Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Korpus- und Computerlinguistik

Detection of historical spelling variations using cluster analysis and string similarity

Markus Opolka

markus.opolka@fau.de

Student registration no.: 22004202

Date: 20th January, 2021

Supervisor: Prof. Dr. Stefan Evert

Contents

1 Introduction **1**
1.1 Natural Language Processing . . . 1
1.2 NLP and Historical Data . . . 3
1.3 Spelling Variation in Historical Languages . . . 4
1.4 Research Questions . . . 7

2 Related Work **8**
2.1 Substitution List Methods . . . 8
2.2 Rule-based Methods . . . 8
2.3 String-matching Methods . . . 9
2.4 Statistical Models . . . 10
2.5 Machine Learning Models . . . 10

3 Approximate String Matching and Clustering Theory **12**
3.1 Approximate String Matching . . . 12
3.1.1 Levenshtein Distance . . . 14
3.1.2 Jaro Similarity . . . 14
3.1.3 IBM Similarity . . . 15
3.2 Clustering Theory . . . 15
3.2.1 Agglomerative Hierarchical Clustering . . . 17
3.2.2 Affinity Propagation . . . 18
3.2.3 Clustering Evaluation . . . 19
3.2.4 Clustering Visualisation . . . 21

4 Corpus Selection and Preliminary Analysis **25**
4.1 Corpora Analysis . . . 26
4.2 Exploratory Data Analysis . . . 28

5 Spelling Variation Detection **31**
5.1 Evaluation Methods . . . 31
5.2 Experiments . . . 32
5.3 Results . . . 33

6 Conclusion **39**

Appendices **41**

A String Similarity Measures **41**
A.1 Levenshtein Distance . . . 41
A.2 Jaro Similarity . . . 42
A.3 IBM Similarity . . . 43

B Preliminary Corpus Analysis **44**

C Preliminary Cluster Analysis **44**
C.1 german-anselm 45
C.2 swedish-gaw 46
C.3 spanish-ps 47
C.4 portuguese-ps 48

D Cluster Analysis Results **49**
D.1 Affinity Propagation 49
D.2 Agglomerative Hierarchical Clustering 50

Time's arrow neither stands still nor reverses. It merely marches forward.

BoJack Horseman, Time's Arrow

1 Introduction

1.1 Natural Language Processing

Natural language processing (NLP) is an interdisciplinary research field, situated at the intersection of computer science and linguistics [16]. NLP combines a wide range of technologies - such as rule-based systems, statistical models and machine learning - to enable computer systems to process and understand human language. Research tasks in this field range from low-level operations, like word segmentation or named entity recognition, to complex real-world applications, such as spell check software or automated translation.

The first programmable computer was created not even a 100 years ago and even though these machines were nowhere near the processing power available today, using them for NLP was a very early vision. In the 1950s and 1960s researchers already turned their attention to computer-based translation. Inspired by the cryptanalysists of the Second World War, also known as code breakers, this undertaking was seen from a cryptographic perspective, a message in a foreign language - the code - simply needed to be transformed into an understandable language [48]. Even though NLP proved to be quite a difficult task and the initial expectations were not met, this work laid the foundation for future researchers.

Over the course of the 20th century language processing grew from its humble beginnings as a scientific and philosophical endeavour into a global billion dollar industry [1]. Beyond the pure research, many companies find themselves in need for efficient human-to-machine communication or content analytics to investigate large quantities of language data. Not only has customer communication increased with the rise of social media, but this data also offers many opportunities for new markets. The integration of NLP solutions can therefore be seen in many industries like advertising, healthcare, manufacturing and robotics.

Early NLP methodology focused mainly on rationalistic methods based on hand-coded rules derived through introspection. In rule-based machine translation for example, the fundamental idea was to create a general representation of meaning, using complex handcrafted linguistic rules for parsing and processing the source language. This representation would then be used to produce a translation [4]. Another example would be the ELIZA conversation program [49], which used decomposition and reassembly rules to simulate a human-machine communication. However, the development of these systems remained difficult. Most rule-based systems were highly adapted to a specific domain (e.g. the translation of weather reports from French to English), thus limiting their general application. In addition, maintainability remained a central issue, since overlap in a complex network of rules had to be avoided constantly.

With the transition from costly mainframe systems towards the more affordable personal com-

[1] FBI101933: NLP market size, share and industry analysis.

puter architecture ubiquitous today, a paradigm shift took place in the 1990s. These new strategies can be labelled as data-driven or corpus-based methods, as cheaper storage and more computational power made it viable to work with large quantities of real language data. For example, statistical machine translation (SMT) systems use bilingual parallel corpora to calculate the probability distributions of language structures and their respective translations. This means, an SMT system will automatically determine the most likely translation of source and target language based on a statistical model [14].

More recently, in the 2010s, machine learning models have gained popularity. They have found their way into many scientific and commercial sectors due to the availability of abundant computing power and less complex software architectures. In essence similar to statistical methods, machine learning applies algorithms to train a system to perform a certain task, without it being explicitly programmed. Such a system can, for instance, be trained to recognize part-of-speech tags by looking at large quantities of annotated text data. This is referred to as supervised learning, and it is a common practice in contemporary NLP [53]. In recent years, machine learning has proven to be a very effective method for various classification tasks across many research fields.

The data processing architecture of these data-driven approaches is oftentimes similar in design, commonly referred to as the NLP pipeline. In this design, various modules are serialised, so that the output of one feeds into the input of the next. At each module the data is transformed or annotated in a different way. A large task can thus be broken up into smaller more focused work units. For example: raw unannotated language data is first lexically analysed (tokenised), in the next module this sequence of tokens is then annotated with part-of-speech tags and finally a named-entity recognition module runs on the enriched tokens. These stages vary depending on the task at hand. See Figure 1: NLP Pipeline Schema.

Figure 1: NLP Pipeline Schema

Despite having many advantages over rule-based techniques, data-driven NLP still has its limitations. With data being the central component in these models, many drawbacks arise from the quantity and quality of the input data. In practice, very large amounts of data are required in order to train the parameters of a specific model. For example: Google's BERT model - one of the most influential publications in recent times - is trained on the BooksCorpus (800 million words) and the English Wikipedia (2,5 billion words) [18]. As of 2021, the largest language model ever created is GPT-3 175B, which was trained on 300 billion tokens [15].

Data-driven NLP models tend to perform worse when insufficient training data is available [21]. Furthermore, with a large portion of the NLP efforts being English-centric, many of the methods available do not perform well when applied to other languages, either due to a lack of resources or adaptations. As a response, new techniques and research areas are emerging to adapt existing technologies for so-called "Low-Resource" languages. Examples are languages with small speaker

communities, varieties such as dialects or registers, or historical languages (e.g. Old English); the latter of which are the focus of this work. As Soboroff and Audrey [47] summarise:

> The area of Natural Language Processing (NLP) has made significant progress over the last 30 years as a result of the development of machine learning techniques that employ statistical inferences from large training data. These training data were often carefully hand-crafted at huge expense, collected over many years, and targeted a small set of commonly used languages. For various application scenarios, these techniques work quite well. However, they tend to quickly fall apart when these data resources are not readily available. Being able to rapidly retarget existing technologies developed for one language to a new language is tremendously useful in many contexts [...].

1.2 NLP and Historical Data

The digital record of the 21st century will be an invaluable boon for tomorrow's historians. Scholars will have a complete and reliable record of the past, which will provide an audit trail of the actions, thoughts, deeds and misdeeds of those alive today. But there is still a massive analogue backlog of past millennia not available in a digital form and there are many reasons to transform this cultural heritage into a computer-readable format.

One important reason is preservation, to ensure it not being irrecoverably lost. While digital data is still very prone to corruption and loss [2], it has one key advantage over analogue data: it can easily be duplicated and stored at multiple geographically different locations, thus minimising the risk of complete data loss. Another reason is accessibility, for scholars and the general public alike. In this area, digital archives offer vast advantages over analogue ones. Not only can they be accessed globally and simultaneously, as opposed to a single document in a national archive or institutional repository, digitalisation also offers greater access equality, as computer screen readers or zoom functionality can aid visually impaired users; or simply the comfort to access data virtually for mobility impaired users.

There are many ongoing commercial and publicly funded digitisation projects aimed at cultural preservation and improved accessibility. Most prominent might be the Google Books Library Project [3], with the goal of scanning the collections of research libraries such as the Harvard University Library and making them digitally searchable. Publicly funded examples would be the National Digital Information Infrastructure and Preservation Program of the Library of Congress [4], established in 2000 or the European Digital Cultural Heritage and Europeana Expert Group [5], founded to promote online accessibility of cultural material and digital preservation of its member states.

With this growing amount of digitalised culture, new research fields are emerging to investigate

[2] The US-American social network MySpace lost all content uploaded to its site before 2016, due to a faulty server migration. Canadian-American Actress Margot Kidder lost drafts of her autobiography, due to a computer virus.

[3] `https://books.google.com/`

[4] `https://www.loc.gov/`

[5] `https://ec.europa.eu/digital-single-market/en/digitisation-digital-preservation`

it. For example, the field of digital humanities combines various disciplines, such as philosophy, history, linguistics, literature, art, archaeology, music or other cultural studies, with quantitative and computer-based methodologies [39]. This includes pattern recognition, visual computing, text analytics and database technologies for researching cultural heritage. For the humanities, knowledge and information from the past is crucial, as its methodologies and subjects rely on retrospective. Historical languages and NLP play a key role in this inquiry, however, oftentimes processing these languages correctly poses a problem.

Historical languages are natural languages that were spoken at some point in the past. Today's linguists can study the historical records and debate different stages and changes a language has taken in its lifetime. Examples of proposed changes would be the English Vowel Shift or German Consonant Shift, which are used to place language stages in very specific time frames (e.g. Old, Middle and Modern English). This is true for extinct languages as well as ones still spoken today.

However, these temporal borders can be fuzzy, because past and present are ever moving, making the definition of what is historical and what is contemporary not trivial [39]. Not only can there be scientific dissent on how - or even if - changes took place in the past, the gradual transformation into more contemporary versions of a language makes this identification generally difficult. For example, English from the 17th century might still be understood by today's speaker, despite having considerable changes to its lexicon and grammar. Thus, when and how to classify the borders of any historical language is a complex undertaking.

That being said, from an NLP standpoint historical languages may be treated similar to languages actively spoken today, because they do not alter the underlying assumptions of the methodology. This means that many of today's solutions can be applied to them. However, they do pose additional challenges to current NLP techniques. Historical language data usually falls into the low-resource category, which can be due to insufficient records, either not being available in a computer-readable format or missing entirely, or spelling variation splitting the data into numerous subsets. The latter issue will be the addressed in this work.

1.3 Spelling Variation in Historical Languages

One key issue when processing historical data is spelling variance [7] [19] [39]. This is because NLP techniques generally assume that the input text is written in a single standardised language. A reason for this might be the requirement for lexical information about a word, for example WordNet [6]. Slight variation or mistakes in spelling would yield no result in such databases. This seemingly trivial issue can be extended to any situation in which a query is performed and an exact match is expected. And whilst there are techniques to mitigate this, it might increase the overall error-rate.

Another reason is that many data-driven NLP techniques assume that the parameters learned from the training data can be applied to new unknown data. One such parameter can be the surface form, the string of characters that make up the word. With a stable distribution of such

[6] WordNet is a lexical database for the English language created by the Princeton University.

strings or substrings (e.g. suffixes) in a specific context within a corpus, a model can be used to classify unseen data. In this context, spelling variation means the variation of this surface form. For example, when trying to apply part-of-speech tags, the first-person singular nominative case personal pronoun in 21st century German has the surface form *ich* (or *Ich* depending on placement). In a corpus of contemporary German, this surface form does not vary and thus a part-of-speech tagger can easily learn its respective tag. However, in Middle Low German - a dialect group between 1200 and 1600 that has not undergone standardisation - numerous variations of the contemporary surface form can be found: *yck, ick, jk, ik, yk, jck, jc, ic* [6]. While this might be a naive example, an entire corpus with no uniform distribution of spelling poses a serious challenge.

Additionally, this variation does not only complicate automated data annotation, it also decreases the usability of unannotated data. For example, simple text metrics such as keyword frequency will be distorted when there is no standardised spelling, as numerous variations will skew the calculation. Whilst expert users can anticipate some of the variation by thorough investigation of the sources, it increases the amount of work and decreases the accessibility for general audiences.

To elaborate on this issue, Piotrowski [39] categorised historical spelling along two dimensions: synchronic and diachronic variation. Diachronic variation refers to changes of spelling conventions over time. A contemporary orthography can differ from a historical spelling due to various factors, e.g. changes in pronunciation or standardisation efforts. Recent examples are the German spelling reform in 1996 or the French *rectifications* of 1990. Synchronic variation refers to spelling variance due to the absence of a nationally standardised orthography. Oftentimes this is the reason for older texts having a greater variance in spelling, not only in different geographical regions of the same speaker community, but also in texts of the same author. As Piotrowski [39] states:

> [..] nationally or even supranationally standardized orthographies are a relatively recent development. For example, German orthography was formally regulated as late as 1901 and Portuguese orthography in 1911. Spelling was not completely arbitrary before these standardization efforts, but in the absence of a standard, there existed no concept of "correct" spelling. Instead, there were typically different "schools" of spelling or written dialects and a much wider range of acceptable spellings, often reflecting regional pronunciations.

How can contemporary NLP techniques cope with such variation, when they are designed to operate on standardised orthography? One routine approach would be to gather more training data to ensure the models are robust enough to handle the variations [33]. This is feasible for most contemporary applications, especially when additional data can be acquired with reasonable effort (e.g. web-scraping). Another method would be data augmentation [46], which is used to artificially increase the training sample size by generating sensible variations from the existing data. Finally, there is always the option to use human labour for manual processing and evaluation, with the obvious drawback of usually being timely and costly. Besides the latter method, many of these are generally not feasible for historical text. Gathering further

data might not be applicable when the historical data is just not available or would introduce even more variations. And while data augmentation is very suited for some applications (e.g. neural networks for image recognition), so far it has not been proven as a general solution for natural language processing [41]. One could argue that having a solution for generating sensible variations of words that are in line with any existing historical variations, might probably be as difficult as removing the existing variations.

To use the existing NLP toolchain for historical data, transforming the historical spelling into a contemporary form has been proposed as early as 1980 [19]. Similar techniques are used in computer mediated communication (CMC) [2], as this data also contains a high variation of non-standard spelling, such as phonetic spellings or number homophones. This is usually referred to as normalisation in the CMC context, which - at its core - means the reduction of variation by mapping the various surface forms onto a single form. For example:

$$\{tmw,\ tmrw,\ 2morow\} \rightarrow \{\text{tomorrow}\}$$

In a historical NLP context, the mapping of historical to modern spelling has been referred to as normalisation [42], canonicalisation [27] and spelling modernisation [39]. While having a similar goal, there are subtle differences in the scope. From simply replacing historical characters with their contemporary counterparts, to transforming entire words into a modern spelling. However, some researchers also include the translation into a contemporary meaning into this process. In this work, the preferred term for this mapping will be 'spelling modernisation' or simply 'modernisation', meaning: mapping a string of characters in a historical language to its corresponding contemporary version, without including any further semantic aspect.

There are several interesting benefits to this approach. Mainly, the available NLP tools - which are designed for contemporary text - can be applied seamlessly. With a contemporary spelling, every subsequent stage in the pipeline can be used as if the input were modern text. At the same time, the original spelling can be preserved as annotation, allowing for simple access to the historical form and the advantages this entails (e.g. research into their distribution over larger corpora). Finally, users can use a contemporary - and thus more familiar - orthography for queries, reducing the overall entry barrier for working with historical data.

However, many of these modernisation tools apply supervised methods and thus require training data [11]. This means, there is the need for datasets containing pre-defined mappings of historical to contemporary surface forms. Depending on the language, this data might not be available. While there are existing projects that have produced training data ready to be imported, this data might be unsuitable for the task at hand (e.g. a mismatch in time period).

Another approach to the issue of historical spelling variation has been offered by Barteld [6] in 2017. This approach focused on the detection of spelling variation, instead of mapping historical onto contemporary spelling. Therefore, it can be applied in cases where no contemporary form exists. The original proposal uses corpus data to generate potential pairs of spelling variants and then applies a machine learning model to detect similar variations.

