Universität Zürich[UZH]

Bachelor's thesis
presented to the Faculty of Arts and Social Sciences
of the University of Zurich
for the degree of
Bachelor of Arts UZH

Evaluation of Approaches for Duplicate Detection in Swiss CRM Systems

Author: Artem Shkabruk
Student ID Nr: 19-924-810

Examiner: Dr. Jannis Vamvas

Department of Computational Linguistics

Submission date: 01.06.2024

GitHub https://github.com/ashkabruk/Bachelor-Thesis

Email artem.shkabruk@uzh.ch

Abstract

This thesis assesses the effectiveness of various duplicate detection algorithms within the complex context of the swissICT CRM system. During the review, a notable scarcity of datasets featuring German and Swiss names was identified among existing methods and benchmark datasets for duplicate detection. To address this, I developed a custom dataset tailored to provide a language-specific benchmark. The evaluation uncovered that using different n-gram similarity thresholds unveiled hidden interactions that significantly enhanced algorithm performance. Furthermore, incorporating a rule-based approach to extract and compare email prefixes with full names substantially improved the outcomes, particularly for the Simple Match algorithm, which relies on direct string comparisons. The Simple Match algorithm significantly outperformed other algorithms on the real-world dataset, indicating that preprocessing and incorporating additional rule-based approaches with more fields lead to superior results compared to other well-known duplicate detection algorithms.

Contents

1 Introduction

Duplicate data entries within Customer Relationship Management (CRM) systems represent a persistent issue, causing substantial inefficiencies and financial costs. These duplicates typically arise from users inadvertently creating multiple accounts, resulting in redundant profiles that complicate data management and analysis. Identifying and resolving these duplicates is crucial, particularly during CRM migrations where data integrity is paramount.

The challenges of duplicate detection are amplified when dealing with personal names, which are prone to variations due to grammatical errors, typos, special characters, and phonetic similarities. Addressing these challenges is essential for maintaining accurate and reliable CRM systems, which are critical for businesses to manage customer interactions, track payments, and tailor marketing strategies effectively.

This thesis is motivated by the practical challenges faced at swissICT, where the absence of an effective duplicate detection mechanism in the current CRM system has led to errors in billing, event registrations, and customer communication. The impending transition to a new CRM provider underscores the need for robust duplicate detection algorithms to ensure a smooth and error-free migration.

Despite the availability of various duplicate detection algorithms, there is a noticeable gap in research specifically addressing duplicate detection in databases with a high prevalence of German names. This thesis aims to bridge this gap by evaluating different algorithms, customizing them for German-centric databases, and constructing a comprehensive dataset for testing and validation.

In this thesis, I will explore the efficacy of phonetic-based and edit-distance-based algorithms for detecting duplicates. I will also investigate the impact of preprocessing steps on data quality and the overall performance of these algorithms. Through a series of experiments and evaluations, I aim to identify the most effective approaches for duplicate detection in CRM systems and provide practical recommendations for their implementation.

2 Motivation

At swissICT, I have frequently encountered the challenge of duplicate detection. Given my part-time role, prioritizing efficiency over extensive research has been key. Thus, I have primarily relied on simple matching between full names and emails, which has proven effective in identifying the majority of duplicates.

The proliferation of simple duplicates can be attributed to issues with the backend structure of the contact form. Slow internet connections or processing delays may prompt users to inadvertently submit the form multiple times, resulting in duplicate entries in my CRM system.

Beyond pragmatic considerations, my passion for data analysis has also spurred my interest in this area. I find immense satisfaction in identifying discrepancies within datasets and refining them to accuracy. Moreover, as I approach graduation, this research serves as my final paper. In exploring potential topics, I noticed a significant gap in research regarding duplicate identification in database systems, particularly those predominantly using the German language with its distinctive names. A quick web search for ”Duplikatserkennung Namen” confirms the existence of some programs and companies offering software solutions for duplicate detection (DeepVA (2024), Group (2021)). However, the specific methods and processes these solutions use for finding duplicates are not well-documented or transparent.

Therefore, I endeavor to address this gap by evaluating various algorithms, devising custom adaptations, and constructing a dataset for comprehensive testing. I am dedicated to providing valuable insights and methodologies that will improve duplicate detection in German-centric database systems and prepare swissICT with a strong foundation for migrating their CRM system.

3 Problem Definition

3.1 swissICT

swissICT has been advocating for ICT in Switzerland since 1955, connecting users, providers, and individuals as the sole ICT association. Today, with 2,500 members, including over 900 corporate members, swissICT boasts the largest Swiss network of ICT professionals.

With over 180 experts currently involved in its 20 swissICT specialist groups, the largest Swiss ICT network drives forward current topics and regularly invites participation in knowledge exchange through events, training sessions, and conferences.

Among swissICT's core services are the "ICT Salärstudie", defining ICT professions on the "Berufe der ICT" platform, the Digital Economy Award, the "Arbeitswelten Konferenz", the "Honorarstudie", the Digital Excellence Checkup, and the self-analysis tool ICT Career Advisory (swissICT (2024)).

3.2 Background on the Problem

Having duplicate or erroneous data within a database, particularly in a CRM system handling billing, event registrations, and contact information, can incur significant costs. Since CRMs are typically third-party systems, users are reliant on the functionalities they offer. In swissICT's case, the absence of duplicate detection in their CRM system has led to errors during invoice generation and event registration. For instance, the same person might end up receiving duplicate confirmation emails or invoices, or a member could be billed twice for the same service.

Another aspect to consider is the potential for inaccuracies in reporting and analysis, as well as user confusion regarding the correct record. While duplicate records may share the same full name and email address, they may contain conflicting information, making it challenging to determine which version to retain or delete when necessary.

Furthermore, the impending transition of swissICT to a new CRM provider underscores the necessity for a robust duplicate detection algorithm. Switching providers necessitates ensuring the completeness and accuracy of the data to mitigate additional costs and efforts associated with data migration. Failure to do so may result in extra expenses from the provider's end due to the supplementary effort required to prepare the data for migration.

3.3 CRM Structure of swissICT and Creation of Duplicates

The CRM framework utilized by swissICT is distinctive, particularly due to its diverse membership tiers. These tiers, categorized into three main groups with subcategories, share identical treatment in terms of functionality, with only pricing variations distinguishing them: "Einzel" (Single), "Kombi" (Combined), and "Firma" (Company).

A company membership entails granting memberships to all individuals within the organization. These individuals are exclusively associated with their company when registering for events and have their own contact details. However, they may miss out on certain benefits available to single members. swissICT addresses this discrepancy by offering individuals already registered under a company the option to apply for a special ”Kombi” membership, which provides similar benefits as single membership but at a reduced price.

Single membership, on the other hand, is straightforward.

Challenges can arise, particularly for ”Kombi” members. swissICT does not mandate the use of separate contact information from their company for individuals with this membership. Consequently, many individuals utilize their company contact details, including address, email and phone number. Furthermore, it is not clear, whether an individual utilizes a service as a ”Firmen” member or ”Kombi” member, since the email address can be the same. Often, individuals from the same company use the same phone number for all individuals, further complicating the detection of duplicates based on these fields and resulting in a high number of false positives.

While a more sophisticated algorithm could be developed, incorporating additional exceptions and rules, I aim to maintain replicability for other CRMs or database systems. The complexity of the CRM structure magnifies the occurrence of unintended duplicates due to the flexible membership structure.

Clear instructions and rules on contact information can significantly reduce the incidence of duplicate creation.

In addition to the primary reasons for duplicate creation outlined when applying for membership, there are further factors at play.

As detailed in 3.1, swissICT not only offers memberships but also sells various items on its website. While membership is not mandatory for purchasing these items, a login is required. Members already possess login credentials, whereas non-members can create new logins, which do not necessitate the completion of all the contact information typically required for membership applications. Notably, a phone number is not mandatory for these logins.

Challenges arise when individuals, already registered as members under a company, are unaware of their existing membership status and seek to make purchases from the online shop. In such cases, they create new logins using the same email address under which their company registered them as members, inadvertently generating new duplicates. The backend does not check, whether a new email address already exists in the CRM system. While these instances may not strictly qualify as duplicates, as individuals intend to make private purchases rather than transactions on behalf of their company, ideally, individuals should use a personal email address to register, thereby avoiding complications as discussed in 3.2.

Furthermore, it is possible for individuals to forget their previous login credentials and create new logins, resulting in genuine duplicates.

