Master's Programme in Mathematics and Operations Research

Experimental evaluation of record linkage algorithms in a secure banking environment

Christian Segercrantz

Aalto University

MASTER'S THESIS

Aalto University
MASTER'S THESIS 2023

Experimental evaluation of record linkage algorithms in a secure banking environment

Christian Segercrantz

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.
Otaniemi, July 31, 2023

Supervisor: Prof. Alexander Jung
Advisor: PhD Giuseppe Scavo

Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

Author
Christian Segercrantz

Title
Experimental evaluation of record linkage algorithms in a secure banking environment

School School of Science

Master's programme Mathematics and Operations Research

Major Systems and Operations Research **Code** SCI3055

Supervisor Professor Alexander Jung

Advisor PhD Giuseppe Scavo

Level Master's thesis **Date** July 31, 2023 **Pages** 82+24 **Language** English

Abstract

This thesis studies record linkage algorithms in secure banking environments. Financial crime prevention laws require financial institutions to identify and monitor high-risk customers and transactions using in-house and third-party data sources. Examples of such data are lists of terrorist financers, sanctioned persons, and convicted financial criminals. The data from the various sources may not share unique identifiers, such as social security numbers or business IDs, which makes it non-trivial to determine whether two records belong to the same person. Thus, we must find ways to combine data sets by matching the records in a secure and privacy-preserving banking environment. The environment requires vetted, robust, and resource-optimized solutions and implementations.

This thesis studies theory and previous works on deterministic and probabilistic record linkage methods, their theoretical frameworks, and their application areas. Furthermore, we implement solutions to two record linkage problems based on deterministic and probabilistic methods. The problems are based on linking juridical person records respectively natural person records.

To perform deterministic record linkage, we create a pipeline using blocking, string embedding, and string similarity algorithms. We evaluate two different string embedding models: TF-IDF and Word2Vec. For probabilistic record linkage, we use the Python package Splink. The package is based on the Fellegi-Sunter model, which shows how to calculate the posterior probability of a pair being a match or a non-match. The package uses the expectation-maximization algorithm to find the values for the posterior probability and the m- and u-variables.

The results show that deterministic and probabilistic record linkage are feasible within the scope of this thesis and have different application areas. We find that different parameters have a significant impact on embedding-based models. The outcome of the TF-IDF-based model mainly relies on the values of the feature vector length and the string similarity threshold. Conversely, the Word2Vec-based model is affected nearly exclusively by random bucket projection distance threshold. The probabilistic model requires comparably vast resource investments to function as desired, and the causalities between input and output are not easily detected. However, when correctly set up, the method can model complex underlying data distributions.

This thesis provides a framework for the industry to use record linkage. The framework consists of two methods, deterministic and probabilistic record linkage, and recommendations on how to use them.

Keywords record linkage, experimental, entity matching, deterministic, probabilistic, Splink, secure bank environment, financial crime prevention, Fellegi-Sunter, string embedding

Författare
Christian Segercrantz

Titel
Experimentell utvärdering av sammanlänkningsalgoritmer i en säker bankomgivning

Högskola Högskolan för teknikvetenskaper

Magisterprogram Mathematics and Operations Research

Huvudämne Systems and Operations Research **Kod** SCI3055

Övervakare Professor Alexander Jung

Handledare Filosofie doktor Giuseppe Scavo

Arbetets slag Diplomarbete **Datum** 31 juli 2023 **Sidantal** 82+24 **Språk** Engelska

Sammandrag

Det här diplomarbetet studerar algoritmer för sammanlänkning av uppgifter i en säker bankomgivning. Lagar mot finansbrott kräver att finansiella institutioner identifierar och övervakar högriskkunder och -transaktioner med hjälp av interna och tredje partens datakällor. Exempel på sådan data är listor på finansierare av terrorism, sanktionerade personer och dömda finansbrottslingar. Datan från de olika källorna delar nödvändigtvis inte unika identifierare, såsom personnummer och FO-nummer, vilket gör det icke-trivialt att avgöra ifall två dataposter tillhör samma person. Därmed är vi tvungna att hitta sätt att kombinera dataset genom att matcha uppgifter i en säker och integritetsbevarande bankomgivning. Omgivningen kräver granskade, robusta och resursoptimerade lösningar och implementeringar.

Det här diplomarbetet studerar teori och tidigare arbeten om deterministiska och probabilistiska metoder för sammanlänkande av uppgifter, deras teoretiska koncept och tillämpningsområden. Utöver detta implementerar vi lösningar på två sammanlänkningsproblem baserade på deterministiska och probabilistiska metoder. Dessa problem baserar sig på sammanlänkning av juridiska respektive naturliga personers uppgifter.

För att utföra deterministisk sammanlänkning skapar vi en pipeline med hjälp av blockindelning, stränginbäddning och algoritmer för stränglikhet. Vi evaluerar två modeller baserade på stränginbäddning: TF-IDF och Word2Vec. För det probabilistiska sammanlänkandet använder vi Splink-paketet för Python. Paketet baserar sig på Fellegi-Sunter modellen, som anger posteriori-sannolikheten att ett par är en match eller icke-match. Paketet använder väntevärdesmaximerings-algoritmen för att hitta värden för posteriori-sannolikheten samt m- och u-variablerna.

Resultaten visar att deterministisk och probabilistisk sammanlänkning är genomförbart inom diplomarbetets omfattning och att de har olika tillämpningsområden. Vi visar att olika parametrar har signifikant inverkan på inbäddningsbaserade modeller. Resultatet av den TF-IDF-baserade modellen beror främst på värdena av attributvektorns längd och stränglikhetens tröskelvärde. Word2Vec-baserade modellen påverkas däremot nästintill endast av avståndströskelvärdet för slumpmässig kategoriprojicering. Den probabilistiska modellen kräver jämförelsevis stora resursinvesteringar för att fungera som önskat, och orsakssambanden mellan input och output är inte lätta att upptäcka. Dock kan metoden modellera komplexa underliggande datadistributioner när den är korrekt konfigurerad.

Detta diplomarbete ger ett ramverk för industrin för att använda sammanlänkning av uppgifter. Ramverket består av två metoder, deterministisk och probabilistisk uppgiftslänkning, och rekommendationer om hur man använder dem.

Nyckelord sammanlänkning, experimentell, entitetsmatchning, deterministisk, probabilistisk, Splink, säker bankomgivning, förebyggande av ekonomisk brottslighet, Fellegi-Sunter, stränginbäddning

Preface and acknowledgements

This is literally the final work I am completing for my degree. I started my career at Aalto University nine years ago and today it comes to an end. I am thankful for the education and opportunities I have received. As the journey ends, I know I will feel many emotions, among them surely happiness, sadness, and pride. However, currently I am mostly relieved.

Firstly, I want to thank Nordea Bank for providing and supporting me with this thesis. I especially want to thank my advisor PhD Giuseppe Scavo for his guidance, patience, and help. I want to thank Prof. Alex Jung for supervising my thesis. Thank you to my parents for all their support during not only my thesis writing or time at the university, but during my whole education. I want to thank my sisters for the inspiration and guidance they have provided. Thank you Astrid and Anna for supporting me both grammatically and content-wise with the thesis.

Further, the orange support troupe and the elegant enjoyers of bubbles also deserve a thank you for all the fun during late hours. I additionally want to thank every single person I have had the honour to work together with during my volunteer career, all the working-groups, committees, and boards. A special thanks among these groups goes to Östmetron and "The board who said NO". Finally, a huge thank you to Teknologföreningen and Aalto University Student Union for making all of this possible.

The time at the university has given me lessons for life. The lectures, exercise sessions, and projects have prepared me for the technical work I will be engaging in, while my student activities have prepared me for life situations and general human interactions. These will, surely, provide me with tools for life.

Iuvenis in aeternum,
Christian Segercrantz
Otnäs, 31.7.2023

Contents

1. Introduction

Financial institutions, such as banks, must abide by laws dictating actions against financial crime and money-laundering [1, 2]. This means monitoring customers that are convicted criminals or at high risk of performing criminal actions. From the financial institutions ' perspective, records of high-risk individuals can be found in both internal and external sources. By joining data sets, we can obtain enriched data on which we can perform tasks, such as research, advanced analytics, or machine learning [3]. However, a significant challenge arises when the records do not share a unique identifier with the institution's internal data sets. Linking records without unique identifiers is also a prevalent problem in domains like history [4], medicine [3, 5] and social sciences [6, 7]. This thesis aims to find a feasible way to join data sets that do not share a common unique identifier, given the constraints of the restricted, secure, and privacy-preserving banking environment.

As the author writes this thesis for the author's employer, Nordea Bank, we must consider unique constraints when conducting and writing this thesis. These constraints exist due to the privacy and security measures required by the bank. For example, the author cannot use or publish customer information in the study. Similarly, the author must do all programming in an in-house controlled environment. Additionally, these environments are resource restricted and require well-optimized solutions.

We must find methods available in our environment that satisfy the above environmental requirements and performance expectations. We can use Cartesian join and run pair-by-pair comparison algorithms on the complete data set. However, since Cartesian join scales as $\mathcal{O}(\prod_i N_i)$, where N_i is the length of the i:th data set, the amount of comparisons required grows polynomially. Additionally, simple pair-wise comparison methods do not provide good results. We need to find more efficient and accurate

approaches. Hence, we look for an alternative approach using advanced record linkage methods.

Record linkage, also known as data matching, data linkage, entity resolution, or entity matching, is the task of identifying the records that refer to the same entity or individual, i.e., person, across multiple data sets. A record is a collection of information about a person, either juridical or natural [8]. Record linkage is required when joining data sets that do not share a unique identifier, such as a social security or business identification number. Thus, one must match the records using the person's characteristics. These characteristics can be, for example, name, date of birth, address, gender, and industry.

Furthermore, record linkage can be divided into deterministic and probabilistic categories. This thesis investigates both categories and applies them to relevant problems. For problems where the deterministic approach is suitable, embedding-based solutions, commonly used in natural language processing, are applied. The probabilistic record linkage framework, initially proposed by Fellegi and Sunter [9], is employed for settings suitable for the probabilistic methods. Both deterministic and probabilistic record linkages have been applied to a wide variety of problems [3, 4, 5, 6, 7, 10]. Deciding upon which method to use depends on the result and a range of resource-related questions, such as time, data quality, and desired result [11].

This thesis investigates the available methods and tools, considering their relevant areas of use and the previously mentioned restrictive environment. We evaluate and report the impact of factors, such as parameters and design choices, on run-time and performance metrics. Finally, the thesis recommends what model to use depending on the nature of the linkage problem.

Chapter 2 introduces the data used in the thesis. We give an overview of the theory and mathematics used in the work in Chapter 3. How the theoretical frameworks and methods are used and implemented is discussed in Chapter 4. Chapter 5 introduces the thesis results and discusses their implications. Finally, Chapter 6 presents the conclusions drawn from the study and recommends further research in the field.

2. Data

This chapter describes the data used in the thesis. We present the origins, characteristics, and transformations for each data set. The chapter is split into three sections, one for each data source and one discussing the commonalities of the data.

2.1 Dan & Bradstreet data

One of the two main data sets we use in this thesis is the Dan & Bradstreet (DnB) [12] provided data. Dan & Bradstreet is a commercial company that gathers and harmonises data. Nordea uses the DnB data as a third-party solution for standardising global data on registered companies. The data consists of subsets, of which this thesis uses two: company-based data and associated party-based data.

Company based data
The company-based data consists of registered companies around the world. It contains information about the company and its governance, e.g. business name, country, mother company, and subsidiaries. It consists of 532 387 535 records and 115 attributes which take up approximately 227MB when stored in a Pyspark data frame. We filter the data only to contain companies based in Finland. The filtered data consists of 2 146 245 records.

Associated party data
The associated party-based data consists of the parties associated with the entities of the previous data. Parties can be, e.g. managing directors, significant shareholders, or otherwise persons of interest. The data describe the link between the associated party and the entity and attributes of the associated party, e.g. name, date of birth, address, country of residence, and ownership percentage. The data consist of 249 481 914 records and 86

attributes, which take up approximately 187MB when stored in a Pyspark data frame. We filter the data only to include natural persons with an ownership percentage greater than or equal to 50% and Finland in the country attribute. The filtered data consist of 340 630 records.

2.2 Dow Jones watch list data

The Dow Jones (DJ) dataset [13] is a data set of records of people that are of interest to monitor for financial institutions. The records are snapshots of specific times not updated after the initial addition to the database. This characteristic means that information may change over time, such as an address, country of residence or even name, in the case of juridical persons, and may be outdated compared to other data sets where records are being updated as information changes, such as the previously discussed DnB data.

We receive the data split into two separate data sets, one juridical person-based and one natural person-based data set. We present both data sets here.

Juridical person-based data
The data describes juridical persons, i.e. entities, and why they have been added to the watch list. The data set contains information concerning the reason for being added to the watch list and information about the entity, e.g. name, address, country, and other incident-related information.

The complete data set consists of 254 097 records and 28 attributes which take up approximately 180MB when stored in a Pyspark data frame. However, many records contain aggregations of different versions of the same information, e.g. if a company has changed name or address, multiple versions might be found in the data. The data can be parsed or vertically expanded into separate records. When parsed, the data set consists of 2 037 085 records.

Furthermore, we filter the data only to contain records of juridical persons associated with Finland. The data may include multiple countries per record: Records that match any of the three will be included in cases where we filter the data. The filtered data set consists of 1 346 records.

Natural person-based data

The data describe natural persons and why they have been added to the watch list. The data set contains information about the natural persons, e.g. name and variations, gender, country information, and reason for being added to the list.

The complete data set consists of $3\,131\,919$ records and 39 attributes which take up approximately 189MB when stored in a Pyspark data frame. Similarly to the entity-based data set, the person-based data contains records with aggregations. We get a total of $10\,631\,028$ records by parsing the aggregations to separate records.

Additionally, we filter the data only to include persons connected to Finland. Similarly to juridical persons, each record may contain connections to multiple countries, such as citizenship or residency. Records containing any reference to Finland are kept. The filtered data set consists of $43\,843$ records.

2.3 Common attributes

In order to link the records of the two data sets, the sets need to share attributes used for the linkage. This section presents and discusses the attributes shared between the data sets.

2.3.1 Juridical person-based data

In the juridical person-based data, we can find an overlap of common attributes. The shared attributes contain name, address, city and country. The entities of these data sets will be referred to as juridical persons. The Cartesian join of the data sets produces $2\,888\,845\,770 \approx 2.9 \cdot 10^9$ pairs to compare.

Table 2.1 presents a mock data set miming the attributes of the DnB data. The quality of data is excellent and standardised. Table 2.2 illustrates the data of the DJ data set and the quality of the records. We can see that the quality is very inconstant. Some records, not illustrated in the table, even contain data unrelated to its column, e.g. a reference to connections to other entities in the address attribute.

Table 2.1. A demonstration of the DnB data of juridical person records. The records are created randomly by the author and have no connection to real entities.

Id	Name	Address	City	Country
1	Company LTD	Main street 1	Tampere	Finland
2	AB Yhtiö OY	Pääkatu 24 A 14	Helsinki	Finland
3	Asunto-osakeyhtiö A/S	Rantatie 5	Espoo	Finland
4	Teemu Teekkari T:mi	Pohjolantie 456	Oulu	Finland

Table 2.2. A demonstration of the DJ data of juridical person records. The records are created randomly by the author and have no connection to real entities.

Id	Name	Address	City	Country
1	Company LTD	PL 123		Finland
2	AB Yhtiö OY		Helsinki; 00100	Finland
3	Asunto-osakeyhtiö A/S			
4	Teemu Teekkari T:mi	Pohjolantie 456 A 15	Oulu	Finland

2.3.2 Natural person-based data

The Cartesian join of the two data sets produces $14\,934\,241\,090 \approx 14.9 \cdot 10^9$ pairs to compare. The natural person data in the two data sets have a relatively small overlap of attributes compared to how many attributes natural persons have. Additionally, in some cases, the data quality or ratio of missing data makes some attributes unusable. The attributes that have been concluded usable are name, country of origin or affiliation, and date of birth.

Table 2.3 shows a fictive example of the DnB data set. Table 2.4 highlights a fictive set of records from the DJ data set similar to the one we use. The table highlights the data quality level, where compared to Table 2.3, data may be missing or have differing levels of accuracy.

Data cleaning

To improve the data quality, we apply cleaning and augmentation steps to the data. First, the name is split into first, middle and last names. We perform the split such that the first string before a white space is considered the first name, and the last string after the last white space is considered the last name. The remaining part is considered the middle

Table 2.3. A demonstration of the DnB data of natural person records. The records are created randomly by the author and have no connection to real persons.

Id	Name	Date of birth	Country
1	Teemu Kalevi Teekkari	1976-03	Finland
2	Aino Helmi Kaarina Niminen	2000-04	Finland
3	Antero Mikael-Jussi Koivunen	1994-01	Finland
4	Lucas Johansson	1986-11	Sweden

Table 2.4. A demonstration of the DJ data of natural person records. The records are created randomly by the author and have no connection to real persons.

Id	Name	Date of birth	Country
1	Teemu Kalevi Teekkari		Fin
2	Aino Niminen	2000-04-03	Fin
3	Antero Mikael-Jussi Koivunen	1994	Fin
4	Lucas Johansson	1986-11-2	Swe

name. If only two strings are present in the name, the middle name is considered missing and set to null.

The quality of the date of birth attribute of the two data sets differs significantly. In the DnB data, the date of birth consists of only the year and month. Conversely, the DJ data set may include anything from only a year to the full date of birth. To compare the attribute, these are changed to all have a complete date format, and the missing date is set to one. For example, if we have a value of "1994", it is set to "1994-01-01", and "2000-05" becomes "2000-05-01". We discuss the impacts of this modification on attribute comparison in Section 4.3.1.

Additionally, we use phonetic algorithms to construct attributes that we can use for record comparison. These are particularly useful for comparing names that sound the same but are spelt differently, for example, "Steven" and "Stephen". We create phonetic attributes for first and last names.

The two data sets have different ways of spelling country names. For example, both "Russia" and "Russia Federation" exist in the data sets. Even inside the same data set, notably the DJ data set, the spelling of countries may be different due to how the data set has been created. Hence, we check and map the values to standard spelling. This is also beneficial in later parts when we can be sure there is only one way to spell the name of a country.

This gives us eight attributes for record linkage: full name, first name, phonetic encoding of the first name, middle names, last name, phonetic encoding of the last name, date of birth and country.

3. Methods

This section presents the methods and theoretical frameworks we use in the thesis. First, Section 3.1 explains the concept of record linkage. The section further provides a deep dive into probabilistic and deterministic record linkage. Thereafter, Sections 3.2 to 3.5 present algorithms and methods related to those mentioned above or otherwise used. Finally, in Section 3.6, we present the methodology related to estimating the performance of the employed methods.

3.1 Record linkage

A record is a sub-sample of information about a person, either juridical or natural. A record can take many forms, such as a name, sentence, book, website, or collection of attributes [8]. When stored in databases, records are usually referred to as rows [14, 15]. Hence, in record linkage, we aim to connect different pieces of information, i.e. records, to the same person.

Record linkage aims to classify pairs of records in the product space made of the Cartesian join $\mathbf{C} = \mathbf{A} \times \mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are data sets of records. We classify the pairs into either $\mathbf{M}$, the set of true matches, $\mathbf{U}$, the set of true non-matches and the set of possible matches [16]. We discuss two main classification methods: deterministic and probabilistic record linkage. These are presented in the following sections.

Figure 4.1 illustrates a typical pipeline for a record linkage involving two data sets. The two data sets are first pre-processed to standardise data and have the same attributes available for the rest of the pipeline. Then, we perform a Cartesian join on the data sets to get a set consisting of all possible pairs. Following that, we create data blocks using a blocking algorithm to reduce the required comparisons of the following steps. Finally, we perform the linkage and obtain the linked data set. This linked data

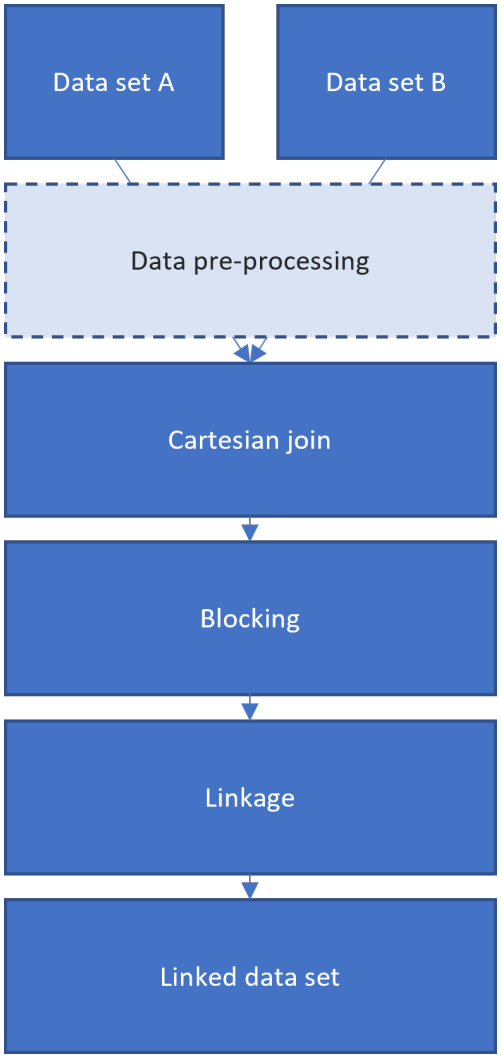


Figure 3.1. A diagram of a typical record linkage pipeline.

set includes the match probability: 1 or 0 for a deterministic record linkage or a number in the range $[0, 1]$ for the probabilistic record linkage.

3.1.1 Probabilistic record linkage

Probabilistic record linkage, sometimes called fuzzy matching, is one of two main categories of record linkage methods. The probabilistic record linkage theory is based on the work by Fellegi and Sunter [9] and is presented here. The method calculates a posterior probability of a comparison pair being a match. To get the probability, we sum up the evidence of the agreement level of each attribute of the comparison pairs. The evidence, or weights, is assigned to each agreement level using variables indicating their probability of being a match or a non-match. The pairs can then be assigned to the matching set **M**, non-matching set **U**, or the set of possible matches. If the probability exceeds a set limit R_{upper}, we assign it to **M**. If the probability is lower than a set limit R_{lower}, we assign it to **U** and else

we assign it to the set of possible matching pairs. The possible matches require human review to sort into **M** or **U**. The sorting of possible matches is beyond the scope of this thesis.

To estimate the weight of each attribute comparison, the method uses two variables: m and u, where m is the probability that two fields match, or are sufficiently close, when the pair belongs to the set **M** and, similarly, u is the probability that the attributes match, or are sufficiently close when the pair belongs to the non-matching set **U**. These variables exist for each attribute we compare.

Take sets **A** and **B** whose elements we denote **a** and **b**. We assume the two sets have at least one common element. We denote the set of pairs created by taking the Cartesian join between the two sets as

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = \mathbf{M} \cup \mathbf{U} \tag{3.1}$$

where **M** denotes the matches between the two sets,

$$\mathbf{M} = \{(\mathbf{a}, \mathbf{b}) : \mathbf{a} = \mathbf{b}, \mathbf{a} \in \mathbf{A}, \mathbf{b} \in \mathbf{B}\}, \tag{3.2}$$

and **U** denotes the non-matches between the two sets,

$$\mathbf{U} = \{(\mathbf{a}, \mathbf{b}) : \mathbf{a} \neq \mathbf{b}, \mathbf{a} \in \mathbf{A}, \mathbf{b} \in \mathbf{B}\}. \tag{3.3}$$

We may additionally use blocking, discussed further in Section 3.2, to filter out some pairs from **C**.

We denote the records of the members of each set $\alpha(\mathbf{a})$ respectively $\beta(\mathbf{b})$. We define the comparison vector γ associated with each pair of records as:

$$\gamma(\alpha(\mathbf{a}), \beta(\mathbf{b})) = \begin{bmatrix} \gamma_1(\alpha(\mathbf{a}), \beta(\mathbf{b})) & \gamma_2(\alpha(\mathbf{a}), \beta(\mathbf{b})) & \dots & \gamma_{k-1}(\alpha(\mathbf{a}), \beta(\mathbf{b})) & \gamma_k(\alpha(\mathbf{a}), \beta(\mathbf{b})) \end{bmatrix} \tag{3.4}$$

where each element represents a specific comparison, e.g. the comparison of first names, dates of birth, or home country. To simplify, we will denote $\gamma(\alpha(\mathbf{a}), \beta(\mathbf{b}))$ as $\gamma(\mathbf{a}, \mathbf{b})$ or simply γ. Γ denotes the set of all possible realisations of γ.

We denote the conditional probability $\gamma(\mathbf{a}, \mathbf{b})$ when $\mathbf{a} = \mathbf{b}$ with

$$m(\gamma) = m(\gamma(\mathbf{a}, \mathbf{b})) = P\left[\gamma(\mathbf{a}, \mathbf{b}) | (\mathbf{a}, \mathbf{b}) \in \mathbf{M}\right] \tag{3.5}$$

$$= \sum_{(\mathbf{a}, \mathbf{b}) \in \mathbf{M}} P\left[\gamma(\mathbf{a}, \mathbf{b})\right] P\left[(\mathbf{a}, \mathbf{b}) | \mathbf{M}\right]. \tag{3.6}$$

Respectively, we denote the conditional probability of $\gamma(\mathbf{a}, \mathbf{b})$ when $\mathbf{a} \neq \mathbf{b}$ as $u(\gamma)$.

Based on the comparison vector $\gamma(\mathbf{a}, \mathbf{b})$, we want to classify the pair $(\mathbf{a}, \mathbf{b})$ as a match, denoted A_1, possible match, denoted A_2, or non-match, denoted A_3. We do this using linkage rules, denoted by L, which we define as a mapping from the space of comparison vectors Γ onto a set of random decision functions $\mathbf{D}$ where

$$d(\gamma) = \{P(A_1|\gamma), P(A_2|\gamma), P(A_3|\gamma)\}, \quad d \in D, \gamma \in \Gamma \tag{3.7}$$

$$\text{s.t.} \sum_{i=1}^{3} P(A_i|\gamma) = 1 \tag{3.8}$$

Two types of errors can occur with each linkage rule: Type I and Type II. A Type I, i.e. a false positive error, occurs when we classify a non-matching pair as a match. This happens with probability

$$P(A_1|\mathbf{U}) = \sum_{\gamma \in \Gamma} u(\gamma) P(A_1|\gamma). \tag{3.9}$$

Similarly, a Type II error, i.e. a false negative, occurs when we classify a matching pair as a non-match and has probability

$$P(A_3|\mathbf{U}) = \sum_{\gamma \in \Gamma} m(\gamma) P(A_3|\gamma). \tag{3.10}$$

As stated by Fellegi and Sunter [9], an optimal linkage rule L is such a rule for which it holds that

$$P(A_2|L(\lambda, \mu)) \leq P(A_2|L'(\lambda, \mu)), \tag{3.11}$$

for every $L'(\lambda, \mu)$ where $\lambda = P(A_3|\mathbf{M})$ and $\mu = P(A_1|\mathbf{U})$. We can define the cost of the linkage error in various ways based on the application and requirements.

We calculate the posterior probability for each pair as

$$\hat{q}_{i,j} = P(\mathbf{M}|\gamma_{i,j}, \hat{\theta}), \tag{3.12}$$

where $\hat{\theta}$ is the the set of the estimated parameters m, u, and the prior probability p. With the posterior probability $\hat{q}_{i,j}$, we can sort the pairs into $\mathbf{M}$ and $\mathbf{U}$ using the limits R_{upper} and R_{lower}. One may also choose only to use one limit and sort all pairs directly into $\mathbf{M}$ and $\mathbf{U}$. This thesis only

uses one limit. Furthermore, in case blocking is used, any pair that does not belong to any block is automatically assigned to U.

In most cases, m and u can be estimated accurately from the data. Take, for example, an attribute with months as options. Seeing that $u = \frac{1}{12}$ is trivial. Estimating m is more complex, as it requires knowing the distribution of the attribute's values. One can use techniques like Expectation Maximization to estimate the values of m and u. Expectation Maximization is discussed further in Section 3.3. Similarly, p can be defined using prior knowledge or estimated from the data by defining some deterministic rules.

Seminal papers have further improved the Fellegi-Sunter model. Winkler [17] introduced the possibility of using multiple comparison levels per attribute. This allows us to compare an attribute using, e.g. exact match and a string similarity metric. Zhu et al. [18] proposed adjusting the weights based on the rarity of the compared value, i.e., performing term frequency adjustments. This may mean increasing or decreasing the weight of a variable based on its rarity.

There are limitations to the probabilistic method and the data. The model assumes that the attributes are independent and identically distributed (IID) [9]. However, this is rarely the case in real-world examples. For example, address, city, zip code, and country information are not independent. However, this assumption can be relaxed in most cases without severe consequences [19] or even with improved performance [20].

Another limitation is the dependency on accurate estimates of the probability parameters m and u. Without enough pairs belonging to M, the model can have trouble estimating the variables accurately, and the produced matches' quality may not be high [7]. In order to solve this problem, blocking, discussed in Section 3.2, is crucial. By using good blocking rules, we can filter out pairs that belong to U and thus have to perform less computationally complex actions on them [7].

3.1.2 Deterministic record linkage

Compared to probabilistic record linkage, deterministic uses an "all-or-nothing" approach. If a pair agrees on enough comparison levels, it is determined to be a match; if the pair does not agree, it is determined to be a non-match. All comparison rules and weighting are given equal importance when establishing if the pair is a match. Thus, the pairs will either be a match or a non-match; no possible matches exist for the

deterministic record linkage.

