         P̃ ← TTRKJoin(Ã, B, 0, 0, k_dp, ρ*; sim) using I±;
15         dp ← DiscriminatePower(P̃);
16         if dp > dp* then
17             dp* ← dp;
18             Ψ* ← Ψ;
19 τ ← max{τ | EstimateNumPairs(τ, 0, ∞, Ψ*) ≥ k|A|} ;
20 τ_r ← max{τ_r | EstimateNumPairs(0, τ_r, ∞, Ψ*) ≥ k|A|} ;
21 ρ* ← QualityToRank(q, τ, τ_r, k, Ψ*);
22 P ← TTRKJoin(A, B, τ, τ_r, k, ρ*);
23 return P;
```

For both invocations of `TTRKJoin` we need to determine the tokenizer, token weighting scheme, the set similarity measure, and join parameters τ and τ_r.

[12]Contrast this to similarity threshold parameters, for which it is easy to fall into the failure modes of producing almost no pairs or way too many.

Algorithm 12 outlines the proposed method for doing so.

Choosing Token Set Model and Similarity Measure

To determine the token set model and set similarity measure we pick the combination we heuristically find to be the most discriminate (see line 1-18). We define the measure for discriminatory power of a token set model and similarity measure sim on dataset (A, B) as

$$dp = 1 - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \left[\frac{1}{k_{dp} - 1} \sum_{b \in \mathcal{B}_a} \left[\frac{sim(a, b)}{\max_{b' \in \mathcal{B}} sim(a, b')} \right] - 1 \right]$$

where $\mathcal{A}$ and $\mathcal{B}$ is A and B converted to token sets and $\mathcal{B}_a$ is the k_{dp} most similar token sets to a. We can run `TTRKJoin` with $\tau = \tau_r = 0$ and $k = k_{dp}$ to get the necessary pairs. To avoid a costly evaluation we estimate the discriminatory power by only using a subset $\widetilde{A}$ of A and with the desired approximation quality. Table 4.4 shows the six configurations that are evaluated. We consider these to be robust options for most datasets. Note that Dice similarity measure is left out because it will, per definition, have the same discriminatory power as Jaccard.

Token Set Model				
Tokenization		Token Weight		Set Similarity Measure
3-gram*	×	TF-IDF	×	Jaccard
word				Cosine
				Overlap

Table 4.4: Configuration space for unsupervised ShallowBlocker. *Only if the total number of characters in the dataset are below 100 000 000.

Choosing Hybrid Join Conditions

We argued in Section 4.5 that there are strong potential benefits of mixing different types of join conditions. In an unsupervised setting we have limited signals to guide how we determine such conditions. We argue that a reasonable approach is to balance the pruning power of the absolute similarity threshold, relative similarity threshold, and local cardinality threshold in a (τ, τ_r, k)-join so that each condition would in isolation produce the same number of pairs. That way we dynamically adapt to data and we get to exploit the complementary

strengths of the three join types. The combined conditions will consistently filter out pairs of low absolute similarity, relative similarity, and similarity order.

Specifically, we determine τ, τ_r, and k that would hit the pair budget ($k|A|$) if used in isolation. Obviously, this is per definition the case for the provided k. For τ and τ_r, we binary search for the highest value that produce at least $k|A|$ estimated number of pairs according to a sample of recorded trajectories as described in Section 35 (see line 19 and 20). Note that the k parameter of the `TTRKJoin` will still guarantee that we stay within the pair budget no matter what τ and τ_r is set to.

4.9.2 Supervised

In a supervised setting we assume a subset of known matches $\widetilde{M} \subseteq M$. This enables us to estimate the recall of a solution. Furthermore, using the proposed techniques in Section 4.7 we can quickly estimate the recall, an upper bound of $|P|$, and an upper bound of the runtime for any run of `TTRKJoin`. Our proposed method exploits this to tune the parameters of `TTRKJoin` to optimize a user-defined objective function f based on recall, $|P|$, and runtime. The only requirement of f is that its codomain is totally ordered. This is powerful because it lets users define their desired trade-off of blocking behaviour expressed explicitly in terms of these three main ways in which we judge blocking methods.

Algorithm 13 describes our proposed supervised version of ShallowBlocker — which we will denote AutoShallowBlocker when we want to explicitly distinguish it from the unsupervised version. The goal is to do two invocations of `TTRKJoin`, one in each direction. We first acquire the recall conditions and some randomly sampled search trajectories for each configuration of token set model and set similarity measures (see Table 4.5) in both directions (see line 2-15). Then we jointly optimize the parameters for both joins (line 17) before we execute them (line 18-24). Note that since ρ^* are among the parameters we optimize we also let ShallowBlocker choose the degree of approximation. All details except the optimization routine have already been covered.

Optimization

The optimization problem has four categorical and eight continuous variables in total for the two joins. We evaluate a solution by estimating recall, $|P|$, and runtime in both directions with the corresponding recall conditions and search trajectories, combining them, and feeding them to the objective function f. Since the objective function f is user-defined and only required to produce

Algorithm 13: ShallowBlocker($A, B, \widetilde{M}, f$), Supervised

Input: A and B are collections of strings. $\widetilde{M} \subseteq M$ are known matches and f is an objective function that accepts recall, $|P|$, and runtime and returns an a value from a totally ordered set.

Output: Pairs $P \subseteq A \times B$ produced such that $f(\text{recall}, |P|, \text{runtime})$ is high by best effort.

Constants: $N = 500; q_p = 0.95$; $D = \{0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$;

Precompute TF-IDF weights and $\mathcal{O}$ for A and B using all tokenizers;
$\Psi_{ab}, \Psi_{ba} \leftarrow$ empty maps from $(\texttt{tms}, sim)$ to sampled TTRK trajectories ;
$\Theta_{ab}, \Theta_{ba} \leftarrow$ empty maps from $(\texttt{tms}, sim)$ to recall conditions;
foreach *token set model* `tsm` **do**
  $\mathcal{A} \leftarrow \texttt{ToTokenSets}(A, \texttt{tms})$;
  $\mathcal{B} \leftarrow \texttt{ToTokenSets}(B, \texttt{tms})$;
  $\widetilde{\mathcal{A}} \leftarrow$ sample N sets from $\mathcal{A}$;
  $\widetilde{\mathcal{B}} \leftarrow$ sample N sets from $\mathcal{B}$;
  foreach *similarity measure sim* **do**
    $I^{\pm} \leftarrow \texttt{BuildTTRKIndex}(\mathcal{B}, 0; sim)$;
    $\Psi_{ab}[\texttt{tsm}, sim] \leftarrow \texttt{RecordTrajectories}(I^{\pm}, \widetilde{\mathcal{A}}; sim)$;
    $\Theta_{ab}[\texttt{tsm}, sim] \leftarrow \texttt{FindRecallCond}(\mathcal{A}, I^{\pm}, \widetilde{M}, D; sim)$;
    $I^{\pm} \leftarrow \texttt{BuildTTRKIndex}(\mathcal{A}, 0; \texttt{sim})$;
    $\Psi_{ba}[\texttt{tsm}, sim] \leftarrow \texttt{RecordTrajectories}(I^{\pm}, \widetilde{\mathcal{B}}; sim)$;
    $\Theta_{ba}[\texttt{tsm}, sim] \leftarrow \texttt{FindRecallCond}(\mathcal{B}, I^{\pm}, \widetilde{M}, D; sim)$;
$\texttt{tsm}_{ab}, sim_{ab}, \tau_{ab}, \tau_{r.ab}, k_{ab}, \rho^*_{ab},$
$\texttt{tsm}_{ba}, sim_{ba}, \tau_{ba}, \tau_{r.ba}, k_{ba}, \rho^*_{ba} \leftarrow \texttt{OptimizeTTRKJoins}(f, \Psi_{ab}, \Psi_{ba}, \Theta_{ab}, \Theta_{ba})$;
$\mathcal{A}_{ab} \leftarrow \texttt{ToTokenSets}(A, \texttt{tsm}_{ab})$;
$\mathcal{B}_{ab} \leftarrow \texttt{ToTokenSets}(B, \texttt{tsm}_{ab})$;
$P_{ab} \leftarrow \texttt{TTRKJoin}(\mathcal{A}_{ab}, \mathcal{B}_{ab}, \tau_{ab}, \tau_{r.ab}, k_{ab}, \rho^*_{ab}; sim_{ab})$;
$\mathcal{A}_{ba} \leftarrow \texttt{ToTokenSets}(A, \texttt{tsm}_{ba})$;
$\mathcal{B}_{ba} \leftarrow \texttt{ToTokenSets}(B, \texttt{tsm}_{ba})$;
$P_{ba} \leftarrow \texttt{TTRKJoin}(\mathcal{B}_{ba}, \mathcal{A}_{ba}, \tau_{ba}, \tau_{r.ba}, k_{ba}, \rho^*_{ba}; sim_{ba})$;
$P \leftarrow P_{ab} \cup \texttt{flipPairs}(P_{ba})$;
return P;

Token Set Model		Set Similarity Measure
Binary weighted 3-grams*		Jaccard
TF-IDF weighted 3-grams*		Dice
Binary weighted words	×	Cosine
TF-IDF weighted words		Overlap

Table 4.5: Configuration space for supervised ShallowBlocker. *Only if the total number of characters in the dataset are below 100 000 000.

values from a totally ordered set it is a black-box optimization problem. The number of variables is manageable and evaluation is quick. Furthermore, it is reasonable to assume, given the nature of the problem, that the objective function is not overly sensitive. Our experience is therefore that most random and local search heuristics are effective enough in finding good solutions.

We opt for a relatively simple approach. Draw ten random solutions, pick the best one, and then hill climb. Repeat 32 times (can be done in parallel) and pick the best solution. In hill climbing, try all exponentially spaced increments/decrements of the numeric parameters. It is important to note that k_{ab} or k_{ba} can be zero, so there might not be a `TTRKJoin` in both directions.

There is a considerable chance of overfitting on recall if the number of known matches $|\widetilde{M}|$ is not large enough. Therefore, we acquire the recall conditions with varying regularization parameter values D as described in Section 35. The regularization parameter $d \in D$ we end up using is picked through 3-fold cross validation of all values in D. Note that $|P|$ and runtime is not as prone to overfitting because we are in control of the number of samples those estimates are based on and can simply choose a large enough number of samples. While for recall we can not control the number of known matches.

4.9.3 Deduplication

So far we have only discussed blocking across two sets of records. A special case of this is deduplication, where we only have one set of records A and want to find all duplicates within A. To avoid unnecessary clutter we have not incorporated this within the presentation of our proposed techniques and method. However, they are all trivially adapted to deduplication. The main differences is that `TTRKJoin` must take care to not return self-pairs and that ShallowBlocker should only do `TTRKJoin` in one direction (from $\mathcal{A}$ to $\mathcal{A}$).

4.9.4 Constants

There are several constant parameters used throughout ShallowBlocker. This is not contradictory to the method being hands-off. For all practical purposes these are part of the method. They do not need tuning, are not user-provided, and are in essence no different from the network architecture and training details of a deep learning method or similar details from other methods. None of the constants are, by their nature, sensitive as long as they are large enough (e.g., sample sizes). So we have simply picked round numbers of reasonable sizes that we observe provide stable results.

4.10 Experimental Setup

4.10.1 Datasets

	# Records		# Matches		# Characters				# Words			
Dataset	$\|A\|$	$\|B\|$	Train	Test	Min	Median	Mean	Max	Min	Median	Mean	Max
Amazon-Google	1363	3226	699	234	10	58	60	222	2	9	9.6	33
Beer	4345	3000	40	100^{+}	25	68	69	153	5	11	11.3	29
DBLP-ACM*	2616	2294	1332	444	17	129	132	421	4	19	19.3	66
DBLP-GoogleScholar*	2616	64 263	3207	1070	7	110	111	344	2	17	16.8	58
iTunes-Amazon*	6907	55 923	78	100^{+}	54	141	152	606	10	24	25.8	88
Walmart-Amazon*	2554	22 074	576	193	19	97	102	913	3	16	17.1	172
Abt-Buy	1081	1092	616	206	13	173	193	555	2	28	30.7	88
Company	28 200	28 200	10 000	5640	0	4311	5177	29 035	0	640	792.4	2007
Songs	1 000 000		10 000	136 011	31	86	90	440	6	12	13.0	72
Citeseer-DBLP	1 823 978	2 512 927	10 000	548 787	1	122	126	2541	0	17	18.2	312

Table 4.6: The different datasets used in our experiments. Note that some datasets have been slightly adapted to fit our experimental setup. See Section 4.10.1. *These datasets has a "dirty" version [91]. $^{+}$We have labelled additional matches to get 100 test matches.

We use the DeepMatcher [91] and Falcon [28] datasets from the Magellan Data Repository [30]. Table 4.6 shows key characteristics for the different datasets. We ignore the known non-matches for all datasets since they are not used by any method in our experiments or relevant for measuring recall.

We use the provided train/test split for the DeepMatcher datasets. The validation sets are not used in the experiments. All supervised methods only get the training matches and must do validation within those. We omit the Fodors-Zagats dataset because it is tiny, trivial for all tested methods, and do not have enough matches (even if we label more) to reliably measure recall. For Beer and iTunes-Amazon we manually label additional matches to get 100 test matches.

The Songs and Citeseer-DBLP datasets from Falcon [28] do not have any train/test split. We therefore split of 10 000 matches to training and the rest to testing. Songs is a deduplication dataset — i.e., one set of records that is matched to itself. Importantly, we remove self-matches from the dataset since they are trivial.

4.10.2 Baseline Methods

		Default hyperparameter values		
		Recall target		
Method	Hyperparameters	0.90	0.98	Hyperparameter space
Token Blocking	-	-	-	-
HVTB	-	-	-	-
Minhash LSH	τ	0.08	0.04	$[0, 1]$
	Tokenization	Word		3-gram, word
	b	Optimize eq. false		$[1, \ldots, \text{numPerm}]$
	r	positive and negative		$[1, \ldots, \lfloor \text{numPerm}/b \rfloor]$
PPJoin	τ	0.20	0.14	$[0, \infty)$
	Similarity measure	Cosine		Jaccard, Cosine, Dice, Overlap
	Tokenization	Word		3-gram, word
τ-join	τ	0.22	0.16	$[0, \infty)$
	Similarity measure	Cosine		Jaccard, Cosine, Dice, Overlap
	Tokenization	Word		3-gram, word
	Token weighting	TF-IDF		Binary, TF-IDF
DeepBlocker	k	240	Not possible	$[1, \ldots, \max(\lvert A \rvert, \lvert B \rvert)]$
AutoBlock	k	-	-	$[1, \ldots, \max(\lvert A \rvert, \lvert B \rvert)]$
k-join	k	10	32	$[1, \ldots, \max(\lvert A \rvert, \lvert B \rvert)]$
	Similarity measure	Cosine		Jaccard, Cosine, Dice, Overlap
	Tokenization	Word		3-gram, word
	Token weighting	TF-IDF		Binary, TF-IDF

Table 4.7: Baseline methods and their hyperparameters. Also included is the default hyperparameter values when used in an unsupervised setting and the hyperparameter space.

We perform our experiments on a variety of baseline methods covering different method categories. See them listed in Table 4.7. First, we briefly introduce the different methods. Then we discuss how we determine hyperparameters in an unsupervised and supervised context. For all methods were it is relevant the largest string collection is indexed and the smallest queried (i.e., we ensure $|A| \leq |B|$).

Token Blocking

It is one of the most ubiquitous blocking methods and is considered a naive approach. Simply produce all pairs with at least one overlapping token. We use word tokenization and run it using `TTRKJoin` with $\tau = 1, \tau_r = 0, k = \infty, \rho^* = \infty$ and Overlap similarity.

