Personal Names Records Detection And Linkage In Unstructured Dutch Text For Anonymisation

Mohamed, Rodaina

Award date:
2021

Technische Universiteit Eindhoven University of Technology

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Personal Names Records Detection And Linkage In Unstructured Dutch Text For Anonymisation

Master Thesis
The Erasmus Mundus
Joint Master Degree Program in
Big Data Management and Analytics
(BDMA)

Rodaina Mohamed

Supervisor:
Dr. Natalia Sidorova

Eindhoven, August 2021

Abstract

A huge amount of text is being generated daily such as emails, reports, articles, documents, postings, ..etc. Text analytics has consequently become an increasingly valuable field for research. Text analytics can provide businesses with beneficial insights about the market and the clients' sentiment towards their products or services. However, text analytics can be very challenging, given the unstructured and inconsistent nature of this text. Furthermore, global and local privacy regulations have restricted businesses by different laws that enforce individuals' privacy. Consequently, it is crucial to preserve the privacy of individuals mentioned in any textual data to be analyzed with minimal information loss. This project aims to investigate the challenges that entail anonymizing personal names in unstructured Dutch text. The objective is to enable text analytics without breaching individuals' privacy nor losing links between personal name records. Given the inconsistency of syntax of personal names, detecting and linking names in an unstructured text can be a complex task. In this project, we tackle two types of inconsistencies: (1) Typographical errors, and (2) Name variants (e.g., nicknames, or alternative spellings). Our approach consists of 3 phases. The first phase is detecting all personal name records in the given unstructured Dutch text. The accuracy of the personal name detection, using Stanza named-entity recognition system and the list-based combination approach, is 92%. Personal names with different typographical error types were detected from the text with an accuracy of up to 90%. This was achieved using the properties of edit-based distance (e.g., Levenshtein distance). The second phase is linking records by matching co-referent personal name records that refer to the same individual. Using learnable similarity classifications, different static similarity scores were used as features to train a machine learning model to classify co-referent and non-co-referent name pairs. The model was able to classify pairs of names with an accuracy of approximately 79%. Finally, in the last phase, all personal name records were anonymized using one-way hashes, that do not allow data retrieval. Moreover, all the links were stored between records that can assist in further text analytics.

Keywords Text Analytics . Data Anonymisation . Named-entity Recognition . Learnable Similarity Classification . Static Similarity Scores

Preface

This master thesis presents the result of my graduation for Erasmus Mundus Joint Master Degree Program in Big Data Management and Analytics (BDMA) and was undertaken at Mathematics and Computer Science department of Eindhoven University of Technology (TU/e).

Firsly and foremost, I am so grateful for my supervisor, Natalia Sidorova, for her supervision, thorough guidance, and critical feedback. I truely appreciate her patience and support throughout the whole duration of this project. Secondly, I would like to express my gratitude towards my secondary advisor, Fatemeh Shafiee, for her continuous support that helped me overcome a lot of the challenges faced during this project.

Thirdly, I would like to thank my family, especially my mother, Somia, and my father, Ahmed, for all the emotional support that kept me going despite the all the hardship. I would also like to express my appreciation and gratitude to my brothers, Abdelrhman and Ziad, and my sister, Nejar, for motivating me to strive for a better future.

Finally, I would like to thank my great friend Sara Bashir for being always here for me whenever I needed her. She has been always supportive, patient, and kind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This master thesis is a graduation project for Erasmus Mundus Joint Master Degree in Big Data Management and Analytics (BDMA) with specialization in Business Process Analytics at Mathematics and Computer Science department of Eindhoven University of Technology (TU/e). The first section of this chapter covers an introduction to the context of this graduation project. In the second section, the research question which we have aimed to answer is formulated. The next section presents the steps carried out while doing the project. Finally, this chapter ends with an outline of the following chapters.

1.1 Thesis Context

Humans generate a massive amount of unstructured text data on regular basis through emails, social media postings, instant messaging, documents, and others. Many corporations store massive amounts of this unstructured text to earn a competitive advantage in their markets. Consequently, text analytics has become crucial for businesses that are willing to enhance their business intelligence strategy through their data assets. Text analytics aims to analyze and obtain information from unstructured text. However, privacy concerns have been raised when analyzing such data. Different forms of text can contain identifiers such as surnames, given names, date of birth, or address information. Consequently, data anonymization has become necessary in many fields of research including text analytics. The term "anonymization" refers to the process of removing the ability to identify individuals in a data set. Data anonymization has become a tool to extract valuable insights from data analysis while minimizing the risks to people involved. When personal data is anonymized, it is no longer considered to be personal data. As a result, the processing of such data falls outside the scope of the Data Processing Act according to the The Dutch Data Protection Authority (Dutch DPA) [2]. One example could be medical research where most of the data can include patients' identifiers such as medical reports. To be able to analyze these data for insights, all personal identifiers have to be anonymized.

Many pieces of research have been conducted to find a way to anonymize those identifiers without completely comprising the usability of the data for secondary analysis. Many challenges are encountered in solving the problem of detecting names in unstructured text. The first challenge is the typographical errors and misspellings of names in the text. Personal names in an unstructured text can be inconsistent and mistyped which makes it challenging to detect them. Another challenge is that personal names in the text usually have a many-to-many relationship with individuals. Meaning that one individual can be referred to in the text by different name variants such as nicknames, whereas two similar names can refer to two different individuals. Therefore, the objective is to find co-referent name variants before anonymization to minimize information loss without compromising privacy.

In the next section, the goal of this project and the defined research question are discussed.

1.2 Problem statement

This project aims to investigate and understand how accurately we can detect personal names in unstructured Dutch text. In addition, we explored ways to link personal name records in the text that refer to the same individuals to minimize information loss after anonymization. The research goals of this thesis project were formulated as follows:

Research Goal Anonymising personal names in unstructured Dutch text to allow text analytics without breaching the privacy of individuals mentioned in the text.

Identifying and linking personal names in an unstructured text can be a huge benefit that unleashes the potential of textual data for analytics. It is the main issue that when solved, can consequently solve the problem of comprising the usability of the data when anonymized. This can only happen when personal names records are linked before anonymization so that all name variants that refer to the same individuals are recognized. At the beginning of this project, we investigated methods by which we can most accurately detect names in unstructured Dutch text including names with different types of typographical errors. Afterward, we proposed a supervised classification approach for record linkage. This method relies on learnable similarity metrics between name pairs to decide if they are co-referent pairs or not. Co-referent names and non-co-referent names are classified using a machine learning model that is trained on features extracted using labeled pairwise string comparisons (e.g., static similarity scores). To conclude, the following research question was proposed. The details of how we tackled these research questions are provided in chapter 4.

Research Question How to detect all personal names in unstructured Dutch text as well as linking all the possible variations of records that refer to the same individual to minimize information loss after anonymization?

1.3 Research Approach

To achieve the previously mentioned goals of this project, the following steps were undertaken:

Literature Review: At the beginning and during the project, we did a thorough examination of the existing literature on named-entity recognition, string similarity metrics, record linkage, and data anonymization. This review helped us to get a better vision and learn more about the state of the art.

Tools: To identify names in the text, we utilized Stanza[1], an open-source python natural language processing package. It includes tools for text tokenization, part of speech and morphological tagging, and named entity recognition. Moreover, we used SK-learn[2], a python machine learning library for predictive analysis, to build a classification model for record linkage. Finally, we decided to utilize inforcehub[3] to anonymize personal names in the text with on-way encrypts that does not allow data re-retrieval.

Method: The followed method to tackle the proposed research questions was divided into three modules. The first module addresses the problem of detecting names in unstructured Dutch text. The second module is dedicated to record linkage of personal names where we classify co-referent and non-co-referent name pairs in the text. Finally, the last module is the phase where all names in the text are anonymized and links between names are stored.

[1]https://stanfordnlp.github.io/stanza/
[2]https://scikit-learn.org/stable/
[3]https://inforcehub.readthedocs.io/en/latest/modules/anon.html

Test case generation: To evaluate the method followed to achieve the main goals of this project, a test case was generated. Firstly, a function was created to embed random typographical errors to the personal names in the text. Pieces of text were generated with personal names with different types of typographical errors which are insertion, deletion, transposition, substitution, and replication errors. This is meant to evaluate our method in detecting names in unstructured Dutch text with typographical errors. On the other hand, we asked different Dutch individuals to provide us with a list of Dutch names with their variants to evaluate the performance of the classification model in identifying co-referent and non-co-referent pairs.

Evaluation of results: The project's final phase is to assess and evaluate the outcomes. This step is critical in any research since it gives an overview picture of the benefits and drawbacks of a proposed strategy. It will also make it easier to comprehend how to improve the research in future studies. In this thesis, we evaluated by investigating how effective the followed method is in allowing text analytics after anonymization of personal names in the text.

1.4 The Structure of This Thesis

This thesis is structured as follows:

Chapter 2 provides background information of the theories and tools used throughout this project such as named-entity recognition, and different similarity metrics including edit-based such as Levenshtein distance and Jaro-Winkler, and token-based such as cosine similarity and Jaccard similarity.

Chapter 3 presents an overview of the state-of-the-art methods and techniques related to named-entity recognition, record linkage, and data anonymization.

Chapter 4 presents the method and implementation steps followed to achieve the main goals of this project.

Chapter 5 presents the results and the performance evaluation.

Chapter 6 is the last chapter where the whole performed work and the outcome are summarized. Furthermore, possible developments and contributions are discussed.

Chapter 2

Preliminaries

In this section, we introduce concepts and tools utilized throughout the research conducted for this chapter.

2.1 Named Entity Recognition

Named entity recognition (NER) is an information extraction task that recognizes and categorizes mentions of various named entities in textual data, such as people's names, organizations' names, places, dates/times, monetary values,..etc. NER is a form of natural language processing (NLP) that is concerned with computers processing and analyzing natural languages. NER tools leverage supervised machine learning where the recognition of the named entities is typically modeled as a sequence prediction task. The objective is to assign a specific tag to each word in an input sentence of text. In this research, we utilized Stanza, a tool developed by the Stanford NLP group [3] in 2020, which contains a collection of accurate and efficient tools for the linguistic analysis of many human languages.

2.1.1 Stanza

Stanza [1] is a python package that contains various NER tools. It was built with neural network components that enable efficient training and evaluation with annotated data. It contains tools, which can be used in a pipeline, to convert a string containing human language text into lists of sentences and words, to generate base forms of those words, their parts of speech and morphological features, to give a syntactic structure dependency parse, and to recognize named entities. Stanza can recognise names in the text by considering the structure of the words and their position in the sentence. Figure 2.1 provides a summary of Stanza full neural network pipeline for robust text analytics, including tokenization, multi-word token (MWT) expansion, lemmatization, part-of-speech (POS) and morphological features tagging, dependency parsing, and named entity recognition.

[1]https://stanfordnlp.github.io/stanza

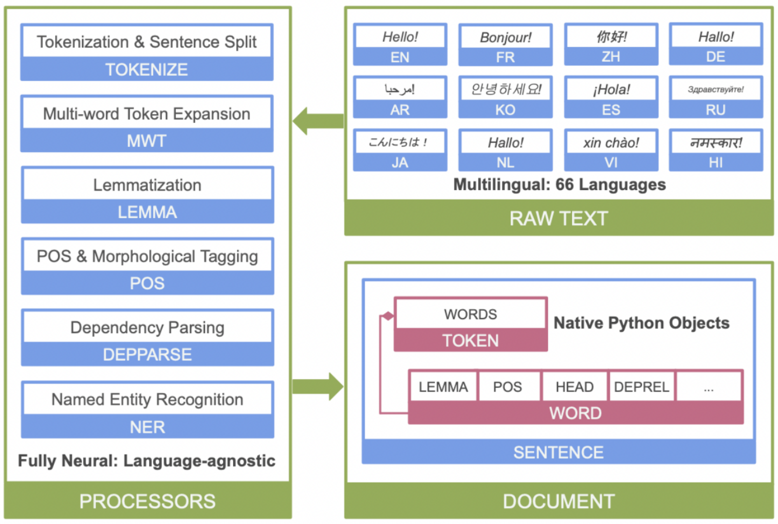


Figure 2.1: An overview of Stanza's neural network NLP pipeline [1]

2.2 Similarity Measures for Text Analysis

This section provides an overview of different similarity measures used in this research for the following two purposes.