Deterministic record linkage is commonly based on rule-based comparison. These rules may be exact or partial matches using string similarity algorithms or edit distances. Such partial matching algorithms are presented later in this chapter. This thesis uses a deterministic record linkage model based on string embeddings.

In natural-language processing (NLP) tasks, one creates numerical representations of the strings. We call such representations "string embeddings". This representation aims to encode semantic and syntactic information about the string and its context. String embedding enables numerical algorithms and machine learning methods to process the data.

There are multiple ways to perform string embedding. The most basic way is one-hot encoding. It consists of creating a new feature for each string present in the corpus. Handcrafted features using, for example, n-grams and part-of-speech are a traditional and a step more advanced method. State-of-the-art applications use neural network-based algorithms to create and optimise embeddings [21].

This section discusses two different string embedding-based methods: TF-IDF and Word2Vec. We present the principles of these models and the modifications we make to the base models.

3.1.3 Term frequency–inverse document frequency

The "term frequency–inverse document frequency" (TF-IDF) method is a token-based method for describing how important words are in the context of both the document and the corpus. In the context of this study, the document represents a single string to compare, and the corpus is the collection of all strings. As the term "term frequency–inverse document frequency" suggests, the method is based on how much weight a term has in the document and how specific the term is. The term frequency [22] can be calculated as

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\Sigma_{t' \in d} f_{t',d}}, \tag{3.13}$$

where $f_{t,d}$ is the count of the term t in document d. In other words, each term is weighed based on how many times it occurs in the document.

The inverse document frequency [23] can be calculated as

$$\mathrm{idf}(t,D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{3.14}$$

where $|D|$ denotes the number of documents and $\{d \in D : t \in d\}$ denotes

the collection of those documents where the term t appears in them. We use the following modified version:

$$\mathrm{idf}(t, D) = \log\left(\frac{|D| + 1}{|\{d \in D : t \in d\}| + 1}\right). \tag{3.15}$$

We do this to solve the problem of division by zero when the term t is not in the corpus.

We then calculate the TF-IDF as the product of the two previous terms such that

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D). \tag{3.16}$$

Hashing trick

The hashing trick [24], also known as feature hashing, is a technique used to map text or words to numerical vectors. We use it to save memory when creating the embedded vectors due to setting a vector length N that is less than (or equal to) the corpus size $|D|$. Since we only need to save numerical indices and $N \leq |D|$ columns, the hashing trick saves memory when we store the sparse vectors created by the model. Additionally, no global dictionary needs to be precalculated, removing overhead. The downside of this method is that hash collisions might appear depending on the relative sizes of the corpus $|D|$ and the vector length N. Some hash collisions are inevitable if we choose a vector length N that is less than the corpus size $|D|$. However, when the size N is large, collisions become rare enough not to impact the downstream operations and results.

We calculate the hash of a document as follows. We choose a number N, which is the length of the new vector, i.e. the dimensions of the vector space. We define $\mathrm{hash}(t)$ as some hashing function. We can then define the function $h(t)$ as

$$h(t) = \mathrm{hash}(t) \mod N \tag{3.17}$$

where $\mod$ is the modulo operator. Thus, we calculate the element of the vector representation for each document D as

$$v_i = \Sigma_{t \in D} \mathbb{1}_{h(t)=i} \tag{3.18}$$

where v_i is the i:th element in the vector representing D.

3.1.4 Word2Vec

Word2Vec [25] is a natural-language process (NLP) used to compute dense string embeddings of words. The algorithm takes in words or strings of arbitrary lengths and outputs a numerical vector of predetermined length. We use these vectors for downstream processing in machine learning, data analytics, or other algorithms. We can use the Word2Vec model for tasks like text or word similarity comparison, recommendation systems, and sentiment analysis. Word2Vec is a widely used and effective algorithm for generating word embedding and can be used in a wide range of NLP tasks [25, 26].

Word2Vec is a neural network, or deep learning, based method. Mikolov et al. [25] implemented the so-called continuous-bag-of-words (CBOW) and skip-gram model architectures to lessen the computational complexity of the model. These methods allow the model to use context information around the word of interest without becoming as complex as a feed-forward neural network.

The Word2Vec algorithm is trained on a large corpus of words to provide accurate results. The algorithm user trains the algorithm to predict a word's context, i.e., neighbouring words (in case of skip-gram), or to predict the word based on its context. In the original paper, Mikolov et al. [25] trained the algorithm on the Google News corpus. This corpus contained the one million most frequent words of six billion possible tokens.

Word2Vec has some limitations. A standard limitation of any neural network-based algorithm is the need for extensive training data. Additionally, the algorithm does not handle out-of-vocabulary words. If the word does not exist in the training set, there is no way for the algorithm to learn how to embed the word. Additionally, the semantics of words spelt the same but have different meanings, e.g. "bat", as in the animal or a hitting bat, are not considered.

3.2 Blocking

In record linkage, we would like to compare every potential pair of records using a Cartesian join. However, this might be computationally infeasible since the Cartesian join scales as $\mathcal{O}(\prod_i N_i)$. Blocking is a technique that attempts to reduce the number of pairs compared while keeping the recall high [27]. There are multiple ways to perform it, depending on the frame-

work used. One method that this thesis uses is a rule-based method. The rule filters out pairs that do not match on a particular attribute. Another method is only to consider such pairs of vectors which have a distance d lower than the threshold T in some projected space. We discuss both these methods later in the thesis.

Blocking has multiple use cases. It lessens the computational load of any following operations applied. By using blocking, we reduce the comparisons to $\mathcal{O}(\prod_i Bn_{max,i})$ [27], where B is the number of blocks created and $n_{max,i}$ the size of the largest block created. It also improves the precision of the operation, although at the risk of the expense of the recall. Due to the potential costs, benefits, and repeatability, Christen and Goiser [28] recommend reporting the used blocking rules.

3.2.1 Locality-sensitive hashing

One way to perform blocking is through locality-sensitive hashing (LHS). Contrary to most uses of hashing that strive to minimise collision of the hashed strings, locality-sensitive hashing aims to maximise hash collisions of similar items [29]. A locality-sensitive hashing family $\mathcal{F}$ consists of a set of functions $h : M \to S$ that maps elements from the metric space to buckets $s \in S$. We define the following properties for the family $\mathcal{F}$: the metric space $\mathbf{M}$, the threshold $R > 0$, an approximation factor $c > 1$, and probabilities P_1 and P_2. The following conditions have to be satisfied for $\mathcal{F}$:

1. if $d(p, q) \leq R$, i.e. the distance between two points p and q are less than the threshold R, then the probability that $h(p) = h(q)$ is at least P_1.

2. if $d(p, q) \geq cR$, i.e. the distance between two points p and q are greater than the approximation factor c times threshold R, then the probability that $h(p) = h(q)$ is at most P_2.

The greater the probability P_1 is compared to P_2, the more interesting the family is.

Random bucket projection

Random bucket projection (RBP) is a form of locality-sensitive hashing [30]. It functions as a dimensionality reduction technique. Using randomly constructed buckets, or vectors, it projects high-dimensional data onto a lower-dimensional space.

Let us take the high dimensional vector $\bar{x} \in \mathbb{R}^n$, that we want to hash, and the bucket vector $\bar{r} \in \mathbb{R}^n$. We random sample the bucket vector elements from some distribution. A normal distribution, i.e. $r_i \sim \mathcal{N}(0,1)$, or a uniform distribution, i.e. $r_i \sim \mathcal{U}(0,1)$ are common choices.

Next, we need the hashed vector representation. The algorithm projects the original vector $\bar{x}$ onto the bucket vector $\bar{r}$'s coordinate system and takes the operation's sign. We perform this calculation by taking the sign of the dot product of the two vectors $\text{sgn}(\bar{r}^\top \bar{x})$.

We determine the dimension of the lower-dimensional representation by generating k bucket vectors such that $k < n$ and performing the above-mentioned operation for each vector. I.e. the hashed vector becomes $\bar{h} = \Big[\text{sgn}(\bar{r}_1^\top \bar{x}), \text{sgn}(\bar{r}_2^\top \bar{x}), ..., \text{sgn}(\bar{r}_{k-1}^\top \bar{x}), \text{sgn}(\bar{r}_k^\top \bar{x})\Big]$.

Since we lower the dimensionality of the vectors, we obtain a reduction in computational complexity and memory requirements for the following operations. Additionally, due to the dimension reduction, there is a higher probability that similar data points will have hash collisions and have the same lower dimensional representation.

3.3 Expectation maximization

Expectation maximisation (EM) is an iterative numerical method for finding the maximum likelihood estimate of a model [31]. One can use the EM algorithm when it is unfeasible to solve the maximum likelihood estimate directly. Thus, the EM algorithm works as a general-purpose algorithm for maximum likelihood estimation whenever other ways of optimising the likelihood function are complex. An example is mixture models, where the parameters of each component's underlying distributions and latent variables are unknown [32]. We would need to integrate over all possible values, rendering the option to optimise the likelihood directly infeasible. This work focuses on models with unknown parameters and latent variables.

The algorithm works in two steps: The E-step and the M-step. The E-step, *expectation* step, estimates the posterior probability of the value of the latent variables based on the current values of the parameters θ. The M-step, *maximisation* step, maximises the complete data log-likelihood of the unknown variables θ regarding the data and the latent variables.

We can summarise the algorithm as follows:

$$\theta^{(t+1)} = \arg\max_{\theta} \mathbb{E}_{\mathbf{Z}\sim p(\cdot|\mathbf{X},\theta^{(t)})}[\log p(\mathbf{X}, \mathbf{Z}|\theta)], \tag{3.19}$$

where θ^t is the unknown variable at interation step t, $\mathbf{X}$ is the known data, $\mathbf{Z}$ is the latent variable, and $\mathbb{E}$ is expectation operator. In the setting of this thesis, the latent variable is whether a pair is a match, and the unknown variables are, e.g., the m- and u-variables. The algorithm runs until some convergence criteria is met, e.g., a maximum number of iterations is reached or the difference between some variables for t and $t+1$ is less than a threshold value.

3.4 String comparison algorithms

The string comparison algorithms, or string similarity metrics, are popular measures often used in deduplication and record linkage tasks to identify potential matches between two datasets based on common attributes such as names, addresses, or dates of birth. In this section, we introduce the algorithms used in this thesis.

3.4.1 Jaro similarity

Jaro similarity, proposed by Jaro [33], measures the edit distance between two strings. The algorithm measures insertions, deletions, and substitutions required to transform one string into another. It does not penalise for transpositions (swaps) of characters close enough to each other. However, it is not a true mathematical distance metric as it does not satisfy the triangle inequality.

The algorithm compares the similarity between two strings by comparing the characters of each string and their placements, i.e., it considers the characters' order and any transpositions (swaps) needed to make the strings match.

The output of the algorithms is a number between zero and one. Zero means that the strings are a perfect match, and one means that the strings have no similarity at all.

The similarity is defined as

$$\text{sim}_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{3.20}$$

where m is the number of matching characters, $|s_i|$ is the length of string s_i, and t the number of transpositions. The metric considers two characters matching if they are the same and are no further apart than $\frac{\max(|s_1|,|s_2|)}{2} - 1$. The transpositions t are the sum of matching characters not in the right position divided by two.

The three parts of the second case in Equation 3.20 have specific interpretations:

1. The proportion of matching characters in string 1.

2. The proportion of matching characters in string 2.

3. The proportion of matching characters that are transposed. The number is divided by m and not the total characters because it only accounts for the characters that match.

The algorithm is useful for comparing strings containing typos, misspellings, or other minor differences. When strings are entered manually, e.g., through surveys and thus prone to human errors, string similarity algorithms are helpful when doing string matching and record linkage.

A feature of the algorithm, sometimes a shortcoming, is that it does not consider the length of the words the algorithm compares, which can lead to false matches in very long or short strings. It also cannot distinguish between transposition and some other type of error. The algorithm is also sensitive to outliers and, thus, is not robust.

Jaro-Winkler similarity

The Jaro-Winkler similarity is a variant of the Jaro similarity described above [17]. The Jaro-Winkler similarity emphasises strings that match at the start of the strings. The algorithm uses a prefix scale p to positively weight the prefix of length l of the strings.

The Jaro-Winkler similarity is calculated as

$$\text{sim}_{jw} = \text{sim}_j + lp(1 - \text{sim}_j), \tag{3.21}$$

where l is the length of the common prefix to consider and p is the weighting factor of how much the common prefix string is adjusted positively. l can take a maximum value of 4 and p a maximum value of 0.25, otherwise the similarity would become larger than 1.

Similarly to the Jaro similarity, the Jaro-Winkler similarity is not a true mathematical metric as it does not satisfy the triangle inequality or identity axiom.

3.4.2 Levensthein distance

The Levensthein distance [34] is an edit distance. It can be used to determine how many insertions, deletions, or substitutions are required to convert one word into another. The Levensthein distance follows the triangle inequality. Additionally, the following bounds can be given on the distance:

1. The distance is, at most, the length of the longer string.

2. The distance is zero if and only if the strings are equal.

3. The distance is at least the differences in the string sizes.

The Levensthein distance can be calculated as

$$\operatorname{lev}(a,b) = \begin{cases} 0 & \text{if } |a| = 0 \vee |b| = 0 \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \operatorname{head}(a) = \operatorname{head}(b) \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise} \end{cases} \tag{3.22}$$

where a and b are the strings to compare, head is the first element of the string, and tail is all but the first element.

Since the Levensthein distance is not normalised, longer strings with multiple errors give larger distances than short strings with fewer errors. For example, "Hello" and "Helo" have a distance of one while "Greetings" and "Greting" has two. However, these two examples might look equally right or wrong for a human since a large percentage of both are correct.

The computational complexity of the Levensthein distance can become

a problem, especially in long strings. Backurs and Indyk [35] show that for two strings of length n, the Levensthein distance cannot be calculated in less time than $\mathcal{O}(n^{2-\epsilon})$ for any ϵ greater than zero. This holds unless the strong exponential time hypothesis is false. Thus, performing many comparisons using the Levensthein distance is a computationally complex task and may be unfeasible regarding computation time or memory.

3.5 Phonetic encoding

When we perform record linkage on names, we must consider that names may be pronounced the same way but spelt differently. In this case, phonetic encoding is useful for us. Phonetic encoding encodes homophones, words pronounced the same but with differing meanings or spelling, to the same encoding. In our case, we may have names that may be misspelt but pronounced the same. For example, we pronounce "Steven" and "Stephen" the same but spell them differently.

3.5.1 Soundex

The Soundex algorithm is a phonetic encoding algorithm. Since it was patented in 1918, many variations have been created and used. A version used by the U.S. government is maintained by The National Archives and Records Administration (NARA) [36]. Here we will shortly explain the general concept of the algorithm.

The Soundex code consists of a letter and three digits. The letter is the first letter of the original string. From the remaining string, we disregard all vowels and, depending on the version, some other letters. For example, the NARA version also disregards H and W. Finally, we encode the three first letters of the remaining string according to a mapping table. The mapping of NARA [36] can be seen in Table 3.1. If the string is too short, i.e. it does not have three letters left, the code will be padded with zeroes at the end to create the four-character long code. For example, Christian becomes C623, and Ida becomes I300.

Table 3.1. The Soundex letter mapping table [36].

Number	Represents the letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

3.6 Performance metrics

To be able to compare the outcome of different methods, performance metrics are necessary. This study uses three performance metrics: recall, precision and F_1-score. These metrics measure the outcome of some (binary classification) model or method compared to a ground truth. We use these metrics to account for the size imbalance of the sets M and U. The performance metrics are presented shortly in this section.

3.6.1 Precision

Precision measures the validity of the outcome or, in lay terms, how large a part of the returned values belongs to the ground truth. We can calculate the precision as

$$\text{Precision} = \text{Pr} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}. \tag{3.23}$$

The metric ranges between zero and one, where higher indicates a better result.

3.6.2 Recall

Recall measures the completeness of the outcome or, in lay terms, how many algorithm-indicated positives belong to the ground truth. Recall can be calculated as

$$\text{Recall} = \text{Rc} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}. \tag{3.24}$$

and ranges between 0 and 1, where higher is better.

3.6.3 F_1-score

F_1-score is the harmonic mean of precision and recall. The F_1-score is defined as

$$F_1 = \frac{2}{\frac{1}{\mathrm{Rc}} + \frac{1}{\mathrm{Pr}}} = 2\frac{\mathrm{Pr} \cdot \mathrm{Rc}}{\mathrm{Pr} + \mathrm{Rc}}. \tag{3.25}$$

and ranges between 0 and 1, where higher is better.

4. Implementation

This section presents the implementation of the theoretical frameworks, methods, and algorithms in the setting of this thesis. Section 4.1 gives an introduction to the technical setup used for the programmatic work conducted in relation to the thesis. Section 4.2 and Section 4.3 present the implementation of the deterministic, respectively, probabilistic record linkage methods. Finally, we discuss the usage and creation of the ground truth data set used to evaluate the methods used.

4.1 Setup

One of the critical parts of the study is the technical setup used to run the algorithms. Due to the nature of the business at a bank, the robustness and security of the computing environment are of high importance. This exceptional environment means that the user must thoroughly vet hardware or software before use.

The setup used for computation is an in-house Hadoop cluster running Spark 2.4 [37] and Python 3.7. The author uses a Livy endpoint connection from Cloud Pak For Data (CP4D) to connect to the cluster and its data. Thus, we use CP4D as a code editor and interface while all data is located on the Hadoop cluster and the calculations run on it.

Hadoop has a separate UI and server for saving logs of jobs. This particular setup has some derivative effects on the logs, and understanding the impact is essential, especially when working with software that outputs log messages instead of printed strings. Additionally, one cannot directly output graphical objects and plots created by the Hadoop cluster into CP4D through the Livy connection.

4.2 Deterministic record linkage implementation

This section discusses the pipelines and implementations used for the deterministic record linkage implementing the string embedding-based methods discussed in Section 3.1.2. The pipeline consists of the following parts:

- Data preprocessing
- String embedding
- Blocking through random bucket projection
- String similarity comparison

First, we present the pipeline in general and then each part in depth. Figure 4.1 presents a diagram of the pipeline.

The pipeline takes two separate data sets with two attributes as input: the ID and the entity describing string, i.e. the *entity string*. The IDs remain untouched during the pipeline as their purpose is to connect back to the original record and data set. The entity strings go through the preprocessing steps to harmonise the data and filter out non-informative parts and noise. The strings are then tokenised, i.e. split into vectors consisting of the words of the string. We then embed these tokenised vectors into numeric vectors. We divide the data sets into blocks of pairs to reduce the number of potential matches. Finally, we compare each pair's *entity strings* using a string similarity method and those above a certain threshold are considered matches.

4.2.1 Data preprocessing

The preprocessing stage of the pipeline is crucial for the method; the methods used for embedding the vectors are sensitive to the content of the corpus. Since we use the methods for names of either natural or juridical persons, the strings mostly contain names and rare words. Due to the rarity, we wish to, in most cases, preserve the original form of the string. However, some standardisation steps are necessary.

Standardising the case of the letters, in our case to lower case, makes all names and strings written with the same characters count as the same

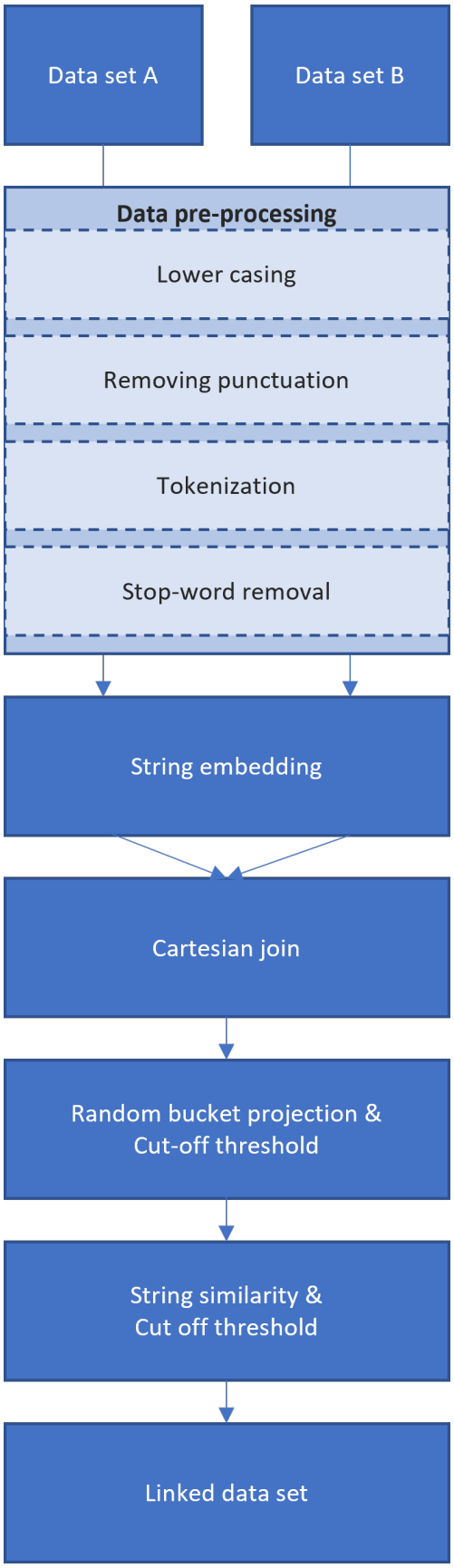


Figure 4.1. A diagram of the string embedding-based deterministic record linkage pipeline.

word. This standardisation uses the Python string method "*lower*" provided in the base Python package.

Punctuation and some special characters are removed, such as ";" (semi-colon), "," (comma), "." (dot), ":" (colon), and "\" (backslash). Other special characters, like "&" (ampersand) and "–" (hyphen), are left since they can be essential parts in some names, e.g. double names of natural persons. We execute the removal using the "*regexp_replace*"-function applied to the entity string columns of a PySpark data frame.

Next, we tokenise the strings. Any characters separated by whitespaces, hyphens, or ampersands are considered separate words and tokenised. In other words, in the upcoming embedding step, we encode all strings separated with the aforementioned special characters from string to a numeric value. We perform the tokenisation using the hashing trick, discussed in Section 3.1.3, to make the memory requirement smaller. We use PySparks "*RegexTokenizer*"-function for this.

The definition of stop words is commonly used words in the used language. E.g. "the", "a", and "is" are common stop words in English. Such words usually contain little to no information about the string or its context. Removing stop words lessens the computational load and creates a more robust model. In the context of this work, the concept of stop words is more challenging. When talking about natural persons, stop words may be titles, like "Mr." or "Ms.". For juridical persons, "Incorporated", "LLC", or "PLC" are examples of such words in English. When defining what words to remove, it does not matter only which language and country is in question but also the context of the corpus. We accomplish the task using PySparks "*StopWordsRemover*" function.

We can apply the preprocessing step to transform strings such as "Nordea Bank OYJ" or "AB Nordea Bank" into the vector $\begin{bmatrix}\text{"nordea"} & \text{"bank"}\end{bmatrix}$. This vector can then be given to the embedding step of the pipeline to transform into a vector of numeric values.

4.2.2 Embedding the strings

This section presents the central part of the pipeline, the embedding. The theory behind the two methods used to create the embeddings, TF-IDF and Word2Vec, are presented in Sections 3.1.3 and 3.1.4 respectively. Furthermore, this section presents the methods' usage, training, and tuning. PySpark implements both methods, which are named "*HashingTF*" and "*Word2Vec*" respectively.

Term frequency–inverse document frequency

PySpark supplies two different ways of counting term frequencies of words: a traditional bag-of-words style, where the algorithm assigns each word a column in a matrix, or a method utilising the Hashing Trick, presented in Section 3.1.3. These are implemented in PySpark as "*CountVectorizer*" respectively "*HashingTF*". Additionally, PySpark provides the "*IDF*" function to calculate the inverse document frequency.

As previously discussed in Section 3.1.3, there are clear benefits and risks of using the hashing trick over the bag-of-words approach. The hashing trick reduces the computational load due to lessened memory requirements and overhead. Due to the benefits, we use the method implementing the hashing trick. The risk of using the hashing trick is its impact on the model outcome; hash collisions may cause poor model performance. In the context of this work, this is a significant factor. It is, however, essential to note that the output of both aforementioned methods is sparse vectors.

The "*HashingTF*" has a critical parameter: the dimension of the embedded vector. This parameter directly affects the amount of potential hash clashing and memory usage. Theoretically, one would like to maximise this number while keeping the model within memory limitations to minimise hash clashes. In practice, one must ensure that the pipeline is stable on each run, i.e. does not crash due to running out of memory.

The algorithm's output is a sparse vector containing the TF-IDF weights of each token in the previously produced vector. For example, $\begin{bmatrix}\text{"nordea"} & \text{"bank"}\end{bmatrix}$ can become $\begin{bmatrix}0 & 1 & 0 & 0 & 1\end{bmatrix}$ if we use a vector length of five.

Word2Vec

The Word2Vec method, in contrast to TF-IDF, produces a dense distributed vector representation of the terms in the corpus. As discussed in Section 3.1.4, the Word2Vec model considers context around the words to create the document's distributed representation. Specifically, the implementation available in PySpark is based on the skip-gram model.

The model has the following hyperparameters, which require defining:

- the dimension of the output vectors
- the minimum number of appearances of a word to be included in the model

- the maximum allowed document length
- the maximum number of iterations of the model
- the step size of the optimiser

Of these, the dimension of the vectors is critical. As discussed in the previous section, the larger the output, the higher the required memory and amount of information possible to embed into the vector. The maximum allowed document length, the maximum number of iterations of the model, and the step size of the optimiser are kept at their default values, defined by the function, to lessen the complexity of the tuning step.

The minimum token appearance, while usually an important hyperparameter to reduce noise, has a different implication in the setting of this thesis. Since we apply the algorithm to names, infrequent words are significant as some may uniquely identify their respective entities. This context-specific irregularity leads us to keep the parameter as 0, allowing all words, in order not to filter out any vital information.

4.2.3 Blocking through random bucket projection

Due to the number of candidate pairs, e.g. approximately $14.9 \cdot 10^9$ for the natural persons, it is unfeasible to use computationally complex similarity measures before or after embedding the strings. Thus, we apply blocking through locality-sensitive hashing using random bucket projection to reduce the required comparisons. PySpark implements this in the "*BucketedRandomProjectionLSH*"-function. We discuss the technique in depth in Section 3.2.1.

The blocking aims to increase precision and reduce computational complexity while retaining as high a recall as possible. Thus, we aim to filter out as many pairs that do not match while keeping all true matches. We do this by assigning every vector created in the proceeding step to a bucket of length j. Once we have these vectors, we discard those pairs with an Euclidean distance greater than d.

4.2.4 String similarity comparison

As a final step in the embedding-based method, we use the similarity of the strings to compare the pairs. This step does not improve the recall but aims

to improve the precision by filtering out pairs with unintentional hash clashes due to the vector embedding and random bucket projection. Since the previous steps have aimed to filter out most of the false negatives, we can now apply more computationally complex algorithms to the remaining pairs.

The tokenised string will be compared instead of comparing the original entity string. We order the vector of tokens alphabetically not to have the order of the words impact the outcome. Next, we combine the vector elements into a string with whitespaces separating it. Finally, we calculate the string similarity metric between the two strings. All pairs below a user-defined threshold are filtered out.

The core of this step is the string similarity algorithm used for the comparison. Choosing which metric to use ought to be derived from the result one wants to achieve. We can consider two categories of algorithms. Firstly, those that return a normalised similarity value between 0 and 1. Secondly, edit distances that return an integer describing how many operations are required to transform one string to the other. Contrary to the definition, the algorithm's implementation may turn the scale around such that the returned value is $1 - \mathrm{sim}_j$. Thus, 1 equals a perfect match and 0 a complete non-match.

The Jaro and Jaro-Winkler string similarities, discussed in Section 3.4.1, are good candidates for the choice of metric. For the implementation, we use the JaroWinkler-package's *"jarowinkler_similarity"*-function [38]. The function takes in any string or array of objects and compares them using the string similarity algorithm. Contrary to the theoretical description of the methods, the package's functions reverse the similarity scale so that one equals a perfect similarity and zero no similarity.

4.3 Probabilistic record linkage implementation

This section discusses the setup and implementation of the probabilistic record linkage solution. The solution uses Splink [39], a python package based on the research by Enamorado, Fifield, and Imai [7] and developed by the British Ministry of Justice. Splink uses the probabilistic record linkage theory created by Fellegi and Sunter [9] and the expectation-maximisation algorithm [31] to estimate the unknown linkage variables m and u. We present the theory in Section 3.1.1. In this section, we first give an overview of the workflow of the algorithm and then present the core concepts in

more detail.

For each attribute we compare, we start by defining the comparisons, i.e. the case statements. We can define the statements using Splinks built-in comparisons or define them manually. Each comparison will have at least two case statements, of which one is the case for all comparisons not in any other case statement. However, one can have as many statements as necessary, such as multiple string similarity comparisons.

We then evaluate the prior probability that a random pair is a match. We do this by defining a set of deterministic rules that retrieves a sub-sample of all possible pairs. We construct these rules to have as high precision as possible. We then estimate the recall of the sub-sample. Based on this, we can estimate how rare a random match is in the data and, thus, the prior probability of a pair being a true match.