HVTB [97]

A significant improvement over Token Blocking for most cases, and is reported to achieve, or be comparable to, state-of-the-art results. Instead of producing all pairs with at least one overlapping token, HVTB only considers non-unique tokens with above average TF-IDF score and thereby pruning away pairs that only have less impactful tokens in common. Furthermore, pairs with below average number of overlapping tokens are pruned. The original source code is not available so we implemented according the the authors specification. For a fair comparison, we also include our own optimized version: HVTB$^+$. The three main improvements are using integer representation of tokens, parallelization, and eagerly pruning pairs with one common token upon construction if overlaps greater than one have already been observed. It produces the same pairs but with greatly reduced runtime and memory consumption.

τ-join

Classical absolute threshold-based similarity join achieved by running `TTRKJoin` with $\tau_r = 0, k = \infty, \rho^* = \infty$ fixed.

k-join

Classical local cardinality threshold-based similarity join achieved by running `TTRKJoin` with $\tau = 0, \tau_r = 0, \rho^* = \infty$ fixed.

PPJoin [143, 144]

Popular state-of-the-art τ-join blocking method [100]. We use the highly optimized implementation from Mann et al. [87]. However, note that this implementation uses, and is optimized for, unweighted sets — as is common for PPJoin. We include this in our experiments because it is a widely used method and point of comparison.

Minhash LSH

Well-known technique using locality-sensitive hashing [46] for approximate similarity joins [18, 19]. We use the implementation from the datasketch[13] package. This is what the authors of AutoBlock [152] used in their experiments. For fair comparison we implement a parallelization layer on top to exploit all available CPU cores. We use the recommended default setting of 128 permutations.

DeepBlocker[127]

State-of-the-art unsupervised deep learning-based blocker. It avoids the need for labeled training data by using either a encoder-decoder setup which is trained for reconstruction or by generating synthetic labeled data. We use the authors provided implementation[14]. However, note that we implemented a GPU accelerated top-k Cosine similarity search using FAISS [61] as the described by the authors since the provided source code only includes a brute force CPU version. Based on the reported results and recommendation of the authors we use the Hybrid model for the "textual" datasets (Walmart-Amazon, Abt-Buy, and Company) and AutoEncoder for the rest.

AutoBlock[152]

Supervised deep learning-based blocker that embeds records and uses locality-sensitive hashing to find similar embeddings. Known labeled matches are used to train the deep network, but one must still specify the local cardinality threshold k. The original source code is not available so we implement it according to the authors description using PyTorch and FALCONN. We also parallalize the LSH nearest neighbor search for a fair comparison.

4.10.3 Unsupervised Baselines

All baseline methods except AutoBlock are unsupervised. However, most of them still have one or more hyperparameters that must be set manually. In an unsupervised setting we are not able to tune them per dataset and must rely on defaults. To give all methods an equal footing we chose the hyperparameters for each method that yields the best overall results across all datasets on the test sets. We find Cosine similarity measure, word tokenization, and TF-IDF token

[13] `https://github.com/ekzhu/datasketch`
[14] `https://github.com/qcri/DeepBlocker`

weighting to be optimal for methods that have any of those hyperparameters. For Minhash LSH we set b and r automatically to the configuration that balances false positives and negatives equally.

Most baseline methods have a similarity threshold τ or local cardinality threshold k as the main hyperparameter balancing recall against number of returned pairs. For the experiments that target a specific recall level, we set this parameter to the most aggressive (highest τ, lowest k) that achieves the recall target on all datasets[15] on the test set. For τ we only consider increments of 0.01. Table 4.7 shows the hyperparameters for recall targets of 0.90 and 0.98. It is important to note that this way of determining hyperparameters yields best case results for the baseline methods by exploiting knowledge about the test set and not incurring any runtime cost for picking hyperparameter values. Therefore, it is not strictly comparable to results from supervised methods.

4.10.4 Supervised Baselines

The Supervised version of ShallowBlocker (denoted AutoShallowBlocker) uses known matches $\widetilde{M}$ and an objective function f to set all parameters. The only supervised baseline method, AutoBlock, has a hyperparameter that must be set: k. Furthermore, most of the other unsupervised methods can be tuned. So in order to compare against the baseline methods fairly we extend them with a supervised approach similar to ShallowBlocker for determining their hyperparameters from the training set. We will denote these modified supervised baselines with a Sup- prefix (for supervised) — e.g., Sup-DeepBlocker.

Assume we are given training matches $\widetilde{M}$ and a recall target R. We can determine the hyperparameters by enumerating all combinations of categorical parameters, picking the threshold (τ or k) for each combination that achieves recall R on $\widetilde{M}$, and then picking the combination that will return the fewest pairs. In order to avoid overfitting, and thereby missing the recall target, we use regularization. $\widetilde{M}$ is split into 3-fold cross-validation splits. We determine the hyperparameters using the train split of each fold with some threshold regularization margin d. The regularization margin d is increased until the average recall of validation splits is at least R with 95% confidence (not adjusting for multiple testing). The final hyperparameters are picked using the entire $\widetilde{M}$ and the chosen d.

For the methods with a similarity threshold τ we can simply calculate the

[15] Except the Company dataset since it contains so many wrongly labelled matches that it is unrealistic to reach high recall.

similarity for all train matches and pick the R quantile of the similarities to determine the highest threshold that would recall R of the matches $\widetilde{M}$. We estimate the number of returned pairs by running the method on a sample of 1000 strings from A and the entire B and multiplying the number of pairs with $|A|/1000$. The margin d adjusts the threshold to $\tau \cdot (1 - d)$ and uses the same values as ShallowBlocker, For Minhash LSH we pick the b and r that minimizes probability of false positives while also reaching the recall target.

For the methods with a local cardinality threshold k we can determine the smallest threshold that would recall R of the training matches by doubling k and running the method on the subset of A that are in the training matches and the entire B until at least R of the matches are found. Then using the result to find the minimum between $k/2$ and k that still recall at least R of the matches. The configuration that return the fewest pairs are simply the one with the lowest k. The margin d adjusts the threshold to $k + d$ and is increased in powers of two.

Note that we make sure to avoid redundant work where possible for the different methods in this supervised approach. For example, we only train the deep learning model and compute the embeddings for each record once for DeepBlocker — not for every fold and value of k. Additionally, note that we do not do this for PPJoin since Sup-τ-join would be a superset of Sup-PPJoin.

4.10.5 Hardware

All CPU-only experiments are run on an AMD Ryzen 5950X system with 64GB RAM. Because of practical resource constraints all experiments with deep learning models using GPU are run on a different machine with a Nvidia V100 GPU and Intel Xeon Gold 6148 CPU with 128GB. While the systems are not equivalent, and therefore not directly comparable, there will always be a divide between the CPU-only and GPU accelerated methods since the latter exploits a fundamentally different compute resource. The GPU machine is the more capable and expensive, so at least the deep learning methods can not be said to be hampered by the lack of compute and memory.

Resource Setting	Method	Dataset AG	B	DA	DG	IA	WA	D-DA	D-DG	D-IA	D-WA	AB	C	S	CD
Recall															
Unsupervised	Token Blocking	1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.999	1.000	1.000	0.995	0.964	OOM	OOM
	HVTB	0.868	0.210	1.000	0.993	0.860	0.990	1.000	0.993	0.200	0.990	0.893	0.923	0.576	OOM
	HVTB+	0.868	0.210	1.000	0.993	0.860	0.990	1.000	0.993	0.200	0.990	0.893	0.923	0.576	0.998
	MinHash LSH	1.000	1.000	1.000	0.999	1.000	0.995	1.000	0.999	1.000	0.995	0.937	0.699	OOM	OOM
	PPJoin	0.991	1.000	1.000	0.996	1.000	1.000	1.000	0.996	1.000	1.000	0.917	OOM	OOM	OOM
	τ-join	0.983	1.000	1.000	0.993	1.000	1.000	1.000	0.993	1.000	1.000	0.903	0.455	0.998	1.000
	DeepBlocker	0.996	1.000	0.998	0.979	0.988	0.991	0.998	0.983	0.994	0.992	0.995	0.714	0.901	TIME
	k-join	0.974	1.000	1.000	0.968	1.000	0.984	1.000	0.968	1.000	0.984	0.966	0.863	0.906	0.996
	ShallowBlocker ($q=0.95$)	0.980	1.000	1.000	0.975	0.956	0.968	1.000	0.976	0.956	0.968	0.995	0.767	0.904	0.990
	ShallowBlocker ($q=0.99$)	0.981	1.000	1.000	0.974	0.970	0.964	1.000	0.976	0.970	0.964	0.995	0.666	0.906	0.989
	ShallowBlocker ($q=1$)	0.977	1.000	1.000	0.970	0.964	0.962	1.000	0.972	0.964	0.962	0.995	0.809	0.900	0.988
Supervised ≤ 100 matches	Sup-MinHash LSH	0.978	0.996	0.942	0.956	0.960	0.973	0.924	0.948	0.960	0.973	0.923	0.949	OOM	OOM
	Sup-τ-join	0.974	1.000	0.935	0.953	0.984	0.981	0.935	0.941	0.984	0.981	0.939	0.940	0.980	0.979
	Sup-AutoBlock	0.754	0.706	0.712	0.853	1.000	0.940	0.655	0.807	0.926	0.742	0.900	0.243	OOM	OOM
	Sup-DeepBlocker	0.956	1.000	0.984	0.954	0.990	0.981	0.984	0.963	0.986	0.983	0.976	ERR	ERR	0.936*
	Sup-k-join	0.989	1.000	0.993	0.976	0.996	0.980	0.991	0.980	0.994	0.967	0.933	0.962	0.985	0.977
	ShallowBlocker	0.992	1.000	0.940	0.964	0.986	0.985	0.920	0.950	0.988	0.988	0.938	0.924	0.978	0.967
Supervised ≤ 1000 matches	Sup-MinHash LSH	0.938	-	0.928	0.940	-	0.921	0.928	0.947	-	0.921	0.932	0.942	OOM	OOM
	Sup-τ-join	0.965	-	0.936	0.933	-	0.928	0.936	0.932	-	0.928	0.935	0.926	0.930	0.944
	Sup-AutoBlock	0.891	-	0.825	0.899	-	0.906	0.882	0.862	-	0.878	0.907	0.506	0.975*	OOM
	Sup-DeepBlocker	0.932	-	0.985	0.930	-	0.931	0.983	0.930	-	0.933	0.938	ERR	ERR	0.932*
	Sup-k-join	0.964	-	0.993	0.936	-	0.914	0.991	0.930	-	0.921	0.965	0.930	0.964	0.979
	ShallowBlocker	0.982	-	0.955	0.933	-	0.940	0.937	0.943	-	0.947	0.913	0.926	0.930	0.948
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	0.935	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	0.918	0.918	0.928
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	0.577	0.959*	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	0.931*	0.925
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	0.915	0.956	0.979
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	0.920	0.926	0.924
$\widetilde{k}$															
Unsupervised	Token Blocking	513	3685	1848	34879	51042	3987	1848	34526	51042	3987	426	27078	OOM	OOM
	HVTB	5.39	27.7	13.2	41.0	1384	41.2	13.2	41.8	278	41.2	9.00	**5887**	25.6	OOM
	HVTB+	5.39	27.7	13.2	41.0	1384	41.2	13.2	41.8	278	41.2	9.00	**5887**	25.6	**150**
	MinHash LSH	418	3078	997	18528	31742	2332	997	18292	31742	2332	178	13438	OOM	OOM
	PPJoin	45.8	2168	80.8	958	2431	107	80.8	912	2431	107	12.1	OOM	OOM	OOM
	τ-join	11.0	5.85	4.99	16.2	165	26.6	4.99	15.9	165	26.6	5.49	0.626	25.9	50.3
	DeepBlocker	240	240	240	240	240	240	240	240	240	240	240	240	184	TIME
	k-join	9.99	9.99	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	6.61	10.00
	ShallowBlocker ($q=0.95$)	4.92	3.80	4.87	6.23	**2.49**	**4.21**	4.87	6.48	**2.49**	**4.21**	6.12	2.80	6.92	2.32
	ShallowBlocker ($q=0.99$)	4.07	3.75	**4.22**	5.30	2.95	4.98	**4.22**	5.43	2.95	4.98	5.09	1.90	5.71	**2.02**
	ShallowBlocker ($q=1$)	**3.54**	**3.46**	**4.12**	**4.90**	2.73	4.72	**4.12**	**5.12**	2.73	4.72	**4.59**	2.57	**5.35**	**1.92**
Supervised ≤ 100 matches	Sup-MinHash LSH	72.8	1535	1.48	286	3053	570	1.85	440	3053	570	165	25952	OOM	OOM
	Sup-τ-join	6.85	21.6	1.01	3.81	15.4	10.3	1.01	3.59	15.4	10.3	7.29	10337	17.8	0.927
	Sup-AutoBlock	595	2.90	6.61	4763	39168	1660	10.7	27626	24459	4555	527	444	OOM	OOM
	Sup-DeepBlocker	61.6	106	1.000	252	202	120	1.000	134	240	133	126	ERR	ERR	7.00*
	Sup-k-join	10.6	56.9	1.000	11.4	57.0	**7.40**	1.000	12.2	58.6	**4.40**	**1.80**	25580	27.1	2.00
	ShallowBlocker	**4.73**	**4.41**	**0.951**	**2.68**	**5.69**	12.8	**0.923**	**2.56**	**6.35**	11.8	1.91	**424**	**10.4**	**0.721**
Supervised ≤ 1000 matches	Sup-MinHash LSH	43.7	-	1.47	173	-	194	1.97	250	-	194	175	25325	OOM	OOM
	Sup-τ-join	5.11	-	1.00	2.86	-	2.88	1.00	2.84	-	**2.88**	6.50	1198	8.68	0.532
	Sup-AutoBlock	532	-	31.0	914	-	353	23.5	17682	-	8252	222	593	99.1*	OOM
	Sup-DeepBlocker	22.0	-	**1.000**	15.0	-	15.6	1.000	15.4	-	16.0	49.2	ERR	ERR	2.75*
	Sup-k-join	5.20	-	**1.000**	6.40	-	2.60	1.000	6.20	-	**2.80**	3.40	1725	11.2	2.00
	ShallowBlocker	**2.85**	-	**0.961**	**2.28**	-	**2.29**	**0.937**	**2.32**	-	**2.82**	**1.16**	**237**	**5.93**	**0.459**
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	24796	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	620	7.69	0.470
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	505	77.7*	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	471*	2.00
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	97.9	9.46	2.00
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	**70.1**	**6.10**	**0.389**
Runtime															
Unsupervised	Token Blocking	26ms	191ms	96ms	1.65s	6.31s	214ms	95ms	1.62s	6.29s	219ms	24ms	1m21s	OOM	OOM
	HVTB	44ms	49ms	66ms	748ms	5.11s	243ms	71ms	757ms	1.23s	244ms	31ms	18m17s	5m41s	OOM
	HVTB+	33ms	24ms	25ms	168ms	356ms	70ms	26ms	**150ms**	134ms	70ms	14ms	**28.19s**	10.83s	1m10s
	MinHash LSH	380ms	982ms	447ms	8.53s	19.95s	2.05s	439ms	8.44s	19.73s	2.04s	240ms	32.47s	OOM	OOM
	PPJoin	60ms	5.00s	210ms	2.85s	16.81s	360ms	200ms	2.72s	16.80s	350ms	30ms	OOM	OOM	OOM
	τ-join	17ms	**20ms**	**20ms**	**158ms**	213ms	**65ms**	**23ms**	159ms	220ms	73ms	19ms	7.36s	13.41s	4m49s
	DeepBlocker	47.66s	1m5s	47.94s	9m44s	8m58s	3m49s	51.67s	9m4s	8m53s	53m59s	5m30s	4h52m	44m6s	TIME
	k-join	**15ms**	**21ms**	25ms	**155ms**	**140ms**	69ms	25ms	**154ms**	**145ms**	**64ms**	**17ms**	18.21s	4.30s	3m37s
	ShallowBlocker ($q=0.95$)	760ms	866ms	1.01s	2.00s	2.11s	1.06s	1.02s	1.99s	2.17s	1.07s	921ms	11.07s	**3.18s**	**45.69s**
	ShallowBlocker ($q=0.99$)	760ms	874ms	1.02s	2.11s	1.92s	803ms	1.02s	2.11s	1.99s	813ms	919ms	15.25s	**3.33s**	2m26s
	ShallowBlocker ($q=1$)	392ms	447ms	682ms	2.85s	2.34s	688ms	676ms	2.83s	2.46s	686ms	493ms	29.01s	3.43s	4m14s
Supervised ≤ 100 matches	Sup-MinHash LSH	11.60s	22.58s	13.06s	40.20s	36.74s	25.54s	13.12s	34.38s	38.22s	25.86s	6.83s	6m54s	OOM	OOM
	Sup-τ-join	1.46s	3.70s	1.41s	**7.07s**	**8.63s**	4.87s	1.46s	**6.59s**	**9.32s**	4.91s	2.16s	6m8s	1m19s	2m17s
	Sup-AutoBlock	2m49s	3m52s	6m2s	5m36s	26m25s	4m37s	3m43s	6m42s	14m45s	3m54s	5m37s	48m40s	OOM	OOM
	Sup-DeepBlocker	1m0s	1m21s	52.51s	8m57s	8m46s	3m26s	49.03s	9m2s	8m54s	55m15s	5m15s	ERR	ERR	1h40m*
	Sup-k-join	**755ms**	**854ms**	**419ms**	8.14s	9.76s	**2.00s**	**430ms**	8.46s	10.57s	**2.00s**	**652ms**	2m29s	42.51s	13m31s
	ShallowBlocker	2.93s	3.24s	2.00s	7.97s	10.53s	4.14s	2.00s	8.05s	10.90s	4.16s	3.18s	**45.76s**	**13.46s**	**23.96s**
Supervised ≤ 1000 matches	Sup-MinHash LSH	7.56s	-	11.71s	35.53s	-	16.37s	11.47s	37.97s	-	16.73s	7.74s	5m1s	OOM	OOM
	Sup-τ-join	1.33s	-	1.28s	**5.56s**	-	3.86s	1.33s	**5.64s**	-	3.90s	2.11s	5m29s	52.66s	1m41s
	Sup-AutoBlock	2m19s	-	5m53s	6m2s	-	4m35s	3m56s	9m33s	-	5m16s	3m49s	48m21s	2h51m*	OOM
	Sup-DeepBlocker	52.01s	-	53.83s	9m5s	-	3m22s	50.37s	9m0s	-	56m5s	5m16s	ERR	ERR	1h38m*
	Sup-k-join	**821ms**	-	**480ms**	22.17s	-	**2.58s**	**488ms**	22.63s	-	**2.68s**	**1.04s**	6m46s	56.57s	11m43s
	ShallowBlocker	4.08s	-	3.58s	12.22s	-	5.36s	3.62s	12.24s	-	5.36s	4.38s	**1m13s**	**9.60s**	**27.56s**
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	**4m19s**	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	4m36s	39.86s	1m25s
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	59m39s	2h22m*	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	49m31s*	1h41m
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	55m57s	41.48s	12m27s
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	4m53s	**16.99s**	**1m7s**

Table 4.8: Performance of all methods across all datasets in both unsupervised and different supervised settings when the recall target is 90%. See Table 4.10 for memory usage. The highest performing method for each dataset and resource setting according to each measure is underlined, while methods that are within 5% of this is bold. Methods that failed to reach to recall target is grayed out. OOM: Out of memory. ERR: Hitting limits of FAISS. TIME: Exceeding 24h time limit. *One or more runs crashed.