1. Detection of personal names in the text with typographical errors.
2. Record linkage of personal name variants that refer to the same individual.

The similarity measures are divided into 3 types: edit-based , token-based, and phonetics.

2.2.1 Edit-based Similarity Measures

The edit-based measures approach compares two strings by counting the minimum number of operations required to transform the string into the other. This method calculates character-by-character distances between two terms by examining certain combinations of the following two factors (1) the number of identical characters, and (2) the number of edit operations used to convert one name to the other (the operations being: insert, erase, and transpose) [4]. This type of measure is able to detect minor differences between strings such as typographical errors. For example, if we are to compare the string 'Micheal' and 'Michiel', the edit-based measures can easily detect the resemblance between the two strings. This enables us to recognize name variants in an unstructured text that resulted from data entry errors or misspellings. We will consider the following edit-distance functions:

Levenshtein Distance

Levenshtein distance also called as minimum edit distance, measures the number of edits to transform one string to another. it assigns a unit cost to all edit operations according to the following equation [5]. The Levenshtein distance between two strings a,b (of length $|a|$, $|b|$ respectively) is given by $lev_{a,b}$ where:

$$lev_{a,b}(i,j) = \begin{cases} \text{Max(i,j)}, & \text{If } Min(i,j) = 0, \\ Min \begin{cases} \text{lev}_{a,b}(i-1,j)+1 \\ \text{lev}_{a,b}(i,j-1)+1 \\ \text{lev}_{a,b}(i-1,j-1)+1_{a_i \neq b_j} \end{cases} & \text{otherwise.} \end{cases}$$

Where *i,j* presents the terminal character position of string *a, b* respectively. a_i refers to the character of string *a* at position *i*. Similarily b_j refer to the character of string *b* at position *j*. $1_{a_i \neq b_j}$ indicates that the function is equal to 0 is $a_i \neq b_j$ and equals 1 otherwise. For example, the levenshtein distance between 'Anna' and 'Antje' is 3. Since , at a minimum, 3 edits are required to change one into the other corresponding to the following operations :

1. substitution of 'n' to 't'
2. substitution of 'a' to 'j'
3. insertion of 'e' at the end

Jaro-Wrinkler

Jaro-Wrinkler is a string metric measuring an edit distance between two sequences. It is a variant proposed in 1990 by *William E. Winkler* of the Jaro distance metric [6]. Promising results were achieved in the record-link literature using variants of this method that is based on the number and order of the common characters between two strings. The Jaro Similarity sim_j of two given strings s_1 and s_2 is

$$sim_j = \begin{cases} 0 & \text{if } m = 0, \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise.} \end{cases}$$

where $|s_i|$ is the length of the string s_i, m is the number of "matching characters" , and t is the number of transpositions . Note that, two characters from s_1 and s_2 respectively, only matches if they are the same and more than $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$ far characters apart. The number of matching (but different sequence order) characters divided by 2 defines the number of transpositions.
Jaro–Winkler similarity [6] uses a prefix scale p which higher score to strings that share the same prefix of length ℓ. Given two strings s_1 and s_2, their *Jaro–Winkler* similarity sim_w is

$$sim_w = sim_j + \ell p(1 - sim_j),$$

where sim_j is the *Jaro similarity* for strings s_1 and s_2, ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters, and p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes. Note that, p should not exceed 0.25, otherwise the similarity could become larger than 1. The standard value for this constant in Winkler's work is $p = 0.1$. The *Jaro–Winkler distance* d_w is defined as $d_w = 1 - sim_w$.

2.2.2 Phonetics-based Similarity Measures

In this approach, two strings are compared by the difference between the phonetic representation. These can also be referred to as the common key method. Those methods reduce the names to a key or code based on their pronunciation. One of the well-known common key methods is Soundex, patented in 1918, it maps different transformations of the letter depending on how similar this letter sounds to different numbers [4]. This allows names with similar pronunciations to be mapped to the same values. For example, Cyndi, Canada, Candy, Canty, Chant, Condie share the code C530. Another more accurate approach is the Metaphone algorithm, developed in 1990, followed by the Double Metaphone algorithm [2]. The Double Metaphone returns two codes: a primary key and a secondary key so you have more chances to match items. According to the algorithm, there are 3 matching levels as follows:

1. The strongest match when primary keys of two names match.
2. The normal match when a primary key of one name matches a secondary key of another name.
3. The weakest match when secondary keys of two names match.

2.2.3 Token-based Similarity Measures

Token-based similarities are more complicated measures than edit-based. They examine the text as a set of tokens (words) which allows for the semantic sense of words to be considered as well as the processing of vast documents. Since you can use word vector representations (word2vec) to explain each word in the text and then compare vectors, semantic meaning plays a role. Furthermore, using a bag of words approach and the TF-IDF tool, it is possible to compare the semantic similarity of entire texts (although not between independent words). Similarities based on tokens are commonly used in various fields. It is, without a doubt, the most well-known method of working with texts. Nonetheless, it is inapplicable to a wide variety of scenarios. To compute token-based distances, strings are considered multisets of tokens. Two strings are compared by looking at the intersection between their tokens. Table 3.1 shows an example of the term frequency vector of sample names 2-character tokens

Name/Token	'al'	'lb'	'be'	'er'	'rt'	'ro'	'ob'
Albert	1	1	1	1	1	0	0
Robert	0	0	1	1	1	1	1

Table 2.1: Term-frequency vector using 2grams

Jaccard Similarity

The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient, is a metric for determining how similar and diverse sample sets are. The Jaccard coefficient, which is defined as the size of the intersection divided by the size of the union of the sample sets, is a measure of similarity between finite sample sets.

Cosine Similarity

The cosine similarity metric is used to determine how similar strings are regardless of their size. It calculates the cosine of the angle formed by two vectors projected in a multi-dimensional space mathematically. In this project context, those vectors are arrays that contain letters counts of two strings. Because of the cosine similarity, even if two identical strings are separated by the Euclidean distance (due to the size of the strings), they are likely to be oriented closer together. The higher the cosine resemblance, the smaller the angle. The cosine similarity is described mathematically as the division between the dot product of vectors and the product of the euclidean norms or magnitude of each vector as shown in the following formula [7].

$$similarity = cos(\theta) = \frac{A.B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

where A_i and B_i are components of vector A and B respectively.

Chapter 3

Literature Review

In this chapter, we explored existing research focusing on detecting, linking, and anonymizing names in unstructured text. This review provides us with useful information on the direction of ongoing research and potential ways to contribute to this field of study. Anonymisation of private data in a structured text has been widely studied in recent years, whereas anonymizing data in unstructured remains a manual task that needs researchers' attention. The main challenge with unstructured text is that it is schema-less which makes it very difficult to quantify possible privacy risks. We studied different approaches from recent research papers that investigate record linkage, similarity metrics, and data anonymization in text analytics. The following sections show the outcome of the investigation done regarding this topic.

3.1 Anonymization of Unstructured Data via Named-Entity Recognition

Hassan and et al. [8] developed an approach that aims to build a system that is able to detect attributes that have any privacy implications in unstructured text. As a proof of concept, the paper focused on locating disease names in medical records. The paper proposes a model of the recognizer for named entities that consists of 3 main steps as follows:

1. Tokenizer
2. Feature extractor
3. Conditional Random Field (CRF) Model

In the first step, the tokenizer splits the sentences of a given text into tokens. Those tokens are then used in the next step to extract features using a window of 3 words (current word, previous word, and next word). In the feature extraction step, four features were considered for each token/word as follows:

- Word stem, where stems of the words are extracted (e.g. the stem of 'illness' is 'ill').
- Word length, which denotes the length of the word/token.
- Word Shape, which represents the shape of the word (e.g. 'lowercase, 'uppercase', 'capitalized', or 'mixed').
- Word POS, which is the part of speech for the word.

Lastly, in the third step of the model architecture, the CRF model takes the features extracted from the second step and produces a sequence of tags for the whole sentence. Consequently, the proposed model based on CRF is able to extract diseases names from a given medical record by predicting a sequence of corresponding tags. The machine learning-based approach proposed in this paper is argued to out-performs the dictionary-based named-entity recognition approaches where problems usually arise when the entities to be detected are not in the dictionary in use.

3.2 Combining Neural Networks and Knowledge-based Approaches to Named-Entity Recognition in Polish

This paper proposes another framework for named-entity recognition that combines knowledge-based feature extractors and a deep learning model including contextual word embeddings, long short-term memory (LSTM) layers and conditional random fields (CRF) inference layers [9]. The general architecture of the proposed framework consist of the following four modules:

1. An entity linking model based on Wikipedia
2. A feature extractor module, which integrates the aforementioned entity linker and a number of lexicons.
3. LSTM model
4. CRF model

To support the detection of named entities, the presented approach uses a number of feature extractors where some use static resources and others are based on heuristic rules. The purpose of the feature extractor module is to assign labels to words in the input sequence. One of the distinctive features of this system is the use of Wikipedia as a supplementary resource to help in detecting named entities. The use of Wikipedia involves labeling the articles with tags corresponding to named entity categories (e.g. people, organizations, ..etc.). An entity linking method was utilized from the Wikipedia Miner toolkit so that linked entities were assigned labels that were then inputted into the deep learning model. This was achieved by creating a mapping between mentions of named entities in text and their reference in the knowledge base (e.g. Wikipedia) which are a pre-defined set of labels. The data pipeline of the proposed system starts by creating a vectorized representation of a word by concatenating a pre-trained word embedding output, a trainable character-level encoder, and a set of one-hot vectors from the feature extraction module. A number of bidirectional LSTM layers then compute a hidden word representation. This representation is given to a CRF output layer, which is in charge of predicting a label sequence Y that maximizes the probability $P(Y|X)$, where X is the word vector sequence. Finally, the two deep learning models results in two output sequence Y_{main} for main categories and Y_{sub} for sub-categories of named-entities.

3.3 Learning to Combine Multiple String Similarity Metrics for Effective Toponym Matching

Santos and et al. [10] presents a performance evaluation on the use of different string similarity metrics over toponym matching task. In addition, the paper reports the results of experimentation involves the use of supervised machine learning for combining multiple similarity metrics. The experiments were conducted based on a dataset of five million pairs of toponyms, half of which are matching pairs that correspond to alternative names for the same place. Firstly, the paper presents some characterization statistics for the dataset such as the difference in length between the pairs of the matching and the non-matching class, the different similarity measures range between the pairs, and the number of pairs per country. It was shown that different similarity metrics achieve very similar results in terms of matching quality. The best results in terms of accuracy were achieved using the Jaro-Winkler score followed by the Damerau-Levenshtein metric. The paper also presents experiments on the dataset involving machine learning with a two-fold cross-validation methodology. Experimentation was conducted using 13 different similarity metrics as features for 4 different classification models which are support vector machines, random forests, extremely randomized trees, and gradient boosted trees. These models were used to combine multiple similarity metrics and avoid the manual tuning of similarity thresholds. The decision trees model is proven to achieve good results in matching toponyms with an accuracy of 78.63%.

