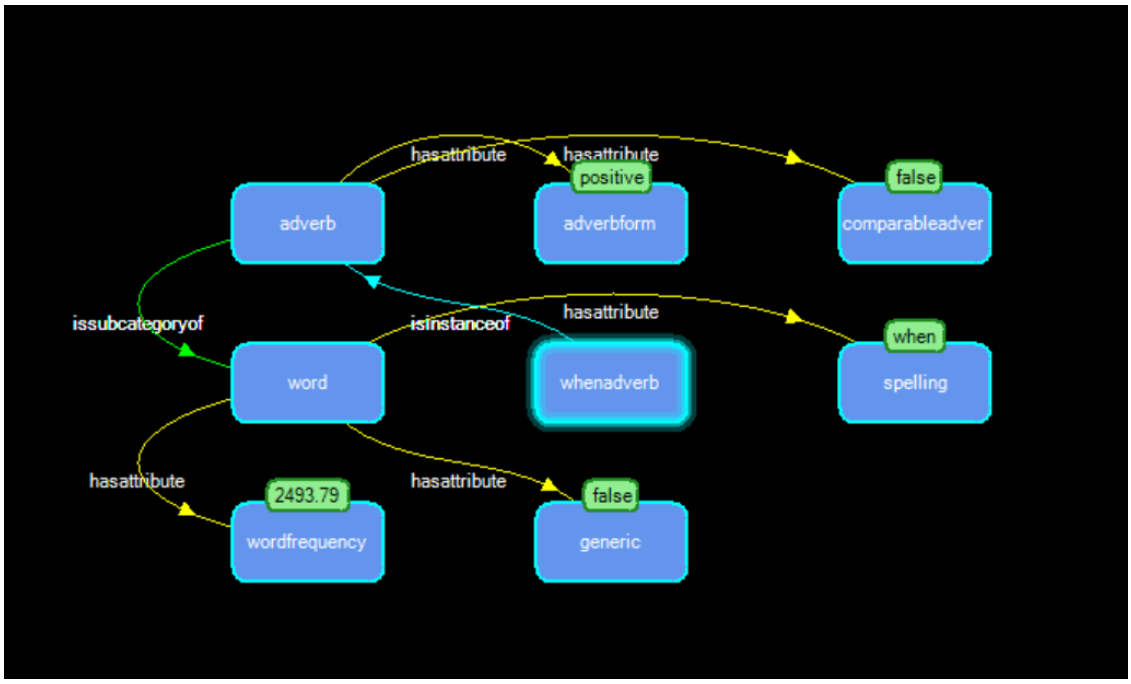
CHALMERS
UNIVERSITY OF TECHNOLOGY



Implementation of interpretable methods for paraphrasing and text disambiguation

Master's thesis in Engineering Mathematics and Computational Science

KLARA CARLSTRÖM

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
DIVISION OF VEHICLE ENGINEERING AND AUTONOMOUS SYSTEMS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis in Engineering Mathematics and Computational Science

Implementation of interpretable methods for paraphrasing and text disambiguation

KLARA CARLSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Applied Artificial Intelligence group
Chalmers University of Technology
Gothenburg, Sweden 2023

Implementation of interpretable methods for paraphrasing and text disambiguation
KLARA CARLSTRÖM

Examiner and supervisor:
Mattias Wahde, Department of Mechanics and Maritime Sciences

Master's Thesis
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Applied Artificial Intelligence group
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: The image shows a small part of the knowledge-graph based language model, the implementation of which is presented in this thesis. The node corresponding to the adverb *when* is shown in the centre of the image, surrounded by the closest neighbouring nodes.

Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Implementation of interpretable methods for paraphrasing and text disambiguation
Master's thesis in Engineering Mathematics and Computational Science
KLARA CARLSTRÖM
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

In this project, starting from an interpretable language model based on knowledge graphs, four essential methods for natural language processing (NLP) have been developed, namely (i) paraphrasing, (ii) part-of-speech tagging, (iii) semantic similarity analysis, and (iv) text simplification. The methods yield good results on a small dataset and thus offer promising prospects for continuing research on interpretable NLP. Applications of NLP are becoming increasingly embedded in our daily lives in applications such as voice assistants, automatic language translation, opinion mining and medical diagnostics. One of the reasons behind the exponentially growing interest in NLP is the development of deep neural network (DNN) models that have achieved outstanding performance on various NLP tasks. However, the domination of DNN models has been followed by deep concerns regarding the black-box nature of such systems. By contrast, the language model used here is fully interpretable, paving the way for safe and accountable NLP.

Keywords: natural language processing, conversational AI, interpretable AI, paraphrasing, text disambiguation, knowledge graphs

Acknowledgements

Thanks to my supervisor Mattias Wahde for providing excellent guidance throughout my thesis project and for introducing me to the immensely interesting topic of natural language processing and intelligent agents.

Klara Carlström, Gothenburg, January 2023

Contents

1
Introduction

> *"It is hard from the standpoint of the child, who must spend many years acquiring a language... it is hard for the adult language learner, it is hard for the scientist who attempts to model the relevant phenomena, and it is hard for the engineer who attempts to build systems that deal with natural language input or output. These tasks are so hard that Turing could rightly make fluent conversation in natural language the centerpiece of his test for intelligence."*
>
> – Page 248, *Mathematical Linguistics, 2007* [1].

Natural language processing (NLP) is the broad and interdisciplinary field that deals with enabling machines to process and generate human language. In recent years, research within NLP has seen a notable upsurge and is now becoming increasingly embedded in our daily lives [2] [3] [4]. Some applications of NLP include voice assistants [5], automatic language translation [6] [7], email spam detection [6], text summarization [8], conversational agents [9], opinion mining [10], and medical diagnostics research [11].

Due to the unstructured nature of human language, NLP is inherently hard and despite the extensive use of NLP in various applications, it is still very much an open area of research with many unresolved problems [8]. In recent years the introduction of deep learning into the field of NLP has propelled significant advances, and deep neural network (DNN) models have consequently come to almost completely dominate the field [12] [13] [14]. However, due to the black-box nature of such models, an increasing number of researchers have raised concerns regarding their use in high-stakes situations where a high degree of accountability and transparency is required, such as in decision-making that may deeply affect peoples' lives [14] [15] [16]. The lack of transparency and accountability of black-box models have started to receive more attention lately and in an attempt to alleviate these problems efforts have been made to create separate posthoc models that *explain* the black-box models [17]. This means that a second model is built such that it replicates the behaviour of the first model as closely as possible. There are, however, numerous problems with this approach. A model that replicates behaviour only provides an approximation of the original model, and these approximations are often not reliable and can be misleading [18] [17] . Thus, it is crucial to put more effort into building models that are inherently *interpretable*, i.e. models in which the decision-making process consists of interpretable primitives that are human-comprehensible, and thus provide their own explanation [17].

An example where transparency is of utmost importance is in the case of task-oriented agents, which are conversational agents that need to be able to carry out precise and consistent interaction with users on specific and sometimes critical subjects [19]. As a contribution to solving this issue, a recent example of an inherently interpretable dialogue manager is DAISY [20] [21], the development of which has been motivated by the following points: (i) it is crucial that a conversational system should be comprehensible to developers such that it is possible to troubleshoot errors and modify or expand the system, (ii) it is of equal importance that a user should be able to query the system and receive a clear description of why a certain decision was made. Moreover, it is important to (iii) avoid dependency on large amounts of training data, and (iv) to alleviate the risk of incorporating unwanted biases in the system (which is often a consequence of using large amounts of uncurated training data, see 2.3) [21].

In recent work, and as an extension of the DAISY dialogue manager mentioned above, Wahde et al. have proposed a framework for an interpretable knowledge-graph based language model based on the English language [22]. The work in this thesis mainly consists of improving and expanding the skeleton language model introduced by Wahde, and to develop four interpretable methods for common NLP tasks, namely (i) part-of-speech tagging, (ii) text simplification, (iii) semantic similarity analysis, and (iv) paraphrasing.

1.1 Aims and objectives

The overall aims of the thesis are to (i) build an interpretable knowledge graph-based language model that represents the English language and (ii) use it in developing reliable and interpretable methods for natural language processing tasks. The intention is to construct a proof of concept by demonstrating the developed methods on a limited set of questions pertaining to the Nobel prize.

1.1.1 First objective

The first objective is to improve and expand the existing language model proposed by Wahde (see above). Automatic methods will be developed to build the model such that it easily can be expanded to cover multiple language domains in future work.

1.1.2 Second objective

The second objective is to develop four methods that, in conjunction with the language model, can be used to perform four different NLP tasks. These methods include: (a) a part-of-speech tagger method that takes an input sentence and outputs the correct part of speech of each word in the sentence, (b) a simplification method

that takes an input sentence and rephrases it to a lexically and syntactically simpler sentence whilst maintaining the semantic meaning of the original sentence, (c) a similarity method that takes two sentences as input and outputs whether the two sentences are semantically similar or not, and (d) a paraphrasing method that takes a sentence as input and outputs a paraphrased (but semantically similar) version of the sentence.

1.2 Scope

As the aim of the thesis is to demonstrate a proof of concept, the language model and the resulting NLP capabilities will not cover the full English language but will mostly be limited to cover a predefined set of questions pertaining to the Nobel prize. Due to the versatility of the English language, covering all possibilities of paraphrasing is a daunting task even for a limited dataset. Therefore, the resulting language model and NLP capabilities will not be exhaustive even for the predefined Nobel prize questions.

1.3 Outline

In chapter 2, some background and related work, such as previous knowledge graph-based language models and previous work on NLP subtasks is given. In chapter 3, the work and methods of this thesis is presented, including the definition and generation of the language model and the implementation of the NLP subtasks. Chapter 4 presents the results of applying the developed model and methods on a number of example sentences, and a subsequent discussion based on the results. The thesis is concluded in chapter 5 and pointers to areas of future research are given.

2
Background

This chapter presents some background and a selection of previous related work on language models (2.1.1), and on the four NLP tasks that are the subject of this thesis: POS tagging (2.2.1), text simplification (2.2.2), semantic similarity analysis (2.2.3), and text paraphrasing (2.2.4). The chapter gives a brief account on the history of NLP and the diaspora of both so-called rule-based and data-driven methods. The methods developed as part of this thesis are, however, excludingly of a rule-based nature which is why, occasionally, more light is shined on this family of approaches.

2.1 Natural language processing

Natural language is defined as any language that has developed naturally as a means of communication among humans, through repetition and without conscious planning [23]. In contrast, there are artificial and constructed languages such as computer programming languages and international auxiliary languages (e.g. esperanto) which are not considered as variations of natural language. Natural Language Processing (NLP) is an interdisciplinary field that concerns the engineering of computational methods used for programming a computer in such a way that it can process and understand human language [24]. The field is at the intersection between computational linguistics, cognitive science, computing science and artificial intelligence [13]. Some NLP tasks include speech recognition, dialogue systems, part-of-speech tagging, sentiment analysis and automatic language translation [13] [25]. Applications of NLP greatly effect the quality of human-computer interaction and areas of application range everywhere from business and education to healthcare and various sorts of services in peoples' everyday lives [19] [26] [4].

2.1.1 Language models

As opposed to programming languages which are constructed to follow a strict set of rules to be unambiguous and precise, natural language is full of ambiguities [7]. The variations of ambiguity are numerous and include part-of-speech ambiguities (such as whether a verb is simple past or past participle), semantic ambiguity (such as polysemic words; e.g. the word *run* has 29 distinct senses in Webster's Seventh Dictionary, further divided into around 125 sub-senses) [27], syntactic ambiguity (such as the one in *Sam went for a walk with her friend in the red shirt*), reference ambiguity (e.g. *Ethel told Lucy that her pie was wonderful*) and so on [28].

By using and simultaneously developing natural langauge, humans have, slowly and gradually over a long time, learnt to master the art of perceiving and communicating elaborate and refined meanings [7]. The intricate nuances of natural language, however, is the main problem when trying to make a computer process information in this form [28]. To be able to achieve this, the language needs to be modelled in some way, i.e. the regularities of the language need to be captured and characterized, or described, such that the description can be used to predict or disambiguate future use of the language [28] [29]. Designing an appropriate language model (LM) is at the core of all NLP tasks and the different approaches to solving NLP problems are closely connected with the approach chosen for the language modelling [28].

The approaches to language modelling that are seen in the literature can be broadly categorized into three families, that each has taken centre stage during different time periods [28] [30]. The first research within NLP dates back to the early 1950s. The earliest approaches to language modelling fall within the *linguistic* or *knowledge-based* family; these models were syntax-oriented and mainly based on hand-crafted rules and formal grammars [31] [30]. While having the advantage of being interpretable by humans and easy to debug, the rule-based systems were considered to have several limitations [13]. These limitations included the fact that NLP methods must extract not only syntactic information but also semantic information from text; i.e. the usage of a word is not only governed by what part of speech the word belongs to but also to a high degree on the (syntactic and semantic) context in which it is used [12]. Additionally, since the language processing using these methods was dependent on predefined rules, the models were poor at handling uncertainty, and building the hand-crafted models was considered a laborious and time-consuming project [13]. Thus, eventually a reorientation occured and, around the 1980s, statistical NLP was born [30]. For some time onward, data-driven machine learning approaches such as k-nearest neighbors, naïve Bayes, hidden Markov models, decision trees and random forests were widely used, until the next transformation happened following the first proposals of neural models for NLP [12]. Machine learning methods became popular largely because, with them, researchers do not need to hand-craft precise and exact rules for the language but can let statistical models, with automatically tuned parameters, *learn* the rules from large amounts of training data [13]. The statistical models have been considered very successful as they can learn to (more gracefully than the hand-crafted models) handle uncertainty, generalize across different domains and achieve almost human-like performance in natural language output [12]. From approximately 1990 to around 2010, NLP was dominated by the so-called ”shallow” machine learning algorithms, after which deep-structured machine learning algorithms entered the field. The shift toward deep learning is described as equally revolutionizing as the shift from rule-based methods to shallow machine learning as it had an equally transforming impact on model performance [13]. However, as is further discussed in section 2.3, deep learning methods do not come without significant drawbacks.

2.2 Subtasks of NLP

For this thesis, four methods for subtasks of NLP have been developed. This section presents some background and previous work on these subtasks.

2.2.1 Part-of-speech tagging

A part of speech (POS), also known as word class, is a category of words that share similar grammatical properties [32]. In the English language, the following 10 parts of speech are traditionally differentiated between: *verb*, *noun*, *adjective*, *adverb*, *preposition*, *numeral*, *determiner*, *pronoun*, *conjunction*, and *interjection* [33]. POS tagging is the process of automatically labelling each word in a sentence with the correct word class (and sometimes even more explicit grammatic information) [34]. Labelling the words in a text with the correct POS is an important first measure in discerning the linguistic structure of a text [35]. POS tagging is, moreover, often referred to as the backbone of NLP, as it lays such an important foundation in the preprocessing stage of many other NLP tasks [34] [36] [4]. Useful applications include the tagging of large text corpora that can be used for linguistic studies, tagging words for text indexing and retrieval, identifying word classes for machine translation, word sense disambiguation, question answering parsing, as well as encoding and decoding pronunciation in speech processing [36] [4].

Tremendous effort has been made by researchers to improve the accuracy of current POS tagging methods; nevertheless, the field still faces challenges when it comes to reducing error rates [4]. The reason that POS tagging is nontrivial is the fact that many words are ambiguous and belong to different parts of speech depending on the context [4] [35]. The word *tag*, for example, can be either a verb or a noun and without context in the form of surrounding words it is impossible to deduce which POS is appropriate for a specific instance of the word in a text. Therefore, a POS tagger needs to be a system that uses context to assign the correct POS to a given word [35].

There are numerous variations of POS tagging systems but the overall architecture is often similar [37]. One of the first steps is typically *tokenization*, which means that the input text is split into tokens that are feasible for further analysis. Usually, each string of non-blank characters in the text constitutes a token but this varies depending on the structure of the natural language that is processed [37]. In the next step, the tokenized input is passed through a program, called a POS tagger, that assigns the appropriate POS tag to each token. Considering the ambiguous sentence *"We can can a can."*, a successful POS tagger may produce the POS tags shown in Table 2.1 [37]. The output from the tagger may be either explicit or expressed in a compact tag as seen in the table.

There is a wide spectrum of methods that have been used to encode the decision-making processes of POS taggers. The majority of the current taggers are encoded with statistical a priori knowledge of all the possible POS tags of a given word as well as contextual interdependencies between tokens and their respective tags [37]. The

Token	Explicit POS specification	Compact POS tag
We	personal pronoun, first person, unspecified gender, plural and nominative case	Pp1-pn
can	modal verb, indicative present	Voip
can	main verb, infinitive	Vmn
a	indefinite article, unspecified gender, singular	Ti-s
can	common noun, neuter gender, singular	Ncns

Table 2.1: Examples of two variants of POS tagging of an ambiguous sentence [37].

collection of a priori knowledge that is required for a tagger to perform its job is what can be referred to as the language model. An overwhelming majority of the LMs that are available today are encoded in a data-driven statistical manner such that the POS tagger learns the LM from training data [37]. The data-driven methods are attractive since, given the availability of adequate training data, reasonable performance can be achieved with minimal human effort [37]. One example of a recent DNN-based state-of-the-art POS tagger is the Bidirectional Long Short-Term Memory Deep Neural Network with a CRF layer (BI-LSTM-CRF) which, when used on the Penn TreeBank (PTB) POS-tagging data set produced a POS-tagging accuracy of 97.55% [38]. Using a similar model, others have reported a POS-tagging accuracy of 97.85% [39]. Despite impressive results, a significant drawback of the data-driven approaches involving statistical LMs is that the error rate seems to remain at a few percentage points, despite the numerous efforts that have been made to improve the accuracy of the systems [37]. Furthermore, the statistical data-driven methods suffer from the usual problems of lack of interpretability (which are discussed in section 2.3).

As opposed to the data-driven models, LMs for POS tagging can also be non-statistical and deterministic, and based on hand-coded grammar rules [37]. An example of a system that is purely rule-based and grammar-based is the EngCG-2 tagger which is capable of performing POS disambiguation of arbitrary English texts with an accuracy of 99.7% [40]. This accuracy is higher than any reported accuracy of a statistical tagger. However, the development of the model took several years [37].

There are also hybrid methods that combine statistical and hand-crafted rule-based approaches [37] [28]. One example of such a method is a transformation-based error-driven learning method in which the POS tagging process is rule-based but the rules are automatically learned from an annotated training corpus [41]. Testing of this method has resulted in a reported accuracy of 97.2% when all the words in the test set were known to the tagger, and an overall accuracy of 96.6% when the tagger was used on a data set that included unknown words [41]. A number of rules that the system learned are seen below[1]:

[1] AT = article, HVD = had, IN = preposition, MD = modal, NN = sing. noun, NP = proper noun, PPS = 3rd sing. nom. pronoun, PPO = obj. personal pronoun, TO = infinitive to, VB=

1. TO IN NEXT-TAG AT (Change from TO to IN if next tag is AT)
2. VBN VBD PREV-WORD-IS-CAP YES (Change from VBN to VBD if previous word is capitalized)
3. VBD VBN PREV-1-OR-2-OR-3-TAG HVD (Change from CBD to VBN if any of the previous 3 words are HVD)
4. VB NN PREV-1-OR-2-TAG AT (Change from VB to NN if any of the previous 2 words are AT)
5. NN VB PREV-TAG TO (Change from NN to VB if previous tag is TO)

To summarize, the current average performance of state-of-the-art POS taggers lies at an accuracy of around 97-98% [37], which sounds like an impressive achievement. It should, however, be noted that considering an average sentence length of 30 words this level of accuracy would mean that on average every third sentence has 2-3 tagging errors [37]. This error rate could be detrimental to higher-level processing of the text [37].