Concrete implementations aside, conceptually this approach has not been extensively researched

yet and will therefore be the focus of this work. The following section will outline the project.

1.4 Research Questions

In summary, due to the design of contemporary NLP methods, they do not work well with historical language data, largely due to a general lack of training data and potential spelling variations within the sparse data available. Modernisation, i.e. the mapping of historical to contemporary spelling, has been proposed as a solution. This would allow the seamless integration with contemporary NLP techniques. However, many modernisation tools still require labelled training data, which can be unavailable for a specific language or time period.

Another proposal is the detection of spelling variation, without modernisation, which has not been extensively researched until now. At the time of writing, there only is the original proposal by Barteld [6] focusing exclusively on the detection of spelling variations. This work aims to reduce the research deficit in this area, as well as the area of unsupervised methods for working with historical language data. More specifically, it will provide a closer look at unsupervised cluster analyses to identify similar spellings, with focus on the following research questions:

- Q1: Can spelling variations within a corpus be made accessible to researchers using unsupervised methods?
- Q2: Does cluster analysis offer a viable solution for the detection of spelling variations?
- Q3: Which, if any, algorithms are best suited for the detection of spelling variations?

In order to answer these questions, the structure of this work is as follows: Section 2 will provide an overview of the scientific state of the art and related publications. Section 3 will provide the theoretical foundations and evaluation methods, along with details on cluster visualisation techniques specific to the task at hand. Section 4 will consist of a preliminary data analysis aimed at evaluating the viability of a cluster analysis, as well as details on data preprocessing. Section 5 will contain the primary cluster analyses and the results. Section 6 will conclude this work.

2 Related Work

This section will provide an overview of existing research regarding historical spelling variations in NLP. As the major part of this research has been on spelling modernisation, the focus will lie on these methods. The few studies that have examined methods for solely detecting spelling variations will be explicitly mentioned. The following studies are broadly categorised by their central method applied, similar to previous classifications [11].

2.1 Substitution List Methods

In this approach, a list of historical-to-modern mappings is used to find and replace spelling variations within a given text. This can either be entire words or various characters that are to be replaced, resulting in a more homogeneous text. While the replacement is conceptually and computationally simple and thus very efficient, the overall application of this approach is rather limited. This is because oftentimes this approach requires hand-crafted lists, which can be timely and therefore costly to produce; mostly fitted to the respective project and thus not generally applicable.

An example implementation of this would be the VARD (Variant Detector) tool [42] [5]. In its first iteration, the VARD tool used a search-and-replace component with a pre-processed word list for spelling normalisation. Another example would be the Norma tool [10], which includes various normalisation components, among them a word-to-word mapping component. This mapper automatically compiles a substitution list from training data. When a word can be mapped to multiple contemporary words, the mapping with the highest frequency is chosen.

The main advantage of this method lies in the pre-defined data from historical sources. While this can be labour-intensive, it also can be highly adapted to the task at hand and thus result in very high precision. However, with a broad and sometimes unpredictable distribution of spelling variations, it can be difficult to include every variant in a pre-defined list. Besides being one component in a larger toolchain, this method has therefore not been adopted more broadly in recent times.

2.2 Rule-based Methods

Rule-based methods implement rewrite rules to encode the relationship between diachronic variations and contemporary spelling. This means, applying a set of character-rewrite rules to an historical input string in order to generate a modern counterpart. In contrast to simple substitution lists, these rules typically incorporate some form of context information to identify appropriate replacements (e.g. adjacent characters or words). The complexity of these methods can vary depending on the implementation, with the rules either being hand-crafted or learned from training data.

The earliest implementations of this are Fix [19] in 1980 and Koller [29] in 1983. Summed up briefly, Koller used hand-crafted context-aware rules in order to transform historical German into a contemporary form. For example, the rule “w u *K *K” will transform the character “w” between two consonants into the character “u” (e.g. *zwm* → *zum* (to)).

Subsequent research extended this technique. Most prominently the VARD2 tool [5], which uses several modules to modernise spelling variations, including character replacement rules. It has also been shown that context-aware replacement rules can be derived automatically [13]. Another common approach is the application of phonological sound change rules, in which phonetic transcription is used to model sound changes between historical and contemporary words. Porta el al. [40] used this to analyse spelling variations in Old Spanish. They phonetically transcribed contemporary word forms in a modern lexicon, applied a set of rules expressing the phonetic and phonological changes and then transcribed the resulting forms back to graphemes.

Besides being well researched, these rule-based methods can be highly adapted to a specific language and thus yield very useful results. However, this also means that these rules are highly specific to one language variation - or rather one dataset - and may need to be adjusted when applied to other data (i.e. the same language from another time period). In order to generate or adjust these rules automatically training data is required, which either needs to exist or to be created manually; which again can be rather labour-intensive.

2.3 String-matching Methods

String-matching methods use various forms of approximate string matching to find spelling variants in a contemporary lexicon [28]. In regular string comparison the characters in both strings are compared one by one, meaning a single character mismatch returns a complete mismatch. Approximate - or fuzzy - string matching tries to identify non-exact matches in two character sequences. For example, the misspelled input “tommorrow”, of the correct spelling “tomorrow”, only differs by a single character and should therefore be treated as a match. This is often applied in information retrieval and spell-checking software [35].

There are several approaches to approximate string matching. Examples include finding the longest common subsequence (LCS), calculating a numeric similarity measure or compute an edit distance between two strings. One of these distance measures is the Levenshtein-Damerau distance, which is calculated by the minimal number of character deletions, insertions, and substitutions required to transform one string into another. Examples for these operations are:

- Insertion (example → exaample)
- Deletion (example → exaple)
- Substitution (example → exemple)
- Transposition (example → exmaple)

There are also weighted variants of string distances, which can assign lower costs to express more likely edit operations [37]. This is useful to encode expected changes in the history of a language (e.g. consonant or vowel shifts). Additionally, these edit weights can be learned from a training set, in order to customise the algorithm to a specific dataset. The previously mentioned Norma tool [10] also includes such a component. Barteld [6] used distance metrics in order to detect spelling variation in absence of reference to a contemporary standard. Meaning, the detection of pairs of strings that are variants of the same morphological word. This is so

far the only example of spelling variation detection without modernisation.

One central benefit to this approach is that approximate string matching can be applied without training data and predefined rules, as only a contemporary lexicon is required for the comparisons [38]. Of course, weighted variants of this method do require - potentially non-existent - training data, but do offer a higher precision in return.

2.4 Statistical Models

From a statistical point of view, spelling modernisation can be treated as a translation task. The historical variant "simply" needs to be translated into its contemporary counterpart, which has led to the application of various character-based machine translation systems. Statistical machine translation (SMT) systems use bilingual parallel corpora to calculate probability distributions of language structures and their translations. This means that an SMT system will automatically determine the most likely translation of source and target language based on a statistical model. A simplified example would be to assign a probability $P(Target|Source)$ to every sentence pair in the corpus. Thus, the probability that a translator produced the target sentence $Target$ given the source sentence $Source$. The best translation can then be found by searching for the highest probability $P(Target|Source)$ in the corpus [14].

Most commonly, character-based statistical machine translation (CSMT) is applied for spelling modernisation [45] [36]. CSMT models are trained on character sequences instead of word sequences, thus the model will predict the next character instead of the next word. This approach is reasonable since the modernisation should address (minor) changes in spelling rather than a translation of entire words.

In general, statistical NLP methodology is very well researched and statistical machine translation in particular has received great attention by researchers and the industry. There are many CSMT tools and frameworks to assist researchers; and the performance in historical spelling modernisation has been demonstrated repeatedly [11] [37]. However, as with many supervised methods, there is a need for training data which might not be available.

2.5 Machine Learning Models

Due to the lower barrier of entry and availability of abundant computing power, machine learning models have gained popularity in the 2010s and have found their way into many scientific and commercial sectors. In brief, machine learning systems perform tasks without explicit instructions. Instead, these systems "learn" the given task from vast amounts of data while fine-tuning parameters autonomously over many tries. Often cited "neural networks" are a subcategory of machine learning algorithms. Such a system could, for instance, be trained to recognise part-of-speech tags by looking at large amounts of annotated text data. This is referred to as supervised machine learning, and it is a common practice in NLP [53].

As in statistical approaches, spelling modernisation is treated as a translation problem, thus character-based neural machine translation (CNMT) has mostly been researched [12] [30]. In essence, these models work by combining two neural networks. The first network acts as encoder,

it processes an input sequence (e.g. historical word) and returns its own internal state. The second network acts as decoder, its input is the encoder's internal state. It is trained to predict the next characters of the target sequence (e.g. contemporary word), given previous characters of the target sequence. There have also been sequence labelling experiments [3], which use word alignments to predict entire word sequences.

One major problem with any machine learning model lies in the large quantity of training data that is required to achieve satisfactory performance. Additionally, this data has to be balanced to avoid overfitting the model (similar statistical approaches). However, one additional aspect of neural networks is the computation time required. Since the parameters of the network have to be adjusted over numerous iterations with large amounts of data, training the model can take hours or even days [52]. Furthermore, CNMT struggles when encountering rare words or structures, since these might be scarce within the training data and the model cannot adjust its parameters accordingly [50]. Nevertheless, there is a vast amount of research in the field and there are many tools and frameworks available.

In conclusion, there have been numerous different approaches to spelling modernisation. Many of which are based on state-of-the-art NLP methodology, with statistical and machine learning methods at their core. The requirement for specific labelled training data is the primary disadvantage of many methods, as the language data of a given time period cannot be generalised. While there have been some advances in this area, the problem is far from being solved.

As mentioned, only a few studies have examined methods for detecting spelling variations in historical data without a mapping to a contemporary form. This work is indented to extend the research in this specific area. It will introduce a cluster analysis approach based on approximate string matching for the detection of spelling variations. The next section will introduce both of these concepts and provide theoretical details.

3 Approximate String Matching and Clustering Theory

Cluster analysis or clustering allows to automatically group similar observations into a smaller number of clusters, it therefore seems an optimal solution for finding similar spelling variations. There are, however, some considerations to be made beforehand. Mainly, how to express the similarity of two words in a numerical way? As already mentioned, approximate string matching tries to answer this question. These algorithms can be used to express textual similarity in a numerical way, instead of a simple Boolean expression.

Combining these techniques in order to find historical spelling variations has been applied by Amoia and Martinez [1]. They describe the spelling modernisation of a corpus of cook books written in German between 1569 and 1729, containing roughly 500.000 tokens. For this, the authors first utilise clustering techniques based on string similarity measures to detect variations of words. Second, they apply paraphrase recognition techniques in order to identify semantic variants, meaning variants that are not realised as similar strings but rather different denominations of the same object. Finally, they integrate these results to generate a dictionary of variants, that can be used to extract a modernised spelling for each token within the corpus.

In their work they apply the standard Levenshtein distance, the rationale behind this choice and details on the implementation are - unfortunately - not elaborated upon. This work will try to extend the initial cluster analysis applied by Amoia and Martinez in 2013. The primary focus will be on the combination of clustering algorithms and string matching algorithms, in order to determine if and how these can be used for the unsupervised detection of spelling variations.

3.1 Approximate String Matching

Approximate string matching algorithms provide techniques to express string similarity. Computers store textual data as strings and oftentimes the underlying data structure is a sequence of characters. Since contemporary computer architecture is binary, these characters need to be encoded binary. Early systems used an eight bit (one byte) encoding named ASCII for this, which was replaced by the more flexible UTF-8 encoding. When two strings are compared, this comparison is thus done byte by byte. For example, Python's Cpython implementation uses the C built-in standard library function *memcmp()*, which compares raw regions of memory:

```
return memcmp(PyUnicode_1BYTE_DATA(a), PyUnicode_1BYTE_DATA(b),
              PyUnicode_GET_LENGTH(a) * PyUnicode_KIND(a)) == 0;
```

This means, a single character mismatch will result in a complete mismatch; which is expected behaviour in most cases. There is, however, a need for matching similar strings. For example, when implementing a spell-check application that should detect that "huose" is the misspelled word "house". Other common applications are the deduplication of records (e.g. names or addresses), or even finding similar DNA patterns. In such cases, regular string matching would result in a mismatch:

```
Python 3.6.9 [GCC 8.4.0] on linux
>>> 'house' == 'house'
```

```
True
>>> 'house' == 'huose'
False
```

There is a variety of algorithms to solve this issue in different manners, that can broadly be categorised as such:

- Token-based, in which various subelements of two strings are compared to each other. These can be for example: n-grams for a word, or words themselves in a document.
- Edit-based, which compute the number of operations needed to transform one string to another. A lower number of edits can thus be interpreted as more similar.
- Phonetic, which utilise phonetic information to determine similar strings.
- Hybrids, which are algorithms that combine various other measures.

All of the above express string similarity as some numerical score, ready to be used in mathematical operations. It is important to note that the terms 'distance measure' and 'similarity measure' are oftentimes used interchangeably. To clarify, while distance measures can be interpreted as the similarity of two strings, the two terms do not denominate the same operation. This work will use the term 'similarity measure' to denote numerical values that can be interpreted as the similarity of two strings.

In order to determine how the similarity measure impacts a cluster analysis, three distinct measures were chosen for further examination. These measures are: the Levenshtein distance, the Jaro similarity and the IBM similarity. The rationale behind this choice will be explained in the description of each algorithm. All of these string similarity measures were implemented in Python, also including simple unit tests to ensure correct calculations. All code is publicly available [7].

It was decided not to include phonetic similarity measures, as these algorithms tend to be highly adapted to a specific language. For example, the Soundex algorithm converts English words into encodings of four characters. These encodings contain the initial character of a given word, no vowels and similar consonants expressed as number:

```
Python 3.6.9 [GCC 8.4.0] on linux
>>> import fuzzy
>>> soundex = fuzzy.Soundex(4)
>>> soundex('dwayne')
'D500'
>>> soundex('duane')
'D500'
```

Future research might investigate the use of such phonetic similarity measures.

[7] https://github.com/martialblog/master-thesis-code/

3.1.1 Levenshtein Distance

The Levenshtein distance [31] computes the minimum number of characters to be inserted, deleted or substituted in order to transform one string into another. For example, the string "cat" can be transformed into the string "at" with one delete operation, therefore the Levenshtein distance between these strings equals one. From this, the similarity can be interpreted as: the lower the distance, the more similar two strings are.

There are also weighted variants of this measures. These can be used to assign lower or higher costs to edit operations, in order to express the likelihood of such operations. This has not been adopted in this work, as it requires means of calculating these weights, which usually includes labelled data.

See: Figure 2: Levenshtein distance definition, Appendix A.1: Levenshtein distance implementation in Python.

$$f_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} f_{a,b}(i-1,j) + 1 \\ f_{a,b}(i,j-1) + 1 \\ f_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

Figure 2: Levenshtein distance
Where f is the Levenshtein distance between two strings, with string a up to the character at index i and string b up to the character at index j.

This similarity measure was included in this work because it is widely adopted and has been applied in historical spelling analyses [1] [38] [37]. Furthermore, it does not require language-dependent adjustments (e.g. definition of vowels or consonants) and might thus be more adaptable.

3.1.2 Jaro Similarity

The Jaro similarity [24] computes the weighted sum of percentage of matched characters and their transpositions. Transpositions are matching characters which are not in the same position. This numerically expresses the similarity of two strings between zero and one, with zero meaning no similarity and one being a complete match. For example, the Jaro similarity of the strings "history" and "hystorie" equals 0.77; the Jaro similarity of the strings "history" and "example" equals zero, as they do not share any characters.

See: Figure 3: Jaro similarity definition, Appendix A.2: Jaro similarity implementation in Python.

The choice to include this similarity measure is mainly based on it calculating a normalised score (between zero and one), which could be beneficial for parameter tuning. Furthermore, it does not require language-dependent adjustments (e.g. definition of vowels or consonants).

A variation of this is the Jaro–Winkler similarity, which gives more favorable ratings to strings

$$f(a,b) = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}) & \text{otherwise} \end{cases}$$

Figure 3: Jaro similarity
Where m is the number of matching characters and t half the number of transpositions in strings a,b.

that have a matching prefix. This variation has not been applied in this work, based upon a preliminary data analysis that will be discussed in section 4.