3.4 Name2Vec: Personal Names Embeddings

The paper [11] proposes creating name-embeddings by employing Doc2Vec methodology by which we can predict names in text that refers to the same entities. The method considers each word as a document and each letter is considered to be a word. This study is motivated by the record linkage research, where name similarity measures are crucial features in determining records that refer to the same entity. Word embedding is the process of mapping vocabulary to vectors of real numbers. The vocabulary is composed of 26 lower-case English letters. After the name-embedding models were generated, cosine similarity scores were calculated between all the name pairs. For test purposes, random name pairs were generated to test the effectiveness of the model to distinguish between matched and unmatched name pairs. There are various parameters were used that can affect the model which are: epochs (number of iterations over corpus during model training), vector_size (the dimensions of the features vector), and window (the maximum distance between the current and predicted word within a document). Different values for each parameter were considered to result in 864 unique combinations of parameters to test. The outcome of the best-performing model was then analyzed to evaluate the performance and the consistency. Moreover, the effect of the parameters was investigated to show the effect of each of the parameters on the quality of the model.

3.5 Private Record Linkage: Comparison of Selected Techniques for Name Matching

P. Grzebala and M. Cheatham [12] present an extensive systematic analysis that examines common flaws to name-based data entry that includes typographical errors, optical character recognition errors, and phonetic errors. The results are compared to typical name-matching metrics on unprotected data, and the article examines the applicability, accuracy, and speed of three different basic methods to this challenge (along with numerous variants). While these methods are not novel, this study examines a variety of datasets that feature systematically induced defects typical to name-based data input. the main contributions of this paper can be summarized as follows:

- A nuanced analysis of the effectiveness of various distinct name-based similarity metrics using a sophisticated name matching benchmark creation tool. Moreover, the paper examines the impact of the threshold value used to each of the measures in the research, which takes into account a variety of real-life mimicking sources of mistake.
- The accuracy of privacy-preserving similarity metrics is compared to that of regular string metrics on unprotected data. This is done to determine the compromise of accuracy to support data privacy.
- The privacy-preserving similarity metrics' computational performance is also compared to that of regular string metrics.

The paper firstly discusses some of the challenges that make record linkage of names a difficult task. These challenges involve name spellings that can be malformed in a variety of ways, including punctuation, abbreviation, pronunciation, spelling, writing order, usage of prefixes, typos, and optical recognition mistakes, to mention a few. Furthermore, privacy considerations have made it difficult to locate publicly available data that may be used as a benchmark, particularly a collection of names that truly reflects global name distribution rather than being US-centric. The absence of appropriate benchmarks was a significant challenge during the research, prompting the usage of a recently released name matching benchmark creation method. The benchmark datasets were created using an advanced personal data generation tool called "GeCo". The tool has two main functionalities: data generation and data corruption. The paper introduces six different data corruption techniques as follows:

1. Missing values.
2. Character edits (random character of string edit operations(e.g., insertion, deletion, substitution, and transposition)).
3. Keyboard edits (simulating human typing mistakes)
4. Optical character recognition (OCR) errors (simulate OCR software mistakes).
5. Phonetic edits (replace a substring by its phonetically matching variation).
6. Categorical value swapping (replaces an attribute value with one of its possible variations).

The uncorrupted dataset was cross joined with each of the corrupted datasets of each of the string metrics to evaluate the performance of the string metrics. The metrics considered were the Soundex algorithm and four variations of the Q-gram techniques to be compared with Jaro and a normalized version of Levenshtein. Joining datasets based on Soundex encodings was not a viable option because it failed to find a correct match for more than half of the records when the name is a corrupted record contained one error, and nearly 70% of the records when the name is a corrupted record contained two errors in a single name. For linking datasets based on encrypted names, Q-grams-based algorithms appear to be a promising choice. While their precision is slightly lower than that of unencrypted data metrics like Jaro or Levenshtein, this can be easily remedied by altering the threshold value that determines when two q-grams are likely to correspond to the same name.

3.6 Conclusion

In the previously mentioned papers, different ways were proposed for detecting and linking records in textual data. Some rely on dictionary-based NER which can be very useful for structured data. However, problems usually arise when entities do not exist in the dictionary in use. Other research papers leverage the power of machine learning in solving such problems. Deep learning models were used to detect and link entities using different similarity metrics to extract features. On the other hand, different similarity measures were analyzed to find the advantages and disadvantages of using certain measures for certain use cases. In this project, we combine different similarity measures for different use cases such as detecting personal names with typographical errors and recognizing co-referent name variants. Furthermore, we utilize machine learning models in learning similarities between co-referent names for record linkage. In the next chapter, the method and implementation steps that address the research question for this project are presented.

Chapter 4

Method

With the large amounts of textual data that humans produce every day, there should be a systematic way to analyze this data while preserving the privacy of the subjects the data might refer to. We aim to create a pipeline that receives the unstructured text document as an input and outputs the same text where different personal names are linked for anonymization. Anonymizing names risk the utility of the data for secondary analysis. The evaluation of the utility of the data can be dependent on the intended use of this data. Since the intended use of the data can widely vary, we think of the utility of the data in terms of information loss. We studied the possible information loss that can result from anonymizing names in textual data and attempt to keep this information after anonymization with minimal risk of re-identification. In this project, we tackle the problem of information loss due to name variants. Our objective is to keep linkage between different subjects to enable recognizing names that belong to the same subject after anonymization. Therefore, we minimize the information loss resulting from anonymization. Figure 4.1 outlines the procedures taken to achieve this goal.

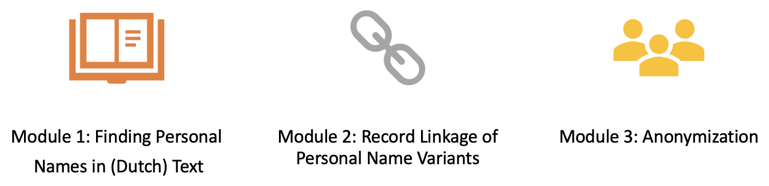

Module 1: Finding Personal Names in (Dutch) Text
Module 2: Record Linkage of Personal Name Variants
Module 3: Anonymization

Figure 4.1: Summary of defined methodology for the proposed approach

The method consists of three modules, where each separate performs a specific task as a part of the whole pipeline. Briefly, in module 1, we explore efficient ways to recognize individuals' names in unstructured text. In module 2, we use similarity measures to recognize possible name variants. Lastly, in module 3, we anonymize all names while storing links between them.

4.1 Module 1: Finding Personal Names in unstructured Dutch Text

In most western countries, personal names usually consist of a given name, an optional middle name, and a surname or a family name. However, this format is not always followed especially in unstructured text. 'Elizabeth J. Smith', 'Liz Smith', and 'E. J. Smith' can refer to the same individual. Furthermore, people often use (or are given) nicknames in informal unstructured text. These forms are usually a short form of their given names (like 'Rick' for 'Richard' or 'Tina' for

'Christina'). In Dutch, typical forms for nicknames are to truncate parts of the name and add a (t)je, k(e), p, or s to it (like 'Anje' or 'Anke' for 'Anna' and 'Betje' or 'Bep' for 'Elisabeth') [13]. Therefore, recognizing all these different types of variations is a challenging task. The following cases are examples of the inconsistent syntax that can be found in unstructured text, e.g., user-generated text.

- Spelling variations (e.g., 'Eric' and 'Erik') due to typographical errors that don't affect phonetical structure of a name;
- Phonetic variations (e.g., 'Sinclair' and 'St. Clair')
- Alternative names, such as nicknames, married names or other deliperate name changes;
- Out of order components (e.g., 'Diaz, Carlos Alfonzo' - 'Carlos Alfonzo Diaz'); and
- Initials (e.g., 'J. E. Smith' - 'James Earl Smith')

Named Entity Recognition (NER) is the task of recognizing and classifying named entities in the text according to a pre-defined set of entity types. The term was coined to the MUC-6 conference [14]. The objective is to detect general entity types such as persons, organization, locations, etc. in any given text. In this module, we focus on personal names detection in unstructured text and the followed approach in addressing particular challenges in finding person names entities in Dutch text. The following section outlines the initial approach to find names in the text. It also provides an overview of the possible risks of using such a method and its impact on the quality of the results.

4.1.1 Named Entity Recognition Model and List-based Hybrid Approach for Names Detection

To detect names in unstructured text, we started by exploring two different methods for NER. The first method was the list-based approach, whereas the second method was NER systems. We collected a data set of 3,302 Dutch given names and 88,223 surnames from Marteens institute[1] [15] Dutch first names and surnames database. Using this dataset, our initial approach was to use a list-based approach. This approach was applied by comparing the whole text to the Dutch names dataset. Firstly, NER models were used to tokenize the text into words. Those tokens were equated to the list of Dutch given names and surnames. Although this method is easy to implement and somehow efficient in detecting a wide range of names, this efficiency does not hold when faced with unstructured text where names contain initials or misspellings, or variations.

Consequently, we decided to follow a hybrid approach that incorporates NER systems in detecting personal names in text. NER systems use linguistic grammar-based techniques as well as machine learning models. They use pre-trained models to locate and classify named entities in textual data. The precision of these tools can differ depending on the format and structure of the text. In our application, we used Stanza [2] which is a Python open-source library that contains various NER tools. Stanza has a NER module that recognizes mention spans of a particular entity type (e.g., person or organization) in input sentences and it supports 8 languages including Dutch. Using Stanza's module, we were able to extract a list of names from the text. Finally, names recognized by Stanza and those detected by being compared to the names list were compiled into one list.

The combination of these methods does not ensure detecting all personal names especially in the free-form text where names can be misspelled or mistyped. To minimize some of those risks, we modified our previous method to analyze the text and enable us to detect misspelled names and typographical errors in the text in the names detecting phase. In the next section, we highlight the steps followed to minimize some of the risks of the initial method.

4.1.2 Detecting Personal Names with Typographical Errors

In the context of anonymization, detecting typographical errors helps us in linking mistyped personal name records that refer to the same individuals. The objective of adding this step is to

[1]https://www.meertens.knaw.nl/cms/nl/collecties/databanken

[2]https://stanfordnlp.github.io/stanza

find personal names with typographical errors in the text and correct them before resuming further. After the text was processed, the list of detected names, using the names list and NER systems, was compiled. The rest of the text was then inspected for possible names with typographical errors. Stanza's part-of-speech (POS) module was used to retrieve labels for all the words in the text. These labels are mainly the universal POS (UPOS) tags that present the morphological features of each word. The idea of using these tags is to identify nouns and proper nouns in the text that can refer to a mistyped name that was not detected as a name in the text. Despite that, a mistyped name part-of-speech can be neither a noun nor a proper noun in some specific cases, most of the misspelled names fall in this category. Moreover, the choice of specific tags (i.e nouns and proper nouns) enables us to reduce the number of words to be compared to the detected names list; therefore, the number of false positives is reduced. All the extracted nouns and proper nouns are then compared to the list of names that were detected in the previous step using Levenshtein edit-based distance. Levenshtein edit-based distance measures the number of edit operations between two strings (i.e. deletion, insertion, transposition, and substitution). Edit-based distance measures are well-fitted for detecting words with typographical errors that result from minor changes in a string. When the Levenshtein distance between a detected name and another word in the text is less than a specific threshold, this indicates a potential typographical error.

Note that, this method relies on the assumption that each personal name in the text is mentioned once in the correct form. In case a name is consistently mistyped throughout the whole text, extracted nouns and proper nouns from the text can be all compared to the whole list of names collected for names detection. However, this solution can result in many false positives, where nouns and proper nouns can be similar to more than one name in the names list. For example, the word 'Algred' was extracted from the text as a potentially mistyped name, and compared to our names list, it would be matched with 'Albert', 'Alfred', or 'Aldret'. Although we can still find similar names in the text, finding multiple similar names can be very rare. On the contrary, when a name is compared to the collected list of most possible names, one name can be matched with multiple names at a time. Consequently, limiting our sample to names previously detected in the text can improve the accuracy of the results.