2.2.2 Text simplification

Automatic text simplification (TS) is the process of reducing the linguistic complexity of text whilst retaining its original semantic meaning. The purpose is to make the text more easily comprehensible and improve its readability [42]. Text simplification is an important task for many reasons. For instance, it can be used to make text accessible to readers with low reading abilities, such as second language readers [43], people with diagnoses such as autism [44] and dyslexia [45], children, or people with low literacy levels [42]. Automatic TS is also, similarly to POS tagging, used as a preprocessing technique in various other NLP tasks to facilitate higher-level processing [42].

Automatic TS is an actively studied research field and has, like most NLP tasks, seen an upsurge in conjunction with the growth in statistical data-driven ML techniques for NLP [42]. However, despite the active interest of the research community in the field, the situation is described as far from satisfactory and far from reaching a saturation stage [42].

TS can be divided into several different approaches, two of which are lexical and syntactic simplification [42]. Lexical simplification means simplifying the individual words in the text by identifying complex words and replacing them with less complex synonyms. Syntactic simplification refers to reducing the grammatical complexity of complicated syntactic structures (such as passive relative clauses) [42]. Research on either lexical or syntactic simplification can be roughly classified into two main approaches, namely rule-based and data-driven [42]. The first rule-based lexical simplification system was proposed in 1998 [46] to simplify English newspaper texts to make them comprehensible to aphasic readers. This system consists of two main

verb, VBN = past part. verb, VBD = past verb.

components: (i) an analyser component that performs a syntactic analysis and POS tag disambiguation of the input text and (ii) a simplifier component that simplifies the analysed text [46]. The simplifier component includes a lexical simplifier as well as a syntactic simplifier. The lexical simplifier works by feeding the words in the input text, one by one, into the WordNet lexical database [47] (see a description of WordNet in 2.4) and thus retrieving a list of synonyms for each word [46]. Then, the word frequency of each synonym is extracted from the Oxford Psycholinguistic Database [48] and the synonym with the highest frequency is selected and substitutes the original complex word in the text [46]. The project that this system was a part of has widely influenced subsequent research on lexical simplification systems [49]. The typical pipeline of lexical simplification systems is seen in Figure 2.1, where the example sentence *"The cat perched on the mat."* is simplified. The main difference between this pipeline and the system just described is the addition of the word sense disambiguation step where, in this example, the synonym *"roosted"* was ruled out, as it does not apply properly to the context of cats [49].

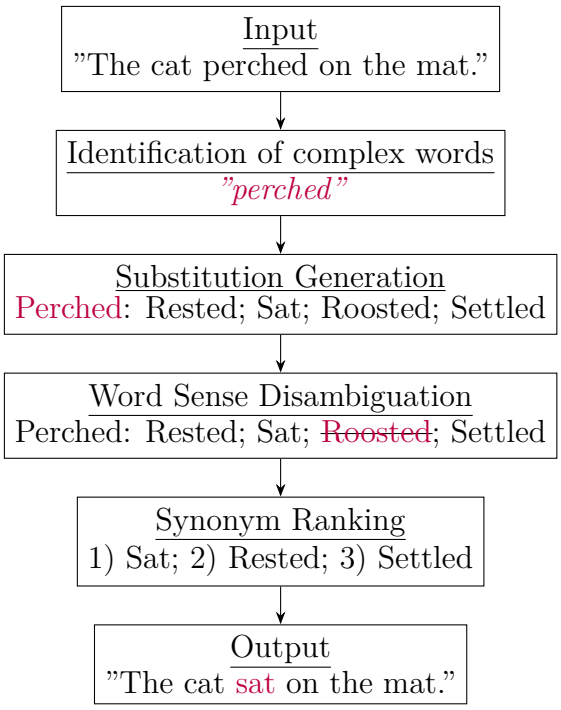


Figure 2.1: The typical lexical simplification pipeline. In the example, the sentence "The cat perched on the mat." is simplified to "The cat sat on the mat.".

The complex word identification and synonym ranking steps (seen in Figure 2.1) share common characteristics as both steps require a way to measure lexical complexity (and "ranking" of words) [46]. Choosing the method for measuring or evaluating text complexity is in general a central point in TS [42]. Traditionally, it is a measure that is hard to define and there is a wide spectrum of methods and text complexity scores that have been used by researchers within the field [42] [50]. For evaluating lexical complexity, i.e. complexity of individual words, word frequency is often used, which can be obtained from a number of sources [51] [52] [42]. For assessing complexity of a block of text, containing more than one word, a combination of automatic and manual methods is frequently used, where the automatic method commonly includes computing various readability indices (such as the Flesch Reading Ease score [53], the Fog Index, and the SMOG Grading score [54]) [55] [56] [57]. The manual approach typically entails letting human experts evaluate the system on a sentence level. These experts usually assess the performance of the system by considering three aspects, namely degree of simplification, preservation of grammatical correctness, and preservation of meaning

[51] [58] [59] [54] [60] [55].

Syntactic simplification is usually carried out in three stages: (i) analysing the structure of the input text and identifying its parse tree (a tree representation of the relations between the grammatical components of the sentence [61]), (ii) modifying the parse tree, according to a set of rules, to simplify the sentence structure (such as rearranging or dropping clauses), (iii) regenerating a readable sentence from the modified parse tree [42]. Similarly to lexical simplification, syntactic simplification can also be categorized roughly into rule-based or data-driven solution approaches [42]. Most syntactic simplification systems are, however, rule-based as the data-driven ones have been shown to produce less syntactic simplification [42]. Meanwhile, the lexical rule-based approaches have been considered limited in terms of performance. Therefore, hybrid approaches to TS have been proposed and shown to produce superior results compared to either excludingly rule-based or excludingly data-driven approaches [42]. In this thesis, however, data-driven methods are not considered due to the drawbacks mentioned earlier in this thesis and in 2.3.

2.2.3 Semantic textual similarity

Semantic similarity is the notion of conceptual distance between two objects [62]. Semantic textual similarity (STS) is, more specifically, the measure of semantic equivalence between two blocks of text, such as two sentences [63]. Measuring STS is a complex and challenging problem as the semantic meaning of words is highly dependent on the context [62]. Applications of semantic similarity analysis in NLP include estimating relatedness between search engine queries [64], generating keywords for online advertising [65], analyzing results in biomedical applications [66] [67] [68], online information retrieval [69], text summarization [70], and text categorization [71]. Given the wide range of applications, the approaches used to measure semantic similarity are highly varied [62], and only a brief account of some of the methods will be given here. Rather than resulting in a binary decision, i.e. either similar or *not* similar, STS methods often output a ranking or percentage of similarity between texts [63]. STS can be said to measure the degree of synonymy, i.e. the amount of shared characteristics between two objects or concepts [72]. For instance, *car* and *plane* are considered semantically similar *to some degree*, as they are both means of transport, they are both fuel-driven machines with wheels and engines etc.. A similar reasoning holds for *scientist* and *actor*. The words *actor* and *movie* are, however, not considered semantically similar according to the definition, despite the fact that they are clearly related, because they belong to different branches of taxonomy [73]. I.e. *actor* and *scientist* <*is-a*> *person*, whereas *movie* <*is-a*> *product* [73]. The relationship between *movie* and *actor* falls within the wider concept of measuring *semantic relatedness* [63].

The methods used to compute STS can be roughly classified into the following families: (i) knowledge-based methods, (ii) corpus-based methods, (iii) deep neural network-based methods, and (iv) hybrid methods [63]. In knowledge-based methods an underlying knowledge base or, equivalently, knowledge source is used to derive

information about similarity between terms [63]. The knowledge base may consist of e.g. an ontology, a lexical database, a thesaurus, a dictionary etc [63]. From the knowledge base one can thus derive a structured, unambiguous representation of concepts, and the semantic relations between them [63]. Two lexical databases that are widely used in knowledge-based semantic similarity methods are WordNet and Wiktionary [63]. WordNet is (as further described in 2.4) a lexical database over the English language that can be visualized as a graph, where the nodes represent the meaning of the words and the edges represent the relationships between words [73]. Wiktionary[2] is an open-source lexical database that covers more than 6 million words from 4000 languages [74]. Wiktionary is organized such that each sense of a given word has its own entry. Compared to WordNet, however, Wiktionary does not contain well-established taxonomic lexical relationships and is therefore less well suited for semantic similarity analysis [74].

Knowledge-based methods can be further classified into (i) edge-counting methods, (ii) feature-based methods, and (iv) information content-based methods [63]. In edge-counting methods, the knowledge-base is considered a graph, in which the nodes represent words that are taxonomically connected by edges [75]. The similarity between two terms is then inversely proportional to the number of edges between the terms [75]. In feature-based methods, similarity between two words is computed in terms of the overlap between the *gloss* of the given words [63]. A gloss represents the meaning of a word in a dictionary, and a *glossary* is a collection of glosses [63]. For example, using the Lesk measure, similarity of two words can be estimated by studying the overlap of words present in their respective glosses as well as the glosses of other words that they are connected to in e.g. WordNet [76]. Lastly, information content-based methods measure similarity in terms of the information content (IC) value of different terms [63]. The IC value of a concept is a measure of the amount of information that is obtained from the concept when it occurs in a text [77]. A high IC value indicates more specificity and less ambiguity whereas a lower IC value indicates that the word has a more abstract meaning [73].

Corpus-based semantic similarity methods is a family of methods that measure semantic similarity between concepts by extracting information from large text corpora, based on the assumption that ”similar words occur together, frequently” [78]. In corpus-based methods, two words are considered similar if their surrounding contexts tend to be similar [73]. Drawbacks of these methods include the fact that they do not take the actual meaning of words into account and thus leave room for word sense ambiguity [63]. In general, this makes them more suited for measuring semantic relatedness rather than semantic similarity [73]. Additionally, corpus-based methods depend on the access to huge high-quality corpora, the building and curating of which is very resource-demanding. Furthermore, despite the growing access to large amounts of text on the internet, the ”ideal corpus” is yet to be defined [63].

The third family of methods that was mentioned is the family of DNN-based methods. Similarly to many other NLP tasks, DNN-based methods have outperformed

[2]https://en.wiktionary.org/

most traditional methods for STS methods [63]. However, they share the dependency on large high-quality corpora. Additionally, their implementation requires huge computational resources and their black-box nature makes them less well suited for applications that require a high degree of interpretability (see further in 2.3).

2.2.4 Paraphrasing

Paraphrasing is the task of expressing the same, or almost the same semantic meaning of an input sentence, in a different wording, whilst maintaining grammatical and syntactical correctness [79]. Paraphrasing methods also include recognizing and extracting paraphrases from corpora, and is thus closely related to methods for semantic similarity analysis [80]. NLP related applications of automatic paraphrasing include text summarization [81], question answering (QA) [82], information retrieval [83], machine translation [84], and conversational agents [85]. Automatic paraphrasing is very useful in knowledge-based QA systems for bridging the gap between the user's questions and the knowledge-based assertions in the system [84] [82] [86] [87]. Additionally, paraphrase generation is an important tool for generating textual training data for other NLP tasks [84].

There are three main challenges of sentence paraphrasing, the first two being: (i) words can have multiple meanings, and (ii) two words that are considered synonyms are generally not interchangeable in all contexts [88]. The word *great* can e.g. mean either *of a substantial amount* or *very good*, but only one of these is appropriate in the sentence *"Walking home, he was stricken by a great pain in the knee."*. Therefore, it is not sufficient to rely on generic domain-independent lexical resources [88]. The third challenge is (iii) the fact that two sentences may be paraphrases of each other without there being any one-to-one correspondences between single words or phrases within the sentences [88]. The latter makes sentence paraphrasing a problem of a different dimension compared to paraphrasing of smaller lexical units [88].

Furthermore, the notion of semantic similarity is not trivial in the first place. Considering sentences (1)-(3) below, (1) and (2) are paraphrases of each other. When it comes to sentence (3), most people would accept this variant as a paraphrase of (1) and (2) as well; however, it may be argued that in sentence (3) there is some ambiguity to whether the construction of the bridge has been completed or not [80]. To account for such fine distinctions, and the fact that changing the wording of a sentence or phrase may always tweak the conveyed meaning somewhat, even if to an infinitesimal degree, the clause *almost the same* is included in the definition of paraphrasing (see the beginning of this section) [80].

(1) Company X constructed the new bridge.
(2) The new bridge was constructed by Company X.
(3) Company X is the constructor of the new bridge.

As a way to generalize the paraphrasing capabilities of a system, the paraphrasing methods may be based on *templates* such as sentences (4)-(6), where X and Y

may be replaced with arbitrary noun phrases. Additional restrictions may also be added to the templates. These may be syntactic, such as requiring either uncountable or countable nouns, or semantic, such as requiring Y to be an object containing letters [80].

(4) X wrote Y.
(5) Y was written by X.
(6) X is the writer of Y.

Additional to paraphrasing, there is a neighbouring concept called *textual entailment*. Textual entailment methods generate, recognize, or extract pairs $<T, H>$ of expressions (T = *True*, H = *Hypothesis*) such that a human that reads and trusts the truth of T would deduce that H is most probably also true [80]. An example of entailment is seen in sentences (7)-(8) below, where (7) textually entails (8) but the reverse is not true (if e.g. Y is a symphony composed by X) [80].

(7) X painted Y.
(8) Y is the work of X.

In addition to the definition given at the beginning of this section, paraphrasing can also be defined in terms of symmetrical and asymmetrical textual entailment [89]:

Definition 2.2.1 *A symmetrical paraphrase is a pair of natural language expressions <E_a, E_b>, such that each expression entails the other one, i.e. $E_a \models E_b$ and $E_b \models E_a$.*

Definition 2.2.2 *An asymmetrical paraphrase is a pair of natural language expressions <E_a, E_b>, such that one expression entails the other one, but at least one expression is more general or contains more information than the other one, i.e. either $E_a \models E_b$ and $E_b \not\models E_a$ or $E_b \models E_a$ and $E_a \not\models E_b$.*

In the literature, a distinction is made between paraphrasing/textual entailment methods for *recognition* and methods for *generation* [80]. A recognizer system takes as input a pair of language expressions (specific or templates) and outputs a judgment (that may be either probabilistic or binary) that indicates whether or not there is any entailment relation between them [80]. Thus, a paraphrase recognizer is practically a semantic similarity analyser. To a paraphrasing/textual entailment generator, the input is a single language expression and the output is normally a set of language expressions that entail or are entailed by the input [80]. The output set normally has to be as large as possible, whilst containing as few errors as possible, to be considered a high-performing system [80].

Similarly to other NLP tasks, there is a large variety of paraphrasing methods [90]. In the last few decades, data-driven corpus-based approaches have become extremely popular [90]. Many of these methods rely on the notion that a language possesses distributional structure, which refers to the idea that sentences and phrases are not

formed by arbitrarily combining components of the language [90]. Rather, elements of the language tend to occur only in certain positions relative to other elements of the language. Furthermore, it is assumed that the positions of members of a certain class relative to positions of members of some other class can be described in terms of probabilistic measures, and that it thus is possible to compute the *distribution* of every member in the corpus [90]. Then, based on these ideas, words or phrases that share the same distribution are assumed to have similar meanings [90]. A weakness of this assumption is the fact that many terms that are distributionally similar cannot be used interchangeably in paraphrasing [90]. Consider e.g. the elements of the pairs <*boys*, *girls*>, <*cats*, *dogs*>, <*high*, *low*> which often occur in similar contexts, and thus have similar distributions, but are not semantically interchangeable [90].

2.3 Drawbacks of deep neural networks

The deep neural networks that are currently extensively used in NLP are composed of layers of artificial neural networks and often have billions of trainable parameters [12]. One of the most significant drawbacks of the DNN methods is that they are so complex that they are practically black boxes and are thus too complicated for a human to comprehend [14] [91] [17]. This stands in contrast with the naturally interpretable models of the rule-based paradigm in the early days of NLP [13]. As opposed to a black-box system, an interpretable system is a system that is composed of interpretable primitives such that the decision-making steps can be comprehended by a human [21].

The increasing use of algorithmic decision-making in society combined with the upsurge of neural models in applications has led an increasing number of people to caution against the rise of a ”black-box” society and stress the importance of algorithmic transparency [92] [17] [91] [13] [14]. In both America and the European Union the increasing awareness has led to new legislation that restricts the use of black-box algorithms in public decision making [93] [94] [17]. Interpretability of models is important for several reasons. For instance, the question of accountability is important both for model builders and users. System builders need to understand whether the system is working as intended, whether the predictions are sensible, and whether relevant legislation and regulations are conformed to [95]. For users, it is crucial to understand how to use the model in the correct way and that the output is unbiased and fair [95] [17]. There are unfortunately several examples of cases where incorrect use has led to severe negative consequences. These examples include polluted air that was incorrectly classified as safe to breath and dangerous prisoners that were released from prison prematurely due to poor automatic bail decisions [17]. Moreover, interpretability is crucial in situations where a model needs to be combined with external data, which is likely to occur if the model is used in any social context, as many such contexts tend to have a high variability depending on e.g. geographical parameters and societal development [17].

Another disadvantage of deep neural models is that they require massive amounts

of data to be trained [96]. For many applications, the data needs to be carefully preprocessed and curated to prevent the model from learning unwanted biases such as e.g. language that is violent, racist or sexist [19]. This work takes an enormous amount of time and resources. Furthermore, high-quality data is especially significant in high-stake situations, such as in the case of task-oriented agents that need to provide users with precise and consistent information [97].

2.4 Previous work on interpretable knowledge-based language models

In contrast to supervised, corpus-based machine learning methods which require large amounts of manually annotated training data, so called knowledge-based systems have proven to be a useful alternative for NLP tasks such as word sense disambiguation [98]. A knowledge base is a system that stores information about data structures and relationships between them [99]. Knowledge-based approaches to NLP typically use knowledge graphs (KGs) to represent the knowledge base [73]. A formal definition of a KG is as follows:

Definition 2.4.1 *A KG is defined as a directed labeled graph, $G = (V, E, \tau)$, where V is a set of nodes, E is a set of edges connecting those nodes; and τ is a function $V \times V \rightarrow E$ that defines all triples in G.*

Nodes of KGs are typically composed of a set of concepts $C_1, C_2, ..., C_n$ that represent conceptual abstractions, and a set of instances $I_1, I_2, ..., I_m$ that represent entities in the real world [73]. The knowledge graph that is most frequently used is based on WordNet [98], which is an online lexical database that contains information about words in the form of more than 166 000 pairs (f, s), where f is a word form and s is a word sense [47]. WordNet also includes information about semantic relations between words, more specifically the relations synonymy, antonymy, hyponynmy, meronymy, troponymy, and entailment [47]. The words are arranged according to the semantic relations between them into so-called synsets, such that words that are connected via a given semantic relation are in the same synset [62]. When WordNet is visualized as a knowledge graph, synsets constitute the nodes in the graph and the edges represent the relations between synsets [73].

Despite including useful information about word relations, lexical databases such as WordNet tend to suffer from sparseness in the availability and density of relations [100]. Many synsets lack sufficient information for an algorithm to be able to choose the appropriate word sense for ambiguous words, and contextual knowledge from actual text is not included [100] [98]. To remedy this problem, various attempts have been made to incorporate semantic relations from other sources into the lexical databases, such as from sense annotated corpora [100] [101]. Despite the advantages of such hybrid approaches, that type of solution does not circumvent the corpus-dependency problem. To this end, attempts have been made to incorporate external semantic knowledge into the actual knowledge graph, which has been proven to increase the accuracy of the system compared to only basing the graph

on WordNet [100]. The resulting capacity of the system to differentiate between appropriate and inappropriate senses of ambiguous words has, however, still not been satisfactory. This makes researchers suggest that WordNet might not be an ideal knowledge base to use in these applications [100].