3.1.3 IBM Similarity

IBM developed a similarity measure in 2010 to clean noisy text in informal communications such as social media or short text messaging [17]. As there is no official denomination, it will be referred to as IBM similarity. This measure is a token-based and edit-based hybrid, as it combines an edit distance - in this case the Levenshtein distance of the string's consonant skeleton - with a longest common subsequence ratio. This means, it requires a language-dependent definition of consonants or vowels.

It is important to note that the original implementation has a constraint that requires the first characters of each string to match, this constraint was dropped in the implementation in this work. The rationale behind this decision was a preliminary data analysis that will be discussed in section 4.

See: Figure 4: IBM similarity definition, Appendix A.3: IBM similarity implementation in Python.

$$f(a,b) = \frac{(\frac{|(LCS(a,b)|}{|a|})}{levenshtein(a',b')}$$

Figure 4: IBM similarity (adapted)
Where LCS is the longest common subsequence shared in string a and b. a' and b' denote the string's consonant skeleton.

This similarity measure was included to analyse the effects of a language-dependent calculation, as it requires a way to compute consonant skeletons. Furthermore, it also represents a normalised score between zero and one; similar to the Jaro similarity.

3.2 Clustering Theory

Broadly speaking, clustering allows to group similar observations into a smaller number of clusters based on the variables of each individual observation, emphasising either similarities or differences. This technique is applied in various research fields and industrial sectors. The core of this section will focus on clustering theory, evaluation and visualisation to detect spelling variations within corpora of historical language.

There are many algorithms to conduct a cluster analysis, each with its own set of parameters, as well as various ways to define similarity [51]. Even the definition of what is a cluster highly depends on the underlying algorithm. To cite Hennig [23]:

> Depending on the application, it may differ a lot what is meant by a "cluster", and this has strong implications for the methodological strategy. Finding an appropriate clustering method means that the cluster definition and methodology have to be adapted to the specific aim of clustering in the application of interest.

The following is a non-extensive overview of different clustering methods:

- Centroid- or exemplar-based clustering assigns each object to its closest centroid, the prototype - or exemplar - of each group.
- Density-based clustering connects areas of high object density into clusters.
- Connectivity-based clustering connects objects based on their distance to form groups, meaning closer objects are more likely in the same cluster.

Consequently, several criteria need to be taken into consideration when applying a cluster analysis. These are primarily, but not exclusively: the objects (data points) and their features, the similarity measure, how the objects are compared to each other and finally the clustering purpose. To cite Hennig [23]:

> [...] there is no such thing as a universally "best clustering method". Different methods should be used for different aims of clustering. The task of selecting a clustering method implies a proper understanding of the meaning of the data, the clustering aim and the available methods, so that a suitable method can be matched to what the application requires.

To briefly reiterate, the overall goal of this work is the detection of spelling variations, meaning the variation of surface forms. An example of this would be the historical forms *yck, ick, jk, ik, yk, jck, jc, ic*, each one of which denominates the German first-person singular nominative case personal pronoun. Since an *a priori* declaration of all these variations is not always feasible, as it necessitates a profound knowledge of the data and is most likely very labour-intensive, the aim will be to use cluster analysis to retrieve groups of similar strings.

The two clustering algorithms included in this work are: affinity propagation clustering (APC) and agglomerative hierarchical clustering (AHC). The main reasons for the inclusion of affinity propagation clustering were twofold: first, APC does not require an estimation of the number of clusters beforehand. Second, the notion of an exemplar/centroid in each cluster could be used to identify a prototypical spelling variation.

Agglomerative hierarchical clustering was included for three reasons: first, AHC does not require a predefined number of clusters. This is crucial, as the true number of clusters is unknown and cannot be determined beforehand. Second, its underlying bottom-up approach, meaning each observation starts in its own cluster and pairs of clusters are then merged repeatedly. This allows for the likely existence of unique types with no spelling variation. Third, the resulting

hierarchical structure can be leveraged to find similar objects that are closely related.

Algorithms that rely on a predefined number of clusters (e.g. k-means, BIRCH) are not applicable in this work, mainly because it is unfeasible to determine this number beforehand, given the structure of the data and the overall goal. Spectral clustering, in which dimensionality reduction is performed to cluster a lower number of variables, could be adequate given the dimensionality reduction analysis in section 4, but was excluded due to unsatisfactory results in a small preliminary analysis.

This work uses the Python sklearn library for all cluster analyses [8].

3.2.1 Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering offers a bottom-up approach, meaning each observation starts in its own cluster and pairs of clusters are merged repeatedly. The AHC algorithm is as follows: first, each observation starts in its own cluster. It then repeatedly identifies the two clusters that are closest together and merges these. This iterative process continues until all the clusters are merged; resulting in an hierarchical tree-like structure.

The two main parameters for this algorithm are: a similarity metric, expressing the distance between observations, and a linkage criterion, expressing the distance between two clusters. The algorithm will merge the pairs of clusters that minimise this linkage criterion [44]. The Python sklearn implementation also offers a linkage distance threshold parameter, which specifies a value above which the clusters will not be merged.

An example similarity metric would be euclidean distance, the square root of the sum of the square differences. See Figure 5: Euclidean distance.

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Figure 5: Euclidean distance

There are several common linkage criteria: single linkage, in which the distance between two clusters is defined as the minimum distance between their objects. This minimises the distance between the closest observations of pairs of clusters. See Figure 6: Single linkage.

$$d(C_i, C_j) = d_{min}(C_i, C_j) = min_{x \in C_i, y \in C_j} d(x, y)$$

Figure 6: Single linkage

Complete linkage, in which the distance between two clusters is defined as the maximum distance between their objects. This minimises the maximum distance between observations of pairs of clusters. See Figure 7: Complete linkage.

[8] `https://github.com/scikit-learn/scikit-learn`

$$d(C_i, C_j) = d_{max}(C_i, C_j) = max_{x \in C_i, y \in C_j} d(x, y)$$

Figure 7: Complete linkage

Average linkage, in which the distance between two clusters is defined as the average distance between their objects. This minimises the average of the distances between all observations of pairs of clusters. See Figure 8: Average linkage.

$$d(C_i, C_j) = d_{avg}(C_i, C_j) = \frac{1}{|C_i| * |C_j|} * \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$$

Figure 8: Average linkage

And finally Ward's method, which minimises the variance of the clusters being merged. See Figure 9: Ward linkage.

$$d(C_i, C_j) = d_{wrd}(C_i, C_j) = \sum_{x \in C_i \cup C_j} d(x, cen_{ij}) - [\sum_{x \in C_i} d(x, cen_i) + \sum_{x \in cen_j} d(x, cen_j)]$$

Figure 9: Ward linkage
Where cen_C is the cluster centroid.

3.2.2 Affinity Propagation

First published in 2007 by Frey and Dueck [20], affinity propagation identifies exemplars for each object in the dataset and places all objects with the same exemplar in the same cluster. This is done by viewing all objects as nodes in a network and transmitting messages in between, informing each object about the relative attractiveness to the sender. The APC algorithm is described by Frey and Dueck as such:

> Affinity propagation takes as input a collection of real-valued similarities between data points, where the similarity $s(i, k)$ indicates how well the data point with index k is suited to be the exemplar for data point i. [...] The "responsibility" $r(i, k)$, sent from data point i to candidate exemplar point k, reflects the accumulated evidence for how well-suited point k is to serve as the exemplar for point i, taking into account other potential exemplars for point i. [...] The "availability" $a(i, k)$, sent from candidate exemplar point k to point i, reflects the accumulated evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from other points that point k should be an exemplar.

The algorithm then iteratively updates the responsibilities and availabilities as follows: Figure 10: Responsibility computation, Figure 11: Availability computation.

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\}$$

Figure 10: Responsibility computation

$$a(i,k) \leftarrow \min\{0, r(k,k) + \sum_{i' \notin \{i,k\}} \max\{0, r(i',k)\}\}$$

Figure 11: Availability computation

The message-passing proceeds until a consensus is reached. Once the sender is associated with one of its targets, that target becomes the point's exemplar. All points with the same exemplar are placed in the same cluster. The Python sklearn implementation also offers a damping factor, which specifies the extent to which the current value is maintained relative to incoming values.

3.2.3 Clustering Evaluation

In order to validate different parameters and to compare clustering algorithms, their results need to be evaluated reliably. However, in contrast to supervised classification - in which the performance of a model can be verified by number of errors or the precision and recall - the evaluation of an unsupervised cluster analysis is not trivial. This is because different algorithms have different definitions of what a cluster is and the optimal number of these clusters might depend on the specific task [23], meaning that the evaluation highly depends on the initial objects and the overall task.

Hennig [22] states six basic principles for the validation of a cluster analysis (CA):

> **Use of external information** External information is information that has not been used to generate the clustering. Such information can stem from additional data or from background knowledge. However, such information is often not available.
>
> **Significant tests for structure** Significance tests against null models formalizing no clustering structure at all are often used to justify the interpretation of a clustering.
>
> **Comparison of different clusterings on the same data** Often, the agreement of clusterings based on different methods is taken as a confirmation of clusters.
>
> **Validation indexes** In some sense, the use of validation indexes is similar to that of different clusterings, because many CA methods optimize indexes that could otherwise be used for validation.
>
> **Stability assessment** The stability of clusters can be assessed by techniques such as bootstrap, cross-validation, point deletion, and addition of contamination.
>
> **Visual inspection** Recently [...], it has been recognized that all formal approaches of cluster validation have limitations due to the complexity of the CA

problem and the intuitive nature of what is called a cluster. Such a task calls for a more subjective and visual approach.

More broadly, this can be summarised in two categories: external and internal validation. External information, for example comparing the results to an existing classification, would be similar to the validation of supervised methods. While such an external evaluation offers very definite and comparable results, the use of labels also poses its main problem. As these labels are oftentimes not available and would likely - and paradoxically - render the need for a CA obsolete. Additionally, it might prevent the discovery of another, potentially better, clustering.

The use of internal factors, which tries to evaluate the clusters with the data itself, offers a more flexible solution. As mentioned, this is often expressed in numerical indexes, scores or structural tests. Examples for such internal measures would be the sum-of-squared-error (SSE) or the Silhouette coefficient. SSE is the sum of the squared differences between each object and its cluster's mean, to express the variation within a group. If all objects within a cluster are identical, the SSE would then be zero. The Silhouette coefficient is based on the idea of cohesion (the sum of the weight of all links within a cluster) and separation (the sum of the weight between nodes in the cluster and nodes outside the cluster). Central advantage and disadvantage of these evaluation measures is their general purpose approach, offering a unified scoring for different clustering applications is advantageous; however, a single numeric value might not be sufficient to express an optimal result.

As no single option appears sufficient on it's own, this work will apply various methods for the validation of the cluster analyses. These methods are: the comparison of different clusterings on the same data, the use of validation indexes and finally visual inspection.

As already mentioned, this work will apply both affinity propagation and agglomerative hierarchical clustering with various parameters. The performance of these different models on the same data will be compared by examining performance indicators (i.e. number of clusters, number of objects per clusters and n-largest clusters) and evaluated by using a custom validation index.

Since neither of the mentioned internal measures is optimal for the given CA, it was decided to devise a custom cohesion measure to assess the effectiveness of the clustering. However, this can not be interpreted as conclusive evidence of performance, but rather as an indicator, especially useful for parameter tuning.

This measure is the average string similarity of each cluster, applying the previously described Jaro Similarity. See Figure 12: Averaged Jaro Similarity Index. Since the grouped objects are strings, it is reasonable to assume that the previously discussed string similarity measures can be used to express how closely related these objects are. From these values, simple descriptive measures such as central tendency (arithmetic mean, median) and spread (variance, standard deviation) can be calculated. This is intended to give better insight into the similarity of the strings within each cluster.

In addition to this index, it was decided to also rely on human evaluation using visualisation.

$$index = \frac{\sum_{i=0}^{C} \sum_{j=0, j \neq i}^{C} jaro(x_i, x_j)}{|C|}$$

Figure 12: Averaged Jaro Similarity Index

The reasoning behind this is the aforementioned complexity in evaluating cluster analyses. The next section will provide details on this decision and the custom implementation, with the primary focus on agglomerative hierarchical clustering. Since the affinity propagation results can simply be expressed as a list of clusters.

3.2.4 Clustering Visualisation

Visualisation is a common technique in data analysis, ranging from simple sortable tables to very complex renderings of multivariate data. Not only does this aid experts in exploratory data analysis, it can also help communicate findings to a general audience. While this facilitates the identification of meaningful groups, creating and exploring these visualisations can be subjective, more time consuming and require domain expertise [9]. Especially when working with linguistic data, as visual properties beyond the text itself are not trivial to define.

There are various options to choose from when visually exploring a cluster analysis. A very simple example would be scatterplots, in which spatial proximity of objects highlights the individual clusters. This is obviously not always applicable, for example when the data is high-dimensional or spatial placement is not meaningful. Since agglomerative hierarchical clustering is used in this analysis, a sensible option would be a tree-like structure, more specifically: a dendrogram. Dendrograms are tree-like visualisations representing the hierarchy based on degree of similarity, for example in biological taxonomy. See Figure 13: Dendrogram example.

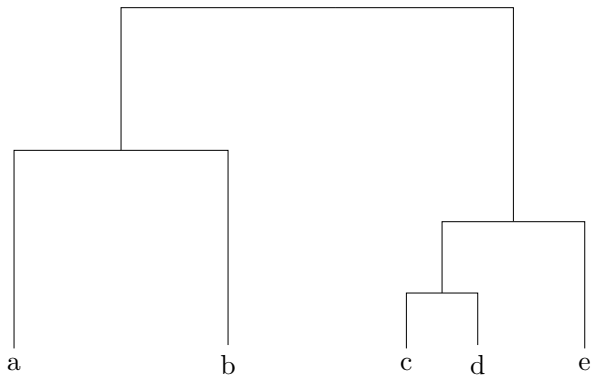


Figure 13: Dendrogram Example

They are simple to understand and can use spacial distances or colouring to transport additional information. Clusters are defined by cutting branches off the dendrogram. One limitation of this diagrams is, however, that it does not scale well for large trees. Bisson and Blanch [9] summarise this quite concisely:

> [...] visualization of large sized trees is known to be difficult since the number

of leaves grows exponentially with the depth of the tree. Practically, when dealing with a dendrogram containing more than a few hundred leaves, any node-link representation of the tree becomes unfeasible.

One proposed solution to this suboptimal use of space are treemaps. A treemap recursively subdivides rectangles and nests them within each other, optionally applying colour as an additional dimension. This design was first proposed by Ben Shneiderman in the 1990s [25] to display a hierarchical tree in a space-constrained layout. See Figure 14: Treemap example.

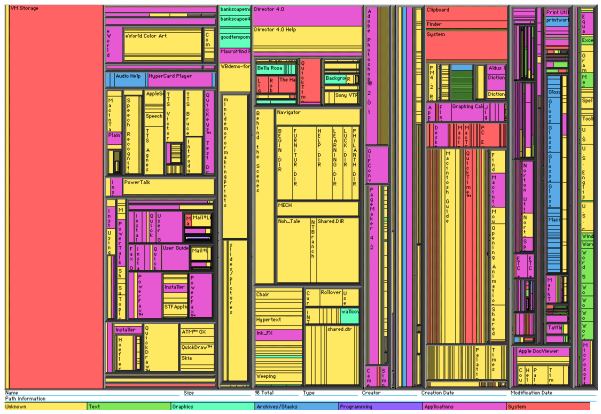


Figure 14: Treemap Example (TreeViz)

There are adaptations that use various shapes to achieve a better rendering. Circular packing, for example, uses circles instead of rectangles. Traditional treemaps are commonly used to visualise tree-like computer filesystem layouts (e.g. Unix) and offer an unmatchable density of information. However, they are more suited for a top-down approach, which makes them unsuitable for the proposed bottom-up analysis; since the focus will lie more on the leaves of each cluster.

Another widely used technique are heatmaps, often in combination with dendrograms when used to display a hierarchical cluster analysis. Heatmaps are two-dimensional table-like structures, in which rows and the columns are re-ordered according to the hierarchical clustering results, again using the cell colour as an additional dimension. See Figure 15: Heatmap example.