Next, we get an initial value for u for each case statement. We do this by taking a random sub-sample of the Cartesian product of all pairs and estimating u based on the sub-sample. We assume that true matches in this sub-sample are very unlikely and that the pairs are most likely non-matches.

We employ the EM algorithm to define m and tune u. We do this by defining local blocking rules. Each rule creates a sub-sample on which we apply the EM algorithm to tune the m- and u-variables on a case-statement level. The algorithm only tunes those attributes that each rule does not use to create the sub-sample. For example, if we create a sub-sample using a blocking rule such that the first name needs to be an exact match between the pair, then we do not tune the variables related to the first name. The algorithm moves to the following rule once a maximum amount of iterations is reached or the algorithm converges, i.e. the change in model parameters between iterations is less than a set value.

Finally, we use the trained model to predict the match probabilities. We do this on a sub-sample of the data defined using the global blocking rules. We create the comparison vector for each pair in the sub-sample, use the trained variables to assign a weight to the vector elements, and sum up the values to get the match probability.

Many of the previous steps suggest using sub-samples instead of the complete data. This is due to two reasons. Firstly, the sub-sample needs to be small enough to run in a feasible amount of time in the environment. Secondly, the sub-sample created by the blocking rules must contain between 1% and 99% true matches to function correctly, i.e., to properly tune

the variables.

4.3.1 Case statements

Case statements are at the core of the probabilistic model. The case statements are a series of levels of comparisons for each attribute that describe the level of agreement. Building these with the desired outcome in mind is vital. We can view the case statements as a series of if-else statements, each progressing in descending order of agreement. The exception to this is missing data, which is handled separately.

Let us take as an example the case statements of the "first name" attribute. An example based on the "first name" attribute could look something like this:

1. If either data point is missing, the agreement level is set to -1

2. Else, if both data points match exactly, set the agreement level to 3

3. Else, if the data points have a Jaro-Winkler similarity of 0.94 or higher, set the agreement level to 2

4. Else, if the data points have a Jaro-Winkler similarity of 0.88 or higher, set the agreement level to 1

5. Else, set the agreement level to 0

We can make a complete model by creating statements for each attribute to compare the records.

We can use different comparisons depending on the attribute type and by utilising context information. For example, character and string-based attributes are suitable to compare using string-based similarity metrics. We can compare dates using either time-based differences, where a difference in days is more similar than a difference in months, or string similarity, where each differing digit is weighted equally independently of its position or meaning.

Choosing what kind of similarity to use might depend on the context. In the case of a date, using string similarity might be beneficial if we suspect a typo due to manual data ingestion. In contrast, time-based similarity can be suitable if we compare records from two systems that possibly return

different dates that are temporally close to each other.

The following parts will discuss the chosen case statements and the reasoning behind the choice for each attribute. The comparison vector consists of the results of comparisons between the following attributes:

1. First name

2. Middle names

3. Last name

4. Date of birth

5. Country

Note that all parts handle missing data or null values the same: by not assigning any weight to that comparison. Thus, we will not mention it hereafter. We present a complete list of the case statement in Appendix C.1.

We compare first and last names using identical settings. We use three different comparison levels: exact match, soundex match, and all other comparisons. Additionally, we apply term-frequency adjustments to both comparisons.

We compare the middle names using an exact match, Levensthein distance for thresholds at one and two, and all other comparisons. We do not use Soundex for middle names as the numbers of middle names vary and may cause unexpected interactions. Additionally, we apply term-frequency adjustments to the middle name comparisons.

The algorithm compares the date of birth at different temporal levels. Note that all the dates are in the format "YYYY-MM-DD". Due to us filling in missing data, we artificially create a lot of first of months and first of January dates. First, we consider exact matches between the dates. The second comparison level compares the year and month, such that the seven first characters must match and either of the two dates' last two characters are "01". Thirdly, we compare the year similarly to the previous comparison, i.e., the four first characters must match and either of the dates' last five characters are "01-01". Finally, we consider all other comparisons. Due to filling in the missing data, we apply term-frequency adjustments to the comparison.

We perform the country comparison as an exact match, as the country column contains no deviating values, and apply term-frequency adjustments.

4.3.2 Blocking

Blocking is a technique for reducing potential pairs to compare, discussed in Section 3.2. It is a core functionality of Splink, as blocking allows using large amounts of data. By using well-formulated rules for creating blocks, we can keep the number of pairs to compare low to reduce computational complexity.

One should define the blocking rules so that the blocks created filter out as many non-matching pairs as possible while keeping as many matching pairs as possible. Additionally, for a computational advantage, the blocks should be of similar size as the algorithm's run-time depends on the largest block. Each pair will only be evaluated once despite how often it appears in different blocking rules. Due to this solution by Splink, one does not have to worry about creating duplicate pairs and impacting the evaluation time.

We use global blocking rules [39] to limit the number of pairs we evaluate in the prediction phase. We construct these rules such that they maximise recall. The tuned model then weighs and sums the attributes to predict matches and non-matches.

We use local blocking rules [39] in the expectation-maximisation step of the algorithm. We maximise precision in the construction of the local blocking rules to have a balanced label occurrence. The parameters are trained separately for each rule. The EM algorithm is run until convergence for each rule's training. Tuning is turned off for the u- and m-variables corresponding to attributes used to create a rule. We cannot tune the variable estimates for those variables since all pairs' comparison levels would be equal.

4.3.3 Term-frequency adjustments

Splink also includes the option to adjust the evidence level based on the frequency of the term found that matches. Down-weighting common options gives more significant evidence to pairs matching rare values, i.e., attributes with cardinality skew.

Let us take last names as an example. The Finnish last name 'Korhonen'

is, as of 2023, the most common last name [40]. This difference in rarity means that similarity on 'Korhonen' is less evidence of a match than a more uncommon last name. The model infers the term-frequency adjustments from the data during training.

4.4 Ground truth

Standard data metrics are helpful tools for comparing the performance of different methods, models, and choices of parameters. We use recall, precision, and F1-score to evaluate the results. However, the aforementioned methods require labelled data and due to the nature of our study, our data is unlabelled, and the methods are unsupervised.

We construct ground truth data sets of the data to solve the problem. We construct a ground truth for each of the two problems: the juridical person records and the natural person records. The ground truth will consist of all Dow Jones records and their corresponding matches from the DnB data set.

Juridical persons
We use a simple matching scheme that removes certain generic words and abbreviations, like "AB", "OY", "LTD", etc. and transforms the string to lowercase. The complete list of words removed can be found in Appendix A. We perform an exact match join on the two data sets to find matches. The remaining companies in the DJ subset are then gone through by hand and matched in the DnB dataset. The ground truth consists of 1694 matches; all other pairs are non-matches.

Natural persons
We use a deterministic matching scheme to construct the ground truth set. The author received the scheme from financial crime prevention experts familiar with the data. We join the data sets by matching on first name, last name, birth year, and birth month. The ground truth consists of 413 matches; all other pairs are considered non-matches.

5. Results

The section presents the results of the used algorithms divided into two main categories: deterministic and probabilistic methods. Section 5.1 first presents results related to the dimensionality of the data and how that affects the choice of method. Secondly, Section 5.2 presents the challenges faced due to the technical environment and infrastructure presented in Section 4.1. Section 5.3 first presents the results of the parameters for each of the embedding-based models and then presents the results of the optimal models. Section 5.4 presents the results of the probabilistic model. Finally, Section 5.5 compares the deterministic and probabilistic methods and the results of the respective model to each other.

5.1 Data dimensionality

Among the two methods, the deterministic and the probabilistic, used in this study, there are vast differences in requirements and functionality based on the dimensionality of the data. These differences and their impact will be discussed in this section.

5.1.1 Deterministic record linkage

As discussed in Chapter 4, the embedding-based methods take a single attribute, the *entity string*, as input for the record linkage. The additional ID is only used to link it back to the original dataset and record. This means that any information to be used for the linkage needs to be added to the string. This section reviews the results of the amount of information injected into the entity string.

The baseline information used in all cases is the juridical person's name with the "stop words" removed. The list of removed words can be found in Appendix A. The following attributes can also be added to the *entity string*:

address, city, and country.

The results show that any additional information outside the company name leads to a worse-performing algorithm where precision and recall decrease. There may be multiple causes for this: the model highlights unique information through the TF-IDF and Word2Vec algorithms. Adding information common among many records, such as the city or country, will not add any unique information. Alternatively, when the strings get longer minor differences in how the information has been written may add up and lead to differences between the strings. The address, for example, may have differing levels of information: one may have the name of the road and the street number and the other the road, street number, floor number, and apartment number.

The best result is gained by not including anything but the name in the string. If one is sure that the data is entered identically in both data sets, one could see an increase in performance by using more information in the entity string.

5.1.2 Probabilistic record linkage

At the core of the probabilistic record linkage method is the summation of the evidence given by different attributes. For this method to be usable, multiple attributes are required. Using the method with a single attribute is similar to writing a chain of if-else clauses for a deterministic match. Additionally, if the data has more attributes, it becomes easier for the user to create blocking rules and engineer the algorithm's training.

Furthermore, as per Enamorado, Fifield, and Imai [7], the algorithm will only perform well if the two data sets overlap considerably. However, well-constructed blocking rules may be a solution to this problem.

5.2 Environmental challenges

We can attribute many issues faced during this work to the restrictiveness of our environment. As discussed in Section 4.1, the work setup is both very restricted and outdated in the context of the speed at which the field of machine learning and data science develops [41]. Without a great understanding of the particular setup, its strengths, and its weaknesses, work using the setup may become very challenging. To give an example of the challenges encountered, consider that the average time to install a

Python package is two months.

One of the most significant challenges is to use pure 2.4 PySpark. Due to the environment, we cannot use additional Java Archives files (.jar) [42], and hence any non-native functions. The new DataFrame API with its methods, introduced in Spark 3.0, is also unavailable. Not using the current major version makes finding help and guidance more challenging, adding a layer of difficulty to engineering and programming.

Writing new distributed Spark functions is complex and requires advanced computer science knowledge, programming of parallel computers and distributed systems. Hence, many experienced programmer resources are required to build an algorithm that would support any needs that may arise. As this is outside the scope of the thesis, only native PySpark functions have been used.

Due to the setup described in Section 4.1, the logs of anything run on the cluster are not returned to the interface but saved to separate log files. Thus, a significant part of Splinks informative messages, which are output as logs instead of printed strings, are saved to the Hadoop log server instead of being shown to the user. Performing iterative tests on the model is challenging and time-consuming since the user must download the logs and search for the desired output in the log file.

Additionally, graphical objects and plots, on which Splink relies to convey information to the user, are impossible to view. This restriction means that the user has to use an alternative environment or setup to perform the analytical part of the creation of the model. These alternative environments are, however, only suited for running a sub-sample of the data due to hardware restrictions. This also means that the user cannot be sure whether the model is performing as expected when moving from a subset of the data on which they perform the exploratory modelling to the complete data.

When running Splink on PySpark, Splink uses the Spark SQL engine as the backend for running calculations. Hence, any restrictions or shortcomings of the Spark SQL engine also apply to Splink, including existing implementations of used algorithms and the lack of algorithms. For example, Spark SQL does not natively have a Jaro or Jaro-Winkler string similarity algorithm implemented. Splink has implemented a custom Java Archives file which introduces needed algorithms. However, as previously mentioned, we cannot add this to the environment to solve the problem. Thus, we must only choose case statements and blocking rules that do not

use algorithms not implemented in Spark SQL Engine.

5.3 Deterministic record linkage results

This section presents the results of the deterministic record linkage. First, we examine the impact of the different parameters on the time requirements for the algorithm to run, the memory required for the final result and the performance metrics. We separately present the TF-IDF-based and Word2Vec-based pipelines' results. Consecutively, we compare the results of both to each other. Finally, we present the optimal model performance on the two record linkage problems: juridical and natural persons.

The parameter impact on the data set of juridical person records, presented in Chapter 2, is investigated. The data related to juridical person records are chosen due to the previously discussed data dimensionality challenges, as the author hypothesises that the model will perform worse on the natural person-based data.

The complete pipeline has eight different parameters: *"Length of embedded vector"*, *"Length of bucket"*, *"Number of hash tables"*, *"Use address information"*,*"Use city information"*, *"Use country information"*, *"String similarity threshold"*, and *"RBP distance threshold"*. The pipeline includes constructing the entity string, performing RBP, and comparing string similarity. We can see additional information about the parameters and their values in Section 5.3. Based on the discussion in Section 5.1.1, we descope the Boolean parameters related to including additional information in the *"Entity string"*-field. Additionally, two of the values do not seem to impact the result outside of being non-zero: *"Length of bucket"* and *"Number of hash tables"*. Running multiple iterations of our model is a time-consuming task, and we hence also descope parameters *"Length of bucket"* and *"Number of hash tables"*. We end up with the three following parameters: *"Length of embedded vector"*, *"String similarity threshold"*, and *"Distance threshold"*.

We evaluate the algorithm and the choices of parameters against five metrics:

1. Relative time: The wall time needed to execute the complete algorithm divided by the fastest iteration of the algorithm.

2. Recall

Table 5.1. The parameters of the interactive deterministic approach. We present each parameter, its data type, and possible values.

Parameter	Data type	Possible values
Length of embedded vector	Integer	[1 .. ∞)
Length of bucket	Float	[0,1]
Number of hash tables	Integer	[1 .. ∞)
Use address information	Boolean	{True, False}
Use city information	Boolean	{True, False}
Use country information	Boolean	{True, False}
String similarity threshold	Float	[0,1]
RBP distance threshold	Float	[0, ∞)

3. Precision

4. F1-score

5. Memory: The memory required for the final data frame.

We give detailed explanations of the recall, precision and F1-score in Section 3.6. The effect of the parameter values on the time is complex to determine; a bias in the data may be attributed to differences in the cluster load depending on the time of day. Thus, even though we may see a trend in the variables affecting time, it may be attributed to the cluster's load. We are cautious when interpreting results and outliers in the time metric.

The algorithms will be run 1000 times for each of the two models, the TF-IDF- and Word2Vec-based models. We draw a value for each parameter from a uniform random distribution for each iteration. The value of the string similarity threshold is in the interval $[0, 1]$, and the value for the RBP distance threshold is a square root of the integer values in the range $[0, 9]$. For the TF-IDF model, the feature vector length is a power of 10, and we draw the exponent from the interval $[2, 6.69]$, i.e. the length is in the range $[100, \sim 5000000]$. For the Word2Vec model, the feature vector length is an integer in the range $[1, 7]$. These ranges of values for the feature vector length and RBP distance threshold have been chosen based on initial tests indicating that larger values than the previously given ones lead to errors and crashes due to the system running out of memory.

5.3.1 TF-IDF-based pipeline

This section highlights the parameters' effect on the TF-IDF-based embedding pipeline. The results will be measured using the five performance metrics: relative time, F1-score, precision, recall and memory usage of the

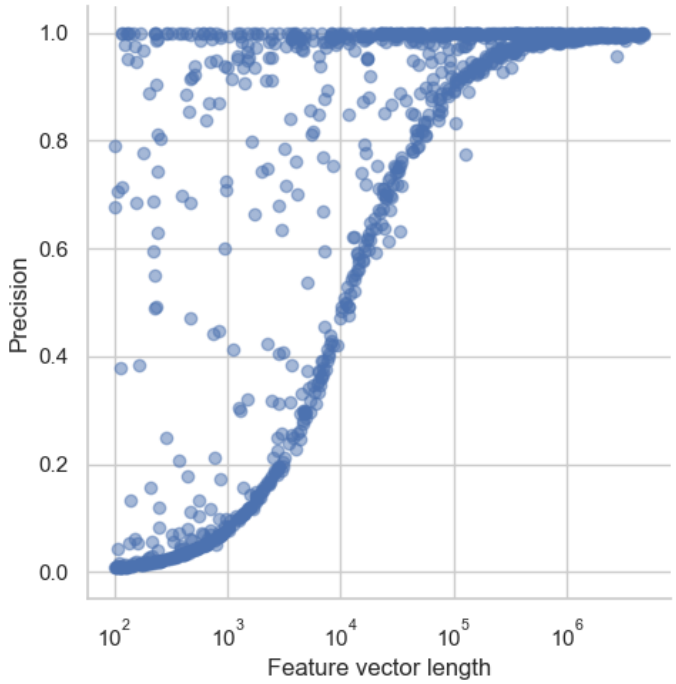


Figure 5.1. The TF-IDF-based algorithm's precision as a function of the feature vector's length.

final data frame. All plots related to the parameters of the TF-IDF pipeline can be seen in Appendix B.2 in addition to that presented in this section.

Feature vector length

This section presents the effect of the feature vector length on the performance metrics. Notice that the x-axis is logarithmic in all the figures in this section.

In Figure 5.1, we can see the precision as a function of the feature length. We observe an evident positive trend between the variable and the metric. We can see that the feature vector length creates a minimum possible precision with a logistic-like form. However, we can see a few outliers lie below this curve.

The F1-score displays a strong positive correlation between the length and the F1-score. This phenomenon can be seen in Figure 5.2. The F1-score is mainly affected by the strong effect of the feature vector length on the precision.

Figure 5.3 shows the memory usage as a function of the feature vector length. We find no significant trend between the variables. Due to the sparsity of the vector, the size of the vector may become large but still require only little memory, as only a fraction of the elements are non-zero, and memory usage can thus be handled efficiently.

To summarise the results of the effects of the feature vector length, we

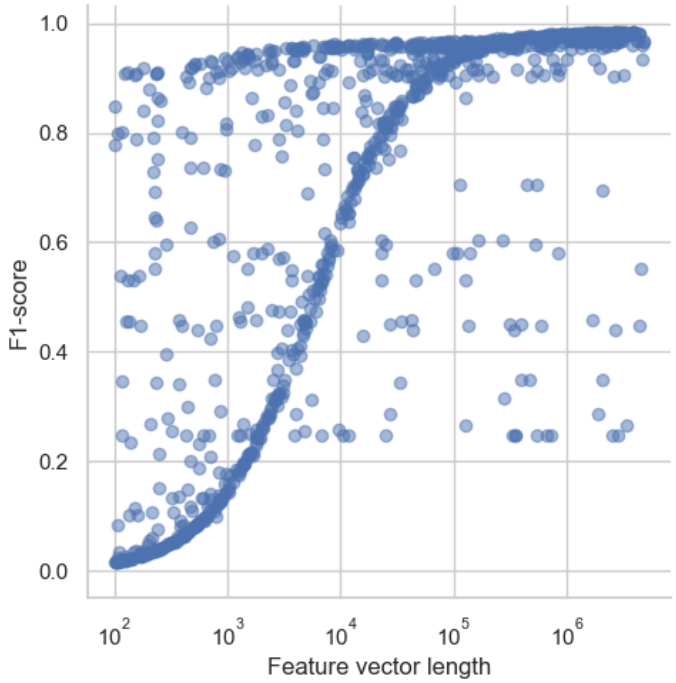


Figure 5.2. The F1-score of the TF-IDF-based algorithm as a function of the feature vector's length.

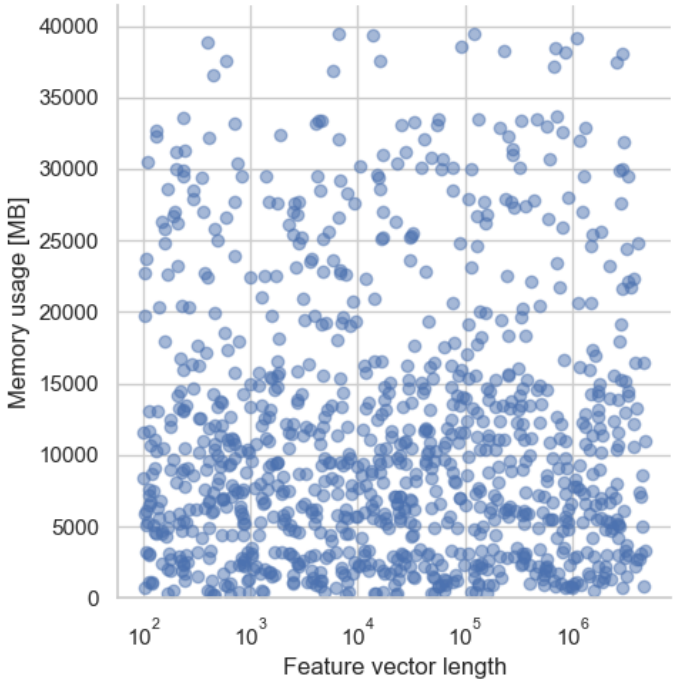


Figure 5.3. The memory usage of the final data frame of the TF-IDF-based pipeline as a function of the feature vector's length.

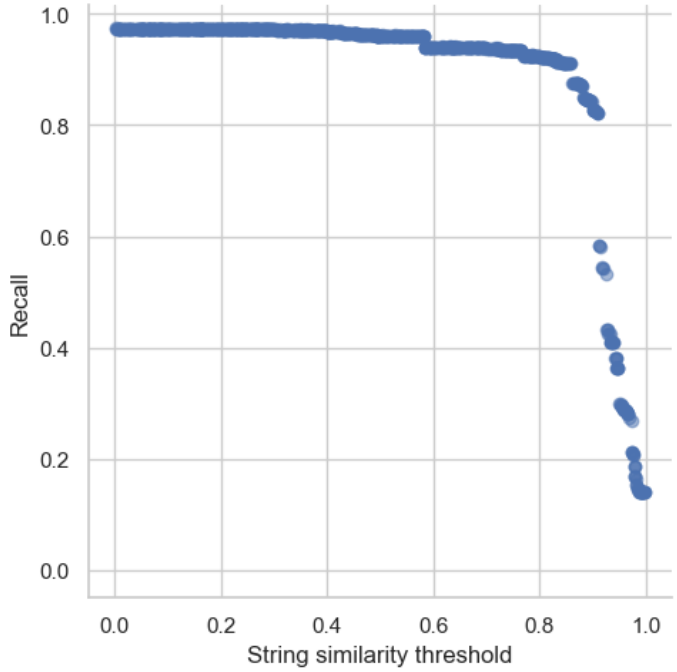


Figure 5.4. The recall of the TF-IDF-based algorithm as a function of the string similarity threshold.

can conclude that increasing the feature vector's length positively impacts the precision. Additionally, at the high end of the range, the relative time increases as well. This is due to the hash clashes described earlier; with longer feature vectors, fewer hash clashes occur. The user is to consider this trade-off based on the size of the problem at hand and the desired outcome.

String similarity threshold

This section examines the impact of the string similarity threshold on the performance metrics. Figure 5.4 depicts the recall as a function of the threshold. The trend is evident: a higher threshold decreases the recall. We can see a slight drop in the recall at approximately 0.59 and a more significant one after 0.82, a negative exponential trend. This drop can be explained by the fact that the string similarity threshold only filters out pairs, so the fewer that are filtered, the higher the chance that more true positives remain in the set.

In Figure 5.5, we can observe the effects of the string similarity threshold on the precision. After 0.6, the worst-case precision increases linearly. Furthermore, at approximately 0.92, the precision is equal to 1. The context explains this phenomenon of a cut-off threshold: the higher the string similarity, the more likely the two strings are to be a true match. Additionally, in Figure 5.6, we can see that the feature vector length plays

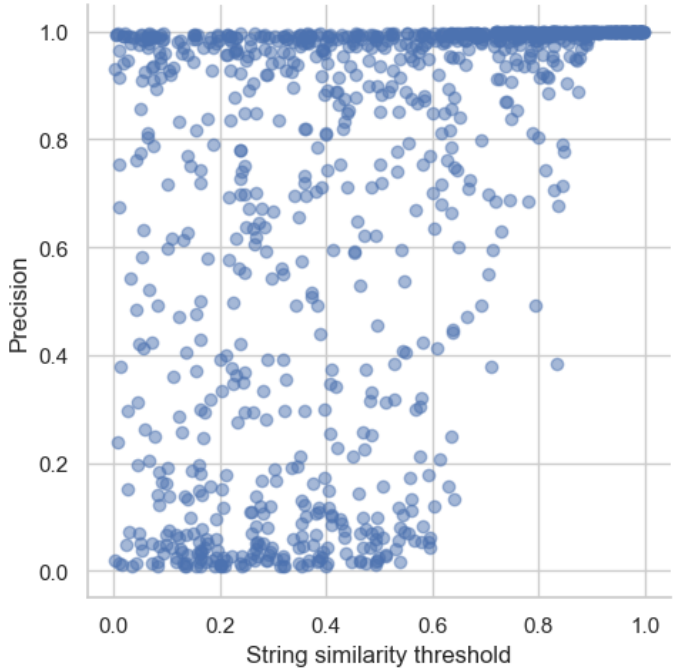


Figure 5.5. The precision of the TF-IDF-based algorithm as a function of the string similarity threshold.

a vital role in the outcome of the string similarity threshold. When using a longer feature vector, the string similarity threshold can be lower and produce similar precision.

In Figure 5.7, we can see the combined effects of recall and precision. As the harmonic mean of recall and precision, the F1-score exhibits characteristics of both metrics: the apparent drop-off at the higher values of the parameter and the evident increase after a specific cut-off value of 0.6.

Figure 5.8 shows the memory usage as a function of the string similarity threshold. We find a negative trend of $\alpha \approx -852.94$. The nature of the string similarity explains this phenomenon: a higher similarity threshold filters out more potential pairs, and the final result is a relatively smaller data frame.

To summarise, the effects of the string similarity threshold on the performance metrics are apparent. At values around 0.6, the parameter's value starts positively affecting the algorithm's outcome. Similarly, at 0.82 it starts having a negative impact. When choosing the parameter value, we decide between filtering out true positives and allowing false positives. Additionally, higher values lead to less memory usage.

Random bucket projection distance threshold

This section examines the impact of the RBP distance threshold on the performance metrics. Figure 5.9 illustrates the effect of the RBP distance

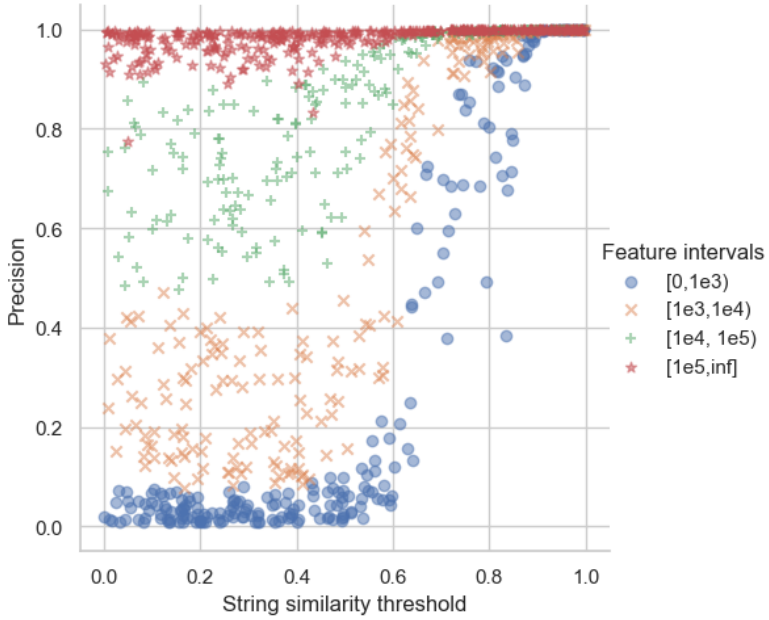


Figure 5.6. The precision of the TF-IDF-based algorithm as a function of the string similarity threshold divided into intervals by the feature vector length.

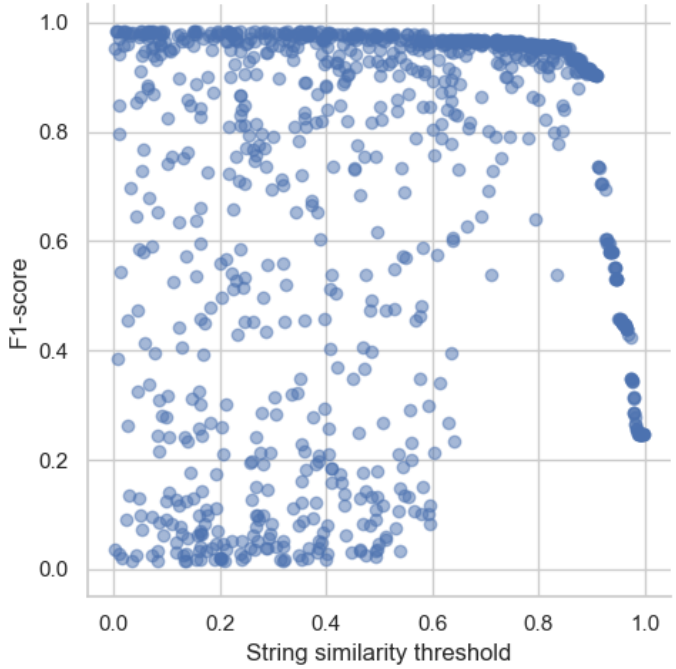


Figure 5.7. The F1-score of the TF-IDF-based algorithm as a function of the string similarity threshold.