2.5 N-gram models

A common approach in statistical language modelling is to make the Markov assumption that the probability of a token t_k in a text only depends on the j preceding tokens [61]. This can be expressed in terms of the following equation:

$$P(t_k|t_1, ..., t_{k-1}) \approx P(t_k|t_{k-j}, ..., t_{k-1}) \tag{2.1}$$

This approach to language modelling is also referred to as *n-gram* language modelling [61]. An *n-gram* is a sequence of n consecutive tokens in a text. What constitutes a token depends on the context; it could be either a word, a letter, or something else. For example, we may consider the sentence *"Which scientist received the Nobel prize in Physics in 1903?"*, and define a token as a word in the sentence. Not considering the question mark, this sentence then has 9 1-grams, namely the set {[*Which*], [*scientist*], [*received*], [*the*], [*Nobel*], [*prize*], [*in*], [*Physics*], [*in*], [*1903*]}. The 2-grams of the sentence are {[*Which scientist*], [*scientist received*], [*received the*], [*the Nobel*], [*Nobel prize*], [*prize in*], [*in Physics*], [*Physics in*], [*in 1903*]}. For any given sentence, n-grams can be generated in a similar way for all n such that n is larger than 0 and less than the number of tokens in the sentence.

In this thesis, the technique of splitting a sentence into n-grams of all possible n-gram lengths is used, but in a non-statistical context. This is described further in Chapter 3.

3
Implementation and methods

This chapter presents the methods that have been developed and implemented in this thesis. These include the development of the interpretable knowledge-based language model and the implementation of the four interpretable NLP methods for (i) POS tagging, (ii) semantic similarity analysis, (iii) paraphrasing, and (iv) text simplification.

The structure of the chapter is as follows: (i) a description of the language model architecture is presented, (ii) the generation of the language model is explained, (iii) brief accounts of various implementation details such as auxiliary methods and important classes are given, (iv) the developed algorithms of the four NLP tasks are presented.

3.1 Components of the language model

As described in 1.1.1, the first objective of this thesis is to expand and improve the language model that has been proposed as part of recent work by Wahde et al. [22]. The LM that Wahde et al. propose is a special case of a generalized knowledge graph. In line with the description of KGs in 2.4, the KG proposed by Wahde consists of a set of nodes that are interconnected by a set of edges. The KG defines three types of edges, namely **instance edges** (isInstanceOf), **attribute edges** (hasAttribute) and **subcategory edges** (isSubcategoryOf). For the special case where the KG is a language model, additional **custom edges** can be defined that are relevant to describe relations in the context of language.

For this thesis, the following custom edges have been used in the LM: isInflectionOf (for verbs), isPluralOf (for nouns), isOrdinalOf (for integers), isComparativeOf (for adjectives and adverbs), isSuperlativeOf (for adjectives and adverbs), isSimilarTo (asymmetrical similarity relation applicable to phrases), isBiSimilarTo (symmetrical similarity relation applicable to phrases), isVersionOf (applicable to words). An example of a small and incomplete LM that is built according to the described structure is shown in Figure 3.1.

As can be seen in the figure, a node does not represent a synset (as in WordNet) but can be either a single word, or a phrase that in turn is composed of a *wordBase* that points to a list in which each element is either a *word* or a *wordOption*. As seen in Figure 3.1, both *word* and *wordOption* are subcategories of *wordBase*. The text

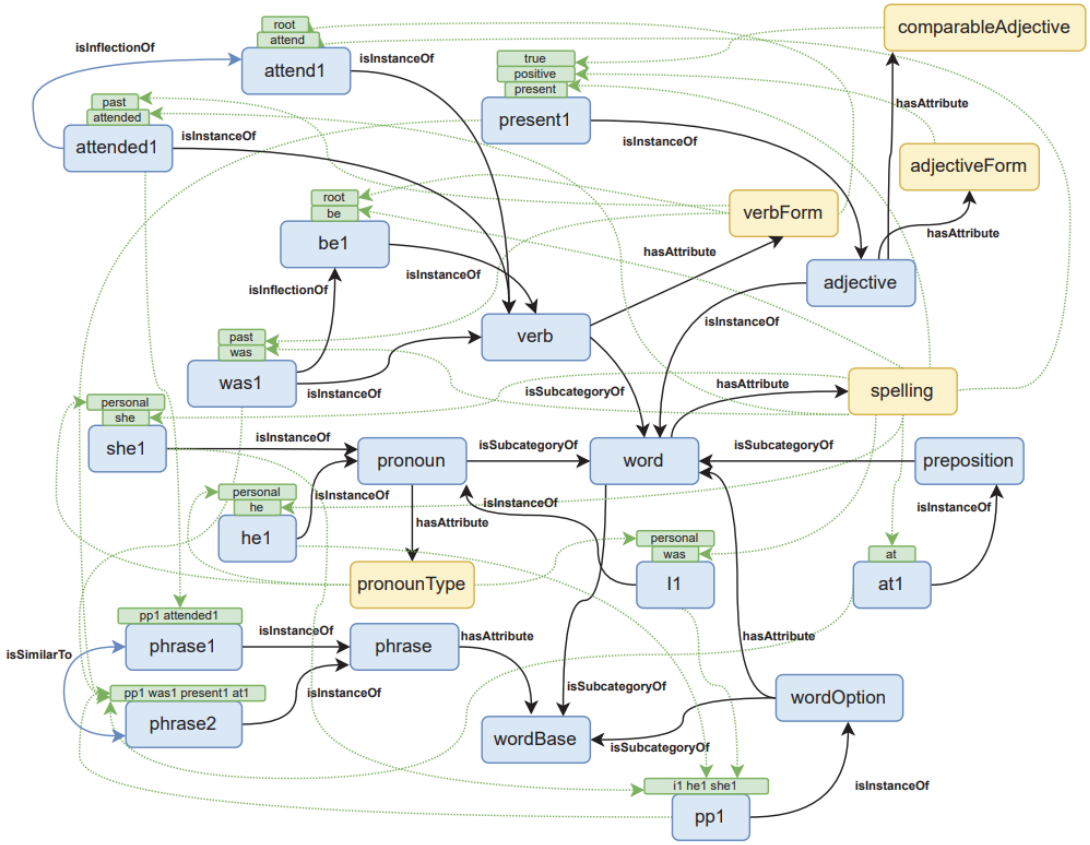


Figure 3.1: An example of a small and incomplete LM, visualized as a graph network (M. Wahde, personal communication, 12 Dec, 2022).

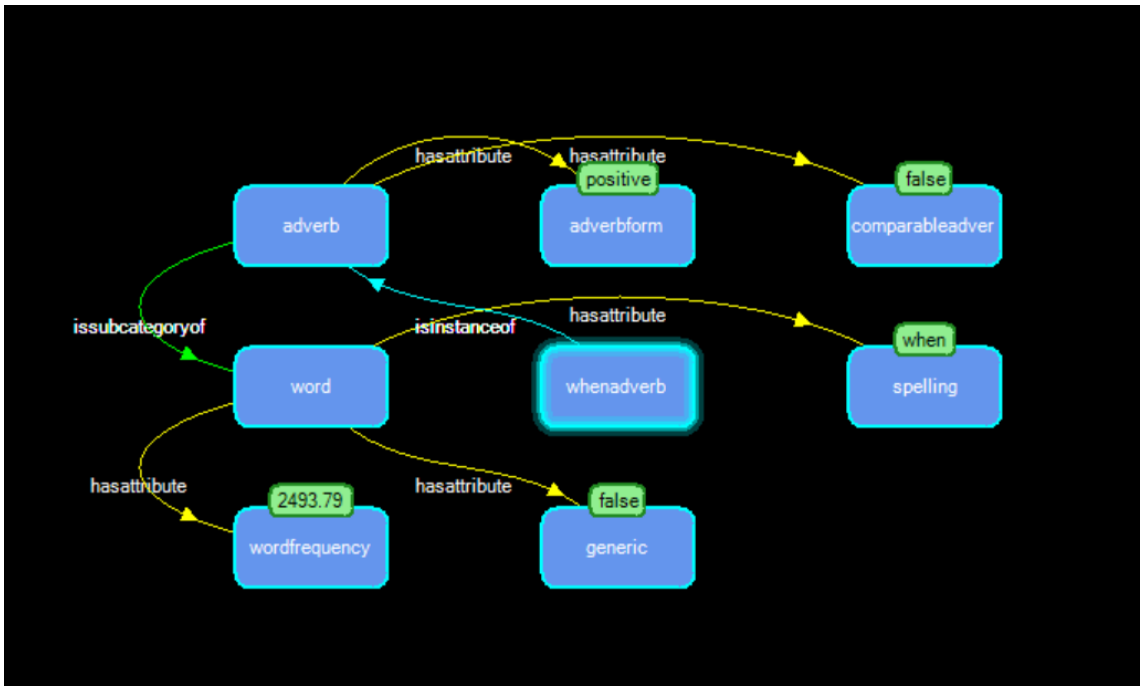


Figure 3.2: A small part of the LM developed in this thesis, where the node for the adverb *when* is visualized along with the closest neighbouring nodes.

that is shown inside any node is the identifier of the node (e.g. *she1* is the identifier of a node that is an instance of pronoun, has the spelling *she* and the pronoun type *personal* etc). Identifiers are used since every node has to be well-defined. Since most words have several meanings, using only spelling to identify them would be ambiguous. Moreover, as seen in Figure 3.1, a node can have a number of different abstraction levels, i.e. a node can be a specific word instance (such as *she*) or be a generic node such as a POS (e.g. *pronoun*) or the even more generic nodes *word* or *phrase*. In this way, the syntactical relationships that are present in the English language are represented in the model and can be used to define rules for language processing. Considering a specific example, the node *verb* is a subcategory of *word* which is a subcategory of *wordBase*. Moreover, *verb* has the direct attribute *verbForm* (the allowed values of a given attribute are specified when defining the LM, which will be described later), and also the indirect attribute *spelling* which is inherited from *word*. Thus, a fully defined verb in the LM needs to have a specified verb form and spelling, such as the node *attend1* which has the specified spelling *attend* and verb form *root*.

An example from the LM that was developed for the thesis is shown in Figure 3.2, where the node for the adverb *when* is visualized (in the centre of the image) as well as the most closely connected neighbouring nodes. The graph shown in Figure 3.2 is only a subgraph of the full LM; e.g. all adverb instances in the LM are connected to the node *adverb* but the full graph would be too detailed to visualize in a small figure. As seen in Figure 3.2, the node *word* has an attribute called *wordfrequency*, which has not yet been mentioned but will be described more in detail further on.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Language model
%%%%%%%%%%%%%%%%%%%%%%%%%%
[LMNode: identifier=wordbase]
[LMNode: identifier=word,isSubcategoryOf=wordbase,hasAttribute=spelling,hasAttribute=wordFrequency,hasAttribute=generic]
[LMNode: identifier=generic,isType=string{false*;true}]
[LMNode: identifier=wordFrequency,isType=double]
[LMNode: identifier=spelling,isType=string]
[LMNode: Identifier=wordOption,isSubcategoryOf=wordbase,hasAttribute=word]
[LMNode: Identifier=wordOptionSpecific,isSubcategoryOf=wordbase,hasAttribute=word]
[LMNode: Identifier=wordPowerOption,isSubcategoryOf=wordbase,hasAttribute=word]

%%%%%%%%%%%%%%%%%%
% Parts of speech:
%%%%%%%%%%%%%%%%%%
%
% Noun:
%
[LMNode: identifier=noun,isSubcategoryOf=word,hasAttribute=nounForm,hasAttribute=nounType,hasAttribute=possessiveNoun]
[LMNode: identifier=nounForm,isType=string{singular*;plural}]
[LMNode: identifier=nounType,isType=string{countable*;uncountable}]
[LMNode: identifier=possessiveNoun,isType=string{false*;true}]
%
% Verb:
%
[LMNode: identifier=verb,isSubcategoryOf=word,hasAttribute=verbForm]
[LMNode: identifier=verbForm,isType=string{root;singular3;presentParticiple;pastSimple;pastParticiple;presentSimple}]
%
% Adjective:
%
[LMNode: identifier=adjective,isSubcategoryOf=word,hasAttribute=adjectiveForm,hasAttribute=comparableAdjective]
[LMNode: identifier=adjectiveForm,isType=string{positive*;comparative;superlative}]
[LMNode: identifier=comparableAdjective,isType=string{true*;false}]
%
% Adverb:
%
[LMNode: identifier=adverb,isSubcategoryOf=word,hasAttribute=adverbForm,hasAttribute=comparableAdverb]
[LMNode: identifier=adverbForm,isType=string{positive*;comparative;superlative}]
[LMNode: identifier=comparableAdverb,isType=string{true*;false}]
%
```

Figure 3.3: A small part of the textual representation of the LM developed in this thesis.

3.1.1 Textual representation of the language model

Now that the conceptual structure of the LM has been presented, this section will describe the textual representation of the LM. A small part of the textual representation of the LM used in this thesis is seen in Figure 3.3. This figure shows the first lines of the text document that constitutes the textual definition of the LM. The textual representation is parsed in a computer program which results in the visual representation, part of which was seen in Figure 3.2. As seen in Figure 3.3, each node is defined on a separate line, within square brackets, starting with the node prefix *LMNode* and the identifier. Subsequently, various edge relations and attributes are defined. E.g. the node *word* has the edge relation *isSubcategoryOf=wordbase* (which connects it to the node *wordbase*), and the attributes *spelling*, *wordFrequency*, and *generic* (indicating that it is not the specific word instance *word* but the generic node *word* which has specific instances as subcategories). Some attributes are nodes and some attributes are *value nodes*. If an attribute is a *value node*, it contains the *isType* declaration, as e.g. is seen in Figure 3.3 for the *generic* attribute which can have either of the values *true* and *false*. The value nodes are the bright orange nodes in Figure 3.1, and the regular nodes are shown as blue. The values of the value nodes are shown in the green boxes directly above the node to which the value attribute applies (see e.g. the word frequency value and the adverb form value in Figure 3.2).

3.1.2 Phrases

As mentioned in 2.4, the lexical database WordNet (which most current KG-based LMs are based on) includes relations between words and word forms but lacks the aspect of textual context. WordNet does provide example sentences for the words but not in a format that can be easily used for computational purposes, such as automatic paraphrasing tasks. The LM used in this thesis, however, incorporates the concept *phrase* which enables computational processing of contextual knowledge. By defining instances of *phrase*, word instances are put into context and thus word sense disambiguation can be carried out. To understand the concept of phrases in the LM, consider again the LM example in Figure 3.1, where the nodes with identifiers *phrase1* and *phrase2* are visualized in the bottom left corner of the graph. As seen in the figure, both *phrase1* and *phrase2* are instances of *phrase* and a similarity relation is declared between them, represented by an *isSimilarTo* edge. Every phrase has a *wordBase* attribute which defines the words that constitute the phrase. The *wordBase* attribute points to a list which may be made up of either specific word instances or instances of *wordOption* (or a mix of the two). The respective word bases of *phrase1* and *phrase2* are shown in the green boxes directly above the phrase nodes. The word base of *phrase1* is {*pp1, attended1*} and the word base of *phrase2* is {*pp1, was1, present1, at1*}. As seen in Figure 3.2, most of the elements in these two word bases are instances of some POS which in turn is a subcategory of *word*, whereas *pp1* is an instance of *wordOption*. The *wordOption* points to a list of the possible words that *pp1* could be substituted with, and this list is shown in the green box directly above the *pp1* node, namely: {*I1, he1, she1*}. This means that *phrase1* has the three possible spellings {*I attended, he attended, she attended*} and *phrase2* has the possible spellings {*I was present at, he was present at, she was present at*}. These phrases can be used to disambiguate the meaning of the word *present* and clarify that the word *present*, when used in this context, is the specific word instance *present1* which is an adjective and not e.g. the noun (as in *I gave him a present*). The similarity relation between the two phrases can be used to paraphrase either *phrase1* or *phrase2*.

To exemplify the difference between a LM based on WordNet and the LM used in this thesis, we may consider the phrase *I was present at the meeting*. To paraphrase this phrase using WordNet, the synset of the adjective *present* could be retrieved (given that it is known by the program that *present* is an adjective), which contains the words *attendant, ever-present, existing, here, naturally occurring, omnipresent*, and *ubiquitous*. This information does not help much in forming a paraphrased version of the sentence that most people would accept as natural (e.g. *I was existing at the meeting* is not a natural way of saying *I was present at the meeting*). With the LM that is presented here, however, the phrase can be directly paraphrased to *I attended the meeting*, despite *present* and *attended* belonging to different word classes.

```
% nobel prize in chemistry/physics/biology
[LMNode: identifier=thePrizeDiscipline, isInstanceOf=phrase, wordbase = {nobelAdjective,prizeNoun,in11,genericDiscipline11},
isBiSimilarTo = specificNobelPrizePhrase2, isBiSimilarTo = theDisciplinePrize]

% chemistry/physics/biology nobel prize
[LMNode: identifier=theDisciplinePrize, isInstanceOf=phrase, wordbase = {genericDiscipline11,nobelAdjective,prizeNoun},
isBiSimilarTo=specificNobelPrizePhrase2, isBiSimilarTo=thePrizeDiscipline]

% nobel chemistry/physics/biology prize
[LMNode: identifier=specificNobelPrizePhrase2, isInstanceOf=phrase, wordbase = {nobelAdjective,genericDiscipline11,prizeNoun}]
```

Figure 3.4: Textual representation of three different phrases in the LM, that are interconnected by similarity edges. Each of the phrases shown contains an instance of a *wordOptionSpecific*.

```
% first, second, third...
[LMNode: identifier=genericOrdinal11,isInstanceOf=wordOptionSpecific,word={genericOrdinal}]

% person/scientist/individual/laureate/researcher/woman/man
[LMNode: identifier=personSpecific11, isInstanceOf=wordOptionSpecific, word =
{personNoun,scientistNoun,individualNoun,womanNoun,manNoun,researcherNoun1,laureateNoun1}]

% chemistry/physics/biology
[LMNode: identifier=genericDiscipline11,isInstanceOf=wordOptionSpecific,word={chemistryNoun,physicsNoun,biologyNoun}]
```

Figure 3.5: Textual representation of three instances of *wordOptionSpecific* in the LM.

3.1.2.1 wordOption, wordOptionSpecific, wordPowerOption

As described in previous sections, each instance of *phrase* in the LM has a *wordBase* attribute which points to instances of *word* or *wordOption*. In addition to *wordOption*, two similar concepts exist in the LM, namely *wordOptionSpecific* and *wordPowerOption*. The *wordOptionSpecific* concept has been introduced for the purpose of enabling more generic phrases whilst maintaining specific similarity relations. The *wordOption* can be said to represent synonymity relations between words. If a phrase contains a *wordOption* in a certain position, it means that all the words in the *wordOption* are interchangeable in that specific phrase. On the contrary, if a phrase contains a *wordOptionSpecific* in a certain position, there may still be a list of different words that can be used in that position but for a similarity relationship to hold between this phrase and some other phrase, the other phrase must also contain the same *wordOptionSpecific*. As an example we may consider the phrases shown in Figure 3.4, and the definition of the *genericDiscipline11* node in Figure 3.5. As seen in Figure 3.5, *genericDiscipline11* is a *wordOptionSpecific* and represents either of the nodes {*chemistryNoun,physicsNoun,biologyNoun*}. It is not shown in the figure but these are all instances of *word*. The fact that *genericDiscipline11* is a *wordOptionSpecific* means that the phrase *thePrizeDiscipline* in Figure 3.4 can have either of the four spellings {*nobel prize in chemistry, nobel prize in physics, nobel prize in biology*}, but for the similarity relations to hold with the two other phrases shown in the figure, the *wordOptionSpecific* must be the same word in both phrases that the relation applies to. I.e. *"nobel prize in **chemistry**"* is similar to *"**chemistry** nobel prize"* but not to *"**physics** nobel prize"* etc. However, if *genericDiscipline11* had been a *wordOption* rather than a *wordOptionSpecific* the similarity would have been valid regardless of the discipline.