After detecting all the names including the misspelled names in the text, we aim to identify all records in the text that refer to the same individuals. In the following module, we explore record linkage research in identifying co-referent and non-co-referent names in Dutch text.

4.1.3 Example

Input Text "**Anna** houdt van Engelse en wiskunde. Haar wiskundeleraar heet **Michial**. **Micheal** is een zeer goede leraar. Haar lerares Engels heet **Mieke**. **Miek** geeft **Antje** graag veel opdrachten. "

In this step, we extract and compile all the names detected by NER in combination with the name's list. Three individuals were mentioned in the above text i.e 'Anna', 'Michael', and 'Mieke'. Two of the names (i.e. Anna, Micheal) were detected using NER, whereas one name was detected using the names list. Table 4.1 shows a list of all detected using the NER and list-based combination method. There are other mentions of the same people which different spelling mistakes and variations (e.g., 'Anna' is referred to as 'Antje', Micheal is misspelled as 'Michial', and 'Mieke' is misspelled as 'Miek'). As shown in this example, the NER tool does not always detect the complete list of names in the text. On the flip side, completely relying on the list-based approach would limit the results to those names that exist in the list. Therefore, combining a NER tool with a list of names can be essential to ensure better accuracy in finding as many personal names in the text as possible.

Table 4.1: Output of first step of module 1: Finding Personal Names

Name	**Source**
Anna	NER
Micheal	NER
Mieke	Name's List
Antje	NER

Firstly, we extract all the nouns and proper nouns from the text excluding all the names that were already detected as shown in table 4.2.

Table 4.2: Output of second step of module 1: Detecting Misspelled Names and Typographical Errors

Words	**Tags**
wiskunde	noun
Michail	proper noun
leraar	noun
lerares	noun
Engels	proper noun
Miek	proper noun
opdrachten	noun

All the extracted names in table 4.2 are compared to the ones detected in the text in table 4.1 using Levenshtein distance. After we compute all the Levenshtein similarities with a threshold equals 2. We finally yield the following table 4.3 with all the names including the mistyped names.

Table 4.3: Final Output of second step of module 1

Mistyped Name	**Original Name**	**Levenshtein Distance**
Michail	Michael	1
Mieke	Miek	1

4.2 Module 2: Record Linkage of Personal Name Variants

When names are the only unifying data point, correctly matching similar names becomes more crucial; however, the variability and sophistication of names make name matching a particularly difficult challenge. Missed matches can be caused by nicknames, translation mistakes, multiple spellings of the same word, and other factors. Although there are various search methods available, name search necessitates a radically different approach than document search. To link those name forms in unstructured text, we need to find ways to compare them and relate them to each other.

For example, if we encountered the name 'Alexander' in a given text then the name 'Alex', we can assume that those two names refer to the same individual. However, matching the two names requires computing an appropriate similarity measure that would capture the interchangeability of the two names. In this section, we discuss the approach followed to link different name variants in unstructured text. Firstly, we started our analysis by exploring similarity measures that can assist in solving the addressed problem.

4.2.1 Computing Similarities

Our initial approach was to compute a similarity distance between the names in the list of names derived from the document. At first, we computed the Levenshtein distance between Dutch names in our collected names list and names detected from the text. Table 4.4 shows an example of computations between names from the collected names list and names recognized from a sample unstructured text.

Table 4.4: Sample Levenshtein distance computations

Names from names list	Names from text	Levenshtein distance
Roger	Rogar	1
Daniel	Daniil	1
Daniela	Daniil	2
Andrea	Andrey	1
Audrey	Andrey	1

These pre-computed distances can map names in the text to one or multiple similar names in the collected names list. However, the following issues emerged as a result of this strategy. The first concern was that some of the names did not have any reference in the names list. The second concern was that, although our names list does not have any identical names, knowing the similarity gap between a name in the text after anonymization and other names from the names list will facilitate re-identifying the name in the document. The pre-computed distances between the anonymized names and the list of stored names will reveal details such as the number of letters in the anonymized name and its variations.

To avoid the risk of re-identification by storing the Levenshtein distance between names in text and the names list, we decided to compute similarities only among names inside the text. This will allow exploring the closeness of the names to each other without comparing them to any external data source. On the other hand, it was found that static similarity measures cannot accommodate variability that may occur due to data entry errors, or multiple ways of expressing the same word or individual such as misspellings, abbreviations, nicknames, or alternate spellings. To capture the variability of name forms in unstructured text, we considered three types of similarity measures:

1. **Type 1** Edit-based distances that capture the spelling differences between different occurrences of a name in text. For our research purposes, we considered the following 2 edit-based distances:

   - **Levenshtein Distance** The Levenshtein distance is the minimum edit operations to convert one string to another discussed in chapter 2.

   Benefits Approximate string matching aims to find matches for short strings in a large number of longer texts in situations where only minor variations are predicted. In other words, Levenshtein distance is mostly used to check misspellings. This has a wide range of applications, for instance, spell checkers, correction systems for optical character recognition. In our project, Levenshtein distance can represent similarity between misspelled names or typographical errors. Table 4.5 shows some example of the Levenshtein ratio between different co-referent name variants. The results show that the Levenshtein similarity can accurately match spelling variations of names (like ‘Robbert’ and ‘Robert’).

Table 4.5: Levenshtien distance between different name variations types

Name 1	Name 2	Ratio	Name Variant type
‘Michael’	‘Michiel’	1	Spelling variation
‘Robbert’	‘Robert’	1	Spelling variation
‘Marie’	‘Rie’	2	Alternative naming
‘Antonia’	‘Tonie’	3	Alternative naming
‘Francisca’	‘Ciskca’	5	Alternative naming
‘Robert’	‘Bob’	4	Alternative naming
‘Anna’	‘Netje’	5	Alternative naming

- **Jaro-Winkler Distance**, is another edit-based distance, derived from the Jaro measure which is the weighted sum of the percentage of matched characters from each string and increased strings. Winkler increased this measure for matching initial characters, then re-scaled it by piecewise function, whose interval and weights depend on the type of string. [16]. In other words, Winkler’s measure gives more importance to words with identical prefixes.

  Benefits The Jaro-Winkler similarity measure gives more importance to names that have the same prefix. This was found effective in matching some name forms since fewer errors are typically found at the beginning of names [17]. The Winkler measure can perform well with parsed names that are divided into given- or surnames. Table 4.6 shows some examples of a computed Jaro-Winkler distance between different co-referent name pairs. The results show that the Winkler measure is useful for matching alternative naming (i.e. nicknames), especially if they share similar prefixes.

Table 4.6: Jaro-Winkler distance between different name variations types

Name 1	Name 2	JW Distance	Name Variant type
‘Frederik’	‘Fred’	0.90	Alternative naming
‘Erik’	‘Eric’	0.88	Spelling variation
‘Robert’	‘Bob’	0.67	Alternative naming
‘Antonia’	‘Tonie’	0.71	Alternative naming
‘Francisca’	‘Ciskca’	0.39	Alternative naming
‘Marie’	‘Rie’	0.0	Alternative naming

Disadvantages Although these comparisons are quick, those distances do not catch linguistic complexity. All edits are assigned the same amount of weight. Thus, changing “c” to “p” has the same weight as changing “c” to “k,” although in English, the latter substitution can more specifically mean a similar word, as in ‘Catherine’ vs. ‘Katherine.’. Moreover, those measures ignore the similarity between strings given that are far apart in terms of the difference in size(e.g., ‘Alexander’ - ‘Alex’). They can also be sensitive to string length. For example, the Levenshtein distance between ’Anna’ and ’Pete’ is 4 which is the same as the Levenshtein distance between ‘Bernardina’ and ‘Bernardientje’. Although the ‘Bernardina’/ ‘Bernardientje’ pair is more similar than the ‘Anna’/‘Pete’ pair, the Levenshtein distance is the same. Moreover, it does not take into account the semantic meaning of the strings in comparison.

Use Cases Edit-based distances can directly capture resemblance between words such as ’Michael’ and ’Michiel’. This means that can detect minor differences between strings that can happen due to typographical errors or misspellings.

2. **Type 2** Token-based distances that compare two strings by looking at units (Tokens) of strings (e.g., n-grams).

   - **Jaccard Similarity Coefficient** is one of the simplest token-based distances which is defined as the magnitude of the intersection (common tokens)divided by the magnitude of the union (unique tokens) of two sets [1]. Those sets represent multisets of tokens of the two strings to be compared [4].

     Benefits Jaccard similarity simply counts the number of members which are shared in both sets. This can be helpful for co-referent name pairs that share the same root. Table 4.7 shows that Jaccard similarity results can be similar to edit-based distances. It gives high scores for co-referent names that have the same length and shares most of their characters.

Table 4.7: Jaccard similarity between different name variations types

Name 1	Name 2	Jaccard Index	Name Variant type
‘Robbert’	‘Robert’	1.0	Spelling variation
‘Frederik’	‘Fred’	0.66	Alternative naming
‘Erik’	‘Eric’	0.60	Spelling variation
‘Francisca’	‘Ciskca’	0.44	Alternative naming
‘Antonia’	‘Tonie’	0.37	Alternative naming
‘Robert’	‘Bob’	0.28	Alternative naming
‘Anna’	‘Netje’	0.0	Alternative naming

   - **Cosine Similarity** Cosine similarity approximates how similar two words vectors are by computing the cosine of the angle between two vectors to quantify how similar two words are. The smaller the angle, the higher the cosine similarity. These vectors can be term-frequency vectors created by computing the frequencies of the string tokens [4].

     Benefits Cosine similarity is a widely used measure in the information retrieval community since it excels in representing similarities between names that are far apart by edit-based distances. In this regard, results from table 4.8 show that the cosine similarity gives higher scores for co-referent names than the Jaccard similarity index.

Table 4.8: Cosine simialrity between different name variations types

Name 1	Name 2	Cosine Similarity	Name Variant Type
‘Robbert’	‘Robert’	0.95	Spelling variation
‘Frederik’	‘Fred’	0.90	Alternative naming
‘Erik’	‘Eric’	0.75	Spelling variation
‘Francisca’	‘Ciskca’	0.68	Alternative naming
‘Antonia’	‘Tonie’	0.60	Alternative naming
‘Robert’	‘Bob’	0.47	Alternative naming
‘Anna’	‘Netje’	0.0	Alternative naming

Disadvantages Token-based distances can be more efficient for phrases from multiple words rather than single words or short phrases from several words.

Use Cases Those metrics are advantageous because even if the two similar strings are far apart by the edit-based distances, chances are they may still be oriented closer together. Furthermore, Cosine Similarity gives the length of the strings less importance. In other words, uneven lengths of strings do not affect the accuracy of the cosine similarity measure between the two strings. For instance, the similarity between 'Alex' and 'Alexander' is better represented using a token-based distance since they have a significant intersection even though they vary significantly in length.

3. **Type 3** Phonetics-based algorithms investigate the similarities between strings depending on how close they sound.

   - **Hamming Distance** is an edit-based distance between two strings depending on how they closely sound. However, hamming distance is only measured between two strings of equal length. To take into consideration the content of the words regardless of the order and size, we obtain the factor of distance given by the following formula [18]:

$$Distancefactor = \frac{MAX(len(str1), len(str2)) - hammingDistance}{MAX(len(str1), len(str2))}$$

Benefits This distance measure is more descriptive than other lexical edit-based distance such as Levenshtein, which gives the same weight to all edit operations. For example, the Levenshtein distance between 'Sophia' and 'Sofie' is 3, whereas the hamming distance between the same pair is 1. This is because the hamming distance account for interchangeable characters due to their phonetical similarity (i.e. 'ph' and 'f'). Table 4.9 shows hamming distance between different co-referent name pairs.. Results show that Hamming distance match spelling variations more accurately than other edit-based distances such as Levenshtein distance; however, it fails to match alternate names that are significantly different than the original name.