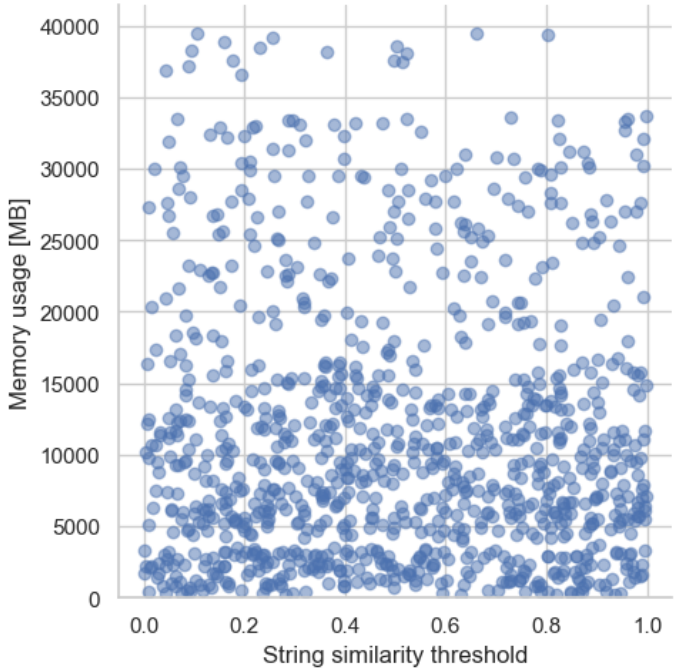


Figure 5.8. The memory usage of the final data frame of the TF-IDF-based pipeline as a function of the string similarity threshold.

threshold on the F1-score. The data expresses a negative $\alpha \approx -0.005$ correlation. Compared to the other two parameters, the RBP distance threshold does not impact the F1-score significantly.

We examine the relationship between the distance threshold and the memory usage using Figure 5.10. We can see a positive $\alpha \approx 175.13$ trend between the distance threshold and the memory usage of the final data frame. Since more pairs are allowed through the pipeline, additional memory is required to store the data.

The effects of the distance threshold parameter are minor. The method of random bucket projection and filtering by threshold is intended to bring up the precision at lower parameter values and decrease the computational load at a potential cost of the recall. The results do not reflect this intention. However, the author notes that unless we use a low distance threshold, the memory required during the pipeline evaluation causes additional memory requirements and can produce errors and crashes. Due to the setup, we cannot capture the memory usage of the intermediate objects, but the mentioned behaviour has been experienced. Thus, the author recommends using a low threshold in practice.

Summary

Figure 5.11 highlights the combined effect of the three variables on the F1-score. As previously discussed, we can see that the feature vector length

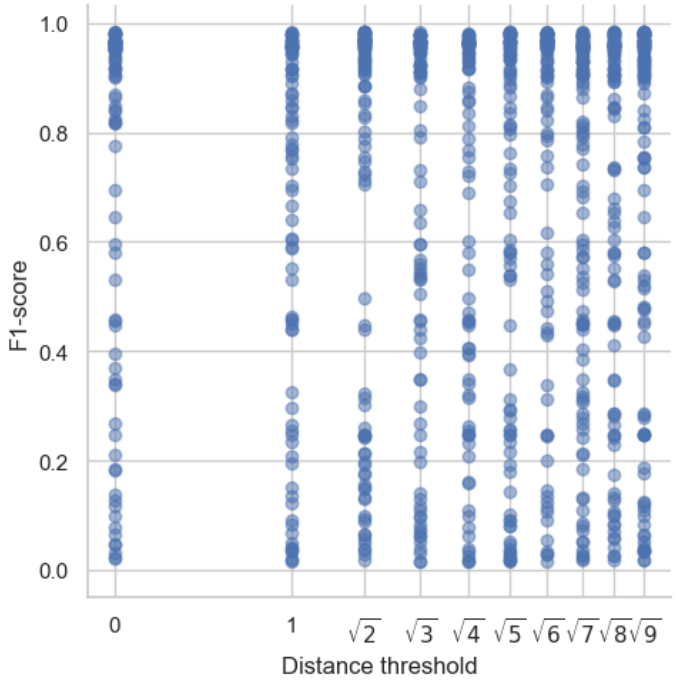


Figure 5.9. The F1-score of the TF-IDF-based algorithm as a function of the RBP distance threshold.

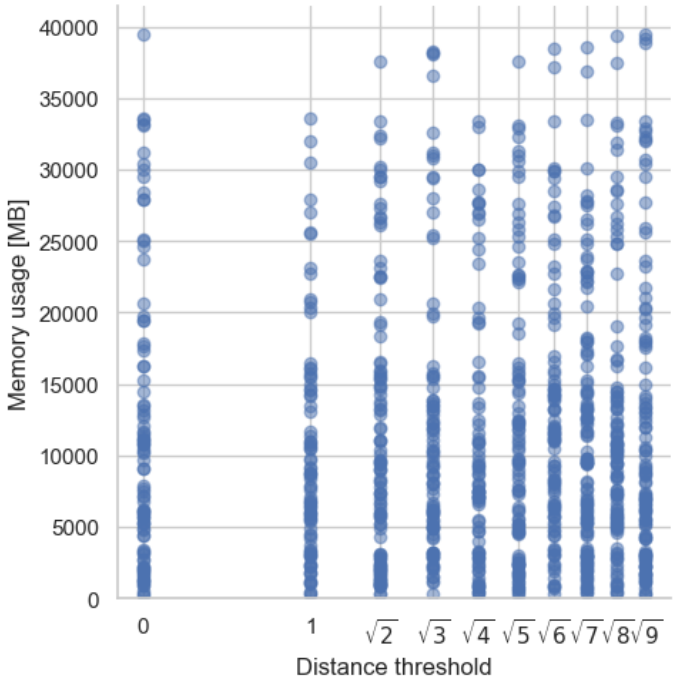


Figure 5.10. The memory usage of the final data frame of the TF-IDF-based pipeline as a function of the RBP distance threshold.

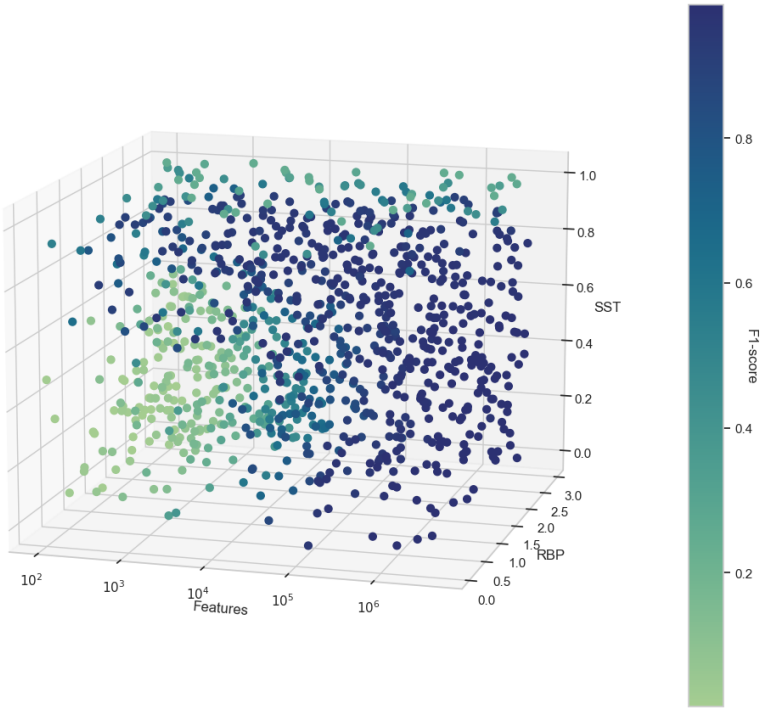


Figure 5.11. A scatter plot highlighting the combined effect of the variables of the TF-IDF-based model on the F1-score. "Features" stand for feature vector length, "SST" for string similarity threshold and "RBP" for random bucket projection distance threshold. The feature vector length is on a logarithmic scale.

Table 5.2. The optimal range of values for the parameters of the TF-IDF-based pipeline.

Parameter	Optimal range
Feature vector length	$[10^5, 10^{6.69}]$
String similarity threshold	$[0.84, 0.92)$
RBP distance threshold	$\{0\}$

and the string similarity threshold impact the F1-score significantly compared to the RBP distance threshold. By choosing the right combination of string similarity threshold and feature vector length, we can achieve high F1-scores.

Memory usage mostly stays the same based on the variables' values. However, since the reported measure is the final data frame memory usage, it does not have the required memory during the pipeline. With increasing RBP distance thresholds, we encounter crashes due to memory errors. The author recommends using values in the ranges indicated in Table 5.2.

5.3.2 Word2Vec based pipeline

This section highlights the parameters' effect on the Word2Vec-based embedding pipeline. The results will be measured using the five performance metrics: relative time, F1-score, precision, recall and memory usage of the final data frame. All plots related to the parameters of the Word2Vec

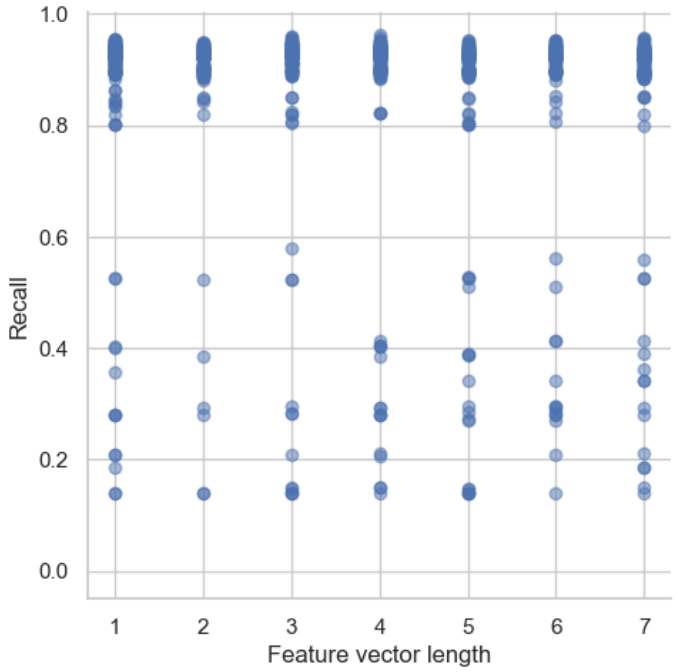


Figure 5.12. The recall of the Word2Vec-based algorithm as a function of the feature vector's length.

pipeline can be seen in Appendix B.2 additionally to those presented in this section.

Feature vector length

We can see the recall as a function of the feature vector length in Figure 5.12. Generally, a high recall is present at all levels, with some outliers below the $\mathrm{Rc} = 0.6$ line. Using the string similarity threshold as a second variable, we can see that the outliers result from a very high string similarity threshold. We can observe the string similarity threshold as a second explanatory variable in Figure 5.13. The feature vector length does thus not have a significant impact.

In Figure 5.14, we can see the precision as a function of the feature vector length. The plot shows us clusters at the high and low end of the y-axis, with some points between the clusters. If we examine Figure 5.15, we can see that, similarly to the recall above, it is a single interval of the string similarity threshold that is the source of the data points between the clusters. Additionally, those belonging to the two categories, with a string similarity threshold of less than 0.84, are clustered at the bottom, with a few outliers at the top.

Figure 5.16 illustrates the F1-score as a function of the feature vector length. As discussed earlier, due to the nature of the F1-score, we can see both the precisions and the recalls behaviour in the F1-score. No

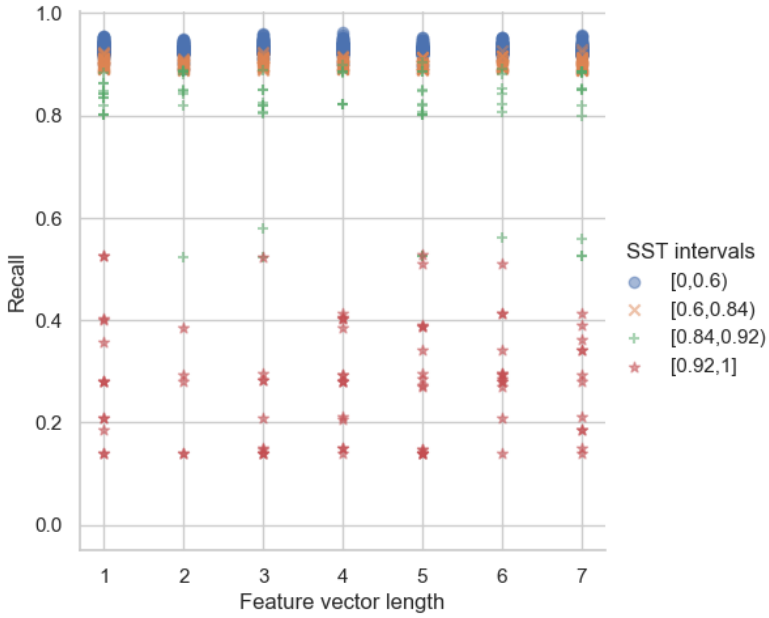


Figure 5.13. The recall of the Word2Vec-based algorithm as a function of the feature vector's length divided into string similarity threshold intervals.

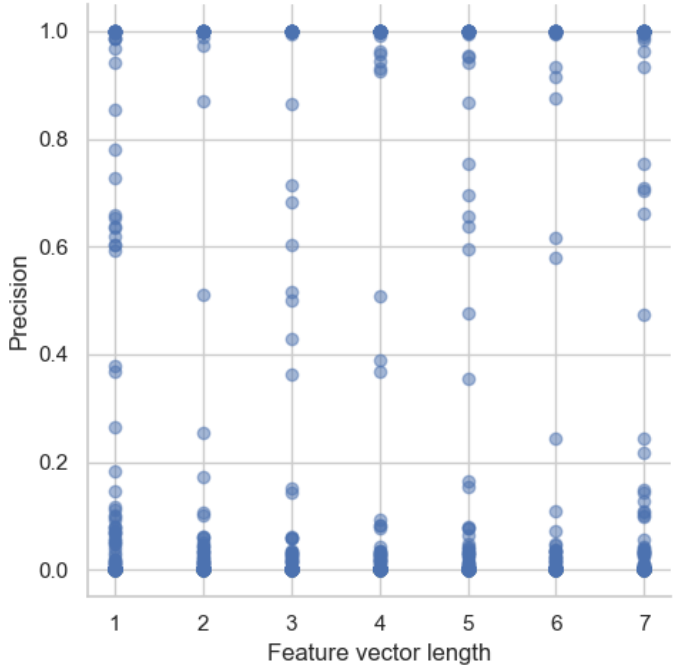


Figure 5.14. The Word2Vec-based algorithm's precision as a function of the feature vector's length.

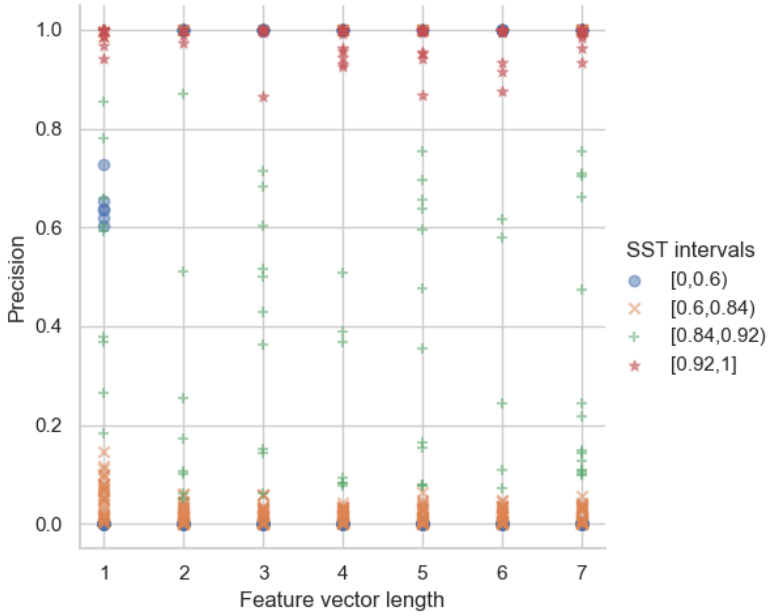


Figure 5.15. The Word2Vec-based algorithm's precision as a function of the feature vector's length divided into categories based on the string similarity threshold.

correlation exists between a higher feature vector length and a higher F1-score. We can confirm this by fitting a line to the data and seeing that $|\alpha| \leq 0.001$, where α is the slope. Figure 5.17 shows us how the RBP distance affects the F1-score as a secondary explanatory variable. Only when the distance is zero can the algorithm achieve an F1-score of more than 0.83.

In Figure 5.18, we can see how the feature vector length impacts the memory usage of the final data frame. We find no significant differences between the lengths and only a slope of $\alpha = 48.261$ for the fitted line. This slope is insignificant in the scale of the y-axis.

Based on the figures in this section, we conclude that the feature vector length does not significantly impact performance metrics. In combination with the string similarity threshold, we can obtain decent values. However, as we present later, we can obtain excellent F1-scores with any examined feature vector length above two as long as the RBP distance threshold is low enough.

String similarity threshold

In this section, we examine the effects of the string similarity threshold on the performance metrics. We begin by examining the recall, which we can see in Figure 5.19. We see a nearly identical curve as for the TF-IDF version, which we display in Figure 5.4 — a similar small drop just prior to 0.6 and a sharp downward slope after 0.83.

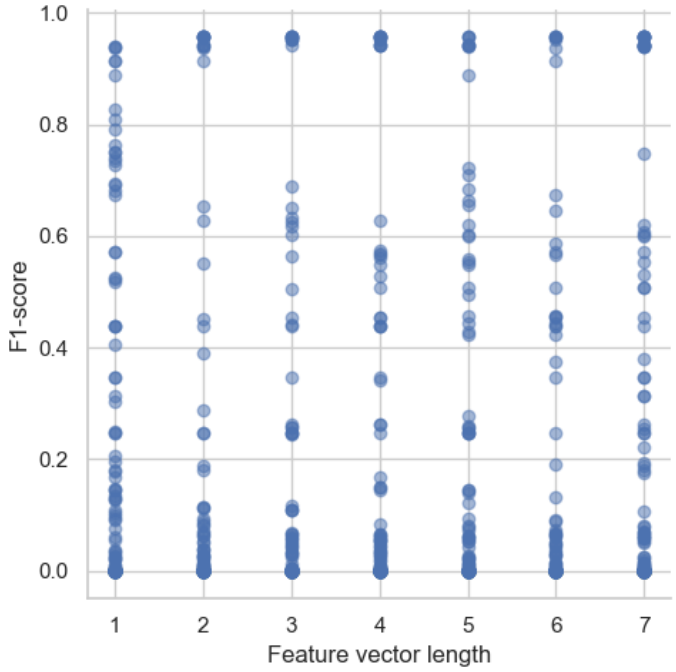


Figure 5.16. The F1-score of the Word2Vec-based algorithm as a function of the feature vector's length.

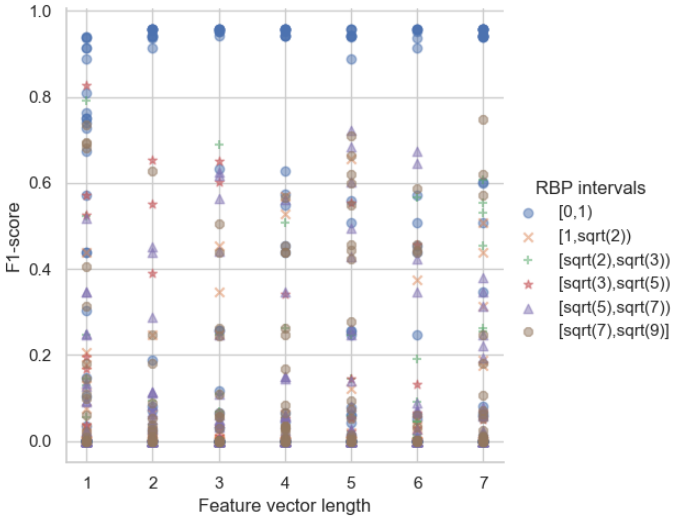


Figure 5.17. The F1-score of the Word2Vec-based algorithm as a function of the feature vector's length divided into RBP distance intervals.

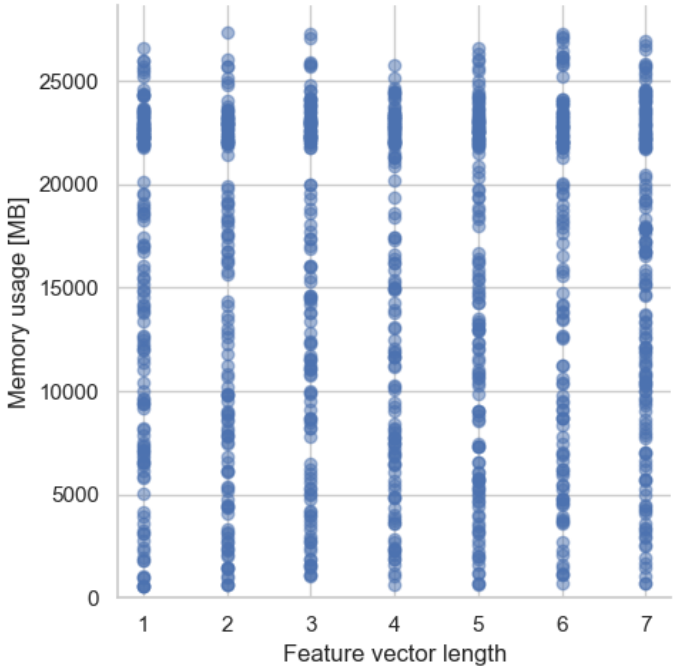


Figure 5.18. The memory usage of the final data frame of the Word2Vec-based pipeline as a function of the feature vector's length.

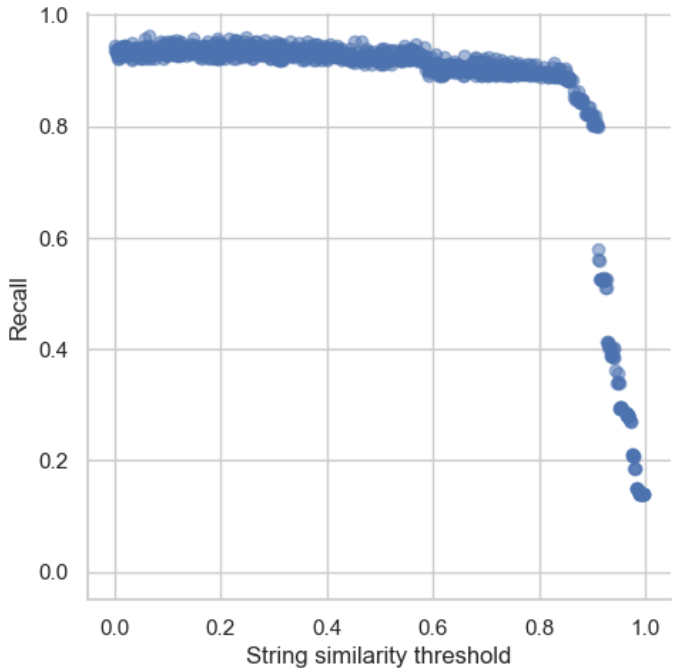


Figure 5.19. The recall of the Word2Vec-based algorithm as a function of the string similarity threshold.

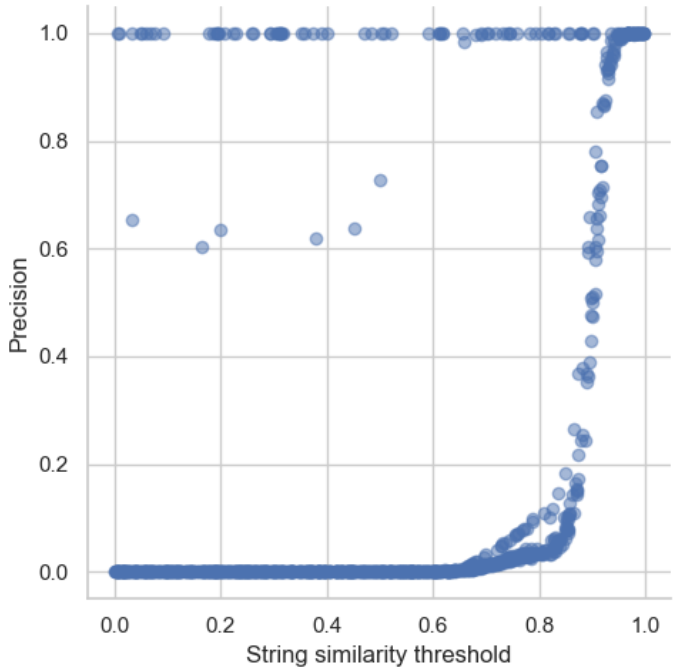


Figure 5.20. The precision of the Word2Vec-based algorithm as a function of the string similarity threshold.

The precision, which we can see in Figure 5.20, displays unexpected behaviour. For the worst-case scenario precision, we can see a very low precision for the string similarity threshold until approximately 0.7, at which point it rises sharply. However, simultaneously, we can see points with very high precision across the x-axis. By examining Figure 5.21, which shows us the RBP distance as a second explanatory variable, we can see that an RBP distance threshold of zero gives a high precision. The figure also shows a few outlier points that behave differently. We find no explanation for these.

The F1-score, seen in Figure 5.22, displays the same features in the precision and recall. Additionally, Figure 5.23 shows us the effect of the RBP distance value on the precision: An RBP value of zero behaves differently from the other values. Thus, when choosing string similarity value, we must know the RBP value to make a correct evaluation. Additionally, choosing values over 0.9 impacts the F1-score negatively in all scenarios.

In conclusion, the results presented in this section indicate that the string similarity threshold is not an essential factor in the outcome of the pipeline since having the correct RBP distance threshold renders the effect of the string similarity threshold insignificant. This holds as long as the parameter is below 0.9. However, examining all three parameter values shows that setting a string similarity value is recommended.

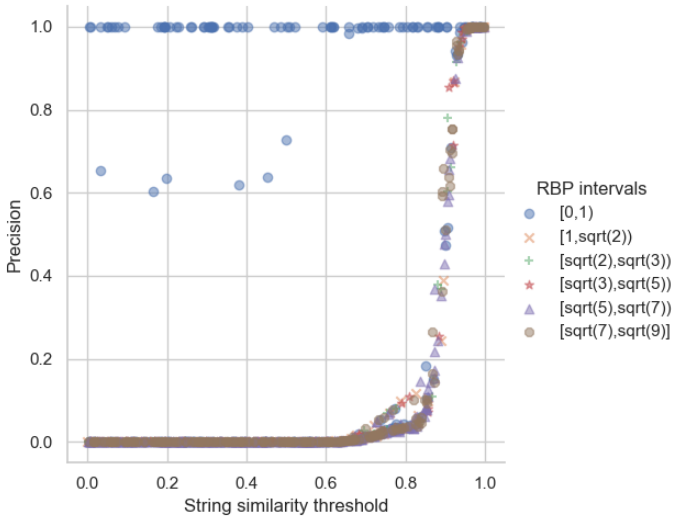


Figure 5.21. The precision of the Word2Vec-based algorithm as a function of the string similarity threshold divided into categories based on the RBP distance value.

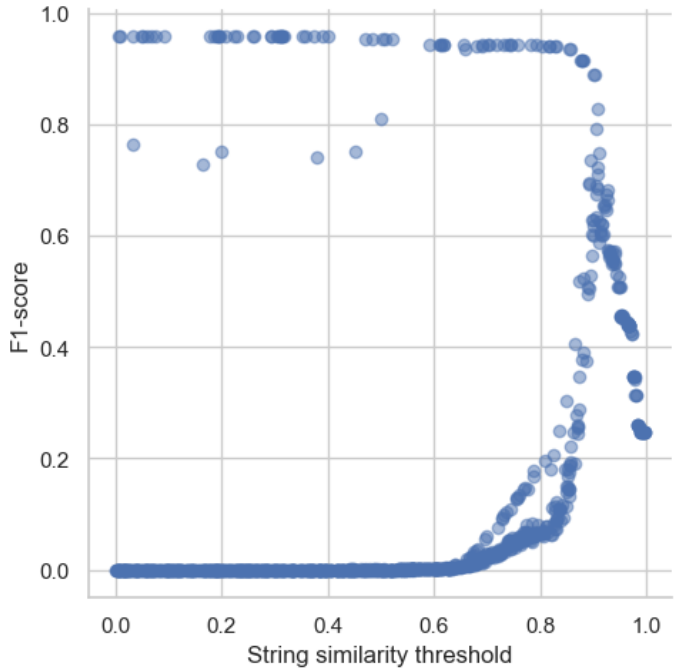


Figure 5.22. The F1-score of the Word2Vec-based algorithm as a function of the string similarity threshold.

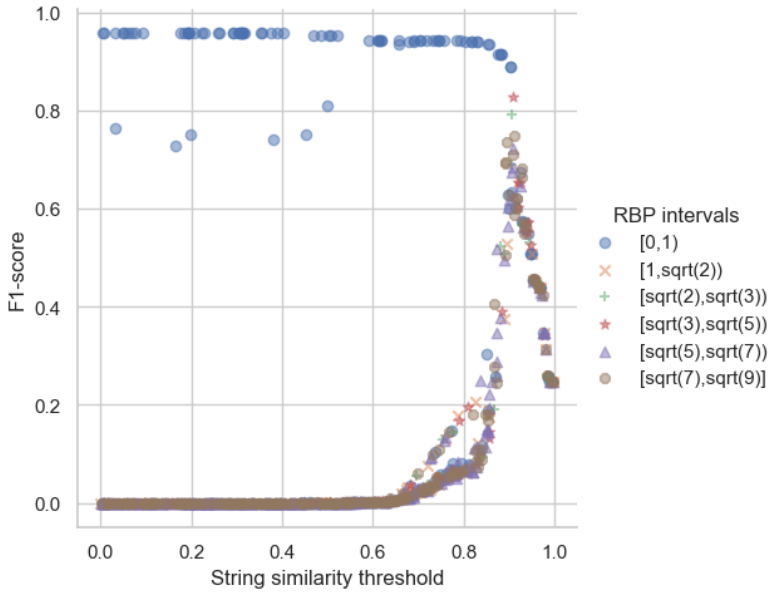


Figure 5.23. The F1-score of the Word2Vec-based algorithm as a function of the string similarity threshold divided into categories based on the RBP distance value.