```
% the
[LMNode: identifier = theDeterminerPowerOption, isInstanceOf=wordPowerOption, word={theDeterminer}]

% that/who
[LMNode: identifier = thatWhoOption, isInstanceOf=wordOption, word={thatPronoun3, whoPronoun3}]
```

Figure 3.6: Example of an instance of *wordPowerOption* in the LM.

```
% share the prize/award
[LMNode: identifier=sharePhrase2, isInstanceOf=phrase, wordbase={shareVerb,theDeterminerPowerOption,prizeAwardOption}]

% share the prize/award with anyone/someone/anybody/somebody
[LMNode: identifier=sharePrizePhrase2, isInstanceOf=phrase, wordbase=
{shareVerb,theDeterminerPowerOption,prizeAwardOption,withPreposition,anyoneSomeoneOption},
isBiSimilarTo=sharePrizeAnyOtherPhrase2,isBiSimilarTo=sharePrizeAnotherPhrase2, isBiSimilarTo=sharePrizeOtherPhrase2,
isSimilarTo=sharePhrase2]
```

Figure 3.7: Textual representation of two phrases that each contains an instance of *wordPowerOption*.

Lastly, there is the *wordPowerOption* which makes a similarity relation valid for all word options even if one of the phrases involved has a *wordOptionSpecific*. An example where this can be used is for the phrases (a) *"share his/her/their prize with anyone"*, (b) *"share his/her/their prize with some other laureate"* and (c) *"share the prize with anyone"*. In phrases (a) and (b), the possessive pronouns should be *wordOptionSpecific* since the phrases are similar if and only if the pronouns match. Moreover, it may also be considered valid that, regardless of the pronoun option, both phrases (a) and (b) are similar to phrase (c). For that purpose the *wordPowerOption* has been defined. In this case, the word *the* in phrase (c) is declared as *wordPowerOption*, as seen in Figure 3.6 and Figure 3.7.

3.1.2.2 Generic instances of words

The generic *word* node has an attribute called *generic*. This attribute may be either *true* or *false*. Every instance that is an instance of *word* or an instance of a node that is a subcategory of *word* inherits the attribute *generic*. If an instance is generic it means that it represents *all* words in the LM of the given word class. Some generic nodes can be seen in Figure 3.8. If e.g. *genericVerbRoot* (which is seen in the figure) is part of a word base in a phrase, it means that any root verb can be used at that position in the given phrase. The concept of generic word instances makes it very convenient to create rules that automatically apply to an immense number of phrases.

3.1.2.3 The verbRestriction attribute

Instances of *phrase* have an attribute called *verbRestriction* which takes either of the values *true* and *false*. If *verbRestriction=true* for some phrase, it means that the verb forms of the verbs in the phrase are relevant to resolve ambiguity. If we e.g. consider the phrase *"Who received the Nobel prize?"*, we do not need to confirm the verb form of *received* because it is clear from the context that it has to be simple past. However, if the phrase is simply *"received the Nobel prize"*, the verb form is ambiguous as this phrase may be part of e.g. either the previously mentioned

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generic instances of word types:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[LMNode: identifier=genericWord,spelling=genericWord,isInstanceOf=word, generic=true, wordFrequency = 0.01935090]

%%% GENERIC VERB
[LMNode: identifier=genericVerbRoot,spelling=genericVerbRoot,isInstanceOf=verb,verbForm=root, generic=true, wordFrequency = 0.01935090]
[LMNode: identifier=genericVerbPastSimple,spelling=genericVerbPastSimple,isInstanceOf=verb,verbForm=pastSimple, generic=true, wordFrequency = 0.01935090]
[LMNode: identifier=genericVerbPastParticiple,spelling=genericVerbPastParticiple,isInstanceOf=verb,verbForm=pastParticiple, generic=true, wordFrequency = 0.01935090]


%%% GENERIC PRONOUN
[LMNode: identifier=genericPronounPersonalSubject,spelling=genericPronounPersonalSubject,isInstanceOf=pronoun,pronounType=personalSubject, generic=true, wordFrequency =
0.01935090]
[LMNode: identifier=genericPronounInterrogativeSubject,spelling=genericPronounInterrogative,isInstanceOf=pronoun,pronounType=interrogativeSubject, generic=true,
wordFrequency = 0.01935090]
[LMNode: identifier=genericPronounIndefiniteSubject,spelling=genericPronounIndefiniteSubject,isInstanceOf=pronoun,pronounType=indefiniteSubject, generic=true,
wordFrequency = 0.01935090]
[LMNode: identifier=genericPronounRelativeSubject,spelling=genericPronounRelativeSubject,isInstanceOf=pronoun,pronounType=relativeSubject, generic=true, wordFrequency =
0.01935090]

%%% GENERIC ORDINALS
[LMNode: identifier=genericOrdinal, spelling=genericOrdinal, isInstanceOf=integer, integerForm=ordinal, generic=true, wordFrequency = 0.01935090]

%%% GENERIC INTEGERS
[LMNode: identifier=genericInteger, spelling=genericInteger, isInstanceOf=integer, integerForm=natural, generic=true, wordFrequency = 0.01935090]

%%% GENERIC NOUN
[LMNode: identifier=genericNoun,spelling=genericNoun,isInstanceOf=noun, generic=true, wordFrequency = 0.01935090]
[LMNode: identifier=genericNounPlural, spelling=genericNounPlural, isInstanceOf=noun, nounForm=plural, generic=true, wordFrequency = 0.01935090]

%%% GENERIC ADJECTIVE
[LMNode: identifier=genericAdjective,spelling=genericAdjective,isInstanceOf=adjective, generic=true, wordFrequency = 0.01935090]
[LMNode: identifier=genericAdjectivePositive, spelling=genericAdjectivePositive, isInstanceOf=adjective, adjectiveForm=positive, generic=true, wordFrequency = 0.01935090]
```

Figure 3.8: Examples of instances of *generic*-type nodes in the textual representation of the LM.

sentence or the sentence *"Who has received the Nobel prize?"*, in which case the correct verb form of *received* would be past participle. For such ambiguous phrases in the LM, *verbRestriction* is set to *true* so that no processing occurs with this phrase until the verb form in the given context is confirmed.

3.2 Generating the language model

Data for specific word instances have been added to the LM by automatically parsing word data from Wiktionary[1]. As an example, the result of the parsing for the word *prize* is seen in Figure 3.9, where the first node is the singular noun *prize* and the second node is the plural noun form. After that follows six different versions of *prizeNoun* which are the six different senses of the word, as a noun, that are listed in Wiktionary. After the noun entries, entries for *prize* as a verb follow, for a number of different verb forms. The bottom nodes seen in the figure represent the adjective version.

In addition to parsing word entries from Wiktionary, external word frequency data has been integrated in the LM such that each word node contains a word frequency item. The word frequency data was computed by Wahde, as part of a previous project, by analysing a corpus containing $255,575$ distinct words. The corpus consisted of radio transcripts from American NPR, and relative word frequency was computed for each word in terms of number of occurrences per one million words. Taking into account (i) the size of the corpus (in terms of distinct words) and (ii) the fact that the corpus consists of spoken dialogue covering a wide spectrum of different subjects, the computed word frequencies are likely to be fairly representative of general every day use of the English language. The word frequency data was automatically added to the LM text document. The purpose of integrating word

[1] https://www.wiktionary.org/

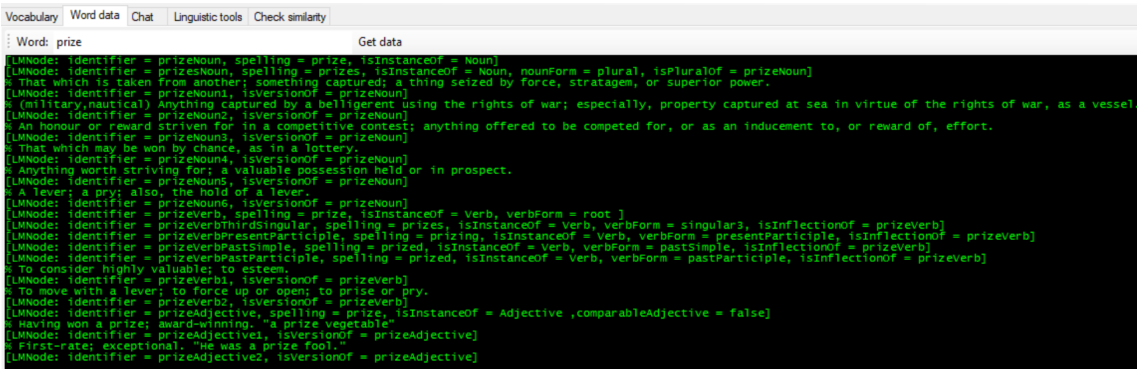


Figure 3.9: Output resulting from automatic parsing of word data from Wiktionary, for the word *prize*.

frequency data into the LM is to be able to perform text simplification, which will be discussed in more detail in subsequent sections.

As a starting point, Both words and phrases have been added to the LM (mainly) to cover a number of questions pertaining to the Nobel prize . These questions are the following:

(1) "Who won the Nobel prize in Physics in 1901?"
(2) "Which scientist received the Chemistry Nobel prize in 1903?"
(3) "In what year did Becquerel win the Nobel prize?"
(4) "Did Becquerel share his prize with anyone?"
(5) "How many women have won the Nobel prize twice?"
(6) "Which woman was the first one to receive a Nobel prize?"

Phrases relevant to the listed questions were added manually to the LM, but with the help of an automatic phrase search tool that has been previously developed by Wahde. The phrase search tool is a program that searches for phrases in a large corpus consisting of radio transcripts, based on some input parameters provided by the user, and outputs the phrases that it finds. More specifically, the tool searches for sentences in the corpus that contain certain words that are present in the input phrase, but do *not* contain certain other specified words. The input parameters include (i) *Paraphrase search*, which is the phrase to be searched for, (ii) *Target phrase*, which is the words in the phrase that should not be included in the search (i.e. in the output sentences), *Non-overlap penalty* which regulates how the output sentences should be ranked based on their similarity with the original phrase. An example usage of the phrase search tool is seen in Figure 3.10, where the phrase to be searched for is *"nobel prize"* (at the start of the program, all letters in the corpus are converted to lower case). In the figure, both words are marked with the * symbol, meaning that both words are *required* in the output sentence. No target phrase was specified in this case so the search is practically simply a search for sentences that include *nobel prize*. The output provides a systematic way to add relevant phrases to the model in the context of the Nobel prize.

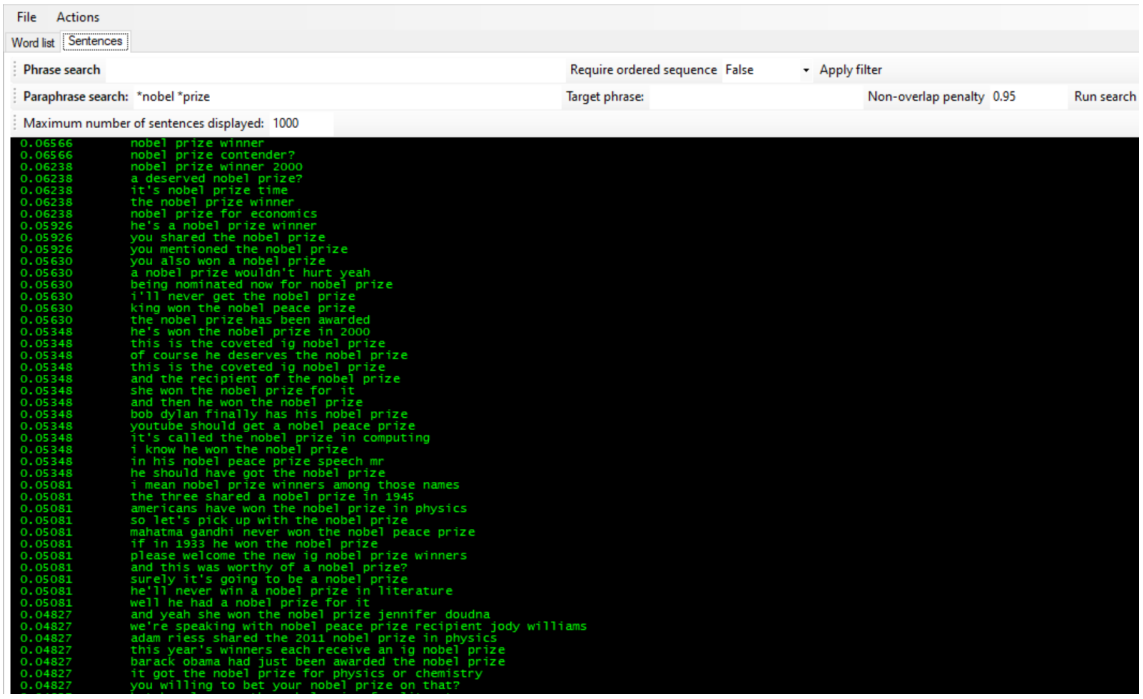


Figure 3.10: Example usage of a phrase search tool that was used to search for relevant phrases in a large corpus.

3.3 Developing four methods for NLP

In this section, the four NLP methods (the development of which constitutes the second objective of this thesis, see 1.1.2) are presented. As stated earlier, these methods include (a) a POS tagger method that takes an input sentence and outputs the correct POS of each word in the sentence, (b) a simplification method that takes an input sentence and rephrases it to a lexically and syntactically simpler sentence whilst maintaining the semantic meaning of the original sentence, (c) a semantic similarity analysis method that takes two sentences as input and outputs whether the two sentences are semantically similar or not, and (d) a paraphrasing method that takes a sentence as input and outputs a paraphrased (but semantically similar) version of the sentence. The four methods have the names (a) **Disambiguate**, (b) **Simplify**, (c) **CheckSimilarity**, and (d) **Paraphrase**, and the implemented algorithms of these are presented as pseudocode in 3.3.5-3.3.8. The rest of the chapter is dedicated to, first, giving a general overview of the implementation and, subsequently, explaining relevant auxiliary methods and classes that are referred to in the algorithms.

3.3.1 General overview of the implementation

A graphical user interface (GUI) has been implemented, through which a user can run the four NLP methods that have been developed for this thesis. The NLP application is implemented such that all of the four NLP methods are placed in a class called LanguageProcessor(), which also contains a number of auxiliary meth-

ods. When the program is started from the GUI and the LM has been loaded by the user, an instance of LanguageProcessor() is created with the LM as input. The user is then able to run any of the four NLP methods separately from the GUI.

The general procedure in all four NLP methods is such that the input sentence (that is provided by the user) is first cleaned and processed into a list of n-grams. Note that the term *sentence* is used here but there is no requirement that the input text is an actual full sentence. Subsequently, the input n-grams are iterated through in descending order of length and for each n-gram, the program performs a binary search over the phrases that exist in the LM to find phrases that have the same spelling as the current n-gram. If such a phrase is found and it fulfills certain requirements (e.g. pertaining to verb forms), phrases in the LM that are *similar* to this phrase (i.e. connected to this phrase's node by an *isSimilarTo* edge or an *isBiSimilarTo* edge) are collected in a list (except in the POS tagging method where no paraphrasing occurs and thus no similarity relations need to be considered). Among all the *similar* phrases, one is selected based on certain criteria (the nature of which depends on the specific NLP task) and the original sentence is paraphrased by substituting the current n-gram with the phrase that was provided and selected from the LM. If no *similar* phrase is found that meets the criteria, no paraphrasing occurs for this n-gram and the algorithm simply continues to the next iteration. (Note that a phrase can have multiple equivalent spellings, i.e. it may contain instances of *wordOption*. If that is the case for the matching phrase, the alternative spellings of that phrase are also considered *similar* phrases.) In the POS tagging method, instead of retrieving *similar* phrases, any matching phrase that meets the relevant criteria is used for disambiguating the input, i.e. storing POS information, but not for paraphrasing.

In all four methods, a recursive call is made every time a paraphrasing or a disambiguation update is carried out. I.e., when a *similar* phrase has been selected and the n-gram has been substituted with the new phrase or the disambiguation data of the input has been updated, a recursive call is made on the new version of the sentence. In that way, the original sentence is iteratively and recursively updated to a new version. The recursion continues until no further updates are made, or until a specified number of maximum recursive calls has been made.

Each algorithm is described in more detail in the following sections.

3.3.2 Preprocessing

This section describes the text preprocessing that is implemented in all four NLP methods.

3.3.2.1 The Clean() method

Before any further processing of the input text is done, the input is cleaned. The cleaning takes place in a method referred to as **Clean**, and consists of (ii) removing

characters in the input sentence that are not alphanumeric or white space, and (ii) converting all remaining characters to lower case.

3.3.2.2 The Tokenize() method

After cleaning, the input sentence is tokenized which means that the sentence is split into a list of the tokens that make up the sentence, in the order that they occur in the sentence. Throughout the whole implementation, the tokens are the individual words that make up the sentence, i.e. tokens are sequences of characters separated by white space. The tokenization step is referred to as **Tokenize** in the algorithms in the subsequent sections.

3.3.2.3 The MakeNgrams() method

As briefly mentioned in 2.5, dividing a text into n-grams (for some range of values of n) is a commonly used approach in language modelling. The n-gram technique plays a central part in the present implementation. In all four NLP methods, the input sentence is processed by iteratively comparing each n-gram in the input with existing phrases in the LM. The n-grams are iterated through in descending order of length (since longer phrases contain more information) and in the natural order in which they occur in the input sentence. In each of the four NLP methods, the n-grams are generated at an early stage as part of the preprocessing of the input sentence. This is done by calling the **MakeNgrams** method on the tokenized input sentence. In the **MakeNgrams** method, the n-grams are formed by iterating through the tokenized input and saving sequences of consecutive words, one n-gram length at a time. The method results in a list of tuples. Each tuple consists of an integer that represents the n-gram length and a list containing all the n-grams in the input of that length, in the order that they occur in the input. The list of tuples is sorted in order of n-gram length.

3.3.3 Key classes

All programming methods that were developed for this thesis were implemented in the object-oriented class-based programming language C#. This section describes some key classes that each constitutes a central part of the functionality throughout all four NLP methods.

3.3.3.1 The Disambiguation class

As stated in 2.2.1, POS tagging is an important backbone of many NLP tasks as it is a very useful tool to disambiguate the linguistic structure of a text and thus facilitate higher level processing. Furthermore, it has been shown that preprocessing a text by performing POS tagging increases the performance of word sense disambiguation of the words in the text [102]. The idea of addressing seemingly separate NLP tasks as a whole has been expressed and implemented by several researchers [103] [104] [28]. L. Padró states that ”it seems logical that the more information we have, the better results we will produce at a given task [...] this is more or less what

we humans do when understanding a natural language utterance: we use all kinds of information –lexical, syntactical, semantic, etc" [28].