Resource Setting	Method	Dataset													
		AG	B	DA	DG	IA	WA	D-DA	D-DG	D-IA	D-WA	AB	C	S	CD
							Recall								
Unsupervised	Token Blocking	1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.999	1.000	1.000	0.995	0.964	OOM	OOM
	HVTB	0.868	0.210	1.000	0.993	0.860	0.990	1.000	0.993	0.200	0.990	0.893	0.923	0.576	OOM
	HVTB+	0.868	0.210	1.000	0.993	0.860	0.990	1.000	0.993	0.200	0.990	0.893	0.923	0.576	**0.998**
	MinHash LSH	1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.999	1.000	1.000	0.985	0.862	OOM	OOM
	PPJoin	1.000	1.000	1.000	0.999	OOM	1.000	1.000	0.999	OOM	1.000	0.981	OOM	OOM	OOM
	τ-join	0.987	1.000	1.000	0.997	1.000	1.000	1.000	0.997	1.000	1.000	0.981	0.660	0.999	1.000
	DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	k-join	0.991	1.000	1.000	0.993	1.000	0.990	1.000	0.993	1.000	0.990	0.990	0.892	0.980	0.998
	ShallowBlocker (q=0.95)	0.996	1.000	1.000	0.990	0.998	1.000	1.000	0.989	0.998	1.000	1.000	0.865	0.980	0.997
	ShallowBlocker (q=0.99)	0.987	1.000	1.000	0.988	0.980	0.992	1.000	0.988	0.980	0.992	1.000	0.796	0.981	0.995
	ShallowBlocker (q=1)	0.987	1.000	1.000	0.987	0.980	0.994	1.000	0.988	0.980	0.994	0.995	0.853	0.980	0.995
Supervised ≤ 100 matches	Sup-MinHash LSH	0.997	1.000	0.994	0.991	0.992	0.995	0.994	0.999	0.992	0.995	0.969	0.949	OOM	OOM
	Sup-τ-join	0.987	1.000	0.998	0.989	0.980	1.000	0.998	0.991	0.980	1.000	0.971	0.963	0.999	0.993
	Sup-AutoBlock	0.830	0.792	0.742	0.925	1.000	0.952	0.701	0.809	0.950	0.802	0.932	0.243	OOM	OOM
	Sup-DeepBlocker	0.998	ERR	0.995	0.990*	1.000	1.000*	0.996	0.989*	1.000	1.000*	1.000	ERR	ERR	0.975
	Sup-k-join	1.000	1.000	1.000	0.995	1.000	1.000	1.000	0.995	1.000	1.000	0.961	0.964	0.998	0.992
	ShallowBlocker	0.995	1.000	0.980	0.989	0.994	0.993	0.987	0.989	0.992	0.992	0.972	0.937	0.995	0.974
Supervised ≤ 1000 matches	Sup-MinHash LSH	0.993	-	0.989	0.991	-	0.991	0.992	0.990	-	0.991	0.983	0.949	OOM	OOM
	Sup-τ-join	0.987	-	0.993	0.989	-	0.992	0.993	0.989	-	0.992	0.989	0.963	0.995	0.995
	Sup-AutoBlock	0.924	-	0.898	0.953	-	0.951	0.964	0.869	-	0.885	0.928	0.506	OOM	OOM
	Sup-DeepBlocker	0.997	-	0.994	0.992	-	0.982	0.992	0.990*	-	0.988	0.994	ERR	ERR	ERR
	Sup-k-join	1.000	-	0.993	0.995	-	0.979	0.991	0.996	-	0.991	0.979	0.964	0.997	0.992
	ShallowBlocker	0.997	-	0.989	0.992	-	0.992	0.991	0.992	-	0.993	0.987	0.962	0.992	0.990
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	0.949	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	0.963	0.986	0.986
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	0.577	OOM	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	ERR	**0.982***
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	0.964	0.992	0.994
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	0.964	0.990	0.988
							$\widetilde{k}$								
Unsupervised	Token Blocking	513	3685	1848	34879	51042	3987	1848	34526	51042	3987	426	27078	OOM	OOM
	HVTB	5.39	27.7	13.2	41.0	1384	41.2	13.2	41.8	278	41.2	9.00	5887	25.6	OOM
	HVTB+	5.39	27.7	13.2	41.0	1384	41.2	13.2	41.8	278	41.2	9.00	5887	25.6	150
	MinHash LSH	487	3587	1381	27536	41566	3322	1381	27160	41566	3322	276	19166	OOM	OOM
	PPJoin	124	2963	310	4580	OOM	287	310	4422	OOM	287	26.7	OOM	OOM	OOM
	τ-join	22.0	13.3	**10.3**	54.0	337	60.4	**10.3**	52.6	337	60.4	**10.9**	1.49	**40.4**	**210**
	DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	k-join	31.8	31.9	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	32.0	21.2	32.0
	ShallowBlocker (q=0.95)	16.2	16.3	16.1	11.4	9.84	16.3	16.1	11.8	9.84	16.3	21.2	8.81	21.4	7.38
	ShallowBlocker (q=0.99)	**10.6**	9.65	10.8	11.3	**5.71**	10.5	10.8	11.7	**5.71**	10.5	14.6	5.60	16.9	**5.24**
	ShallowBlocker (q=1)	**10.2**	**8.96**	**9.92**	**10.7**	**5.58**	**9.87**	**9.92**	**11.0**	**5.58**	**9.87**	13.6	6.11	**15.4**	**4.90**
Supervised ≤ 100 matches	Sup-MinHash LSH	351	2789	23.1	2427	10017	2272	23.1	10091	10017	2332	239	25952	OOM	OOM
	Sup-τ-join	21.2	1524	1.82	8.42	17.0	54.8	1.82	8.82	17.0	54.8	8.82	27056	27.4	**1.69**
	Sup-AutoBlock	1074	24.6	10.9	23984	39168	2017	17.3	27912	28541	7081	648	444	OOM	OOM
	Sup-DeepBlocker	1041	ERR	5.40	1246*	1314	940*	5.80	649*	903	951*	**552**	ERR	ERR	111
	Sup-k-join	63.8	**185**	17.0	33.6	112	45.6	17.0	41.0	112	58.4	3.00	27078	**56.4**	**3.00**
	ShallowBlocker	**11.4**	**25.4**	**1.02**	**6.76**	**15.4**	**23.8**	**1.09**	**6.67**	**11.5**	**24.0**	2.37	11870	**19.6**	0.999
Supervised ≤ 1000 matches	Sup-MinHash LSH	390	-	6.95	2213	-	1294	7.34	2111	-	1588	274	25952	OOM	OOM
	Sup-τ-join	23.9	-	1.29	12.9	-	22.4	1.29	12.7	-	22.4	32.3	27056	22.8	2.01
	Sup-AutoBlock	1186	-	62.3	21349	-	2275	330	19356	-	8527	415	593	OOM	OOM
	Sup-DeepBlocker	700	-	2.40	1194	-	149	2.40	947*	-	199	**275**	ERR	ERR	ERR
	Sup-k-join	22.8	-	**1.000**	19.4	-	7.40	**1.000**	17.8	-	**10.4**	5.60	27078	**53.3**	3.00
	ShallowBlocker	**9.01**	-	**1.01**	**4.66**	-	**16.5**	**1.01**	**5.02**	-	17.8	**3.25**	25303	**16.8**	**0.805**
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	25952	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	27056	**18.6**	**1.10**
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	505	OOM	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	ERR	**17.8***
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	27078	35.1	4.00
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	26921	**14.4**	**0.782**
							Runtime								
Unsupervised	Token Blocking	26ms	191ms	96ms	1.65s	6.31s	214ms	95ms	1.62s	6.29s	219ms	24ms	1m21s	OOM	OOM
	HVTB	44ms	49ms	66ms	748ms	5.11s	243ms	71ms	757ms	1.23s	244ms	31ms	18m17s	5m41s	OOM
	HVTB+	33ms	24ms	25ms	168ms	356ms	**70ms**	26ms	**150ms**	134ms	**70ms**	14ms	28.19s	10.83s	**1m10s**
	MinHash LSH	731ms	1.58s	835ms	16.09s	29.71s	3.79s	820ms	15.94s	29.83s	3.77s	294ms	48.91s	OOM	OOM
	PPJoin	150ms	6.78s	600ms	10.04s	OOM	710ms	600ms	9.69s	OOM	710ms	40ms	OOM	OOM	OOM
	τ-join	18ms	**18ms**	24ms	**150ms**	299ms	74ms	**23ms**	**158ms**	311ms	**73ms**	19ms	10.71s	24.59s	8m48s
	DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	k-join	**16ms**	22ms	**23ms**	161ms	**168ms**	**71ms**	25ms	159ms	**173ms**	**73ms**	**17ms**	21.45s	10.29s	5m34s
	ShallowBlocker (q=0.95)	789ms	909ms	1.08s	2.54s	2.66s	1.26s	1.07s	2.55s	2.73s	1.25s	944ms	13.26s	**4.28s**	2m18s
	ShallowBlocker (q=0.99)	786ms	912ms	1.07s	2.41s	2.64s	1.23s	1.06s	2.41s	2.71s	1.24s	952ms	19.76s	**4.43s**	4m20s
	ShallowBlocker (q=1)	413ms	487ms	716ms	3.04s	3.01s	937ms	703ms	3.03s	3.15s	949ms	521ms	33.77s	6.20s	6m43s
Supervised ≤ 100 matches	Sup-MinHash LSH	16.02s	23.96s	19.11s	42.57s	51.04s	23.71s	19.21s	45.05s	53.67s	24.28s	7.44s	7m30s	OOM	OOM
	Sup-τ-join	1.75s	5.55s	2.64s	9.91s	**9.16s**	5.79s	2.82s	9.86s	**9.54s**	5.80s	2.35s	2m18s	1m39s	**2m40s**
	Sup-AutoBlock	2m52s	4m35s	6m16s	6m55s	**26m17s**	4m51s	3m57s	6m47s	15m22s	4m0s	5m37s	48m26s	OOM	OOM
	Sup-DeepBlocker	1m5s	ERR	51.13s	9m13s*	9m1s	3m35s*	50.57s	9m1s*	8m51s	55m42s*	**5m20s**	ERR	ERR	1h40m
	Sup-k-join	**1.04s**	**1.64s**	**574ms**	10.12s	13.31s	**2.81s**	**592ms**	10.40s	14.31s	**2.87s**	849ms	2m38s	1m11s	14m6s
	ShallowBlocker	2.99s	3.17s	2.00s	**8.03s**	10.75s	4.27s	2.00s	**8.07s**	11.14s	4.28s	3.38s	1m17s	**12.41s**	25.02s
Supervised ≤ 1000 matches	Sup-MinHash LSH	16.36s	-	13.71s	40.27s	-	34.43s	13.46s	38.18s	-	30.76s	9.76s	7m31s	OOM	OOM
	Sup-τ-join	1.74s	-	2.20s	**8.09s**	-	**5.32s**	2.31s	**8.34s**	-	5.44s	**2.97s**	2m18s	1m25s	2m39s
	Sup-AutoBlock	3m4s	-	6m25s	11m19s	-	5m56s	5m4s	9m54s	-	5m24s	4m3s	47m41s	OOM	OOM
	Sup-DeepBlocker	1m3s	-	50.17s	9m8s	-	3m44s	50.16s	9m7s*	-	54m6s	5m36s	ERR	ERR	ERR
	Sup-k-join	**1.37s**	-	**472ms**	35.64s	-	3.99s	**482ms**	36.81s	-	**4.47s**	1.59s	10m23s	1m23s	13m53s
	ShallowBlocker	4.52s	-	3.99s	13.82s	-	5.90s	4.00s	13.92s	-	5.88s	4.89s	2m0s	**17.93s**	**38.05s**
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	7m48s	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	2m20s	50.69s	1m35s
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	59m30s	OOM	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	ERR	**1h36m***
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	1h31m	**2m14s**	16m30s
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	5m51s	**32.85s**	**1m16s**

Table 4.9: Performance of all methods across all datasets in both unsupervised and different supervised settings when the recall target is 98%. See Table 4.11 for memory usage. The highest performing method for each dataset and resource setting according to each measure is underlined, while methods that are within 5% of this is bold. Methods that failed to reach to recall target is grayed out. OOM: Out of memory. ERR: Hitting limits of FAISS. *One or more runs crashed.