At swissICT, employees have the capability to manually input individuals into the CRM system. In instances where an individual forgets their login credentials and reaches out to swissICT support for as-

sistance, there is a potential for erroneous data entry due to the conversion from speech to text, since only the email and password have to be correct to successfully log in. This transition can introduce phonetic equivalents and typographical errors, leading to inaccuracies in the CRM records. In the scenario where your name is ”Filip Neuhaus” and your email prefix is ”neuhaus,” there is a possibility that an employee could erroneously type your name as ”Philipp Neuhaus.”

4 Literature Overview

Nauman and Herschel (2022) provides a comprehensive introduction to the most prevalent algorithms used in duplicate detection. This resource is invaluable as it outlines various algorithmic categories, including token-based, edit-based, and phonetic-based approaches. The text extensively covers the implementation and evaluation of these algorithms. Moreover, it highlights significant gaps in the availability of common datasets and documentation for these algorithms, which has prompted me to develop a custom dataset and undertake the implementation of these algorithms myself.

Elmagarmid et al. (2007) discusses the application of machine learning techniques in duplicate detection. This exploration revealed a notable scarcity of training and testing datasets specifically for German full names, which consequently influenced my decision to avoid these machine learning approaches due to the lack of suitable data.

Wilz (2005) conducts empirical tests on various phonetic algorithms tailored for German names and assesses their effectiveness. Based on his findings, I have chosen to implement Kölner Phonetik, as it was among the best performers in the phonetic category evaluated in his study. For the purpose of comparison, I also implemented both the standard Soundex algorithm and a customized version of Soundex.

Leskovec et al. (2020) explores Locality Sensitive Hashing (LSH) and its efficacy in conjunction with Jaccard similarity. This discussion is particularly relevant as it presents LSH as a highly efficient alternative to the computationally intensive $O(n^2)$ implementations traditionally used in n-gram Jaccard and Cosine similarity calculations. The insights from this research have significantly influenced the methodologies employed in my project to enhance the efficiency of similarity computations.

5 Understanding Duplicates in Data

Duplicates in the realm of data management denote occurrences where two or more records within a dataset exhibit identical or markedly similar characteristics. Not all entries labeled as duplicates truly represent the same individual. Consider the name ”Peter Müller,” a combination of common first and last names. A database can contain multiple entries for Peter Müller, but it is unclear whether they all refer to the same person. To distinguish between individuals, I could potentially use additional information such as phone numbers, email addresses, or physical addresses. However, certainty remains elusive. For instance, an individual might have registered in a CRM system ten years ago, forgotten their login details, and created a new account two years later. Over such a period, it is plausible that their email address, phone number, and residence could have changed.

Due to the significant effort involved and the scarcity of real-world examples, I will not include such data in the custom dataset. These cases are exceptions and should ideally be resolved manually.

The ensuing subsections outline various types of duplicates, which have been created as mentioned in

3.3, prevalent in the swissICT dataset, accompanied by illustrative examples and effective methodologies for detection.

In this thesis, the primary focus will be on the use of full names for the detection of duplicates. Names are the most common input for duplicate detection tasks, particularly evident in popular English-language duplicate datasets such as CORA and the RESTAURANT dataset. Focusing on full names enhances the replicability of the methods discussed. Furthermore, the identification of duplicate and erroneous names is of utmost importance. For instance, when issuing invoices, ensuring the accuracy of both first and last names is critical. Although the accuracy of addresses is also crucial, detecting faulty addresses poses significant challenges due to the lack of reliable verification methods.

Section 3.2 further discusses why phone numbers were not utilized for detection purposes. Additionally, I will conduct experiments to detect duplicates both with and without considering email fields.

5.1 Simple Duplicates

Simple duplicates manifest when the first name and last name precisely match between multiple instances within a dataset. These duplications can stem from diverse sources such as data entry errors, system glitches, or duplicate records sourced from disparate origins.

- Christian Streich appears multiple times in the dataset.

For detecting simple duplicates, straightforward comparison across all entries in the dataset suffices.

5.2 Typos

Typos emerge when one or more characters within a name are erroneously entered or misspelled, often due to human oversight during data input or transcription, thereby introducing inconsistencies.

- ”Robert” may be mistyped as ”Rober” or ”Robett”.
- ”Ammann” might be erroneously transcribed as ”Ammanm”.

For identifying typos, employing edit distance measures like Levenshtein Distance is effective albeit computationally intensive, as elaborated in 10.1.

5.3 Phonetic Equivalents

Phonetic equivalents denote names that sound alike when pronounced, despite differences in spelling.

- ”Christian” and ”Kristian”.
- ”Michelle” and ”Michel”.

Detecting these duplicates can be challenging due to varying pronunciations across languages. Utilizing rule-based phonetic similarity algorithms like Kölner Phonetik aids in identifying most duplicates, albeit with potential false positives.

5.4 Middle Names

Middle names introduce complexity in duplicate detection, particularly when one instance includes a middle name while the other does not.

- ”Marie Lena Ammann” may be duplicated as ”Marie Ammann” or ”Lena Ammann”.
- These middle names can appear shortened and similar challenges apply to double surnames.

A rule-based approach, segmenting names and employing simple matching or more sophisticated algorithms, suffices for detecting such duplicates.

5.5 Composite Names

Composite names comprise multiple words, such as ”van Hausen” or ”de la Cruz,” posing challenges in duplicate identification due to variations in substrings and spacing.

- ”Frederik van Hausen” may appear as ”Frederik Hausen”.

Similar to middle names, a rule-based approach suffices for detection.

5.6 Recurring Names

In languages like German, where ”Name” can refer to the full name, first name, or last name, duplicates may arise where the first name recurs redundantly within the last name field.

- ”Johann Basten” may appear as ”Johann Johann Basten” or ”Johann Basten Basten”.

Similar to approaches for middle names and composite names, a rule-based strategy is effective for detection as well as n-gram similarity.

6 Idea

There are several algorithms in finding duplicates, some popular ones are listed under sections 9 and 10, yet it is unclear which metrics and techniques are the current state-of-the-art (Elmagarmid et al. (2007)). Elmagarmid et al. (2007) also states, that the absence of standardized, extensive benchmark datasets presents a significant barrier to the continued advancement of the field. Without such datasets, it becomes exceedingly challenging to effectively compare and evaluate new techniques against established ones.

The objective is to construct a proprietary dataset tailored for benchmarking prevalent duplicate detection algorithms designed for identifying name duplicates. This dataset aims to mirror the characteristics of a standard Swiss/German database, albeit on a smaller scale.

In the process of selecting algorithms, Wilz (2005) conducted benchmarks of phonetic-based algorithms, as illustrated in Figures 3 and 4. I plan to combine these insights with other widely used methods to conduct a comprehensive evaluation on real-world data from a CRM system.

7 Datasets

7.1 swissICT Dataset

This dataset includes all individuals currently registered through a form linked to the main webpage of swissICT. It comprises 21,998 entries, of which 3,574 denote an entity labeled "[Zentrale]", indicating a company's headquarters. Additionally, the dataset includes 241 entries missing a first name, 200 entries without a last name, and 938 entries lacking an email address. As illustrated in Figures 1 and 2, the dataset predominantly consists of names that are Swiss or German in origin.

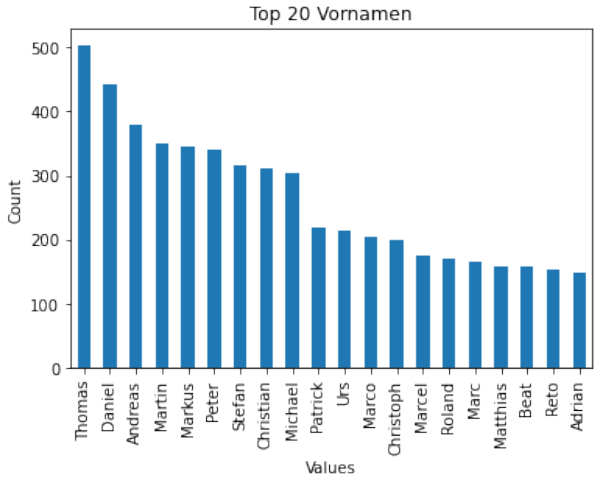

Figure 1: Top 20 most common first names in the swissICT dataset

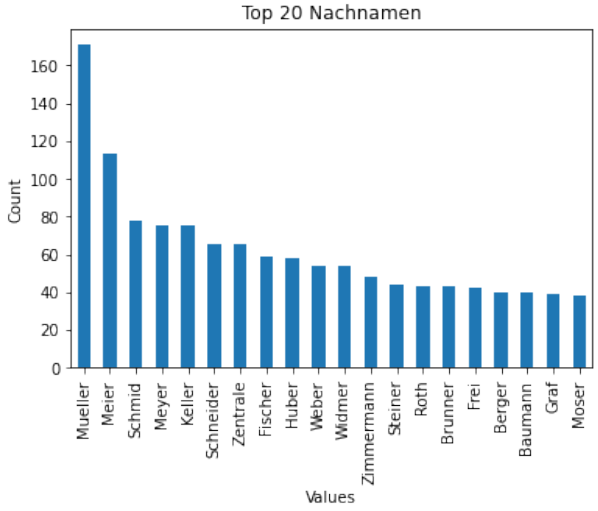

Figure 2: Top 20 most common last names in the swissICT dataset

7.2 Custom Dataset

How should I evaluate the effectiveness of different duplicate detection algorithms? Numerous English-language datasets designed for this purpose exist, yet they fail to capture the nuances of my specific