Table 4.9: Phonetics edit distance between different name variations types

Name 1	Name 2	Hamming Distance Factor	Name Variant type
'Michael'	'Michiel'	1.0	Spelling variation
'Robbert'	'Robert'	1.0	Spelling variation
'Marie'	'Rie'	0.75	Alternative naming
'Antonia'	'Tonie'	0.47	Alternative naming
'Francisca'	'Ciskca'	0.44	Alternative naming
'Robert'	'Bob'	0.0	Alternative naming
'Anna'	'Netje'	0.2	Alternative naming

Disadvantages These algorithms compare words based on how they sound. However, these methods do not take into account the semantic meaning of words. In our case, we need to match variations of names. Those variations also include nicknames that are phonetically very different than the name (e.g., Robert - Bob).

Use Cases Phonetics-based measures can be very effective in detecting errors. Moreover, it can recognize similar names with different spellings (e.g., Eric - Erik).

To clarify the differences between those measures, let's consider the following pairs for comparisons: pair (1) is 'Alex' and 'Alexander', pair (2) 'Antonia' and 'Tonny', pair (3) 'Robert' and 'Bob', pair (4) 'Robert' and 'Albert', and pair (5) 'Robert' and 'Rublev'. Table 4.10 shows the values for different similarity measures comparing the aforementioned pairs of names. The three pairs reflect some variations of names that might refer to the same individual.

Table 4.10: Similarity measures overview

Similarity Measures	**Pair 1**	**Pair 2**	**Pair 3**	**Pair 4**	**Pair 5**
Levenshtein Ratio	0.62	0.29	0.44	0.66	0.5
Jaro-Winkler	0.89	0.53	0.67	0.78	0.70
Jaccard Index	0.44	0.57	0.28	0.50	0.33
Cosine Similariy	0.75	0.57	0.47	0.67	0.50
Hamming Distance Factor	0.44	0.42	0.0	0.66	0.50

The results (see Table 4.10) show that different measures can detect similarities between different forms of text. For example, the similarity between pairs 1,3 is best captured using edit-based distances, where token-based similarity is better suited to represent the similarity between pair 2. On the other hand, the dissimilarity between pair 4 and pair 5 is best represented by the Jaccard similarity. This evaluation merely depends on the type of application that the analysis aims for. For this project, our objective is to match all these forms of names in unstructured text.

Static approaches, such as cosine and Levenshtein similarities, impose a predefined penalty for each unit (character or multi-character token) of difference between two strings. Considering the variability of personal names in unstructured text, relying on a single static similarity measure has proven ineffective. Therefore, we decided to explore learnable similarity metrics which can be trained on classified examples of co-referent and non-co-referent names.

4.2.2 Supervised Classification for Record Linkage

Learnable similarity metrics use machine learning classifiers trained on features extracted using labeled pairwise string comparisons (e.g., static similarity scores). This can be achieved using classification similarity learning which is a supervised machine learning task that targets learning a similarity function to decide whether a new pair of unlabeled objects belongs to the same class. By considering pairs of strings labeled as similar or dissimilar as training instances, we face a binary classification problem that can be solved through a classifier that decides whether a pair of strings is similar. The classifier is trained on different static similarity metrics as a set of features between each pair of strings. This kind of input data injects the classifier with additional knowledge because the use of a distance measure is an implicit match between the characteristics of two objects.

Model Training Dataset

Firstly, we searched for an appropriate dataset by which we can train and test a classification model on co-referent names and non-co-referent names. The dataset[3] was web-scrapped from a Wikipedia page [13] for Dutch names hypocoristic forms using Beautiful Soup. Beautiful Soup [4] is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping. The scrapped dataset contains 150 Dutch names (76 females, 74 males) with 1757 different forms for all the names. However, this dataset was still unusable for classification since it only contains co-referent names. This problem can be referred to as the problem of classification class imbalance. To overcome this addressed problem, we manually created 1820 non-co-referent pairs of names to balance the distribution of the classes for binary classification. Figure 4.2 shows the difference in the number of characters between co-referent and non-co-referent pairs. It indicates that there is no specific correlation between the difference in the number of characters and if the pair is a match or not.

[3]https://en.wiktionary.org/wiki/Appendix:Dutch_diminutives_of_given_names
[4]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

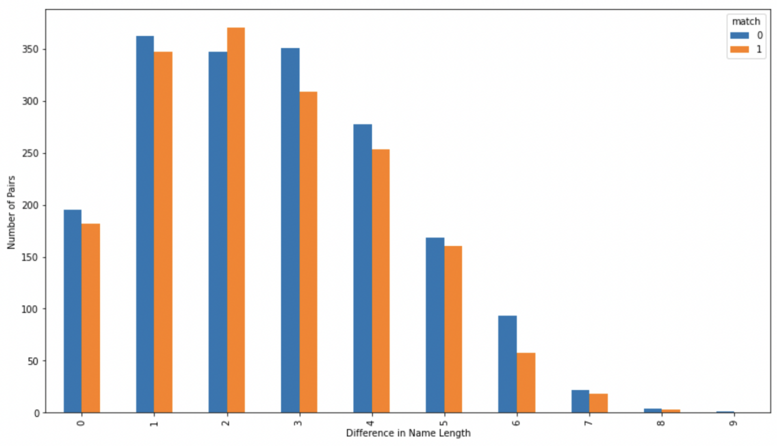


Figure 4.2: Distribution for the difference in the number of characters per matching/non-matching pair.

As part of our characterization of the dataset, we analyzed the distribution of the name pairs according to different similarity measures. This can be related to the difficulty in performing automated classification. Figure 4.3 shows the results of the Levenshtein distance scores between pairs which shows that non-matching pairs have higher values. Figure 4.4 shows the results of the cosine similarity scores that are better representative of the co-referent pairs as values increase for the co-referent pairs. This indicates that token-based similarities such as Cosine similarity are better suited for this dataset, considering the clear cut between values for matching and non-matching name pairs.

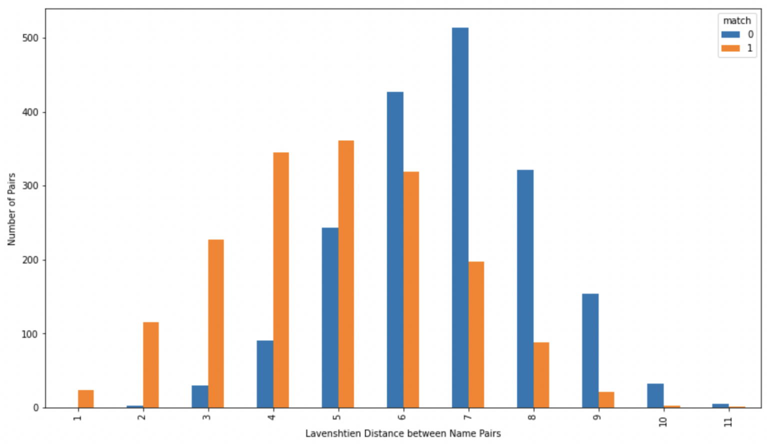


Figure 4.3: Distribution for the number of pairs according to the Damerau-Levenshtein similarity between the strings

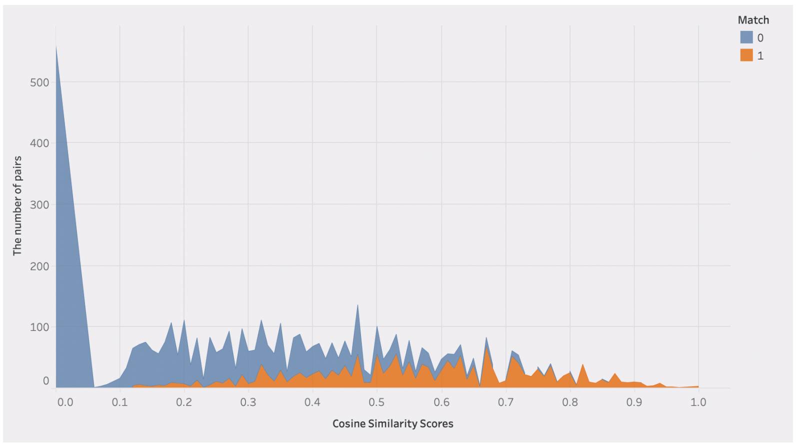


Figure 4.4: Distribution for the number of pairs according to the Cosine similarity between the strings

Classification Models

In a supervised learning setting, the objective to decide the target outcome, whether two names are co-referent or not. Under this scope, the Wikipedia dataset [13] was used to train and test the classification model. The dataset contains a target class called "Match" where data is labeled with either 1 or 0.

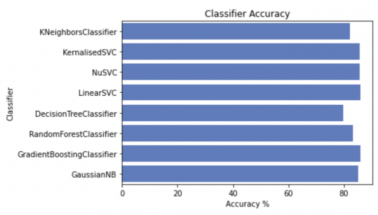
- 1: co-referent pair
- 0: non-co-referent pair

The objective of creating a classification model is to be able to combine multiple similarity measures for better performance than single static measures. Five semantic similarity measures were used as training features. Two of those measures are token-based, 3 are edit-based distances, and *FuzzyWuzzy*[5] score. Firstly, all the similarity scores were normalized to range between (0 and 1), 1 being the most similar and 0 being the least similar pair. Secondly, the dataset samples were split into a train and test set (80% training, 20% testing). Finally, several classification models were trained and tested to compare their results.

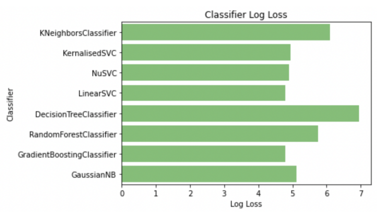
Models were evaluated using three metrics: accuracy score, log loss, and the area under the receiver operating characteristic curve (ROC AUC). Model accuracy is defined as the number of classifications a model correctly predicts divided by the total number of predictions made. Accuracy is a crucial component in assessing the model; however, it captures one aspect of the model evaluation. Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in the case of binary classification). The more the predicted probability diverges from the actual value, the higher the log-loss value is. Another important metric that is indicative of the quality of the model is the ROC - AUC. The AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve, while AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes [19]. The higher the AUC, the better is the model in distinguishing co-referent and non-co-referent pairs.

Figure 4.5 shows an overview of all the models' performance in terms of accuracy, log-loss, AUC score. The linearSVC (linear Support Vector Classification) and the gradient boosting classifier were ones the best performing models in terms of accuracy with 86%, the log-loss is 4.7, AUC score is 0.9. However, the gradient boosting classifier was found to have significantly (approximately 8 times) higher computational speed than the linear support vector classifier in both training and testing. Therefore, the gradient boosting classifier was chosen for our classification problem. Table 4.6 provides a summary for all the models, that were trained and tested on our dataset, along with their accuracy, log-loss, and AUC score after testing.