Resource Setting	Method	Dataset													
		AG	B	DA	DG	IA	WA	D-DA	D-DG	D-IA	D-WA	AB	C	S	CD
Memory Usage (GB)															
Unsupervised	Token Blocking	0.14	0.32	0.21	1.85	6.04	0.40	0.21	1.84	6.04	0.38	0.13	18.7	OOM	OOM
	HVTB	0.12	0.14	0.12	0.28	0.98	0.17	0.13	0.28	0.39	0.17	0.12	8.00	38.3	OOM
	HVTB+	0.11	0.13	0.12	0.24	0.44	0.15	0.12	0.23	0.22	0.15	0.11	**5.10**	2.23	9.30
	MinHash LSH	0.17	0.45	0.21	1.62	4.69	0.43	0.21	1.60	4.72	0.43	0.15	10.1	OOM	OOM
	PPJoin	**0.01**	**0.07**	**0.01**	**0.05**	0.29	**0.01**	**0.01**	**0.05**	0.29	**0.01**	**0.01**	OOM	OOM	OOM
	τ-join	0.12	0.12	0.12	0.29	0.24	0.19	0.12	0.29	0.24	0.19	0.12	1.86	2.16	5.77
	DeepBlocker	13.1	13.2	13.2	13.4	13.6	13.3	13.2	13.4	13.6	14.8	13.2	18.7	39.9	TIME
	k-join	0.12	0.12	0.12	0.29	**0.21**	0.18	0.12	0.29	**0.21**	0.18	0.12	1.87	**1.34**	**4.05**
	ShallowBlocker ($q=0.95$)	0.41	0.43	0.47	0.76	0.76	0.55	0.47	0.76	0.76	0.55	0.42	2.14	**1.38**	**3.88**
	ShallowBlocker ($q=0.99$)	0.41	0.43	0.47	0.76	0.77	0.56	0.47	0.76	0.77	0.56	0.42	2.14	**1.37**	**3.89**
	ShallowBlocker ($q=1$)	0.41	0.43	0.47	0.77	0.78	0.56	0.47	0.77	0.78	0.56	0.42	2.12	**1.35**	**3.89**
Supervised ≤ 100 matches	Sup-MinHash LSH	0.21	0.37	0.22	1.21	1.08	0.56	0.22	1.01	1.06	0.56	0.21	**18.7**	OOM	OOM
	Sup-τ-join	**0.17**	0.30	**0.17**	**0.36**	**0.24**	**0.26**	**0.18**	**0.36**	**0.25**	**0.26**	**0.18**	9.16	2.81	**4.03**
	Sup-AutoBlock	11.1	10.9	10.9	13.5	66.2	12.0	10.9	23.9	**38.1**	13.3	**11.1**	15.6	OOM	OOM
	Sup-DeepBlocker	**13.0**	**13.1**	**13.1**	**13.4**	**13.5**	**13.2**	**13.1**	**13.4**	**13.6**	**14.8**	**13.1**	ERR	ERR	**45.1***
	Sup-k-join	**0.18**	**0.18**	0.19	1.05	0.92	0.45	0.19	1.12	0.91	0.45	**0.17**	18.1	3.75	9.93
	ShallowBlocker	1.74	2.60	2.03	2.92	3.32	2.37	2.03	2.96	3.31	2.37	1.95	**4.11**	**2.18**	5.30
Supervised ≤ 1000 matches	Sup-MinHash LSH	0.20	-	0.22	1.17	-	0.40	0.22	1.18	-	0.40	0.21	18.4	OOM	OOM
	Sup-τ-join	**0.17**	-	**0.18**	**0.36**	-	**0.26**	**0.18**	**0.37**	-	**0.26**	**0.18**	**2.78**	**2.19**	**4.02**
	Sup-AutoBlock	11.1	-	11.1	12.0	-	11.4	11.0	19.6	-	14.8	10.9	15.6	54.2*	OOM
	Sup-DeepBlocker	13.0	-	13.1	13.4	-	13.2	13.1	13.4	-	14.8	13.1	ERR	ERR	45.1*
	Sup-k-join	0.19	-	0.20	1.10	-	0.48	0.20	1.12	-	0.49	**0.18**	**4.93**	**3.22**	10.0
	ShallowBlocker	1.82	-	2.33	3.11	-	2.40	2.33	3.10	-	2.40	2.06	4.13	**2.30**	5.50
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	**17.7**	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	**2.92**	**2.14**	**4.02**
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	15.6	52.7*	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	73.9*	45.0
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	**8.16**	**3.14**	**10.1**
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	4.18	2.67	6.17

Table 4.10: Memory usage of all methods across all datasets in both unsupervised and different supervised settings when the recall target is 90%. This complements Table 4.8. The lowest memory usage each dataset and resource setting is underlined, while methods that are within 5% of this is bold. Methods that failed to reach to recall target is grayed out. OOM: Out of memory. ERR: Hitting limits of FAISS. TIME: Exceeding 24h time limit. * One or more runs crashed.

4.11 Experiments

4.11.1 Recall Target

The experiment compare the performance characteristics between methods when targeting the same recall level. Table 4.8 and 4.10 report results when targeting 90%, while Table 4.9 and 4.11 report results when targeting 98%. We consider 90% and 98% as representative of low and high recall. For each method we report recall, the number of returned pairs, runtime, and memory usage. We normalize the number of returned pairs $|P|$ and report the effective cardinality $\widetilde{k} = |P|/\min(|A|, |B|)$ for easier comparison across dataset sizes. All reported numbers are the average over five runs, and all are given a maximum runtime of 24 hours.

Resource Setting	Method	Dataset													
		AG	B	DA	DG	IA	WA	D-DA	D-DG	D-IA	D-WA	AB	C	S	CD
		Memory Usage (GB)													
Unsupervised	Token Blocking	0.14	0.32	0.21	1.85	6.04	0.40	0.21	1.84	6.04	0.38	0.13	18.7	OOM	OOM
	HVTB	0.12	0.14	0.12	0.28	0.98	0.17	0.13	0.28	0.39	0.17	0.12	8.00	38.3	OOM
	HVTB+	0.11	0.13	0.12	0.24	0.44	0.15	0.12	0.23	0.22	0.15	0.11	5.10	2.23	9.30
	MinHash LSH	0.19	0.47	0.24	2.06	5.53	0.58	0.24	2.11	5.46	0.58	0.16	13.8	OOM	OOM
	PPJoin	**0.01**	0.14	**0.02**	**0.15**	OOM	**0.02**	**0.02**	**0.15**	OOM	**0.02**	**0.01**	OOM	OOM	OOM
	τ-join	0.12	**0.12**	0.12	0.29	0.28	0.19	0.12	0.29	0.29	0.19	0.12	1.88	3.40	12.6
	DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	k-join	0.12	**0.12**	0.13	0.29	**0.21**	0.18	0.13	0.29	**0.22**	0.18	0.12	1.87	**1.69**	4.89
	ShallowBlocker ($q=0.95$)	0.42	0.46	0.48	0.77	0.78	0.57	0.48	0.77	0.78	0.57	0.44	2.12	2.34	**4.02**
	ShallowBlocker ($q=0.99$)	0.42	0.45	0.48	0.77	0.78	0.57	0.48	0.78	0.78	0.57	0.43	2.14	1.89	**3.97**
	ShallowBlocker ($q=1$)	0.42	0.45	0.48	0.78	0.79	0.56	0.48	0.78	0.79	0.56	0.43	2.12	1.81	**3.97**
Supervised ≤ 100 matches	Sup-MinHash LSH	0.23	0.43	0.23	1.33	2.04	0.45	0.23	1.69	2.05	0.46	0.24	18.9	OOM	OOM
	Sup-τ-join	**0.17**	0.34	**0.18**	**0.37**	**0.25**	**0.26**	**0.18**	**0.37**	**0.25**	**0.26**	0.18	18.4	3.59	**4.02**
	Sup-AutoBlock	11.3	10.9	10.9	22.2	66.3	12.1	10.9	24.8	45.0	14.5	11.1	15.6	OOM	OOM
	Sup-DeepBlocker	13.3	ERR	13.1	13.7*	14.2	13.4*	13.1	13.6*	14.0	14.8*	13.2	ERR	ERR	59.4
	Sup-k-join	**0.18**	**0.21**	0.19	1.10	0.81	0.43	**0.19**	1.06	0.77	0.43	0.17	18.9	4.52	10.1
	ShallowBlocker	1.74	2.61	2.03	2.94	3.31	2.37	2.03	2.96	3.31	2.37	1.95	11.5	**2.71**	5.31
Supervised ≤ 1000 matches	Sup-MinHash LSH	0.23	-	0.22	1.18	-	0.62	0.22	1.17	-	0.44	**0.25**	18.7	OOM	OOM
	Sup-τ-join	**0.18**	-	**0.18**	**0.36**	-	**0.26**	**0.18**	**0.37**	-	**0.26**	**0.19**	18.5	3.26	**4.01**
	Sup-AutoBlock	11.4	-	11.2	21.9	-	12.2	11.5	20.0	-	14.8	11.0	15.6	OOM	OOM
	Sup-DeepBlocker	13.2	-	13.1	13.7	-	13.2	13.1	13.7*	-	14.8	13.2	ERR	ERR	ERR
	Sup-k-join	0.19	-	0.20	1.08	-	0.48	0.20	1.12	-	0.48	0.19	19.1	4.18	10.0
	ShallowBlocker	1.81	-	2.33	3.10	-	2.39	2.33	3.09	-	2.40	2.06	19.3	**2.74**	5.51
Supervised ≤ 10 000 matches	Sup-MinHash LSH	-	-	-	-	-	-	-	-	-	-	-	18.9	OOM	OOM
	Sup-τ-join	-	-	-	-	-	-	-	-	-	-	-	18.7	**2.79**	**4.03**
	Sup-AutoBlock	-	-	-	-	-	-	-	-	-	-	-	15.6	OOM	OOM
	Sup-DeepBlocker	-	-	-	-	-	-	-	-	-	-	-	ERR	ERR	45.0*
	Sup-k-join	-	-	-	-	-	-	-	-	-	-	-	19.7	4.04	10.1
	ShallowBlocker	-	-	-	-	-	-	-	-	-	-	-	20.3	2.97	6.17

Table 4.11: Memory usage of all methods across all datasets in both unsupervised and different supervised settings when the recall target is 98%. This complements Table 4.9. The lowest memory usage each dataset and resource setting is underlined, while methods that are within 5% of this is bold. Methods that failed to reach to recall target is grayed out. OOM: Out of memory. ERR: Hitting limits of FAISS. * One or more runs crashed.

When used supervised, ShallowBlocker uses the objective function

$$f(\text{recall}, |P|, \text{runtime}) = (\text{recall}, \text{cost})$$

where cost is

$$\text{cost}(|P|, \text{runtime}) = \widetilde{k} + c_{\text{rt}}\text{runtime}$$

and ordering of $x = (\text{recall}_x, \text{cost}_x)$ and $y = (\text{recall}_y, \text{cost}_y)$ is given by

$$x \geq y = \begin{cases} \text{cost}_x \leq \text{cost}_y & \text{if recall}_x \geq R \wedge \text{recall}_y \geq R \\ \text{recall}_x \geq \text{recall}_y & \text{otherwise} \end{cases}$$

We set c_{rt} to a conservative value of 0.01, meaning we consider a unit increment in effective cardinality and a runtime increase of 100 seconds equal.

The unsupervised methods use the best hyperparameters that achieve the desired recall on all the test sets as described in Section 4.10.3. We remind the

reader that this means the same hyperparameters are used across all datasets, and not tuned per dataset. For unsupervised ShallowBlocker we report the results when setting the quality q to 0.95, 0.99, and 1.

The supervised methods get a training set of known matches different from the test set. In order to study how the number of training matches affects the results, we run each method with maximum 100, 1000, and 10 000 sampled training matches.

Unsupervised Pair Effectiveness

We observe that ShallowBlocker is, with good margin, the method that is able to achieve the desired recall with the least number of returned pairs, both for low and high recall levels. The contrast is most pronounced with high recall. DeepBlocker is not able to achieve 98% recall across all datasets because FAISS is unable to return enough pairs. Furthermore, baseline methods using unweighted tokens (Token Blocking, PPJoin, Minhash LSH) struggle significantly more than those relying on weighted tokens (HVTB, τ-join, k-join). We assume this can be attributed to the excellent discriminatory power of TF-IDF. Unsurprisingly, k-join is more stable than τ-join because it explicitly limits the number of pairs. On the other hand, τ is able to prune more aggressively on some datasets but end up returning way to many pairs on others. We see that ShallowBlocker is able avoid exploding $|P|$ while still exploiting the pruning power of similarity thresholds — showing the strength of using (τ, τ_r, k)-joins.

Supervised Pair Effectiveness

We see that ShallowBlocker achieves the highest effectiveness in most supervised cases, and often with a substantial margin. The differences tends to be larger for the high recall target, which makes sense because effective pruning is harder. However, note that τ-join and k-join are sometimes more effective with few training matches, and ShallowBlocker slightly misses the high recall target for a few datasets. This is mainly due to ShallowBlocker having more parameters and expressive power, and therefore more prone to variance with few training matches. The attentive reader might have noticed that unsupervised ShallowBlocker actually outperforms all the supervised methods in a few cases. Here it is important again to remember that the hyperparameters of the unsupervised methods were picked using the test sets to show best case performance. The supervised methods can only rely on the train set and must pick more conservative parameters to incorporate the resulting uncertainty.

While DeepBlocker significantly outperforms AutoBlock, both deep learning-based methods are dramatically less effective than ShallowBlocker. We see that this also make them struggle with the large datasets.

Runtime

For small and medium datasets we see that τ-join and k-join are overall the fastest. The reason ShallowBlocker is slower is because of the overhead it incurs for automatically selecting join parameters and performing two joins — both in the unsupervised and supervised setting. It is however kept consistently within a few seconds of the fastest baselines, and importantly, pays of in competitive runtimes for the large datasets Songs and Citeseet-DBLP. Note that runtime is tightly coupled to the pair effectiveness and the number of returned pairs.

The runtime of PPJoin and Minhash LSH variate a lot more, and can be high for even moderately sized datasets such as iTunes-Amazon. Mainly because it needs very permissive thresholds to achieve the target recall. DeepBlocker is the slowest of all the unsupervised methods by a significant margin, and the slowest together with AutoBlock in the supervised setting. The embedding training is the bottleneck for small datasets, while the pair generation using nearest neighbor search on the embeddings is the bottleneck for larger datasets. For the largest dataset, Citeseer-DBLP, DeepBlocker is not able to finish within the time limit of 24 hours in the low recall setting.

The overhead of supervised ShallowBlocker is generally more than unsupervised ShallowBlocker, as shown by the higher runtimes for most datasets. However, on Citeseer-DBLP we also see that supervised ShallowBlocker can leverage training data to choose join conditions and approximation degree that actually significantly reduce the runtime.

Approximate Joins

We see, as expected, using approximate joins instead of exact joins has an overall negative effect on the pair effectiveness of unsupervised ShallowBlocker (except for iTunes-Amazon and Walmart-Amazon in the low recall setting, which we think is mostly noise). The loss in effectiveness is more noticeable in the high recall setting, especially for $q = 0.95$.

The goal of using approximated joins is to reduce runtime. For datasets the exact ShallowBlocker already processes fast we actually observe an increase in runtime. The reason is that extra overhead of determining ρ^* exceeds any reduction in the already low runtime. However, note that for the largest dataset,

Citeseer-DBLP, we are able to significantly reduce the runtime in exchange for moderate increases in $|P|$. For example, in the high recall setting ShallowBlocker achieve a 36% (2.5 minutes) reduction in runtime by an increase in $|P|$ of only 7% when using $q = 0.99$. So while approximate joins are not a silver bullet for increased speed while achieving the same recall, and that it makes sense to run with $q = 1$ by default, we think it is a valuable option to have at disposal for large datasets.