Heatmaps are especially applicable for illustrating hierarchical clusters with additional dimensions for variables. Given the use of tree-like structures in the axis, heatmaps have the same limitations as dendrograms when applied to large datasets. Additionally, displaying extra dimension is not a requirement in this analysis, since there are no further attributes available for each datapoint. In summary, these common visualisation techniques are not feasible for very large clusters. This is often the case when working with large corpus data, especially when no further features are available for highlighting or subsetting. While there are of course implementations that can mitigate this by applying functions such as filtering, zooming or panning, the suboptimal use of space as underlying issue remains. It was thus decided to focus on more adequate renderings.

As a solution to overcome these limitations and to optimise the use of space, a hybrid visualisation was chosen. Bisson and Blanch [9] have proposed a visualisation technique named “Stacked

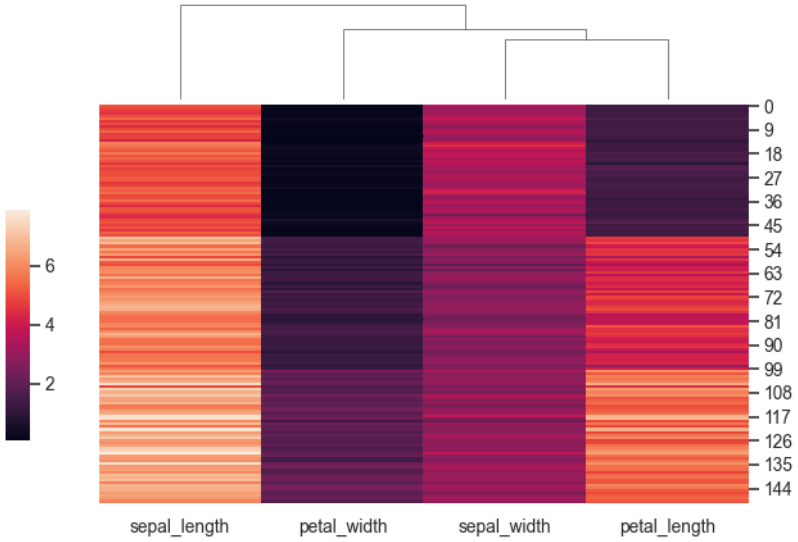


Figure 15: Heatmap Example (python-seaborn)

Trees" to address similar challenges in biology and bioinformatics. One benefit of this method is the optimised used of space, making it an optimal choice for displaying large hierarchies.

There a two central ideas behind the stacked tree visualisation. First, the leaves are at the focus of the rendering, as they contain the objects under study. Second, the hierarchy needs to be accessible in order to grasp the overall organisation of the clusters. To achieve this, the intermediate structure between root and leave is collapsed into a single object: the stack. This creates a condensed tree, capable of rendering large quantities of objects. See Figure 16: Stacked Tree example.

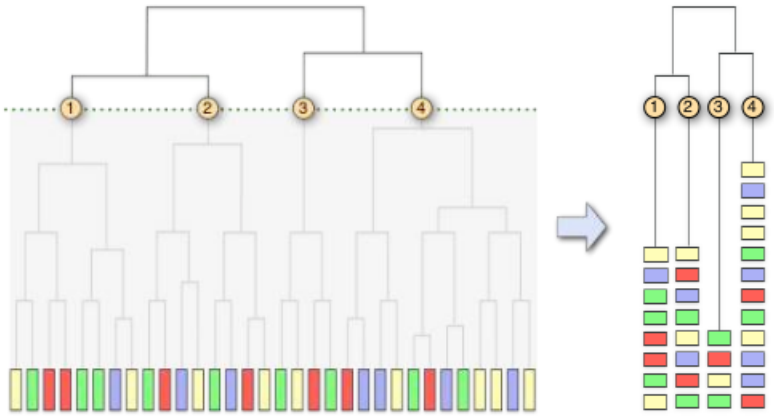


Figure 16: Stacked Tree example (Bisson and Blanch)

To enable a detailed look into the resulting cluster analysis and to identify spelling variations, it was decided to implement an interactive tool for the visualisation. This is in line with the stacked tree proposal by Bisson and Blanch in 2012, as it also emphasises interactive adjustment of parameters.

While there are interactive tools for cluster visualisations (e.g. HCE [9], Clustergrammar [10]), they did not satisfy the requirements for this use case (e.g not publicly available, not operating system independent, etc.). The implementation in this work is an open source web application, simply accessible via a web browser. This solution offers various options for interactivity. Furthermore, as it is based on open source technologies, it enables future researchers to adapt and adjust the tool for their needs.

At its core is the JavaScript data visualisation library d3.js [11], which utilises Scalable Vector Graphics (SVG), HTML5 and Cascading Style Sheets (CSS). These open standards enable future researchers to easily adjust or improve upon the provided solution. As the available predefined d3.js hierarchical clustering layouts did not yet provide a stacked tree solution, a custom layout had to be written. A proposal to include this new layout into the default library has been opened and is currently pending.

The tool in its current form provides the following primary features:

- **Variable cut-off parameter** This parameter can be used to interactively adjust the number of clusters displayed. As clusters are defined by cutting branches off the dendrogram, this cut-off parameter reorganises the tree structure depending on the cluster distances.
- **Clickable stacks** The rendered stacks and intermediate nodes can be selected to display all leaves within the subtree.
- **Searchable stacks** The selected stacks can be searched via an HTML input field. This search supports regular expressions.
- **Searchable tree** The entire tree structure can be searched via an HTML input field, any match will highlight the path to each match. This search supports regular expressions.

While it would have been possible to also include the cluster analysis into this tool, this would have increased the overall complexity and restricted the analysis to JavaScript. It was thus decided to follow a loose coupling design, meaning the cluster analysis and rendering tool remain separate components. This makes the tool unopinionated about how the hierarchical clustering is performed and can thus be adapted by future researchers. To ensure this loose coupling, the web tool uses JavaScript Object Notation (JSON) as input format. JSON is a lightweight data-interchange format, it is human-readable and easy to parse and generate. Most importantly, it is a language-independent data format, despite its name.

All code is publicly available [12].

[9] `https://www.cs.umd.edu/hcil/multi-cluster`
[10] `https://github.com/ismms-himc/clustergrammer2`
[11] `https://d3js.org`
[12] `https://github.com/martialblog/master-thesis-code/`

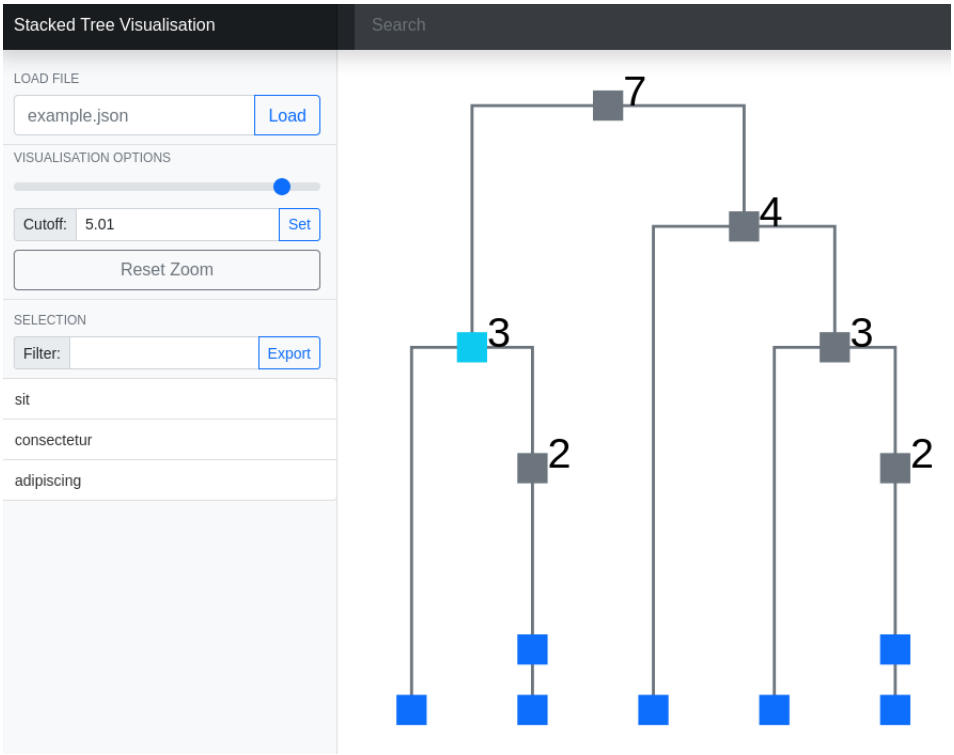


Figure 17: Stacked Tree Web Application

4 Corpus Selection and Preliminary Analysis

This section will describe what corpora were included in the cluster analysis and how this data was preprocessed. Furthermore, it will include a preliminary analysis that was performed in order to gain insight into the data. The primary objective of this preliminary analysis was to validate the feasibility of the subsequent cluster analysis.

All data for this preliminary analysis and any further experiments is a subset of the corpora used by Bollmann [11] in 2019. It was decided to include two germanic languages (German, Swedish) and two romance languages (Spanish, Portuguese). The central reason for this is to examine the effectiveness in various languages, while at the same time providing comparability between the various language families. Each dataset is comprised of tab separated files, that include both historical and corresponding contemporary words. All files were already tokenised and split into training, testing and development data. For example the beginning of the german development dataset:

```
$ head -n 8 german-anselm.dev.txt
deſe diese
wort wort
ſpricht spricht
vnſer unser
liber lieber
here herr
iheſus jesus
```

criſtus christus

Splitting data into development, training and test subsets is a common practice in data science. This is done as a safeguard, to ensure that a model is exclusively trained on the training data and its performance evaluated on different data, thus guaranteeing the model's performance on unknown data. For this preliminary analysis only the development-portion was used, as this subset already provided a reasonable token count:

- German (Anselm, development): 45.993 tokens
- Swedish (GaW, test [13]): 29.460 tokens
- Spanish (Post Scriptum, development): 11.680 tokens
- Portuguese (Post Scriptum, development): 26.854 tokens

The German Anselm corpus was created in 2013 during the Anselm project at the Ruhr-University Bochum. It contains 50 texts from the 14th–16th century, with an average length of 6000 tokens. It is noteworthy that these texts are the same document written in different dialects from Early New High German, Middle Low German, and Middle Dutch. The Swedish Gender and Work corpus (GaW) was created at the Uppsala University in 2008. It contains court records and church documents from approximately 1550 to 1880. They are written in Early Modern Swedish. The Spanish and Portuguese Post Scriptum corpus (P.S.) was created during the Post Scriptum project from 2012 to 2017 at the University of Lisbon. It consists of 2369 letters in Portuguese and 2446 letters in Spanish written between the 16th and the early 19th century.

All data and the preprocessing code are publicly available [14].

4.1 Corpora Analysis

The first part of this preliminary analysis will examine various surface features from the selected corpora. This was done to evaluate the spelling variations within the corpora, as well as the differences between the historical and modernised version. It also provided the motivation behind the previously discussed adjustments to the string similarity measures and will serve as a source for their language-dependent parameters.

It was decided to deliberately include only information computed from surface forms in this analysis, in order to realistically simulate a scenario in which further tools are unavailable for the historical data (e.g. POS-tags, word embeddings, etc.). Thus, the central features are based on types and tokens. The term 'token' refers to the total number of words within corpus, regardless of how often they are repeated. The term 'type' refers to the number of distinct words in a corpus. The surface features and the rationale behind them are as follows:

- Token count, which will simply serve as cross-validation between historical and modernised version and as a general insight into each dataset.

[13] The Swedish GaW development-portion only includes 2650 tokens.
[14] https://github.com/martialblog/histnorm

- Type count, combined with the token count, serves as an estimation of lexical or spelling variance between historical and modernised version.
- Type-Token ratio, which serves as a normalised - and thus comparable - estimation of lexical or spelling variance between each corpora.
- Hapax legomenon count, as an estimation of lexical and spelling variance between historical and modernised version.

The type-token ratio (TTR) is a widely used and simple measure of lexical diversity. It has been applied to analyse vocabulary development in language acquisition, however, there is some controversy with the interpretation of this measure [43]. Nonetheless, in this analysis it can be applied to investigate if and how much the modernisation affects the data. It serves as a measure for spelling variation between versions of the same corpus, as well as a comparable measure between different corpora.

As can be seen in the results, there is a considerable reduction in lexical variation across all datasets; noticeable in the lower type count and lower type-token ratio. See Appendix B: Preliminary Corpus Analysis. This was to be expected, since it can be assumed that several historical tokens are mapped to the same modernised token.

In particular the German Anselm corpus shows the largest reduction in types and hapax legomenon. It can be assumed that the Spanish, Portuguese and Swedish corpus have less variations per type, and thus have a lower reduction in lexical variation. A further investigation revealed this to be the case, as can be seen by the average variation count per corpus:

Corpus	Average variation count
German	19.48
Swedish	4.38
Spanish	4.58
Portuguese	5.86

Figure 18: Average spelling variation count per modernised token

The German dataset is evidently an outlier in this analysis. As already described, the corpus consists of 50 versions of the same medieval text written in a great variety of dialects in Early New High German. The reason behind this higher variation count might therefore be the repetition of the same word in numerous spellings. While the deeper linguistic reasons behind this deviation are not of particular interest here, the data still merits a closer look. The German Anselm corpus clearly illustrates the previously described issues with spelling variation in historical data. The following are all spellings for the five most common tokens in this corpus:

- *die* (the): *dev, dye, d, dÿ, dẏ, de, der, dü, die, dÿe, d, diu, du, du, div, dy, dv, drie, dy, dv, dew, di*
- *kind* (child): *kīd, chint, kīt, chinde, kinden, kinde, chin, kint, chynd, kindt, chindt, chīd, kin., cinde, kȳt, chintt, kind, kynde, chind, kynt, kinth*

- *und* (and): *vndt, vn, ind, unˉ, vnˉ, vnn, enˉ, vˉd, vnd, en, ende, vˉ, vnde, vnnd, inˉ, und, vmd*
- *er* (he): *he, her, er, hie, ir, hi, hey*
- *sprach* (spoke): *fpch, fpṛc, fprch, fprach, fpṛc, fprache, gefprach, sprach, spṛch, fprauch, fpraich, fchprach, gefprache, fpracht, fpräch, spṛach, fpach*

While these historical spellings are likely to be understood by a 21st century native-speaker, especially when viewed contextually in a phrase, there are words that vary greatly from a contemporary spelling. The variation "ende" exists in contemporary German with a different meaning (*Ende*, the end). Variations like "en" or "in" might be confused with the preposition (*in*, in). Other variations like "hie", "he", "hey" do not share any resemblance to the contemporary spelling "er" and are more similar to a contemporary English spelling, exposing their shared origin. These variations in particular might pose a challenge to any automated detection and demonstrate how the inclusion of sentence context might aid the process. This argument has already been made by Jurish [27], in which a strict distinction between type-wise and token-wise modernisation techniques was made. The former looking at each input word independently and the latter making use of the context in which a given word occurs.

What insights were gained from this analysis? First, spelling modernisation generally decreases the lexical diversity. This was to be expected, since many spelling variations are mapped to one word. Second, when using string distance measures (e.g. Levenshtein distance) for spelling modernisation, it is not trivial to determine a value for when strings should be considered similar. Third, when applying string similarity measures, neither matching prefixes nor suffixes can be assumed. As a consequence, any prefix/suffix constraints were dropped in the similarity measures.

4.2 Exploratory Data Analysis

The second part of this preliminary analysis explores the feasibility of a cluster analysis using string similarity measures. It is comprised of dimensionality reduction and visualisation techniques in order to explore how the data might cluster together. Dimensionality reduction projects a high-dimensional onto a low-dimensional space in a way that meaningful information is retained. This has become a common technique in exploratory data analysis [15]. The rationale behind this step was to examine if clusters can be observed when the data is in its numeric form.