In line with this reasoning, a key aspect of the program that has been developed in this thesis is the implementation of a class called Disambiguation. In each of the four implemented NLP methods, an instance of Disambiguation is passed as input, then modified and updated as the method proceeds, and passed along in every recursive call. The instance of Disambiguation holds all the current information about the (possibly rephrased) input sentence. In that way all information that becomes accessible about the input sentence as the method proceeds is collected, used, and refined as the method progresses. The Disambiguation class keeps track of (i) the words in the sentence (as a list of words), (ii) the LM nodes of the respective words (as a list of nodes), and (iii) the retrieved disambiguation information about the sentence, i.e. the POS tags for the respective words (which are stored in the nodes). In this way, when processing the input sentence and searching for a matching phrase in the LM, matching can be done not only based on spelling but also on POS tag information. This results in a more refined matching procedure and opens up for more matching possibilities; e.g. with phrases that have the *verbRestriction=true* attribute and thus require information about verb forms; as mentioned previously, these phrases cannot be matched solely based on spelling but also require POS information.

The Disambiguation class has the method **Update**, which, when a phrase has been found in the LM that matches an n-gram in the processed sentence, updates the node list and the connecting disambiguation information of the sentence. Additionally, the Disambiguation class has the method **RephraseUpdate**, which first rephrases the current sentence (by substituting the given n-gram with the given semantically equivalent phrase, that are both passed as input to the method) and then updates the node list, word list, and disambiguation data accordingly.

The Disambiguation class also has the **CheckRestriction** method, which is further described in 3.3.4.

3.3.3.2 The ExpandedPhrase class

Since a phrase in the LM may contain not only specific words but also instances of *wordOption* and/or *wordOptionSpecific* (see 3.1.2.1), a single phrase may have multiple possible spellings. For example, we may have the following node in the LM:

$$\begin{aligned}&\text{[LMNode: identifier=example, isInstanceOf=phrase, wordbase=}\\&\qquad\text{\{whoPronoun, wonOption, theDeterminer, prizeNoun\}]}\end{aligned} \tag{3.1}$$

Consider that *whoPronoun*, *theDeterminer*, and *prizeNoun* are all identifiers of word instance nodes, such as the one in 3.2, and *wonOption* is an instance of *wordOption*, and thus has a node such as the one in 3.3 (where the node *getVerbPastSimple* has the spelling "got").

[LMNode: identifier=whoPronoun, spelling=who, isInstanceOf=Pronoun, wordFrequency=3172.54184270] (3.2)

[LMNode: identifier=wonOption, isInstanceOf=wordOption, word={winVerbPastSimple,getVerbPastSimple}] (3.3)

Then, the example phrase (3.1) has two possible spellings, namely (i) *who won the prize* and (ii) *who got the prize*. Thus, a phrase in the LM is often a compact representation of multiple spellings. This means that in order to search for a given phrase based on spelling, all phrases in the LM need to be *expanded*, i.e. all possible spellings of every phrase need to be generated and stored, whilst maintaining the connection to the node of the phrase (so that the program still knows which node a given spelling belongs to).

For the purpose of keeping track of the spellings and other relevant information about a phrase, a class called ExpandedPhrase was created. An instance of ExpandedPhrase has (i) the spelling of the given *expanded* version of the phrase, (ii) the node of the phrase, (iii) the node list of the compact representation of the phrase (such as the word base in 3.1), (iv) the node list of the *expanded* representation of the phrase (such as {*whoPronoun*, *getVerbPastSimple*, *theDeterminer*, *prizeNoun*} in the case of one of the spellings of the example phrase 3.1), and (v) a list of word frequencies of the words in the spelling of the given version of the phrase.

3.3.4 Handling phrases

The core functionality of the implemented NLP methods described here is the handling of phrases, the implementation of which is also the main novelty of the LM presented here. In this section, the most relevant stages of phrase handling are described.

3.3.4.1 Phrases in the language model

When the LM is loaded from the GUI, the text file that represents the LM is parsed and the nodes that have the attribute *isInstanceOf=phrase* are collected in a list called *phraseList*. An instance of a class called LanguageModel is created and *phraseList* is a field belonging to this instance. This means that whenever an instance of LanguageModel exists within a certain scope, the *phraseList* can be accessed. In the algorithms that are presented further on in this thesis, the retrieval of the phrase list is represented by the command LanguageModel.*phraseList*.

3.3.4.2 The MakePhraseDictionary method

After the phrases have been retrieved from the LM by calling LanguageModel.*phraseList*, they are organized into a so-called dictionary consisting of key-value pairs. This step is referred to as the **MakePhraseDictionary** method. The dictionary contains all

the phrases in *phraseList*, arranged such that each *key* is a phrase length (number of words in a phrase) and the corresponding *value* is a list containing the phrases with this length in the LM. Organizing the phrases in a dictionary enables fast searching for phrases with a specific length in the LM.

3.3.4.3 The ExpandPhrases/ExpandConditionalPhrase methods

After phrases of a given length have been retrieved from the dictionary containing all phrases in the LM, these phrases are expanded such that all possible spellings are represented in a list of ExpandedPhrase instances. This step is carried out in the **ExpandPhrases** method and is done by generating all possible combinations of any present wordOptions/wordOptionSpecifics in a given phrase. The purpose of this step is to be able to search for phrases in the LM that match the current input n-gram.

In **Simplify**, **CheckSimilarity**, **Paraphrase**, the procedure is such that when a matching phrase is found in the LM, the phrases that are *similar* to the matching phrase are also retrieved and expanded. The expansion of the *similar* phrases is done in a method called **ExpandConditionalPhrase**. In this method, an expanded spelling is only considered valid if any wordOptionSpecific that is present in a *similar* phrase is also present in the input phrase. This procedure differs from the one in **ExpandPhrases** in which all candidate nodes in any wordOptionSpecific set are expanded and considered for further analysis.

Both **ExpandPhrase** and **ExpandConditionalPhrase** respectively result in a list of instances of the ExpandedPhrase class, sorted alphabetically based on the spelling field.

3.3.4.4 The ExpandFrequencies method

In **Simplify**, any paraphrasing is done (partially) based on word frequencies of the words in a given phrase. The word frequencies of the words in an expanded phrase are retreived in a method called **ExpandFrequencies**. This method simply takes the nodes in the node list of the expanded phrase instance and retrives the word frequency from the LM for each node.

3.3.4.5 The CheckRestriction method

A given phrase in the LM may have the attribute *verbRestriction=true* which means that this phrase can only be matched with an n-gram if it is ensured that the verb forms in the phrase are consistent with the verb forms in the n-gram. E.g. consider the n-gram *won the prize* in the input sentence *Who won the prize?*. Only considering spelling, this n-gram would match with the following phrase:

$$\begin{aligned}&\text{[LMNode: identifier=example2, isInstanceOf=phrase, wordbase=}\\&\qquad\text{\{wonOption2, theDeterminer, prizeNoun\}]}\end{aligned} \tag{3.4}$$

In this case, *wonOption2* is the following wordOption:

$$\begin{aligned}&\text{[LMNode: identifier=wonOption2, isInstanceOf=wordOption,}\\&\qquad\qquad\text{word=\{winVerbPastParticiple,getVerbPastParticiple\}],}\end{aligned} \tag{3.5}$$

where *winVerbPastParticiple* is the past participle of *win* which has the spelling *won* and *getVerbPastParticiple* is the past participle of *get*, which (in America) has the spelling *gotten*. This means that, without using any restriction on verb form, the n-nram *won the prize* in the input sentence could be substituted with *gotten the prize* and result in the sentence *Who gotten the prize?* which is grammatically incorrect.

Thus, the attribute *verbRestriction* is an important feature to distinguish between eligible and ineligible phrases in any paraphrasing or disambiguation stage. When a phrase is matched with an input n-gram based on spelling, the *verbRestriction* attribute is always checked. If *verbRestriction=true* the method **CheckRestriction** is called. In this method, the nodes in the matching phrase are compared with the disambiguation data (see 3.3.3.1) of the input sentence. If no information about verb forms is available in the disambiguation data, or if the verb forms differ between the matching phrase and the input sentence disambiguation, the verb restriction is not considered fulfilled and the matching phrase is not passed on for further analysis.

3.3.4.6 The GetSimilarToNodes method

When a phrase in the LM has been found to match a given n-gram in the input sentence, the phrases that are *similar* to the matching phrase are retrieved. This is done by calling the **GetSimilarToNodes** method. In **Paraphrase** and **Simplify**, only output edge similarity (including bisimilarity) is considered since, in paraphrasing, an n-gram should not be substituted with a phrase that it is not similar to, even if that phrase, in turn, is similar to the n-gram. e.g. the phrase *"in what year"* can be substituted with *"when"* (even if the question becomes less specific) but the opposite is not true since *"when"* might, more appropriately, refer to *"at what time"* or *"on which day"* etc, depending on context. Similarly, *"Which scientist won the prize?"* is similar to *Who won the prize?"*, but the opposite is not necessarily true since the recipient does not have to be a scientist (given the available context).

Whether only output similarity or both directions of similarity should be considered is passed to **GetSimilarToNodes** as an input boolean argument. In **CheckSimilarity** both directions of similarity is considered as this method analyses similarity of two sentences rather than paraphrasing. In that context, the sentences *"In what year did Becquerel win the Nobel prize?"* and *"When did Becquerel win the Nobel prize?"*, are considered similar.

3.3.5 The part-of-speech tagging algorithm

This algorithm takes a sentence as input and outputs POS information about each word in the input, if such information for the given word could be found in the LM.

The algorithm proceeds by iteratively and recursively trying to match the n-grams in the input with existing phrases in the LM. The iterative matching is done by iterating through the n-grams in the input in descending length order and, for each n-gram, searching for phrases in the model that the n-gram matches with, based on spelling. If such a phrase is found, and any existing phrase restrictions are fulfilled, the disambiguation information of the input is updated with the information that is contained in the found phrase. If this update leads to an expansion of the already known disambiguation information, a recursive call is made with the updated disambiguation information as input. Otherwise, the iteration through the input n-grams is simply continued.

The overall method is divided into two algorithms, Algorithm 1 which conducts some necessary preprocessing of the input sentences and prepares a dictionary containing the phrases in the LM, and Algorithm 2 which recursively disambiguates the input.

Algorithm 1 Call POS tagger algorithm

```
1: procedure CallDisambiguate(input)                                    ▷
2:     cleanedInput ← Clean(input)
3:     inputWords ← Tokenize(cleanedInput)
4:     disambiguation ← new Disambiguation(inputWords)
5:     phraseList ← languageModel.PhraseList // get list of all the phrases in the
       LM
6:     phraseDictionary ← MakePhraseDictionary(phraseList)
7:     disambiguation ← Disambiguate(disambiguation, phraseDictionary)
8:     return disambiguation
9: end procedure
```

Algorithm 2 POS tagger algorithm

```
1: procedure Disambiguate(string input)                                 ▷
2:     inputNgrams ← MakeNgrams(disambiguation.WordList)
3:     for each ngram length (in descending order) do
4:         ngramsCurrentLength ← get the ngrams in inputNgrams with the current
           length
5:         phrasesCurrentLength ← get the phrases with the current length in
           phraseDictionary
6:         expandedPhrasesCurrentLength ←
           ExpandPhrases(phrasesCurrentLength)
7:         for each ngram in ngramsCurrentLength do
8:             matchingPhrases ←
               expandedPhrasesCurrentLength.BinarySearch(ngram)
9:             for each phrase in matchingPhrases do
10:                isRestrictedPhrase ← true if current phrase has verbRestriction = true,
                   otherwise false
11:                if isRestrictedPhrase then
12:                    isOkay ← CheckRestriction(phrase, ngram)
```

```
13:                 end if
14:                 if isOkay or not isRestrictedPhrase then
15:                     disambiguation ← disambiguation.Update(phrase)
                        // update with the information that exists in the matching phrase
16:                     if any new update occured then
17:                         disambiguation ← Disambiguate(disambiguation, phraseDictionary)
18:                         return disambiguation
19:                     end if
20:                 end if
21:             end for
22:         end for
23:     end for
24:     return disambiguation
25: end procedure
```

3.3.6 Similarity analysis

This algorithm takes two sentences, *sentence1* and *sentence2*, as input and outputs *true* if they are semantically similar, and *false* if they are not. The idea of this algorithm is that one of the sentences, *sentence1*, is considered the *template* and then the algorithm iteratively and recursively tries to paraphrase the other sentence, *sentence2*, until it is equal to *sentence1* or until all n-grams in *sentence2* have been iterated through and no further substitutions of phrases can be made. The iterative paraphrasing is done by iterating through the n-grams in *sentence2* in descending length order and, for each n-gram, searching for phrases in the model that the n-gram is similar to and thus can be substituted with, such that the original sentence becomes paraphrased but retains its semantic meaning. Paraphrasing of *sentence2* occurs if (i) the found similar phrase exists in the *template* sentence or (ii) if the found similar phrase contains a verb that exists in the *template* but is missing in *sentence1* (the idea of this is that the algorithm sometimes has to find the correct verb first and can then, in a subsequent step, change the verb form). As soon as *sentence2* has become paraphrased such that it is equal to *sentence1* (i.e. has the same words, in the same order as *sentence1*), the algorithm returns *true*; otherwise, at the end of the algorithm, it returns *false*.

The overall method is divided into two algorithms, Algorithm 3 which conducts some necessary preprocessing of the input sentences and prepares a dictionary containing the phrases in the LM, and Algorithm 4 which recursively tries to paraphrase *sentence2* such that it becomes more similar to *sentence1*.

Algorithm 3 Call similarity analysis algorithm

```
1: procedure CallCheckSimilarity(sentence1, sentence2)                ▷
2:     cleanedInput ← Clean(sentence1)
3:     cleanedTemplate ← Clean(sentence2)
4:     disambiguationInput ← Disambiguate(cleanedInput)
5:     disambiguationTemplate ← Disambiguate(cleanedTemplate)
```

```
    phraseList ← languageModel.PhraseList // get list of all the phrases in the LM
    phraseDictionary ← MakePhraseDictionary(phraseList)
    templateWords ← Tokenize(cleanedTemplate)
    ngramsTemplate ← MakeNgrams(templateWords)
    isSimilar ← CheckSimilarity(cleanedInput, cleanedTemplate, templateWords, ngramsTemplate, cleanedInput, phraseDictionary, disambiguationInput, disambiguationTemplate)
    return isSimilar
end procedure
```

Algorithm 4 Similarity analysis algorithm

```
procedure CheckSimilarity(previousVersion, cleanedTemplate, templateWords, ngramsTemplate, cleanedInput, phraseDictionary, disambiguation, disambiguationTemplate) ▷
    if (cleanedInput = cleanedTemplate) then
        return true
    end if
    inputWords ← Tokenize(cleanedInput)
    inputNgrams ← MakeNgrams(inputWords)
    for each ngram length (in descending order) do
        ngramsCurrentLength ← get the ngrams in inputNgrams with the current length
        phrasesCurrentLength ← get phrases with current length in phraseDictionary
        expandedPhrasesCurrentLength ← ExpandPhrases(phrasesCurrentLength)
        for each ngram in ngramsCurrentLength do
            matchingPhrases ← expandedPhrasesCurrentLength.BinarySearch(ngram)
            for each phrase in matchingPhrases do
                isRestrictedPhrase ← true if current phrase has verbRestriction = true, otherwise false
                if isRestrictedPhrase then
                    isOkay ← CheckRestriction(phrase, ngram)
                end if
                if (isOkay or not isRestrictedPhrase) then
                    disambiguation ← disambiguation.Update(phrase) // update with information in matching phrase
                    similarToPhrases ← GetSimilarToNodes(phrase, considerInputSimilarity = true)
                    for each phrase in similarToPhrases do
```

```
22:                         phraseSpellings ← ExpandConditionalPhrase(phrase)
23: | | | | | | for each spelling in phraseSpellings do
24:                             hasMissingVerb ← CheckMissingVerbs(disambiguation,
                                disambiguationTemplate, phrase)
25:                             existsInTemplate ← true if new phrase exists in
                                cleanedTemplate
26: | | | | | | | if existsInTemplate or hasMissingVerb then
27:                                 disambiguation ←
                                    disambiguation.RephraseUpdate(phrase, ngram)

28: | | | | | | | | if (updated sentence == cleanedTemplate) then
29:                                     return true
30: | | | | | | | | else if (updated sentence not equal to previous
31: version AND not equal to version before previous version) then
32:                                     isSimilar ←
                                        CheckSimilarity(previousVersion, cleanedTemplate,
                                        templateWords, ngramsTemplate,
                                        cleanedInput, phraseDictionary, disambiguation,
                                        disambiguationTemplate)
33:                                     return isSimilar
34: | | | | | | | | end if
35: | | | | | | end if
36: | | | | | end for
37: | | | | end for
38: | | | end if
39: | | end for
40: | end for
41: end for
42:     return false
43: end procedure
```

3.3.7 Paraphrasing

Similarily to the previous methods, this method is divided into two submethods: (i) one that does some preprocessing and calls the recursive paraphrasing method, and (ii) the recursive paraphrasing method itself.

The idea of this method is that it, with a certain probability referred to as *paraphrasing probability*, performs a random, but eligible, paraphrasing of the input sentence, such that the semantic meaning of the original sentence is retained. Randomisation is incorporated at four stages of the recursive algorithm: (i) when choosing a random n-gram length to start the iteration from, (ii) when choosing randomly among the phrases in the LM that are found to match the spelling of the input (and that fulfill any relevant restrictions), (iii) when a random matching phrase in the LM has been chosen, a random decimal number between 0 and 1 is generated; if this number is less than the *paraphrasing probability*, a (iv) random spelling is chosen among the

similar phrases of the matching phrase. The variables in the algorithm that are assigned by means of randomisation are marked in red in Algorithm 6 to facilitate for the reader.

Algorithm 5 Call paraphrasing algorithm

```
1: procedure CallParaphrase(input)                                   ▷
2:     cleanedInput ← Clean(sentence1)
3:     disambiguation ← Disambiguate(cleanedInput)
4:     phraseList ← languageModel.PhraseList // get list of all the phrases in the
       LM
5:     phraseDictionary ← MakePhraseDictionary(phraseList)
6:     paraphrasedInput = Paraphrase(cleanedInput, phraseDictionary,
       cleanedInput, 0, disambiguation)
7:     return isSimilar
8: end procedure
```

Algorithm 6 Paraphrasing algorithm

```
1: procedure Paraphrase(cleanedInput, phraseDictionary, originalInput, num-
   berOfIterations, disambiguation)
2:     previousVersion ← cleanedInput
3:     inputWords ← Tokenize(cleanedInput)
4:     inputNgrams ← MakeNgrams(inputWords)
5:     randomMaxNgramLength = random integer in (0, length(inputWords))
6:     for each ngram length <
                randomMaxNgramLength (in descending order) do
7:         ngramsCurrentLength ← get the ngrams in the input with the current
           length
8:         phrasesCurrentLength ← get the phrases with the current length in
           phraseDictionary
9:         expandedPhrasesCurrentLength ← ExpandPhrases(phraseList)
10:        for each ngram in ngramsCurrentLength do
11:            matchingPhrases ←
               expandedPhrasesCurrentLength.BinarySearch(ngram)
12:            for each phrase in matchingPhrases do
13:                isRestrictedPhrase ← true if current phrase has
                   verbRestriction = true, otherwise false
14:                if isRestrictedPhrase then
15:                    isOkay ← CheckRestriction(phrase, ngram)
16:                end if
17:                if (isOkay or not isRestrictedPhrase) then
18:                    allMatchingPhrases.Add(phrase)
19:                    disambiguation ← disambiguation.Update(phrase)
20:                end if
21:            end for
22:        end for
23:        if allMatchingPhrases is not empty then
```

```
24:             randomMatchingPhrase ← choose a matching phrase randomly from
                allMatchingPhrases
25:             similarToPhrases ←
                GetSimilarToNodes(phrase, considerInputSimilarity = false)
26: | | |       for each phrase in similarToPhrases do
27:                 currentPhraseSpellings ← ExpandConditionalPhrase(phrase)
28:                 phraseSpellings.Add(currentPhraseSpellings)
29: | | |       end for
30:             p ← random number in (0, 1)
31: | | |       if p <= paraphraseProbability AND
                  phraseSpellings not empty          then
32:                 similarPhrase ← chooose random phrase in phraseSpellings
33:                 paraphrasedSentence ← disambiguation.RephraseUpdate(similarPhrase)
34:                 numberOfIterations ← numberOfIterations + 1
35: | | | |       if paraphrasedSentence not equal to oldVersion then
36:                     paraphrasedSentence ← Paraphrase(paraphrasedSentence, phraseDic-
                        tionary, originalInput, numberOfIterations, Disambiguation)
37:                     return paraphrasedSentence
38: | | | |       end if
39: | | |       end if
40: | |       end if
41: |     end for
42: |     if paraphrasedSentence not equal to originalInput OR numberOfIterations > 10 then
43:         return paraphrasedSentence
44: |     else
45:         numberOfIterations ← numberOfIterations + 1
46:         paraphrasedSentence ← Paraphrase(paraphrasedSentence, phraseDic-
            tionary, originalInput, numberOfIterations, Disambiguation)
47:         return paraphrasedSentence
48: |     end if
49: end procedure
```

3.3.8 Text simplification

The simplification method is, similarly to **Paraphrase**, **CheckSimilarity**, and **Disambiguate**, divided into two submethods (i) one for preprocessing and for calling a recursive simplification method, and (ii) the recursive simplification method itself. The simplification algorithm works similarly to previous methods by iterating through the n-grams in the input sentence (in descending order of length) and, for each n-gram, searches for matching phrases in the LM. The simplification method, however, has the additional criterion that the input sentence is paraphrased only if the paraphrased version results in a *simpler* sentence than the original one.