Memory Usage

Unsurpricingly, the deep learning-based methods use substantially more memory than classical methods. PPJoin is the most nimble on a majority of the datasets because of its highly optimized implementation over unweighted tokens. However, as with runtime, memory usage is heavily coupled to the number of returned pairs — and PPJoin is therefore not able to finish within the memory limit on the largest datasets. ShallowBlocker generally has a higher memory overhead on smaller datasets because of its extra bookkeeping for choosing parameters and running two joins. Nonetheless, for larger datasets this overhead is outweighted by more effective join conditions resulting in low memory usage.

4.11.2 Effectiveness Trade-off

We study the trade-off between recall and the number of returned pairs $|P|$ in detail. Again, for easier comparison across datasets we report the effective cardinality $\widetilde{k}$ instead of $|P|$. Figure 4.8 shows recall plotted against $\widetilde{k}$ across all datasets for each method. We traverse the trade-off front of each baseline by varying the only threshold (similarity or cardinality) parameter or recall target. Similarly, for ShallowBlocker we vary k and for AutoShallowBlocker we use the objective function

$$f(\text{recall}, |P|, \text{runtime}) = \text{recall} - c_k\widetilde{k} - c_{\text{rt}}\text{runtime}$$

and vary c_k (using $c_{\text{rt}} = 0.01$ as before).

We observe many of the same trends as in the previous experiment. ShallowBlocker and AutoShallowBlocker generally demonstrate the best pair effectiveness, while τ-join and k-join are the strongest baselines. Baselines using unweighted tokens (Token Blocking, Minhash LSH, and PPJoin) are overall considerably less effective, but PPJoin is effective on datasets like DBLP-ACM, Songs, and Citeseer-DBLP. The deep learning methods are by far the least ef-

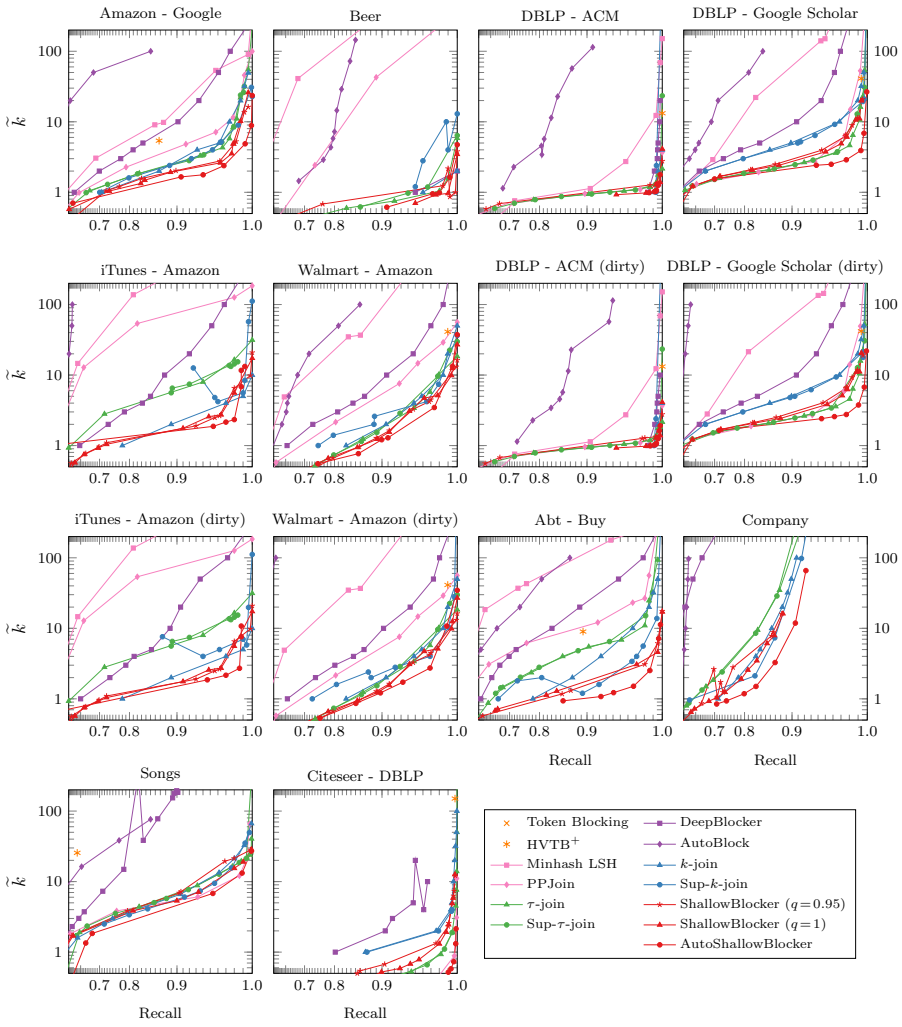


Figure 4.8: Plot for every dataset showing the trade-off between number of returned pairs (expressed as $\widetilde{k} = |P|/\min(|A|, |B|)$) and recall for all methods. Note that we have chosen a range of $\widetilde{k}$ and recall we deem interesting, which means sometimes methods are not visible in the plot. Token Blocking, for example, is not visible in any plot because its $\widetilde{k}$ is so large.

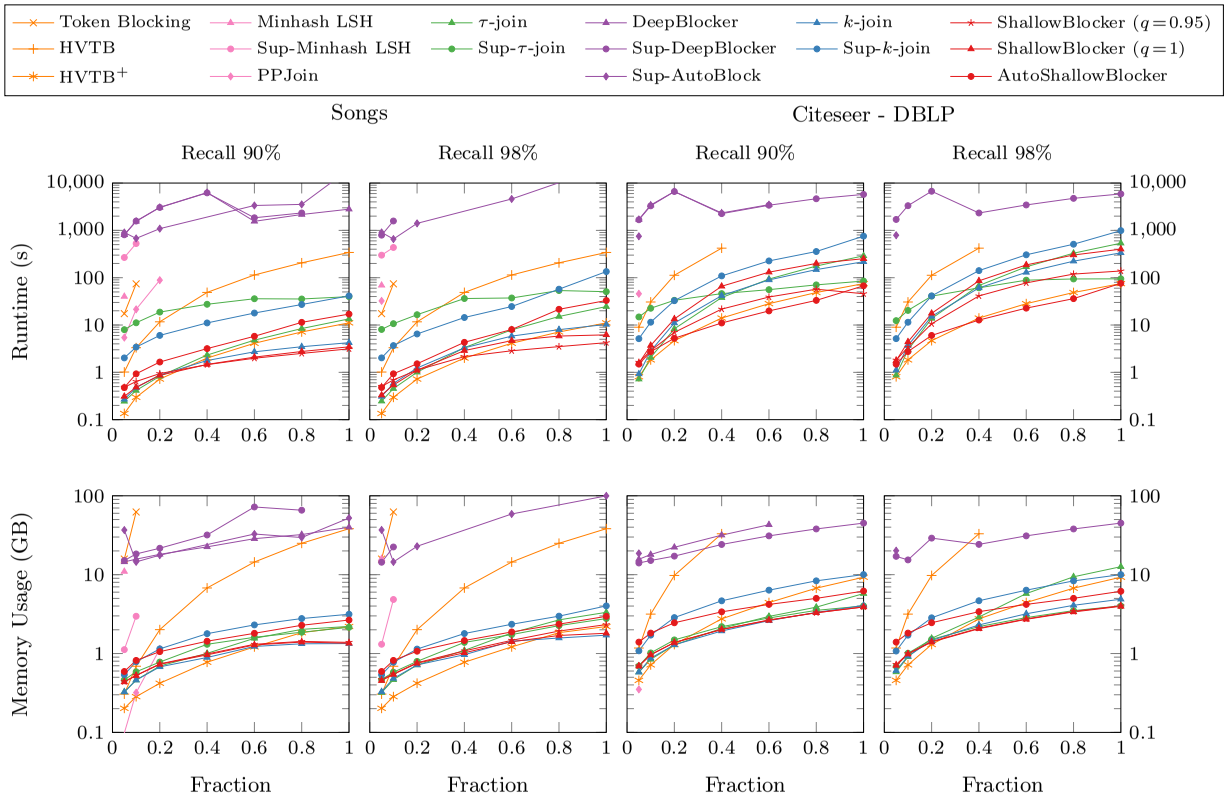


Figure 4.9: Runtime and memory consumption for all methods on Songs and Citeseer-DBLP datasets with 90% and 98% recall target when we vary the dataset size. Methods with lines that stop early hit the memory limit or similar.

fective. Note that most methods not significantly effected by dirty data, except for AutoBlock — which do rely more on attribute boundaries.

4.11.3 Scalability

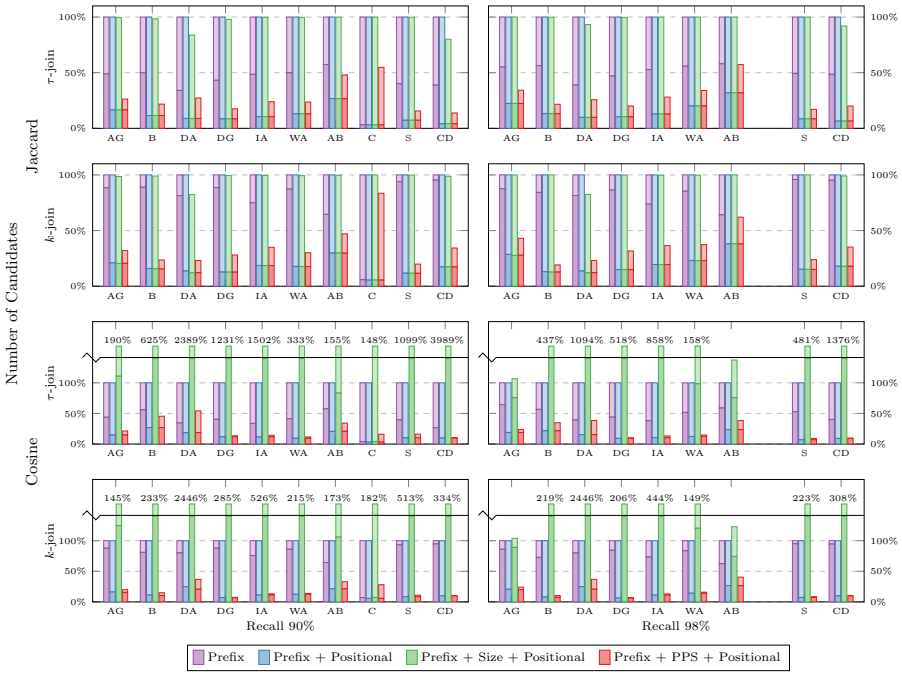


Figure 4.10: The number of pre-candidates and candidates when using different filter configurations across all datasets for τ-join and k-join using Jaccard and Cosine with hyperparameters that achieve 90% and 98% recall. The Company dataset is omitted for 98% since it is not possible to achieve that recall level. The numbers are relative to the number of pre-candidates when using only prefix filtering. The full height of each bar is the number of pre-candidates and the bottom is the number of candidates.

Figure 4.9 shows the runtime and memory consumption of different methods when running on different fraction sizes of the two largest datasets, Songs and Citeseer-DBLP. We report for configurations that achieve both 90% and 98%

as in Section 4.11.1.

From the plots we see clearly that Token Blocking, Minhash LSH, PPJoin, and AutoBlock scales poorly. They are mostly not able to process the entire datasets. This is mainly due to low pair effectiveness and the need for conservative hyperparameters to meet the recall target. On the other hand, HVTB$^+$, τ-join, k-join, ShallowBlocker, and AutoShallowBlocker all scale reasonable well and have similar characteristics. Note that the improved HVTB* is considerably more memory efficient than HVTB, which is not able to process Citeseer-DBLP within the memory constraints. As we saw in Section 4.11.1, ShallowBlocker and AutoShallowBlocker have a larger overhead than many of the leaner baselines. They have higher runtime and memory consumption for the smallest fraction, but scale well due to aggressive pruning when the datasets grow. The deep learning methods have a much more extreme overhead. They are dominated by long training time for smaller fractions and then increasingly dominated by nearest neighbor search on embeddings for the larger ones. Unsupervised DeepBlocker and Sup-AutoBlock are not able process the whole Citeseer-DBLP dataset.

4.11.4 Prefix-Partitioned Suffix Filtering

We examine the effect of our new proposed pre-candidate filter: Prefix-Partitioned Suffix (PPS) Filter.

Number of Pre-Candidates

			τ-join						k-join					
Measure	Recall	Filters	DG	IA	WA	C	S	CD	DG	IA	WA	C	S	CD
Jaccard	90%	Prefix	**153ms**	158ms	**67.3ms**	1m9s	6.66s	27.2s	187ms	212ms	**71.8ms**	49.9s	19.6s	27m17s
		Prefix + Pos.	**150ms**	132ms	**67.2ms**	1m9s	4.07s	13.5s	177ms	**163ms**	**69.2ms**	**46.6s**	6.25s	6m58s
		Prefix + Size + Pos.	**151ms**	138ms	**69.2ms**	1m8s	4.05s	12.9s	184ms	171ms	**69.4ms**	**47.5s**	6.51s	7m11s
		Prefix + PPS + Pos.	**157ms**	**125ms**	**69.1ms**	**1m5s**	**3.57s**	**10.4s**	**164ms**	**161ms**	**69.3ms**	**46.5s**	**4.91s**	**4m38s**
	98%	Prefix	167ms	184ms	**71.3ms**	-	15.7s	1m15s	203ms	230ms	74.9ms	-	1m28s	31m35s
		Prefix + Pos.	**159ms**	150ms	**70.6ms**	-	7.05s	30.0s	183ms	**176ms**	**67.6ms**	-	19.3s	8m10s
		Prefix + Size + Pos.	**156ms**	155ms	**68.5ms**	-	7.15s	30.3s	184ms	183ms	73.2ms	-	20.2s	8m28s
		Prefix + PPS + Pos.	**155ms**	**138ms**	**70.4ms**	-	**5.83s**	**21.4s**	**167ms**	**170ms**	**70.5ms**	-	**13.9s**	**5m28s**
Cosine	90%	Prefix	**155ms**	127ms	**70.5ms**	32.6s	5.58s	13.1s	180ms	172ms	73.7ms	26.6s	23.0s	21m52s
		Prefix + Pos.	**152ms**	**119ms**	**68.4ms**	28.8s	3.62s	**9.39s**	**157ms**	**133ms**	**69.5ms**	**25.1s**	4.76s	2m13s
		Prefix + Size + Pos.	184ms	434ms	72.1ms	1m3s	46.8s	12m18s	260ms	563ms	78.4ms	48.1s	1m58s	1h12m
		Prefix + PPS + Pos.	**151ms**	**116ms**	**68.5ms**	**26.2s**	**3.40s**	**9.04s**	**157ms**	**132ms**	**69.9ms**	**24.0s**	**4.32s**	**1m51s**
	98%	Prefix	**156ms**	136ms	72.7ms	-	18.5s	46.3s	207ms	191ms	76.6ms	-	2m13s	26m8s
		Prefix + Pos.	**152ms**	**124ms**	**68.8ms**	-	6.20s	18.1s	**159ms**	141ms	**71.9ms**	-	13.0s	2m38s
		Prefix + Size + Pos.	219ms	490ms	81.2ms	-	1m38s	19m3s	293ms	590ms	81.8ms	-	5m2s	1h20m
		Prefix + PPS + Pos.	**153ms**	**121ms**	**70.5ms**	-	**5.38s**	**16.3s**	**160ms**	**134ms**	**71.3ms**	-	**10.4s**	**2m14s**

Table 4.12: The runtime when using different filter configurations across all datasets for τ-join and k-join using Jaccard and Cosine with hyperparameters that achieve 90% and 98% recall.