use case. The CORA dataset, which includes bibliographic information about scientific papers, comes closest to resembling my scenario. However, its ”authors” column often lists multiple names and spans a variety of ethnic backgrounds. This is a significant deviation from my focus on databases typical in Swiss companies, which predominantly feature Swiss or German names. Using the CORA dataset to benchmark the Kölner Phonetik algorithm would not accurately demonstrate its efficacy in processing the common German name structures found in such environments. Consequently, there is a clear need for a custom dataset that more accurately reflects the ethnic composition of Switzerland, focusing on individual full names.

This custom dataset aims to mirror a Swiss name dataset, reflecting the linguistic diversity observed in Switzerland. According to Statista, approximately 62% of the population in Switzerland speaks German as their primary language, followed by 22% speaking French and 8% speaking Italian. While language preference does not necessarily dictate ethnic background, it serves as a reasonable proxy for name replication. Therefore, the dataset maintains similar proportions, with the remaining 8% allocated to more exotic names. However, there is a concern that the allocation for exotic names might be underestimated.

Name Generation Process: The names in this dataset were generated through a multi-step process. Initially, names were sourced from the SwissICT dataset, then anonymized manually. Additionally, a fake name generator (Random Name Generator) was utilized to diversify the pool of names further.

Email Data Analysis: Email addresses serve as valuable indicators to detect potential errors in name entries or email formatting. For instance, discrepancies such as ”Filip Neuhaus” associated with an email prefix like ”philipp.neuhaus” can signal inaccuracies.

In order to fabricate fake emails corresponding to fabricated names, I conducted a comprehensive analysis of the SwissICT dataset, extracting 100 random samples. Through this sampling process, I identified recurring patterns in email structures. These patterns typically consist of combinations involving the first name (”vor”), last name (”nach”), or their initials (”v” for the first letter), alongside common prefixes such as ”info” for firm addresses or specific inboxes like ”media@” or ”support@”. Moreover, numerical values (”[number]”) are sometimes appended to the email addresses, as seen in examples like ”philipp.neuhaus1990” or ”philipp.neuhaus1”.

Format	**Count**
vor.nach	55
vnach	10
vor	9
v.nach	6
others	6
info	5
nach	4
vornach	3
vor.nach[number]	2

Table 1: Counts of different email formats in the swissICT dataset by taking 100 random samples

The dataset amounts to roughly 225 entries, 65 of which are duplicates. The different types of duplicates I incorporated are listed in section 5.

When using smaller test sets, the variability in accuracy measurements can increase due to the limited number of examples, which might not adequately represent the overall distribution of the data. Conversely, larger test sets tend to provide more stable and generalizable results, as they are more likely to capture the diversity of cases, including various types of duplicates and unique entries.

As I manually input data into the dataset, it becomes increasingly important to accurately denote whether each new entry is a duplicate, and if so, to identify its location. The process of adding new entries grows more time-consuming as the dataset expands, which in turn places greater demands on my resources. This increasing complexity underscores the need for efficient data management practices.

8 Preprocessing

Data preprocessing encompasses various actions such as manipulation, filtration, or augmentation applied to data prior to analysis, and it frequently serves as a critical phase in the data mining process. Due to loosely controlled data collection methods, anomalies such as out-of-range values, infeasible data combinations, and missing values commonly arise (Tableau (2021)).

The preprocessing pipeline deployed can significantly impact the conclusions derived from subsequent analyses. Therefore, ensuring the representation and quality of data is imperative prior to initiating any analytical procedures (Pyle (1999)).

Since the resulting preprocessed names will be compared with each other, a standardized format for all names is needed.

In the preprocessing phase for this thesis, I employed a series of techniques to enhance the quality and consistency of the data. Initially, I applied the ”unidecoded” function to address any potential encoding issues present in the data. This function systematically replaced accented characters, such as ”ä,” ”ö,” ”ü,” ”Ä,” ”Ö,” and ”Ü,” with their respective ASCII equivalents (”ae,” ”oe,” ”ue,” ”Ae,” ”Oe,” and ”Ue”). In the subsequent stage, I installed the ”unidecode” Python library (Tomaz Solc (2009)). This library's function replaced all non-ASCII characters in the text with their corresponding representations. It was necessary to manually transcribe the Umlaut characters beforehand, as the unidecode library only transformed them into single-letter equivalents (”ä” to ”a”).

Following the unidecoding step, I implemented the ”replace_non_letters” function to further refine the data. This function was designed to delete any non-alphabetic characters, effectively removing punctuation marks, special symbols, and numerical values. Hyphens are an exception to this, since they can serve to connect two names like ”Hans-Peter”, so they are replace by a whitespace. By focusing solely on alphabetic characters and whitespace, I ensured that the text data remained consistent and suitable for subsequent analyses.

Ultimately, the strings were converted to lowercase letters exclusively. This step ensured uniformity across the dataset, minimizing discrepancies caused by case sensitivity. These preprocessing procedures were applied to the first name, last name, and email prefix, resulting in a standardized format suitable for subsequent tasks.

Since my primary focus is on working with the full name, it is crucial to determine the most effective

format for this purpose. I considered two main approaches: splitting the name into first and last names, then running an algorithm on each component separately, or keeping the name as a single, continuous string. Each approach has its own merits and potential drawbacks. If I split the name into first and last names, I can apply algorithms specifically optimized for each part, potentially increasing the accuracy of detection for certain types of duplicates, such as those involving typos or phonetic variations in either the first or last name. However, this approach complicates the comparison process, as it requires matching two separate strings for each name entry, which can be computationally intensive.

On the other hand, keeping the name as a full string simplifies the comparison process by treating the name as a single entity. This approach ensures that all variations and duplications are captured in one pass, reducing computational overhead. The primary consideration then becomes how to handle spaces within the full name. Removing all spaces could lead to confusion and false positives, as names like ”John Smith” and ”Johns Mith” would be treated identically. Therefore, I opted to retain spaces but ensure consistent formatting by replacing any multi-character spaces with a single space.

Thus, the inputs for the different algorithms were formatted as ”first_name last_name”. This format strikes a balance between simplicity and accuracy, allowing my algorithms to process the names effectively while preserving the inherent structure and separation of the first and last names. Additionally, I tested both the proposed format and a format without spaces on the custom dataset. The results indicated that Soundex, in particular, performed better with the proposed format. According to Christen (2012), maintaining a consistent and simple structure in data preprocessing enhances the performance of record linkage and duplicate detection algorithms by reducing complexity and potential sources of error. This standardized format is optimal for subsequent tasks, ensuring that the data remains consistent and comparable across all stages of processing.

9 Phonetic Based Similarity

Wilz (2005) analyzed various phonetic-based algorithms, examining their performance in identifying positives (see Figure 3) and their error rates (see Figure 4) among last names.

When applying phonetic algorithms, I can observe surjective behavior. For example, Soundex maps different names that sound similar to the same code: ”Robert” and ”Rupert” to ”R163”. In this case, the phonetic algorithm’s function from the set of names to the set of Soundex codes is surjective because each Soundex code corresponds to at least one name, demonstrating how different names can be considered similar based on their phonetic representations. The same goes for other phonetic algorithms.