As described in 2.2.2, text simplification can be carried out on a number of different levels. In this thesis, the implemented simplification is a combination of a lexical and syntactic approach carried out at phrase level, thus taking context and

grammatical cohesion into account, unlike most automatic LS systems of today [42]. In Figure 2.1, the typical lexical simplification pipeline of current systems was presented. The present implementation adheres to this pipeline to a large extent, however not exactly. In the present implementation, substitution generation is carried out for every word in the input without first identifying complex words. Word sense disambiguation is a built-in feature considering that all substitutions are carried out at phrase level, as well as considering the POS disambiguation procedure. Synonym ranking is carried out by comparing simplicity scores of all retrieved *similar* phrases.

In this thesis, degree of simplicity of a phrase is measured in terms of the average word frequency of the words in the phrase, and the number of words in the phrase. I.e. generally, *phraseX* is considered simpler than *phraseY* if the average word frequency of the words in *phraseX* is higher than the average word frequency of *phraseY*, *unless phraseY* is much shorter. To account for the wide range in word frequency (the word *the* occurs 50 000 times per one million words compared to *was* which occurs 6 700 times) the frequencies are normalized by taking the base 10 logarithm. The exact expression used for phrase simplicity in the algorithm is

$$\frac{\log_{10}(wordFrequencyAverage)}{numberOfWords} \tag{3.6}$$

This expression is partly inspired by the Swedish readability index LIX [105] which is based on the following formula (*words* and *sentences* refer to the number of words and number of sentences in the text; a word is considered long if it contains more than 6 characters [105]):

$$\frac{words}{sentences} + \frac{(long\ words) \cdot 100}{words}, \tag{3.7}$$

and the automated readability index (ARI) for English texts [106] in which the following formula is used (where *characters* refers to the number of characters in the text):

$$\mathrm{GL} = 4.71\ \frac{characters}{words} + 0.5\ \frac{words}{sentences} - 21.43. \tag{3.8}$$

The ARI is used to determine the appropriate grade level (GL) of English school texts. The ARI formula (3.8) has been empirically established by means of multiple regression and correlating each factor with already assigned grade levels of school texts [106]. For both LIX and ARI, a lower index indicates a simpler text and vice versa. The ideas of the formulas are similar; a text that has long words and many words per sentence is considered complicated. However, they do not take commonness of words into account, unlike many other implementations of text simplification systems (see 2.2.2). Since the simplicity score in the present implementation is used on singular phrases rather than on texts containing multiple sentences, it is not relevant to include number of sentences in the formula. In future work, word length (or other parameters) could be included, but for this thesis word frequency and number of words was considered sufficient and eligible as a simplicity measure.

Furthermore, if one (or both) of the phrases in a comparison only contains one

word, the minimum word frequencies of the phrases are compared and the phrase with the highest minimum word frequency is considered the simplest.

Algorithm 7 Call simplification algorithm

```
1: procedure CallSimplify(input)                                          ▷
2:     cleanedInput ← Clean(input)
3:     disambiguation ← Disambiguate(cleanedInput)
4:     phraseList ← languageModel.PhraseList // get list of all the phrases in the
       LM
5:     phraseDictionary ← MakePhraseDictionary(phraseList)
6:     simplification = Simplify(cleanedInput, cleanedInput,
       phraseDictionary, disambiguation)
7:     return simplification
8: end procedure
```

Algorithm 8 Sentence simplification algorithm

```
1: procedure Simplify(previousVersion, cleanedInput, phraseDictionary, disam-
   biguation)                                                             ▷

2:     simplification ← cleanedInput
3:     inputWords ← Tokenize(cleanedInput)
4:     inputNgrams ← MakeNgrams(inputWords)
5:  |  for each ngram length (in descending order) do
6:  |      ngramsCurrentLength ← get the ngrams in
       |      inputNgrams with the current length
7:  |      phrasesCurrentLength ← get phrases with current length in
       |      phraseDictionary
8:  |      expandedPhrasesCurrentLength ←
       |      ExpandPhrases(phrasesCurrentLength)
9:  |  |  for each ngram in ngramsCurrentLength do
10: |  |      matchingPhrases ←
       |  |      expandedPhrasesCurrentLength.BinarySearch(ngram)
11: |  |  |  for each phrase in matchingPhrases do
12: |  |  |      isRestrictedPhrase ← true if current phrase has
       |  |  |      verbRestriction = true, otherwise false
13: |  |  |  |  if isRestrictedPhrase then
14: |  |  |  |      isOkay ← CheckRestriction(phrase, ngram)
15: |  |  |  |  end if
16: |  |  |  |  if (isOkay or not isRestrictedPhrase) then
17: |  |  |  |      disambiguation ← disambiguation.Update(phrase) // update with
       |  |  |  |      the information
       |  |  |  |      in the matching phrase
18: |  |  |  |      similarToPhrases ←
       |  |  |  |      GetSimilarToNodes(phrase, considerInputSimilarity = true)
19: |  |  |  |      ngramWordFrequencies ← GetWordFrequencies(phrase)
20: |  |  |  |  |  for each phrase in similarToPhrases do
```

```
21:                         phraseSpellings ← ExpandConditionalPhrase(phrase)
22:                         phraseSpellings ← ExpandFrequencies(phraseSpellings)
23:                         expandedSimilarPhrases.Add(phraseSpellings)
24:     |   |   |   |   |   end for
25:                         simplestSimilarPhrase ← the phrase in
                            expandedSimilarPhrases that has the highest simplicity score
26:     |   |   |   |   |   if simplestSimilarPhrase is simpler than ngram then
27:                         beforeSimplification ← simplification
28:                         simplification ← disambiguation.
                            RephraseUpdate(previousVersion, simplestSimilarPhrase, ngram)
29:     |   |   |   |   |   |   if simplification not equal to previousVersion then
30:                             previousVersion ← beforeSimplification
31:                             simplification ← Simplify(previousVersion,
                                simplification, phraseDictionary, disambiguation)
32:     |   |   |   |   |   |   end if
33:     |   |   |   |   |   end if
34:     |   |   |   |   end if
35:     |   |   |   end for
36:     |   |   end for
37:     |   end for
38:         return simplification
39: end procedure
```

4

Results and Discussion

This chapter presents some results of applying the implemented methods that were presented in Chapter 3, followed by a discussion.

4.1 Results

This section first presents the results of (i) paraphrasing, (ii) simplification, and (iii) POS disambiguation, followed by (iv) the results of semantic similarity analysis and, finally, (v) some demonstration of generic phrases.

4.1.1 Paraphrasing, text simplification & POS disambiguation

Figure 4.1 shows the output of the implemented program when the sentence *"Who won the Nobel prize in Physics in 1901?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated.

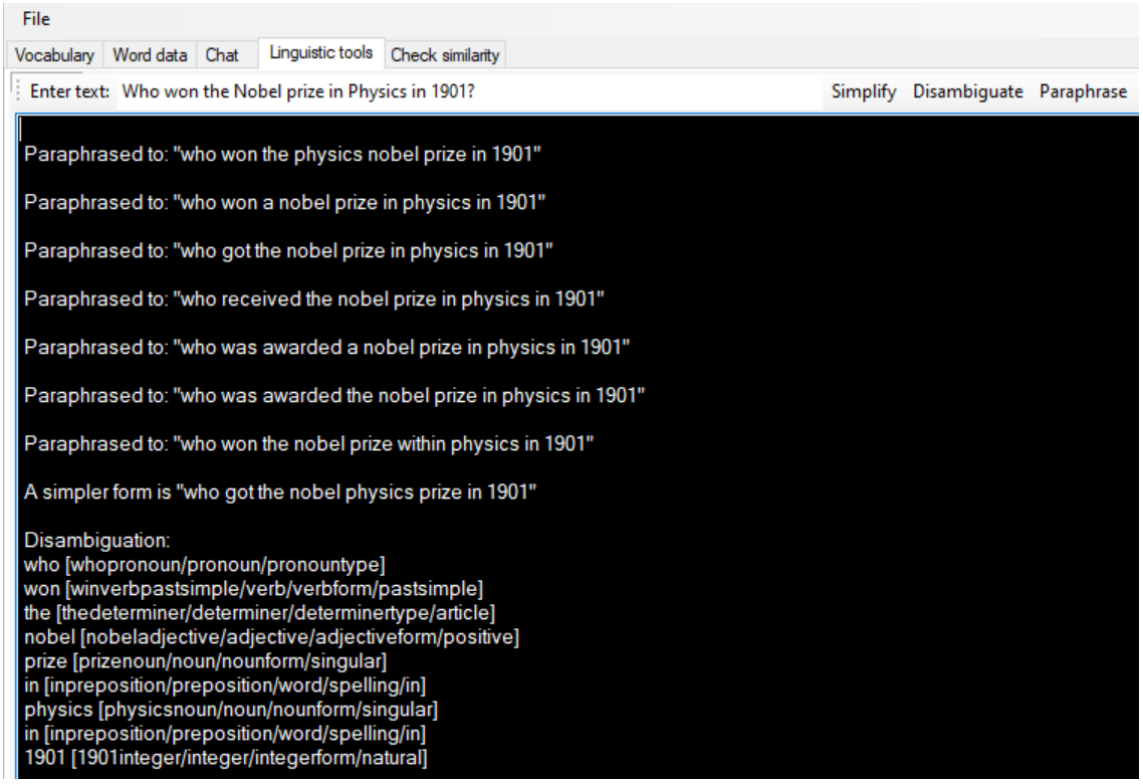


Figure 4.1: Results of applying three of the developed NLP methods on the sentence *"Who won the Nobel prize in Physics in 1901?"*.

Figure 4.2 shows the output of the implemented program when the sentence *"Which scientist received the Chemistry Nobel prize in 1903?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated.

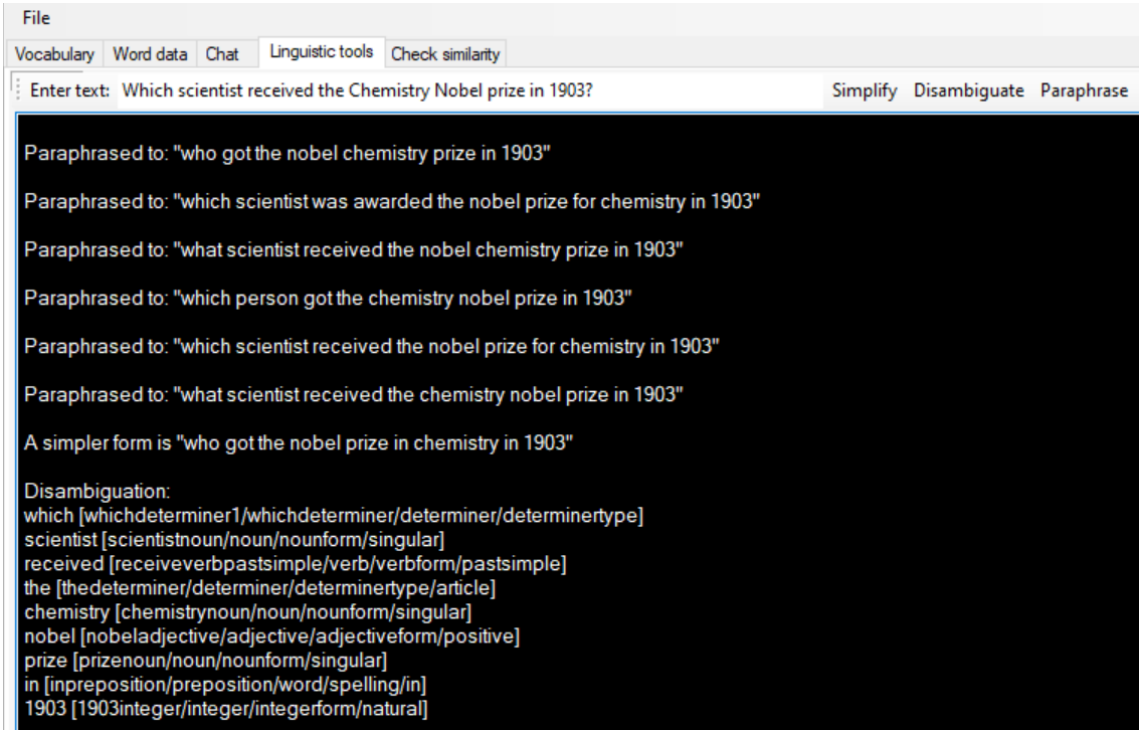


Figure 4.2: Results of applying three of the developed NLP methods on the sentence *"Which scientist received the Chemistry Nobel prize in 1903?"*.

Figure 4.3 shows the output of the implemented program when the sentence *"In what year did Becquerel win the Nobel prize?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated.

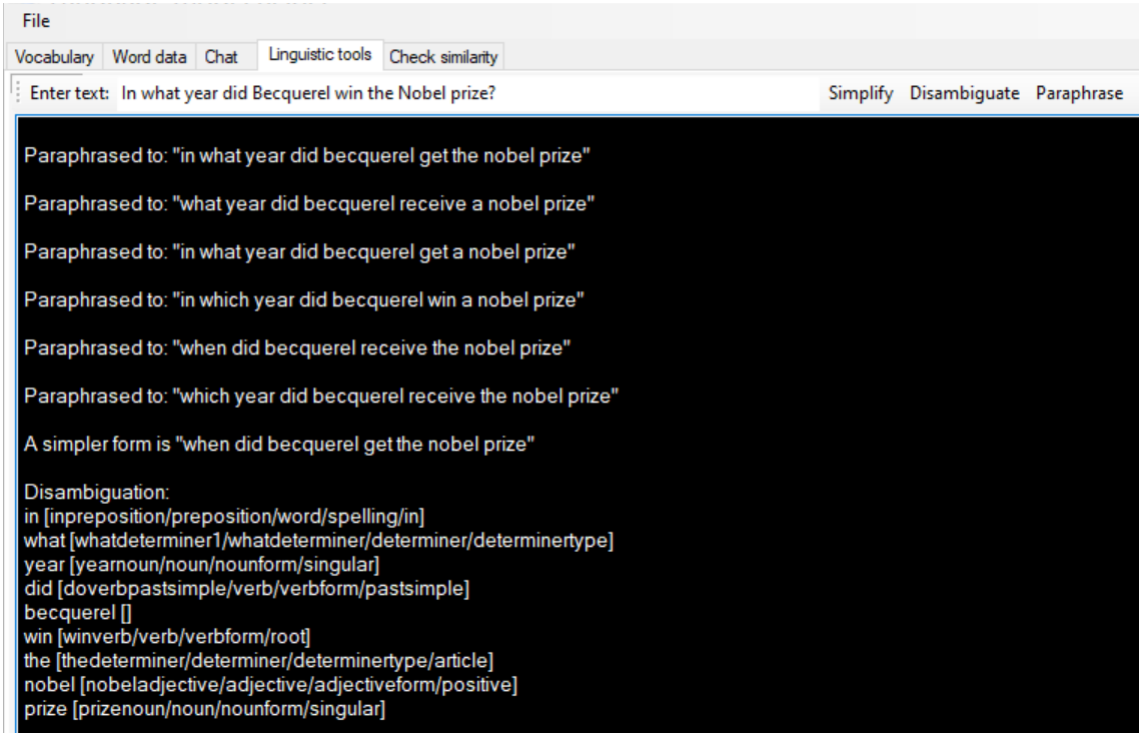


Figure 4.3: Results of applying three of the developed NLP methods on the sentence *"In what year did Becquerel win the Nobel prize?"*.

Figure 4.4 shows the output of the implemented program when the sentence *"Did Becquerel share his prize with anyone?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated.

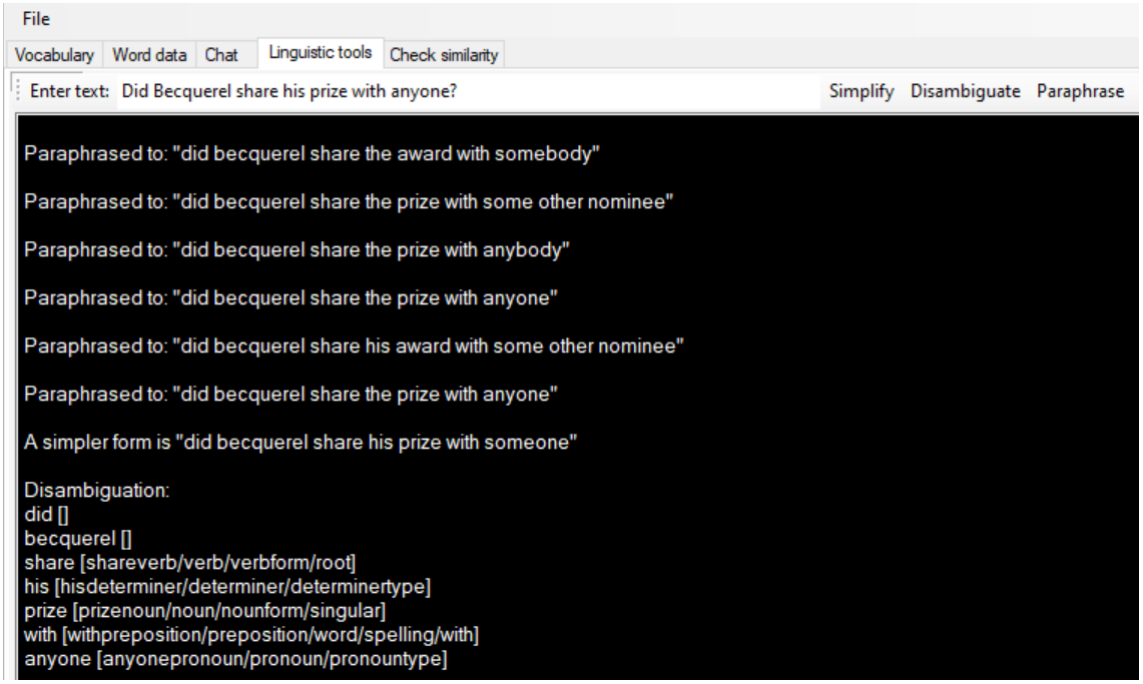


Figure 4.4: Results of applying three of the developed NLP methods on the sentence *"Did Becquerel share his prize with anyone?"*.