Distance threshold

In this section, we investigate the impact of the RBP distance on the performance metrics. We can see the relative time as a function of the distance in Figure 5.24. We can observe a slight positive linear trend of $\alpha \approx 0.158$. However, the magnitude of this trend is minor compared to the scale of the relative time and the range of the RBP distance threshold.

The distance threshold significantly impacts the precision, visible in Figure 5.25. We can see that at a distance equal to zero, the precision acquires only values above 0.6 and mostly values close to 1. A negative $\alpha \approx -0.202$ correlation exists between the distance threshold and the precision. In Figure 5.26, we can see that the lower values at a distance equal to 0 result from a low feature vector length. Similarly, Figure 5.27 shows that the lower precision values at distance zero all have lower string similarity threshold, below 0.6. Similarly, at the other RBP distance threshold values, only string similarities in the highest interval lead to high precision values.

As previously concluded, the F1-score combines the behaviours of precision and recall. Figure 5.28 summarises the effect of the distance threshold well as a distance threshold of zero provides the best F1-score.

The impact of the RBP distance threshold is significant. Using an RBP distance threshold of zero significantly improves the pipeline results. Additionally, combined with a low string similarity threshold, we achieve the best F1-scores.

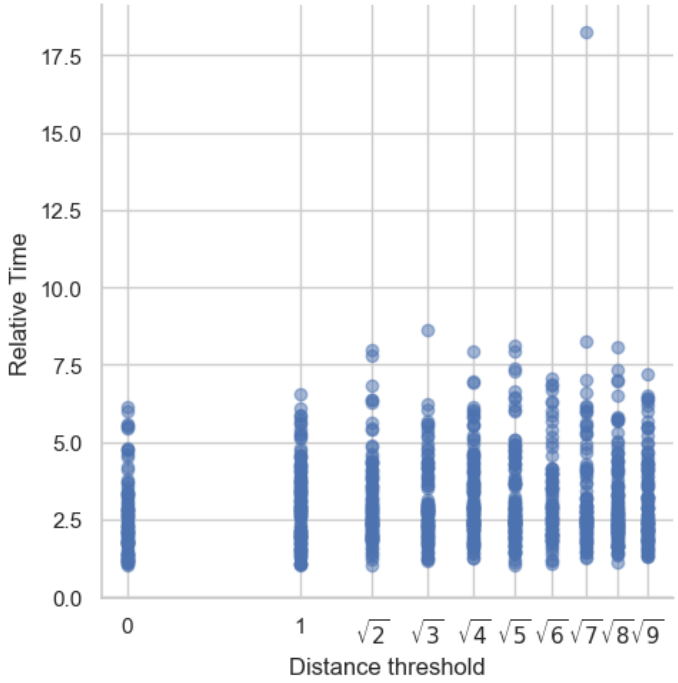


Figure 5.24. The relative time needed to run the Word2Vec-based algorithm as a function of the random bucket projection distance threshold.

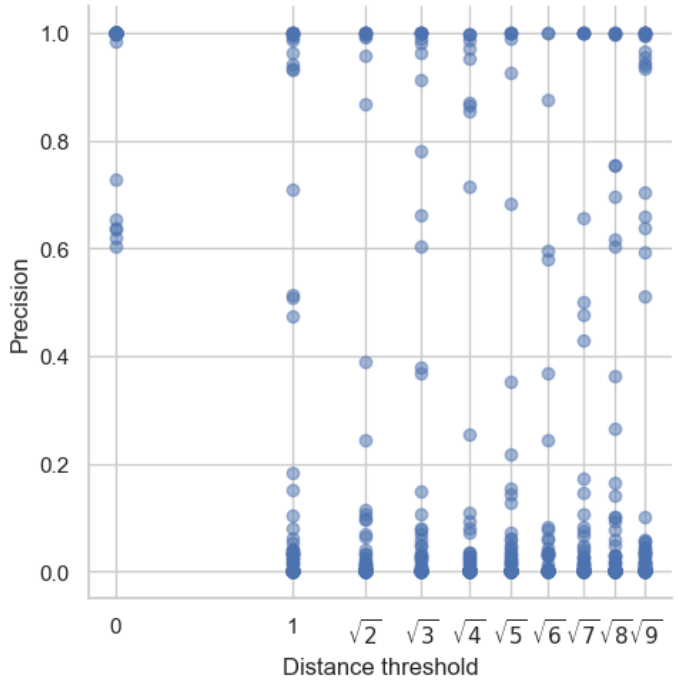


Figure 5.25. The precision of the Word2Vec-based algorithm as a function of the random bucket projection distance threshold.

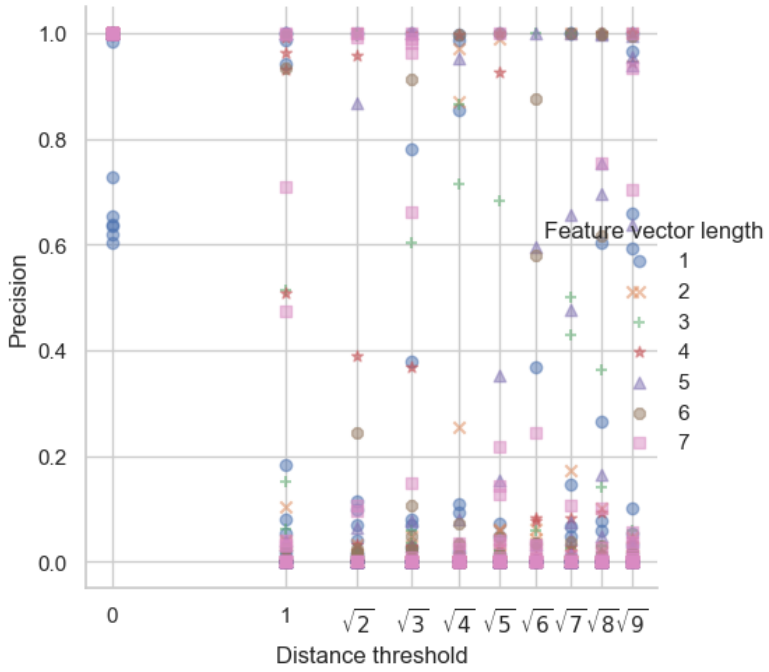


Figure 5.26. The precision of the Word2Vec-based algorithm as a function of the random bucket projection distance threshold categorised based on the feature vector length.

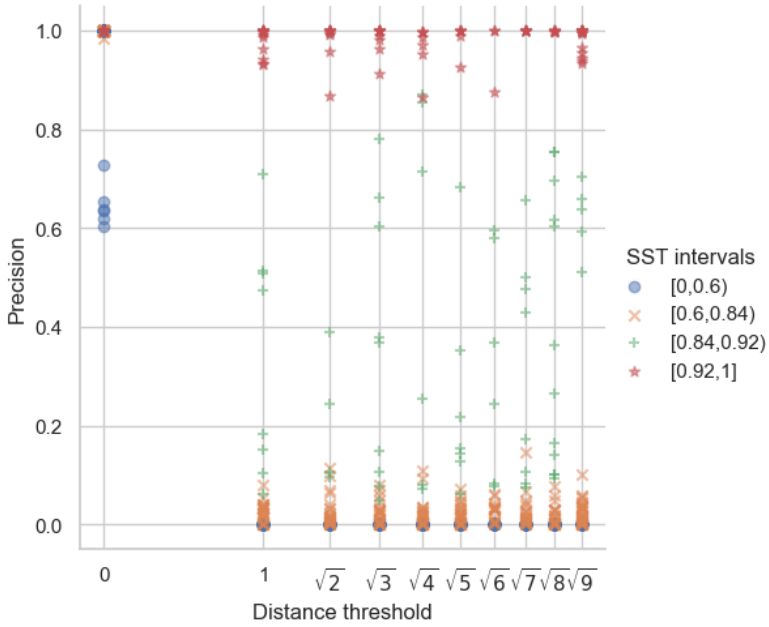


Figure 5.27. The precision of the Word2Vec-based algorithm as a function of the random bucket projection distance threshold categorised based on the feature vector length.

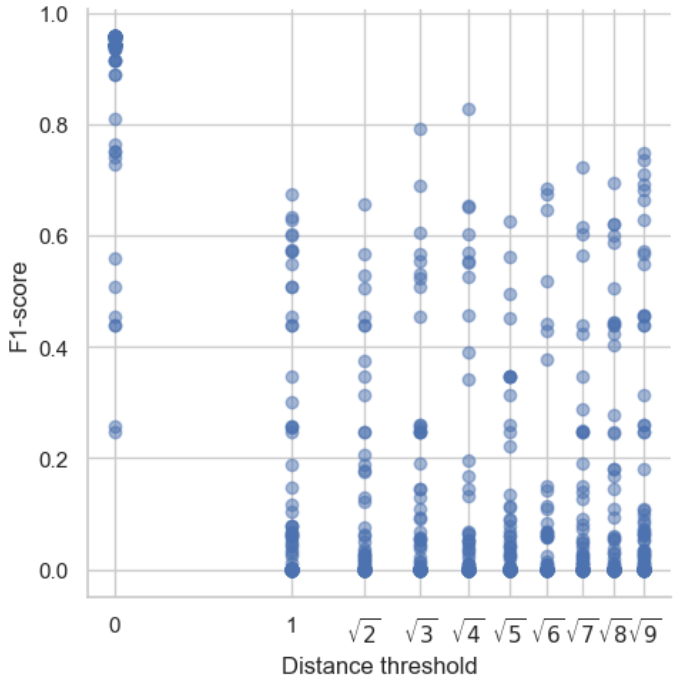


Figure 5.28. The F1-score of the Word2Vec-based algorithm as a function of the random bucket projection distance threshold.

Table 5.3. The optimal range of values for the parameters of the Word2Vec-based pipeline.

Parameter	Optimal range
Feature vector length	$[2, 7]$
String similarity threshold	$[0.6, 0.84)$
RBP distance threshold	$\{0\}$

Summary

Figure 5.29 summarises the effects of the three variables on the F1-score. As discussed earlier, the RBP distance threshold is the most influential of the three variables. The key takeaway is setting the RBP distance to 0, keeping the string similarity threshold below 0.84 and having a feature vector length of more than two. The author recommends using values in the intervals seen in Table 5.3

5.3.3 Comparing embedding method

This section compares the TF-IDF-based embedding algorithm to the Word2Vec-based embedding algorithm. One of the main problems set out to solve is the methods' feasibility and performance. Thus, we want to compare the models to find differences in performance and whether one is superior. To determine differences and superiority, we plot the results of the two models and compare them, using all iterations and only the iterations using the optimal interval of the parameters.

Figure 5.30 presents the violin plot of the relative time needed for each

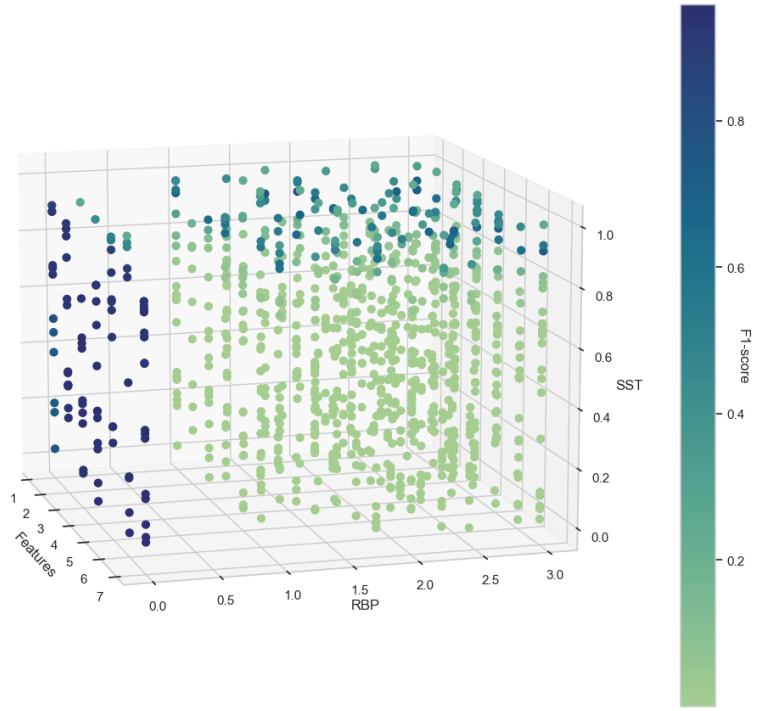


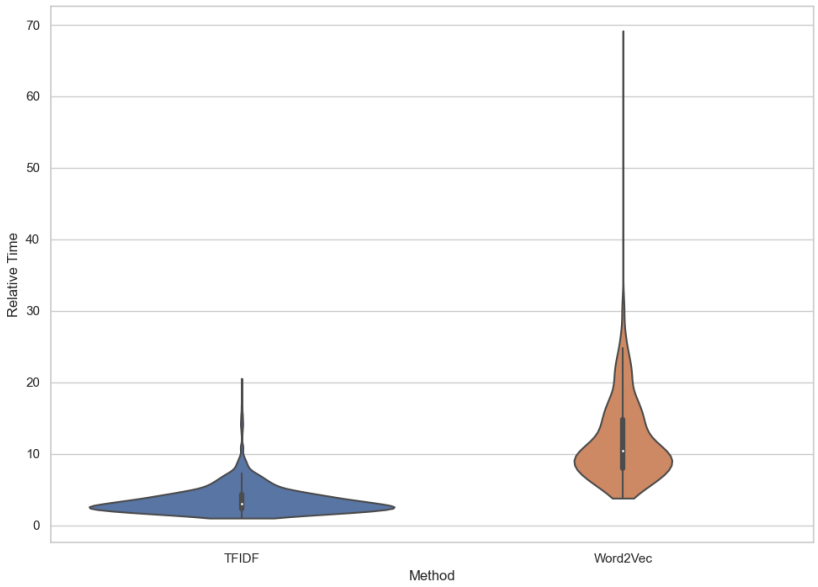
Figure 5.29. A scatter plot highlighting the combined effect of the variables of the Word2Vec-based model on the F1-score. "Features" stand for feature vector length, "SST" for string similarity threshold and "RBP" for random bucket projection distance threshold. The feature vector length is on a logarithmic scale.

model. Note that the relative time here is compared to the combined shortest time of both runs. Both Figure 5.30a and Figure 5.30b show the same result: the TF-IDF model requires less computational time.

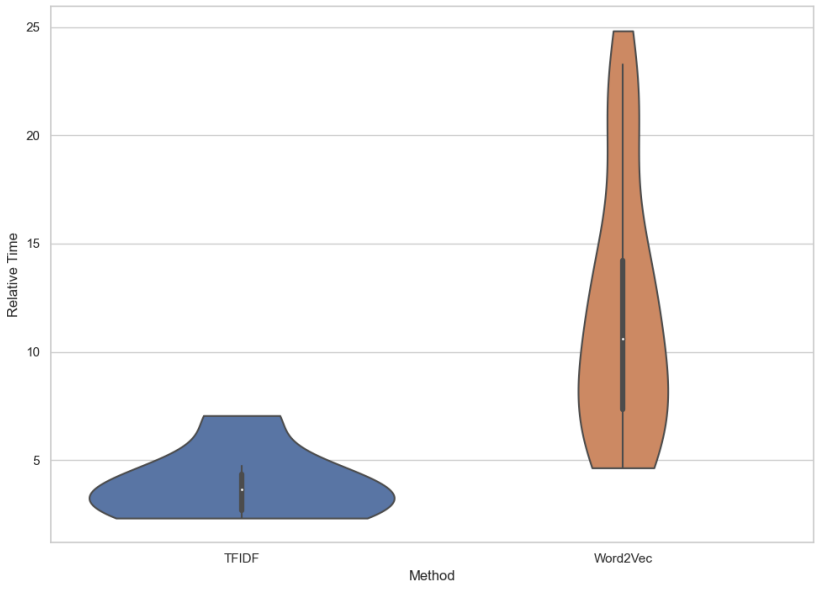
The F1-score comparison, presented in Figure 5.31. Figure 5.31a, shows that the median and largest part of the mass lays close to zero for the Word2Vec-based model while the opposite is true for the TF-IDF model. Figure 5.31b shows that our choice of the optimal values for the Word2Vec model does not give us an unambiguous result but that the choice of a parameter within the interval may still lead to bad F1-scores. Contrary, the TF-IDF model performs well or exceptionally well in the chosen interval.

Figure 5.32 presents the models' precision. Figure 5.32a shows similar behavior displayed by the F1-score. Figure 5.32b attempts to show the precision using the optimal parameter intervals. However, the TF-IDF-based results are tightly grouped around one; thus, no kernel density can be estimated. For the Word2Vec model, the contrary is true; the result cluster around zero and one. Thus, a similar phenomenon is seen. We conclude that the TF-IDF-based model outperforms the Word2Vec-based model as the precision is clustered around one in the optimal range.

Figure 5.33 illustrates the recall of the model results. The Word2Vec-based model indicates superiority in the optimal parameter interval, as

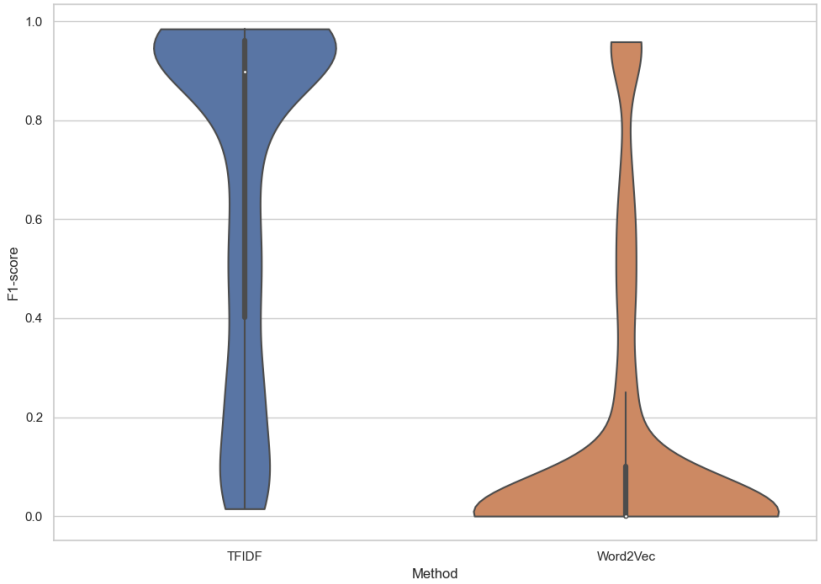


(a) Using all data.

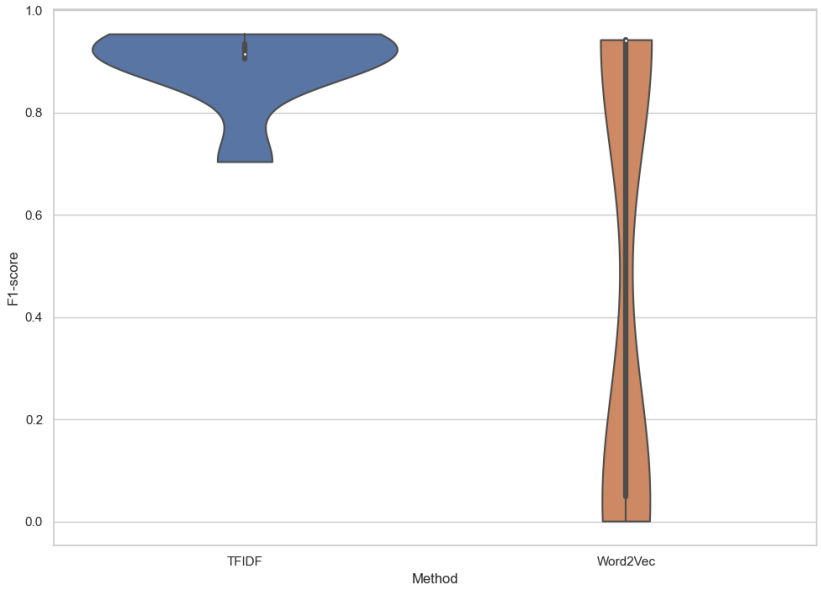


(b) Using the optimal parameter intervals seen in Tables 5.2 and 5.3.

Figure 5.30. The TF-IDF- and the Word2Vec-based algorithms plotted against relative time.

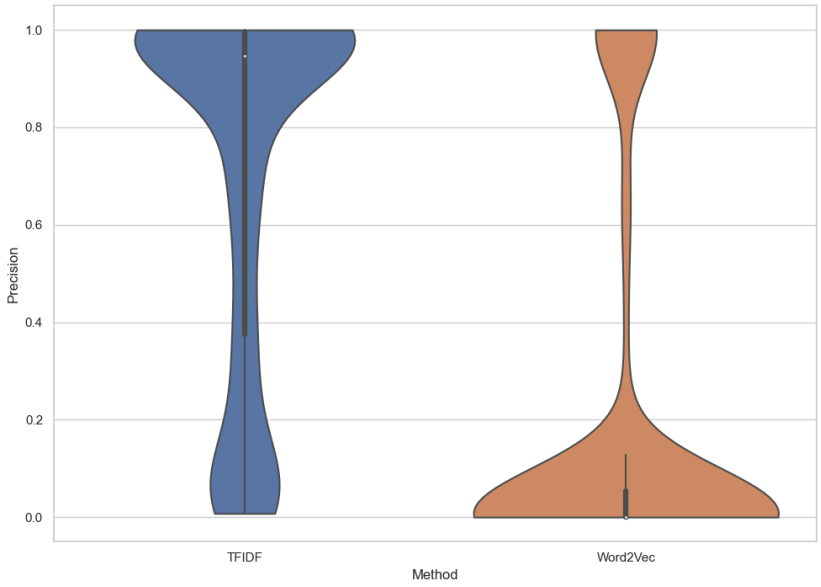


(a) Using all data.

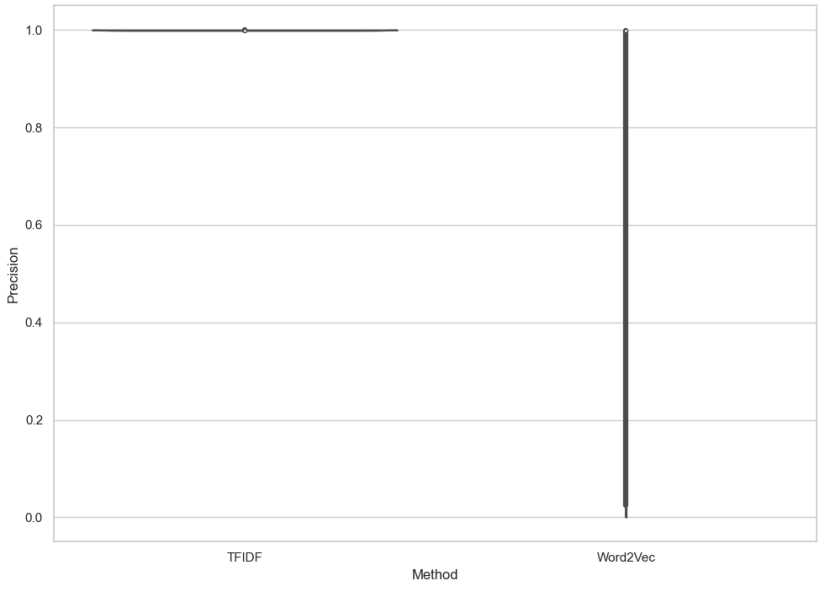


(b) Using the optimal parameter intervals seen in Tables 5.2 and 5.3.

Figure 5.31. The TF-IDF- and the Word2Vec-based algorithms plotted against f1-score.



(a) Using all data.



(b) Using the optimal parameter intervals seen in Tables 5.2 and 5.3.

Figure 5.32. The TF-IDF- and the Word2Vec-based algorithms plotted against precision. Due to the value being tightly clustered respectively evenly spread out, estimating a kernel density for the optimal parameter interval is not possible.

seen in Figure 5.33b. Meanwhile, the models behave similarly when all the iterations are used. Regarding the recall, we conclude that the Word2Vec-based model outperforms its counterpart.

Figure 5.34a shows the memory usage of the final data frame using all iterations. In the worst-case scenario, the TF-IDF-based solution uses more memory. However, the TF-IDF-based model outperforms its counterpart in the optimal range, seen in Figure 5.34b. The memory usage of the final data frame can mainly be explained by poor precision, which leads to more false positives being included in the final data frame.

Based on the above evidence, the author recommends using the TF-IDF-based solution due to the lower time requirements and higher performance across the spectrum of the variables, especially in the range of the optimal parameter interval. In the following sections, we use the TF-IDF model to evaluate the deterministic record linker's performance and compare it with the probabilistic record linker.

5.3.4 Optimal model results on the juridical person records

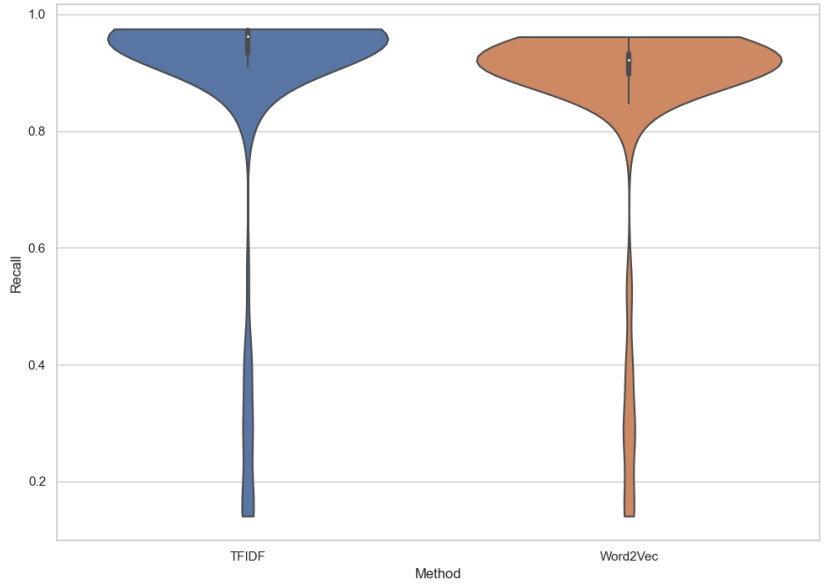
This section presents the result of the optimally tuned embedding-based approach on data of juridical person records. We choose the parameters based on the results of the previous section and seen in Table 5.4. The results can be seen in Table 5.5. We can see an exceptional performance, with an F1-score of 98.58% at approximately 10 minutes of run time.

Table 5.4. The model parameters used in the optimal embedding-based record linker for the entites

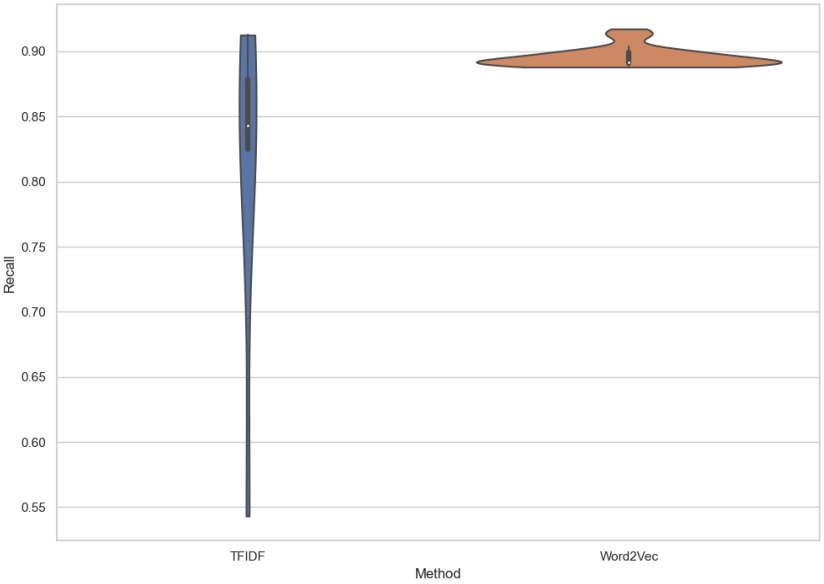
Name	Value
Model	TF-IDF
Feature vector length	$5 \cdot 10^6$
RBP distance threshold	0
String similarity threshold	0.8

Table 5.5. The deterministic record linkers performance metrics for the juridical person record linkage using the setup in Table 5.4.