In order to apply these mathematical tools to the textual data, it is transformed into a distance matrix using a string similarity measure. A distance matrix is an $n \times n$ matrix containing the pairwise distances of elements in a given set. For example, a list of points $\{x_i, i = 1, \ldots, n\}$ in the Euclidean space $\mathbb{R}^{\mathrm{d}}$ of dimension d. A matrix $D \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ is called a metric distance matrix, when its entries are the distances between pairs of x_i and x_j and they satisfy the basic Euclidean metric properties:

- Non-negativity, $D_{i,j} \geq 0$
- Symmetry, $D_{i,j} = D_{j,i}$

[15] For example: `https://www.linguistik.phil.fau.de/projects/efe/`

- Self-distance, $D_{i,j} = 0$ iff $i = j$
- Triangle inequality $D_{i,j} \leq D_{i,k} + D_{k,j}$

In this analysis, this means each previously described string similarity measure will be used to generate such a distance matrix. In order to optimise efficiency, one can simply calculate the pairwise distances between observations and then convert this vector-form to a square-form distance matrix. This matrix then contains all words on each axes and their corresponding string similarity to each other in the cells. The following artificial example illustrates the final result:

	lorem	ipsum	dolor
lorem	0	4.0	4.0
ipsum	4.0	0	5.0
dolor	4.0	5.0	0

Figure 19: Example distance matrix using the Levenshtein distance

As described, dimensionality reduction projects a high-dimensional onto a low-dimensional space in a way that meaningful information is retained. For example: the historical portion (development) of the German Anselm corpus has 7617 types, resulting in a 7617×7617 matrix. As each row represents the distance of one type to another type, this can be seen as similarity vector in an 7617-dimensional space. A reduction in dimensionality will reduce the size of this vector, while at the same time preserving the expressed similarity. This enables a projection of the high-dimensional data onto a human-understandable space (i.e. two or three dimensions).

One widely used method for this is principal component analysis (PCA) [26], which identifies the highest sources of variation - the principal components - in the data. This enables a more human-understandable analysis in a two or three dimensional space, used in many linguistic investigations [8]. PCA uses matrix decomposition to reduce matrices into constituent parts, in order to perform a linear mapping of the data. There are, however, also methods that use graph structures to achieve similar goals.

Examples of these are t-Distributed Stochastic Neighbor Embedding (t-SNE) [32] and Uniform Manifold Approximation and Projection (UMAP) [34]. At their core, both methods construct a high-dimensional graph representation of the data and then optimise a low-dimensional graph to be as structurally similar as possible. This is done by using weighted graph representation, with edge weights representing the likelihood that two points are connected.

For this preliminary analysis t-SNE and UMAP have been chosen to reduce the dimensions of the similarity vectors. A PCA/MDS analysis did not yield favorable results in a small preliminary study. Only the types are used in the present calculation because using the tokens will likely skew the clustering, as they will naturally group together. This also greatly reduces the number of calculations required, thus speeding up the computation.

This analysis used the Python t-SNE implementation from sklearn [16] and a Python UMAP

[16] `https://github.com/scikit-learn/scikit-learn`

implementation [17].

As seen in the previous analysis, there is a reduction in types from historical to modernised version across all corpora. The examples above have shown that this is due to different historical types being mapped to the same modern type. Thus, it can be hypothesised that these historical types should have a similar vector representation and will therefore cluster together. Furthermore, this clustering might be observable in human-understandable dimension, at least to some extend. See Appendix C: Preliminary Cluster Analysis.

The results show that the preprocessed data express distinct clusters even after the dimensionality reduction, as can be seen in the two dimensional scatterplots. Across all datasets, the t-SNE algorithm seemed less applicable than the UMAP algorithm. In this analysis, the UMAP algorithm generally identified more distinct groups. Especially in combination with the Levenshtein distance, the resulting clusters are expressed very clearly. This could be due to the larger numeric distinction between each input.

It is noteworthy, that this output - or any output with reduced dimensions - could have also been used as input for a cluster analysis. However, it was decided against this approach, since it can not be guaranteed that the data subspace picked by the dimensionality reduction is the most suitable. Future research might further investigate this.

What could be gained from this analysis? First, clustering in general appears feasible on the datasets and the preprocessing worked as intended. Second, if the two dimensional visualisation are an indication for clustering performance, then the Levenshtein distance shows the most potential for distinct clusters; while the Jaro and IBM similarity show more nuanced clusters overall.

[17] `https://github.com/lmcinnes/umap`

5 Spelling Variation Detection

The primary focus of this work is to investigate how and if string similarity measures and clustering algorithms can be applied for the unsupervised detection of historical spelling variations. The previous section has demonstrated that this should be achievable to some extent. Of particular interest is, what combination of algorithms is best suited for the task of spelling variation detection is, as well as the behaviour across different languages.

To reiterate, the following combinations of input data, string similarity measures and clustering algorithms were applied in this work:

- **Corpora** German-Anselm (development), Swedish-GaW (test), Spanish-Post-Scriptum (development), Portuguese-Post-Scriptum (development).
- **String Similarity Measures** Levenshtein Distance, Jaro Similarity, IBM Similarity.
- **Clustering Methods** Affinity Propagation, Agglomerative Hierarchical Clustering.

This section will first provide details on how the effectiveness of the cluster analyses was determined. As already mentioned, this is not a trivial task and requires a very close look at the final results. Secondly, it will describe details on how the various cluster analyses were performed. Finally, the results will be presented and discussed.

5.1 Evaluation Methods

To determine the effectiveness of the various cluster analyses (CA), several performance indicators were defined. Additionally, baseline values for each indicator were established in order to evaluate the CA results. This was achieved by using the modernised section of each corpus to create an "optimal clustering"; mapping the historical tokens to their respective modernised token. This allowed for the calculation of a gold-standard baseline for each indicator, which also served for parameter tuning. These indicators are as follows:

- **The number of clusters** defines the optimal number of clusters that should be created.
- **The average number of objects within a cluster** indicates approximately the desired number of objects within each cluster.
- **The number of clusters with one element** indicates how many elements in a given corpus are not in any cluster.
- **Jaro Similarity Index** defines how similar objects within each cluster are to each other, based on the Jaro similarity. As these numeric values are available for each cluster, both the arithmetic mean and standard deviation can be calculated, which provides a more detailed insight.

Together with this quantitative evaluation, a qualitative approach with a visual inspection was applied. This was achieved with the custom interactive visualisation tool, developed to gain a better insight into the data. Each CA result was therefore exported into a JSON file and the quality of the clustering was inspected. It is noteworthy that this is also applicable when

the definition of a baseline via an optimal clustering is not possible. As this visual approach is intrinsically subjective, several reproducible steps were defined in order to provide a more systematical procedure. These steps are:

- The inspection of elements within the largest clusters determined from the baseline.
- The inspection of elements within the largest clusters determined from the CA.
- A comparison of how similar the elements in these clusters are.
- An investigation of the distance to elements that should be in each respective cluster.

This combination of both quantitative and qualitative evaluation were first used for parameter tuning. This is an essential step, because each parameter needs to be optimised for each combination of clustering algorithm (or rather linkage methods in the case of AHC) and string similarity measure; as well as size of the input matrices. The similarity measure was notably influential on the parameters, which was predictable because of major differences in the expression of similarities. For example:

```
levenshtein('duane', 'dwayne') == 2
jaro('duane', 'dwayne') == 0.82
ibm_similarity('duane', 'dwayne') == 0.2
```

5.2 Experiments

As described in section 4, each dataset was transformed into three metric distance matrices, one per string similarity measure. The development-portion was used for this cluster analysis, as these provide a reasonable token count (except for the Swedish GaW Corpus, as already mentioned). Again, only the types are used in the calculations, to ensure equal strings won't skew the CA and to reduce the overall number of calculations required [18]. This yielded distance matrices of the following dimensions:

- German Anselm (development): 7617×7617
- Swedish GaW (test): 8823×8823
- Spanish Post Scriptum (development): 3077×3077
- Portuguese Post Scriptum (development): 6286×6286

It is noteworthy that the IBM similarity uses consonant skeletons, which are created by removing all vowels from a string. This might result in many empty - or very short - strings for the romance languages, which was anticipated in the experiments in order to study the effects of a language-dependent measure. The following is an example distance matrix for the Spanish Post Scriptum dataset using the IBM similarity:

Each of the distance matrices was then used to perform a cluster analysis, using both affinity propagation clustering and agglomerative hierarchical clustering. Parameter tuning for these

[18] All preprocessing and cluster analysis calculations were done on the Google Cloud Platform (n1-standard-8, 8 vCPUs, 30 GB of memory). Preprocessing the largest dataset (roughly 8800 elements) took around 40 minutes. Each cluster analysis took around 5 to 10 minutes, depending on the size of the dataset.

	escapulario	bendicion	hablado	bendita	notiçia
escapulario	0.000000	0.000918	0.001322	0.000230	0.000230
bendicion	0.000918	0.000000	0.000494	0.034294	0.000494
hablado	0.001322	0.000494	0.000000	0.005102	0.000816
bendita	0.000230	0.034294	0.005102	0.000000	0.001276
notiçia	0.000230	0.000494	0.000816	0.001276	0.000000
...	...	...	...	...	...

Figure 20: Example Distance Matrix (IBM, Spanish)

analyses was done in an iterative process. After each calculation with a specific setting, the results were evaluated against the baseline, optimising for the optimal number of clusters as well as an optimal Jaro similarity index. Additionally, an examination of the largest clusters and a visual inspection were performed. This process was repeated until a further approximation to the baseline was no longer possible.

All experiments are publicly available as an interactive Python JupyterLab Notebook [19].

5.3 Results

Overall, the affinity propagation clustering (APC) did not yield satisfactory results. As can be seen in the evaluation results, the algorithm generally identified less than expected clusters, with a higher than expected object count. See Appendix D.1: APC Results. It was suspected that the primary issue with APC is the presence of "clusters" with just one element, of which there are many in each dataset (German: 1149, Swedish: 5366, Spanish: 2106, Portuguese: 3475). This assumption was confirmed by counting the number of these one-element clusters of each APC analysis, which revealed that there are none.

When inspecting the objects in the largest clusters, it was confirmed that the objects are indeed very heterogeneous. For example:

```
# APC|Levenshtein|German
auc, tag, av, al, davō, mas, aws, naẏ, kam
ic, div, mir, gir, iff, ih, mi, mois, iw, si
ÿm, js, mvz, wz, m, zv, zw, xpūs, mȳ, m

# APC|Levenshtein|Swedish
wrächer, lesther, afreser, brytter, probsten
smt, slott, godtt, sm:t, sätet, s:m:tt, sm:tt
wrack, åbohl, richz, hooror, härom, h:f:n
```

Tuning the damping factor did not improve the results and sometimes caused the algorithm to fail entirely, resulting in no clustering at all. As can be seen, the strings in the clusters could be considered similar and there are some spelling variations grouped together. However, this is mainly due to the large clusters containing many strings and is therefore not practical. Overall,

[19] https://github.com/martialblog/master-thesis-code/

the Jaro similarity yielded more sensible results and the strings containing numbers were almost always identified properly. For example:

```
# APC|Jaro|Spanish
servir, escritas, escrivirme, escrivo
alegrare, abrebiar, arebatamiento, aquellas
11, 1773, 1787, 31, 1699, 15, 12, 19, 13, 166

# APC|Jaro|Portuguese
apertadisimamte, fazendosse, asseo, senadores
818, 1576, 17, 1516, 18, 1744, 1773, 1671, 17
3, é, 400$, hè, 5, 28, 641, 20, ẽ, 30, 24, 29
```

In summary these results suggest that affinity propagation clustering is not applicable for identifying spelling variations using string similarity measures. Different similarity measures had very little impact on the overall performance, even across different languages. Finally, the Jaro similarity appears to yield better results when compared to the Levenshtein distance and IBM similarity.

In contrast to this, the agglomerative hierarchical clustering (AHC) did generate very promising results. As can be seen, the algorithm returned a very close approximation to the baseline. See Appendix D.2: AHC Results. It was possible to tune distance threshold parameter accordingly and with very high accuracy. A closer look at the objects within each cluster did confirm these findings. For example:

```
# AHC(Ward)|Levenshtein|German
ÿm, m, mȳ, m, ym, ẏm, my, m, mȳ, m:, mā
ker, zer, ter, ſer, wer, her, oer, eer, fer
vn, ẏn, vnn, vön, byn, nün, yn, tvn, n, syn

# AHC(complete)|Levenshtein|Swedish
då, du, dy, döö, dö, död, d
sm, s:m, sså, små, s, såå, så, s.
th, thå, thy, uth, vthj, åth, uthj, thä, vth

# AHC(single)|Levenshtein|Spanish
yjo, o, 1o, lo, yzo, yo, po
dize, hize, dice, hice, vive, dige, dixe, dije
digo, hico, fio, dixo, hizo, ijo, io, hijo

# AHC(complete)|Levenshtein|Portuguese
àos, bos, hos, pos, áos, dos, fos, gos, os, vos
va, qa, fa, xa, a, ha, hũa, ba, ja, yva
dé, dã, d, du, dá, dũ, dê, d', qd, dó
```

The choice of linkage criterion appears to be a major factor for the performance of the cluster analysis. While the Jaro index suggests that the single linkage offers optimal results, in many cases it resulted in one large cluster containing all objects. This happened regardless of language

and string similarity measure. This outcome was expected to some extent, as Henning [23] states:

> Both Single and Complete Linkage are rather too extreme for many applications, although they may be useful in a few specific cases. Single linkage focuses totally on separation, i.e., keeping the closest points of different clusters apart from each other, and Complete Linkage focuses totally on keeping the largest dissimilarity within a cluster low. Most other hierarchical methods are a compromise between these two extremes.

The average and Ward linkage criteria appear to be a more suitable choice and both yielded very interesting results. This was valid across all examined languages. For example:

```
# AHC(average)|Levenshtein|Spanish
dize, dice, dige, dixe, dije
açer, aber, ayer, acer, aver
dejo, dexo, dego, devo, debo

# AHC(average)|Levenshtein|Swedish
syyn, sön, s:n, syhn, syn, swän, son, spån
kommo, kom4, kom, kom1, komm, kom10, koo, kom20
sm, s:m, sså, små, s, såå, så, s.

# AHC(Ward)|Jaro|Portuguese
sã, sẽ, s., s, sĩ, só
3o, jo, 1o, 4o, 8o, ão
hè, há, hũ, hẽ, hê, hé

# AHC(Ward)|Jaro|German
ieglicher, iegleichē, iegleicher, iegliche
liep, liebẏ, liebw, liebē, liefs, liebi
ſeineu, ſiene, ſienem, ſeine, ſeinen, ſiener
```

As can be seen in the above examples, the Jaro similarity in particular generated very promising results. The same was observed in the APC analyses. Multiple spelling variations were accurately clustered together, as well as inflectional/derivationally related forms. While there are exceptions, mainly regarding shorter words, the following examples confirm that certain spelling variations were identified very accurately:

```
# AHC(complete)|Jaro|German
parmherzichait, barmherczichaÿt, parmherczichait
reine, reinen, beweinen, zweine, weinend
anzhelm , anſelme, anſelm , anſhelmuſz, anſhelm

# AHC(average)|Jaro|German
verteilt, vertaÿlet, viertavſent, viertawſent
reyne, meÿnen, engegen, weynen, meÿnem, eynes
pfening, phennyge, pfenninge, pfennige, pfenning
```

```
# AHC(single)|Jaro|Spanish
asegurar, asegurarlo, assegurar, asegurado
escrivo, escrivi, escrito, escribi, escribe
garcia, grazia, gracia, grazias, gracias

# AHC(complete)|Jaro|Portuguese
dé, dã, d, dá, dũ, dê, d', dó
lembrança, lembrava, lembrase, lembrcas
escrevera, escreverte, escreveme, escreveve
```

It was observed that words with less then 5 characters are oftentimes "wrongly" grouped together, meaning they are not spelling variations but different words. They are, however, too similar and are thus in the same cluster, usually when they differ by a single character. This poses a particular issue with romance languages, as there are many examples for these highly similar words with very few characters. There are, however, also examples for this issue in the germanic languages:

```
# AHC(complete)|Levenshtein|Spanish
cada, dada, nada
hesa, pesa, cesa, vesa
bien, vien, qien, tien

# AHC(complete)|Levenshtein|Swedish
röra, höra, böra, köra, föra, fyra, göra
hade, lade, sade
hoor, koor, boor, noor
```

In contrast to this, spelling variations of longer words are accurately grouped together. When compared to the optimal clusters from the contemporary spelling, a frequent shortcoming was that groups of spelling variations were not in the same cluster. However, it was noted that these spelling variations oftentimes share a similar characteristic and therefore this might also be advantageous, depending on the point of view.