[5]https://pypi.org/project/fuzzywuzzy/

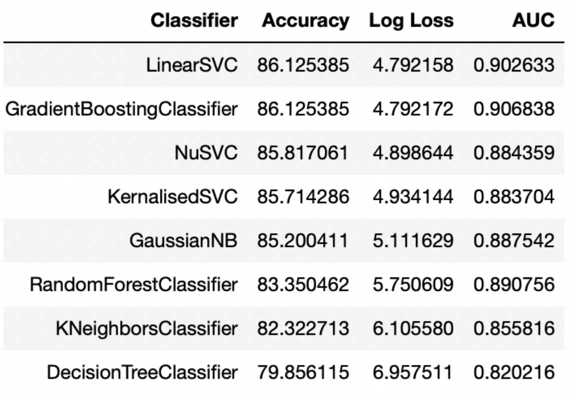


(a) Classifiers accuracy scores



(b) Classifiers log loss scores

Figure 4.5: Model Comparison

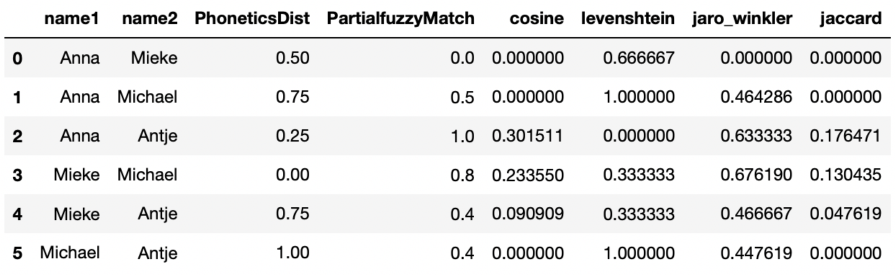


Classifier	Accuracy	Log Loss	AUC
LinearSVC	86.125385	4.792158	0.902633
GradientBoostingClassifier	86.125385	4.792172	0.906838
NuSVC	85.817061	4.898644	0.884359
KernalisedSVC	85.714286	4.934144	0.883704
GaussianNB	85.200411	5.111629	0.887542
RandomForestClassifier	83.350462	5.750609	0.890756
KNeighborsClassifier	82.322713	6.105580	0.855816
DecisionTreeClassifier	79.856115	6.957511	0.820216

Figure 4.6: Overview of the models performance in terms of accuracy, log loss and AUC

4.2.3 Example

Using the same example from the first module, The input for this module would be the pairings of all the names detected in Module 1. For each pair, we compute all the similarity metrics needed as features for the classification model as shown in figure 4.7.



	name1	name2	PhoneticsDist	PartialfuzzyMatch	cosine	levenshtein	jaro_winkler	jaccard
0	Anna	Mieke	0.50	0.0	0.000000	0.666667	0.000000	0.000000
1	Anna	Michael	0.75	0.5	0.000000	1.000000	0.464286	0.000000
2	Anna	Antje	0.25	1.0	0.301511	0.000000	0.633333	0.176471
3	Mieke	Michael	0.00	0.8	0.233550	0.333333	0.676190	0.130435
4	Mieke	Antje	0.75	0.4	0.090909	0.333333	0.466667	0.047619
5	Michael	Antje	1.00	0.4	0.000000	1.000000	0.447619	0.000000

Figure 4.7: Output of the first step of module 2: Computing Similarities

In the second step of this module, we take all the pairs and the computed similarity metrics and use the classification model to label co-referent and non-co-referent pairs. Table 4.11 shows the output of the classification model. Notice that, 'Mieke' and 'Michael' are classified as a co-referent pair. However, this can not be confirmed or rejected until further analysis is conducted on the text to see if the two names were exchangeable.

Table 4.11: Output of the second step of module 2: Supervised Classification For Record Linkage

Name1	Name2	Label
Anna	Mieke	non co-referent pair
Anna	Michael	non co-referent pair
Anna	Antje	co-referent pair
Mieke	Michael	co-referent pair
Mieke	Antje	non co-referent pair
Michael	Antje	non co-referent pair

4.3 Module 3: Anonymization

After linking names inside the text using our classification model that classifies co-referent and non-co-referent pairs of names. Personal names should be anonymized while storing the model output to keep a reference for names that possibly refer to the same individual.

4.3.1 Storing Records Links Between Anonymised Names

To store links between names in the text, one option was to cluster all co-referent pairs that might refer to the same individual. However, the relationship between the co-referent pairs was found to be non-transitive. This means that two non-co-referent names can have the same co-referent name. Therefore, if {name1, name2} is a co-referent pair as well as {name2, name4,} this doesn't imply that {name1, name4} is a co-referent pair. Therefore, it was not possible to partition names into groups of co-referent pairs. For illustration, let's take an example of real names where the grouping technique fails to represent the relation between names. The name 'Bert' can be a co-referent name of both 'Robert' and 'Albert'. However, 'Robert' and 'Albert' aren't a co-referent pair so placing them in the same group can be misleading due to the non-transitive relationship between the two co-referent pairs, i.e. {'Robert', 'Bert'} and {'Albert', 'Bert'}. Consequently, each of the name records and their variant was stored with a unique identifier. The output of the classification model between pairs was stored between each of those identifiers.

On the other hand, *FuzzyWuzzy* [6] is used to compute the percentage of the match between each of the name pairs to be stored. *FuzzyWuzzy* is a python library that has a powerful *partial_ratio()* function that allows us to perform substring matching. The *FuzzyWuzzy* partial ratio works by taking the shortest string and matching it with all substrings that are of the same length. Computing this ratio between all pair matches classified by the classification model will enable us to store more linkage information among names in text after anonymization. Table 4.12 shows results of co-referent names extracted from a sample text after applying classification and the *FuzzyWuzzy* similarity ratio. The first two columns of table 4.12 show several name pairs that were labeled as 1, i.e. co-referent names, whereas the *FuzzyWuzzy* ratio column represent the similarity percentage between the two names. Note that, names that are substrings of other names are shown a definite match, i.e. partial_ratio = 100, such as 'Alex' and 'Alexander'. On the other hand, this similarity ratio can be considered as an indicator of the type of variant those two names are, i.e. nicknames, misspellings, ..etc.

[6]https://pypi.org/project/fuzzywuzzy/

Table 4.12: Records linkage before anonymization

Name 1	**Name 2**	**Classification label**	***FuzzyWuzzy* Ratio**
Fabio	Fab	1	100
Daniil	Daniel	1	83
Rublev	Rob	1	67
Robert	Rob	1	100
Alexander	Alex	1	100
Alex	Alber	1	75
Albert	Alber	1	100

The next step is to encrypt all names in the text by replacing every unique names in the given text by a unique code as well as storing the record linkage information computed in the previous steps.

4.3.2 Encryption of Names in Text

In order to anonymize names in text, the *inforcehub* [7] python package was used. The package contains an *Anonymize* module that creates a one-way encrypts (hashes) of the data so that it cannot be re-retrieved. The module contains a class that transforms a Pandas[8] data frame into an anonymized data frame using the python hmac[9] package for encryption. When instantiating an object of this class, the salt init attribute can be specified which enables encryption results to be reproduced. If none is supplied, a randomized password is created instead for security. For this project, we specify the salt init attribute as none to create a unique randomized hash for each of the names recognized in the text. Names are then replaced with these hashes in the text while storing record linkage information between each pair of names.

4.3.3 Example

Using the same example from previous sections, the following paragraphs show the input and output text before and after anonymization. Note that, misspelled names are encrypted with the same identifiers as the original names. Whereas name variants are stored with unique identifiers. Furthermore, record links are stored in a separate table with all the information about the co-referent and non-co-referent pairs as shown in table 4.13.

Input Text “**Anna** houdt van Engelse en wiskunde. Haar wiskundeleraar heet **Michial**. **Micheal** is een zeer goede leraar. Haar lerares Engels heet **Mieke**. **Miek** geeft **Antje** graag veel opdrachten. ”

Output Text “ **17fdf174a857eddf74e2423d09b325af** houdt van Engelse en wiskunde. Haar wiskundeleraar heet **919f2f0becd467856c8f7932b6149823**. **919f2f0becd467856c8f7932b6149823** is een zeer goede leraar. Haar lerares Engels heet **597e1fa1460c793615d4fdb1f3378cbc**. **597e1fa1460c793615d4fdb1f3378cbc** geeft **e07c3d132fe8f9774244e8406bc0480c** graag veel opdrachten. ”

[7]https://inforcehub. readthedocs.io/en/latest/modules/anon.html
[8]https://pandas.pydata.org
[9]https://pkg.go.dev/crypto/hmac

Table 4.13: Records linkage after anonymization

Name 1	**Name 2**	**Classification label**	***FuzzyWuzzy* Ratio**
17fdf174a857eddf74e2423d09b325af	e07c3d132fe8f9774244e8406bc0480c	0	0
17fdf174a857eddf74e2423d09b325af	919f2f0becd467856c8f7932b6149823	0	25
17fdf174a857eddf74e2423d09b325af	e07c3d132fe8f9774244e8406bc0480c	1	50
597e1fa1460c793615d4fdb1f3378cbc	919f2f0becd467856c8f7932b6149823	1	40
597e1fa1460c793615d4fdb1f3378cbc	e07c3d132fe8f9774244e8406bc0480c	0	20
919f2f0becd467856c8f7932b6149823	e07c3d132fe8f9774244e8406bc0480c	0	20

4.4 Conclusion and Future Work

In this chapter, we have addressed some of the challenges faced to detect different variations of co-referent personal names in the unstructured text (i.e. user-generated text). We discussed the mechanism and tools used for personal name detection in the unstructured text as well as typographical errors. Next, research was conducted on different similarity measures that can represent similarities between co-referent name pairs in a given text. We explored edit-based, token-based, and phonetics-based similarities. Five algorithms were introduced: two of which were edit-based, two were token-based, and one is phonetics-based. It was found that edit-based similarities and phonetics-based similarities are most useful for spelling checks and minor typographical errors. Token-based distances are advantageous in accurately representing similarities between strings regardless of the difference in length. This is particularly beneficial in matching alternative naming (i.e. nicknames) since they might differ in size from the original co-referent name.

Each of those similarity measures excels in finding at most 2 different types of name forms (e.g., spelling variations, typos, nicknames, ..etc). However, if we are to find the maximum possible number of name variants, relying on a single static similarity measure was found ineffective. Therefore, a classification model was trained and tested on different types of name forms to classify name pairs in the text as co-referent or non-co-referent. Finally, we encrypt all names in text using a one-way hashing function while storing the record linkage information between each pair of hashes to minimize information loss after anonymization.

For future work, an analysis could be done on different domain-specific textual data to explore record linkage techniques most suitable for domain-specific name variants. Furthermore, the classification model can be trained to label each pair of personal names in a given text by the type of the co-referent name variant. This allows us to choose the most relevant similarity measure for different types of name forms.

Chapter 5

Evaluation and Results

In this chapter, we aim to present and evaluate the results of the proposed methodology. To achieve this goal, we generated an unbiased test case to evaluate the results for each of the method modules. Next, a method analysis was conducted to explore the benefits and the risks of the approach.

5.1 Module 1: Finding Personal Names in Unstructured Dutch Text

In this section, we aim to describe the steps followed to evaluate the results of our method in detecting personal names in unstructured Dutch text. Firstly, we describe the procedures taken to generate a test case, then we analyze and evaluate the obtained results.

5.1.1 Test Case Generation

To test our method in detecting personal names with typographical errors, different types of errors were considered. Typographical errors are usually caused by one of five following types [20]:

1. **Substitution error:** occurs when another character replaces the correct character.
2. **Deletion error:** occurs when a user misses a particular character in a word.
3. **Transposition error:** occurs when two consecutive characters are typed out of order.
4. **Insertion error:** occurs when an additional character is inserted by mistake.
5. **Replication error:** occurs when a character is repeated twice.

To generate similar typographical errors in the test data, a function was created to randomly alter letters in a word. The altered character was replaced by a nearby character on the keyboard which is the most likely to be swapped or inserted during typing. The function relies on the QWERTY keyboard design that is the most commonly used keyboard layout. The following table 5.1 shows examples of different types of typographical errors generated by the function for a single name.