Name	Value
Time	10min 4s
Precision	97.42%
Recall	99.76%
F1-score	98.58%
Memory	603MB

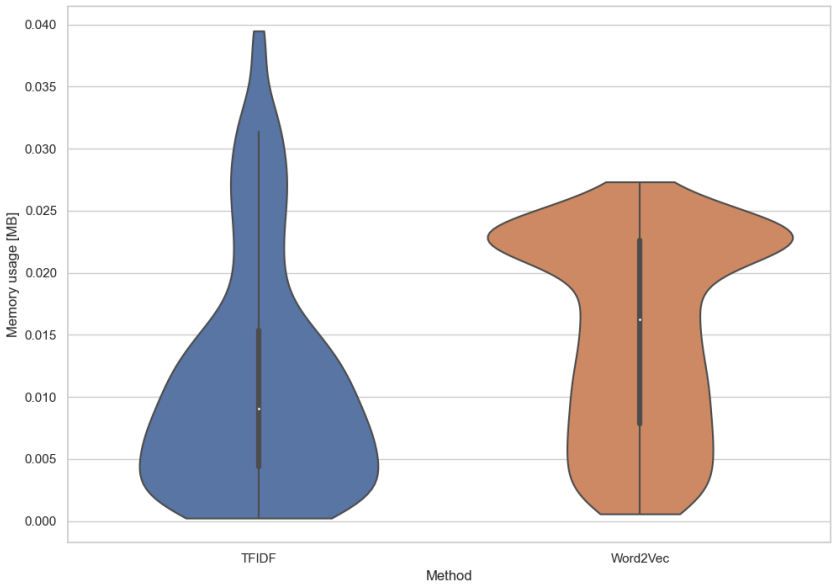


(a) Using all data.

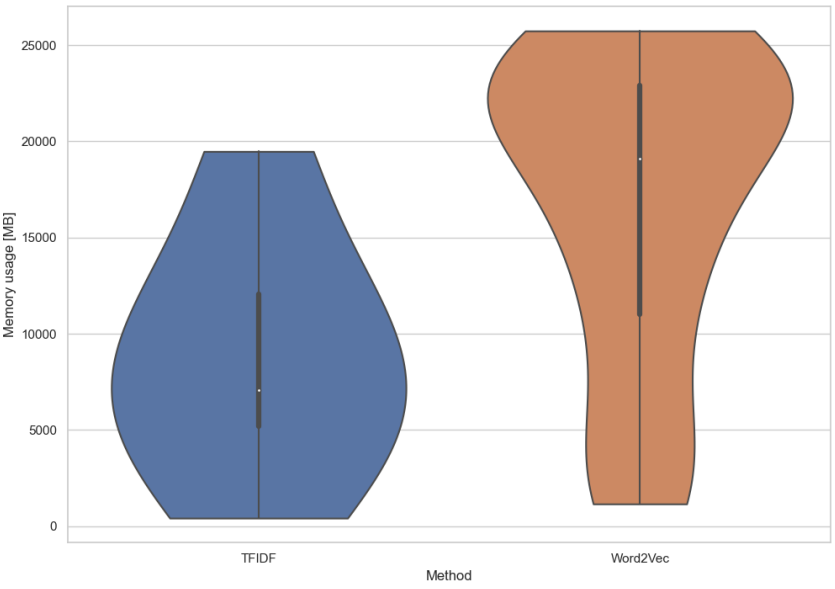


(b) Using the optimal parameter intervals seen in Tables 5.2 and 5.3.

Figure 5.33. The TF-IDF- and the Word2Vec-based algorithms plotted against recall.



(a) Using all data.



(b) Using the optimal parameter intervals seen in Tables 5.2 and 5.3.

Figure 5.34. The TF-IDF- and the Word2Vec-based algorithms plotted against memory usage of the final data frame.

5.3.5 Optimal model results on the natural person records

In this section, we present the results of the embedding-based approach on the natural person-based data. Based on the previously discussed data dimensionality, the author hypothesises that the model will perform worse than on the records of juridical persons. Names are weaker identifiers for natural persons than for juridical persons since more natural persons share names compared to juridical persons. Thus, when only including a name attribute to match on, it is highly likely that the record linkage algorithm will match non-matching persons due to them having the same names.

We are using the name of the natural persons as *Entity string*. Depending on the quality of the data source, the name may contain middle names. Based on the previous results in Section 5.3, we have chosen the model parameters seen in Table 5.6.

Table 5.6. The model parameters used in the embedding-based record linker for the natural persons.

Name	Value
Model	TF-IDF
Feature vector length	$5 \cdot 10^6$
RBP distance threshold	1
String similarity threshold	0.8

Table 5.7 presents the performance metrics. We can see that the precision, at 35.08%, is very low compared to the entity-based data results. At 82.32%, the recall is higher than the precision but also falls behind the juridical person-based results. The F1-score, as the harmonic mean of the recall and precision, reflects the previous results with a value of 49.20%.

When manually reviewing the results, the author finds that many false positives have the same name but are not the same person. The linker cannot differentiate between persons with the same name due to not having any additional information about the persons, such as date of birth or location. This conclusion is in line with the hypothesis presented previously.

Furthermore, the poor precision may stem from alphabetically ordering the words in the vector before comparing the strings. The ordering may benefit the juridical person linkage, but the outcome may hurt natural persons record linkage as first, middle and last names may be compared wrongly. In conclusion, the method does not perform well on natural person records.

Table 5.7. The embedding-based record linker performance metrics for the natural persons using the setup in Table 5.6.

Name	Value
Time	19min 43s
Precision	35.08%
Recall	82.32%
F1-score	49.20%
Memory	994MB

5.4 Probabilistic record linkage results

We present the results of the probabilistic record linkage in this section. Contrary to the deterministic record linkage, we will only explore one set of parameters for the probabilistic linker. This is because the probabilistic method, based on Splink, requires careful engineering of rules, settings and parameters. Furthermore, we will only assess the method on the data sets of natural person records. We have iteratively engineered the setup for the results we present in this section. Appendix C presents the complete setup used for the results.

We employ a set of global blocking rules to limit the number of evaluations required in the prediction phase. We present the global blocking rules and the number of new unique pairs generated in Appendix C.2. The reduction ratio, as defined by Christen and Goiser [28], we achieve with these rules is $99,93\%$ of the complete Cartesian join data set C. Figure 5.35 presents the generated pairs created by each rule. Note that if an earlier rule, i.e. higher up in the figure, includes a pair, it is not included in a later, i.e. lower, rule. Thus, the first rule contributes most of the generated pairs.

Splink calculates the prior probability of a match using a set of deterministic matching rules seen in Appendix C.4. These rules estimate the probability of two random records to match to be $6.49 \cdot 10^{-7}$.

We estimate an initial value for the u-variables using a random sample of 10^9 pairs. The local blocking rules, used for estimating the m-variables and tune u-variables, can be seen Appendix C.3. The appendix also presents the unique pairs each local blocking rule generates.

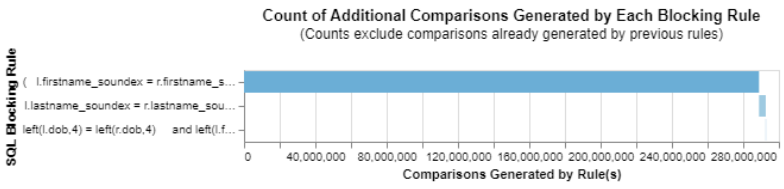


Figure 5.35. A illustration of the effect of the global blocking rules employed. The rules are in the same order as those in Figure 5.35. The plot is created using Splink [39]

Table 5.8. The probabilistic record linker performance metrics for the natural person data. The metrics are given for pairs considered a match if they have a posterior probability of 0.8.

Name	Value
Time	$2h$ 38min 26s
Precision	52.67%
Recall	88.38%
F1-score	66.00%
Memory	571MB

Figure 5.36 presents the tuned weight of each case statement and the *a priori* probablity. Furthermore, Figure 5.37 presents the occurrence percentage for each case statement based on the model's estimates of pairs being matches and non-matches. We can see that the training of the model has been a success due to the higher comparison levels, such as exact match, giving more substantial evidence and due to the large negative magnitude of the prior probability. However, we can see that some of the bars' magnitude and direction are not as expected. This might indicate that the model has diverged from the true distribution.

Firstly, the Soundex match of the first name has an extensive negative bar. Due to the large magnitude, any pair that matches this case statement is practically rendered impossible to be a highly probable match. Figure 5.37 explains the cause of the phenomena. As, according to the model, 100% of the matching pairs have an exact match on the first name, there is no evidence that a Soundex match leads to a match. Hence, the m-variable takes a value of 0 and the u-variable dominates.

Secondly, the year matching of the date of birth displays similar behaviour to the Soundex match of the first name. The model estimates a 0% occurrence of the year matching, as shown in Figure 5.37.

Finally, the two "All other comparisons" of first and middle names follow the same logic. However, these are less of a problem as those comparisons are expected to be negative. However, a well-behaving model is expected to have smaller magnitudes.

By setting the limit for matching and non-matching pairs at a posterior probability of 0.8, we can order the pairs into M and U. Table 5.8 presents the performance metrics at this limit. The F1-score is low compared to the extended run-time of the model. However, when performing a manual review of the indicated false positives, we find many true matches that are not part of our ground truth. Similarly, when reviewing the false negatives, we find that they are genuinely non-matches. Thus, the model works better than the performance metrics show.

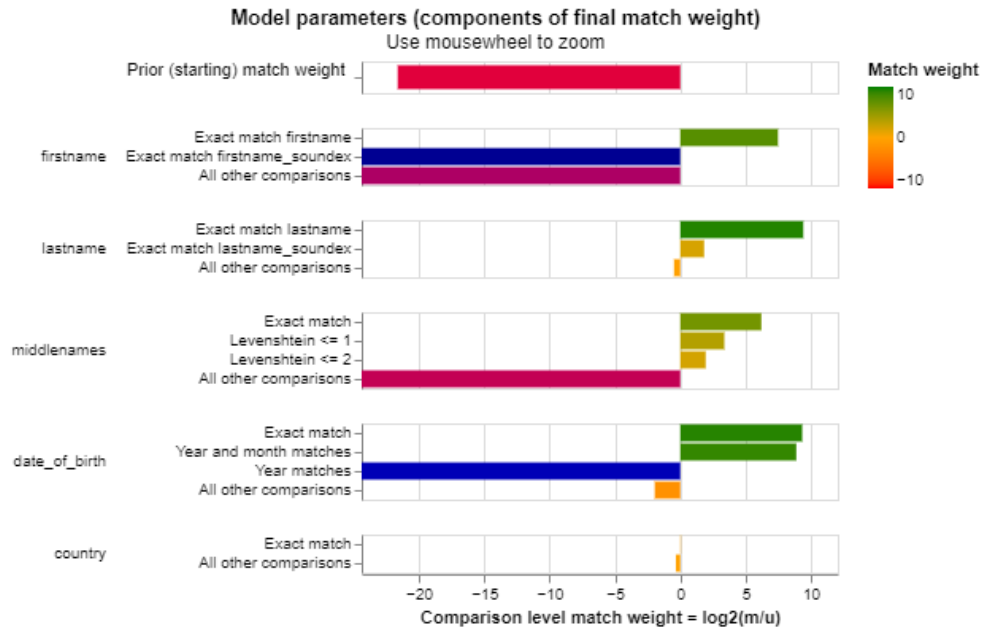


Figure 5.36. A illustration of the match weight, i.e. ratio, between m- and u-variables, for each attribute and case statement. The bars extending to the right side indicate positive evidence that the pair is a match, and vice-versa for the bars extending to the left. The length of the bar indicates the magnitude of the evidence. The purple bars represent a value in the range $[-30, -25]$ and the blue bars $[-65, -60]$. The plot is created using Splink [39].

While engineering the setup, rules and parameters, the author faced challenges in obtaining the desired outcome. We want the model, i.e. the variables, to reflect the underlying distribution of data and errors, and we have some idea of the distribution. During multiple iterations, we saw models that did not reflect our picture of the underlying distribution through, e.g., comparison level match probability with the wrong sign, too great magnitude or a combination of both. The effects and interactions of the method are not transparent and easily understandable. If we add or remove a rule or comparison level, we might see unexpected behaviour of the tuned variables. As discussed, even the final model has flaws; some case statements have m-variable values of 0. By manual review, we know that there are matches that fall into this comparison level.

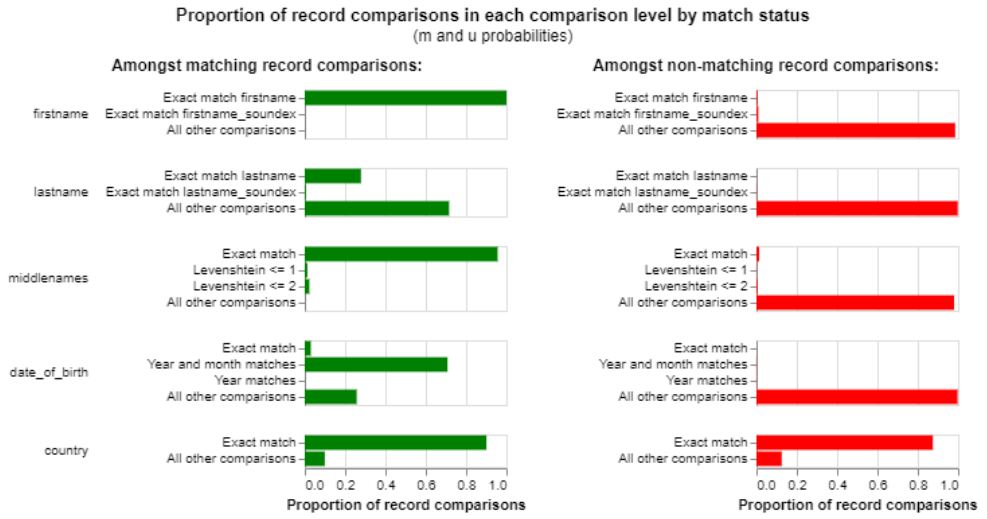


Figure 5.37. A illustration of the occurrence percentage for each attribute and case statement in, by the algorithm estimated, true matches and true non-matches. The plot is created using Splink [39].

5.5 Comparison of deterministic and probabilistic methods

We start by comparing the results of the record linkage of the data set of natural person records. We recall that the deterministic method achieved an F1-score of 49.20% in approximately 20 minutes and that the complete results are visible in Table 5.7. Similarly, the probabilistic method achieved an F1-score of 66.00% in approximately 2.6 hours. Thus we can see that the probabilistic method outperforms the deterministic method on all metrics except for time. However, as we saw previously, the probabilistic method performs better than the metrics indicated due to how we create the ground truth.

Secondly, we review the results of the record linkage of the juridical data set. As discussed previously, we did not perform a record linkage using the probabilistic method for this data due to the nature of the method and the resource requirements. Additionally, the deterministic method has outstanding results on the data with an F1-score of 98.58%. Thus, the deterministic record linker is well suited for linking records of juridical persons using only the name.

There are two prominent factors to consider when deciding which model to use for a record linkage problem. Firstly, one must consider the data. The set of pairs $\mathbf{C}$ created by the Cartesian join, and the number of overlapping attributes should be considered. We recommend using the probabilistic record linker when the number of overlapping attributes is large. The rule-based blocking possibilities must be assessed if $\mathbf{C}$ is large. If we cannot split the data into blocks and get a high reduction ratio, the deterministic

Table 5.9. A presentation of the advantages and disadvantages of the two models based on the findings of this thesis

	Advantagdes	**Disadvantages**
Probablistic method	Can model advanced underlying distributions in the data.	Requires more than one attribute to function.
	Can simultaneously implement multiple string similarity or comparison algorithms and train them separately.	Interactions and causalities can be hard to grasp for the user.
	Provides clear visualisations and insights.	Requires significant overlap of records to be tunable.
		Is computationally more demanding.
		Requires significant engineering and knowledge of the data to setup
Deterministic method	Uses a single attribute to perform the linkage.	Uses a single attribute to perform the linkage.
	Is computationally less demanding.	Suffers from poor precision unless the attribute is unique.
	Requires little engineering to set up.	

record is expected to perform better.

Secondly, consider the available time, personnel resources and know-how of the personnel. Suppose the combination of these three does not provide a sufficient level of knowledgeable working hours and algorithmic run-time. In that case, one should consider using the deterministic record linker as it requires fewer human resources and less computational time to set up and use.

We can conclude that both models are feasible and have their application areas. A summary of the advantages and disadvantages can be seen in Table 5.9. The author urges the potential user to consider the previously mentioned factors and consider and weigh them based on the wanted outcome. Table 5.10 visualises the recommendations in a simple table.

Table 5.10. The recommendation of the method to use based on the findings in the thesis. Attributes refer to the number of attributes usable for linkage. Resources refer to the human and computational resources available to solve the problem.

Attributes \ **Resources**	**Low availability**	**High availability**
Large amount	Requires consideration	Probabilistic
Low amount	Deterministic	Probabilistic

6. Conclusions

From a financial crime prevention perspective, having as much information as possible on the customers is essential. Record linkage is central because it allows the bank to combine internal and external data sources without a shared unique identifier. The record linkage must be feasible in the context of the restrictive and secure environment that the bank IT infrastructure offers. These requirements set certain restraints on the possibilities of methods, as they have to be offered by and usable in the bank's existing environment.

This thesis studied the usage of record linkage in the previously mentioned environment. We presented the theoretical frameworks regarding deterministic and probabilistic record linkage and related key concepts, such as embedding, blocking and string similarity algorithms. Furthermore, we studied and presented the impact of different factors on the outcome of the model and the outcome of the chosen optimal models on the data.

From previous works [11] and the results, we conclude that the two methods excel at linking two different types of records. The deterministic record linker can use minimal data, a single attribute, to perform linkage. Additionally, the deterministic linker requires minimal setting tweaking and engineering for the user to set up. However, when a single attribute is not enough to link the records uniquely, the linker suffers from low precision.

We found that the TF-IDF-based pipeline outperforms the Word2Vec-based pipeline. This result is based on the consistency, the lower time requirement, and the superior F1-score of the result of the TF-IDF-based solutions. We found that the most impactful parameter for the TF-IDF-based pipeline was the feature vector length, and for the Word2Vec-based pipeline, the random bucket projection distance threshold. Furthermore,

the thesis recommended optimal interval ranges for each model.

We achieved an F1-score of 98.58% on the linkage task of juridical person records in a wall-clock time of 10min 4s. The pipeline used the TF-IDF-based string embedding model, with a feature vector length of $5 \cdot 10^6$, Jaro-Winkler similarity threshold of 0.8, and random bucket projection distance threshold of 0. The same method achieved an F1-score of 49.20% on the linkage task of natural person records in a time of 19min 43s using TF-IDF-based string embedding model, with feature vector length of $5 \cdot 10^6$, Jaro-Winkler similarity threshold of 0.8, and random bucket projection distance threshold of 1.

We constructed and presented a setup for the probabilistic record linker, including settings, rules, and case statements. We found that using the constructed setup, we achieved an F1-score of 66.00% in approximately 2.6 hours. Furthermore, we concluded that the performance metrics based on the ground truth may provide pessimistic results due to how the ground truth is created.

Finally, we compared the deterministic and probabilistic methods to each other. We found that the probabilistic method only functions in a setting where more than one attribute is usable for the record linker. Furthermore, the performance increases and the linker becomes easier to use when more attributes are available. Being able to define multiple levels of matching, ranging from exact to partial matches, enables the user to create a rich linkage environment. However, the probabilistic method requires well engineered settings and rules to function properly an provide desired results.

Inversely, the deterministic method excels at linking records that have a unique attribute used for linking. For example, a juridical person's name is a suitable attribute for linkage with the deterministic method. Additionally, minimal engineering is required from the user to use the deterministic record linker.

Based on the findings, the thesis recommended using the methods for scenarios described as suitable above. Additionally, we recommended considering the human and computational resources when deciding which method to use.

6.1 Future research

This section recommends areas of future research. First, we will discuss alternative ways to use the two models formulated in this thesis. Secondly, we propose changes to the architecture of the presented models. Finally, we address models and methods beyond the scope of this thesis.

We found that the deterministic record linker produces poor precision when working with the data of natural person records. The deterministic model could be used as a pre-evaluated global blocking rule for the probabilistic approach. This could be done by running the deterministic record linker with such parameter values that it maximises recall at the expense of precision. By creating an additional attribute for the pairs, we could improve the recall of the global blocking rules of the probabilistic method.

To improve the performance of the probabilistic record linker, we propose to investigate the limit of the probabilistic record linker used to determine which pairs are matches and non-matches. We propose analysing the recall-precision curve to determine an optimal limit.

Studying how to implement and access comparison and string similarity algorithms not available in the Spark SQL engine could benefit linkage problems related to the probabilistic method. For example, being able to use Jaro similarity [33], Jaro-Winkler similarity [17], and Damerau–Levenshtein distance [43] would enable the users of the method to potentially create more flexible, suitable, and accurate linkage models. Similarly, by studying the effects of using different string similarity comparison algorithms for the deterministic record linker, we could learn how different algorithms perform on different problems.

The Word2Vec-based model we used uses n-gram words. However, as our documents, i.e. names, are short compared to general NLP tasks, there is potential to benefit from n-gram letter embedding compared to n-gram word embeddings. The Python package *fastText* [44], based on the research by Bojanowski et al. [45], implements sub-word information. Initial research should be directed on whether or not this is a feasible solution in the environment proposed in this thesis.

TStudies show that Bayesian methods are viable options for record linkage problems [46, 47, 48]. As discussed earlier, the methods must be feasible and scalable to link data sets of large magnitudes. McVeigh, Spahn, and Murray [48] proposes a scalable Bayesian method which is an algorithm of interest.

In our work, we have compared TF-IDF, a numerical statistic, to Word2Vec, a deep-learning model. Based on the paper by Barlaug and Gulla [49], we can see that other models for string-embedding exist, and a neural network can replace steps of the record linkage process. Future research could be directed at finding neural networks that provide improved results on string embedding. The paper also proposes potential in an end-to-end approach using a deep neural network, which had not been seen at the time of the paper's writing. However, if future research finds a suitable method, it could provide an excellent solution to record linkage problems in the restrictive bank environment.

However, most neural network methods are supervised and require large amounts of labelled data. An unsupervised method, similar to Splink, is found in ZeroER [50]. The paper shows that the model achieves comparable performance to supervised approaches using novel techniques, such as a generative model that learns the data distribution of the matches and non-matches.

Bibliography

[1] European Parliament and Council. *Directive (EU) 2015/849 of the European Parliament and of the Council*. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015L0849&from=EN. May 2015.

[2] European Parliament and Council. *Directive (EU) 2018/843 of the European Parliament and of the Council*. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015L0849&from=EN. May 2018.

[3] Douglas P Jutte, Leslie L Roos, and Marni D Brownell. "Administrative record linkage as a tool for public health research". In: *Annual Review of Public Health* 32 (2011), pp. 91–108.

[4] Walter A Rocca et al. "History of the Rochester Epidemiology Project: half a century of medical records linkage in a US population". In: *Mayo Clinic proceedings*. Vol. 87. 12. Elsevier. 2012, pp. 1202–1213.

[5] Eugene Rogot, Paul Sorlie, and Norman J Johnson. "Probabilistic methods in matching census samples to the National Death Index". In: *Journal of Chronic Diseases* 39.9 (1986), pp. 719–734.

[6] Steven Ruggles, Catherine A Fitch, and Evan Roberts. "Historical census record linkage". In: *Annual Review of Sociology* 44 (2018), pp. 19–37.

[7] Ted Enamorado, Benjamin Fifield, and Kosuke Imai. "Using a probabilistic model to assist merging of large-scale administrative records". In: *American Political Science Review* 113.2 (2019), pp. 353–371.

[8] Halbert L Dunn. "Record linkage". In: *American Journal of Public Health and the Nations Health* 36.12 (1946), pp. 1412–1416.

[9] Ivan P Fellegi and Alan B Sunter. “A theory for record linkage”. In: *Journal of the American Statistical Association* 64.328 (1969), pp. 1183–1210.

[10] Miranda Tromp et al. “Results from simulated data sets: probabilistic record linkage outperforms deterministic record linkage”. In: *Journal of Clinical Epidemiology* 64.5 (2011), pp. 565–572.

[11] Stacie B Dusetzina et al. “An overview of record linkage methods”. In: *Linking Data for Health Services Research: A Framework and Instructional Guide [Internet]* (2014).

[12] *Dun & Bradstreet*. URL: https://www.dnb.com/.

[13] *Dow Jones Watch List Feeds*. URL: https://developer.dowjones.com/site/docs/risk_and_compliance_feeds/watchlist_ame_soc/dow_jones_watchlist/index.gsp.

[14] Atul Kahate. *Introduction to database management systems*. Pearson Education India, 2004.

[15] Thomas M Connolly and Carolyn E Begg. *Database Solutions: A step-by-step guide to building databases*. Pearson Education, 2004.

[16] William E. Winkler. “Data Linkage”. In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 301–303. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_750. URL: https://doi.org/10.1007/978-1-4419-5906-5_750.

[17] William E Winkler. “String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage.” In: (1990).

[18] Vivienne J Zhu et al. “An empiric modification to the probabilistic record linkage algorithm using frequency-based weight scaling”. In: *Journal of the American Medical Informatics Association* 16.5 (2009), pp. 738–745.

[19] William E Winkler. *Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage*. US Bureau of the Census Washington, DC, 2000.

[20] CF Jeff Wu. “On the convergence properties of the EM algorithm”. In: *The Annals of Statistics* (1983), pp. 95–103.

[21] Diksha Khurana et al. “Natural language processing: State of the art, current trends and challenges”. In: *Multimedia Tools and Applications* 82.3 (2023), pp. 3713–3744.

[22] Hans Peter Luhn. “A statistical approach to mechanized encoding and searching of literary information”. In: *IBM Journal of Research and Development* 1.4 (1957), pp. 309–317.

[23] Karen Sparck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of Documentation* 28.1 (1972), pp. 11–21.

[24] Kilian Weinberger et al. “Feature hashing for large scale multitask learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 1113–1120.

[25] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[26] Kenneth Ward Church. “Word2Vec”. In: *Natural Language Engineering* 23.1 (2017), pp. 155–162.

[27] Rebecca C Steorts et al. “A comparison of blocking methods for record linkage”. In: *Privacy in Statistical Databases: UNESCO Chair in Data Privacy, International Conference, PSD 2014, Ibiza, Spain, September 17-19, 2014. Proceedings*. Springer. 2014, pp. 253–268.

[28] Peter Christen and Karl Goiser. “Quality and complexity measures for data linkage and deduplication”. In: *Quality Measures in Data Mining* (2007), pp. 127–151.

[29] Jingdong Wang et al. “Hashing for similarity search: A survey”. In: *arXiv preprint arXiv:1408.2927* (2014).

[30] Moses S Charikar. “Similarity estimation techniques from rounding algorithms”. In: *Proceedings of theTthiry-Fourth Annual ACM Symposium on Theory of Computing*. 2002, pp. 380–388.

[31] Arthur P Dempster, Nan M Laird, and Donald B Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B (methodological)* 39.1 (1977), pp. 1–22.

[32] Douglas A Reynolds et al. “Gaussian mixture models.” In: *Encyclopedia of Biometrics* 741.659-663 (2009).

[33] Matthew A Jaro. “Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida”. In: *Journal of the American Statistical Association* 84.406 (1989), pp. 414–420.

[34] Vladimir I Levenshtein et al. “Binary codes capable of correcting deletions, insertions, and reversals”. In: *Soviet Physics Doklady*. Vol. 10. 8. Soviet Union. 1966, pp. 707–710.

[35] Arturs Backurs and Piotr Indyk. “Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)”. In: *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*. 2015, pp. 51–58.

[36] Aug. 2022. URL: https://www.archives.gov/research/census/soundex.

[37] The Apache Software Foundation. *Spark python api docs*. May 2021. URL: https://spark.apache.org/docs/2.4.0/api/python/index.html.

[38] Max Bachmann. *Maxbachmann/Jarowinkler: Python library for fast approximate string matching using Jaro and Jaro-Winkler similarity*. Sept. 2022. URL: https://github.com/maxbachmann/JaroWinkler.

[39] Robin Linacre et al. “Splink: Free software for probabilistic record linkage at scale.” In: *International Journal of Population Data Science* 7.3 (Aug. 2022). DOI: 10.23889/ijpds.v7i3.1794. URL: https://ijpds.org/article/view/1794.

[40] J Clausnitzer. *Finland: Most common surnames 2023*. Mar. 2023. URL: https://www.statista.com/statistics/1017219/most-common-surnames-finland/#:~:text=As%20of%20February%202023%2C%20Korhonen,Nieminen%2C%20M%C3%A4kel%C3%A4%2C%20and%20H%C3%A4m%C3%A4l%C3%A4inen..

[41] Jaime Sevilla et al. “Compute trends across three eras of machine learning”. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–8.

[42] URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html.

[43] Fred J Damerau. “A technique for computer detection and correction of spelling errors”. In: *Communications of the ACM* 7.3 (1964), pp. 171–176.

[44] Armand Joulin et al. “FastText.zip: Compressing text classification models”. In: *arXiv preprint arXiv:1612.03651* (2016).

[45] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *arXiv preprint arXiv:1607.04606* (2016).

[46] Andrea Tancredi and Brunero Liseo. “A hierarchical Bayesian approach to record linkage and population size problems”. In: (2011).

[47] Marco Fortini et al. “On Bayesian record linkage”. In: *Research in Official Statistics* 4.1 (2001), pp. 185–198.

[48] Brendan S McVeigh, Bradley T Spahn, and Jared S Murray. “Scaling Bayesian probabilistic record linkage with post-hoc blocking: an application to the california great registers”. In: *arXiv preprint arXiv:1905.05337* (2019).

[49] Nils Barlaug and Jon Atle Gulla. “Neural networks for entity matching: A survey”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15.3 (2021), pp. 1–37.

[50] Renzhi Wu et al. “Zeroer: Entity resolution using zero labeled examples”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1149–1164.