The following examples illustrate this. They show that the combination of Jaro similarity and single linkage produced a highly accurate cluster of spelling variations. On the other hand, when using complete linkage two separate clusters are generated, one containing variations spelled with "tz" and the other spelled with "cz". This also emphasises that it cannot be conclusively said that there is a single best combination of algorithm and string similarity.

```
# AHC(single)|Jaro|German
[parmherczichait, barmhertzigkeÿt, parmherczigkait, barmherczigkeit,
parm̄herczichaid, parmherczichaÿt, parmherczikeit, pam̄herczichaidt,
parmhertzigkait, parmherczickeit, parmherczikait, barmhertzichait,
barmherczikeit, parm̄herczigkaytt, parmhertzichait, parmhertzikait,
barmhertzikeit, parmherczigkeit, parmherzichait, barmherczigkeit,
barmherczichaÿt]
```

```
# AHC(complete)|Jaro|German
[barmhertzigkeÿt, parmhertzigkait, barmhertzichait, parmhertzichait,
parmhertzikait, barmhertzikeit],

[parmherczichait, parmherczigkait, barmherczigkeit, parm̄herczichaid,
parmherczichaÿt, parmherczikeit, pa ̄rmherczichaidt, parmherczickeit,
parmherczikait, barmherczikeit, parm̄herczigkaytt, parmherczigkeit,
parmherzichait, barmherczigkeit,barmherczichaÿt]
```

Similar results were observed across all languages. While these are not always spelling variations, using single linkage tends to generate larger clusters, when compared to other linkage criteria. For example:

```
# AHC(single)|Jaro|Swedish
[enfaldeligen, befallninghz, beffalningsman, enfaldige,
befalning, befallningsmannen, befalningzmansens, befallningzmannenss,
befallning, beffalningh, befallningzman, befallningsman,
eenfalliga, befalningzmannen, befallningzmannen, befalningzmän,
befallningmannen, befalningzmanses, enfalligheet, eenfalligt,
eenfallige, befallningzmanss, eenfaldiga, befallningman]

# AHC(complete)|Jaro|Swedish
[beffalningsman, befallningsmannen, befalningzmansens, befallningzmannenss,
befallningzman, befallningsman, befalningzmannen, befallningzmannen,
befalningzmän, befallningmannen, befalningzmanses, enfalligheet,
befallningzmanss],
[eenfalliga, eenfalligt, eenfallige],
[enfaldeligen, enfaldige, eenfaldiga]

# AHC(single)|Jaro|Spanish
[contente, contexsto, contener, contenidos,
contexto, contenito, contesto, contento,
contenido],

# AHC(complete)|Jaro|Spanish
[contexto, contenito, contento, contenido],
[contente, contenpla, contener],
[contexsto],
[contenidos]
```

Finally, the application of the IBM similarity did not yield desirable results. This is also reflected in the lower Jaro index, providing further confirmation that this evaluation metric is indeed suitable. Except for very few cases, the overall performance of the IBM similarity was not satisfactory in this use case:

```
# AHC(complete)|IBM|Spanish
```

```
flandes, esquadra, deprisa, debates, desnudez
consorte, confianza, consolada, confesionario
1773, 1699, 1661, 1648, 1706, 1711, 1689

# AHC(complete)|IBM|Portuguese
vendonos, deduzindo, enganoume, escandula
privada, prinmo, provimto, profano, prepozito
comfuzão, comculo, commissario, comsoada
```

To summarise, the results suggest that agglomerative hierarchical clustering is generally suitable for the detection of spelling variations. Across all datasets the algorithm grouped together spelling variations and inflectional/derivationally related forms. However, it is obviously not possible for the algorithm to distinguish between these two linguistic aspects. Human inspection and evaluation is therefore essential.

The choice of string similarity measure and linkage criterion is significant for the performance. As was shown, the IBM similarity did not appear to be applicable in this particular use case. In contrast, the Levenshtein distance yielded more accurate groups. Overall, the Jaro similarity returned the best results, in particular when combined with the average or ward linkage criterion. While the single linkage criterion did not always return optimal results, it cannot be conclusively said that it performed worse.

However, regardless of string similarity measure, the presented method appears to not work well with strings of less then 5 characters. This is likely due to them being too similar, as they oftentimes only differ by a single character; and therefore the clustering algorithm groups them together. Nonetheless, the overall results demonstrate that it is possible to achieve highly accurate groups of spelling variations across various languages using unsupervised clustering algorithms.

6 Conclusion

This work investigated how cluster analysis algorithms in combination with string similarity measures can be leveraged to identify historical spelling variations. For this, an initial preliminary analysis was performed on multilingual corpora in order to inspect each dataset, to validate assumptions about the data and to confirm the feasibility of a subsequent cluster analysis (CA).

The basis for all analyses in this work were metric distance matrices created from the textual data; this was done by calculating pairwise string similarities for all types in each dataset, using various algorithms. This work used custom implementations of the Levenshtein distance, Jaro similarity and IBM similarity; the latter with some adaptations for the task at hand.

These matrices were then used as input for both affinity propagation and agglomerative hierarchical clustering, in order to investigate differences in effectiveness. The evaluation of each algorithm was done by using custom performance indicators and visual inspection; for which a custom web tool was implemented. Overall, the focus of this work lied on the use of unsupervised methods.

Returning to the research questions posed at the beginning of this work: can spelling variations within a corpus be made accessible to researchers using unsupervised methods?

The CA results strongly suggest unsupervised clustering can be applied successfully when no labelled training data is available. The choice of algorithm - both for clustering and string similarity - is central to the performance, however. It also is important to state that this method on its own cannot discriminate between synchronic/diachronic spelling variations and inflectional/derivationally related forms. Depending on the application this can be seen as both an advantage and a disadvantage.

Does cluster analysis offer a viable solution for the detection of spelling variations?

As was shown, CA was used to detect spelling variations in a multilingual dataset. While these corpora are moderately sized - when compared to contemporary web corpora - it is reasonable to assume that many historical datasets do not contain billions of tokens. Thus, the reduction to types and the subsequent cluster analysis did result in a manageable amount of groups, which can be analysed by human researchers with moderate effort. The largest dataset in this work was reduced from tens of thousands of tokens to just a few hundred clusters; admittedly, there also was a larger number of clusters with just one element. Nonetheless, human evaluation is vital for the method presented in this work.

Which, if any, algorithms are best suited for the unsupervised detection spelling variations?

In the preliminary analysis, the Levenshtein distance appeared to be best suited for the task at hand. However, the subsequent cluster analysis revealed that the Jaro similarity yielded more satisfactory results. This means that the choice of string similarity measure needs to be taken into account when trying to detect spelling variations in an unsupervised manner. This was also confirmed by the overall poor performance of the IBM similarity in this task.

From the two clustering algorithms under investigation, agglomerative hierarchical cluster ap-

peared to be very well suited for the detection of spelling variations; while affinity propagation clustering did not yield suitable results. The central issue with APC was identified to be the large amount of "clusters" with just one element in the dataset; meaning many words without any variations, that do not need to be grouped together. Therefore it can be assumed that clustering algorithms that are unable to process such a distribution will perform worse at this task.

In summary, the presented method may not be sufficient for a fully automated retrieval of spelling variations but might significantly aid human researchers with this task, as it can accelerate the analysis of historical corpus data. Furthermore, the method in this work should also be able to augment modernisation efforts; similar to the approach presented by Barteld [6]. More specifically, it might be applied when there is a need to generate possible candidates for modernisation. When no labelled training data is available, it can be used to generate these candidates in an unsupervised manner. Even without a fully automated candidate generation, the presented method can quickly yield a manageable amount of groups for manual annotation.

Furthermore, this work presented various evaluation methods for the clustering of spelling variations. These are: first, a similarity index based on the Jaro similarity, the effectiveness of which has been shown in the CA experiments; and second, an open source tool for visual inspection. Both of them are also applicable when no labelled data for evaluation is available.

Finally, there are several possibilities for future research to expand upon this work. One obvious example would be a detailed investigation of further string similarity measures. Of particular interest would be how weighted edit distances might affect the performance, which could better adapt the method to a particular language or time period. Additionally, it might be of interest to investigate if and how context information can be included in the cluster analysis. This might be a successful strategy when trying to discriminate between synchronic/diachronic spelling variations and inflectional/derivationally related forms.

Appendices

A String Similarity Measures

A.1 Levenshtein Distance

```
def levenshtein(string1, string2):
    if string1 == string2:
        return 0

    if not string2:
        return len(string1)
    if not string1:
        return len(string2)

    rows = len(string1) + 1
    cols = len(string2) + 1
    dist = [[0 for c in range(cols)] for r in range(rows)]

    for j in range(1, rows):
        dist[j][0] = j
    for i in range(1, cols):
        dist[0][i] = i

    for col in range(1, cols):
        for row in range(1, rows):
            cost = 1
            if string1[row - 1] == string2[col - 1]:
                cost = 0
            dist[row][col] = min(
              dist[row - 1][col] + 1,
              dist[row][col - 1] + 1,
              dist[row - 1][col - 1] + cost
            )

    return dist[row][col]

assert levenshtein('', '') == 0
assert levenshtein('foobar', '') == 6
assert levenshtein('foobar', 'foobar') == 0
assert levenshtein('foobar', 'foubar') == 1
assert levenshtein('foobar', 'fuubar') == 2
```

A.2 Jaro Similarity

```
def jaro(string1, string2):
    length1 = len(string1)
    length2 = len(string2)
    if length1 == 0:
        return 0.0
    if string1 == string2:
        return 1.0

    match_bound = max(length1, length2) // 2 - 1
    matches = 0
    transpositions = 0
    flagged_1, flagged_2 = [], []

    for i in range(length1):
        upperbound = min(i + match_bound, length2 - 1)
        lowerbound = max(0, i - match_bound)
        for j in range(lowerbound, upperbound + 1):
            if string1[i] == string2[j] and j not in flagged_2:
                matches += 1
                flagged_1.append(i)
                flagged_2.append(j)
                break

    flagged_2.sort()
    for i, j in zip(flagged_1, flagged_2):
        if string1[i] != string2[j]:
            transpositions += 1

    if matches == 0:
        return 0.0

    return (1/3 *
      ( matches / length1 + matches / length2 +
        (matches - transpositions // 2) / matches)
      )

assert jaro('', '') == 0.0
assert jaro('foobar', '') == 0.0
assert jaro('foobar', 'foobar') == 1.0
assert jaro('foobar', 'barfoo') == 0.444
```

A.3 IBM Similarity

```
from itertools import groupby

def lcs_ratio(string1, string2):
    if not string1 or not string2:
        return 0.0
    ratio = longest_common_string(string1, string2) / len(string1)
    return ratio

assert lcs_ratio('', '') == 0.0
assert lcs_ratio('foo', '') == 0.0
assert lcs_ratio('foobar', 'foobar') == 1.0
assert lcs_ratio('foo', 'bar') == 0.0
assert lcs_ratio('word', 'deoxyribonucleic') == 0.25

def consonant_skeleton(string, vowels='aeiouy'):
    without_vowels = ''.join([char for char in string if char not in vowels])
    deduplicated_consonants = ''.join(
      char for char, _ in groupby(without_vowels))
    return deduplicated_consonants

assert consonant_skeleton('') == ''
assert consonant_skeleton('aeio') == ''
assert consonant_skeleton('foobar') == 'fbr'
assert consonant_skeleton('ffoobbar') == 'fbr'

def ibm_similarity(string1, string2):
    similarity = lcs_ratio(string1, string2) / (
     levenshtein(
       consonant_skeleton(string1),
       consonant_skeleton(string2)) + 1)
    return similarity

assert ibm_similarity('', '') == 0.0
assert ibm_similarity('foobar', '') == 0.0
assert ibm_similarity('foobar', 'foobar') == 1.0
assert ibm_similarity('foo', 'bar') == 0.0
assert ibm_similarity('foobar', 'aeiou') == 0.0416
```

B Preliminary Corpus Analysis

Dataset: german-anselm	historical	modernised
Token count	45.993	45.993
Type count	7617	2361
Type/Token ratio	16.56%	5.13%
Hapax legomenon count	4572	966

Table 3: Preliminary analysis: german-anselm (dev)

Dataset: swedish-gaw	historical	modernised
Token count	29.460	29.460
Type count	8829	6680
Type/Token ratio	30.13%	22.81%
Hapax legomenon count	6070	4134

Table 4: Preliminary analysis: swedish-gaw (test)

Dataset: spanish-ps	historical	modernised
Token count	11.680	11.680
Type count	3077	2546
Type/Token ratio	26.34%	21.80%
Hapax legomenon count	2158	1616

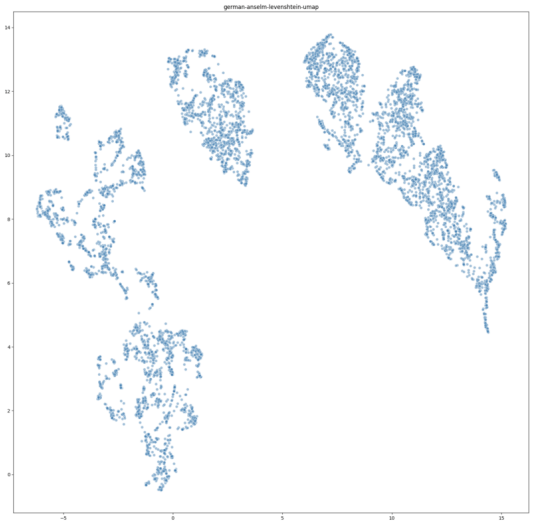
Table 5: Preliminary analysis: spanish-ps (dev)

Dataset: portuguese-ps	historical	modernised
Token count	26.854	26.854
Type count	6286	4576
Type/Token ratio	23.41%	17.02%
Hapax legomenon count	4290	2765

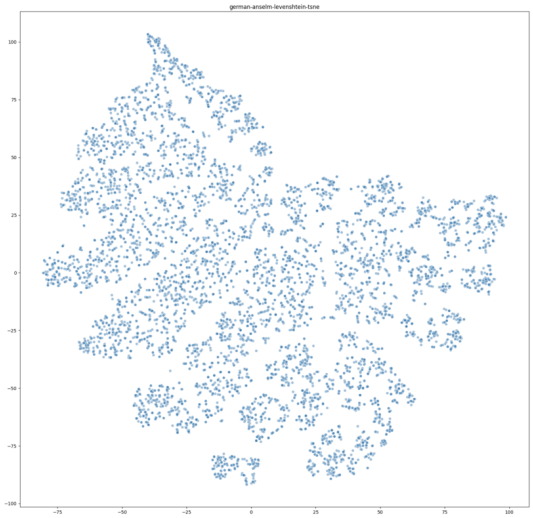
Table 6: Preliminary analysis: portuguese-ps (dev)