Table 5.1: Auto-generated typos

Original word	**Miss-typed word**	**Error type**
Margriet	Marg**e**iet	substitution
Margriet	Mrgriet	deletion
Margriet	Marg**ir**et	transposition
Margriet	Margri**d**et	insertion
Margriet	Ma**rr**griet	replication

Using the previously mentioned function, random typographical errors were generated in the text to evaluate the performance of our method in detecting misspelled personal names for every type of error. Firstly, we evaluated the results of detecting personal names from the text. Then, each type of typographical error was analyzed separately to evaluate the effectiveness of our method in detecting errors.

5.1.2 Method Analysis and Results

In this section, we aim to evaluate the performance of our method in detecting personal names from the text. In this step, we assume that each individual's name is mentioned once in the text in the correct form with no typographical error. The method works by combining detected names using the Stanza NER system and a list of names that were previously collected from various sources. To evaluate this method, we asked 5 Dutch individuals to provide us with a list of Dutch names as well as variations of Dutch names (such as 'Antje' which is the Dutch diminutive for 'Anna'). This sample was used to test our results with an unbiased sample that can mimic real-life data. Consequently, a list of 96 names was collected for testing and evaluation. Those collected names are divided into 72 original names and 24 name variants. Out of the 96 names, 4 original names and 3 name variants were not detected by the algorithm with an accuracy of more than 92%. All the undetected names were found to be names of old foreign origins such as Roman or Greek origins. For these names to be detected, they have to be manually added to the names list.

The rest of the names, 89 personal names, were then used to analyze the effectiveness of the method in detecting mistyped names in the text. A short Dutch text was created consisting of two sentences where the same name is mentioned two times in different parts of speech, once in the correct form and another time with one type of typographical errors. We assume that one of those two occurrences is the correct spelling of the name so that it is detected by our NER and names list combination approach. Afterward, part-of-speech was used to find all other nouns and proper nouns in the text and compare those to all the detected names using a Levenshtein edit-based distance to decide if there is a misspelled name that was not detected. To evaluate this method, we look at the following metrics:

- The number of false positives: which are the words that are detected as names when they are not actual names. This can happen when a word in the text is similar in spelling to one of the personal names detected in the text such as 'Jaap' and 'jaar'. 'Jaap' is a Dutch name that can be short for 'Jacob' or 'Jacobus', whereas 'jaar' is the Dutch word that means 'year'. In case both are mentioned in a text. This can result in a false positive as 'jaar' in this case will be considered as a misspelling for 'Jaap'.

- The number of false negatives: which are the names that are not detected. This can happen for one of the following two reasons:

  1. If there are foreign names that were not detected by NER and are not in our names list.
  2. If a name has more than one type of typographical error. In this case, the edit-based distance between the original name and the misspelled name might be high so that it will not be detected as a potential misspelling.

In the following sections, we analyze separately the ability to detect names with a specific type of typographical error.

Substitution Errors

Substitution errors are one of the most common errors that happen while typing. It is highly dependent on the layout of the keyboard. Intuitively, a character is more likely to be replaced by a nearby character on the QWERTY keyboard. For example, if we are to type a word that contains the letter 'g' and substitution typos is to happen, the altered letter would probably be a ’f’, ’t’, ’y’, ’h’, ’v’, or ’b’. With this in consideration, a function was created for evaluation that randomly alters a random character in a name with a nearby character. Therefore, we can find out if our typo detection method can be effective in such cases. For evaluation, each of the 89 names was inserted into sentences in Dutch, once in a correct form and another with a randomly generated substitution typographical error. The results showed that 100% of mistyped names were able to be identified using Levenshtein edit-based distance. For experimental purposes, Two different thresholds were used for Levenshtein distance. Firstly, we set the threshold to 1 so that mistyped names cannot be more than 1 apart from the original name. Then, we set the threshold to be smaller than or equal to 2. This did not impact the results for this type of typographical error. Since the Levenshtein distance between any two words in which one of them has a single substitution error will always be equal to 1. This is not the case when we have more than one substitution error in a single word. Setting a higher threshold for the edit-based distance between two words can improve the fault tolerance. However, increasing the threshold results in compromised overall accuracy by introducing more false positives where more words can be matched with the original name as mistyped names.

Deletion Errors

Another type of typographical error is the deletion error which happens when the user skips a particular character that should have been entered. To simulate this type of error in our test sample, a function was created to randomly delete a single character from the inserted word. The method was tested with the 89 names with two different Levenshtein edit-based distance thresholds. Setting the threshold to be equal to 1 or 2 did not affect the results. In both cases, only 1 of 89 names was detected as a misspelled name. This happens because after removing 1 letter from the name, especially the first letter, the name’s part-of-speech changes to be neither a noun nor a proper noun. This means that it gets excluded from the sample of words that are compared to the original name. To overcome this problem, the sample of words in the text can be expanded not to be limited to only nouns and proper nouns. However, this increases the risk of false positives as we compare to many more words that might be similar to the original correct name.

Transposition Errors

The transposition error is when two consecutive characters are switched in order. A function was created to simulate the transposition error in the inputted names. The function randomly chooses two consecutive letters and switches their order. The Levenshtein edit-based distance between an original name and the same name with simulated transposition error mostly equals 2 since this error is equivalent to two substitution operations. Consequently, when the names were tested with the error when the edit distance threshold was equal to 1, only 12% of the mistyped names were able to be recognized. Whereas increasing the threshold to 2, improve the results significantly to be 94% of all mistyped names were recognized.

Insertion and Replication Errors

Insertion errors when a character is inserted by mistake in a word, while replication error is when a character is repeated twice. Both errors result in the same edit-based distance from the original word when they occur. When a character is inserted into a word, no matter the position, the Levenshtein edit-based distance between the original word and mistyped word is equal to 1. Therefore, increasing the distance threshold does not impact the results. After simulating these errors and testing, it was found that 99% (88 out of 89) of all mistyped names with insertion or replication errors are recognized using Levenshtein distance. In some cases mistyped names are not detected, despite the small distance between the mistyped name and the original name. These cases occur when the inserted letter changes the part-of-speech of the word so that it is not detected by the NER system as a noun or a proper noun.

5.1.3 Discussion

Combining a NER system with a names' list to detect personal names in Dutch text has proven to be very effective. In this thesis, the names' list resembles Dutch names collected from different online sources. Accuracy can be significantly increased if the list of names contains all peronal names in given textual data. For instance, if the given textual data represent patient reports, the list of names could be patients' names which ensures detecting at least all correctly spelled names in text. Consequently, mistyped names can be easily recognized by comparing them to the detected names using edit-based distance. The sample used for testing consists of 89 names detected by our name detection method out of 96 names. The whole sample was collected by asking 5 Dutch individuals for a list of Dutch names. Figure 5.1 shows the number of detected names for each type of typographical error with 2 different Levenshtein distance thresholds. It was found that deciding the threshold for the Levenshtein distance to match mistyped names can be crucial in finding certain types of errors. Levenshtein distance can be a great choice for substitution, deletion, insertion, and replication errors since with a threshold = 1, we can detect approx. 90% of those types of typographical errors. However, transposition errors require increasing the threshold to be recognized. The goal is to find a balanced threshold that can match mistyped names with their original names without compromising the accuracy that leads to increased rates of false positives.

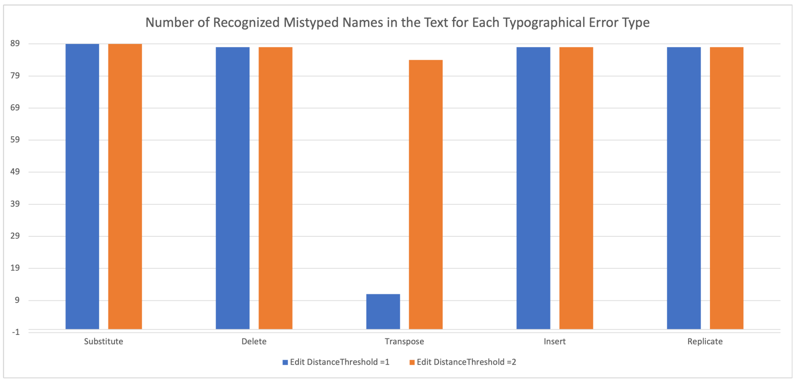


Figure 5.1: Distribution of recognized mistyped names in the text for each type of typographical error

5.2 Module 2: Record Linkage of Personal Name Variants

This section is dedicated to evaluating the second module of this research paper. In this module, we use supervised classification for record linkage. This uses learnable similarity metrics to train a binary machine learning classifier to distinguish between co-referent and non-co-referent name pairs. For evaluation, we firstly discuss how the test case was generated to evaluate the performance of the model. Then, we use this test sample to compute and analyze the results stored after anonymization.

5.2.1 Test Case Generation

To evaluate the performance of the model in distinguishing co-referent names and non-co-referent names, we had to collect an unbiased sample that is different from the dataset that was used for training. Consequently, we asked 5 Dutch people to give us a list of names as well as a list of common Dutch variants for each of the names. The list of the names contained 55 names and their variants that resulted in 165 pairs of co-referent names. To test the classification model, the name pairs the list of co-referent pairs was used. For each of the 165 pairs of names in this dataset, the similarity metrics (e.g., features) were calculated. To provide a comparison, 154 random name pairs were also generated from the same dataset. These random name pairs were cross-checked to ensure they were not present (in either order, as the co-referent pairs dataset contains directed name pairs). A high-quality model should be able to differentiate as many of the matching name pairs as possible from the random name pairs.

5.2.2 Method Analysis and Results

In this section, the test case sample was used to test and evaluate the performance of the model in differentiating the co-referent and non-co-referent name pairs. Firstly, the similarity metrics were calculated and normalized for all the test sample name pairs. All the pairs were given a label 1: as co-referent and 0: as non-co-referent. These labels are assigned to be compared with the output of the binary classifier for evaluation. The classification model that was used for testing is the gradient boosting classifier, which gave the best results when compared to other classifiers. To evaluate the results, we examined the confusion metrics which were calculated using the test case sample data with the actual labels and the model predicted labels. Table 5.2 summarises the classification model confusion matrix results. The model was able to predict the true label with an accuracy of approx. 78% which is a good indication for the performance of the model when tested on new sample data.

Table 5.2: Gradient boosting classifier confusion metrics

		Actual Labels		
		True	False	Total
Predicted Labels	Positive	108	13	121
	Negative	141	57	198
	Total	249	70	319

We could observe that the number of false negatives is relatively high, which indicates that some of the co-referent name pairs were not classified as a match by the model. Therefore, we decided to investigate the false negatives sample to identify the reasons why there are a significant number of false negatives. It was found that the reason for most of the false negatives resulted from the fact that some of the Dutch name variants originated from diminutives old names. Those variants can have one common name origin, but, they do not look similar. For example, some variants of the name 'Bartholomeus' are 'Bart', 'Bartel', or 'Mies'. Despite the fact that both 'Bart' and 'Mies' are Dutch diminutives of the same name, the two names are not alike. Therefore, the model could not match them as co-referent name pairs.

On the other hand, when investigating the false positives, it was found that most of them resulted from token-based similarities between names such as 'Liese' and 'Mies' that share the same three letters (e.g, 'ies'). Although the two names do not share the same first name, they still share a common token that increases the similarity between them and results in a false positive.

5.2.3 Discussion

The similarity learning classification model showed some promising results in distinguishing name variants. The challenge of matching names can be more complicated than just similar name spelling or pronunciation. Human language is sophisticated and matching co-referent names is highly dependent on the context and the field from which the textual data was extracted. On the other hand, results obtained from the classification model can balance information loss and privacy. Storing links between names in the text after anonymization helps facilitate social network analysis of the textual data.