A. Ground truth settings

A.1 Stop words

A regular expression has been used to remove the any stop words from the ground truth strings. The regular expression is the following: `(?i)(\s|\b)(O\.?Y\.?J?(\:.\s)?|R\.?Y(?!\.)|R\.?Y\.?|A\.?B\.?P?(\:.\s)?|K\.?Y\.?|A\.?\/?S\.?|T\:?MI|LTD\.?|PLC|JA|L\.?L\.?C\.?|I\.?N\.?C\.?|S\.?\s?R\.?|P&C|&|\,)(\s|\b)`, which translates to the following list of symbols, words, and alternative spellings of them:

1. OYJ
2. RY
3. ABP
4. KY
5. AS
6. T:MI
7. LTD
8. PLC
9. JA
10. LLC
11. INC
12. SR
13. P&C
14. &
15. ,

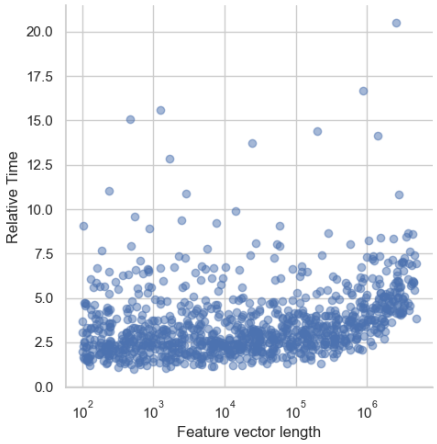
These wore empirically found to be present in the data and removed.

B. Additional parameter plots for the embedding-based pipelines

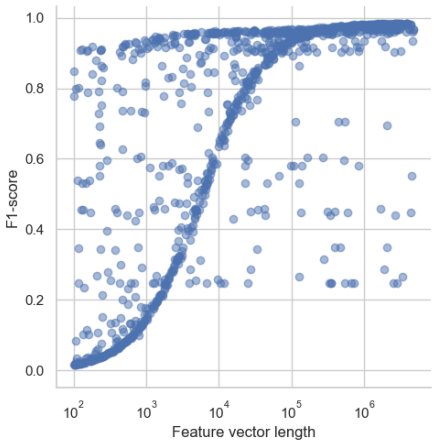
This appendix presents all plots related to the deterministic record linker parameter tuning. The plots are divided into sections by string-embedding model. Furthermore, subsection represent the primary explanatory variable and is further divided into sections by the secondary explanatory variable used to categories the points.

B.1 TF-IDF

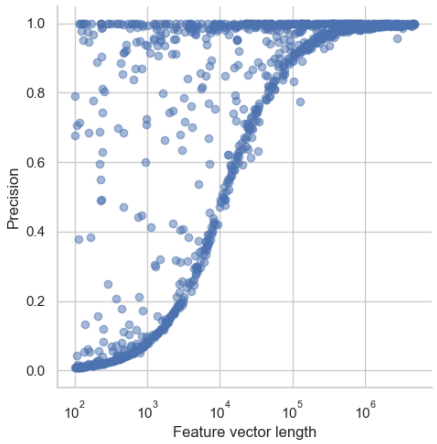
B.1.1 Feature vector length



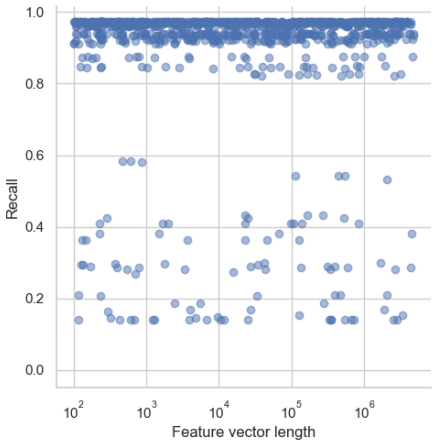
(a) The relative time needed to run.



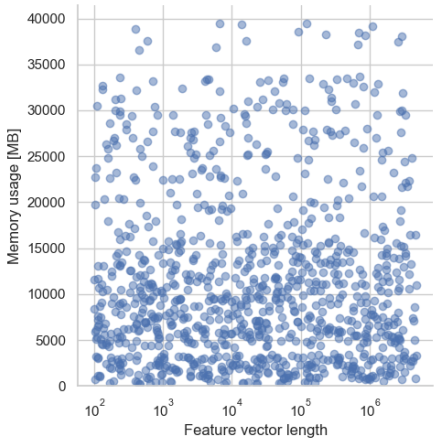
(b) The F1-score.



(c) The precision.



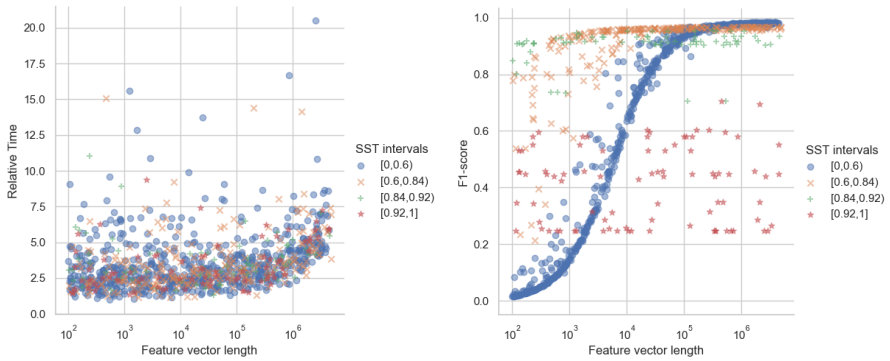
(d) The recall.



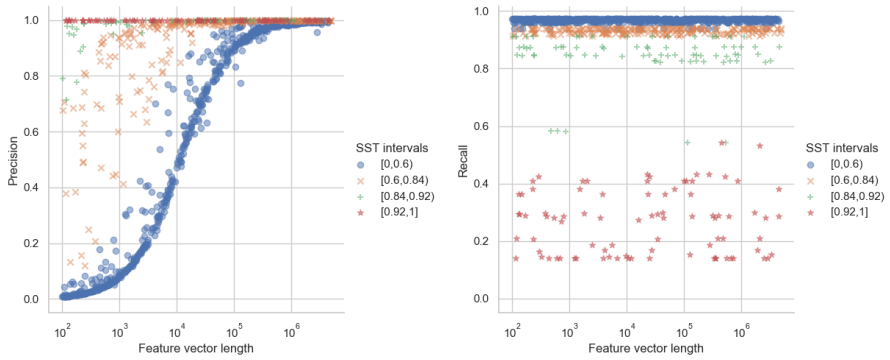
(e) The memory usage [MB].

Figure 2.1. The feature vector's length of the TF-IDF-based algorithm as a function of the performance metrics.

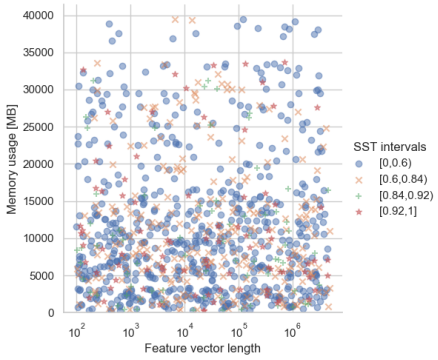
String similarity threshold



(a) The relative time needed to run. **(b)** The F1-score.



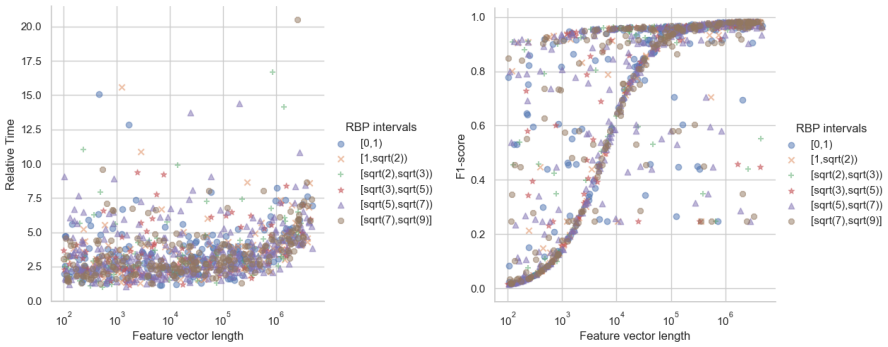
(c) The precision. **(d)** The recall.



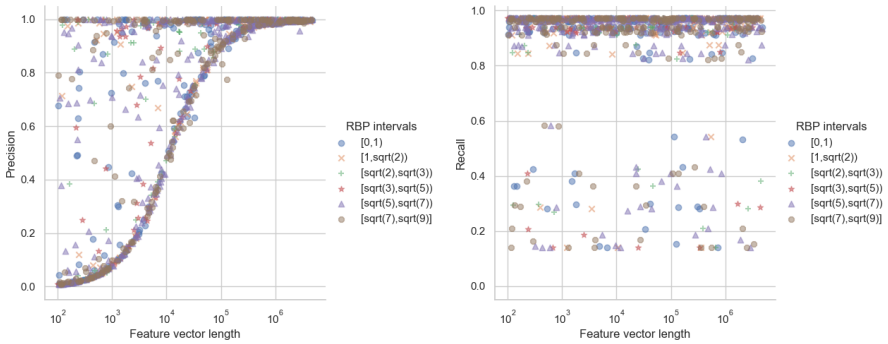
(e) The memory usage [MB].

Figure 2.2. The feature vector's length of the TF-IDF-based algorithm as a function of the performance metrics divided into intervals by the string similarity threshold.

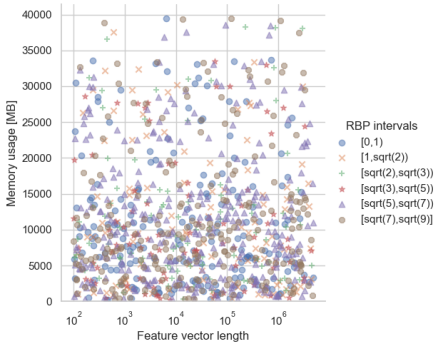
Random bucket projection distance threshold



(a) The relative time needed to run. **(b)** The F1-score.



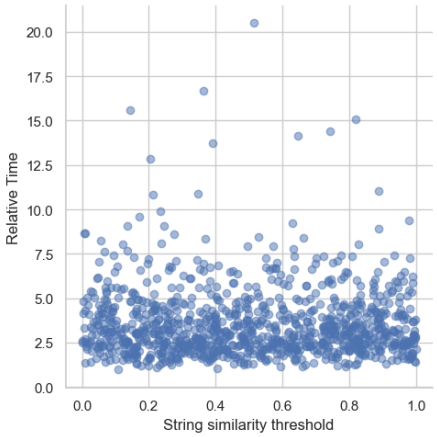
(c) The precision. **(d)** The recall.



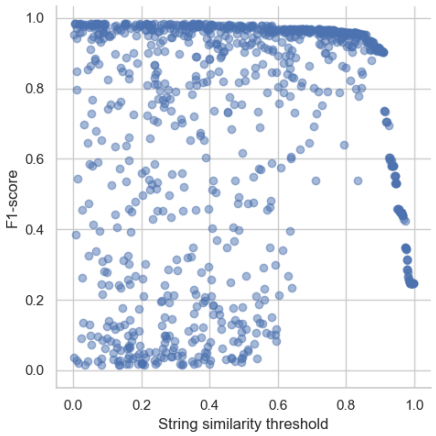
(e) The memory usage [MB].

Figure 2.3. The feature vector's length of the TF-IDF-based algorithm as a function of the performance metrics divided into intervals by the random bucket projection distance threshold.

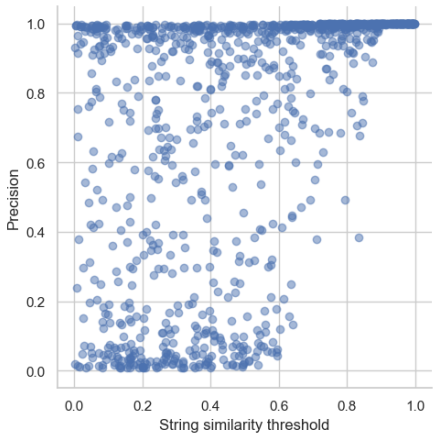
B.1.2 String similarity threshold



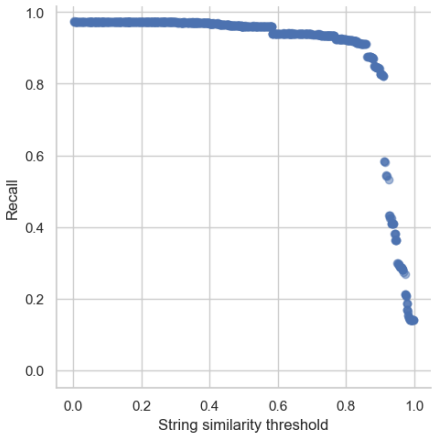
(a) The relative time needed to run.



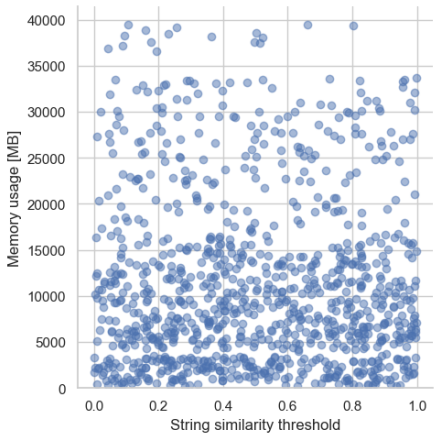
(b) The F1-score.



(c) The precision.



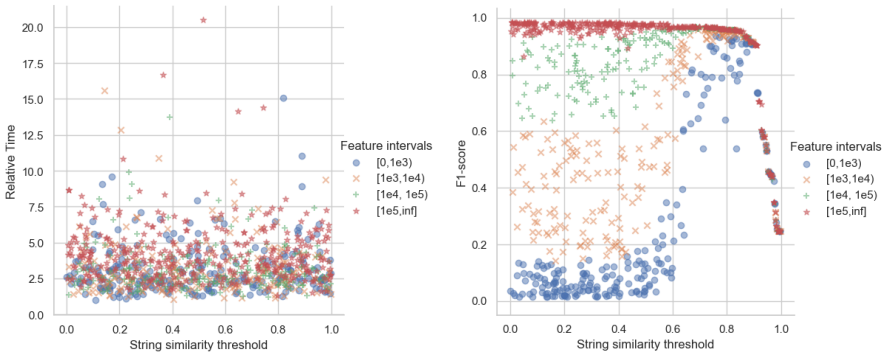
(d) The recall.



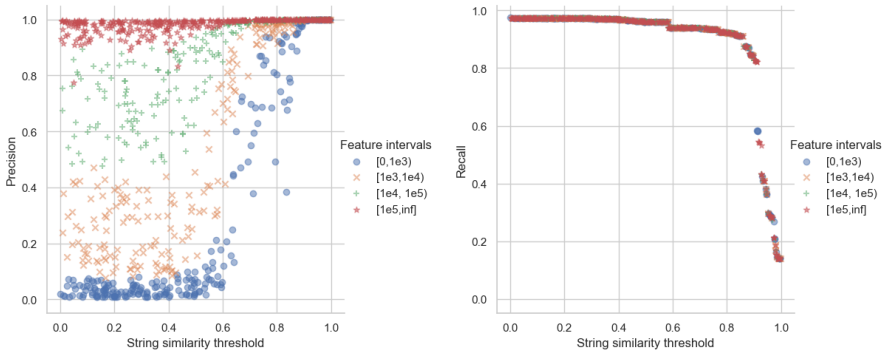
(e) The memory usage [MB].

Figure 2.4. The string similarity threshold of the TF-IDF-based algorithm as a function of the performance metrics.

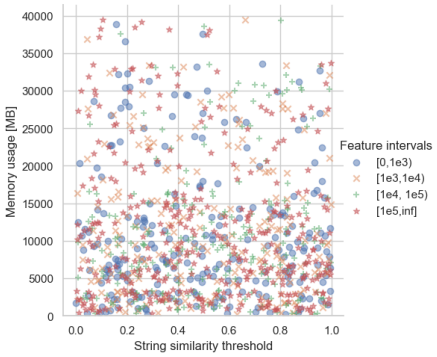
Feature vector length



(a) The relative time needed to run. **(b)** The F1-score.



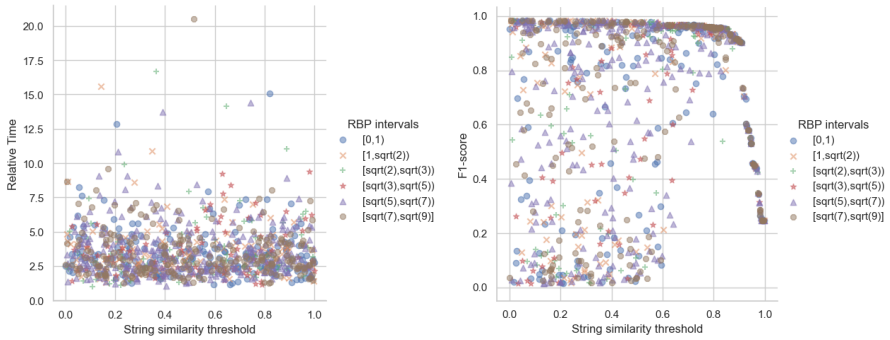
(c) The precision. **(d)** The recall.



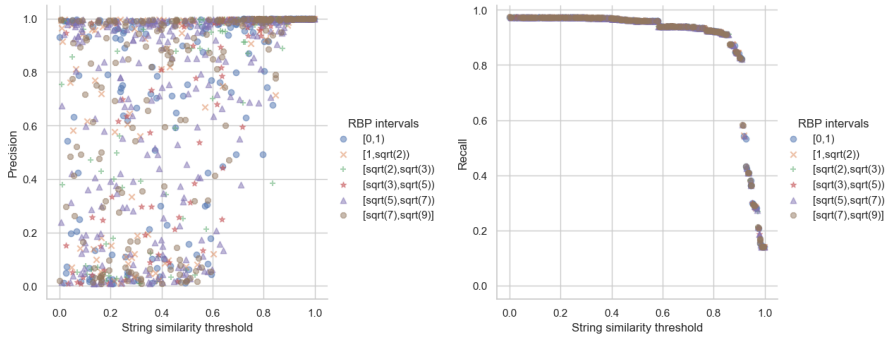
(e) The memory usage [MB].

Figure 2.5. The string similarity threshold of the TF-IDF-based algorithm as a function of the performance metrics divided into intervals by the feature vector's length.

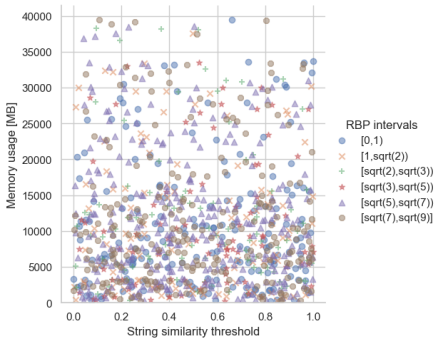
Random bucket projection distance threshold



(a) The relative time needed to run. **(b)** The F1-score.



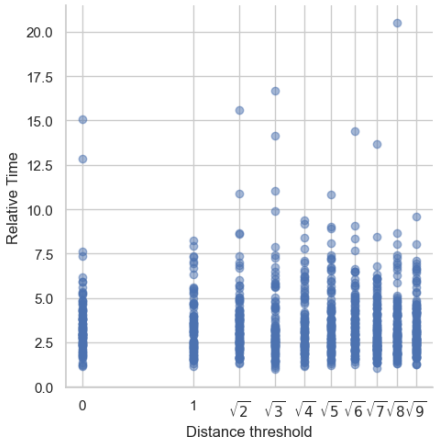
(c) The precision. **(d)** The recall.



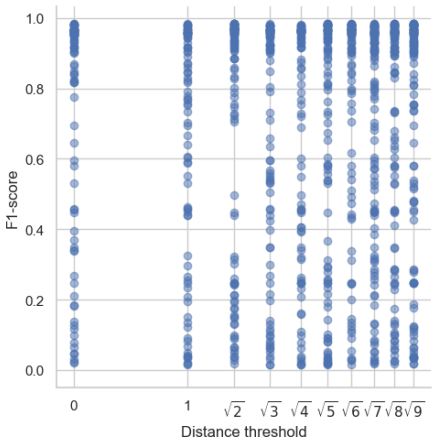
(e) The memory usage [MB].

Figure 2.6. The string similarity threshold of the TF-IDF-based algorithm as a function of the performance metrics divided into intervals by the random bucket projection distance threshold.

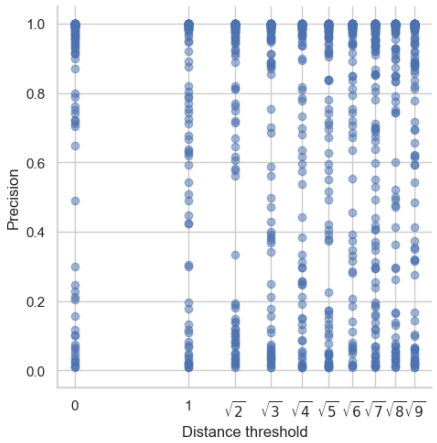
B.1.3 Random bucket projection distance threshold



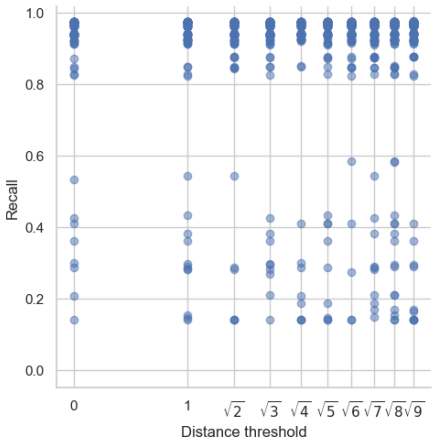
(a) The relative time needed to run.



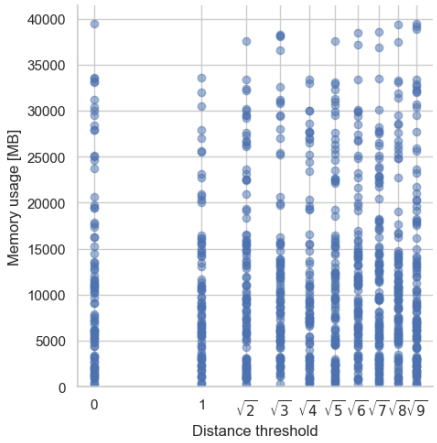
(b) The F1-score.



(c) The precision.



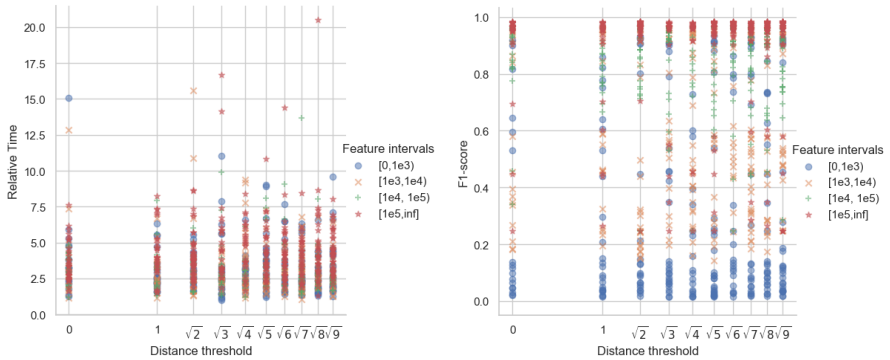
(d) The recall.



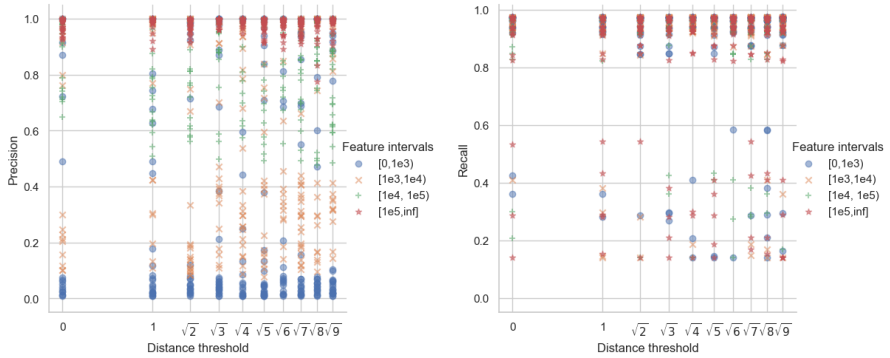
(e) The memory usage [MB].

Figure 2.7. The random bucket projection distance threshold of the TF-IDF-based algorithm as a function of the performance metrics.

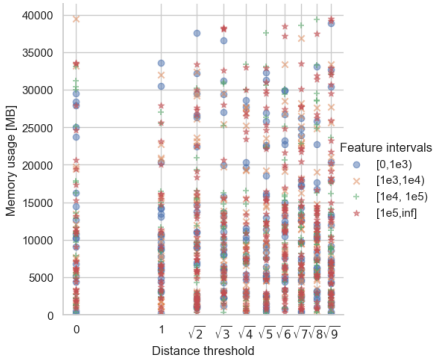
Feature vector length



(a) The relative time needed to run. **(b)** The F1-score.



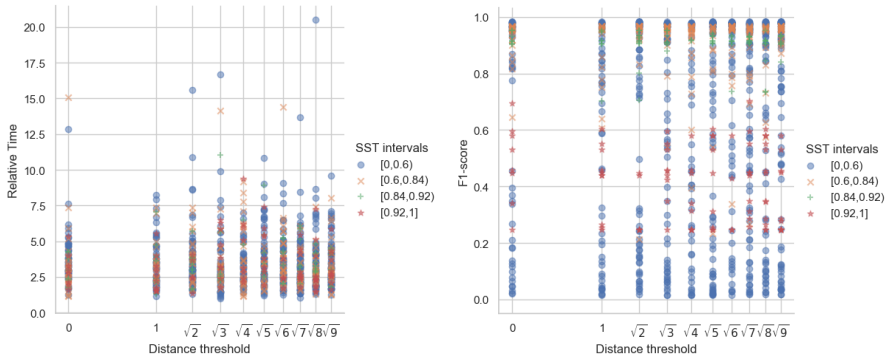
(c) The precision. **(d)** The recall.



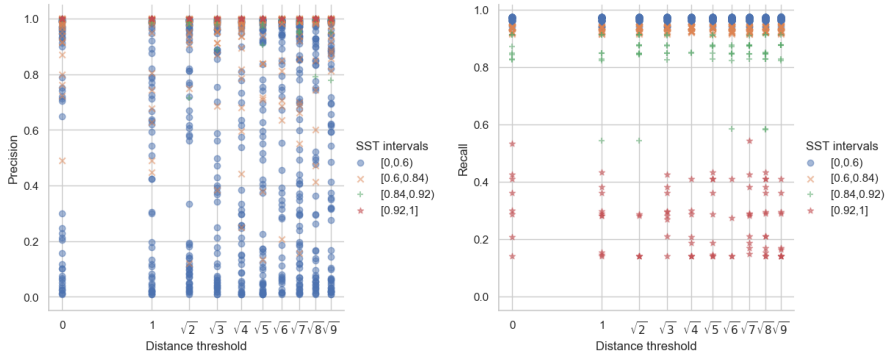
(e) The memory usage [MB].

Figure 2.8. The random bucket projection distance threshold of the TF-IDF-based algorithm as a function of the performance metrics divided into intervals by the feature vector's length.

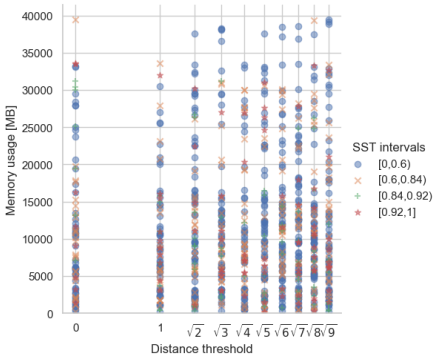
String similarity threshold



(a) The relative time needed to run. **(b)** The F1-score.



(c) The precision. **(d)** The recall.

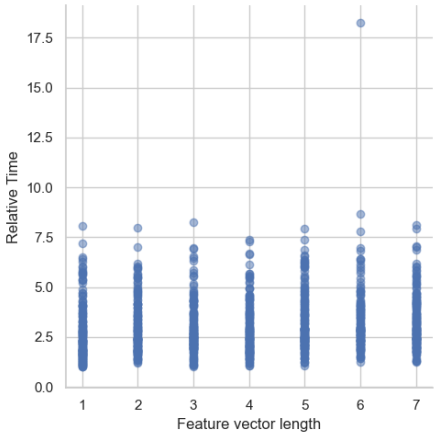


(e) The memory usage [MB].

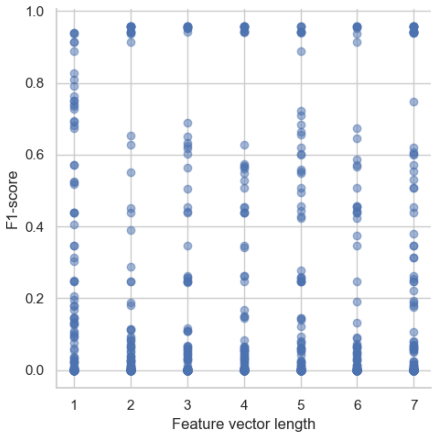
Figure 2.9. The random bucket projection distance threshold of the TF-IDF-based algorithm as a function of the performance metrics divided into intervals by the string similarity threshold.