C Preliminary Cluster Analysis

C.1 german-anselm

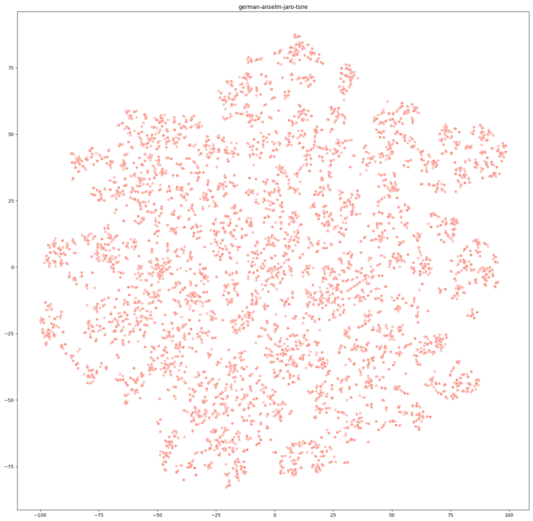


(a) Levenshtein-UMAP

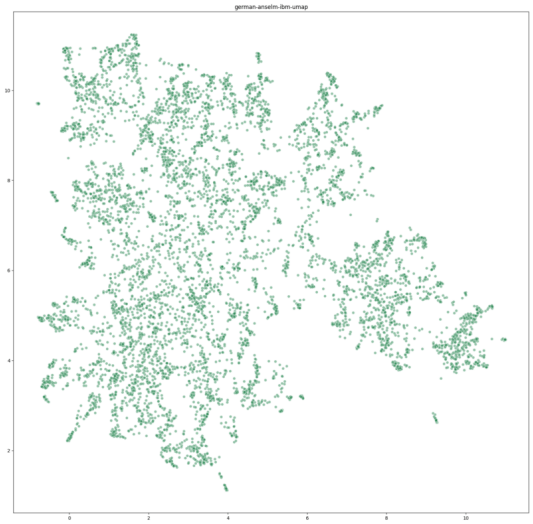


(b) Levenshtein-TSNE

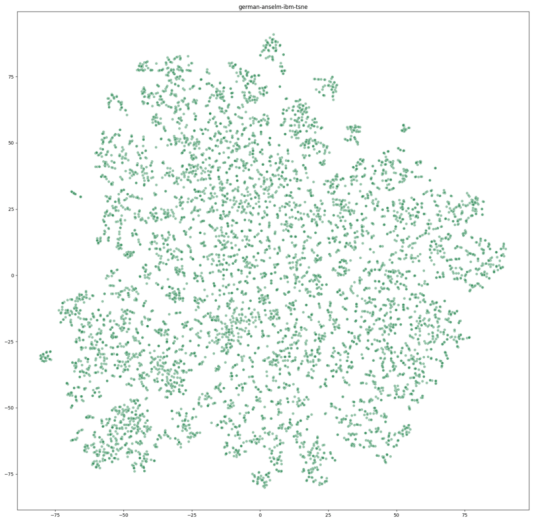
(c) Jaro-UMAP



(d) Jaro-TSNE



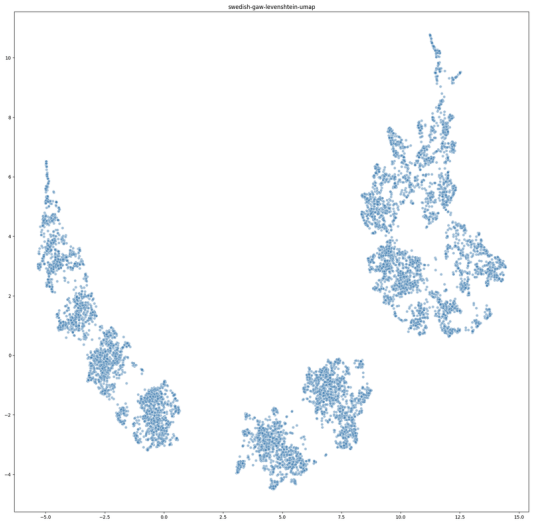
(e) IBM-UMAP



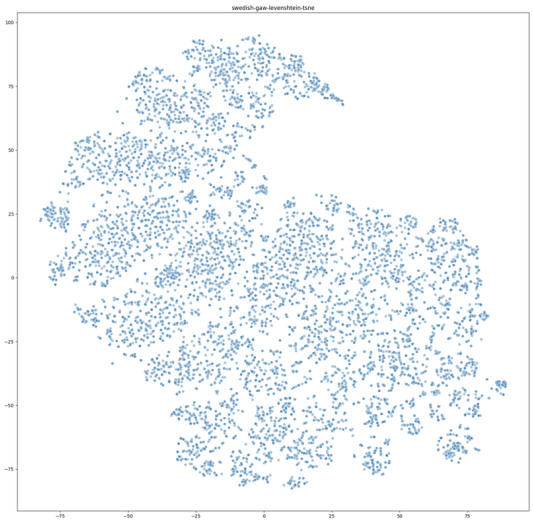
(f) IBM-TSNE

Figure 21: Exploratory clustering using dimensionality reduction (german-anselm)

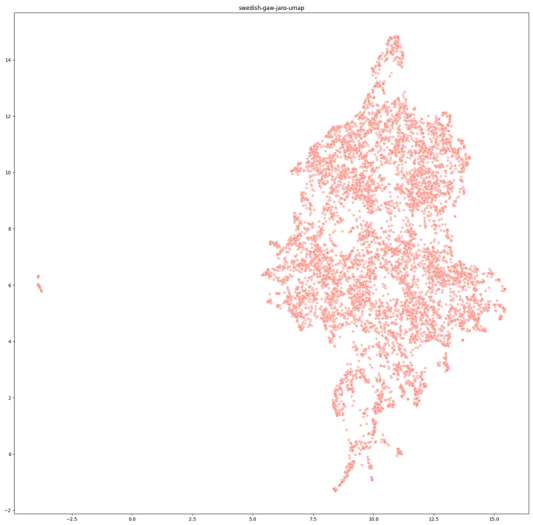
C.2 swedish-gaw



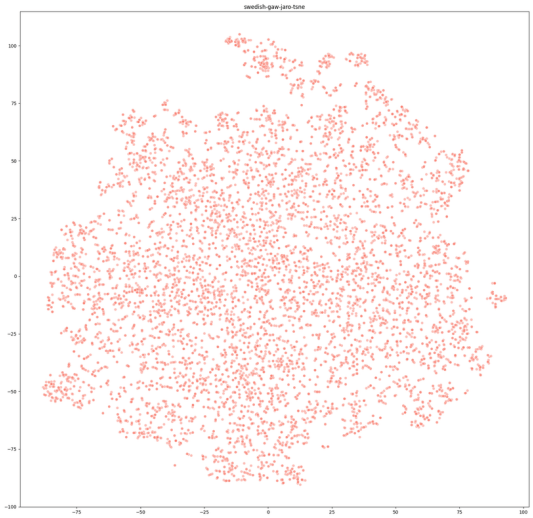
(a) Levenshtein-UMAP



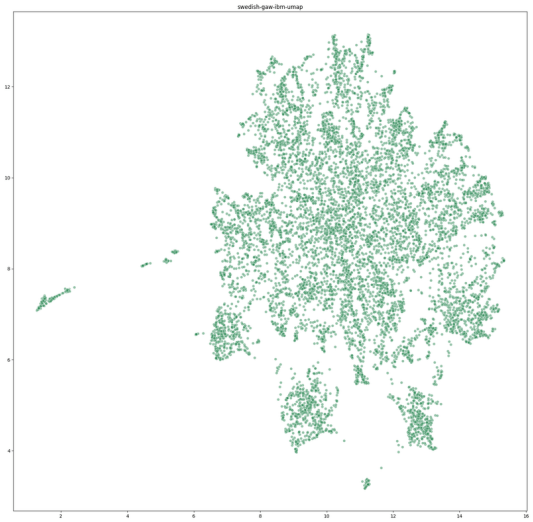
(b) Levenshtein-TSNE



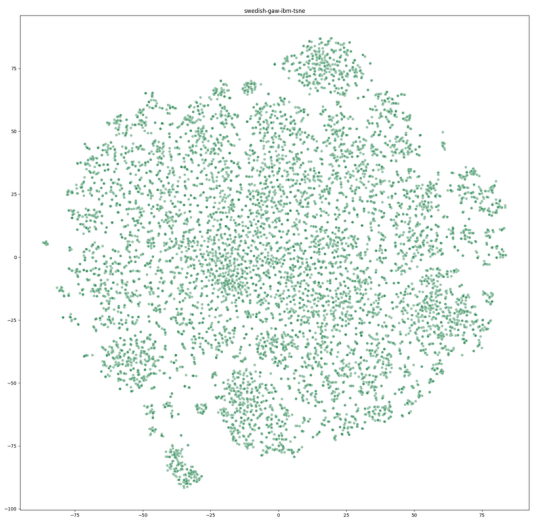
(c) Jaro-UMAP



(d) Jaro-TSNE



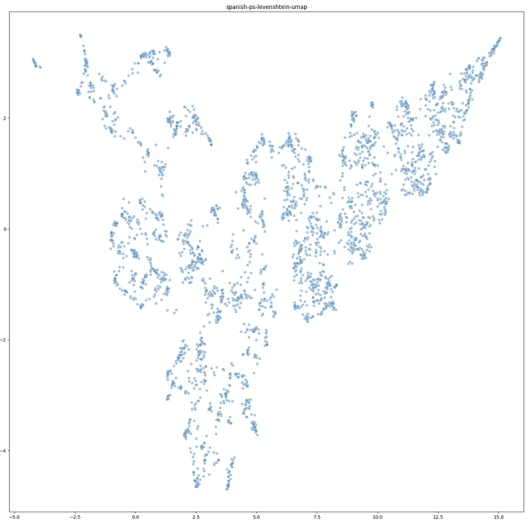
(e) IBM-UMAP



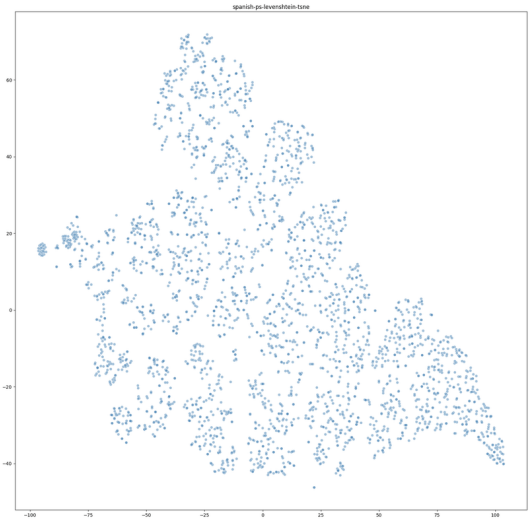
(f) IBM-TSNE

Figure 22: Exploratory clustering using dimensionality reduction (swedish-gaw)

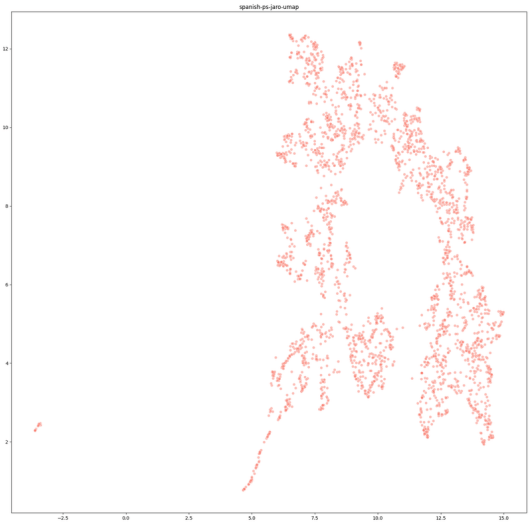
C.3 spanish-ps



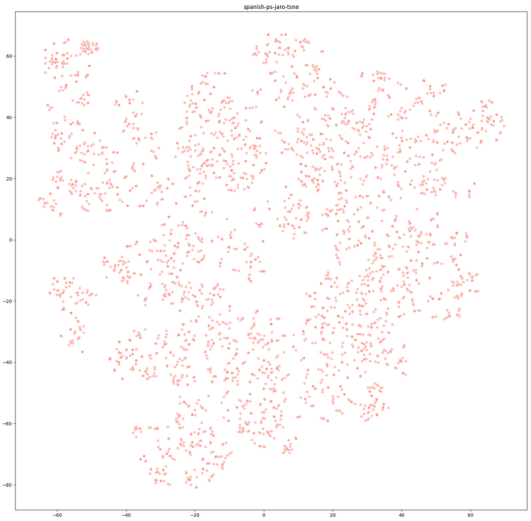
(a) Levenshtein-UMAP



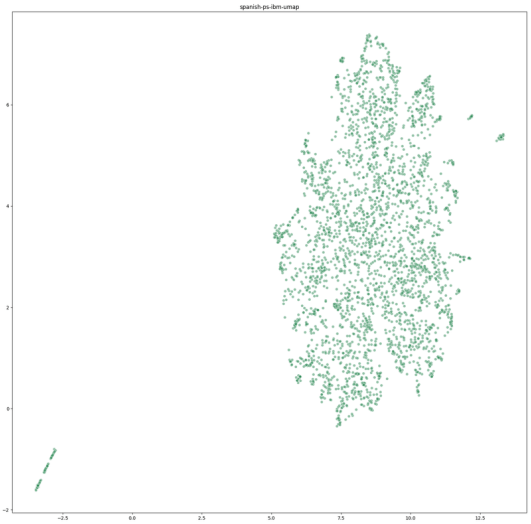
(b) Levenshtein-TSNE



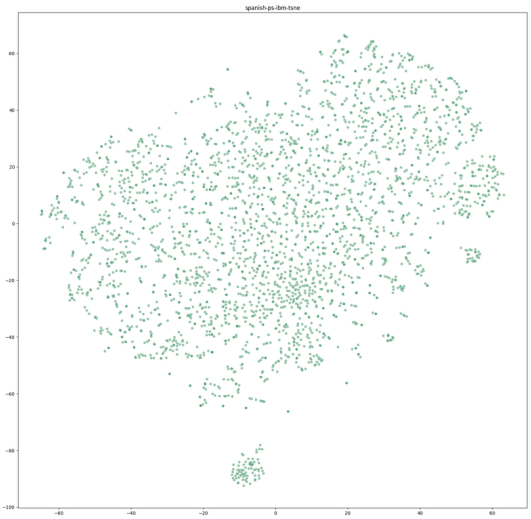
(c) Jaro-UMAP



(d) Jaro-TSNE



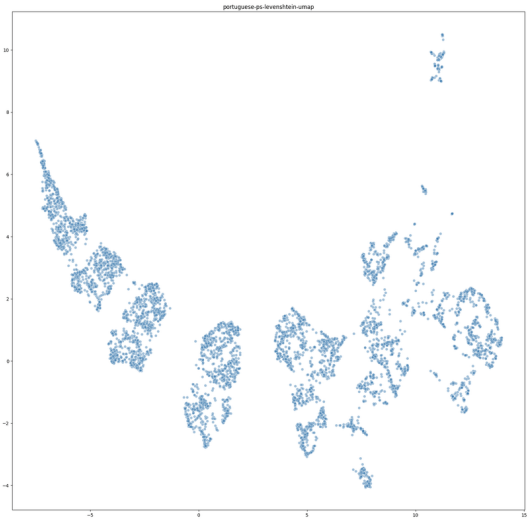
(e) IBM-UMAP



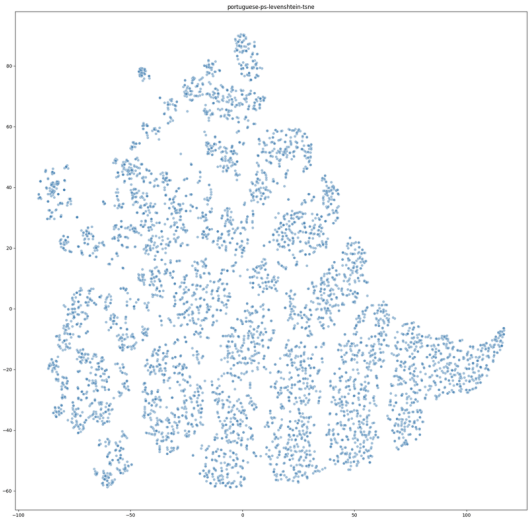
(f) IBM-TSNE

Figure 23: Exploratory clustering using dimensionality reduction (spanish-ps)