Figure 4.5 shows the output of the implemented program when the sentence *"How many women have won the Nobel prize twice?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated.

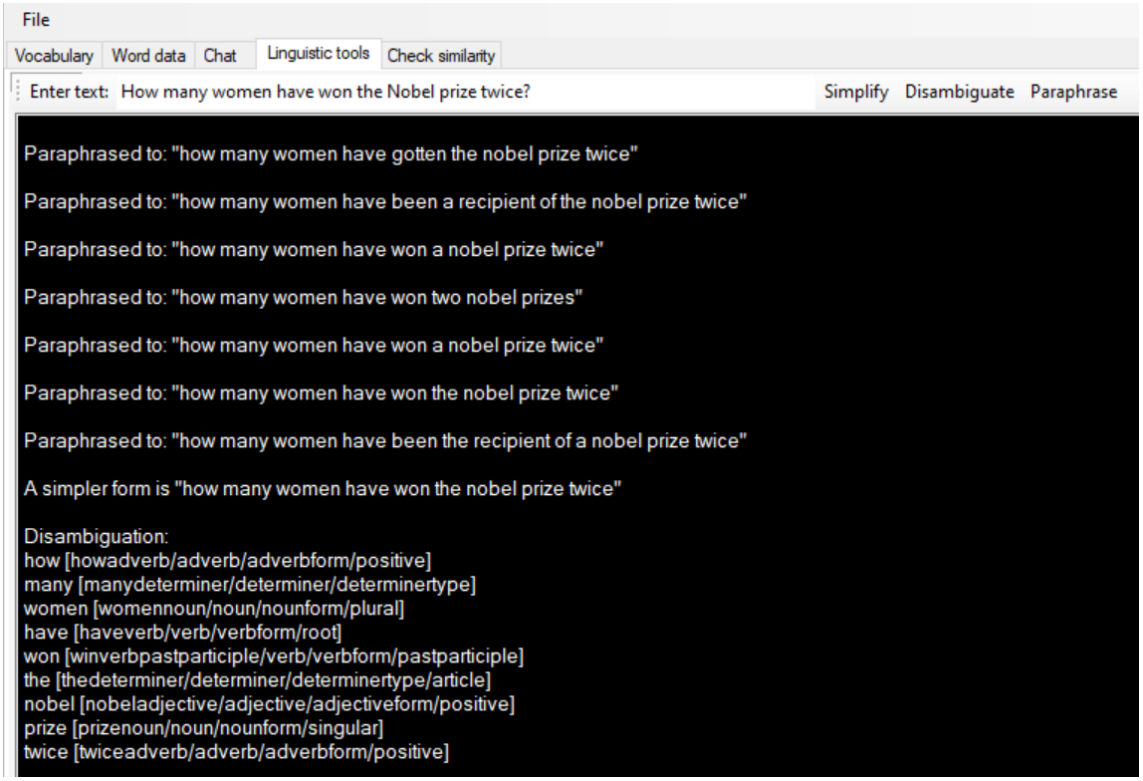


Figure 4.5: Results of applying three of the developed NLP methods on the sentence *"How many women have won the Nobel prize twice?"*.

Figure 4.6 shows the output of the implemented program when the sentence *"Which woman was the first one to receive a Nobel prize?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated.

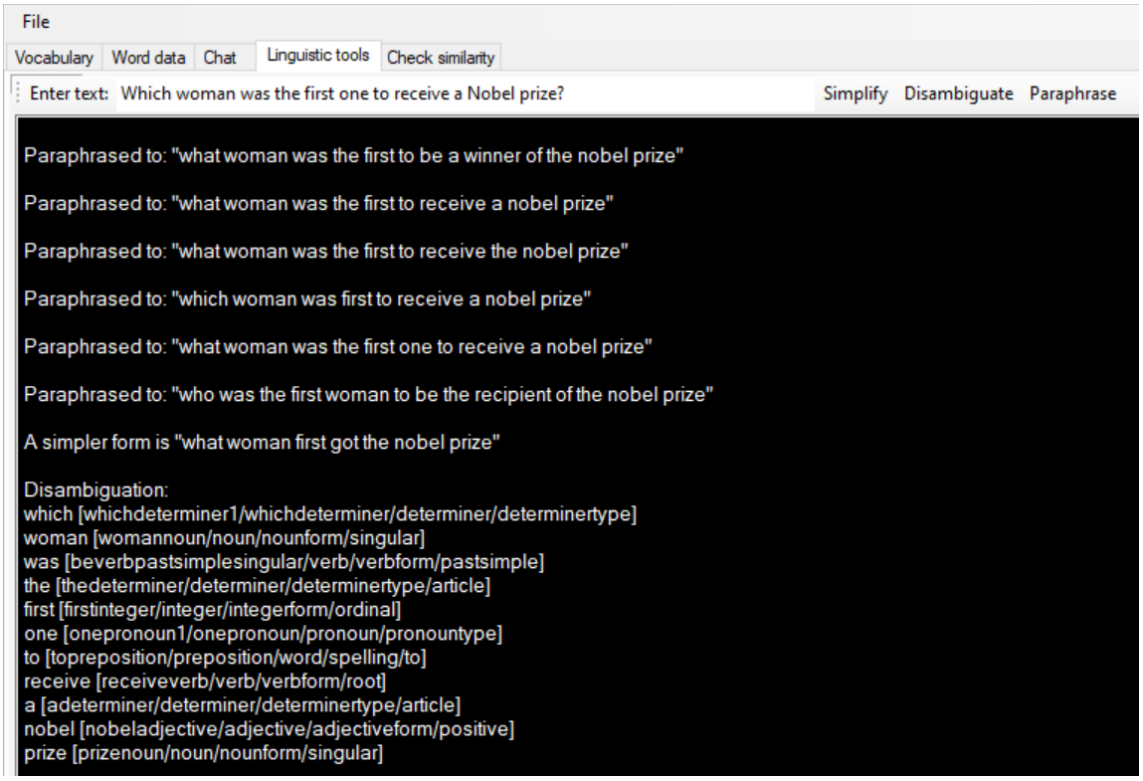


Figure 4.6: Results of applying three of the developed NLP methods on the sentence *"Which woman was the first one to receive a Nobel prize?"*.

4.1.2 Semantic similarity analysis

Figure 4.7 and Figure 4.8 show some example output from the implemented semantic similarity analyser.

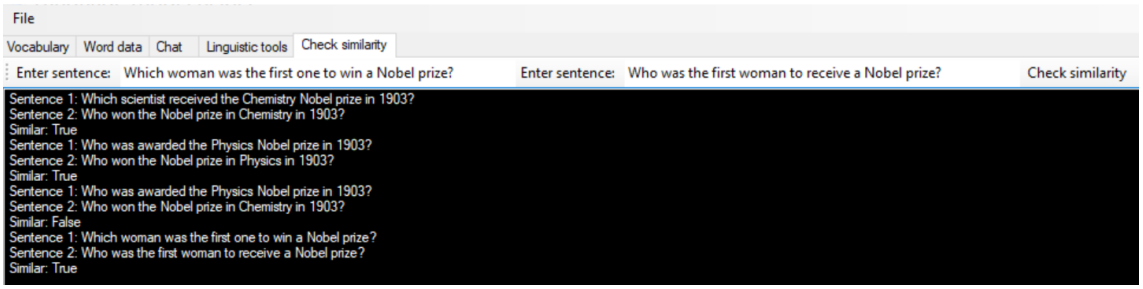


Figure 4.7: Results of applying the developed semantic similarity analyser on four sentence pairs.

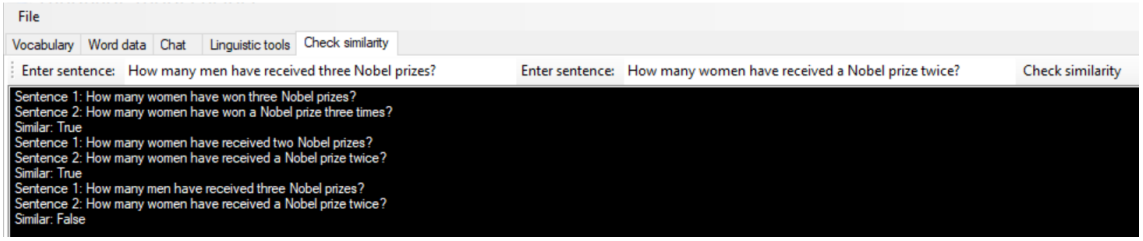


Figure 4.8: Results of applying the developed semantic similarity analyser on three sentence pairs.

4.1.3 Syntactic paraphrasing of generic phrases

Figure 4.9 shows the output of the implemented program when the sentence *"Which woman was the first one that climbed Mount Everest?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated. As seen in the figure, the program could not disambiguate *mount* and *everest* because these words do not exist in the LM. That does not cause any problems in the paraphrasing, however. The textual representation of two of the generic phrases involved in this paraphrasing is seen in Figure 4.10.

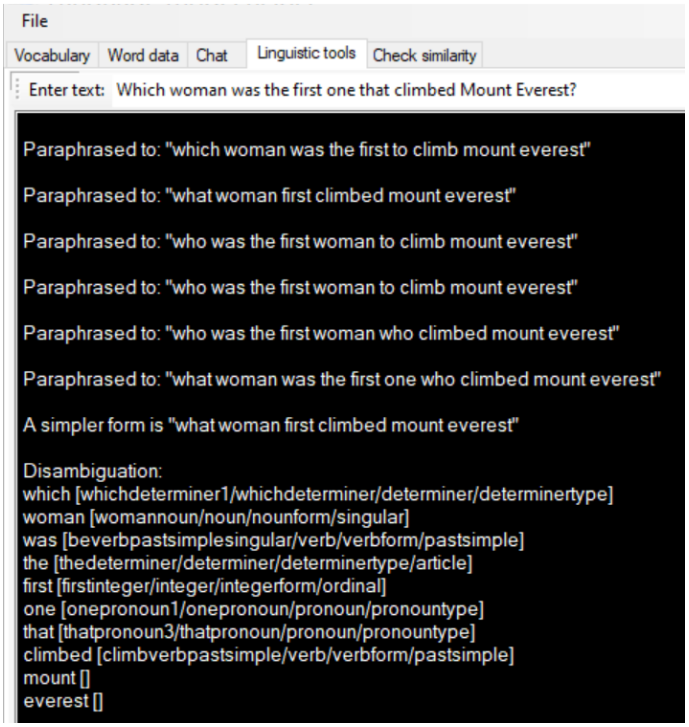


Figure 4.9: Results of applying three of the developed NLP methods on the sentence *"Which woman was the first one that climbed Mount Everest?"*.

Figure 4.11 shows the output of the implemented program when the sentence *"Which duck was the first one to climb Mount Everest?"* is (i) paraphrased, (ii) simplified, and (iii) disambiguated. As seen in the figure, the program could still not disambiguate *mount* and *everest* due to the same reason as in the previous example. Here we also see that the pronoun *who* is not used since *duck* was recognized as a generic

```
% which [*any person noun*] was the [*any ordinal*] one that/who [*any simple past verb*]...
[LMNode: identifier=whichPersonFirst, isInstanceOf=phrase, wordbase =
{what11,personSpecific11,beVerbPastSimpleSingular,theDeterminer,genericOrdinal11,onePronoun1,thatWhoOption,specificGenericVerbPastSimple},
isBiSimilarTo=whoWasTheFirst, isBiSimilarTo=whoWasTheFirstTo, isBiSimilarTo=whichPersonFirstTo2]

% who was the [*any ordinal*] [*any person noun*] to [*any root verb*]...
[LMNode: identifier=whoWasTheFirstTo, isInstanceOf=phrase, wordbase = {whoPronoun,beVerbPastSimpleSingular,theDeterminer,genericOrdinal11,
personSpecific11,toPreposition,specificGenericVerbRoot}, isBiSimilarTo=whichPersonFirstTo2, isBiSimilarTo=whoWasTheFirst]
% isBiSimilarTo=whoWasTheFirst
```

Figure 4.10: Textual representation of two different generic phrases in the LM that are connected via a similarity relation.

noun by the LM and not as a person. The textual representation of two of the generic phrases involved in this paraphrasing is seen in Figure 4.12.

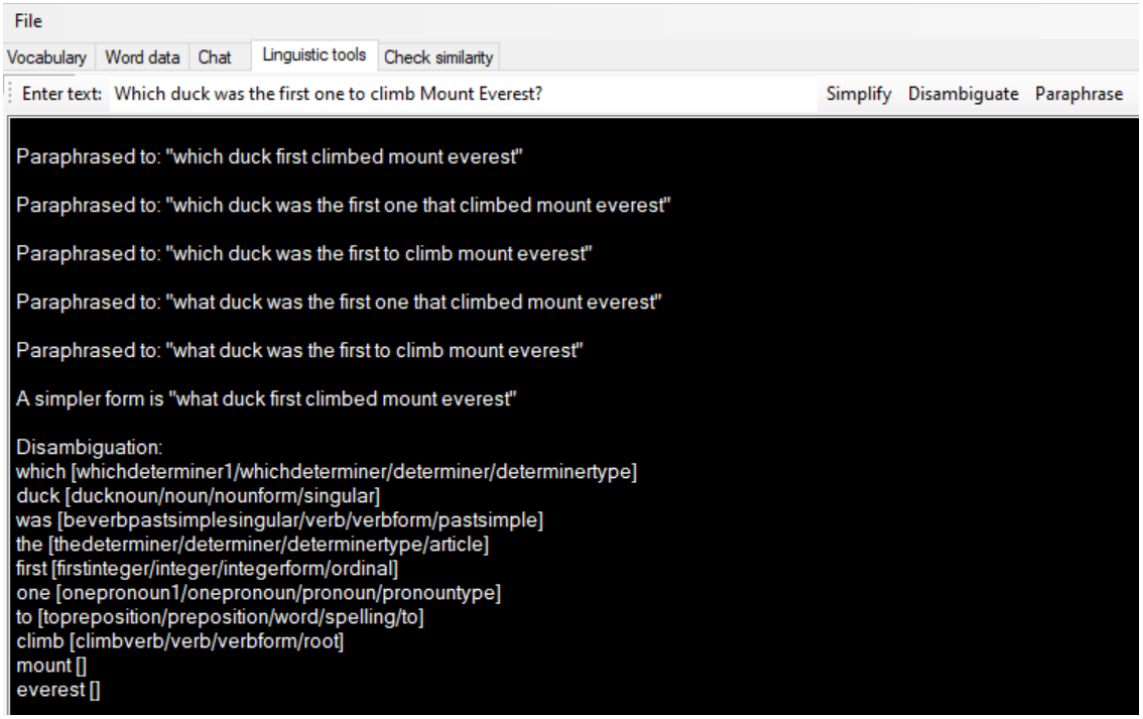


Figure 4.11: Results of applying three of the developed NLP methods on the sentence *"Which duck was the first one to climb Mount Everest?"*.

```
% which [noun] was the first one that [*any simple past verb*]...
[LMNode: identifier=whichNounFirst, isInstanceOf=phrase, wordbase =
{what11,genericNounSingular11,beVerbPastSimpleSingular,theDeterminer,genericOrdinal11,onePronoun1,thatPronoun3,specificGenericVerbPastSimple},
isBiSimilarTo=whoWasTheFirst, isBiSimilarTo=whoWasTheFirstTo, isBiSimilarTo=whichNounFirstTo2]

% which [noun] was the first to [*any root verb*]...
[LMNode: identifier=whichNounFirstTo2, isInstanceOf=phrase, wordbase =
{what11,genericNounSingular11,beVerbPastSimpleSingular,theDeterminer,genericOrdinal11,toPreposition,specificGenericVerbRoot},
isBiSimilarTo=whoWasTheFirstTo, isBiSimilarTo=whoWasTheFirst, isBiSimilarTo=whichNounFirst]
```

Figure 4.12: Textual representation of two phrases containing instances of generic words.

4.2 Discussion

As a starting point for a discussion, some of the results obtained in this thesis have been compared to results produced by two other currently available NLP tools.

One of the most popular online paraphrasing tools is QuillBot[1] [107], which is a ML-based tool that is said to be the best online paraphrasing tool in 2022 and is used by millions of people. When prompted with some of the questions used in this thesis, QuillBot sometimes produces good options of paraphrases, but also quite a large number of incorrect or even nonsensical ones. Some of the QuillBot results are presented below:

Prompt: Which scientist received the Nobel prize in Chemistry in 1903?
Result: In 1903, who scientist was awarded the Nobel Prize in Chemistry?

Prompt: Did Becquerel share his prize with anyone?
Result: Shared his reward with anyone, Becquerel?

Prompt: How many women have won the Nobel prize twice?
Result: How many women have received the Nobel Peace Prize twice?

Prompt: Which duck was the first one to climb Mount Everest?
Result: Which of the following ducks was the first to scale Mount Everest?

Another popular online paraphrasing tool is wordtune[2] which is another ML model-based tool that is trained on enormous datasets of written material. Similarly to QuillBot, wordtune produced a number of good paraphrases for some of the phrases used in this thesis, along with an equally substantial amount of nonsensical options, some of which are seen below:

Prompt: Who won the Nobel prize in Physics in 1901?
Result 1: How much did the Nobel prize in Physics in 1901 go to?
Result 2: Physicist won the 1901 Nobel prize for their work?
Result 3: When was the Nobel prize in Physics awarded in 1901?

Prompt: Which scientist received the Chemistry Nobel prize in 1903?
Result: How was the Chemistry Nobel prize awarded in 1903 to which scientist?

Prompt: In what year did Becquerel win the Nobel prize?
Result: Who won the Nobel prize in Becquerel's year?

Prompt: How many women have won the Nobel prize twice?
Result: Do any women have won the Nobel prize twice?

[1] https://quillbot.com/
[2] https://www.wordtune.com/

The results presented above are only a sub-sample of the paraphrases that were produced by QuillBot and wordtune. As mentioned above, for some phrases, both QuillBot and wordtune did produce good paraphrases additional to the nonsensical results seen above. The problem however, despite such models sometimes achieving impressive results, is their unreliability. In addition, due to the black-box nature of such models, there is no straightforward way to fix the observed errors. In that aspect, the model and methods that were developed in this thesis are superior. For all phrases that the methods that were developed for this thesis were prompted with, high accuracy and precision was achieved. This is stated in the sense that (i) for all cases that were demonstrated in Chapter 4, the paraphrases that were produced are grammatically correct and semantically similar to the input sentence, (ii) the output of the simplifier is considered simpler than the input sentence, (iii) the POS tags that were returned by the disambiguation method are indeed correct, and (iv) semantically similar sentences were detected by the similarity analyser and vice versa. Additionally, the methods implemented in this thesis are fully interpretable, thus enabling complete transparency and easy debugging.

The main disadvantage of the LM that has been developed and applied in this thesis is that a lot of manual work is required to construct the phrases. The phrases and phrase similarities need to be carefully curated not to introduce ungrammaticalities and inappropriate formulations. However, the advantage is that the LM only needs to be built once and for all, and then it can be applied in any application with full transparency. For use in high-stake situations where correctness is prioritised over versatility, this is to be preferred. Furthermore, the process of adding phrases can be further automatised as part of future work.