Recall	Filters	Dataset					
		DG	IA	WA	C	S	CD
90%	Prefix	3.26s	2.64s	710ms	52.6s	12.6s	9m19s
	Prefix + Pos.	**2.80s**	**2.26s**	**649ms**	48.7s	4.49s	2m18s
	Prefix + Size + Pos.	6.12s	13.3s	1.18s	2m5s	4.70s	9m54s
	Prefix + PPS + Pos.	**2.71s**	**2.22s**	**642ms**	**44.7s**	**3.62s**	**1m12s**
98%	Prefix	3.58s	3.35s	1.05s	-	31.8s	23m38s
	Prefix + Pos.	**3.09s**	**2.90s**	**971ms**	-	8.99s	5m53s
	Prefix + Size + Pos.	6.69s	21.9s	2.08s	-	9.35s	15m33s
	Prefix + PPS + Pos.	**2.97s**	**2.84s**	**954ms**	-	**6.35s**	**3m24s**

Table 4.13: The runtime when using different filter configurations across all datasets for unsupervised ShallowBlocker with hyperparameters that achieve 90% and 98% recall.

Figure 4.10 shows the relative number of pre-candidates and candidates for τ-join and k-join using Jaccard and Cosine with thresholds achieving 90% and 98% recall per dataset. Reporting results for τ-join and k-join instead of ShallowBlocker let us control for the effect of different constraint types and similarity measures. We compare the number of pre-candidates and candidates when using prefix filter only, prefix and positional filter, PPJoin-style filters (prefix, size, positional), and `TTRKJoin` filters (prefix, PPS, positional). Note that in order to use size filtering for Cosine we must use L_1 based bounds and thereby the weaker prefix bound from PPJoin. The effect of using the improved L2AP [2] prefix bound without PPS filtering can therefore be observed by looking at the results for using only prefix and positional filtering.

Overall, we observe a reduction between approximately 20% and 90% of the number of pre-candidates when applying PPS filtering. For the majority of cases the reduction is 70% or more. This is significantly more than for PPJoin-style size filtering, which we see have negligible effect for almost all datasets and at most reduce the number of pre-candidates with around 20%. For Cosine the weaker L_1 prefix bound necessary to use size filtering makes it hard to compare PPS against size filtering directly. However, it clearly shows that the L_2 based prefix bound from L2AP is a large improvement.

From the results we observe no major difference in behavior between thresholds for high and low recall. Comparing τ-join and k-join we see the main difference is the relative number of pre-candidates and candidates. τ-join has fewer candidates compared to pre-candidates than k-join. This is mostly because τ-join is able to exploit the similarity threshold to prune when indexing, and thereby reducing the number of duplicates when looking up the inverted lists query time. Therefore, we would expect the runtime effect of PPS filtering

to be greater for k-join. Comparing Jaccard against Cosine we see that PPS filtering is most effective for Cosine — most likely because Jaccard relies on the looser minPrefix of the inverted lists as bound on p_b to get similarly tight suffix bounds.

Runtime

Table 4.12 lists the runtime of the experiments above from Figure 4.10. For clarity and conciseness we only report runtimes for datasets with runtime above 50ms. The remaining datasets are so small that the runtime is practically indistinguishable between different filtering techniques.

Previous work has shown that filtering techniques that generate fewer candidates than PPJoin or AllPairs often ends up being slower because of the extra overhead [87]. However, we see from the reported runtimes that PPS filtering either improves or matches size filtering across the board. The improvements are most noticeable for the largest datasets, while the overhead of any filtering beyond prefix provides no or minimal gains for smaller datasets. Furthermore, we observe the enormous negative impact PPJoin style L_1 size filtering (with corresponding prefix bounds) has when using Cosine. As expected, the improvements are greater for k-join than τ-join.

In contrast to τ-join and k-join, `TTRKJoin` mixes different join conditions and ShallowBlocker mixes different similarity measures. Table 4.13 lists the runtimes for unsupervised ShallowBlocker ($q = 1$) using the different filtering configurations for k yielding 90% and 98% recall. These results show that the runtimes from our controlled experiments on τ-join and k-join are indicative of the effect it has on general use of `TTRKJoin`.

4.12 Conclusion

We have introduced, ShallowBlocker, a novel blocking method based on efficient hybrid set similarity joins and shown through extensive experiments that it is state-of-the-art. Specifically, it is able to achieve the same recall with fewer returned pairs in most cases compared to previous state-of-the-art methods in reasonable time. While it has a non-trivial runtime overhead for small datasets, experiments show it is more efficient on large datasets than all the baselines.

The need for dataset-specific laborsome human engineering and tuning is often highlighted as a key challenge for blocking. It is tempting to think that classical blocking methods are too rigid and therefore will always need man-

ual feature engineering for good results, so we require the automatic feature learning from deep learning to create the best hands-off solutions. However, our work shows that classical string similarity-based techniques can be used to create hands-off blocking methods that outperform state-of-the-art deep learning based methods — even on dirty data. Importantly, our proposed method requires considerably fewer computational resources and is significantly more interpretable. Despite recent advances, our work suggests that classical techniques for blocking are still highly competitive to deep learning methods.

There is still a lot to explore with the proposed techniques and methods. We think (τ, τ_r, k)-join is a powerful primitive and we hope to see other fruitful uses of `TTRKJoin` outside the scope of ShallowBlocker. Even though we see ShallowBlocker outperforming deep learning-based methods on the benchmark datasets used in this paper, it is an inherent limitation that methods based on classical string similarity rely on some level of syntactic similarity. It would be interesting to explore the possibility of combining elements of both ShallowBlocker and deep learning techniques to exploit the strengths of both approaches.

Chapter 5

Discussion

In this thesis, we set out to explore novel deep learning-based approaches to entity matching, how they differ from traditional methods, and their implications. We now go through and address the research questions we outlined at the beginning of the thesis.

5.1 Research Questions

RQ1 What are the contributions from deep learning to entity matching?

Paper A provided a substantial survey of the use of neural networks in the entity matching process. We identified two main and novel contributions deep learning brings to entity matching:

1. **Learned feature extraction and comparison**: One of the challenges of more traditional string similarity-based machine learning approaches is that, even though the method can learn which features to use and how to apply them, it is often necessary to manually tweak or craft similarity features for new datasets. Deep learning approaches are able to learn to extract features and how to compare them. This significantly reduces manual feature engineering efforts. Perhaps even more important, it enables us to exploit powerful semantic similarity. We simply do not have effective techniques for manually crafting semantic similarity features at the same level as deep learning methods.

The drawbacks are that the required amount of training data and computational resources is significantly higher. Deep neural networks can have millions or billions of parameters and operate more or less directly on raw text. Tuning them is a much more daunting task than determining some thresholds or linear combinations of a few select features handcrafted for the task at hand. However, the recent introduction of large pretrained language models greatly reduces these drawbacks. Fine-tuning such models achieves state-of-the-art results with moderate amounts of training data and a fraction of the compute compared to training from scratch.

2. **Coalescing the entity matching process**: Traditionally, entity matching typically follows a (relatively) standard process of clearly separated steps: 1) Data preprocessing, 2) Schema Matching, 3) Blocking, 4) Record pair comparison, and 5) Classification (see Figure 2.2). Deep learning methods enables us to unify or merge several of these steps (as illustrated in Figure 2.4). A single neural network can be responsible for most of the data preprocessing, schema matching, blocking, record pair comparison, and classification. This has the potential to simplify the architecture of entity matching systems.

   On the other hand, this is also an important reason why deep learning solutions are less interpretable. The clearly separated steps of the traditional process make it inherently more interpretable because one can inspect and reason about each step separately. In a more tightly integrated deep learning process it is harder to track and pinpoint why records end up being considered a match or not.

RQ2 How does modern deep learning approaches compare to classical approaches for blocking?

There are a multitude of classical approaches [100]. We saw in paper A and C that deep learning-based blocking relies on approximate k nearest neighbor search of dense embeddings (at attribute and/or record level). The main difference from classical approaches is the use of dense and semantic embeddings instead of sparse and (mostly) syntactic representation (e.g., bag of q-grams/words). Generally, this makes the representation more expressive, but at the cost of making it harder to interpret the representation and build efficient search procedures.

The proposed method ShallowBlocker in paper C and the comparison to state-of-the-art deep learning methods show that classical string similarity-based

methods are still able to outperform deep learning-based methods in terms of recall, pair efficiency, and runtime efficiency — while also being more interpretable — on popular benchmark datasets. The main advantage deep learning has is the ability to learn expressive semantic features. However, since comparison in blocking can not be done explicitly and must happen through the embedding space, the precision of comparison is significantly limited. We suspect expressive semantic features are less important for blocking because syntactic similarity is often good enough in practice to find reasonable candidates that can be classified with higher precision in the downstream explicit record pair comparison, and that tailor-made algorithms on carefully selected sparse syntactic representations can simply recall candidates more reliably and efficient than the constrained embeddings. These challenges can, of course, be overcome in the future, and we look forward to seeing the advances of deep learning for blocking.

RQ3 Can we explain predictions from deep learning matchers and classical matchers in the same frame of interpretation, and if so, how?

We argue this is possible. Paper B proposed LEMON, a model-agnostic explainability method and framework for interpretation based on feature attribution tailored to entity matching. The key to having a homogeneous explanation format across all matchers is the fact that we create them post hoc and only rely on perturbations of the input to study the behavior of the matcher — and therefore do not rely on the inner workings of the matchers. It is fundamentally a feature attribution method. However, as we discuss in the paper, traditional local post hoc feature attribution methods are not suitable for explaining why records do not match because they only explain subtractively (see Section 3.4.2 and results for non-matches in Table 3.2). Therefore, we also propose the notion of attribution potential, which identifies tokens a matcher would find important if it matched better with the other record. Synthetic experiments and experiments on humans show that LEMON is effective on both a widely used state-of-the-art classical method and deep learning method.

RQ4 How does explainbility methods adapted to entity matching improve interpretation of matchers?

In paper B we saw that popular general-purpose local post hoc attribution explainability methods have several fundamental challenges when applying them to entity matching. We identified three: 1) Cross-record interactions effects, 2) Non-match explanations, 3) Variation in sensitivity. Our proposed method,

LEMON, builds upon the widely used LIME [114] method, and addresses all three challenges. User studies from Paper B showed that the rate at which human subjects can construct counterfactual examples after seeing an explanation from our proposed method increases from 54% to 64% for matches and from 15% to 49% for non-matches compared to explanations from a standard adaptation of LIME. Of course, user studies are difficult to execute and analyze correctly. There are many potential sources of noise and error — such as how test users are picked and what precise instructions they are given. We hope to see more user studies from the research community in the future to gain more nuance. However, these results, together with the extensive synthetic experiments, suggest that one can achieve significantly more effective explanations by specifically targeting the unique challenges in explaining entity matching predictions and adapting existing explainability methods to entity matching.

5.2 Implications of Contributions

We briefly discuss potential implications of our contributions.

C1 **Insight and overview of how new deep learning methods compare to classical methods for entity matching**. We hope the work in this thesis help other researchers gain deeper insight into the unique opportunities and challenges deep learning opens up for entity matching, and that it will aid them in developing techniques and methods that exploit the opportunities and overcome the challenges. Entity matching is a ubiquitous and often a highly valuable data integration task in practice if can be sufficiently solved. Therefore, most significant advances are likely to translate into real-world benefit to a wide range of data integration practitioners.

C2 **A state-of-the-art model-agnostic explainability method tailored to entity matching**. Our proposed method, LEMON, provides researchers and practitioners a potentially powerful method for inspecting and interpreting the behaviour of matchers that can be used to troubleshoot or discover ways to improve matchers. Furthermore, it can be used a tool to compare the behavior or fundamentally different matchers (e.g., classical vs deep learning). For researchers, our employed techniques can be used as a basis for developing better or other types of explainability methods for entity matching.

C3 **A state-of-the-art blocking method based on set similarity joins**. Choosing and tuning a blocking strategy can be challening in practice. Our proposed method, ShallowBlocker, improves the state-of-the-art pair effectiveness on popular benchmark datasets in both unsupervised and supervised settings, while being hands-off, interpretable, and relatively fast. ShallowBlocker provides researchers and practioners with a potentially improvement over existing and currently used blocking methods in terms of pair effectiveness and ease of use. The proposed method and the extensive experimental results can act as a starting ground for further research into why deep learning methods are not able to perform at the same level — and ultimately improving them. Furthermore, the (τ, τ_r, k)-join primitive, its heuristic approximation scheme, and `TTRKJoin` can be used as building blocks for future methods.

5.3 Final Thoughts

We set out to explore the novel use of deep learning for entity matching. We were interested in contrasting it to existing classical approaches and methods, and gain novel insight into the benefits and drawbacks. Furthermore, this thesis has focused on interpretability and blocking as two core aspects to highlight the differences.

Deep learning-based methods has some clear advantages over previous methods. In particular, we identified the learning of feature extraction and comparison and the coalescing of the entity matching process as the two main benefits. However, as discussed, both come with trade-offs.

In practice, we do not think that challenges with regard to training data and computational resources will hinder adoption. The progress enabled by large pre-trained language models significantly reduces these challenges. The massive attention they and their ecosystem are getting will only make them more easy to use and accessible. We believe that interpretability will be a bigger hurdle for adoption. Entity matching is often be a core task in data integration workflows for critical domain/business data where the trust in the data quality is key. We hope the work presented in Paper B, which provides a novel state-of-the-art explainability method for entity matching that can explain predictions from deep learning matchers and classical matchers in the same frame of interpretation, will be of help to practitioners.

Existing work, as we saw in Paper A, has demonstrated the effectiveness of deep learning for record pair comparison and classification. It is able to achieve

significantly higher precision than classical approaches across most benchmark datasets. However, as we demonstrated in Paper C, it is less clear whether deep learning provides clear benefits over classical approaches for blocking. In theory, it should be beneficial to datasets that rely more on semantic than syntactic similarity, but that do not result in better recall, pair effectiveness, or runtime on standard benchmark datasets. We think there is substantial room for improvement of deep learning-based blocking and expect to see continued research in this area. We hope the novel state-of-the-art classical blocking method introduced in Paper C will serve as a strong baseline for the research community.

5.3.1 Future Research Directions

While we see many potential avenues for future research on the topics covered in this thesis, we find two directions particularly interesting:

- **Hybrid approaches**: In this thesis, we have seen that classical and deep learning methods have their own strengths and weaknesses. We think the time is ripe for exploring hybrid approaches — mixing more traditional approaches with deep learning approaches. For blocking, one could explore how the computational efficiency of string similarity joins could be used to more effectively prune/reduce the search space for embeddings. For record pair comparison and classification, one could explore the possibility of incorporating string similarity measures as input to a deep learning network. We hope to see substantial contributions to the field from the combination of techniques from these two worlds.

- **End-to-end explainability**: One significant shortcoming of all current explainability methods for entity matching is that they only target classification of explicit pairs downstream for the blocking step. If the reason two records are not matched is because the pair was not recalled in the blocking step none of the current methods would be able to pick up that and express that in the explanation. In the future, we hope to see methods trying to explain matches and non-matches across the entire entity matching process — including blocking. This is not trivial to achieve, but would be valuable because it reduces the complexity of working across the different steps in the entity matching process.

Chapter 6

Conclusion

In this thesis, we have taken an extensive look at the use of deep learning for entity matching. The goal was to gain novel insight into what these new techniques and approaches contribute to entity matching, and how they differ from classical approaches. We had a special focus on interpretability and blocking as two areas of contrast.

The thesis has three main contributions. First, we provide novel insight and overview of how new deep learning compares to classical methods for entity matching. We highlight learned feature extraction and comparison together with coalescing of the entity entity matching process as the two main benefits deep learning brings to entity matching. At the same time, it is a source for challenges in regard to the required amount of training data and computational resources as well as the lack of inherent interpretability.