In the following section, we discuss how personal names are anonymized and how links between names are stored before anonymization without breaching privacy.

5.3 Module 3: Anonymisation

In this section, we aim to discuss how names are anonymised in the text at the end of the data pipeline after detecting and linking all personal name records.

5.3.1 Method Analysis and Results

Anonymisation is the final part of our data pipeline where we aim to anonymize personal names in the text while satisfying the following two criteria:

1. Names cannot be re-identified.
2. Meaningful links between personal name records are stored.

To satisfy those, we chose one-way encrypts that do not allow for the re-retrieval of data. All names are replaced with those hashes in the text while storing linkage information between the names pairs. Mistyped personal names were given the same hashes as the original names, whereas name variants such as nicknames were replaced with unique hashes than the original name. In addition, we stored the *FuzzyWuzzy* library fuzzy match scores between all the name pairs to provide an insight into the degree of similarity between each of the pairs.

5.3.2 Discussion

In this phase, the objective is to save basic information that helps in finding name variants without risking re-identifying the names. This technique is not only assumed to result in less information loss after anonymization but also facilitates secondary analysis done on the given text. Other data can be stored to help to increase the usability of the data for analysis such as the gender of the name (either feminine, masculine, or neutral). However, adding more information will always compromise data privacy. Therefore, it is important to take into consideration if the data to be stored after anonymization can help re-identifying the names. Besides, if the stored data is going to help analyze the text further.

5.4 Conclusion

In this chapter, we presented an evaluation of the results of the data pipeline introduced in this project. In the first phase, we generated a test case of textual data with different types of typographical errors to measure the accuracy of the algorithm in finding mistyped names in the text. In the next phase, we collected different name variants from native Dutch speakers to evaluate the similarity learning classification model in recognizing co-referent and non-co-referent pairs. Lastly, we gave an overview of how names are anonymized in the text to satisfy two important criteria which are the inability to re-retrieve the data and preserving meaningful links between name records.

Chapter 6

Conclusions

This thesis presents a data pipeline that enables us to process unstructured Dutch text to detect personal names records and link names variants that refer to the same individuals. This enables us to anonymize name records while storing links between records that can assist in text analytics without compromising privacy. To achieve the goal of this graduation project, we formulated it as follows:

Research Goal Anonymising personal names in unstructured Dutch text to allow text analytics without breaching the privacy of individuals mentioned in the text.

In order to achieve this goal, different types of personal names inconsistencies in unstructured were investigated. It was found that names with typographical errors and name variants that refer to the same individual would decrease the usability of the data for analytics after anonymization. Therefore, the research was directed to tackle the following research question.

Research Question How to detect all personal names in unstructured Dutch text as well as linking all the possible variations of records that refer to the same individual to minimize information loss after anonymization?

6.1 Summary of Contributions

As discussed in Chapter 4, this project consists of a data pipeline that is divided into three modules. In the first module, the pipeline process unstructured Dutch text to detect all personal names including those with typographical errors. Personal names in the text were detected with an accuracy of 92% of the sample test data which was divided into 75% of original personal names and 25% of name variants. Levenshtein edit-based distance was used to detect five different types of typographical error (i.e. insertion, deletion, transposition, substitution, and replication). By setting the threshold of the Levenshtein distance to be equal to 1, we were able to detect four types of typographical errors (i.e. insertion, deletion, substitution, and replication) with approximately 90% accuracy. Whereas transposition errors needed the Levenshtein distance threshold to be adjusted to be equal to 2 to be detected with reasonable accuracy. This is because transposition errors happen due to switching two consecutive letters in a word resulting in two edit operations.

In the second module, we conducted record linkage research to investigate name variants that refer to the same individual. In this step, we firstly investigated similarity measures that can assist in detecting name variants. It was found that static similarity measures (e.g, Levenshtein distance, cosine similarity, ..etc) can not accommodate the variability that may occur due to the multiple ways of referring to the same individual such as nicknames, abbreviations, or misspellings. Therefore, different similarity metrics were combined using classification similarity learning. A machine learning classifier was trained on features extracted using labeled pairwise string comparisons (i.e.

static similarity scores). The performance of the several machine learning models was compared using three different metrics: (1) the overall accuracy, (2) the log loss, and (3) the ROC - AUC. Gradient boosting classifier was one of the most accurate in the binary classification of co-referent and non-co-referent name pairs. In addition, it had a very high computational speed compared to others with the same accuracy. To test the model, different name variants were collected from multiple Dutch individuals. Using this dataset, we created 319 pairs of name variants. The gradient boosting classifier was able to predict the true labels with an accuracy of approximately 78%.

In the third and last module, we anonymized all the names in the text using one-way hashes that don't allow the re-retrieval of the data. All the classification labels of the gradient boosting classifier model were stored between all the pairs along with a *FuzzyWuzzy* [21] similarity ratio. the objective is to store links between the name records that can minimize information loss that results from data anonymization. The stored links information between records can be further used to assist in conducting text analytics on anonymised data. In the following section, we introduce some of the limitations that we faced throughout the project.

6.2 Limitations

Throughout the project, some challenges had arisen that led to the following work limitations:

1. **Lack of available domain-specific unstructured textual data:** Real-life textual data in a specific domain would have been beneficial for the analysis of possible challenges in detecting and linking records of name variants in unstructured text. It is also important to mention that this might have changed the scope of the project having to focus on domain-specific textual data.

2. **Sample size:** To test machine learning models for record linkage, we were limited by the sample size of the name variants that we were able to collect. Increasing the sample size of the test data would have resulted in a better evaluation of the results.

3. **Evaluation of anonymized data:** Due to time limitations, we have not been able to thoroughly analyze the anonymised data. This could have been done by conducting a secondary textual analysis study to assess the usability of the data and the required information to be stored without compromising privacy.

In the following section, we provide some recommendations to consider while replicating this project for a specific case study.

6.3 Recommendations

The results showed that there is no single best technique for co-referent name matching; however, the following recommendations can be considered in choosing different techniques for different applications:

1. considering the way the names in the text are parsed in a standardized way.

2. Training the learnable similarity classification model on a dataset that is representative of the types of name variants of specific domains. Since semantic similarities are not only language-dependent but also domain-dependent. For instance, name forms used in emails or medical reports can be remarkably different than forms used in social media posts.

3. The choice of the similarity measures is crucial for the types name variants of specific domains.

In the next section, the possible future work for further development of our method is presented.

6.4 Future Research

While conducting research for this project, we thought of ways this work can be more generalised. This section provides an overview of possible future research that can benefit from this project.

Expanding to other languages: This research is expected to be extendable to other languages. In this project, Dutch name variants were studied carefully to choose the suitable similarity measures that can represent the similarity between co-referent name variants. By studying the types of name variants in different languages, we can identify types of similarity measures that can be representative of the similarities between name variants. For example, some languages use name variants that can be phonetically similar in which case it can be easily detected using phonetic-based distances. Whereas, in other languages, name variants can be an abbreviation of the original name in which case token-based distances can be the most representative of the similarity. For languages with Latin alphabets such as English or French, it can be very similar to Dutch which makes this method for names detection and record linkage perfectly applicable. However, this might change for other languages.

Social Network Analysis This project can assist in conducting social network analysis on anonymized textual data. Stored links between name records can provide insights about records that refer to the same individuals and records that are not. Different types of secondary analysis, using the provided record linkage data, can be done to confirm or reject the hypothesis of which records are co-referent or non-co-referent. For example, if two name records are mentioned in the same sentence, this can be an indication the two records do not refer to the same individual.

Bibliography

[1] "Stanza – A Python NLP Package for Many Human Languages." [Online]. Available: https://stanfordnlp.github.io/stanza ix, 6

[2] T. D. D. P. Authority. Tasks and powers of the dutch DPA. [Online]. Available: https://autoriteitpersoonsgegevens.nl/en/about-dutch-dpa/tasks-and-powers-dutch-dpa 1

[3] The stanford natural language processing group. [Online]. Available: https://nlp.stanford.edu/ 5

[4] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *IIWEB'03: Proceedings of the International Conference on Information Integration on the Web*. American Association for Artificial Intelligence, pp. {73–78}. 6, 19

[5] V. I. Levenshtein, ""binary codes with dropout correction, insertions and substitutions of symbols "," vol. 163, no. 4, pp. {845–848}. [Online]. Available: http://mi.mathnet.ru/dan31411 6

[6] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage," in *Proceedings of the Section on Survey Research*, 1990, pp. 354–359. 7

[7] B. Li and L. Han, "Distance weighted cosine similarity measure for text classification," in *Intelligent Data Engineering and Automated Learning – IDEAL 2013*, ser. Lecture Notes in Computer Science, H. Yin, K. Tang, Y. Gao, F. Klawonn, M. Lee, T. Weise, B. Li, and X. Yao, Eds. Springer, pp. 611–618. 8

[8] F. Hassan, J. Domingo-Ferrer, and J. Soria-Comas, "Anonymization of unstructured data via named-entity recognition," in *MDAI*. 9

[9] S. Dadas, "Combining neural and knowledge-based approaches to named entity recognition in polish." [Online]. Available: http://arxiv.org/abs/1811.10418 10

[10] R. Santos, P. Murrieta-Flores, and B. Martins, "Learning to combine multiple string similarity metrics for effective toponym matching," vol. 11, no. 9, pp. 913–938. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/17538947.2017.1371253 10

[11] J. Foxcroft, A. d'Alessandro, and L. Antonie, "Name2vec: Personal names embeddings," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, M.-J. Meurs and F. Rudzicz, Eds. Springer International Publishing, pp. 505–510. 11

[12] P. Grzebala and M. Cheatham, "Private record linkage: Comparison of selected techniques for name matching," in *The Semantic Web. Latest Advances and New Domains*, ser. Lecture Notes in Computer Science, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, and C. Lange, Eds. Springer International Publishing, pp. 593–606. 11

[13] Appendix:dutch diminutives of given names - wiktionary. [Online]. Available: https://en.wiktionary.org/wiki/Appendix:Dutch_diminutives_of_given_names 14, 21, 23

[14] R. Grishman and B. Sundheim, "Message understanding conference-6: a brief history," in *Proceedings of the 16th conference on Computational linguistics - Volume 1*, ser. COLING '96. Association for Computational Linguistics, pp. 466–471. [Online]. Available: https://doi.org/10.3115/992628.992709 14

[15] Databases. [Online]. Available: https://www.meertens.knaw.nl/cms/nl/collecties/databanken 14

[16] W. E. Winkler and Y. Thibaudeau, "AN APPLICATION OF THE FELLEGI-SUNTER MODEL," vol. Statistical Research Report Series RR91/09, p. 22. [Online]. Available: https://www.census.gov/srd/papers/pdf/rr91-9.pdf 18

[17] J. J. Pollock and A. Zamora, "Automatic spelling correction in scientific and scholarly text," vol. 27, no. 4, pp. 358–368. [Online]. Available: https://doi.org/10.1145/358027.358048 18

[18] M. Del, M. Angeles, and A. Espino-Gamez, "Comparison of methods hamming distance, jaro, and monge-elkan." 20

[19] S. Narkhede, "Understanding auc-roc curve," *Towards Data Science*, vol. 26, pp. 220–227, 2018. 23

[20] K. Shah and G. de Melo, "Correcting the autocorrect: Context-aware typographical error correction via training data augmentation," in *Proceedings of the 12th Language Resources and Evaluation Conference*. European Language Resources Association, pp. 6930–6936. [Online]. Available: https://aclanthology.org/2020.lrec-1.856 29

[21] A. Cohen, "fuzzywuzzy: Fuzzy string matching in python." [Online]. Available: https://github.com/seatgeek/fuzzywuzzy 36