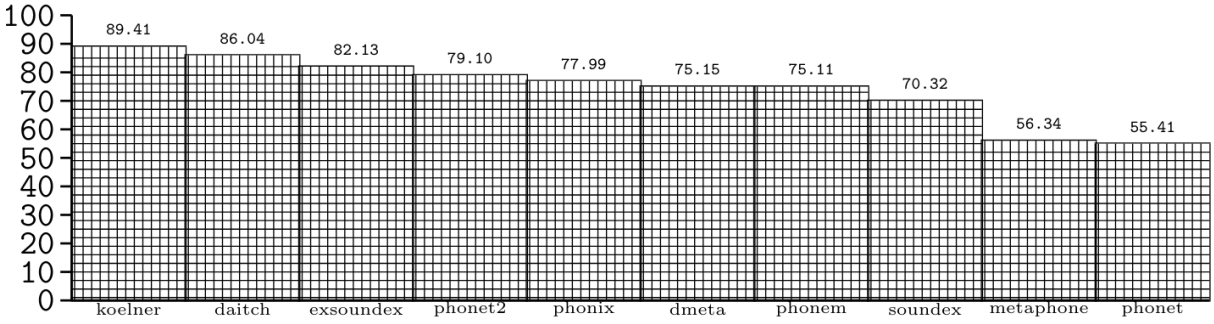

Figure 3: Averaged percentage of names classified as correct for the 100 most frequent last names (Wilz (2005)).

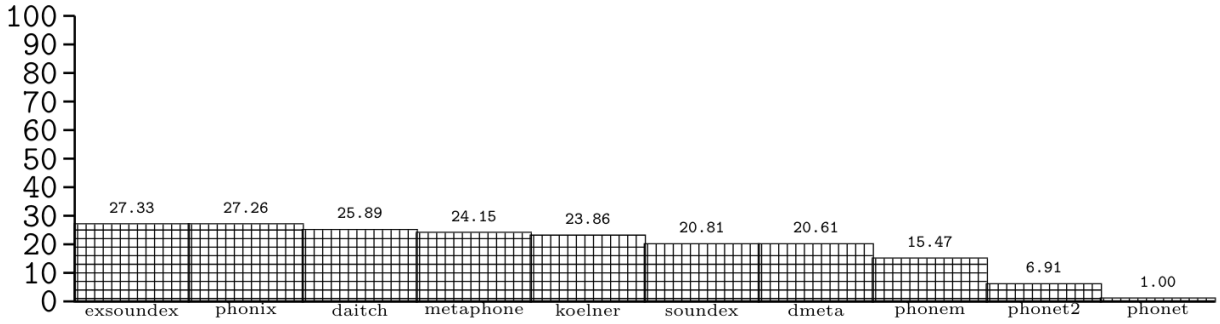

Figure 4: Error rate of the analysed algorithms in % for the 100 most frequent last names (Wilz (2005))

9.1 Soundex

Misspellings of terms can pose significant challenges for database designers, particularly when dealing with names, which can vary in length, have unconventional spellings, and lack uniqueness. Swiss names, influenced by diverse ethnic origins and having four national languages, often share pronunciations but exhibit spelling variations, and vice versa. Similarly, words may be misspelled or possess multiple spellings, especially across cultural or national boundaries.

To address this issue, phonetic algorithms play a crucial role in identifying similar-sounding terms and names. One such family of algorithms, known as Soundex algorithms, has been developed for this purpose (Knuth (1973)).

Soundex algorithms are designed to convert written words, particularly personal names, into coded character strings that encapsulate phonetic similarities. The primary objective of the Soundex algorithm is to facilitate the indexing of names by sound, as pronounced in English, to enhance search capabilities within databases. The algorithm achieves this by mapping each word to a code consisting of a letter followed by numeric digits.

The process begins with the first letter of the word being retained as the leading character in the resultant code, which serves to anchor the phonetic representation to the original word's initial sound. Subsequent letters are then transformed based on a predefined set of rules that assign numeric values to letters based on their phonetic characteristics. These rules are designed to group phonetically similar sounds under the same digit. For instance, the letters B, F, P, and V are encoded as the number 1 because they share similar bilabial or labiodental phonetic properties.

Consonants sharing similar places of articulation, such as the labials, dentals, or velars, are given the same number, while vowels and certain other consonants like H and Y are typically ignored unless they are the first letter. This coding reduces different spellings of similar sounding names to a uniform representation, facilitating the identification of names that sound alike but are spelled differently.

Moreover, the Soundex code typically truncates or pads the resultant string to ensure a fixed length, generally four characters. This standardization allows for the systematic comparison and retrieval of phonetically similar names within large datasets, thereby enhancing the efficiency of database searches focused on auditory equivalence rather than orthographic exactness.

Group XS (2018) provides a Soundex-table for the German language as seen in Table 2. I enhanced the German Soundex table by incorporating ”sch” and ”sh” as a seven, aiming to improve the algorithm's accuracy on diverse datasets, including unconventional names. This adjustment resulted in enhanced performance on the custom dataset.

Code	Letters
0	a, e, i, o, u, y, j, h
1	b, p, v, w, f
2	c, g, k, q, x, s, z
3	d, t
4	l, t
5	m, n
6	r
7	sch, ch, sh

Table 2: Soundex table for German language

9.1.1 Custom Soundex

The traditional Soundex algorithm encodes strings into a limited format of one letter followed by three numerical digits, which represents a significant limitation for strings that predominantly consist of more than twelve characters. As discussed in Section 13, this limitation often leads to a high rate of false positives when dealing with longer strings.

To mitigate this issue, I developed a variant of the Soundex algorithm, referred to as *Custom Soundex* in this paper, which retains the fundamental phonetic encoding mechanism but extends the encoding length to better accommodate longer names. This adjustment aims to reduce collisions and improve the differentiation between similar-sounding but distinct names and the original mapping stays the same.

9.2 Kölner Phonetik

The ”Cologne Phonetic” method is a phonetic algorithm developed by Postel (1969). This method aims to assign a phonetic code, a sequence of numbers, to words based on their pronunciation. Similar to Soundex, it facilitates a similarity search in search functions by assigning the same code to words that sound alike.

The algorithm involves assigning characters to numbers, with at most one letter used as context for selecting the respective digit. This letter can be positioned on both sides of the evaluated character. Special rules apply for word beginnings. While vowels are only rudimentarily considered, umlauts and the ”ß” are omitted due to the method's early development.

Despite its origins, the ”Cologne Phonetic” method remains relevant, particularly in public administration tenders. However, its dissemination beyond this sector has been limited. Nevertheless, it presents a more tailored approach to the German language compared to the widely known Russell Soundex method (Wilz (2005)).

Zeichen	Kontext	Symbol
A, E, I, J, Y, O, U, H		0
B, P		1
D, T	nicht vor C, S, Z	2
F, PH, V, W		3
G, K, Q		4
C	im Anlaut, vor A, H, K, L, O, Q, R, U, X ansonsten, vor A, O, U, H, K, X, Q	4
X	wenn nicht nach C, K, Q	48
L		5
M, N		6
R		7
S, Z		8
C	im Anlaut, nicht vor A, H, K, L, O, Q, R, U, X ansonsten, nicht vor A, O, U, H, K, X, Q nach S, Z	8
D, T	vor S, C, Z	8
X	nach C, K, Q	8

Table 3: Kölner Phonetik Regelwerk. In my own implementation, I classified ”H” as ”0”. This led to slightly better results.

9.3 Other Phonetic Algorithms

Soundex, the inaugural phonetic algorithm developed and patented between 1918 and 1922, remains a staple feature in popular database software even after nearly a century. Over time, various enhancements and adaptations have emerged, giving rise to a multitude of phonetic similarity-based algorithms such as Extended Soundex, Daitch-Mokoff (Mokotoff (2003)), Metaphone (Philips (1990)), Phonix (Gadd (1990)), Phonet und Phonet2 (Michael (1988)), PHONEM (Wilde and Meyer (1988)), and numerous others. While these methods are predominantly tailored for English words and names, incorporating more intricate rules akin to the Kölner Phonetik, they are not as well-suited for the nuances of the German language.

In a detailed examination of performance, Wilz (2005) found that Kölner Phonetik outperformed other methods in identifying plausible positive candidates while maintaining a comparatively lower error rate.

10 Token and Edit Based Similarity

10.1 Levenshtein

Levenshtein distance, also known as edit distance, is a metric used to measure the difference between two sequences. It calculates the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. This metric is particularly useful in applications such as spell checking, DNA sequence analysis, and natural language processing (Levenshtein (1965)).