Secondly, we present LEMON, a state-of-the-art model-agnostic explainability method tailored to entity matching. Since deep learning has limited interpretability compared to classical methods, this method could be a valuable tool in the transition between these paradigms given its ability to explain predictions of all methods in the same way.

Lastly, we present ShallowBlocker, a state-of-the-art blocking method based on set similarity joins. Extensive results highlight that there is still a lot to be desired for deep learning-based blocking. Future work can use this method as a strong baseline to represent classical approaches.

Entity matching is experiencing a paradigm shift that replaces traditional machine learning methods and classical syntactical string similarity techniques with deep learning-based approaches. We hope that this thesis is a valuable

contribution for practitioners and the research community as the development of deep learning for entity matching continues to unfold.

Bibliography

[1] Amina Adadi and Mohammed Berrada. "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)". In: *IEEE Access* 6 (2018), pp. 52138–52160. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2870052. (Visited on 05/08/2021).

[2] David C. Anastasiu and George Karypis. "L2AP: Fast Cosine Similarity Search with Prefix L-2 Norm Bounds". In: *2014 IEEE 30th International Conference on Data Engineering*. 2014 IEEE 30th International Conference on Data Engineering (ICDE). Chicago, IL, USA: IEEE, Mar. 2014, pp. 784–795. ISBN: 978-1-4799-2555-1. DOI: 10.1109/ICDE.2014.6816700. (Visited on 07/04/2022).

[3] Alexandr Andoni et al. "Practical and Optimal LSH for Angular Distance". In: *Advances in Neural Information Processing Systems 28*. Ed. by C Cortes et al. Curran Associates, Inc., 2015, pp. 1225–1233.

[4] A Arasu et al. "Transformation-Based Framework for Record Matching". In: *2008 IEEE 24th International Conference on Data Engineering*. ieeexplore.ieee.org, Apr. 2008, pp. 40–49.

[5] Arvind Arasu et al. "On Active Learning of Record Matching Packages". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA). New York, NY, USA: ACM, 2010, pp. 783–794.

[6] Dzmitry Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate". In: (Sept. 2014).

[7] Andrea Baraldi et al. "Using Landmarks for Explaining Entity Matching Models". In: *EDBT*. OpenProceedings.org, 2021, pp. 451–456.

[8] Nils Barlaug. *LEMON: Explainable Entity Matching*. Aug. 15, 2022. arXiv: 2110.00516 [cs]. URL: http://arxiv.org/abs/2110.00516 (visited on 09/03/2023). preprint.

[9] Nils Barlaug. "LEMON: Explainable Entity Matching". In: *IEEE Trans. Knowl. Data Eng.* 35 (2023), pp. 8171–8184. ISSN: 1558-2191. DOI: 10.1109/TKDE.2022.3200644. (Visited on 09/03/2023).

[10] Nils Barlaug. "Tailoring Entity Matching for Industrial Settings". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM '20: The 29th ACM International Conference on Information and Knowledge Management. Virtual Event Ireland: ACM, Oct. 19, 2020, pp. 3217–3220. ISBN: 978-1-4503-6859-9. DOI: 10.1145/3340531.3418514. (Visited on 09/25/2023).

[11] Nils Barlaug and Jon Atle Gulla. "Neural Networks for Entity Matching: A Survey". In: *ACM Trans. Knowl. Discov. Data* 15.3 (Apr. 12, 2021), pp. 1–37. ISSN: 1556-4681, 1556-472X. DOI: 10.1145/3442200. (Visited on 05/08/2021).

[12] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[13] Roberto J. Bayardo et al. "Scaling up All Pairs Similarity Search". In: *Proceedings of the 16th International Conference on World Wide Web - WWW '07*. The 16th International Conference. Banff, Alberta, Canada: ACM Press, 2007, p. 131. ISBN: 978-1-59593-654-7. DOI: 10.1145/1242572.1242591. (Visited on 12/19/2022).

[14] Zohra Bellahsene et al., eds. *Schema Matching and Mapping*. Springer, Berlin, Heidelberg, 2011.

[15] J Bennett and S Lanning. "The Netflix Prize". In: *Proceedings of the KDD Cup Workshop 2007*. New York: ACM, Aug. 2007, pp. 3–6.

[16] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: (July 2016).

[17] Ondřej Bojar et al. "Findings of the 2018 Conference on Machine Translation (WMT18)". In: *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Oct. 2018, pp. 272–303.

[18] A.Z. Broder. "On the Resemblance and Containment of Documents". In: *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171). June 1997, pp. 21–29. DOI: 10.1109/SEQUEN.1997.666900.

[19] Andrei Z Broder et al. "Min-Wise Independent Permutations". In: *Journal of Computer and System Sciences* 60.3 (June 2000), pp. 630–659. ISSN: 00220000. DOI: 10.1006/jcss.1999.1690. (Visited on 02/02/2023).

[20] Ursin Brunner and Kurt Stockinger. "Entity Matching with Transformer Architectures - A Step Forward in Data Integration". In: *EDBT*. OpenProceedings.org, 2020, pp. 463–473.

[21] Moses S Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing* (Montreal, Quebec, Canada). New York, NY, USA: Association for Computing Machinery, May 2002, pp. 380–388.

[22] S. Chaudhuri et al. "A Primitive Operator for Similarity Joins in Data Cleaning". In: *22nd International Conference on Data Engineering (ICDE'06)*. 22nd International Conference on Data Engineering (ICDE'06). Atlanta, GA, USA: IEEE, 2006, pp. 5–5. ISBN: 978-0-7695-2570-9. DOI: 10.1109/ICDE.2006.9. (Visited on 04/01/2022).

[23] Muhao Chen et al. "Co-Training Embeddings of Knowledge Graphs and Entity Descriptions for Cross-lingual Entity Alignment". In: (June 2018).

[24] Kyunghyun Cho et al. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Oct. 2014, pp. 103–111.

[25] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Berlin Heidelberg: Springer-Verlag, 2012. ISBN: 978-3-642-31163-5. DOI: 10.1007/978-3-642-31164-2. (Visited on 06/21/2020).

[26] Vassilis Christophides et al. "End-to-End Entity Resolution for Big Data: A Survey". Nov. 1, 2019. arXiv: 1905.06397 [cs]. URL: http://arxiv.org/abs/1905.06397 (visited on 06/23/2020).

[27] Alexis Conneau et al. "Very Deep Convolutional Networks for Text Classification". In: (June 2016).

[28] Sanjib Das et al. "Falcon: Scaling Up Hands-Off Crowdsourced Entity Matching to Build Cloud Services". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, May 9, 2017, pp. 1431–1446. ISBN: 978-1-4503-4197-4. DOI: 10.1145/3035918.3035960. (Visited on 01/28/2023).

[29] Sanjib Das et al. *The Magellan Data Repository*. 2016.

[30] Sanjib Das et al. *The Magellan Data Repository*. University of Wisconsin-Madison, 2022. URL: https://sites.google.com/site/anhaidgroup/projects/data.

[31] J Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255.

[32] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. (Visited on 05/18/2021).

[33] Vincenzo Di Cicco et al. "Interpreting Deep Learning Models for Entity Resolution: An Experience Report Using LIME". In: *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management - aiDM '19*. The Second International Workshop. Amsterdam, Netherlands: ACM Press, 2019, pp. 1–4. ISBN: 978-1-4503-6802-5. DOI: 10.1145/3329859.3329878. (Visited on 11/12/2020).

[34] AnHai Doan et al. *Principles of Data Integration*. Waltham, MA: Morgan Kaufmann, 2012. 497 pp. ISBN: 978-0-12-416044-6.

[35] Anhai Doan et al. "Human-in-the-Loop Challenges for Entity Matching: A Midterm Report". In: *Proceedings of the 2Nd Workshop on Human-In-the-Loop Data Analytics* (Chicago, IL, USA). New York, NY, USA: dl.acm.org, 2017, 12:1–12:6.

[36] Mengnan Du et al. "Techniques for Interpretable Machine Learning". In: *Commun. ACM* 63.1 (Dec. 20, 2019), pp. 68–77. ISSN: 0001-0782. DOI: 10.1145/3359786. (Visited on 05/19/2021).

[37] *Duplicate Detection, Record Linkage, and Identity Uncertainty: Datasets.* URL: http://www.cs.utexas.edu/users/ml/riddle/data.html.

[38] Amr Ebaid et al. "EXPLAINER: Entity Resolution Explanations". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019 IEEE 35th International Conference on Data Engineering (ICDE). Macao, Macao: IEEE, Apr. 2019, pp. 2000–2003. ISBN: 978-1-5386-7474-1. DOI: 10.1109/ICDE.2019.00224. (Visited on 11/29/2020).

[39] Muhammad Ebraheem et al. "Distributed Representations of Tuples for Entity Resolution". In: *Proc. VLDB Endow.* 11.11 (July 2018), pp. 1454–1467. ISSN: 2150-8097. DOI: 10.14778/3236187.3236198. (Visited on 07/18/2020).

[40] Ahmed K. Elmagarmid et al. "Duplicate Record Detection: A Survey". In: *IEEE Trans. Knowl. Data Eng.* 19.1 (Jan. 2007), pp. 1–16. ISSN: 1041-4347. DOI: 10.1109/TKDE.2007.250581. (Visited on 05/25/2021).

[41] Jeffrey L Elman. "Finding Structure in Time". In: *Cogn. Sci.* 14.2 (Mar. 1990), pp. 179–211.

[42] Ivan P Fellegi and Alan B Sunter. "A Theory for Record Linkage". In: *J. Am. Stat. Assoc.* 64.328 (Dec. 1969), pp. 1183–1210.

[43] Cheng Fu et al. "End-to-End Multi-perspective Matching for Entity Resolution". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. Macao, China: AAAI Press, 2019, pp. 4961–4967.

[44] Lise Getoor and Ashwin Machanavajjhala. "Entity Resolution: Theory, Practice & Open Challenges". In: *Proceedings VLDB Endowment* 5.12 (Aug. 2012), pp. 2018–2019.

[45] Leilani H. Gilpin et al. "Explaining Explanations: An Overview of Interpretability of Machine Learning". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA). Turin, Italy: IEEE, Oct. 2018, pp. 80–89. ISBN: 978-1-5386-5090-5. DOI: 10.1109/DSAA.2018.00018. (Visited on 05/08/2021).

[46] Aristides Gionis et al. "Similarity Search in High Dimensions via Hashing". In: *Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Sept. 7, 1999, pp. 518–529. ISBN: 978-1-55860-615-9.

[47] Yoav Goldberg. "Neural Network Methods for Natural Language Processing". In: *Synthesis Lectures on Human Language Technologies* 10.1 (Apr. 2017), pp. 1–309.

[48] Ian Goodfellow et al. *Deep Learning*. MIT Press, Nov. 2016.

[49] Ram Deepak Gottapu et al. "Entity Resolution Using Convolutional Neural Network". In: *Procedia Comput. Sci.* 95 (Jan. 2016), pp. 153–158.

[50] Yash Govind et al. "Cloudmatcher: A Hands-off Cloud/Crowd Service for Entity Matching". In: *Proc. VLDB Endow.* 11.12 (Aug. 1, 2018), pp. 2042–2045. ISSN: 21508097. DOI: 10.14778/3229863.3236255. (Visited on 06/21/2020).

[51] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: (Aug. 2013).

[52] Alex Graves et al. "Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition". In: *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*. Springer Berlin Heidelberg, 2005, pp. 799–804.

[53] Riccardo Guidotti et al. "A Survey of Methods for Explaining Black Box Models". In: *ACM Comput. Surv.* 51.5 (Jan. 23, 2019), pp. 1–42. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3236009. (Visited on 11/12/2020).

[54] Sairam Gurajada et al. "Learning-Based Methods with Human-in-the-Loop for Entity Resolution". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19: The 28th ACM International Conference on Information and Knowledge Management. Beijing China: ACM, Nov. 3, 2019, pp. 2969–2970. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3360316. (Visited on 11/01/2020).

[55] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (Dec. 2015).

[56] Thomas N Herzog et al. *Data Quality and Record Linkage Techniques*. Springer Science & Business Media, May 2007.

[57] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780.

[58] Piotr Indyk and Rajeev Motwani. *Approximate Nearest Neighbors*. 1998.

[59] Ekaterini Ioannou et al. "On Generating Benchmark Data for Entity Matching". In: *J. Data Semant.* 2.1 (Mar. 2013), pp. 37–56.

[60] Prateek Jain et al. "Hashing Hyperplane Queries to Near Points with Applications to Large-Scale Active Learning". In: *Advances in Neural Information Processing Systems 23*. Ed. by J D Lafferty et al. Curran Associates, Inc., 2010, pp. 928–936.

[61] Jeff Johnson et al. "Billion-Scale Similarity Search with GPUs". In: *IEEE Trans. Big Data* 7.3 (July 1, 2021), pp. 535–547. ISSN: 2332-7790, 2372-2096. DOI: 10.1109/TBDATA.2019.2921572. (Visited on 02/03/2023).

[62] Mandar Joshi et al. "BERT for Coreference Resolution: Baselines and Analysis". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Nov. 2019, pp. 5803–5808.

[63] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 2020. URL: https://web.stanford.edu/~jurafsky/slp3/ (visited on 06/06/2020).

[64] Daniel Jurafsky and James H Martin. "Speech and Language Processing: An Introduction to Speech Recognition, Computational Linguistics and Natural Language Processing". In: *Upper Saddle River, NJ: Prentice Hall* (2008).

[65] Jungo Kasai et al. "Low-Resource Deep Entity Resolution with Transfer and Active Learning". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, 2019, pp. 5851–5861. DOI: 10.18653/v1/P19-1586. (Visited on 07/20/2020).

[66] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: (Aug. 2014).

[67] T Kohonen. "Adaptive, Associative, and Self-Organizing Functions in Neural Computing". In: *Appl. Opt.* 26.23 (Dec. 1987), pp. 4910–4918.

[68] Nikolaos Kolitsas et al. "End-to-End Neural Entity Linking". In: (Aug. 2018).

[69] Pradap Konda et al. "Magellan: Toward Building Entity Matching Management Systems". In: *Proc. VLDB Endow.* 9.12 (Aug. 1, 2016), pp. 1197–1208. ISSN: 21508097. DOI: 10.14778/2994509.2994535. (Visited on 06/21/2020).

[70] Nihel Kooli et al. “Deep Learning Based Approach for Entity Resolution in Databases”. In: *Intelligent Information and Database Systems*. Springer International Publishing, 2018, pp. 3–12.

[71] Hanna Köpcke et al. “Evaluation of Entity Resolution Approaches on Real-World Match Problems”. In: *Proc. VLDB Endow.* 3.1-2 (Sept. 2010), pp. 484–493. ISSN: 2150-8097. DOI: 10.14778/1920841.1920904. (Visited on 05/11/2021).

[72] Taku Kudo and John Richardson. “SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing”. In: (Aug. 2018).

[73] Guillaume Lample et al. “Neural Architectures for Named Entity Recognition”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016, pp. 260–270.

[74] Yann LeCun et al. “Deep Learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444.

[75] Kenton Lee et al. “End-to-End Neural Coreference Resolution”. In: (July 2017).

[76] Chen Li et al. “Efficient Merging and Filtering Algorithms for Approximate String Searches”. In: *2008 IEEE 24th International Conference on Data Engineering*. 2008 IEEE 24th International Conference on Data Engineering (ICDE 2008). Cancun, Mexico: IEEE, Apr. 2008, pp. 257–266. ISBN: 978-1-4244-1836-7 978-1-4244-1837-4. DOI: 10.1109/ICDE.2008.4497434. (Visited on 03/31/2022).