B.2 Word2Vec

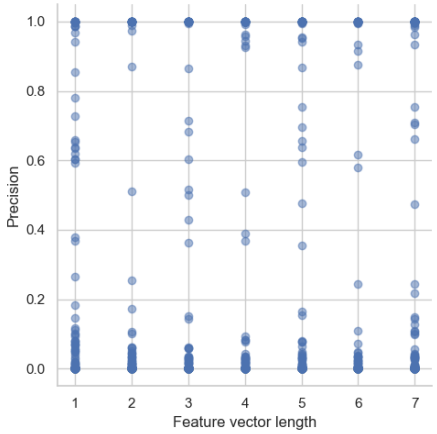
B.2.1 Feature vector length



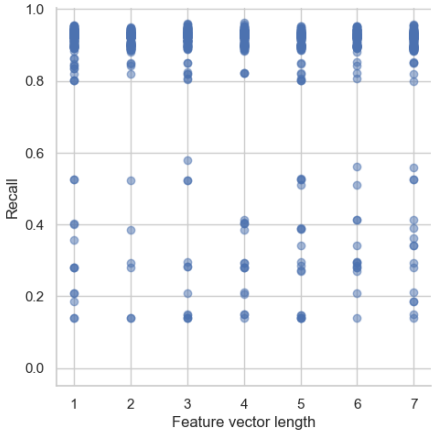
(a) The relative time needed to run.



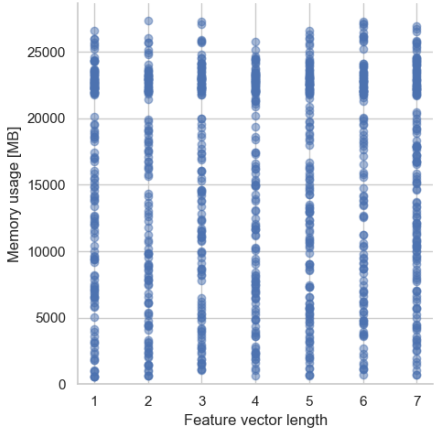
(b) The F1-score.



(c) The precision.



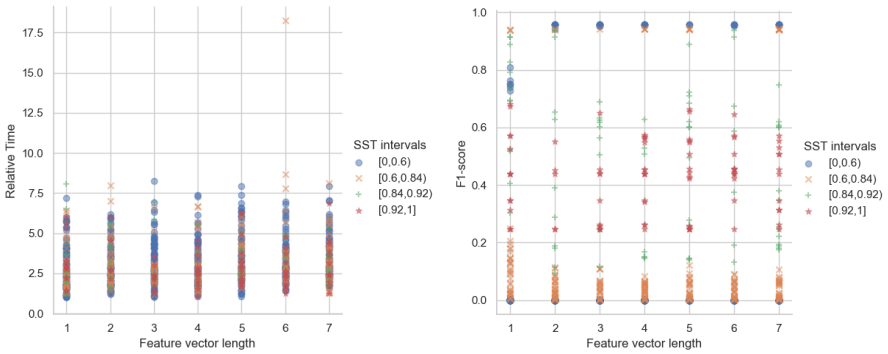
(d) The recall.



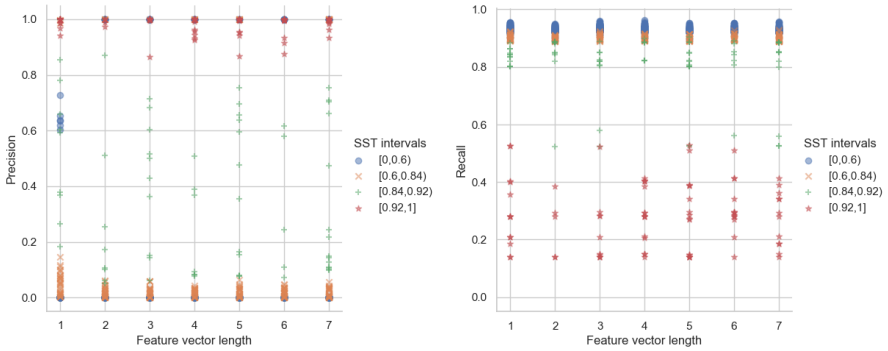
(e) The memory usage [MB].

Figure 2.10. The feature vector's length of the Word2Vec-based algorithm as a function of the performance metrics.

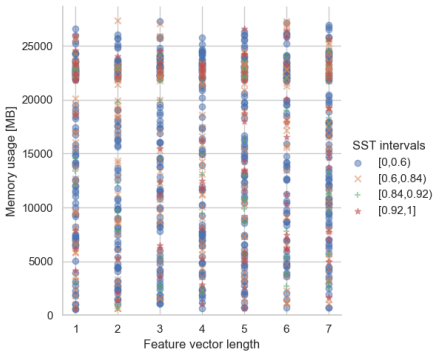
String similarity threshold



(a) The relative time needed to run. **(b)** The F1-score.



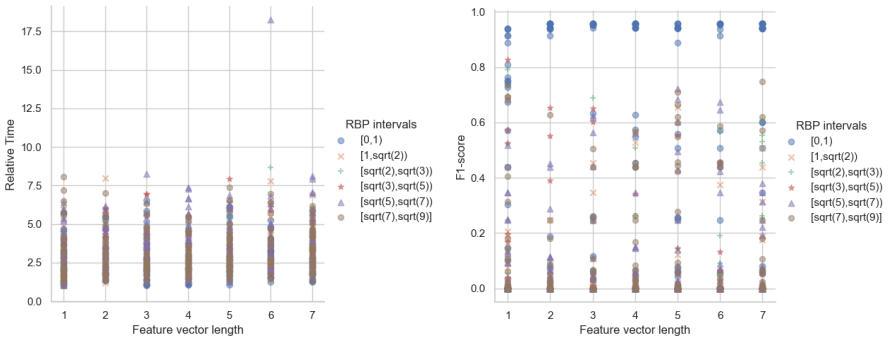
(c) The precision. **(d)** The recall.



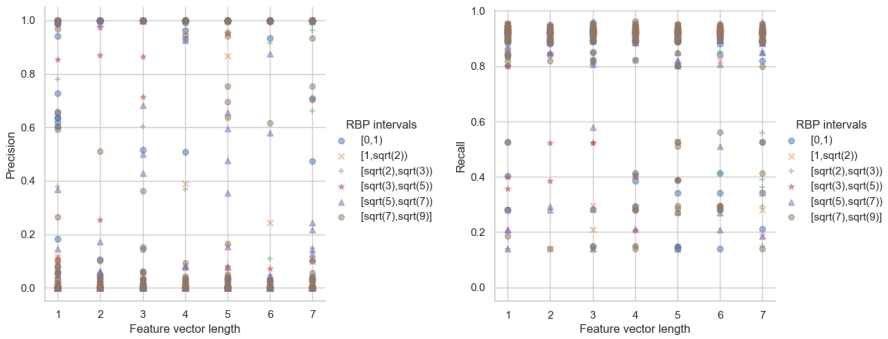
(e) The memory usage [MB].

Figure 2.11. The feature vector's length of the Word2Vec-based algorithm as a function of the performance metrics divided into intervals by the string similarity threshold.

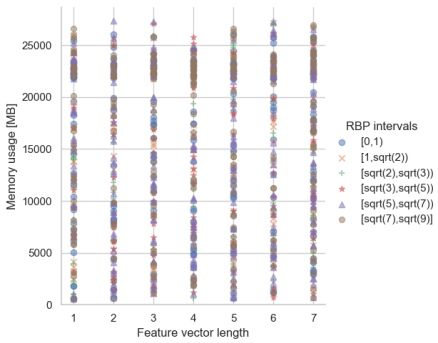
Random bucket projection distance threshold



(a) The relative time needed to run. **(b)** The F1-score.



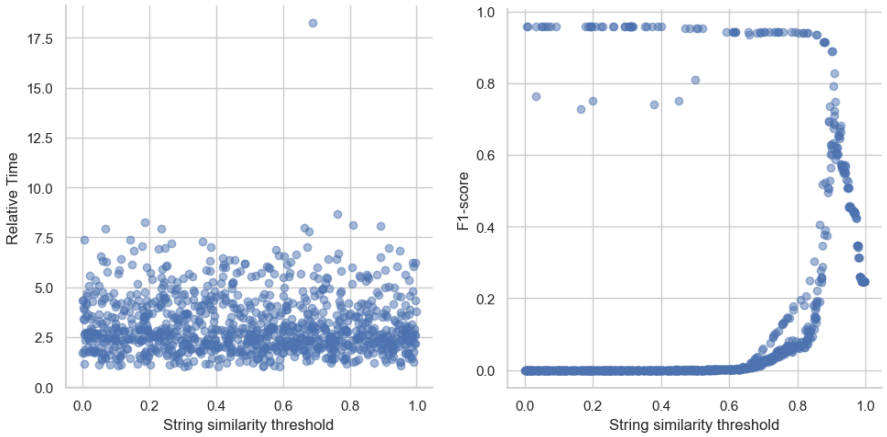
(c) The precision. **(d)** The recall.



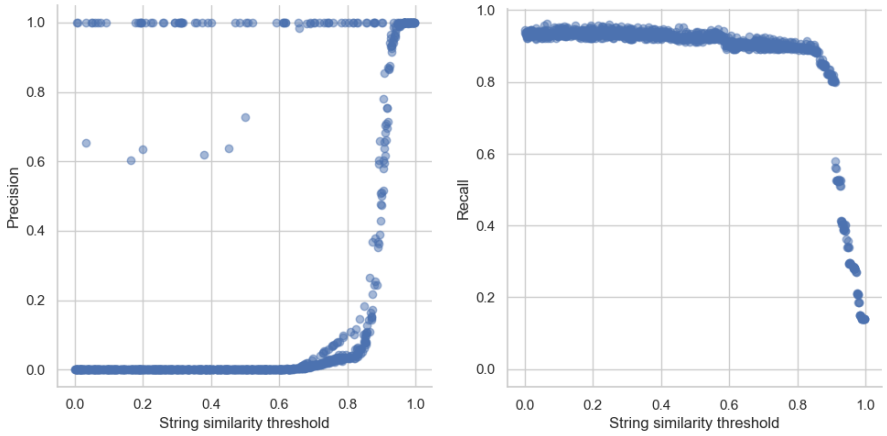
(e) The memory usage [MB].

Figure 2.12. The feature vector's length of the Word2Vec-based algorithm as a function of the performance metrics divided into intervals by the random bucket projection distance threshold.

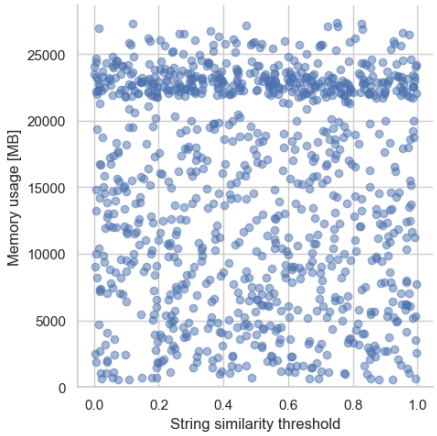
B.2.2 String similarity threshold



(a) The relative time needed to run. **(b)** The F1-score.



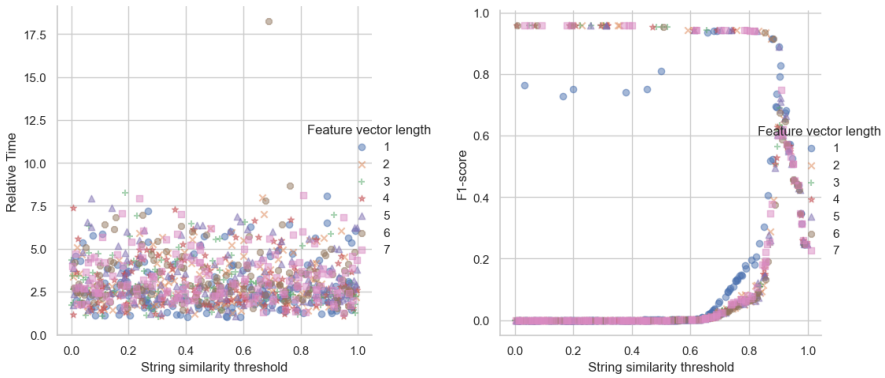
(c) The precision. **(d)** The recall.



(e) The memory usage [MB].

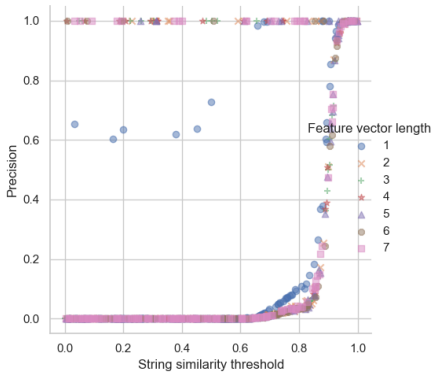
Figure 2.13. The string similarity threshold of the Word2Vec-based algorithm as a function of the performance metrics.

Feature vector length

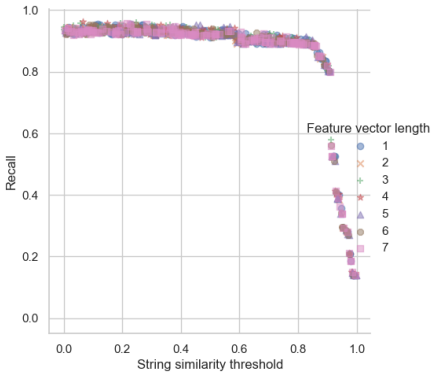


(a) The relative time needed to run.

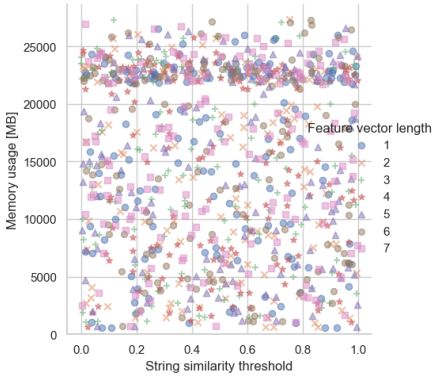
(b) The F1-score.



(c) The precision.



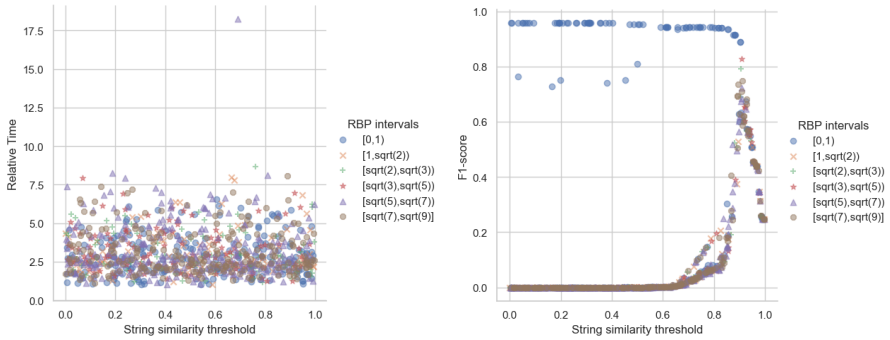
(d) The recall.



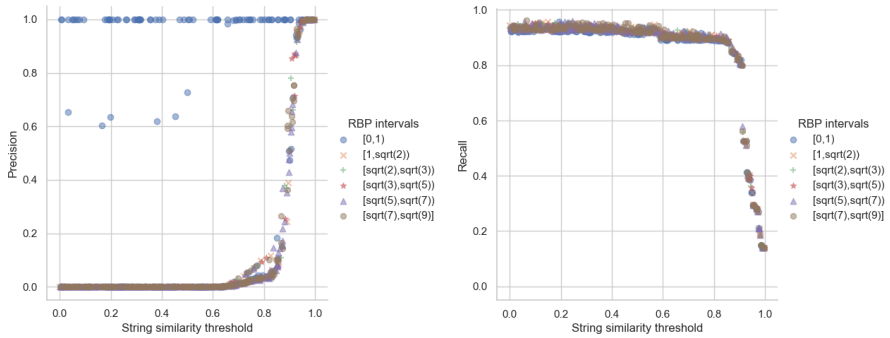
(e) The memory usage [MB].

Figure 2.14. The string similarity threshold of the Word2Vec-based algorithm as a function of the performance metrics divided into intervals by the feature vector's length.

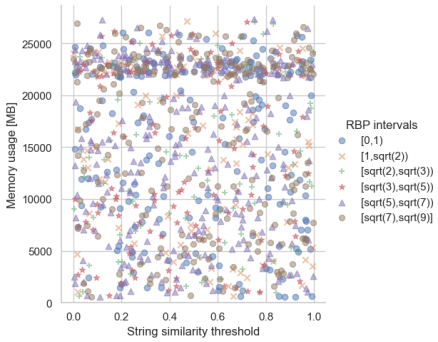
Random bucket projection distance threshold



(a) The relative time needed to run. **(b)** The F1-score.



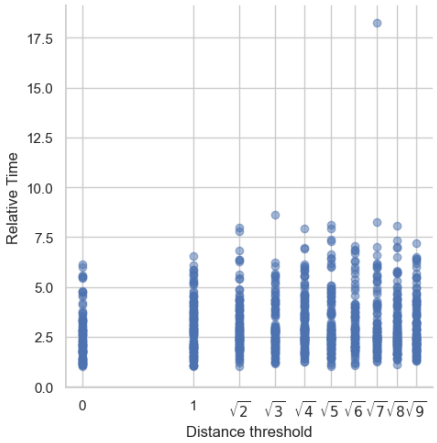
(c) The precision. **(d)** The recall.



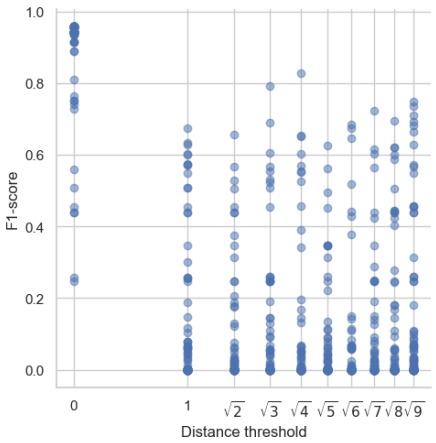
(e) The memory usage [MB].

Figure 2.15. The string similarity threshold of the Word2Vec-based algorithm as a function of the performance metrics divided into intervals by the random bucket projection distance threshold.

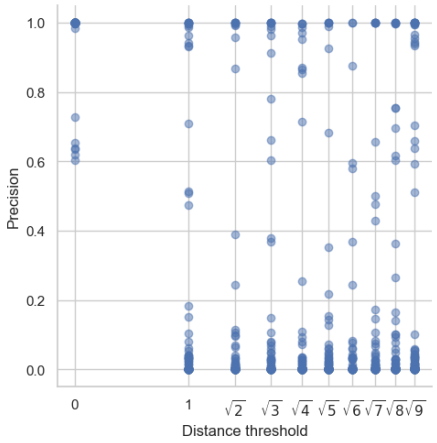
B.2.3 Random bucket projection distance threshold



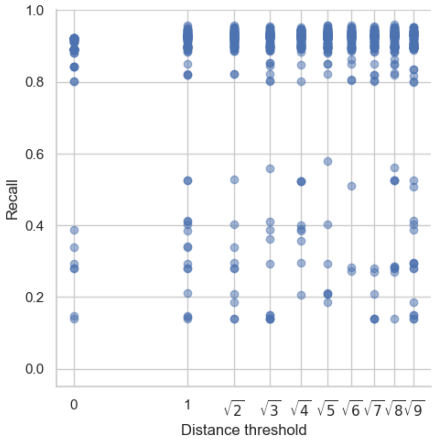
(a) The relative time needed to run.



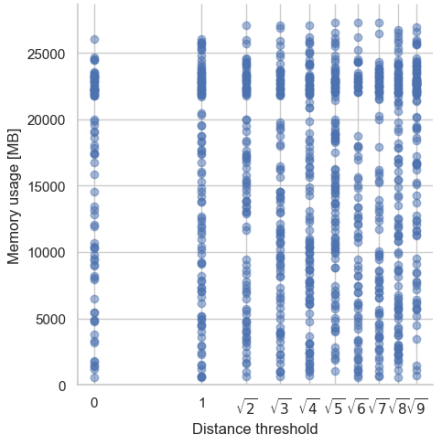
(b) The F1-score.



(c) The precision.



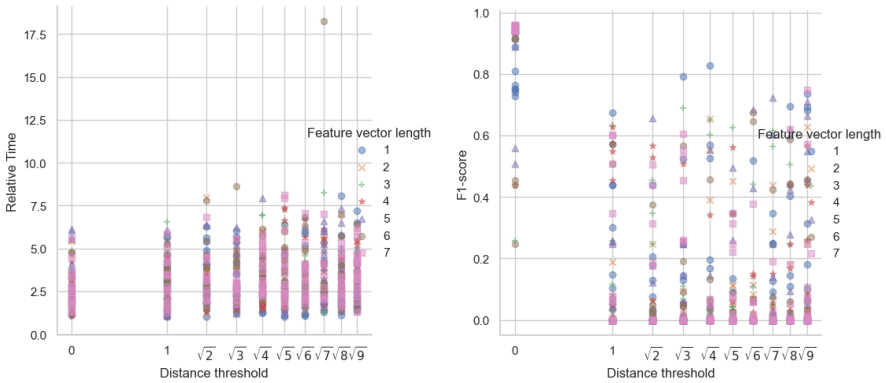
(d) The recall.



(e) The memory usage [MB].

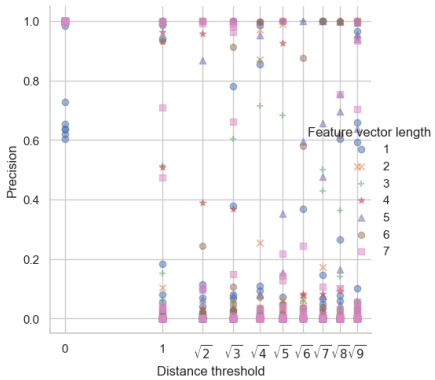
Figure 2.16. The random bucket projection distance threshold of the Word2Vec-based algorithm as a function of the performance metrics.

Feature vector length

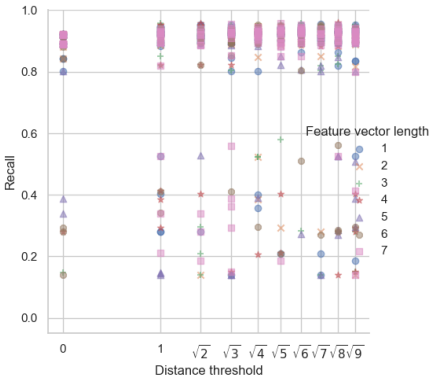


(a) The relative time needed to run.

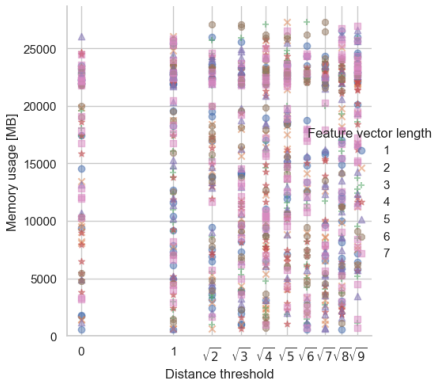
(b) The F1-score.



(c) The precision.



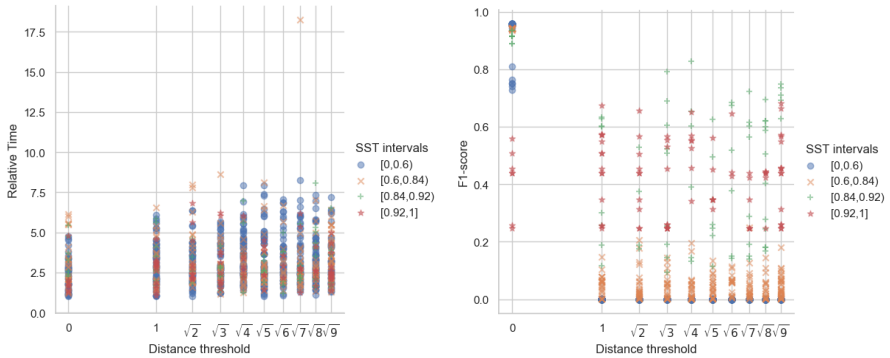
(d) The recall.



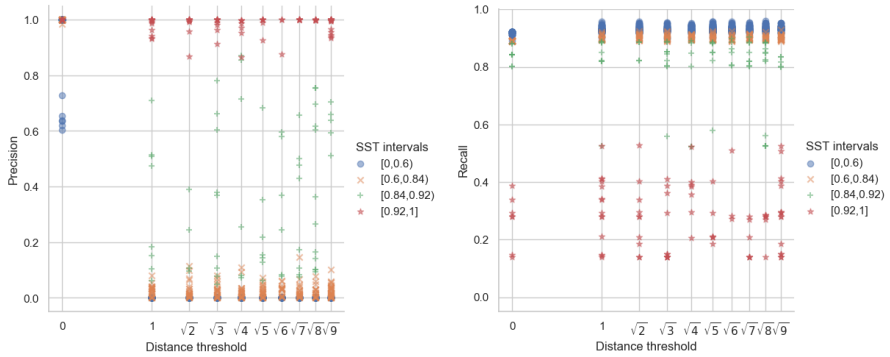
(e) The memory usage [MB].

Figure 2.17. The random bucket projection distance threshold of the Word2Vec-based algorithm as a function of the performance metrics divided into intervals by the feature vector's length.

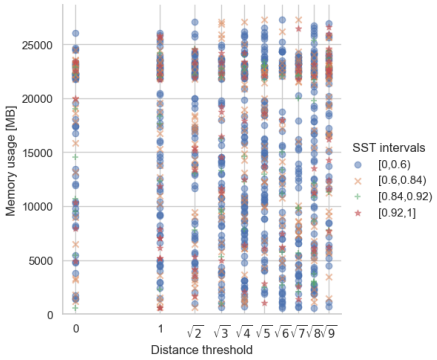
String similarity threshold



(a) The relative time needed to run. **(b)** The F1-score.



(c) The precision. **(d)** The recall.



(e) The memory usage [MB].

Figure 2.18. The random bucket projection distance threshold of the Word2Vec-based algorithm as a function of the performance metrics divided into intervals by the string similarity threshold.

C. Splink settings

This appendix presents the settings used in relation to the Splink-package. The version of the Splink-pakage the thesis uses is version 3.9.2. When referenced to, the "l" and "r" SQL tables stand for the tables representing the DJ respectively the DnB data sets.

C.1 Case statements

C.1.1 First name

The first name is compared on using the following statements:

1. Exact match
2. Exact soundex match
3. All other comparisons

C.1.2 Last name

The last name is compared on using the following statements:

1. Exact match
2. Exact soundex match
3. All other comparisons

C.1.3 Middle names

The middle names is compared on using the following statements:

1. Exact match

2. Levensthein distance of less than or equal to one

3. Levensthein distance of less than or equal to two

4. All other comparisons

Note that in the case of multiple middle names they are not compared separately.

C.1.4 Date of birth

The date of birth is compared on using the following statements:

1. Exact match

2. Year and month matches, i.e., the 7 first characters matches and either date of birth ends in "01"

3. Year matches, i.e., the 4 first characters matches and either date of birth ends in "01-01"

4. All other comparisons

Note that the the date of birth is written in the form "YYYY-MM-DD".

C.1.5 Country

The country is compared on using the following statements:

1. Exact match

2. All other comparisons

C.2 Global blocking rules

1.
```
(    l.firstname_soundex = r.firstname_soundex
     OR l.firstname LIKE CONCAT('%', r.middlenames, '%')
     OR r.firstname LIKE CONCAT('%', l.middlenames, '%'))
   AND left(l.lastname, 1) = left(r.lastname, 1)
```

   generates 288 875 665 new unique pairs

2.
```
l.lastname_soundex = r.lastname_soundex
```

   generates 3 767 685 new unique pairs

3.
```
left(l.date_of_birth,4) = left(r.date_of_birth,4)
 AND left(l.firstname,1) = left(r.firstname,1)
 AND left(l.lastname,1) = left(r.lastname,1)
 AND l.country = r.country
```

   generates 108 362 new unique pairs

C.3 Local blocking rules

1.
```
l.firstname_soundex = r.firstname_soundex
AND l.lastname_soundex = r.lastname_soundex
```

   generates 133 934 pairs

2.
```
l.date_of_birth = r.date_of_birth
OR (left(l.date_of_birth,7) = left(r.date_of_birth,7) AND (right(l.date_of_birth,2) = '0
OR (left(l.date_of_birth,4) = left(r.date_of_birth,4) and (right(r.date_of_birth,5) = '0
```

   generates 2 663 169 pairs

3.
```
'left(l.date_of_birth,4) = left(r.date_of_birth,4) AND l.country = r.country'
```

   generates 13 953 644 pairs

4. `l.country = r.country AND l.firstname_soundex = r.firstname_soundex`

   generates 52 331 322 pairs

5. `l.country = r.country AND l.middlenames = r.middlenames`

   generates 15 085 053 pairs

6. `'l.country = r.country AND l.lastname_soundex = r.lastname_soundex'`

   generates 9 760 947 pairs

7. `'l.middlenames = r.middlenames AND l.lastname_soundex = r.lastname_soundex'`

   generates 44 935 pairs

C.4 Deterministic rules for priori probability

```
    l.firstname_soundex = r.firstname_soundex
AND l.lastname_soundex = r.lastname_soundex
AND left(l.date_of_birth,4) = left(r.date_of_birth,4)
AND l.country = r.country
```