Matrix Representation The computation of Levenshtein distance involves constructing a matrix D where $D[i][j]$ represents the edit distance between the first i characters of the first string and the first j characters of the second string. The matrix is filled using the following recurrence relations:

- $D[i][0] = i$ for all i
- $D[0][j] = j$ for all j
- $D[i][j] = \min \begin{cases} D[i-1][j] + 1 \\ D[i][j-1] + 1 \\ D[i-1][j-1] + (s1[i] \neq s2[j]) \end{cases}$

10.2 N-grams

N-grams are a powerful tool in text analysis, particularly useful in the context of comparing full names. An n-gram is a contiguous sequence of n characters extracted from a string. This method is particularly effective for duplicate detection in names due to its ability to capture and compare overlapping segments of text, which might vary slightly due to typographical errors or different transliterations.

When applied to full names, n-grams allow for a granular comparison by breaking down each name into smaller components. For example, the bigrams for "Albert Schmitt" include 'Al', 'lb', 'be', 'er', 'rt', 't ', ' S', 'Sc', 'ch', 'hm', 'mi', 'it', and 'tt'. These segments help in identifying names that are phonetically or typographically similar, aiding in the detection of potential duplicates even when there are minor discrepancies in spelling.

10.3 N-gram Cosine Similarity

Cosine similarity is a measure used in the fields of information retrieval and computational linguistics to determine the cosine of the angle between two vectors in an inner product space. This metric evaluates the cosine of the angle between two vectors, projecting their point product over the magnitude of the vectors, which is representative of their orientation, not magnitude (Nauman and Herschel (2022)).

In the application of n-gram cosine similarity, each text is converted into a vector of n-grams. The similarity between two texts is then computed as the cosine of the angle between these two n-gram vectors, mathematically expressed as:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|}$$

where A and B are the vector representations of the n-grams for the two compared texts. This calculation involves taking the dot product of the vectors A and B and dividing it by the product of their norms (i.e., the lengths of the vectors). This approach is especially effective in identifying similarities between texts by measuring the orientation of their n-gram vectors, making it suitable for applications that require precision in textual analysis, particularly where the text length varies significantly.

10.4 N-gram Jaccard Similarity

The Jaccard similarity index, a well-established metric in the realms of information retrieval and computational linguistics, measures similarities between finite sets and is defined as the size of the intersection divided by the size of the union of the sample sets (Leskovec et al. (2020)).

In the practical application of n-gram Jaccard similarity, each piece of text is transformed into a set of n-grams, and the similarity between any two texts is quantified by calculating the Jaccard index between their respective n-gram sets. This process is formally represented by:

$$\text{Jaccard Similarity} = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are the sets of n-grams for the two compared texts. By comparing these sets, it is possible to determine the similarity based on the overlap of n-gram occurrences, providing a robust method for assessing textual similarity which is particularly resistant to noise and minor variations in word usage or spelling.

10.5 Thresholding in Similarity Assessments

Setting an appropriate threshold for Jaccard and cosine similarities is essential for accurate duplicate detection within textual datasets. Through empirical evaluation, I computed the Jaccard similarities for all pairs in the custom dataset, disregarding any scores below 0.6 to minimize false positives. A threshold of 0.65 was established based on the lowest similarity score among verified duplicates. A similar process was applied for cosine similarity, resulting in a minimal acceptable score of 0.67. Given the proximity of these scores, I standardized the threshold at 0.65 to maintain a stringent criterion for duplicate identification, thereby reducing the risk of incorrect tagging.

10.6 Challenges with Transitive Similarity

A notable challenge encountered in using Jaccard and cosine similarity for duplicate detection involves the potential for transitive similarity relationships, which can complicate the identification of true duplicates. For instance, consider the case of the names ”Philipp Neuhaus”, ”Filip Neuhaus”, and ”Filip Newhaus”. While ”Philipp Neuhaus” and ”Filip Neuhaus” may demonstrate a high Jaccard similarity, as might ”Filip Neuhaus” and ”Filip Newhaus”, it does not necessarily imply that ”Philipp Neuhaus” and ”Filip Newhaus” share a direct, significant similarity. For phonetic-based similarity, each word was transformed into its phonetic representation using algorithms such as Soundex or Metaphone. Words that fully matched in their phonetic representations were classified as duplicates, and connections were established based on their indices. However, with the use of n-gram Jaccard and cosine similarities, the duplicate classification relies on surpassing a predefined similarity threshold rather than an exact phonetic match.

11 Full Algorithm Implementation

Fortunately, the CRM system provided by swissICT enables the downloading of the database as a single Excel file, which simplifies the process of extracting data and loading it using Python’s ’pandas’ library.

The steps in section 8 are undertaken for the first name, last name and email address.

If the full name conforms perfectly to a format outlined in Table 1, with the exception of ”others” and ”info”, and that resulting format is equal to or the full name is in the corresponding email prefix, then the full name will be employed for string comparison. In cases where a perfect match is not achieved, the trigram cosine similarity between the email prefix and full name will be computed. If the similarity

exceeds 0.65, both the full name and email address prefix will be utilized. Conversely, if the similarity falls below 0.65, only the full name will be considered for string comparison. For instance, in the scenario of ”Filip Neuhaus” with the email prefix ”philipp.neuhaus”, the prefix string will be subjected to comparison after standardizing it to match the full name format. Email addresses in the swissICT database are generally reliable because membership registration requires email confirmation. Although event registration does not mandate a functional email address, instances of incorrect email entries in the CRM system are infrequent. Conversely, for ”Christian Streich” with the email prefix ”support”, only the full name will be further analyzed. Even if a full name appears twice with differing email addresses, employing the email prefix as the comparison string in one instance should ensure that both Jaccard and cosine similarity thresholds are sufficiently low to effectively identify their duplicate significance.

Following these steps, the various algorithms are applied, with details of their specific implementations available in the accompanying code.

12 Evaluation

Assessing duplicates poses significant challenges, especially in accurately quantifying them. A key question is whether to consider a duplicate as identified upon its initial detection or only after all associated duplicates have been located. Essentially, this involves deciding whether a name should be classified simply as a duplicate, or as a duplicate specifically linked to instances with IDs x, y, z. Additionally, there is the issue of handling incorrect detections.

To navigate this complexity, I rely on evaluation metrics to gauge the performance of algorithms in predicting and identifying duplicates. Among the most widely used metrics for this purpose are accuracy, precision, recall, and the F1 score.

The choice of metric depends heavily on the specific task at hand. While accuracy has traditionally been a popular choice due to its simplicity, it may not provide a complete picture of performance. To compute the other metrics—precision, recall, and the F1 score—I need to establish criteria for categorizing True Positives, True Negatives, False Positives, and False Negatives. Additionally, incorporating unique identifiers (IDs) for duplicate entries, such as those found in the “Duplikat mit ID” column of my custom dataset, becomes crucial for correctly referencing and evaluating duplicates. These IDs streamline the process of assessing whether all duplicates have been correctly identified, enhancing the accuracy of my evaluations.

12.0.1 Impact of Transitive Similarity on Evaluation

An important consideration in evaluating duplicate detection algorithms is the concept of transitive similarity. Transitive similarity refers to the phenomenon where if text A is similar to text B, and text B is similar to text C, then it might be inferred that text A is similar to text C. This transitive property can influence the evaluation of similarity algorithms differently.

In phonetic-based algorithms, transitive similarity is generally less of an issue because the algorithms depend on exact phonetic matches. If ”Robert” and ”Rupert” both encode to the same phonetic representation, the relationship is clear and direct. However, token-based and edit-distance-based algorithms, which use thresholds, must handle transitive similarity more carefully.

For example, consider the names ”Philipp Neuhaus”, ”Filip Neuhaus”, and ”Filip Newhaus”:

- ”Philipp Neuhaus” and ”Filip Neuhaus” might exceed the similarity threshold due to their high n-gram overlap.
- ”Filip Neuhaus” and ”Filip Newhaus” might also exceed the similarity threshold.
- However, ”Philipp Neuhaus” and ”Filip Newhaus” might not directly exceed the similarity threshold despite the transitive similarity through ”Filip Neuhaus”.

This scenario highlights a challenge in token-based and edit-distance-based approaches: indirect relationships might not be captured if only direct similarity scores are considered. Consequently, evaluating and fine-tuning thresholds requires careful consideration to balance the detection of true duplicates and the prevention of false positives.

12.1 Evaluation Criteria and Algorithm Selection

The selection of a similarity algorithm and the establishment of appropriate thresholds critically influence the success of duplicate detection efforts. Phonetic-based algorithms excel in identifying phonetically similar entries, which makes them particularly effective for datasets where pronunciation plays a crucial role. Conversely, token-based and edit-distance-based algorithms are adept at managing partial and near-miss matches, making them more appropriate for datasets characterized by minor textual variations.

In summary, while phonetic-based algorithms achieve precision by identifying exact matches, token-based and edit-distance-based approaches offer greater adaptability and robustness by employing threshold-based classifications.

Here is how I counted and defined the evaluation components:

- **True Positive (TP):** If an algorithm has correctly identified a duplicate entry.
- **True Negative (TN):** If an algorithm has correctly identified a non-duplicate entry.
- **False Positive (FP):** If an algorithm has incorrectly identified a non-duplicate entry as a duplicate.
- **False Negative (FN):** If an algorithm has failed to identify a duplicate entry.

In order to calculate the efficiency and precision of the algorithms, I used precison, recall (Carterette (2009)) and F1 Score (Zhang and Zhang (2009)):

12.2 Metrics Calculation

- **Precision** measures the proportion of true positive instances among the instances predicted as positive. It is calculated as:
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
- **Recall** measures the proportion of true positive instances that were correctly identified. It is calculated as:
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1 Score** is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For each prediction, I computed the metrics and then calculated the average across all predictions.

12.3 Time Complexity

When evaluating different similarity algorithms, it is crucial to understand their time complexities, as this impacts their performance and scalability. In Table 4, I outline the time complexities for Simple Match, Soundex, Kölner Phonetik, Levenshtein Distance, and n-gram Jaccard Similarity.

Algorithm	**Time Complexity**	**Explanation**
Simple Match	$O(n)$	Character-by-character comparison.
Soundex and Custom Soundex	$O(n)$	Phonetic encoding based on character processing.
Kölner Phonetik	$O(n)$	Phonetic encoding for German, similar to Soundex.
Levenshtein Distance	$O(n \times m)$	Uses dynamic programming to fill an $n \times m$ matrix.
N-gram Cosine Similarity	$O(n^2)$	Pairwise comparisons between sets of n-grams.
N-gram Jaccard Similarity	$O(n^2)$	Pairwise comparisons between sets of n-grams.

Table 4: Time complexities and explanations of various similarity algorithms.

These implementations are my own, and there may be opportunities to optimize their time complexity further. Given that I programmed them in Python, which is known for being one of the slower modern programming languages (Heer (2022)), optimization is particularly important. Optimizing Levenshtein distance is particularly challenging because it requires constructing an $n \times m$ matrix (Nauman and Herschel (2022)), where n and m are the lengths of the two strings being compared. Although the worst-case time complexity of Jaccard similarity is $O(n^2)$, leveraging MinHash (more on that in section 12.3.1) allows for significantly faster calculations. For example, when applied to the swissICT test dataset containing 22,000 entries, the computation completed in approximately 18 seconds. The MinHash algorithm, implemented in the "datasketch" library, provides an approximate estimation of the Jaccard similarity. While this estimation is highly accurate, it is not error-free.

12.3.1 Jaccard & MinHash

A significant correlation exists between MinHashing and the Jaccard similarity of the sets undergoing MinHashing. Specifically, the likelihood that a MinHash function, applied to a randomly permuted set of rows, will yield identical values for two sets precisely matches their Jaccard similarity (Leskovec et al. (2020)). The MinHash algorithm, as implemented in the "datasketch" library (Zhu (2023)), offers an approximation of this similarity. Although this approximation is generally precise, it is not devoid of error. The error inherent in MinHash estimates is inversely related to the square root of the number of permutations (num_perm), with the standard deviation of the estimate described by $\frac{1}{\sqrt{num\_perm}}$.

Enhancing the number of permutations refines the accuracy of the similarity measurement. For example, employing 128 permutations results in a standard deviation of approximately 0.088 (Leskovec et al. (2020)). Nonetheless, this improvement in accuracy comes at the expense of increased computational time and memory usage. For my analysis, I will employ 128 permutations to balance accuracy with resource utilization.

13 Results

This section presents the performance of various duplicate detection algorithms tested on the custom test dataset, with and without the integration of email data. The performance metrics—precision, recall, and F1 score—were used to evaluate each algorithm's effectiveness. Initial testing on the custom dataset to determine the optimal n-gram size has shown that trigram cosine similarity and bigram Jaccard similarity provide the best results. Further testing for the n-gram threshold resulted in bigram being the best for both similarities. The different thresholds evaluations can be found in the appendix.

Only entries, which were falsely predicted, where taken into the evaluation metrics, thus not inflating the performance in case of the algorithms correctly identifying non-duplicates. One can always adjust this setting in the code by setting the parameter "include_non_dups" in "measure_metrics2" to true.

13.1 Performance without Email Data

The initial evaluations were conducted without incorporating email data to establish baseline performance metrics. These findings are detailed in Table 5. Among the tested algorithms, the trigram cosine similarity algorithm demonstrated superior performance across all metrics, achieving a precision, recall and an F1 score of 0.83. Close behind was the bigram Jaccard similarity. Conversely, the basic Soundex algorithm lagged significantly, with notably lower metrics of a precision of 0.29 and an F1 score of 0.31.

Table 5: Performance metrics without email data

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.55	0.52	0.53
Soundex	0.29	0.38	0.31
Custom Soundex	0.49	0.47	0.48
Kölner Phonetik	0.58	0.54	0.55
Levenshtein Distance 3	0.68	0.67	0.67
Trigram Cosine Similarity	0.83	0.83	0.83
Bigram Jaccard Similarity	0.82	0.75	0.77

13.2 Performance with Email Data and Bigram Jaccard Similarity Threshold

Incorporating email data with a bigram Jaccard similarity threshold significantly enhanced the performance metrics of most algorithms. Notably, the Simple Match algorithm experienced a substantial increase in its F1 score from 0.53 to 0.83. This improvement underscores the critical role of integrating email data in duplicate detection processes within CRM systems. The Custom Soundex and Kölner Phonetik algorithms also showed marked improvements across all metrics, indicating enhanced handling of phonetic and orthographic variations in names. However, Cosine Similarity did not exhibit any performance gains.

The advantage of incorporating email data is illustrated by the case of the standardized full name ”robert frei robert,” which is identified as erroneous in Section 5.6. The corresponding email prefix is ”robert.frei,” which standardizes to ”robert frei.” By calculating the bigram Jaccard similarity, I achieve a score of 0.73, surpassing the threshold of 0.65 required to employ the email prefix for further comparison with other strings. This demonstrates the utility of email data in enhancing the accuracy of duplicate detection.

Table 6: Performance with email data and bigram Jaccard similarity threshold

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.85	0.82	0.83
Soundex	0.31	0.42	0.34
Custom Soundex	0.64	0.63	0.64
Kölner Phonetik	0.69	0.66	0.67
Levenshtein Distance 3	0.77	0.76	0.76
Trigram Cosine Similarity	0.83	0.83	0.83
Bigram Jaccard Similarity	0.90	0.88	0.89

13.3 Performance with Email Data and Bigram Cosine Similarity Threshold

Subsequent testing with email data and a bigram cosine similarity threshold provided deeper insights into the robustness of these algorithms under various conditions. Notably, this testing revealed that employing a bigram configuration improved results specifically for the trigram cosine similarity algorithm. As illustrated in Table 7, the trigram cosine similarity algorithm demonstrated a significant improvement, with an increase of 0.2 in its F1 score. This substantial enhancement, achieved by combining bigram and trigram similarities for analyzing full names and email prefixes, indicates a more effective method. The performance of other algorithms remained unchanged, suggesting that the custom dataset may not include sufficient data or edge cases for a comprehensive evaluation for the different thresholds or the threshold value itself should be varied between cosine and Jaccard similarity.

An illustrative example of why the bigram cosine threshold outperformed the trigram involves the standardized name ”johann johann” and its corresponding standardized email prefix ”johann seeler.” When applying the bigram cosine similarity measure, the score reached 0.654, surpassing the threshold of 0.65. In contrast, the trigram cosine similarity measure scored only 0.592, falling short of the threshold. Consequently, ”johann seeler” was selected as the comparison string. This example demonstrates that integrating bigrams with trigrams can enhance matching accuracy by capturing additional nuances in name similarity. Table 16 in the appendix shows the different results.

Table 7: Performance with email data and bigram cosine similarity threshold

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.85	0.82	0.83
Soundex	0.31	0.42	0.34
Custom Soundex	0.64	0.63	0.64
Kölner Phonetik	0.69	0.66	0.67
Levenshtein Distance 3	0.77	0.76	0.76
Trigram Cosine Similarity	0.85	0.85	0.85
Bigram Jaccard Similarity	0.90	0.88	0.89

13.4 Performance swissICT Dataset

In my dataset of 21,798 entries, 3,292 were unclassifiable, either because they were labeled as [Zentrale] (Section 7.1 talks about this) or lacked both name and email information. This situation complicates the evaluation of my duplicate detection algorithm's performance. Specifically, it is challenging to determine whether the algorithm has successfully identified all duplicates. The vast size of the dataset makes manually searching for false negatives and verifying all true positives impractical.

To gauge the effectiveness of an algorithm, I can indeed count the true positives and false positives, but estimating the total number of true positives in the dataset (including those not detected by the algorithm) is more complex. Thus, to approximate the completeness of the true positives identified by the algorithm, I can employ random sampling.

For this evaluation, I will analyze 30 random samples. While this sample size is modest relative to the 18,500 potential comparisons in the dataset, the manual evaluation of each algorithm's predictions requires significant time, a limited resource. Notably, the Soundex algorithm, which is inherently aggressive in detecting duplicates, identified as many as 930 potential duplicates in some samples, averaging around 200. Given the impracticality of these numbers for the swissICT CRM system, I will exclude Soundex predictions from further consideration. I will assume that all true positives are captured by the combined results of all other algorithms, thus I can also estimate the count of false negatives.

Due to the computational demands of the Levenshtein Distance, I will not deploy this algorithm on the dataset. Additionally, given the superior speed and performance of Jaccard Similarity as demonstrated in Tables 6 and 7, I will also forego running cosine similarity.

Out of the initial 30 random samples, only 4 contained duplicates. Although this might seem representative of the duplicate frequency within swissICT, such a small number of duplicates is insufficient for a robust evaluation of algorithm performance. To address this, I will add 10 more manually selected samples identified by Simple Match as having at least 3 duplicates each and the algorithms having differing predictions. This adjustment increases the total sample size to 40, with 14 samples containing duplicates, better aligning with the duplicate ratio observed in the custom dataset.

Table 8: Performance 40 random samples, 14 of which are duplicates

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	1.00	1.00	1.00
Soundex	0.00	1.00	0.00
Custom Soundex	0.46	1.00	0.63
Kölner Phonetik	0.36	1.00	0.53
Bigram Jaccard Similarity	0.28	1.00	0.44

The results from the evaluation of 40 samples are noteworthy. Simple Match achieved flawless precision by successfully identifying all true positives without any false positives. However, the scenario with Soundex is quite different. Despite achieving perfect recall, which indicates that it missed no duplicates, its zero precision is a significant drawback, demonstrating a high rate of false positive detections that compromise its practical utility. For all other algorithms besides Simple Match, a high recall is com-

mon as they identify numerous candidates. However, most of these are false positives, which minimizes the occurrence of true negatives.

These outcomes suggest that the custom dataset may not accurately reflect the dynamics of a typical CRM system, which often contains identifiable erroneous data primarily associated with full names and email addresses. In contrast, the custom dataset includes various types of erroneous data that are less typical in real-life scenarios. Additionally, the choice of sample size and the selection of different samples could better represent algorithm performance. The assertion that Simple Match maintains perfect precision and recall across diverse data types extracted from the swissICT dataset is unrealistic. It is improbable for any algorithm, including Simple Match, to consistently achieve perfect precision and recall given the inherent complexities and variability in real-world data. Nonetheless, the significant disparity in performance is undeniable, establishing Simple Match as the most effective algorithm among those tested for detecting duplicates.

13.4.1 Using Email Data on swissICT dataset

In total, 1880 email prefixes were utilized for string comparisons. As depicted in Table 9, employing email prefixes for further comparison proved beneficial in various cases, though not universally. Specifically, this approach effectively eliminated issues with double names, middle names, and other name-related abbreviations. However, its usefulness was limited when the email prefixes themselves contained abbreviations, such as in the example ”h l ohndermann.” While ideally both the full name and email prefix would be used for comparison to enhance accuracy, relying solely on the email prefix already provides a solid basis for identifying matches, especially the Simple Match algorithm.

Table 9: The first column contains standardized full names, and the second column includes their standardized email prefixes, which were subsequently utilized for string comparison.

Standardized Full Name	**Standardized Email Prefix**
dr marc k peter	marc peter
cecile cecile fraumann	cecile fraumann
harald ohndermann	h l ohndermann
johannes knaegi	johannes
marian angela marian angela schnurrenberger	marian schnurrenberger
christine huenni harni	christine huenni
andi haeaesli	andreas haeaesli

13.5 Performance CORA and RESTAURANT Datasets

I aim to provide a comprehensive evaluation of the algorithms by testing them on the widely recognized CORA and RESTAURANT benchmark datasets. While these results should not be interpreted as definitive, they offer valuable insights into the algorithms' performance in specific contexts. The RESTAURANT dataset, featuring primarily English restaurant names, includes only non-duplicates or pairwise duplicates, which differs from my approach that searches for all potential duplicate candidates. Similarly, the CORA dataset contains mostly abbreviated full names, presenting a unique challenge for my implementation. The algorithms have not been specifically tailored for these tasks; the preprocessing steps and thresholds remain consistent with those used for the custom and swissICT datasets. Most notably, the algorithms demonstrated notably lower recall on the CORA dataset. This suboptimal performance is primarily attributable to inconsistencies in the dataset, such as some authors' names being

abbreviated while others are not. Additionally, duplicate entries for books often list varying authors, further complicating accurate duplicate detection.

Bigram Jaccard similarity consistently achieved the best performance across both datasets. Remarkably, Kölner Phonetik maintained competitive performance alongside other algorithms, despite its design focusing on the German language. This suggests that some of its rules may be applicable beyond language-specific contexts.

Most notably, the algorithms demonstrated lower recall on the CORA dataset. This suboptimal performance is primarily attributable to inconsistencies in the dataset, such as some authors' names being abbreviated while others are not. Additionally, the presence of varying authors in duplicate book entries further complicates the accurate detection of duplicates.

The difference in performance between CORA (Table 11), RESTAURANT (Table 10) and swissICT dataset (Table 8) indicates, that these datasets cannot be used as a benchmark for duplicate detection in CRM systems.

Table 10: Performance RESTAURANT dataset name column only

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.69	0.69	0.69
Soundex	0.32	0.40	0.34
Custom Soundex	0.65	0.66	0.59
Kölner Phonetik	0.59	0.60	0.60
Levenshtein Distance 3	0.54	0.56	0.55
Trigram Cosine Similarity	0.66	0.67	0.66
Bigram Jaccard Similarity	0.71	0.71	0.71

Table 11: Performance CORA dataset authors column only

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.62	0.21	0.24
Soundex	0.50	0.44	0.36
Custom Soundex	0.62	0.21	0.25
Kölner Phonetik	0.63	0.38	0.39
Levenshtein Distance 3	0.63	0.33	0.34
Trigram Cosine Similarity	0.46	0.43	0.38
Bigram Jaccard Similarity	0.60	0.51	0.49

14 Limitations

This study, while extensive in its examination of duplicate detection algorithms, faces several limitations that must be acknowledged. First, the representativeness of the datasets used, including both the custom dataset derived from the swissICT dataset and widely recognized benchmarks like the CORA and RESTAURANT datasets, may not fully capture the diversity and complexities of real-world data encountered in various CRM systems. This limitation potentially affects the generalizability of the findings, as the specific characteristics of these datasets, such as the predominance of certain types of errors or the

distribution of duplicates, may not be reflective of other environments.

Furthermore, the performance of the algorithms was evaluated primarily through precision, recall, and F1 score metrics calculated on a limited number of manually selected samples. This approach, while necessary due to resource constraints, introduces a degree of subjectivity and potential bias in the sample selection process. The sample size, although augmented by additional manually selected entries known to contain duplicates, remains relatively small given the total number of possible comparisons in the dataset. This constraint may not provide a sufficiently robust basis for statistical generalization to larger and more diverse datasets.

The exclusion of the Levenshtein Distance algorithm from the final evaluation due to its computational demands also limits the study. This exclusion omits a potentially valuable comparative perspective on the efficiency and effectiveness of edit-distance based methods relative to those evaluated. Additionally, the decision to disregard results from the Soundex algorithm due to its high false-positive rate, while justified, eliminates an opportunity to explore the trade-offs between different types of phonetic algorithms in more depth.

Lastly, the computational and time resources available for this study were limited, which constrained the extent to which the algorithms could be fine-tuned and optimized. Future studies could benefit from exploring more advanced machine learning models and incorporating additional features such as contextual and semantic analysis to enhance duplicate detection accuracy.

15 Conclusion

In this thesis, I evaluated various duplicate detection algorithms to identify the most effective approach for handling duplicates in CRM systems, particularly in contexts with a high prevalence of German names. My comprehensive analysis covered phonetic, edit and token-based algorithms, examining their performance with and without the inclusion of email data.

Initially, the algorithms were tested without email data to establish baseline performance metrics. Among these, the trigram cosine similarity algorithm demonstrated the highest effectiveness, with notable precision, recall, and F1 scores, followed closely by the Jaccard similarity algorithm. The Soundex algorithm, however, exhibited the weakest performance, highlighting its limitations in accurately detecting duplicates.

Incorporating email data with an n-gram Jaccard similarity threshold significantly improved the detection capabilities of most algorithms. This enhancement was most pronounced in the Simple Match algorithm, whose F1 score increased dramatically, underscoring the critical role of email data in the duplicate detection process. The Custom Soundex and Kölner Phonetik algorithms also benefited from this inclusion, demonstrating improved precision and recall.

Further testing with an n-gram cosine similarity threshold provided deeper insights, particularly highlighting the superior performance of the Trigram Jaccard Similarity algorithm. This approach achieved the highest F1 score, suggesting that cosine similarity is more effective than Jaccard similarity in aligning similarities between full names and email prefixes. This appraoch indicates, that there are hidden interactions between different algorithms, which could lead to an even better result, than simply using one on its own.

There remains significant potential for enhancement in phonetic matching algorithms for the German language. While Kölner Phonetik effectively addresses specific character combinations, integrating additional rules that consider syllable structure, word length, and edit distance metrics could yield a more robust algorithm.

Ideally, a composite algorithm that merges methodologies from various similarity measures should be developed. By creating a scoring system that leverages the strengths and compensates for the weaknesses of each approach, such a unified algorithm could provide a more balanced and effective solution for phonetic matching in German.

For swissICT, I recommend primarily employing Simple Match, complemented by Kölner Phonetik or Custom Soundex, along with n-gram Jaccard similarity to effectively identify potential duplicates. Additionally, I suggest adopting more rule-based strategies that incorporate universally relevant data fields, such as phone numbers and company names, similar to the successful integration of the email rule. This approach should enhance the accuracy and reliability of duplicate detection within the system.

Overall, the integration of email data and the application of appropriate similarity thresholds are essential for enhancing duplicate detection accuracy. The findings from this thesis provide valuable insights and practical recommendations for improving data quality in CRM systems, ensuring more efficient and accurate customer data management. These results are particularly relevant for systems dealing with complex name structures and variations, as demonstrated in the swissICT dataset.

References

Ben Carterette. 2009. *Precision and Recall*. Springer. https://doi.org/10.1007/978-0-387-39940-9_5050.

Peter Christen. 2012. *Data Matching*. Springer. https://doi.org/10.1007/978-3-642-31164-2_10.

DeepVA. 2024. Duplicate detection use case. Accessed on: 01-05-2024 from `https://deepva.ai/de/usecase/duplicate-detection`.

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16. https://doi.org/10.1109/TKDE.2007.250581.

T. N. Gadd. 1990. Phonix: The algorithm. *Program*, 24(4):363–366.

PIKON International Consulting Group. 2021. Stammdatenbereinigung: Innovationen in der dublettenerkennung. Accessed on: 2024-04-14 from `https://www.pikon.com/de/blog/stammdatenbereinigung-innovationen-in-der-dublettenerkennung/`.

Group XS. 2018. Anpassung des soundex-algorithmus für die deutsche sprache. Accessed on: 2024-03-23 from `https://web.archive.org/web/20180709154422/https://www.groupxs.com/archives/anpassung-des-soundex-algorithmus-fur-die-deutsche-sprache/132.html`.

Niklas Heer. 2022. speed-comparison: A repository comparing the speed of different programming languages. Accessed on: 23-04-2024 from `https://github.com/niklas-heer/speed-comparison`.

Donald E. Knuth. 1973. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley.

Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of Massive Datasets*, 3 edition. Cambridge University Press. https://doi.org/10.1017/9781108684163.

Vladimir L. Levenshtein. 1965. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1(1):8–17.

Jörg Michael. 1988. Nicht wörtlich genommen - Schreibweisentolerante Suchroutinen in dBase implementiert. *ct Magazin für Computer & Technik*, 10:126–131.

Gary Mokotoff. 2003. Soundexing and genealogy. Avotaynu - Publisher of Works on Jewish Genealogy.

Felix Nauman and Melanie Herschel. 2022. *An introduction to duplicate detection*. Springer Cham. https://doi.org/10.1007/978-3-031-01835-0.

Lawrence Philips. 1990. Hanging on the metaphone. *Computer Language*, 7(12).

Hans-Joachim Postel. 1969. Die Kölner Phonetik - Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. *IBM-Nachrichten*, 19:925–931.

D. Pyle. 1999. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers.

Random Name Generator. Random Name Generator for Switzerland. Accessed on: 15-03-2024 from `https://www.random-name-generator.com/de/schweiz`.

swissICT. 2024. Über uns. Accessed on: 24-04-2024 from `https://www.swissict.ch/`.

Tableau. 2021. Guide to data cleaning: Definition, benefits, components, and how to clean your data. Accessed on: 07.05.2024 from `https://www.tableau.com/learn/articles/what-is-data-cleaning`.

Tomaz Solc. 2009. Unidecode: ASCII transliterations of Unicode text. Accessed on: 10-04-2024 from `https://pypi.org/project/unidecode/`.

Georg Wilde and Carsten Meyer. 1988. Doppelgänger gesucht - Ein Programm für kontext-sensitive phonetische Textumwandlung. *ct Magazin für Computer & Technik*, 25.

Martin Wilz. 2005. Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen. Magisterarbeit Universität Köln.

Elizabeth Zhang and Yixin Zhang. 2009. F-measure. In *Encyclopedia of Database Systems*, pages 1147–1147. Springer. https://doi.org/10.1007/978-0-387-39940-9_2140.

Eric Zhu. 2023. datasketch: MinHash, LSH, LSH Forest, Weighted MinHash, HyperLogLog, HyperLogLog++, LSH Ensemble and HNSW. Accessed on: 16-05-2024 from `https://github.com/ekzhu/datasketch`.

A Appendix

Table 12: Performance metrics with Bigram Jaccard similarity threshold

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.846	0.815	0.826
Soundex	0.313	0.419	0.343
Custom Soundex	0.644	0.632	0.636
Kölner Phonetik	0.687	0.663	0.671
Levenshtein Distance 3	0.773	0.760	0.764
Trigram Cosine Similarity	0.829	0.829	0.829
Bigram Jaccard Similarity	0.896	0.881	0.886

Table 13: Performance metrics with Trigram Jaccard similarity threshold

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.692	0.662	0.672
Soundex	0.293	0.377	0.314
Custom Soundex	0.529	0.517	0.521
Kölner Phonetik	0.590	0.566	0.574
Levenshtein Distance 3	0.707	0.693	0.698
Trigram Cosine Similarity	0.829	0.829	0.829
Bigram Jaccard Similarity	0.836	0.791	0.806

Table 14: Performance metrics with Bigram Cosine similarity threshold

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.846	0.815	0.826
Soundex	0.313	0.419	0.343
Custom Soundex	0.644	0.632	0.636
Kölner Phonetik	0.687	0.663	0.671
Levenshtein Distance 3	0.773	0.760	0.764
Trigram Cosine Similarity	0.851	0.851	0.851
Bigram Jaccard Similarity	0.896	0.881	0.886

Table 15: Performance metrics with Trigram Cosine similarity threshold

Algorithm	**Precision**	**Recall**	**F1 Score**
Simple Match	0.846	0.815	0.826
Soundex	0.313	0.419	0.343
Custom Soundex	0.644	0.632	0.636
Kölner Phonetik	0.687	0.663	0.671
Levenshtein Distance 3	0.773	0.760	0.764
Trigram Cosine Similarity	0.829	0.829	0.829
Bigram Jaccard Similarity	0.896	0.881	0.886

Table 16: Performance with email data and n-gram cosine similarity threshold on different n-gram cosine similarities

Algorithm	**Precision**	**Recall**	**F1 Score**
Bigram Cosine Similarity Threshold			
Bigram Cosine Similarity	0.69	0.71	0.70
Trigram Cosine Similarity	0.85	0.85	0.85
Trigram Cosine Similarity Threshold			
Bigram Cosine Similarity	0.68	0.69	0.68
Trigram Cosine Similarity	0.83	0.83	0.83