[77] G Li et al. “Crowdsourced Data Management: A Survey”. In: *IEEE Trans. Knowl. Data Eng.* 28.9 (Sept. 2016), pp. 2296–2319.

[78] Peng Li et al. “Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples”. In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD/PODS ’21: International Conference on Management of Data. Virtual Event China: ACM, June 9, 2021, pp. 1064–1076. ISBN: 978-1-4503-8343-1. DOI: 10.1145/3448016.3452824. (Visited on 04/10/2022).

[79] Wen-Syan Li and Chris Clifton. “Semantic Integration in Heterogeneous Databases Using Neural Networks”. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 1–12.

[80] Wen-Syan Li and Chris Clifton. "SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks". In: *Data Knowl. Eng.* 33.1 (Apr. 2000), pp. 49–84.

[81] Yuliang Li et al. "Deep Entity Matching with Pre-Trained Language Models". In: *Proc. VLDB Endow.* 14.1 (Sept. 2020), pp. 50–60. ISSN: 2150-8097. DOI: 10.14778/3421424.3421431. arXiv: 2004.00584. (Visited on 11/07/2020).

[82] Kevin Lin et al. "Deep Learning of Binary Hash Codes for Fast Image Retrieval". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 27–35.

[83] Stan Lipovetsky and Michael Conklin. "Analysis of Regression in Game Theory Approach". In: *Appl. Stochastic Models Bus. Ind.* 17.4 (Oct. 2001), pp. 319–330. ISSN: 1524-1904, 1526-4025. DOI: 10.1002/asmb.446. (Visited on 05/20/2021).

[84] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". July 26, 2019. arXiv: 1907.11692 [cs]. URL: http://arxiv.org/abs/1907.11692 (visited on 05/02/2022).

[85] Jiaheng Lu et al. "Synergy of Database Techniques and Machine Learning Models for String Similarity Search and Join". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, Nov. 3, 2019, pp. 2975–2976. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3360319. (Visited on 02/02/2021).

[86] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[87] Willi Mann et al. "An Empirical Evaluation of Set Similarity Join Techniques". In: *Proc. VLDB Endow.* 9.9 (May 2016), pp. 636–647. ISSN: 2150-8097. DOI: 10.14778/2947618.2947620. (Visited on 03/31/2022).

[88] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C J C Burges et al. Curran Associates, Inc., 2013, pp. 3111–3119.

[89] Bhaskar Mitra and Nick Craswell. "Neural Models for Information Retrieval". In: (May 2017).

[90] Christoph Molnar. *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable*. 2019.

[91] Sidharth Mudgal et al. "Deep Learning for Entity Matching: A Design Space Exploration". In: *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*. The 2018 International Conference. Houston, TX, USA: ACM Press, 2018, pp. 19–34. ISBN: 978-1-4503-4703-7. DOI: 10.1145/3183713.3196926. (Visited on 06/06/2020).

[92] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Morgan and Claypool Publishers, 2010.

[93] H B Newcombe et al. "Automatic Linkage of Vital Records". In: *Science* 130.3381 (Oct. 1959), pp. 954–959.

[94] Hao Nie et al. "Deep Sequence-to-Sequence Entity Matching for Heterogeneous Entity Resolution". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: ACM Press, Nov. 3, 2019, pp. 629–638. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3358018. (Visited on 05/18/2021).

[95] Jordi Nin and Vicenç Torra. "New Approach to the Re-identification Problem Using Neural Networks". In: *Modeling Decisions for Artificial Intelligence*. Springer Berlin Heidelberg, 2006, pp. 251–261.

[96] K Nozaki et al. "Semantic Schema Matching for String Attribute with Word Vectors". In: *2019 6th International Conference on Computational Science/Intelligence and Applied Informatics (CSII)*. May 2019, pp. 25–30.

[97] Kevin O'hare et al. "High-Value Token-Blocking: Efficient Blocking Method for Record Linkage". In: *ACM Trans. Knowl. Discov. Data* 16.2 (July 21, 2021), pp. 1–17. ISSN: 1556-4681, 1556-472X. DOI: 10.1145/3450527. (Visited on 04/10/2022).

[98] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[99] George Papadakis et al. "A Survey of Blocking and Filtering Techniques for Entity Resolution". In: (May 2019).

[100] George Papadakis et al. "Blocking and Filtering Techniques for Entity Resolution: A Survey". In: *ACM Comput. Surv.* 53.2 (Mar. 31, 2021), pp. 1–42. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3377455. (Visited on 03/30/2022).

[101] George Papadakis et al. "Comparative Analysis of Approximate Blocking Techniques for Entity Resolution". In: *Proceedings VLDB Endowment* 9.9 (May 2016), pp. 684–695.

[102] George Papadakis et al. "How to Reduce the Search Space of Entity Resolution: With Blocking or Nearest Neighbor Search?" Mar. 5, 2022. arXiv: 2202.12521 [cs]. URL: http://arxiv.org/abs/2202.12521 (visited on 03/30/2022).

[103] Ankur Parikh et al. "A Decomposable Attention Model for Natural Language Inference". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Nov. 2016, pp. 2249–2255.

[104] Jeffrey Pennington et al. "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. aclweb.org, 2014, pp. 1532–1543.

[105] Matthew Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pp. 2227–2237.

[106] Burdette Pixton and Christophe Giraud-Carrier. "Using Structured Neural Networks for Record Linkage". In: *Proceedings of the Sixth Annual Workshop on Technology for Family History and Genealogical Research*. 2006.

[107] Kun Qian et al. "Active Learning for Large-Scale Entity Resolution". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore). New York, NY, USA: ACM, 2017, pp. 1379–1388.

[108] Kun Qian et al. "SystemER: A Human-in-the-Loop System for Explainable Entity Resolution". In: *Proc. VLDB Endow.* 12.12 (Aug. 2019), pp. 1794–1797. ISSN: 2150-8097. DOI: 10.14778/3352063.3352068. (Visited on 05/25/2021).

[109] Alec Radford et al. "Language Models Are Unsupervised Multitask Learners". In: *OpenAI Blog* 1.8 (2019), p. 9.

[110] Erhard Rahm and Philip A Bernstein. "A Survey of Approaches to Automatic Schema Matching". In: *VLDB J.* 10.4 (Dec. 2001), pp. 334–350.

[111] Jonathan Raiman and Olivier Raiman. "DeepType: Multilingual Entity Linking by Neural Type System Evolution". In: (Feb. 2018).

[112] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C Cortes et al. Curran Associates, Inc., 2015, pp. 91–99.

[113] Orion F Reyes-Galaviz et al. "A Supervised Gradient-Based Learning Algorithm for Optimized Entity Resolution". In: *Data Knowl. Eng.* 112 (Nov. 2017), pp. 106–129.

[114] Marco Tulio Ribeiro et al. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, Aug. 13, 2016, pp. 1135–1144. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939778. (Visited on 05/09/2021).

[115] Sebastian Ruder. "Neural Transfer Learning for Natural Language Processing". PhD thesis. NUI Galway, 2019.

[116] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *Int. J. Comput. Vis.* 115.3 (Dec. 2015), pp. 211–252.

[117] M Schuster and K K Paliwal. "Bidirectional Recurrent Neural Networks". In: *IEEE Trans. Signal Process.* 45.11 (Nov. 1997), pp. 2673–2681.

[118] Rico Sennrich et al. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Berlin, Germany). Stroudsburg, PA, USA: Association for Computational Linguistics, 2016, pp. 1715–1725.

[119] Z Shang et al. "K-Join: Knowledge-Aware Similarity Join". In: *IEEE Trans. Knowl. Data Eng.* 28.12 (Dec. 2016), pp. 3293–3308.

[120] Rupesh Kumar Srivastava et al. "Highway Networks". In: (May 2015).

[121] Kostas Stefanidis et al. "Entity Resolution in the Web of Data". In: *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea). New York, NY, USA: ACM, 2014, pp. 203–204.

[122] Erik Štrumbelj and Igor Kononenko. "Explaining Prediction Models and Individual Predictions with Feature Contributions". In: *Knowl Inf Syst* 41.3 (Dec. 2014), pp. 647–665. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-013-0679-x. (Visited on 05/20/2021).

[123] Zequn Sun et al. "Bootstrapping Entity Alignment with Knowledge Graph Embedding". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (Stockholm, Sweden). California: International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 4396–4402.

[124] Mukund Sundararajan et al. "Axiomatic Attribution for Deep Networks". In: *International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 17, 2017, pp. 3319–3328. URL: http://proceedings.mlr.press/v70/sundararajan17a.html (visited on 05/09/2021).

[125] Ilya Sutskever et al. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112.

[126] John R Talburt. *Entity Resolution and Information Quality*. Elsevier, Jan. 2011.

[127] Saravanan Thirumuruganathan et al. "Deep Learning for Blocking in Entity Matching: A Design Space Exploration". In: *Proceedings of the VLDB Endowment* 14 (2021). URL: https://www.vldb.org/pvldb/vol14/p2459-thirumuruganathan.pdf (visited on 09/02/2021).

[128] Saravanan Thirumuruganathan et al. "Explaining Entity Resolution Predictions: Where Are We and What Needs to Be Done?" In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics - HILDA'19*. The Workshop. Amsterdam, Netherlands: ACM Press, 2019, pp. 1–6. ISBN: 978-1-4503-6791-2. DOI: 10.1145/3328519.3329130. (Visited on 05/08/2021).

[129] Saravanan Thirumuruganathan et al. "Reuse and Adaptation for Entity Resolution through Transfer Learning". Sept. 28, 2018. arXiv: 1809.11084 [cs, stat]. URL: http://arxiv.org/abs/1809.11084 (visited on 06/24/2020).

[130] Hung Nghiep Tran et al. "Author Name Disambiguation by Using Deep Neural Network". In: *Intelligent Information and Database Systems*. Springer International Publishing, 2014, pp. 123–132.

[131] Iulia Turc et al. “Well-Read Students Learn Better: On the Importance of Pre-training Compact Models”. Sept. 25, 2019. arXiv: 1908.08962 [cs]. URL: http://arxiv.org/abs/1908.08962 (visited on 05/11/2021).

[132] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon et al. Curran Associates, Inc., 2017, pp. 5998–6008.

[133] Norases Vesdapunt et al. “Crowdsourcing Algorithms for Entity Resolution”. In: *Proceedings VLDB Endowment* 7.12 (Aug. 2014), pp. 1071–1082.

[134] Sandra Wachter et al. *Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR*. SSRN Scholarly Paper ID 3063289. Rochester, NY: Social Science Research Network, Oct. 6, 2017. DOI: 10.2139/ssrn.3063289. (Visited on 05/18/2021).

[135] Jiannan Wang et al. “CrowdER: Crowdsourcing Entity Resolution”. In: *Proc. VLDB Endow.* 5.11 (July 2012), pp. 1483–1494. ISSN: 2150-8097. DOI: 10.14778/2350229.2350263. (Visited on 06/22/2020).

[136] Jingdong Wang et al. “Hashing for Similarity Search: A Survey”. In: (Aug. 2014).

[137] Steven Euijong Whang et al. “Question Selection for Crowd Entity Resolution”. In: *Proceedings VLDB Endowment* 6.6 (Apr. 2013), pp. 349–360.

[138] D R Wilson. “Beyond Probabilistic Record Linkage: Using Neural Networks and Complex Features to Improve Genealogical Record Linkage”. In: *The 2011 International Joint Conference on Neural Networks*. July 2011, pp. 9–14.

[139] William E Winkler. “Matching and Record Linkage”. In: *Business survey methods* 1 (1995), pp. 355–384.

[140] William E Winkler. *Overview of Record Linkage and Current Research Directions*. Statistical Research Division, U.S. Census Bureau, 2006.

[141] L Wolcott et al. “Scalable Record Linkage”. In: *2018 IEEE International Conference on Big Data (Big Data)*. Dec. 2018, pp. 4268–4275.

[142] Lin Wu et al. “Cycle-Consistent Deep Generative Hashing for Cross-Modal Retrieval”. In: *IEEE Trans. Image Process.* 28.4 (Apr. 2019), pp. 1602–1612.

[143] Chuan Xiao et al. "Efficient Similarity Joins for near Duplicate Detection". In: *Proceeding of the 17th International Conference on World Wide Web - WWW '08*. Proceeding of the 17th International Conference. Beijing, China: ACM Press, 2008, p. 131. ISBN: 978-1-60558-085-2. DOI: 10.1145/1367497.1367516. (Visited on 03/31/2022).

[144] Chuan Xiao et al. "Efficient Similarity Joins for Near-Duplicate Detection". In: *ACM Trans. Database Syst.* 36.3 (Aug. 2011), pp. 1–41. ISSN: 0362-5915, 1557-4644. DOI: 10.1145/2000824.2000825. (Visited on 03/31/2022).

[145] Chuan Xiao et al. "Top-k Set Similarity Joins". In: *2009 IEEE 25th International Conference on Data Engineering*. 2009 IEEE 25th International Conference on Data Engineering (ICDE). Shanghai, China: IEEE, Mar. 2009, pp. 916–927. ISBN: 978-1-4244-3422-0. DOI: 10.1109/ICDE.2009.111. (Visited on 04/10/2022).

[146] Yujie Xing et al. "Balancing Multi-Domain Corpora Learning for Open-Domain Response Generation". In: *Findings of the Association for Computational Linguistics: NAACL 2022*. Findings of the Association for Computational Linguistics: NAACL 2022. Seattle, United States: Association for Computational Linguistics, 2022, pp. 2104–2120. DOI: 10.18653/v1/2022.findings-naacl.162. (Visited on 09/25/2023).

[147] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: (June 2019).

[148] Zhong Yang et al. "Adaptive Top-k Overlap Set Similarity Joins". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020 IEEE 36th International Conference on Data Engineering (ICDE). Dallas, TX, USA: IEEE, Apr. 2020, pp. 1081–1092. ISBN: 978-1-72812-903-7. DOI: 10.1109/ICDE48307.2020.00098. (Visited on 04/10/2022).

[149] You Li et al. "Schema Matching Using Neural Network". In: *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*. Sept. 2005, pp. 743–746.

[150] Minghe Yu et al. "String Similarity Search and Join: A Survey". In: *Frontiers of Computer Science* 10.3 (June 2016), pp. 399–417.

[151] J Zhang et al. "A Neural Network Based Schema Matching Method for Web Service Matching". In: *2014 IEEE International Conference on Services Computing*. June 2014, pp. 448–455.

[152] Wei Zhang et al. “AutoBlock: A Hands-off Blocking Framework for Entity Matching”. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. WSDM ’20: The Thirteenth ACM International Conference on Web Search and Data Mining. Houston TX USA: ACM, Jan. 20, 2020, pp. 744–752. ISBN: 978-1-4503-6822-3. DOI: 10.1145/3336191.3371813. (Visited on 03/30/2022).

[153] Chen Zhao and Yeye He. “Auto-EM: End-to-end Fuzzy Entity-Matching Using Pre-trained Deep Models and Transfer Learning”. In: *The World Wide Web Conference on - WWW ’19*. The World Wide Web Conference. San Francisco, CA, USA: ACM Press, 2019, pp. 2413–2424. ISBN: 978-1-4503-6674-8. DOI: 10.1145/3308558.3313578. (Visited on 07/20/2020).

[154] Zexuan Zhong et al. “CoLink: An Unsupervised Framework for User Identity Linkage”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. Apr. 2018.

[155] Hao Zhu et al. “Iterative Entity Alignment via Joint Knowledge Embeddings”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (Melbourne, Australia). AAAI Press, Aug. 2017, pp. 4258–4264.

ISBN 978-82-326-7938-6 (printed ver.)
ISBN 978-82-326-7937-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU
Norwegian University of
Science and Technology