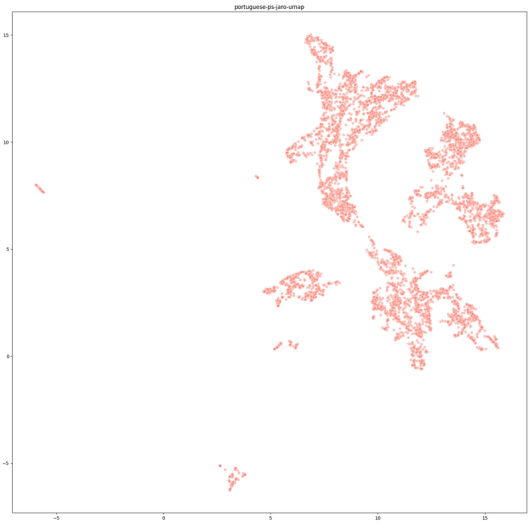
C.4 portuguese-ps



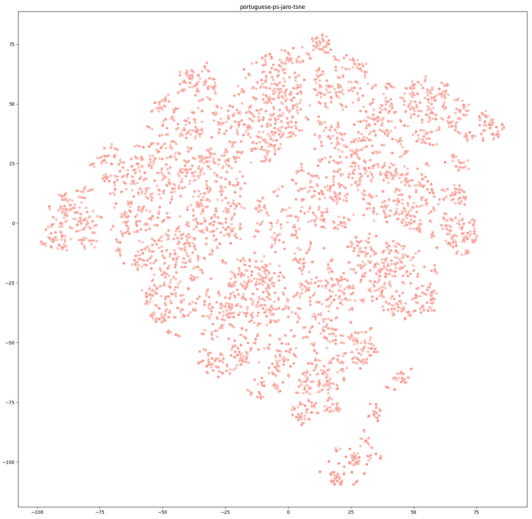
(a) Levenshtein-UMAP



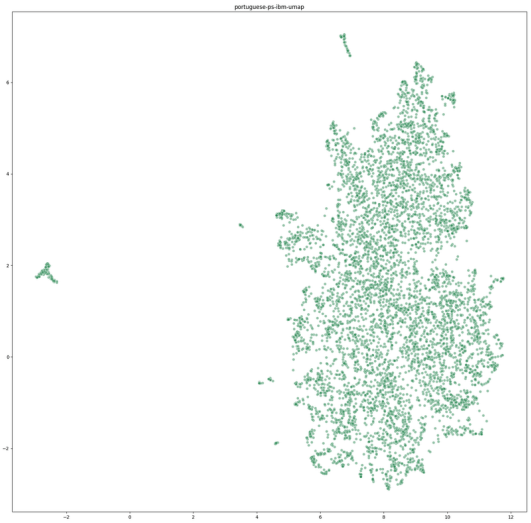
(b) Levenshtein-TSNE



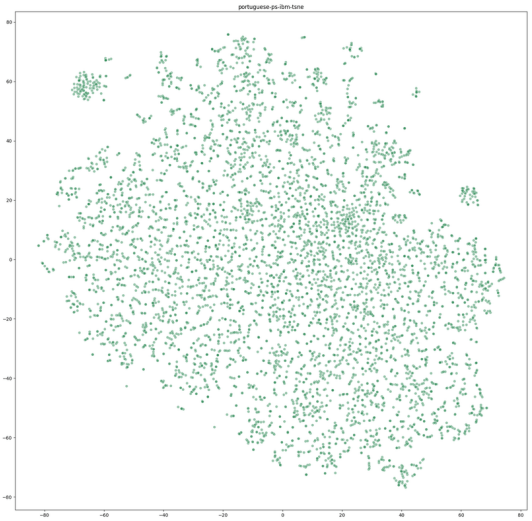
(c) Jaro-UMAP



(d) Jaro-TSNE



(e) IBM-UMAP



(f) IBM-TSNE

Figure 24: Exploratory clustering using dimensionality reduction (portuguese-ps)

D Cluster Analysis Results

- No. Clusters: Number of clusters.
- Avg. Objects: Average number of objects within the clusters.
- Jaro Index (Mean): Arithmetic mean of pairwise Jaro similarity of cluster objects.
- Jaro Index (SD): Standard deviation of pairwise Jaro similarity of cluster objects.

D.1 Affinity Propagation

german-anselm	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	2361	3.50	0.80	0.10
Levenshtein	425	17.92	0.68	0.08
Jaro	622	12.24	0.73	0.07
IBM	1105	6.89	0.77	0.14

Table 7: Affinity Propagation: german-anselm.

swedish-gaw	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	6680	1.35	0.88	0.08
Levenshtein	448	19.69	0.62	0.07
Jaro	845	10.44	0.72	0.08
IBM	1794	4.91	0.76	0.12

Table 8: Affinity Propagation: swedish-gaw

spanish-ps	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	2546	1.25	0.80	0.17
Levenshtein	204	15.08	0.64	0.08
Jaro	279	11.02	0.69	0.09
IBM	644	4.86	0.71	0.13

Table 9: Affinity Propagation: spanish-ps

portuguese-ps	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	4576	1.42	0.82	0.13
Levenshtein	368	17.03	0.39	0.05
Jaro	491	12.80	0.39	0.06
IBM	1012	6.21	0.38	0.12

Table 10: Affinity Propagation: portuguese-ps

D.2 Agglomerative Hierarchical Clustering

german-anselm	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	2361	3.50	0.80	0.10
Levenshtein (single)	2349	3.24	0.85	0.07
Levenshtein (complete)	2244	3.39	0.80	0.09
Levenshtein (average)	2312	3.29	0.81	0.09
Levenshtein (ward)	2461	3.09	0.80	0.09
Jaro (single)	2314	3.29	0.85	0.08
Jaro (complete)	2302	3.30	0.82	0.08
Jaro (average)	2369	3.21	0.83	0.07
Jaro (ward)	2348	3.24	0.83	0.08
IBM (single)	2384	3.19	0.78	0.15
IBM (complete)	2528	3.01	0.80	0.12
IBM (average)	2632	2.89	0.81	0.12
IBM (ward)	2707	2.81	0.79	0.12

Table 11: Agglomerative Hierarchical Clustering: german-anselm

swedish-gaw	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	6680	1.35	0.88	0.08
Levenshtein (single)	6436	1.37	0.87	0.06
Levenshtein (complete)	6671	1.32	0.85	0.08
Levenshtein (average)	6705	1.31	0.86	0.08
Levenshtein (ward)	6726	1.31	0.85	0.09
Jaro (single)	6619	1.33	0.90	0.05
Jaro (complete)	6492	1.35	0.89	0.06
Jaro (average)	6636	1.32	0.90	0.06
Jaro (ward)	6836	1.29	0.90	0.06
IBM (single)	6186	1.42	0.78	0.21
IBM (complete)	6561	1.34	0.75	0.13
IBM (average)	6361	1.38	0.77	0.12
IBM (ward)	6681	1.32	0.73	0.13

Table 12: Agglomerative Hierarchical Clustering: swedish-gaw

spanish-ps	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	2546	1.20	0.80	0.17
Levenshtein (single)	2542	1.16	0.81	0.17
Levenshtein (complete)	2513	1.22	0.81	0.17
Levenshtein (average)	2598	1.18	0.80	0.17
Levenshtein (ward)	2635	1.16	0.80	0.17
Jaro (single)	2612	1.17	0.88	0.06
Jaro (complete)	2554	1.20	0.88	0.06
Jaro (average)	2521	1.22	0.88	0.06
Jaro (ward)	2565	1.19	0.88	0.07
IBM (single)	2665	1.15	0.66	0.13
IBM (complete)	2574	1.19	0.71	0.14
IBM (average)	2522	1.22	0.72	0.14
IBM (ward)	2602	1.18	0.70	0.13

Table 13: Agglomerative Hierarchical Clustering: spanish-ps

portuguese-ps	No. Clusters	Avg. Objects	Jaro Index (Mean)	Jaro Index (SD)
Baseline	4576	1.42	0.82	0.13
Levenshtein (single)	4574	1.37	0.84	0.09
Levenshtein (complete)	4444	1.41	0.83	0.10
Levenshtein (average)	4755	1.32	0.84	0.10
Levenshtein (ward)	4482	1.40	0.83	0.10
Jaro (single)	4576	1.37	0.86	0.07
Jaro (complete)	4442	1.41	0.86	0.07
Jaro (average)	4599	1.36	0.86	0.06
Jaro (ward)	4598	1.36	0.86	0.07
IBM (single)	4586	1.37	0.81	0.13
IBM (complete)	4641	1.35	0.73	0.11
IBM (average)	4491	1.39	0.75	0.11
IBM (ward)	4734	1.32	0.73	0.11

Table 14: Agglomerative Hierarchical Clustering: portuguese-ps

References

[1] Amoia, M., and Martinez, J. M. Using comparable collections of historical texts for building a diachronic dictionary for spelling normalization. In *Proceedings of the 7th workshop on language technology for cultural heritage, social sciences, and humanities* (2013), pp. 84–89.

[2] Aw, A., Zhang, M., Xiao, J., and Su, J. A phrase-based statistical model for SMS text normalization. In *Proceedings of the COLING/ACL on Main conference poster sessions* (2006), Association for Computational Linguistics, pp. 33–40.

[3] Azawi, M. A., Afzal, M. Z., and Breuel, T. M. Normalizing historical orthography for OCR historical documents using LSTM. In *Proceedings of the 2nd International Workshop on Historical Document Imaging and Processing* (2013), pp. 80–85.

[4] Bar-Hillel, Y. The present status of automatic translation of languages. *Advances in computers 1*, 1 (1960), 91–163.

[5] Baron, A., and Rayson, P. VARD2: A tool for dealing with spelling variation in historical corpora. In *Postgraduate conference in corpus linguistics* (2008).

[6] Barteld, F. Detecting spelling variants in non-standard texts. In *Proceedings of the student research workshop at the 15th conference of the European chapter of the association for computational linguistics* (2017), pp. 11–22.

[7] Barteld, F., Schröder, I., and Zinsmeister, H. Unsupervised regularization of historical texts for POS tagging. In *Corpus-Based Research in the Humanities (CRH)* (2015), pp. 3–12.

[8] Biber, D. *Variation across speech and writing*. Cambridge University Press, 1988.

[9] Bisson, G., and Blanch, R. Improving visualization of large hierarchical clustering. In *2012 16th International Conference on Information Visualisation* (2012), IEEE, pp. 220–228.

[10] Bollmann, M. automatic normalization of historical texts using distance measures and the Norma tool. In *Proceedings of the second workshop on annotation of corpora for research in the humanities (ACRH-2), Lisbon, Portugal* (2012), pp. 3–14.

[11] Bollmann, M. A large-scale comparison of historical text normalization systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (2019), pp. 3885–3898.

[12] Bollmann, M., Bingel, J., and Søgaard, A. Learning attention for historical text normalization by learning to pronounce. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017), pp. 332–344.

[13] Bollmann, M., Petran, F., and Dipper, S. Rule-based normalization of historical texts. In *Proceedings of the Workshop on Language Technologies for Digital Humanities and Cultural Heritage* (2011), pp. 34–42.

[14] Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. A statistical approach to machine translation. *Computational linguistics 16*, 2 (1990), 79–85.

[15] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[16] Carstensen, K.-U., Ebert, C., Ebert, C., Jekat, S., Langer, H., and Klabunde, R. *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Springer-Verlag, 2009.

[17] Contractor, D., Faruquie, T. A., and Subramaniam, L. V. Unsupervised cleansing of noisy text. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters* (2010), Association for Computational Linguistics, pp. 189–196.

[18] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (2019), pp. 4171–4186.

[19] Fix, H. Automatische Normalisierung - Vorarbeit zur Lemmatisierung eines diplomatischen altisländischen Textes. In *Maschinelle Verarbeitung altdeutscher Texte. Beiträge zum dritten Symposion, Tübingen 17.–19. Februar 1977* (1980), pp. 92–100.

[20] Frey, B. J., and Dueck, D. Clustering by passing messages between data points. *science 315*, 5814 (2007), 972–976.

[21] Halevy, A., Norvig, P., and Pereira, F. The unreasonable effectiveness of data. *IEEE Intelligent Systems 24*, 2 (2009), 8–12.

[22] Hennig, C. A method for visual cluster validation. In *Classification—The ubiquitous challenge* (2005), Springer, pp. 153–160.

[23] Hennig, C., Meila, M., Murtagh, F., and Rocci, R. *Handbook of cluster analysis*. CRC Press, 2015.

[24] Jaro, M. A. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association 84*, 406 (1989), 414–420.

[25] Johnson, B., and Shneiderman, B. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceeding Visualization'91* (1991), IEEE, pp. 284–291.

[26] Jolliffe, I. T., and Cadima, J. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 374*, 2065 (2016), 20150202.

[27] Jurish, B. More than words: using token context to improve canonicalization of historical German. *JLCL 25*, 1 (2010), 23–39.

[28] Kempken, S., Luther, W., and Pilz, T. Comparison of distance measures for historical spelling variants. In *IFIP International Conference on Artificial Intelligence in Theory and Practice* (2006), Springer, pp. 295–304.

[29] Koller, G. Ein maschinelles Verfahren zur Normalisierung altdeutscher Texte. In *Germanistik in Erlangen* (1983), pp. 611–620.

[30] Korchagina, N. Normalizing medieval German texts: from rules to deep learning. In *Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language* (2017), pp. 12–17.

[31] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, pp. 707–710.

[32] Maaten, L. v. d., and Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research 9*, Nov (2008), 2579–2605.

[33] Manning, C. D., Manning, C. D., and Schütze, H. *Foundations of statistical natural language processing*. MIT press, 1999.

[34] McInnes, L., Healy, J., Saul, N., and Grossberger, L. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software 3*, 29 (2018), 861.

[35] Navarro, G. A guided tour to approximate string matching. *ACM computing surveys (CSUR) 33*, 1 (2001), 31–88.

[36] Pettersson, E. *Spelling normalisation and linguistic analysis of historical text for information extraction*. PhD thesis, Doctoral dissertation, Acta Universitatis Upsaliensis, 2016.

[37] Pettersson, E., Megyesi, B., and Nivre, J. Normalisation of historical text using context-sensitive weighted Levenshtein distance and compound splitting. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013); May 22-24; 2013; Oslo University; Norway. NEALT Proceedings Series 16* (2013), no. 085, Linköping University Electronic Press, pp. 163–179.

[38] Pettersson, E., Megyesi, B., and Nivre, J. A multilingual evaluation of three spelling normalisation methods for historical text. In *Proceedings of the 8th workshop on language technology for cultural heritage, social sciences, and humanities (LaTeCH)* (2014), pp. 32–41.

[39] Piotrowski, M. Natural language processing for historical texts. *Synthesis lectures on human language technologies 5*, 2 (2012), 1–157.

[40] Porta, J., Sancho, J.-L., and Gómez, J. Edit transducers for spelling variation in Old Spanish. In *Proceedings of the workshop on computational historical linguistics at NODALIDA 2013; May 22-24; 2013; Oslo; Norway. NEALT Proceedings Series 18* (2013), no. 087, Linköping University Electronic Press, pp. 70–79.

[41] Raghu, M., and Schmidt, E. A survey of deep learning for scientific discovery. *arXiv preprint arXiv:2003.11755* (2020).

[42] Rayson, P., Archer, D., and Smith, N. VARD versus WORD: A comparison of the UCREL variant detector and modern spellcheckers on English historical corpora. *Corpus Linguistics 2005* (2005).

[43] Richards, B. Type/token ratios: what do they really tell us? *Journal of child language 14*, 2 (1987), 201–209.

[44] Roux, M. A comparative study of divisive and agglomerative hierarchical clustering algorithms. *Journal of Classification 35*, 2 (2018), 345–366.

[45] Scherrer, Y., and Ljubešić, N. Automatic normalisation of the Swiss German ArchiMob corpus using character-level machine translation. In *Proceedings of the 13th Conference on Natural Language Processing (KONVENS)* (2016).

[46] Sennrich, R., Haddow, B., and Birch, A. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2016), pp. 86–96.

[47] Soboroff, I., and Tong, A. NLP in low-resource languages preface. *Machine Translation 32*, 1-2 (2018).

[48] Weaver, W. Translation. *Machine translation of languages 14* (1955), 15–23.

[49] Weizenbaum, J. ELIZA - a computer program for the study of natural language communication between man and machine. *Communications of the ACM 9*, 1 (1966), 36–45.

[50] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[51] Xu, D., and Tian, Y. A comprehensive survey of clustering algorithms. *Annals of Data Science 2*, 2 (2015), 165–193.

[52] You, Y., Li, J., Hseu, J., Song, X., Demmel, J., and Hsieh, C.-J. Reducing BERT pre-training time from 3 days to 76 minutes. *arXiv preprint arXiv:1904.00962* (2019).

[53] Young, T., Hazarika, D., Poria, S., and Cambria, E. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine* (2018).

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte Hilfsmittel verfasst habe.

Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle wörtlich oder dem Sinn nach aus anderen Texten entnommenen Stellen als solche kenntlich gemacht. Dies gilt für gedruckte Texte wie für Texte aus dem Internet.

Alle Stellen und Personen, welche mich bei der Vorbereitung und Anfertigung der Abhandlung unterstützten, habe ich genannt.

Die Arbeit wurde in der vorliegenden bzw. modifizierten Form noch keiner anderen Stelle zur Prüfung vorgelegt und dieselbe hat auch nicht anderen Zwecken - auch nicht teilweise - gedient. Mit einer Plagiatsprüfung bin ich einverstanden.

Mir ist bewusst, dass jeder Verstoß gegen diese Erklärung eine Bewertung der eingereichten Arbeit mit Note „nicht bestanden“ zur Folge hat.

Nürnberg, 20. Januar 2021 Markus Opolka