5
Conclusion and Future Work

5.1 Conclusion

It can be concluded that implementing interpretable methods for (i) paraphrasing, (ii) textual simplification, (iii) semantic similarity analysis, and (iv) POS disambiguation, using an interpretable language model based on a knowledge graph is indeed a viable alternative to black-box neural-based systems in situations where correctness is preferred over versatility. Building a general knowledge graph-based language model has been shown to be a laborious task requiring high linguistic awareness, albeit a rewarding one given the reliability of the resulting system. Conclusively, the work presented in this thesis offers promising prospects for continued research on this topic and on ways to further automatise and generalise the process of constructing an interpretable language model.

5.2 Future work

There is a number of areas into which future work on this topic should be directed. One would be to optimise the program in terms of execution time. This could be done by e.g. introducing parallelisation where appropriate and looking into the choices of data structures throughout the program.

A useful functionality would be to link verb forms for phrases such that it is enough to add a phrase for one verb form and make the program able to transform it to equivalent phrases in other tenses. This was planned to be implemented for this thesis but was not realised due to time constraints.

Another useful functionality would be to introduce a so-called ”skip word” functionality. I.e. being able to construct phrases that consist of nonconsecutive word sequences, such as ”When did xx win a Nobel prize” ≡ ”When was xx the recipient of a Nobel prize” for all proper noun options of xx, but without having to generate *all* possible such phrases.

Finally, an obvious area of improvement would be to add more phrases and phrase relationships to the LM to expand the application domain of the model. Moreover, a relevant focus of future work would be to make the phrase constructions partly automatised. One way of doing this would be to let a user add the spelling of a new phrase whereby the user is prompted about which POS should be intended for each

word in the phrase. Thus, the textual node representation of the phrase could be generated and added to the LM automatically which would speed up the building process.

Bibliography

[1] A. Kornai, Mathematical linguistics. Springer Science & Business Media, 2007.

[2] C. Rao and V. N. Gudivada, Computational analysis and understanding of natural languages: principles, methods and applications. Elsevier, 2018.

[3] Y. Kang, Z. Cai, C.-W. Tan, Q. Huang, and H. Liu, "Natural language processing (nlp) in management research: A literature review," Journal of Management Analytics, vol. 7, no. 2, pp. 139–172, 2020.

[4] A. Chiche and B. Yitagesu, "Part of speech tagging: a systematic review of deep learning and machine learning approaches," Journal of Big Data, vol. 9, no. 1, pp. 1–25, 2022.

[5] G. Terzopoulos and M. Satratzemi, "Voice assistants and smart speakers in everyday life and in education," Informatics in Education, vol. 19, no. 3, pp. 473–490, 2020.

[6] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," Multimedia Tools and Applications, pp. 1–32, 2022.

[7] S. Hiriyannaiah, A. Srinivas, G. K. Shetty, S. G.M., and K. Srinivasa, "Chapter 4 - a computationally intelligent agent for detecting fake news using generative adversarial networks," in Hybrid Computational Intelligence (S. Bhattacharyya, V. Snášel, D. Gupta, and A. Khanna, eds.), Hybrid Computational Intelligence for Pattern Analysis and Understanding, pp. 69–96, Academic Press, 2020.

[8] K. R. Chowdhary, Natural Language Processing, pp. 603–649. New Delhi: Springer India, 2020.

[9] M. Wahde and M. Virgolin, "Conversational agents: Theory and applications," arXiv preprint arXiv:2202.03164, 2022.

[10] C. J. Rameshbhai and J. Paulose, "Opinion mining on newspaper headlines using svm and nlp," International journal of electrical and computer engineering (IJECE), vol. 9, no. 3, pp. 2152–2163, 2019.

[11] T. Cai, A. A. Giannopoulos, S. Yu, T. Kelil, B. Ripley, K. K. Kumamaru, F. J. Rybicki, and D. Mitsouras, "Natural language processing technologies

in radiology research and clinical applications," Radiographics, vol. 36, no. 1, p. 176, 2016.

[12] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," IEEE transactions on neural networks and learning systems, vol. 32, no. 2, pp. 604–624, 2020.

[13] L. Deng and Y. Liu, Deep learning in natural language processing. Springer, 2018.

[14] X. Sun, D. Yang, X. Li, T. Zhang, Y. Meng, Q. Han, G. Wang, E. Hovy, and J. Li, "Interpreting deep learning models in natural language processing: A review," arXiv preprint arXiv:2110.10470, 2021.

[15] P. Karatza, K. Dalakleidi, M. Athanasiou, and K. S. Nikita, "Interpretability methods of machine learning algorithms with applications in breast cancer diagnosis," in 2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), pp. 2310–2313, IEEE, 2021.

[16] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (xai): Toward medical xai," IEEE transactions on neural networks and learning systems, vol. 32, no. 11, pp. 4793–4813, 2020.

[17] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," Nature Machine Intelligence, vol. 1, no. 5, pp. 206–215, 2019.

[18] C. Molnar, Interpretable machine learning. Lulu. com, 2020.

[19] M. Wahde and M. Virgolin, "The five is: Key principles for interpretable and safe conversational ai," in 2021 The 4th International Conference on Computational Intelligence and Intelligent Systems, pp. 50–54, 2021.

[20] M. Wahde, "A dialogue manager for task-oriented agents based on dialogue building-blocks and generic cognitive processing," in 2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA), pp. 1–8, IEEE, 2019.

[21] M. Wahde and M. Virgolin, "Daisy: An implementation of five core principles for transparent and accountable conversational ai," International Journal of Human–Computer Interaction, pp. 1–18, 2022.

[22] M. Wahde, M. Della Vedova, M. Virgolin, and K. Carlström, "An interpretable and transparent language model for exact paraphrasing," Manuscript in preparation.

[23] J. Lyons, Natural Language and Universal Grammar: Volume 1: Essays in Linguistic Theory, vol. 1. Cambridge University Press, 1991.

[24] J. Eisenstein, Introduction to natural language processing. MIT press, 2019.

[25] P. Ranjan and H. Basu, "Part of speech tagging and local word grouping techniques for natural language parsing in hindi," in Proceedings of the 1st International Conference on Natural Language Processing (ICON 2003), Citeseer, 2003.

[26] E. Sadredini, D. Guo, C. Bo, R. Rahimi, K. Skadron, and H. Wang, "A scalable solution for rule-based part-of-speech tagging on novel hardware accelerators," in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 665–674, 2018.

[27] Y. Ravin and C. Leacock, "Polysemy: an overview," Polysemy: Theoretical and computational approaches, pp. 1–29, 2000.

[28] L. Padró, "A hybrid environment for syntax-semantic tagging," arXiv preprint cmp-lg/9802002, 1998.

[29] R. Rosenfeld, "Adaptive statistical language modeling; a maximum entropy approach," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1994.

[30] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," Journal of the American Medical Informatics Association, vol. 18, no. 5, pp. 544–551, 2011.

[31] K. S. Jones, "Natural language processing: a historical review," Current issues in computational linguistics: in honour of Don Walker, pp. 3–16, 1994.

[32] R. W. Brown, "Linguistic determinism and the part of speech.," The Journal of Abnormal and Social Psychology, vol. 55, no. 1, p. 1, 1957.

[33] J. Rijkhoff, "Word classes," Language and linguistics compass, vol. 1, no. 6, pp. 709–726, 2007.

[34] S. Warjri, P. Pakray, S. A. Lyngdoh, and A. K. Maji, "Part-of-speech (pos) tagging using conditional random field (crf) model for khasi corpora," International Journal of Speech Technology, vol. 24, no. 4, pp. 853–864, 2021.

[35] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun, "A practical part-of-speech tagger," in Third conference on applied natural language processing, pp. 133–140, 1992.

[36] A. Voutilainen, Part-of-speech tagging, vol. 219. The Oxford handbook of computational linguistics, 2003.

[37] R. Mitkov, The Oxford handbook of computational linguistics. Oxford University Press, 2022.

[38] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," arXiv preprint arXiv:1508.01991, 2015.

[39] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in Proceedings of the 27th international conference on computational linguistics, pp. 1638–1649, 2018.

[40] A. Voutilainen, "Engcg tagger, version 2," Sprog og Multimedier. Aalborg Universitetsforlag, Aalborg, 1997.

[41] E. Brill, "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging," Computational linguistics, vol. 21, no. 4, pp. 543–565, 1995.

[42] S. S. Al-Thanyyan and A. M. Azmi, "Automated text simplification: a survey," ACM Computing Surveys (CSUR), vol. 54, no. 2, pp. 1–36, 2021.

[43] G. Paetzold and L. Specia, "Unsupervised lexical simplification for non-native speakers," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, 2016.

[44] R. Evans, C. Orasan, and I. Dornescu, "An evaluation of syntactic simplification rules for people with autism," Association for Computational Linguistics, 2014.

[45] L. Rello, R. Baeza-Yates, L. Dempere-Marco, and H. Saggion, "Frequent words improve readability and short words improve understandability for people with dyslexia," in IFIP Conference on Human-Computer Interaction, pp. 203–219, Springer, 2013.

[46] J. Carroll, G. Minnen, Y. Canning, S. Devlin, and J. Tait, "Practical simplification of english newspaper text to assist aphasic readers," in Proceedings of the AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology, pp. 7–10, Citeseer, 1998.

[47] G. A. Miller, "Wordnet: a lexical database for english," Communications of the ACM, vol. 38, no. 11, pp. 39–41, 1995.

[48] P. T. Quinlan, The Oxford psycholinguistic database. Oxford University Press, 1992.

[49] M. Shardlow, "Out in the open: Finding and categorising errors in the lexical simplification pipeline," in Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), pp. 1583–1590, 2014.

[50] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," tech. rep., Naval Technical Training Command Millington TN Research Branch, 1975.

[51] O. Biran, S. Brody, and N. Elhadad, "Putting it simply: a context-aware approach to lexical simplification," in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp. 496–501, 2011.

[52] S. K. Jauhar and L. Specia, "Uow-shef: Simplex–lexical simplicity ranking based on contextual and psycholinguistic features," in * SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012), pp. 477–481, 2012.

[53] A. Siddharthan, "Syntactic simplification and text cohesion," Research on Language and Computation, vol. 4, no. 1, pp. 77–109, 2006.

[54] S. Štajner and G. Glavaš, "Leveraging event-based semantics for automated text simplification," Expert systems with applications, vol. 82, pp. 383–395, 2017.

[55] S. Wubben, A. Van Den Bosch, and E. Krahmer, "Sentence simplification by monolingual machine translation," in Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1015–1024, 2012.

[56] X. Zhang and M. Lapata, "Sentence simplification with deep reinforcement learning," arXiv preprint arXiv:1703.10931, 2017.

[57] Z. Zhu, D. Bernhard, and I. Gurevych, "A monolingual tree-based translation model for sentence simplification," in Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pp. 1353–1361, 2010.

[58] G. Glavaš and S. Štajner, "Simplifying lexical simplification: Do we need simplified corpora?," in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pp. 63–68, 2015.

[59] L. Specia, "Translating from complex to simplified sentences," in International Conference on Computational Processing of the Portuguese Language, pp. 30–39, Springer, 2010.

[60] K. Woodsend and M. Lapata, "Learning to simplify sentences with quasi-synchronous grammar and integer programming," in Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pp. 409–420, 2011.

[61] N. Ranjan, K. Mundada, K. Phaltane, and S. Ahmad, "A survey on techniques in nlp," International Journal of Computer Applications, vol. 134, no. 8, pp. 6–9, 2016.

[62] A. Pawar and V. Mago, "Challenging the boundaries of unsupervised learning for semantic similarity," IEEE Access, vol. 7, pp. 16291–16308, 2019.

[63] D. Chandrasekaran and V. Mago, "Evolution of semantic similarity—a survey," ACM Computing Surveys (CSUR), vol. 54, no. 2, pp. 1–37, 2021.

[64] A. Freitas, J. G. Oliveira, S. O'Riain, E. Curry, and J. C. Pereira da Silva, "Querying linked data using semantic relatedness: a vocabulary independent approach," in International Conference on Application of Natural Language to Information Systems, pp. 40–51, Springer, 2011.

[65] V. Abhishek and K. Hosanagar, "Keyword generation for search engine advertising using semantic similarity between terms," in Proceedings of the ninth international conference on Electronic commerce, pp. 89–94, 2007.

[66] C. Pesquita, D. Faria, A. O. Falcao, P. Lord, and F. M. Couto, "Semantic similarity in biomedical ontologies," PLoS computational biology, vol. 5, no. 7, p. e1000443, 2009.

[67] P. W. Lord, R. D. Stevens, A. Brass, and C. A. Goble, "Investigating semantic similarity measures across the gene ontology: the relationship between sequence and annotation," Bioinformatics, vol. 19, no. 10, pp. 1275–1283, 2003.

[68] T. Pedersen, S. V. Pakhomov, S. Patwardhan, and C. G. Chute, "Measures of semantic similarity and relatedness in the biomedical domain," Journal of biomedical informatics, vol. 40, no. 3, pp. 288–299, 2007.

[69] G. Varelas, E. Voutsakis, P. Raftopoulou, E. G. Petrakis, and E. E. Milios, "Semantic similarity methods in wordnet and their application to information retrieval on the web," in Proceedings of the 7th annual ACM international workshop on Web information and data management, pp. 10–16, 2005.

[70] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," Journal of artificial intelligence research, vol. 22, pp. 457–479, 2004.

[71] Y. Ko, J. Park, and J. Seo, "Improving text categorization using the importance of sentences," Information processing & management, vol. 40, no. 1, pp. 65–79, 2004.

[72] M. A. Hadj Taieb, T. Zesch, and M. Ben Aouicha, "A survey of semantic relatedness evaluation datasets and procedures," Artificial Intelligence Review, vol. 53, no. 6, pp. 4407–4448, 2020.

[73] G. Zhu and C. A. Iglesias, "Computing semantic similarity of concepts in knowledge graphs," IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 1, pp. 72–85, 2016.

[74] M. T. Pilehvar and R. Navigli, "From senses to texts: An all-in-one graph-based approach for measuring semantic similarity," Artificial Intelligence, vol. 228, pp. 95–128, 2015.

[75] R. Rada, H. Mili, E. Bicknell, and M. Blettner, "Development and application of a metric on semantic nets," IEEE transactions on systems, man, and cybernetics, vol. 19, no. 1, pp. 17–30, 1989.

[76] J. J. Lastra-Díaz, J. Goikoetxea, M. A. H. Taieb, A. García-Serrano, M. B. Aouicha, and E. Agirre, "A reproducible survey on word embeddings and ontology-based methods for word similarity: linear combinations outperform the state of the art," Engineering Applications of Artificial Intelligence, vol. 85, pp. 645–665, 2019.

[77] D. Sánchez and M. Batet, "A semantic similarity method based on information content exploiting multiple ontologies," Expert Systems with Applications, vol. 40, no. 4, pp. 1393–1399, 2013.

[78] J. Gorman and J. R. Curran, "Scaling distributional similarity to large corpora," in Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, pp. 361–368, 2006.

[79] A. Siddique, S. Oymak, and V. Hristidis, "Unsupervised paraphrasing via deep reinforcement learning," in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1800–1809, 2020.

[80] I. Androutsopoulos and P. Malakasiotis, "A survey of paraphrasing and textual entailment methods," Journal of Artificial Intelligence Research, vol. 38, pp. 135–187, 2010.

[81] E. Kissner, Summarizing, paraphrasing, and retelling: Skills for better reading, writing, and test taking. Heinemann Educational Books, 2006.

[82] K. McKeown, "Paraphrasing questions using given and new information," American Journal of Computational Linguistics, vol. 9, no. 1, pp. 1–10, 1983.

[83] K. Knight and D. Marcu, "Statistics-based summarization-step one: Sentence compression," AAAI/IAAI, vol. 2000, pp. 703–710, 2000.

[84] A. Gupta, A. Agarwal, P. Singh, and P. Rai, "A deep generative framework for paraphrase generation," in Proceedings of the aaai conference on artificial intelligence, vol. 32, 2018.

[85] P. Shah, D. Hakkani-Tür, G. Tür, A. Rastogi, A. Bapna, N. Nayak, and L. Heck, "Building a conversational agent overnight with dialogue self-play," arXiv preprint arXiv:1801.04871, 2018.

[86] A. Fader, L. Zettlemoyer, and O. Etzioni, "Open question answering over curated and extracted knowledge bases," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 1156–1165, 2014.

[87] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou, "Answering questions with complex semantic constraints on open knowledge bases," in Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 1301–1310, 2015.

[88] R. Barzilay and L. Lee, "Learning to paraphrase: An unsupervised approach using multiple-sequence alignment," arXiv preprint cs/0304006, 2003.

[89] C. Joao, D. Gaël, and B. Pavel, "New functions for unsupervised asymmetrical paraphrase detection," Journal of Software, vol. 2, no. 4, pp. 12–23, 2007.

[90] N. Madnani and B. J. Dorr, "Generating phrasal and sentential paraphrases: A survey of data-driven methods," Computational Linguistics, vol. 36, no. 3, pp. 341–387, 2010.

[91] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in International conference on machine learning, pp. 1885–1894, PMLR, 2017.

[92] F. Pasquale, The black box society: The secret algorithms that control money and information. Harvard University Press, 2015.

[93] J. Mökander, P. Juneja, D. S. Watson, and L. Floridi, "The us algorithmic accountability act of 2022 vs. the eu artificial intelligence act: what can they learn from each other?," Minds and Machines, pp. 1–8, 2022.

[94] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a "right to explanation"," AI magazine, vol. 38, no. 3, pp. 50–57, 2017.

[95] B. Mittelstadt, C. Russell, and S. Wachter, "Explaining explanations in ai," in Proceedings of the conference on fairness, accountability, and transparency, pp. 279–288, 2019.

[96] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?," in Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, pp. 610–623, 2021.

[97] R. Daws, "Medical chatbot using openai's gpt-3 told a fake patient to kill themselves," AI News, 2020.

[98] K. Simov, A. Popov, and P. Osenova, "The role of the wordnet relations in the knowledge-based word sense disambiguation task," in Proceedings of the 8th Global WordNet Conference (GWC), pp. 396–403, 2016.

[99] H. Sato, "A data model, knowledge base, and natural language processing for sharing a large statistical database," in International Conference on Scientific and Statistical Database Management, pp. 207–225, Springer, 1988.

[100] K. Simov, A. Popov, and P. Osenova, "Improving word sense disambiguation with linguistic knowledge from a sense annotated treebank," in Proceedings of the International Conference Recent Advances in Natural Language Processing, pp. 596–603, 2015.

[101] A. Montoyo, A. Suárez, G. Rigau, and M. Palomar, "Combining knowledge- and corpus-based word-sense-disambiguation methods," Journal of Artificial Intelligence Research, vol. 23, pp. 299–330, 2005.

[102] Y. Wilks and M. Stevenson, "Sense tagging: Semantic tagging with a lexicon," arXiv preprint cmp-lg/9705016, 1997.

[103] H. T. Ng and H. B. Lee, "Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach," arXiv preprint cmp-lg/9606032, 1996.

[104] S. Y. Jung, Y. C. Park, K.-S. Choi, and Y. Kim, "Markov random field based english part-of-speech tagging system," in COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics, 1996.

[105] C.-H. Björnsson, Läsbarhet: hur skall man som författare nå fram till läsarna? Bokförlaget Liber, 1968.

[106] R. Senter and E. A. Smith, "Automated readability index," tech. rep., Cincinnati Univ OH, 1967.

[107] T. N. Fitria, "Quillbot as an online tool: Students' alternative in paraphrasing and rewriting of english writing," Englisia: Journal of Language, Education, and Humanities, vol. 9, no. 1, pp. 183–196, 2021.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY