Introducing the component coreference resolution task for requirement specification

Master Thesis

in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

by

Jan Scheffczyk

supervised by

Prof. Dr. Michael Meier

and

Dr. Hanna Claudia Geppert

Department of Computer Science
Faculty of Mathematics and Natural Sciences
Rheinischen Friedrich-Wilhelms-Universität Bonn
Germany
June 20, 2023

Abstract

Consistent terminology is important for successful communication between two parties. If two parties do not agree on a common term for a component in their project, the resulting inconsistency leads to confusion and misunderstandings. In Natural Language Processing (NLP) this type of terminological inconsistency is called coreference since there are multiple terms that refer to a common entity. However, a pronoun that refers to its noun is also considered a coreferences. Our goal is to find coreferences among project components only. Thus, in this thesis, we introduce the novel term Component Coreference Resolution (CCR) to differentiate our task from other coreference resolution tasks. In addition, we propose a pipeline to automatically find component coreferences to aid authors in maintaining a consistent terminology. This is especially relevant if authors from different backgrounds work on shared documents as is the case in requirements engineering. To evaluate our pipeline, we manually constructed a test dataset based on public project requirement collections. This allowed us to control the number of coreferences in the test set precisely.

Our proposed pipeline is build on word vectors. These vectors attempt to capture the meaning of a word as a vector in a high dimensional vector space. For the CCR pipeline we explore three different word embedding models. To adopt publicly available weights for these models to the domain of the test set, we use unsupervised fine-training. Many terms are not single words but rather a phrase. To combine the individual vectors of the multiple words into a single phrase vector, we then systematically explore different phrase-, layer- and subword-pooling strategies. These pooling operations create a phrase vector with uniform length for any input phrase regardless of its length. Finally, to determine if two vectors are coreferent, we compare the resulting phrase vectors. For the comparison, we test cosine-similarity, Jaccard-Index and DynaMax.

During evaluation, we choose to place high emphasize on precision and its correlation to the underlying similarity score. This ensures that our results can be ranked by their similarity. We found that the Jaccard-Index significantly outperforms the generally recommend cosine similarity. Further, we contribute additional evidence that domain fine-trained models can outperform generalized supervised models.

Contents

Acronyms

CCR Component Coreference Resolution. I, 3–5, 12–15, 17, 27, 28, 32–34, 41, 42, 57, 64, 67, 68, 73

CDCR Cross Document Coreference Resolution. 12, 13

CNN Convolutional Neural Network. 22, 26, 27

CR Coreference Resolution. 11–13

EMD Earth Mover Distance. 70

GRU Gated Recurrent Units. 26

KB Knowledge Base. 16, 18, 68, 69

LSTM Long Short Term Memory. 26

MLM Masked Language Modeling. 24, 45, 47

MLP Multi Layer Perceptron. 25, 26

NLP Natural Language Processing. I, 10, 26, 27, 30, 32, 43, 50, 59, 73

NLP4RE Natural Language Processing for requirements Engineering. 13

OOV out of Vocabulary. 27

POS Part of Speech. 43, 44

RE Requirements Engineering. 5, 12–14

RNN Recurrent Neural Network. 26

SOTA State of the Art. 85, 86

STS Semantic Textual Similarity. 27, 28, 35, 66, 67

SVD Singular Value Decomposition. 20, 21, 30

VSM Vector Space Model. II, 11, 18, 41, 43, 68, 69

WSD Word Sense Disambiguation. 10, 16–18, 85–87

1 Introduction

Many statements have more than one plausible meaning. This ambiguity is a feature of natural languages. In casual conversation, ambiguity is often inconsequential or can be resolved by facial expressions, tone of voice, or simply because we are familiar with the other person. In writing, it is more difficult to avoid ambiguity, especially, when writing to a general audience. Often we are convinced that we have written a concise and easily understandable text, and yet we later find out that the reader did not understand what we meant. Or worse yet one might misunderstand us but is unaware of it and proceeds to carry out the wrong instructions. While ambiguity is an essential aspect of natural language, outside of witty word puns, it is mostly resolved subconsciously.

In this thesis, we will focus on a specific source of ambiguity: synonyms. Traditionally, synonyms are not considered as a source of linguistic ambiguity. A synonym should, by definition, not significantly alter the meaning of a sentence when exchanged. By extension, replacing words with words that convey the same meaning (synonyms) will not change whether or not a sentence is ambiguous. We will discuss multiple definitions of synonyms in subsection 2.4. For now it is sufficient to note that we can deliberately make similar concepts more distinguishable by using a distinct term for each. If done consistently this aids the reader to differentiate between two similar terms. Frequently used terms then often get shortened and slowly establish as part of a new terminology for the field. Consider the following example adapted from [Rob+16] which describes the requirements two different stakeholders have for a web gallery.

> "As a visitor, I am able to view the gallery, so that I can see interesting photos about the event region."
> "As an administrator, I am able to edit existing media elements of a media collection, so that I can update the content."

The two user requirements are written from two different perspectives and as such, the language used by the visitor is more casual whereas the administrator uses a more technical language. As such, *photos* and *media elements* probably refer to the same component. We call any two phrases that refer to the same concept coreferent. As such, *photos* and *media elements* are probably *coreferent*. The *media collection*, however, might refer to the *gallery* from the first user story, but it could also refer to an internal data structure. As such, *media collection* and *gallery* are potentially coreferent. These instances should still be detected and flagged for a manual review process. As we can see, it can easily happen that two separate authors work on a requirement set and use distinct vocabulary despite describing the same thing/component/concept.

The goal of this thesis is to compare different approaches that are able to find potential coreferences in a set of project requirements in an automated manner and present them to the author for review. The author can then resolve actual coreferences leaving only conscious distinctions. In more technical terms, we are trying to resolve coreferences between the terms used to describe project components. This specific task has received little attention in the research community so far and as such, no term for it exists yet. We propose the novel term **Component Coreference Resolution (CCR)**.

To tackle this challenging task, we base our work on established methods for text similarity. The challenge is to find a similarity measure that handles the non trivial cases where two phrase have the same meaning but different spelling. Traditional text similarity methods were limited to syntactic features like comparing how many characters need to be changed to transform one input to the other. Recent advances in machine learning created a new category of similarity measures that attempt to also consider the semantic meaning of a phrase during comparison. We aim to provide a broad but coarse literature review of machine learning methods for text similarity to identify works we can utilize in the CCR task. We propose a pipeline based on the most promising methods to address the CCR task. Unlike previous methods, [Wan+20], our pipeline solves the CCR task in a end-to-end manner requiring no additional inputs or annotations to the input. Requirements engineering is a field that requires frequent introduction of new terms to name components unique to the project. Therefore, we will use requirements engineering as a case study for the CCR task and assume the inputs to our system to be requirements for which we want to identify coreferences.

In order to evaluate our pipeline, we need a set of requirements where we know all coreferences. We can then compare these against the predictions of the the pipeline. While there are previous publications addressing coreferences in requirements engineering, neither published the dataset that were used. Thus, in order to evaluate the pipeline, we construct our own dataset based on publicly available requirement specifications. Thus, we not only define the CCR task but also provide a public test dataset along, with benchmarks of our pipeline for it.

Assuming we can resolve all coreferences, a coreferent free requirement set has multiple advantages. First and foremost, the reader can be sure that if two different terms are used that this is intentional and signals a necessary distinction between similar terms. On the other side, the reader can be sure to find all requirements relevant to a project component simply by searching for it. As such the ambiguity of the requirement set is reduced. Merging all coreferent terms leaves only essential terms which makes building a glossary easier.

The remaining thesis is structured as follows: In section 2, we will cover linguistic concepts behind synonyms and coreference and how we can distinguish between the two. We will then discuss the grammatical properties of coreference phrases. The core of our CCR pipeline is based on semantic text similarity. In section 3, we will summarize recent advances in text similarity measures. These advances have been made possible by complex neural networks that are able to map the meaning of a word to a vector. Our pipeline is able to differentiate coreferences from unrelated texts by comparing the vector representations created by these neural networks. Before we can present the pipeline in section 5, we briefly familiarize the reader with the constructed dataset in section 4. We discuss different strategies to adopt the selected neural networks to best address our CCR task. we then come up with a set of configurations to test on the dataset. In section 6, we show that the proposed pipeline achieves results that are significantly better than traditional syntactic similarity methods. We also investigate each configuration parameter separately. This reveals that well established parameter choices like the use of cosine similarity to compare vectors can be replaced by alternatives that are superior for our application. Finally, we offer potential improvements that could be added to the proposed pipeline in section 7 and conclude in section 8.

2 Linguistic Background

2.1 Requirements

The input to our CCR task will be a list of project requirements. In RE a requirement specification document contains all information a developer needs to implement the product. Every function, interaction and property of a project is recorded in a separate sentence. Each sentence should only contain a single functionality. These sentences are referred to as requirements. The core of the specification document is a list of requirements. The following is an example of three individual requirements from the Clarus project[Fan+18]:

> (1) The Clarus system shall implement quality checking processes as soon as data become available.
> (2) The CAS shall enable administrators to manage quality checking rules.
> (3) The Clarus system shall record the methods applied when deriving quality checking information.

While a typical requirement specification document also provides additional texts such as an introduction or a section for the scope of the project, these parts are intended to give a vague idea of the project. The set of individual requirements by itself should specify a complete, correct, feasible, prioritized, and unambiguous product. In this thesis, we will be using a set of requirements as input and identify if a product component has been referred to multiple times using different phrases, as explained in subsection 2.6.

2.2 Word form and meaning

We will adopt the terminology as introduced in [Mil+90]. The term "word" is used for both the series of characters (utterance) as well as for the concept behind the string of characters. To resolve this ambiguity we will refer to the former as *word form* and latter as either *word meaning* or *word sense*. We refer to a series of words as a phrase. A phrase can be a single word form but might include multiple word forms into a unit. Phrase form and phrase sense are defined in analogy to word form and word sense. For most word forms the exact meaning depends on the context in which it is used. Conceptually the context encompasses all the knowledge that is associated with that occurrence of the word. For example:

Grid points must be spaced at most 500m apart.

From this immediate neighborhood, it is impossible to properly determine the exact meaning of *grid*. However, if we also knew that this is a requirement for a power grid in *Example City* we would not only know that a power grid is meant but could actually narrow down to a specific power grid. A word form is monosemous if its meaning in any context only differs slightly and can be grouped into one abstract definition. To sail, for example, always refers to some form of gliding/traveling through a liquid or gas. The description, here *"some form of gliding/traveling through a liquid or gas"*, of the word sense, is called a gloss. Other word forms such as "*glass*" have multiple meanings. One word sense for the material we use in windows and the other for the glass that we drink from. These types of word forms are referred to as polysemous. Linguists [Gil96] further differentiate between multiple meanings that can be traced back to a common concept (polysemous) and those with distinct origins (homonyms e.g. bank). However, since [AJ19] showed that University English majors are unable [1] to differentiate between these two concepts we consider their distinction purely academic. We will therefore refer to any word form with more than one meaning as polysemous. How many senses a particular word has, whether glass actually has two senses or if we can group all of its uses into one sense is up for debate. For a more detailed discussion we refer the reader to [EP07]. Further many word senses have multiple word forms that can express it e.g. to execute and to carry out. Thus the mapping from word form to word sense is a many to many map. Note that rows in Figure 1 express a group of synonym word forms. We will further discuss synonyms in subsection 2.4.

2.3 Lemmatization

We call the lexical form of a word form its lemma. Sometimes lemmas can also be called the lexicalization of a word form. For most humans, the act of lemmatization comes naturally. We know to look up the verb *went* under *to go* in a dictionary. Rather than listing all inflections as separate entries, most dictionaries record just the lexical form. This helps to keep the size of dictionaries to a somewhat reasonable level. While the physical size of dictionaries is not a big concern nowadays most efforts to systematically record relations between words still record these connections only between lemmas. If we want to take advantage of these resources we need to be able to lemmatize any input, for example:

[1] with an accuracy 46% on a binary task

F_1 and F_2 are synonyms; F_2 is polysemous

Word Meanings	Word Forms				
	F_1	F_2	F_3	. . .	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
⋮				. . .	
M_m					$E_{m,n}$

Figure 1: Many-to many mapping between word form and word meaning. From [Mil+90]

the boy's cars are differently colored
the boy car be differ color

There are a number of challenges any lemmatizer needs to solve. First, most languages have irregular word forms e.g. go, went, gone. These need to be handled separately from the regular forms. This includes words that seem to be inflected but are actually a lemma e.g. *walking* is an inflection of *walk* but *bring* is not an inflection of *bre*. Finally, the inflections of two separate lemmas might share the same word form. For example the German word form *gehören* can be the inflection of *hören* (to hear) or *gehört* (to belong). Fortunately, these ambiguities are quite uncommon. [KGS20] found that for English datasets less than 1% of the tokens have ambiguous lemmas.

Lemmatizers are usually evaluated on linguistic treebanks. A treebank is simply a document with syntactic annotations. The term treebank makes sense if we consider that each sentence forms its own syntax tree. Some tree banks have also been annotated with semantic information which allows us to identify the correct lemma for a word. Current approaches [Qi+18; KGS20] manage around 95% accuracy on English texts. A recent evaluation [ORD19] found that we can expect similar accuracy on German texts even with approaches [KGS20] that can handle multiple languages. Languages like Japanese still significantly

benefit from language-specific models [Tak+18].

2.4 Synonyms

Before discussing coreferences in more detail it is helpful to briefly conceptualize synonyms. We will then extend the mental model we developed for the synonyms to coreferences in the next section.

The collective understanding of synonyms is a word pair where one word can be exchanged with the other without altering the general meaning. When searching for synonyms online we are given results and tools that help writers diversify their texts by offering more rare and extravagant alternatives. Despite this intuitive and common understanding of synonyms, there is no scientific consensus on what constitutes as a synonym pair or how to define it. Instead, linguists explore how synonyms are used and in what circumstances.

2.4.1 Strict synonyms

Strict synonyms must be able to replace each other in any context without changing the meaning of the sentence. As such both words of a synonym pair must convey the same idea and their manifestations must be indistinguishable. For example trousers and pants. Both words share the same idea or concept (intensional definition) of a piece of clothing that covers the legs by wrapping around each leg individually. While this is not a perfect definition for pants the point is that based on our mental image alone we can not distinguish between *pants* and *trousers*. Further, this also holds for the physical manifestations of pants/trousers so their extensional definition is also equivalent. Nevertheless one can still argue that this is strictly speaking still not a synonym. After all, they convey a regional difference since in North America pants is used almost exclusively whereas trousers is more commonly used in Great Britain. Due to this regional difference, we also tend to think of pants as looser fitting. As such these terms are not truly and universally interchangeable. Even a minute differences such as *Aluminum* and *Aluminium* can convey a regional difference. Thus depending on the definition, strict synonyms either do not exist at all or are very rare.

[1] Note that the spacy implemention has been updated since the release of the study.

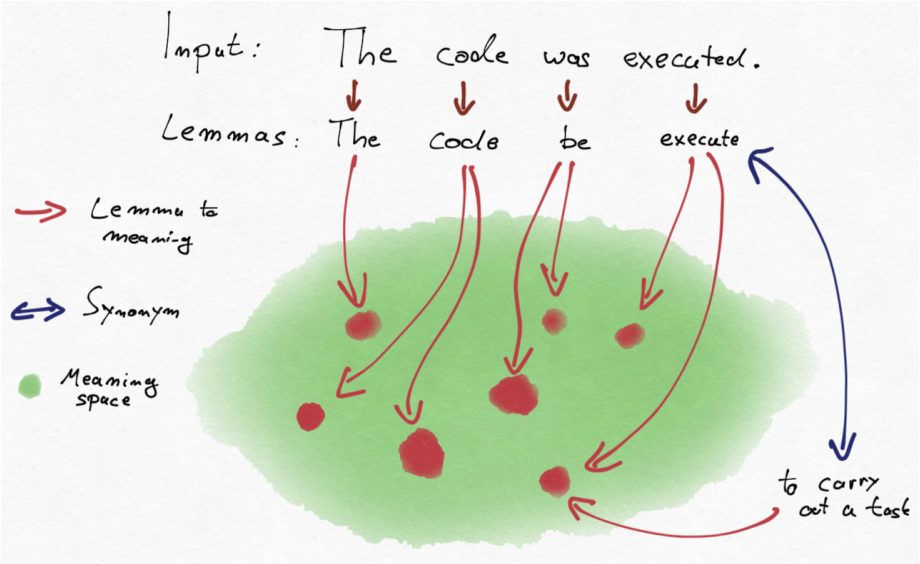


Figure 2: Conceptual definition of linguistic terms in a meaning space. Lemmas with more than one distinct meaning, here red arrow, are called polysemous. Two lemmas that can be mapped to overlapping regions are synonym to each other.

2.4.2 Partial synonyms

Synonym pairs that do not change the meaning of a sentence significantly when exchanged are called partial synonyms. While strict synonyms provide a valuable theoretical concept the distinction provides little benefit for practical tasks. Therefore we will refer to partial synonyms simply as synonyms from here on. Still what constitutes a significant change to classify an example into a synonym pair or not depends both on the context and personal preference. However, in our experience, there is a wide range of examples that can be classified as synonyms independent of personal preference. And then there are instances in which two words are on the border between being synonyms or not. Here personal preference and setting (novel, poem, mail, document, etc.) can lead to different classifications. What makes this even more difficult is that the similarity of two words can depend on their context. For example:

> *to run - to execute*
> Please run the diagnostic tools.
> The regime executed thousands of innocents.

It is evident that we can exchange *run* with *execute* in the first sentence but not

in the second one. For the word "*execute*" from the previous example there are two-word senses. First *to kill* and second *to carry out a task*. One approach that we will discuss more in-depth later is to maintain a list of all word senses for each word and then a list of all synonyms for each word sense. However, even if we had such a list for the time period and language that we are interested in, automatic synonym detection would still not be a trivial task. In order to use these lists, we need to be able to determine the correct word sense for a word. This task is known as Word Sense Disambiguation (WSD) and can be more formally defined as: *WSD algorithms take as input a word in context and a fixed inventory of potential word senses and output the correct word sense for the context.* [JM20]. Identifying the correct meaning of a word is crucial for most Natural Language Processing (NLP) tasks and as such it has a long history in computational linguistics. We will come back to WSD in subsection 2.7.

The previous issue all dealt with the difficulty of transferring the human language comprehension to a machine. There is however another set of problems where even humans can not come to an unanimous decision. For example:

> *distinct (apparent, obvious) - sharp (distinct, well defined)*
> She felt a distinct pain in her left knee.
> There was a distinct edge in the otherwise smooth wall.

In the first sentence, we would argue that the exchange does not really alter the meaning of the sentence. For the second sentence, we intended to keep the same word senses regardless of the change of context also masks the meaning the word adds to the sentences. To add to the confusion another sense of *sharp (cutting, edge-like)* could also be used in either case. As such the use of an adaptable word such as *sharp* confuscates the intent of the author. Instead, the reader gets to slightly shift the meaning to their own personal preference.

So far we have seen that synonym relations connect not between words but word senses. Further word senses are themselves not completely fixed in their meaning. Conceptually we can think of word senses as complex shapes in a continuous meaning space. Each context a word is used in then masks parts of all the associated meaning area of a word sense, Figure 2. This can mask the entire regions of some word senses while leaving the area of another word sense partially unmasked. Each word sense area is defined by the current collective use of the language. Therefore the meaning area of a word sense can change over time and also by geographically separate groups. Some words can also change their word sense entirely. The word *gay*, for example, was primarily used to describe something as carefree, colorful, or happy before it received an entirely new word sense around 1960. And by 2021 the previous word sense has mostly

vanished in favor of describing homosexuality. However, at least conceptually our meaning space stayed the same and just the areas and positions of the word senses of gay changed. The area of this new word sense has a large overlap with the word sense from homosexual and therefore they can in some contexts be used interchangeably. This also helps to conceptualize that some word senses are more synonym than others by defining the similarity through the contextualized overlap between two word sense areas.

2.5 Coreference resolution

Coreference Resolution (CR) is the task of identifying all textual references to the same entity. Unlike synonyms, coreferences can be phrases. Just like synonyms are two word senses that overlap in our conceptual meaning space, two coreferent **phrases** overlap in our conceptual meaning space, Figure 3. Finding a mapping from word forms to some meaning space is already tremendously difficult and far from being solved. To map phrases we not only need to consider the individual word senses that make up the phrase but some phrases carry meaning beyond the individual word senses. We discuss practical approaches for word and phrase meaning mappings and their many shortcomings in the next chapter, section 3. Any model that attempts to present the meaning of words with a vector as we did in our conceptual meaning space is called a Vector Space Model (VSM). Another difference to synonyms is that coreference only really makes sense for entities and not for verbs or adjectives. As such coreferences are always (pro-)nouns or noun phrases. Yet another difference is that unlike synonyms the referents are not necessarily exchangeable. Consider the following example in Figure 3:

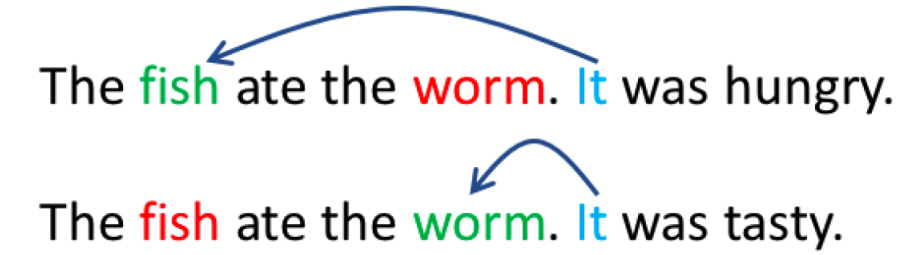


Figure 3: Example of ambiguity in pronoun coreference resolution. Graphic from [ZZS20]

it can not be exchanged with fish and as such, they are not synonym. This is the most common type of coreferences, called an anaphoric reference. An anaphora is a reference back to another phrase that determines the anaphoric meaning, [Mit99]. The anchor phrase of an anaphoric is called the antecedent. In Figure 3 arrows indicate the reference from anaphor to the antecedent. While anaphors can not generally replace the antecedent the antecedent itself can always replace the anaphora. The most obvious and trivial form of coreference is if we use the same phrase multiple times[2]. As multiple mentions of the same phrase are trivial to resolve we want to convert any other coreferences in requirement specifications to trivial (first-order) coreferences.

Pronouns are a common source of anaphors. While the correct resolution of pronouns is crucial for text comprehension and many downstream tasks they are easily resolved by humans. In fact, even instances that contain multiple ambiguities can be resolved unconsciously. Consider the following example from [Ng17]:

> The Queen Mother asked Queen Elizabeth II to transform her sister, Princess Margaret, into a viable princess by summoning a renowned speech therapist, Nancy Logue, to treat her speech impediment.

In the example are three coreferences (1) (Queen Elizabeth II, first occurrence of her), (2) (sister, Princess Margaret, second occurrence of her) and (3) (Nancy Logue, a renowned speech therapist). To resolve the second "her" we consider that Princess Margaret needs to be transformed into a viable princess therefore it is most probable that the speech impediment affects Margaret, not Elizabeth. Thus we had to apply both world knowledge and logic to resolve the pronoun. This illustrates the difficulty of pronoun coreference and in fact, it has been compared to the Turing test. However, this also illustrates that even complex pronoun constellations are unlikely to confuse a reader. Thus we do not consider pronoun and other local coreference resolution problems relevant for resolving ambiguities in Requirements Engineering.

Cross Document Coreference Resolution (CDCR) [BF08] focuses on long distant Coreference Resolution not just in a single document but across a wide range of documents. The term document is somewhat ambiguous but CDCR focuses on finding coreferences between substantial documents that are not directly related. For example finding coreferences in webpages or between books. Coreferences within one project as we are interested in for CCR is not considered. CDCR introduces an additional challenge since identical phrase forms do not necessarily refer to the same entity across multiple documents as they can be from different

[2]This assumption does not necessarily hold across multiple documents

domains [DW15]. One can also consider CDCR as a specific type of document search task. The task would then be to find all coreferences of the search query. As CDCR is connected to search engines many approaches focus on computational feasibility on large document sets such as web indexing. Further, these methods focus on finding the most relevant results but do not attempt to achieve good recall metrics.

Many end-to-end CR approaches we could find use a supervised learning approach and rely on annotated datasets such as OntoNotes ([Hov+06]), ECB+ Corpus ([CV14]) or ACE05 ([Wal05]). All these corpora adopt the entity categories[3] established by ACE. Unfortunately, these types do not capture the phrases that would describe product components, which we are interested in. If a sufficient RE dataset should be available in the future CDCR methods do provide an interesting analog as they provide more sophisticated alternatives to simple pairwise comparisons, see [Jos+19; KR20; Lee+17; HLL20].

2.6 The component coreference resolution task

This thesis aims to assist in the creation of consistent terminology in highly specialized domains through the resolution of coreferencing phrases. We will explore this aim with a case study on a set of software requirements. Requirements define a novel project and as such the more opportunities for inconsistent terminology. To the best of our knowledge [Wan+20] introduced coreference resolution to the field of Natural Language Processing for requirements Engineering (NLP4RE). While coreference resolution perfectly describes what we are trying to achieve, many different branches such as anaphora resolution focus on entirely different aspects of CR. As such we believe it is beneficial to propose the new term of Component Coreference Resolution (CCR) to differentiate the task from other branches of CR. We choose the term **component** to indicate that we are only interested in phrases that are relevant components of a described project rather than any noun phrase. We also decided against *entity* as it is too widely and *named entities* too narrowly defined.

CCR is the task of identifying non trivial coreferences between components in a list of sentences. The sentences in the list are assumed to be related such that identical phrases are always coreferent. Identical phrases are therefore trivially coreferent and not considered. A component is any for the project relevant entity that can be isolated and defined. Typically, any phrase that one would

[3] ACE types: Person, Organization, Location, Facility, Weapon, Vehicle, and Geo-Political Entity (GPEs). Each type is further divided into subtypes (for instance, Person subtypes include Individual, Group and Indefinite).

expect in a glossary for the list of input sentences qualifies as component. In any real document we would need to compare every component we extract with all other components. However, the core of the CCR task is the comparison of two sentences. In our case study these sentences happen to be requirements. For a pair of requirements we need to identify their project components and then compare components. We will focus exclusively on this core component of CCR. We assume input and output to the CCR task as:

Input: Sentence pair

Output: $\begin{cases} \text{Coreferent pair} & \text{if Input contains coreference} \\ \text{Empty} & \text{else} \end{cases}$

Example:

Input: [" *The Clarus system shall be able to implement quality checking rules for each environmental parameter.*", " *The Clarus system shall display unused environmental variables.*"]

Output: [" *environmental parameter*", " *environmental variables*"]

Given a model that can produce the output as specified we can extend this to cover any list of requirements. We simply call the model for every possible requirement pair. There are certainly more elegant solutions but we will leave those to future works.

2.6.1 Linguistic features of project components

Here we will have a closer look at product components and their linguistic features. First, we should note that project components are in part unique and as such, they form a project-specific terminology. Unlike terminology in established fields, the terminology in Requirements Engineering (RE) is much more susceptible to variation as no commonly known and accepted consensus has been reached since a novel product is being specified. In other words, RE often needs to define new project-specific terms. Thus, two authors may come up with different terms for the same concept. Since these terms are new they are non-lexical. However, it is reasonable to assume that they are formed similarly to their already recorded lexical counterparts. [JK95] examined the terms used in English documents from four different domains; fiber optics, medicine, physics & mathematics, and psychology. As these are established fields they simply picked 200 terms from a lexicon of each domain for a total of 800 samples. They found that the average number of words per term is above 2, except for

the medical domain. They reason that this is due to the more dominant usage of Latin terms. Many of the used Latin terms have multiple Latin root words and would actually be multi-word terms in English. For example, *tendonitits* has two roots tendon and itis. Further out of the 800 only 35 terms did not contain a noun. This differentiates the task from synonym detection as synonym can also be found between verbs and adverbs. To actually identify potential coreference phrases we will make use of the finding:

> Almost all terminology (92.5%-99%) contains nouns independent of domain. (1)

We will highlight critical aspects throughout the literature review like Finding 1 and later base the design of our CCR pipeline on these findings.

2.6.2 Sources of ambiguity in requirements engineering

As pointed out in [Kam05] most linguistic ambiguities do not cause trouble in requirements engineering as they can be easily resolved. However there is extensive evidence[4] that terminological inconsistencies such as coreferences can cause misinterpretations and need to be addressed. To summarize, the proposed CCR task identifies if two phrases should be unified, which in turn creates a consistent terminology within the requirement specification.

In general, there are two steps to address terminological inconsistencies. First inconsistencies have to be identified and then they can be resolved in a second step. We will focus on the identification step of the CCR task in this. Some approaches work a similar problem none of them disclose their datasets [Wan+13; PLM15; DSL18; Wan+20]. While it is understandable that datasets from industry partners can not be published it makes comparisons significantly more difficult. We address this issue by providing a small but public validation dataset for the CCR task. Unfortunately, the data sparsity does not allow for supervised models to rely on it for training. As such we will propose an unsupervised approach.

We do not further discuss resolution strategies and user interactions in this thesis. We want to emphasize that a real word application of coreference resolution requires information visualization to properly communicate the found coreferences to the authors. [LC99] proposed a general user interface for requirement defects that provides context information and proposes a resolution that the

[4] [Wan+13; PLM15; DSL18; Das+21; TH19; JK95; Kiy+08]

user can accept or edit. While the proposed interface is simplistic it shows that the identification of requirement defects is not enough. [DSL18] showed that good visualization of coreferent phrases can significantly help to resolve coreferences. They developed an information visualization tool specifically for user stories that can be clearly associated with a stakeholder group. They report an average recall of 0.344 without their tool and 0.68 recall for the group using the tool despite both groups access to the same raw tool data. This shows that the correct representation of coreference pairs is crucial. However, even the best representation is ultimately constrained by the quality of the identification step.

2.7 Word Sense Disambiguation

Word Sense Disambiguation (WSD) is the task of identifying the meaning of a word given its context. If two words share a meaning they should be coreferent. Initially, this sounds exactly like our CCR task but with a focus on word senses rather than phrase senses. A Knowledge Base (KB) is just a collection of word senses and the relations between collected word senses. Most WSD approaches treat the task as a classification problem. For a given sentence we will then need to identify the corresponding word sense. The list of possible word senses for each word is taken from a KB.

2.7.1 Word Networks

There are several ongoing projects dedicated to finding and defining every word sense of all words found in a given language. They manually record word form to word sense mappings and even some sense to sense relations such as synonyms. Since human knowledge is encoded in a lexical manner these graphs are called lexical knowledge bases (LKB). Just like a lexicon the keys in WordNet only record the lemma, subsection 2.3, of each word. Unlike a lexicon, WordNet [Mil95] has to differentiate between different word senses in order to capture word sense relations such as synonyms. They do this by recording multiple separate versions for each word sense of a lemma. For example, if we wanted to look up the word *trashed* we would first need to find the lemma *trash* and would get the following results from WordNet:

> trash.n.1 worthless material that is to be disposed of
> trash.n.2 worthless people
> trash.n.3 nonsensical talk or writing
> trash.n.4 an amphetamine derivative (trade name Methedrine)

trash.v.1 dispose of (something useless or old)
trash.v.2 express a totally negative opinion

Each entry is a sense of the word trash along with its type (noun or verb) and a number that simply counts the senses. We can then pick a single word sense and query WordNet for any linguistic relation to other word senses. For example if we asked for synonyms of *trash.n.1* it would report *[rubbish.n.1, scrap.n.2, waste.n.1,...]*. Thus in order to use tools like WordNet for CCR we need to be able to correctly lemmatize words and then pick the context-appropriate word sense out of the list. Selection of the correct word sense is the WSD task introduced above and will be further explored below, subsection A.1 later in this chapter. However, we have already seen that lemmatization, subsection 2.3, comes with its own challenges. Current methods achieve around 95% accuracy. This is certainly impressive however considering that the average A4 page contains around 500 words we can expect around 25 lemmatization errors per page. So even before we address the significantly more difficult WSD this provides a barrier to the use of LKB.

Among the most well known efforts is WordNet [Mil95]. WordNet is only for the English language and captures not only synonym relations but also hierarchical relations[5]. WordNet is an ongoing research objective and as such it has received numerous updates [MM02] [MR19] since its initial publication. Some continuations are published under a different name such as BabelNet [NP12]. While some publications are working on other languages, e.g. [Bak+21], the English WordNet has received more attention and contains the largest number of relations. Any of these systems are essentially large graph structures that can be manually and procedurally extended. As such it is difficult to compare these methods with machine learning approaches. We can compare the number of synonym relations captured in each graph. We are also able to create a benchmark task where we can evaluate different machine learning approaches against each other. However, publicly evaluating a graph of synonyms on a public benchmark task is futile. Not only does it require two other challenging tasks, lemmatization and WSD, to be solved but any public benchmark could easily be solved by the next publication by adding the benchmark entries to the graph for a perfect score.

To summarize graph structures are limited by their vocabulary and recorded synonym relations and without significant manual efforts unable to adapt to new domains. Further, even a theoretical complete synonym graph would be difficult to utilize in an automated system as the required disambiguation of word senses and lemmatization are itself open fields of research. We provide

[5]Such as hypernyms, homonyms, meronyms, troponyms, and antonyms

an example of a WSD algorithm in Appendix A. A more complete overview of knowledge bases can be found here [Nav18]. For later we will keep in mind that:

> WSD algorithms that use a KB can not directly be applied to phrases. Do to the inherent structure of KB they only work on single words. (2)

3 Semantic Similarity with Vector Space Model

The goal of vector space models is to find a mapping from natural language to such a meaning space. This is the conceptual space we introduced in subsection 2.4. The goal is to find a vector for each word that encodes not only its meaning but also semantic relations. The hope is to place similar words closer together and also encode semantic relations at the same time. This would allow for semantic operations like E(Queen)-E(Female)+E(Male)=E(King) where E is our hypothetical embedding to the meaning space. Given an ideal embedding the phrases that are tightly clustered together should all be coreferent. Our hope is to find a semantic embedding that maps phrases in a way that allows us to differentiate between coreferences and random phrases. We will spend the rest of the chapter covering the general ideas of word embeddings and the major breakthroughs of the last decade.

As it turns out constructing such a mapping is a rather difficult endeavor that has kept the research community busy for the better part of a century. Most approaches are based on the distributional hypothesis which states that *words that occur in similar* ***contexts*** *tend to have similar meaning* [Har54]. Considering how crude this definition of meaning is it yields surprisingly robust meaning spaces with an unsurprising number of shortcomings.

3.1 Statistical word embeddings

The first and most broad interpretation of context was that of a whole document. What exactly constitutes as a document is task dependant but for each document, we simply count how often each word form occurs. Each row, Figure 4, represents the distribution of a word form across different documents. According to our hypothesis, these rows should embed the semantic meaning of a word form as a vector. Therefore these vectors are referred to as word embedding or simply word vectors. Despite the crude approach, [SWY75] showed the usefulness of term-document matrices for search queries. This approach was

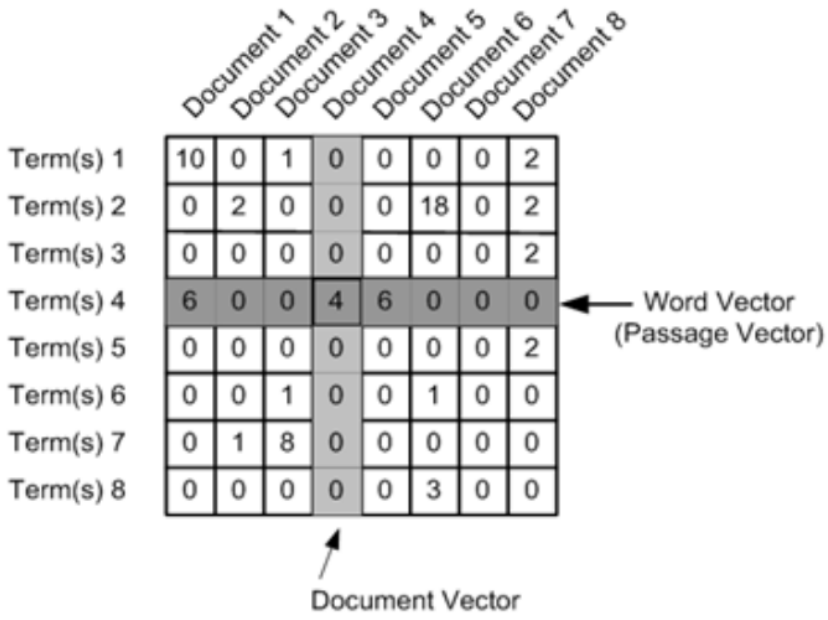


Figure 4: Term-Document matrix. Source not determined, found online[6].

refined by reconsidering what context to use. A term:term matrix records for each word form how often each other word form occurs in the same sentence. Thus sentences now replace the documents as a more local context. As such we get a lot more context samples from the same documents. Instead of considering the whole sentence as context, we can also consider k words before and after each occurrence of the target word. This results in the simple count of co-occurrences of word forms in a k neighborhood. The resulting matrix, Figure 5, will be quadratic and symmetric and each row and column represent one entry in the vocabulary. However, our resulting vectors will be heavily biased since some words naturally occur much more frequently than others. If a target word form co-occurs with a common word form such as "The" it tells us much less about the meaning of our target than the co-occurrence with a rare word. Among the most popular approaches to address this statistical bias are inverse document frequency [Spa88] and pointwise mutual information [CH90]. For a better overview of statistical vector space models, we refer to the survey by [TP10]. Finally, we can use dimensionality reduction methods to enforce a

[6] https://medium.com/analytics-vidhya/featurization-of-text-data-bow-tf-idf-avgw2v-tfidf-weighted-w2v-7a6c62e8b097

Count	This	is	an	example	sentence
This	1	1	0	0	0
is	1	1	1	0	0
an	0	1	1	1	0
example	0	0	1	1	1
sentence	0	0	0	1	1

Figure 5: Co-occurrence matrix with a context window of 3 around the target word. For each word in each sentence we can then record its coocurances.

			Cat	Dog	House	Plank	Boat	is	
Cat	->	(	1	0	0	0	0	0	)
is	->	(	0	0	0	0	0	1	)
House	->	(	0	0	1	0	0	0	)

Figure 6: One-hot encoding with a toy vocabulary of 6 words.

greater correspondence between contexts and improve similarity measures. Traditionally, due to the computational and memory requirements, Singular Value Decomposition (SVD) was only applied to the smaller term:document matrices.

This explicit statistical approach remains relevant today and is utilized in more recent word embeddings such as Glove [PSM14].

3.2 Neural word embeddings

Many recent advances have been powered by neural networks. [Mik+13] were the first to successfully apply a neural network to word embeddings and demonstrate some capability for semantic operations. Their *word2vec* model works by using neighbourhood around a target word form as input and then the network should guess the target that we removed from the input. As neural networks are essentially matrix operation, the text input needs to be converted to a vector representation first. For this they use one-hot encoding to transform the input neighbourhood to a vector. One-hot encoding, Figure 6, is a vector of the size of the vocabulary that can encode a single word. To represent a word the dimension that corresponds to that word is set to 1. It is called one-hot

encoding because only one dimension will be set to 1 at a time. As a semantic embedding one-hot encoding fails but we only use it to convert word forms to a uniform and sparse vector representation. Providing the whole context with one-hot encodings at once to the network as input is called continous bag of words (CBOW). Skip-Gram creates target context word pairs instead and thus splitting a single context into multiple samples that are forwarded through the network individually. The following example illustrates the {input, output} for CBOW and Skip-Gram tasks:

Sentence	*The man who passes the sentence should swing the sword.* [Mar11]
Target	swing
Window	5
CBOW	{sentence should the sword, swing}
SkipGram	{swing,sentence}{swing,should}{swing,the}{swing,sword}

Note that SkipGram predicts the neighbourhood from the target word and CBOW predicts the target word from the neighbourhood. As with all neural networks the model is trained by feeding samples one at a time through the network and computing how the network flow needs to be adjusted based on an objective function. As can be seen in Figure 7 depicting the Skip-Gram task we use a one-hot encoded word (swing) as input and its word pairs as the target (sentence/should/the/sword). As with any one-hot encoding the vectors have the same size as the vocabulary. During training the output is compared to the target. The difference is measured with the cross entropy loss. Based on the loss a small correction is applied to the matrices. As with any local optimization it takes a lot of iterations with different inputs before the system converges to a with near certainty suboptimal solution. What local optimizer lack in accuracy they make up for in ease of use and comparatively minuscule resource requirements. Once the training has converged we can compute the word embedding simply by multiplying the embedding matrix with the one-hot encoding of a desired word. Interestingly if we remove the hidden layer we essentially end up with a co-occurance matrix. Though here the neural network would address the frequency bias for us, as shown in [LG14]. The hidden layer is significantly smaller than the vocabulary. Thus the network is forced to encode statistical information and thus also semantic information. The hope is that the network is able to remove noise and unnecessary information and merge similar contexts. This is analog to truncating a co-occurance matrix with a kernalized SVD a process called latent space analysis. In fact [Dom20] showed that any gradient descent method is approximately equivalent to a kernel machine. While this is of little practical relevance it does allow us to apply insights from SVD to neural networks and vice versa.

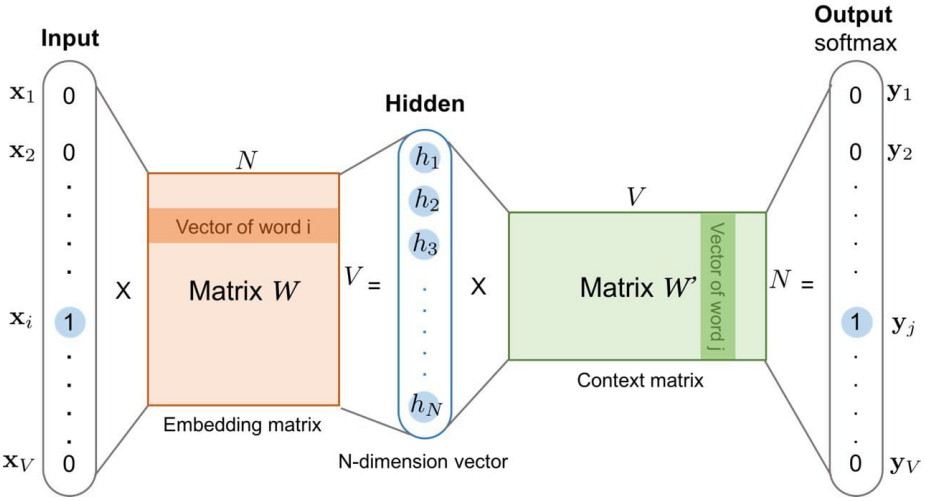


Figure 7: Skip-Gram word2vec model. Source [Wen17]

Both *word2vec* and co-occurance matrices are limited by the vocabulary of their training corpus. If a word is not present in the corpus our methods simply can not provide us with a vector. In effect, they function like simple lookup tables, we can not access unknown entries. This problem is made worse as usually rare words are trimmed since there are not enough training samples. Most methods also avoid lemmatization so that the word embeddings actually also contain grammatical information besides meaning[7]. Including inflections obviously inflates the number of words in the vocabulary. Together inflections and trimming make out of vocabulary (OOV) occurances in practice quite frequent.

To address the OOV problem newer methods use sub word tokenization. If we consider a spectrum from splitting input only between words and on the other side giving every character its own token, then the *word2vec* model lies on one end. On the other extreme are character aware convolutional methods like [Kim+15]. They use a character vector conceptually similar to one-hot encodings of ASCI characters to make characters accessible for a neural network. Then they use a Convolutional Neural Network to learn word embeddings from the separate character tokens. It turns out for most tasks a hybrid approach like FastText[Boj+16] or word pieces[Wu+16] provide better performance as demonstrated in the respective papers. Below we can see an example of word piece tokenization:

[7] Whether or not grammar is actually distinct from meaning depends on ones interpretation of meaning. However having grammatical information is obviously beneficial in some cases necessary to understand a sentence or how a word contributes to a specific sentence.

The EIRENE system shall enable ...
'The', 'E', '##IR', '##EN', '##E', 'system', 'shall', 'enable' ...

As we can see most words were embedded as normal word vectors. However the unknown word "EIRENE" is split into a mix of character and two character tokens. The ## indicate that this token belongs to the same word form of the previous token. While completely unknown words like "EIRENE" are unlikely to receive an embedding that actually captures its meaning, the model does provide a unique set of vectors for the word. This makes word pieces extremely useful for text vectorization. It provides a much more compact alternative to one-hot encoding and as demonstrated solves the out of vocabulary problem. As such many modern approaches rely on methods like these to vectorize text input. Even more impressive, if two known words are combined these approaches have a good chance at correctly splitting them during tokenization. As each part of the word is known sensible embeddings can be provided for some unknown combinations and even inflections. Lets imagine a training set that does not contain the word *airspace*. A word piece model trained on this dataset can still provide a reasonable embedding for *airspace* as long as it has learned embeddings for both *air* and *space*. Even if *airspace* would have been present word pieces would have split the word in two tokens. Word pieces also tends to split word stems increasing the number of training samples for each token. For example *walked*, *walker*, *walks* would be split to *walk ##ed*, *walk ##er* and *walk ##s*.

3.3 From context to contextual embedding

Even after addressing the OOV problem, all models that have been discussed so far still have a common flaw. They map a word form to a single point in a semantic vector space. They create an embedding that is static and only depends on the target word form. This makes it impossible to express polysemous words, words with more than one meaning such as *bank*. In the best case, the vector represents the meaning of the dominant word sense.

However, most approaches do not even account for the possibility of multiple word senses. Instead, all contexts regardless of word sense are aggregated when computing the word vector. As long as the dominant sense makes up a large majority of all word form occurrences, the embedding remains close to the dominant concept. As the proportion of other senses increases, the vector is pulled to the mean of the separate senses in the semantic space. The one-to-one mapping has the benefit that the results of the embedding process can be stored in a simple data structure, as noted before acting in effect like a look-up table. This makes these models very easy to use and they do not require dedicated

hardware after training. The models that use sub-word tokenization do, however, come with memory requirements that are unreasonable for most end-user devices. FastText for example requires about 7 GB.

Conceptually it is quite simple to address the one-to-one mapping problem. Instead of asking the network what the best embedding for our target word is, we ask what the best embedding for a word in a given context is. So using the example from above, we want to create specific embedding for the word *swing* given the sentence "*The man who passes the sentence should swing the sword*". Now we take the neighborhood of "*swing*" from our sentence "*sentence should the sword*" and create a vocabulary vector[8] for it. If we now multiply this vector with the context matrix instead of the embedding matrix, Figure 7 green matrix, from a trained *word2vec* model, we get a vector that specifically represents 'swing' for this given context. Granted this contextualized word embedding is missing the most important piece of information to create a word embedding, the target word. As such this would be a very poor contextualized embedding and unlikely to be of practical use. However, it is contextualized and we can easily verify this by creating an embedding for two separate occurrences of swing. The distinct neighborhoods will lead to different vectors.

To improve the contextualized embedding, we have to take the target word along with its neighborhood into account. With the *word2vec* tasks, neighborhood and target are always separated between input and output. This means it's impossible to create an embedding that takes context and target word into account, as we would need to fix both input and output when evaluating such an embedding. For properly contextualized word embeddings, we need a model with a different training task. A task where the target word can be both in the input and the output. One such task is Masked Language Modeling (MLM). Here the network is given a sequence of words from a text but one word/token is hidden or masked. The network is tasked with finding that hidden word/token. For example, given "*The man who passes the MASK should swing the sword*" the goal is to predict the token for "*sentence*" or possibly multiple tokens if *sentence* was split into multiple tokens. So while the network is tasked with generating the hidden token, it also has to generate tokens for all the other words in the sequence. If we want to create a contextualized embedding for "*sentence*" we now have everything we need on one side of the network. Both our target word "*sentence*" as well as its context had to be vectorized and returned by the network. Thus unlike *word2vec* we can now investigate the embedding matrix with the target token and its context. The natural next question is how the target and context vectors can be combined. We will address the issue in subsection 3.5. In practice the issue is most often avoided by forwarding

[8]The sum of the individual one-hot encodings of the neighborhood words

a sentence through the whole network and look at the last hidden layers as contextual vectors. This second approach also needs a training objective that allows the target word to be in the output. Language models can generate longer texts by using the output of one prediction as input for the next. Similar to *word2vec* the real strength of this task is that almost any text can be used for training with little preprocessing necessary.

[Kim+15] uses such a language model task to create contextualized embeddings. Just like before natural language needs to be converted to a vector representation before it can be processed by a neural network. Instead of using the extremely sparse one-hot encoding here, they encode the natural language with previous static embeddings. By treating the word embeddings as an encoding of language instead of a semantic representation it does not matter that they can not capture polysemy. Instead, the input is just a compact numerical representation of the word forms. The model then can create contextualized representations from the encoded input as described above. [Pet+18] combined the approach with a bidirectional LSTM model to create one of the most well-known models up to date ELMO (Embeddings from Language MOdelling). They showed that the word embeddings learned in the LM task can significantly increase performance on many downstream tasks such as textual entailment, semantic role labeling, coreference resolution, or named entity recognition.

3.4 The age of transformers

The traditional neural networks, like *word2vec*, that just contain a bunch of neurons that are connected in a layer-wise manner are called Multi Layer Perceptron (MLP). In theory, MLPs are universal function approximators which means given enough training samples we can solve any problem. Or at least we can find an approximation to any function. So we can solve any problem that can be formulated as a mapping from some input to some output and isn't random. In the real world, however, there are pesky obstacles such as finite computational or memory capabilities. This makes it necessary to develop an alternative neural model that needs fewer training samples. Having fewer parameters in a model means fewer adjustments need to be made. In turn, less memory is needed, and since fewer neurons are used it also requires less computational power. Unfortunately just reducing the number of parameters in an MLP leads to underfitting for any but the most simple problems. For a thorough overview of traditional deep learning methods, we can recommend [GBC16].

So the real question is how can we get the best results with a reduced set of parameters. One approach that is particularly popular in computer vision is

to limit the connections between layers. In MLPs each neuron of a subsequent layer is connected to every input. Convolutional Neural Network (CNN) reduce the number of connections by only considering a small area in the input and summarizing it. As many modern photos have literally millions of pixels, connecting every neuron in a hidden layer to every input quickly inflates the number of connections. Instead of millions of connections, most CNNs use a kernel size between 5x5 and 11x11. Pooling layers allow different input dimensions to be reduced to one fixed representation. This is great to deal with variable image dimensions but makes the model rotation and to some extend translation invariant. This is fine for most vision tasks. If we build a network to classify whether an image contains cats or not we don't care where the cat is in our image data. In language the order of words in a sentence matters. While this may seem to disqualify CNN for NLP, relying on local information only does give impressive results in some NLP tasks. And since they so significantly reduce the number of parameters they are extremely fast. This allows CNNs to explore more complex model architectures.

The alternative to reducing the number of parameters in the model is to reduce the input size. A smaller input means fewer values that subsequent layers need to connect to and ultimately also fewer parameters overall. Instead of considering a full sentence at once, its tokens are forwarded through the network one at a time. This should be quite intuitive as humans read in the same manner. We read one word at a time but retain the knowledge of previous words to actually understand a sentence. This mechanism of retaining knowledge is achieved by using part of the output of the network as additional input for the next iteration. These networks are known as Recurrent Neural Network (RNN). Gated Recurrent Units (GRU) and Long Short Term Memory (LSTM) are the most popular RNN networks for NLP. As the name suggests LSTMs have been designed to retain information across long sequences. In reality, the finite and often reduced precision of floating-point operations leads to vanishing gradients[9]. So while in theory LSTMs can retain information across a long sequence in practice they are expensive to train and fail to retain information for more than a few words in sequence.

In [Vas+17] they introduced the transformer architecture which addresses the performance concerns of RNNs and the word order problem of CNNs. Transformers are based on the idea of self-attention. MLPs pay attention to all inputs at any time. Most of these inputs are irrelevant and the computation is wasted. In CNN we limit the attention of the network to a small region in the input and ignore the rest. Transformers on the other hand learn themselves what inputs

[9]The gradients used to update the neural network tend towards zero. This leads to in practice no updates at all.

they should pay attention to. As such they have, in theory, infinite support. In other words, no matter how long the input is, transformers can learn which parts are related and relevant. The self-attention mechanism can actually learn to compute a convolution [CLJ19]. As such transformer modules can be seen as a generalization of CNN. Just like CNNs, they fail to consider the ordering of the input, as recently proven by [DSS21]. To address this positional information is encoded directly into the data itself. That allows the network to learn the positional superposition on top of the actual token embeddings and make use of them.

Transformer provided a significant leap on almost any NLP task as demonstrated by the BERT model [Dev+18] that proposes a particular configuration of transformer encoder modules. BERT is the most used transformer architecture and there are dozens of derivatives. Another well know family of transformer architecture is GPT-2 [Rad+18] or GPT-3 [Bro+20] which utilize the decoder modules of the transformer architecture. Unlike BERT it uses a unidirectional language model. This means it only considers tokens from left to right. This works great for text generation but limits the applicability to other tasks. Virtually all improvements in NLP since 2017 have been achieved by models that use some part of the transformer architecture. And thus began the age of transformers.

3.5 From words to phrases

The previous sections provide a rough outline of different approaches to generate semantic embedding for single words. We also discussed that we can create embeddings for subword tokens. For CCR we need to be able to compare the meaning of phrases rather than the meaning of words with each other. Extending tokens to whole phrases is clearly unfeasible. By considering *swing the sword* as one token the number of occurrences per token in any text will be drastically reduced. Further, it would drastically inflate the number of tokens and would significantly boost out of Vocabulary (OOV) occurrences. There are many more reasons why this approach is unlikely to perform well. We can not expand tokens to phrases due to the exploding combinatorics. Instead we need to combine the embeddings of single word forms to a vector that represents the whole phrase. The resulting vector is referred to phrase embedding. Similarly sentence embeddings refer to vectors that represent a whole sentence.

While there are a few works specifically targeted at phrase embedding the lack of a benchmark dataset makes it difficult to judge their results. Both phrase and sentence similarity are subtasks of Semantic Textual Similarity (STS). How-

ever most benchmarks that are meant to evaluate STS performance are sets of sentence pairs with manually annotated similarity scores. They record the similarity in 5 discrete classes from 5 to 1. Sentences with a similarity of 5 have basically the same meaning. Sentences with a similarity of 1 have no semantic connection. Most models return a similarity metric in the range $[0 - 1]$. To compare the predicted similarity metric with the similarity score from the benchmark the correlation between the two is calculated. Many of the benchmarks have been published as a challenge set by the SemEval conference. In 2017 [Cer+17] they provide a collection of all previous SemEval datasets and this has become one of the most prevalent benchmarks in semantic similarity.

Still, fundamentally it doesn't matter if we want to compare phrases or sentences, we still need to somehow aggregate the information from individual word embeddings. That allows us to make use of the sophisticated networks that generate high-quality word embeddings for us. We want to use these embeddings to compute semantic similarity between the target phrases/sentences. The most common approach is to add a small neural network to the embedding layer and train it with the task to compute the similarity. During training[10] the additional network would learn how to aggregate the vectors for us. This would of course require a training dataset with some kind of similarity labeling. Since we have to manually create a training set these approaches are supervised. In fact, the best STS models are supervised methods that require sentence pairs along with their similarity label for fine-tuning. As described before we do not provide a training set for the CCR task. Even if we had enough examples for fine-tuning supervised sentence similarity models it is still not directly applicable to CCR. Supervised STS methods take two text sequences as input and output a similarity score. For the CCR we need to compare only a part of a sentence with a phrase from another sentence. So in order to use the supervised methods directly, we would need to only input the phrases we want to compare. Thus we will remove the contextualization that is, as explained in subsection 3.3, often necessary to correctly disambiguate words. Finally, as these methods have been trained on full sentences using them to compare phrases that are on average only $\frac{1}{4}$ of the length might further degrade the quality of the similarity score. Supervised methods might still yield good results on CCR however we can not fine-tune these methods to our dataset or even domain. Therefore it makes sense to also consider unsupervised methods as these don't require labeled data to be fine-tuned.

So we are left with manually combining token embeddings to a phrase embedding. We take the raw word-embeddings of all words in a phrase from an

[10] Technically this is refered to as fine-tuning and will be further explained in subsubsection 5.2.2

unsupervised model and arithmetically combine them into a single vector that represents whole phrase. Since we are now using unsupervised models we can adopt the word embeddings themselves to our domain by training these models just on requirements. The approach that would retain the most information would be to concatenate all token vectors of a phrase. This will create phrase embeddings where the dimension of the embedding vector depends on the number of tokens in the phrase. This makes it difficult to compare two phrases as most similarity metrics are only meaningful if both vectors are of the same length. For example, computing the cosine similarity between two vectors requires that both vectors have the same dimension and that both vectors use a common basis. Thus to compare phrases we need their embedding to be of fixed length.

> Supervised models extract relevant information from the model for us. In unsupervised models we have to decide how to extract, condense and compare information from the model weights. (3)

We already discussed that we can not use supervised learning methods to combine vectors or learn a similarity function for us. Thus in order to create a fixed size embedding we need some arithmetic operation to combine the vectors. There are two obvious choices, we can either average the vectors (average-pooling) or choose the max value for each vector component among all vectors (max-pooling). To evaluate which operation performs better we also need to choose a distance measure between the vectors. The most common metric is the p-norm of the vector difference. Or for p=2 simply known as the euclidean distance. However, looking at the word embeddings we notice that the norm of the resulting word vectors is very heterogeneous. This is a direct result of the training process. Consider the *word2vec* model. Every new training sample is going to slightly pull the input word embedding towards the target word, subsection 3.2. So by feeding {cat, cute} to the *word2vec* model the embedding of target word *cat* is going to be adjusted towards the vector of *cute*. A more ambiguous word such as *the* is likely to be affected by opposing adjustments that partially cancel each other out. Thus the magnitude of the *cat* vector is larger than that of *the*. This could of course be easily addressed by normalizing word vectors before comparison. The actual downside of the euclidean distance, or any other p-norm, is that it considers any dimension where the two vectors are different. In a sparse high dimensional space where most features are close to zero this means that any time only one of these vectors differs from zero it will affect the distance measure. The cosine similarity uses multiplication rather than the difference and as such ignores a dimension if either vector component is close to zero. This means cosine similarity focuses on overlapping features

instead of absolute differences. To some extend we can actually consider cosine similarity as the correlation between vectors. This is especially useful for high dimensional sparse spaces such as our word embeddings. Thus, the consensus in the NLP community is to use the cosine in between the vectors. Using cosine as the metric average-pooling outperforms max-pooling for semantic similarity [Dev+18] and can be also seen in the results subsubsection 5.2.6. In fact, the average *word2vec* and Glove embeddings [PSM14] still provide a competitive baseline for the STS benchmarks [Cer+17]. Despite encouraging results there is no theoretical justification why average-pooling in combination with cosine similarity should work on embeddings from *word2vec* or any other model. To summarize:

> Unsupervised models can easily be adopted to a specific domain whereas supervised models require substantial manual effort to adopt to a domain. (4)

> In order to combine word vectors of a phrase into a uniformly sized vector we can use pooling operations such as average- or max-pooling. (5)

[Zhe+19] however provided a theoretical justification for an alternative metric. So far we have considered the words to be embedded into a conceptually euclidian meaning space. [Zhe+19] propose to consider vector embeddings as the membership function of a fuzzy set instead. Fuzzy set theory is a well-established extension to classical set theory. Instead of assigning elements as either in a set or not, fuzzy sets allow for degrees of membership. Fuzzy sets are made up of two parts. A universe that contains all possible set elements $\mathbb{V} = \{w_1, w_2, \ldots, w_N\}$. And a membership function that specifies the degree of membership for every element in the universe $\mu : \mathbf{V} \longrightarrow \mathbb{L}$. Traditionally membership degree is scaled to be in $\mathbb{L} = [0, 1]$. To represent the meaning of a word with a fuzzy set we can use the vocabulary as the universe and the relatedness to other words of the universe as the membership functions. For example the fuzzy set for *cat* should contain a whole lot of *cute*, some degree of *animal* and *pet* but not a lot of *airplane*. Measuring the co-occurrence of words in large text corpi is one kind of relatedness measure and thus they can also be interpreted as the membership function of a fuzzy set. The authors also show that we can use word embeddings as membership functions. We have already discussed that word embeddings are analog to SVD reduced co-occurrence vectors and as such, this is consistent with previous findings. If we consider word embeddings as part of fuzzy sets we can also use set similarity measures on word embeddings. They proposed to use the Jaccard Index and achieve signif-

icantly better performance over the conventional cosine similarity. In order to use set similarity, we have to ensure that we only apply valid set operations to our membership functions (word embeddings). Unfortunately, average-pooling is not a set operation and as such we can not apply a set similarity and expect reasonable results. However, the union of fuzzy sets is the max-pooling of the membership functions $A \cup B = (\mathbb{V}, max(\mu, \nu))$. As such the authors promote the use of max-pooling and show that if we use max-pooling we can apply set similarity measurements. They further demonstrate that set similarity (DynaMax) outperforms cosine similarity on all semantic similarity benchmark tasks, subsection 3.5, for all prominent static word embeddings (word2vec, GloVe, fastText). It even demonstrates that set similarity of simple static word embeddings can compete with much more complex and specialized solutions such as universal sentence encoder (USE) [Cer+18b]. This gives us the following options:

> Unsupervised models provide word vectors that can be compared with vector similarity methods such as cosine, Jaccard or DynaMax. (6)

> Set methods require set operations to combine word vectors together. Max pooling is a valid set operation. (7)

3.6 Related works in Requirements engineering

The removal of defects in the specification phase of a project is significantly cheaper than in any subsequent stage. Therefore it is natural to explore a system that prevents defects or detect defects and allows corrections. Prevention is attempted by specifying all requirements in a formal grammar that either doesn't allow certain defects to be expressed or can be automatically evaluated. While formal grammar can be constructed such that some defects can not be expressed, their complexity can be a serious barrier. Similar to program code it not only requires a trained expert to write formal requirements but also to read and understand them. As this can prevent stakeholders from understanding the requirements of their own projects, unsurprisingly, a large majority of requirement specifications use either natural language or a semi-formal language that superimposes sentence templates on natural language. As such systems that can detect requirement defects in natural language are necessary. For an extensive discussion on types of ambiguity in requirement specifications, we refer the reader to [Kam+03]. In the survey, they also discuss methods of addressing lexical and syntactic ambiguities. However, it was difficult to address semantic ambiguity at the time, because in general NLP had not advanced enough. Further, they do not consider coreference detection or similar tasks such as duplicate detection. In fact, a recent mapping survey [Zha+20] showed that the field of NLP4RE is still a niche field of research with about 23 studies on the topic per year over the past 20 years. However, the field enjoys explosive growth probably fueled by the opportunities recent NLP advances offer. The survey only identified 8 papers using word2vec, 2 papers using Glove and none utilized a transformer. Since the release of the survey, there have been a few papers that use a transformer for requirement classification such as [Hey+20; You+20]. Other surveys such as [SJ15; AA21] also show that a large part of published papers are concerned with requirement classification. We found two domain-specific methods that are related to CCR and make use of recent NLP techniques. We will present both in the following sections.

3.6.1 DeepCoref

[Wan+20] aim to resolve coreferences between previously identified multi-word form entities with their DeepCoref model. Unlike Component Coreference Resolution (CCR), subsection 2.6, they **do** consider trivial coreferences that have identical word forms as coreferences. As such achieving high accuracy on their dataset is much easier. Further, their method requires that all occurrences of components/entities have already been identified and are available **as input**

for the model. Thus the input contains a requirement pair but also a single phrase in each requirement that will be compared. The model then only needs to consider these two phrases rather than identifying the correct phrase as in CCR. They consider the context of a phrase to be the requirement that contains the phrase. Each **requirement** pair is fed through a BERT model, left side of the inner box Figure 8 labelled as **context**[11], to obtain a vector representation of each requirement. They extract the vector representation for the each of the two input requirements from the CLS token. The CLS token is a special token in the BERT architecture and can be used for sentence level embedding. They do not motivate their decision to use the CLS token despite plenty of evidence that average pooling outperforms the CLS vector as semantic representation, [RG19; Cho+21].

The **phrases** here labelled **entities** themselves get forwarded through a word2vec model as depicted in Figure 8 right side of the box. The goal is that the vector from the BERT model provides semantic information and the vector from the word2vec model can focus more on syntactic features. Then the a simple fully connected layer extracts the correct informations from these four vectors to perform a binary classification into coreferent or not. Since we have to provide phrase annotations to every requirement pair we also have to manually annotate requirement pairs that do not contain coreferences at all. Since a word2vec model rather than just the embedding matrix, subsection 3.2, is used they are actually able to fine-tune both BERT and word2vec with the classification task. To reduce training time and more importantly the number of required training samples they initialize both models with publicly available weights. They obtain 1853 samples with roughly halve containing coreferences. They achieve above 90% precision and recall. However, since they consider trivial coreferences even a simple Levenshtein distance achieves above 80% precision on their dataset. This makes it even more difficult to compare their results.

Their DeepCoref achieves seemingly impressive results but their implementation relies on pre annotated inputs. This is a luxury most requirement specifications can not provide. Thus in order to use this approach, it is necessary to manually extract high-quality entities first even after training is complete. As they use a supervised approach it also requires the training dataset to be further annotated with coreference flag for the already provided phrase pair. The manual construction of such a dataset is very labor-intensive. Their presented approach requires us to forward every requirement pair through the entire model to compare them. As such, the required model inference scale quadratically with the number of requirements to be tested. Unfortunately [Wan+20] could not publish

[11] Technically they use a window of words around the phrase including the phrase itself as the context rather than the whole requirement but conceptually it makes no difference.

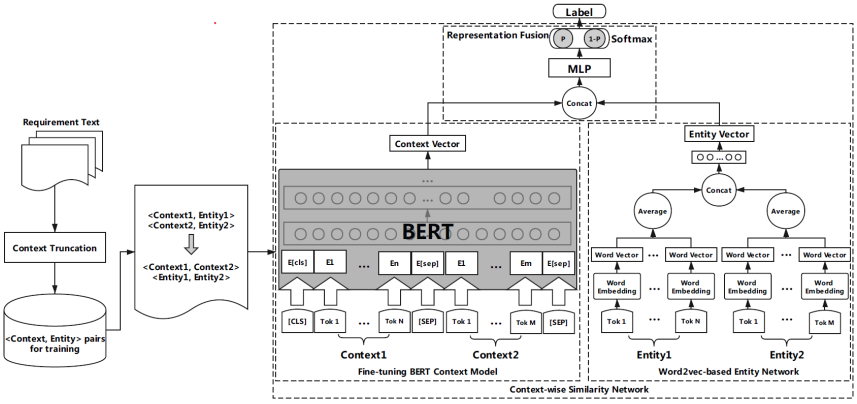


Figure 8: DeepCoref model architecture. The context refers to a n-window around the target phrase including the target phrase itself. So context1 refers to a part around the target word in the first requirement. Context2 then refers to a part of the second requirement of a pair. Entity 1 and 2 are manually identified component phrases that are provided to the model as input. Graphic from [Wan+20]

their dataset. To summarize our findings:

> The number of model inferences scales quadratically with the number of requirements in supervised CCR models. (8)

> Manually annotated inputs will require the same number of manual operations as model inference and should therefore be avoided. (9)

3.6.2 Puber and Fiber

[Das+21] explore similarity and duplicate detection in requirement engineering. They experiment with two versions of the BERT model. The first version they train from random initialization on the entire PURE dataset. They refer to this version as Puber[12]. To generate sentence embeddings they extract the vector of the BERT-specific CLS token.

Their second BERT variation is also trained on PURE but they initialize the weights with a public pre-trained BERT model. They adopt the the BERT

[12]Puber: **PU**RE and B**er**t

model in an unsupervised manner. For now it is sufficient to note that unsupervised domain adoption does not require labelled data for training. They refer to the domain adoption as non-task-specific fine-training and named this second approach Fiber[13]. As with any unsupervised approach, Finding 3, they need to extract and compare information manually. Instead of using the CLS token as with Puber, they test average and max pooling strategy to extract a sentence emebedding with Fiber. This is unfortunate as we can not verify that the performance difference between Puber and Fiber is due to domain adoption or pooling. To compare two sentences they use the cosine similarity.

To evaluate their approach they build a **test** set made up of 800 sentence pairs that are each labeled as either similar or dissimilar. They extracted the requirement pairs from the PURE [FSG18] dataset analog to STS datasets. Fiber showed significant performance lead to Puber that have not been fine-tuned to the PURE dataset. This is Unsurprising as the small Pure dataset with just under 40 000 requirements is not enough to adequately train BERT from scratch in case of Puber. More interesting for us is that they compare their two models to publically availible models. Some of these additional model, such as RoBerta[Liu+19], are extensions to BERT that generally outperform the base version of BERT used in both Puber and Fiber. Another model Universal Sentence Encoder (USE)[Cer+18a] in their comparison is a supervised approach that has been trained on general sentence similarity datasets. In the general setting both of these models outperform BERT on sentence similarity tasks. Despite these advantages Fiber outperformed both models significantly. We take these results as a indication that unsupervised domain is more important than a more sophisticated model or supervised approach without domain adoption. Thus our main takeaway from this approach is:

> Unsupervised domain adoption can outperform generalized supervised models. (10)

[13] Fiber: **Fi**ne-training and B**er**t

4 Dataset Construction

In order to evaluate different models on the CCR task, we need to be able to measure the performance of different models on the task. As there is no public dataset available we created our own. To create a test set for CCR we considered three different approaches:

(1) Manually identification of coreferent phrases in existing requirement datasets such as PURE [FSG18]. However, since the effort of identifying coreferences scales quadratically with the number of requirements, this is unfeasible to manually perform on any substantial requirement set. To illustrate this further consider a requirement set with 400 requirements. We need to compare every unique noun phrase with each other. While most requirements have multiple noun phrases some might be reoccurring in the set which reduces the number of necessary comparisons. If we assume that each requirement has one noun phrase that actually needs to be compared we need to perform 160 000 manual comparisons. Further, it seems that the PURE datasets have been analyzed and corrected prior to release. It thus might actually only contain a handful of coreferences. However, we can only know for sure after the manual identification step. The large number of manual comparisons required also give rise to human error which in turn invalidates the dataset as a benchmark. Using a particular semantic similarity method to prune the necessary comparison introduces a strong bias in favor of that particular method. Even conservative pruning is likely to also remove some actual coreferent pairs. This means that we do not know how many coreferences are actually present in the final dataset. Thus we can not compute common benchmark metrics such as precision or recall.

(2) Construct CCR dataset from existing dataset by manually writing additional requirements that contain coreferences to the source dataset. We pick a random requirement from the source dataset and then select a noun phrase from the requirement. For that noun phrase we come up with a coreferent and write a new requirement that contains the coreferent. During the construction of the second requirement, we also ensure that no other accidental coreferences to the first requirement exists. This will give us a requirement pair that contains one and only one coreference. We also construct requirement pairs that specifically do not contain coreferences. This allows us to treat part of the evaluation as a binary classification. The other part of the evaluation is the identification of the coreferent phrases.

The construction of additional requirements and especially the deliberate search

for coreferences is different from the natural occurrence of coreferences. As such it is reasonable to expect that this approach will also lead to some bias.

(3) Create an entirely new dataset that is authored by several people to naturally give rise to coreferences. The participants would be given a brief description of a fictional project and then be tasked to write down requirements for it. Unlike approach (1) we expect the different authors to introduce coreferences at a higher rate as no terminology has been established. Identical to (1) the effort of detecting coreferences after the requirement acquisition is not feasible for this thesis. Even if we assume higher coreference rates and only collect half as many requirements as in (1) we still need to evaluate at least 40 000 noun phrases. Like (1) this would alleviate the bias compared to (2) as coreferences are not actively searched but occur naturally.

Due to the effort required for (1) and (3) we choose approach (2) to construct the CCR dataset. Further, we believe that (2) offers more robust evaluation metrics. To mitigate the bias we identified for (2) we focus on exchanging noun phrases by shifting formulation to the perspective of a different stakeholder. If a noun phrase is very technical we try to find a more casual coreference. We found the following perspectives useful: technical programmer, casual user, business-oriented accountant. Using the perspective shift as a guideline introduces another bias but it should give us coreferences that are relevant to RE. [DSL18] showed that different stakeholder perspectives are in fact one source of coreferences. But since in our case one author merely impersonates the different stakeholders the results are likely still to some extent different to natural coreferences. Creating requirement pairs is also analog to the dominant semantic similarity benchmark see subsection 3.5 that defines sentence pairs.

4.1 Guidelines during dataset construction

For the construction of the dataset, we choose the *Clarus* project which was published as part of the PURE [Fan+18] requirement collection. The *Clarus* project is specified in a 99 page long document which devotes the majority of its content to defining 273 actual requirements. The remaining content gives a more general idea of the project and scope. Interesting for us is that they also defined a glossary that defines many of the terms that we constructed coreferences for. As such we also extracted the glossary separately as well as the free text from the pdf. In total, we provide three files as part of our dataset.

- **clarus_req_pairs.csv** We use the $ as a row separator as this symbol is

not used in the original pdf. A short sample of the content is given in Table 1 and the file contains five columns:

  - **req1** Contains all 273 original clarus requirements
  - **req2** Contains the constructed requirements. As we have not constructed a partner requirement for every req1 some rows are empty.
  - **type** Is set to CR if requirement pair contains a coreference; N if the pair contains no coreference; or empty if req2 is not set
  - **phrase1** contains the phrase from req1 that was reformulated for req2. Only set for requirement pairs of type CR.
  - **phrase2** contains the reformulated phrase that is coreferent with phrase1. Only set for requirement pairs of type CR.

- **clarus_free_text.txt** which contains the minimally processed text from the pdf including requirements and glossar.
- **clarus_requirements.txt** which contains the all requirements from the Clarus project as well as the constructed requirements.
- **pure_requirements.txt** which contains the all requirements from the PURE requirements collection.

For the construction of the dataset, we split the 273 requirements among 6 participants. This likely gives a greater variety of coreferences as they are constructed by different authors. To ensure that the participants are familiar with the *Clarus* project they were given a 5-minute presentation. Each participant then received a concise summary of the *Clarus* project along with guidelines for the coreference construction in written form. The complete instructions can be found in Appendix B.

During the construction, we focused on creating coreferences that are plausible within the *Clarus* project. We also limit the number of coreferences to one per pair. Some components are used repeatedly throughout the project. Once we can not come up with new unique coreferences we skip requirements that do not contain any other components. If we would allow some coreference pairs to occur more frequently than others our model might be biased towards that pair. By not repeating coreference pairs, we prevent one source of evaluation bias. The participants have also been asked to generate entirely new requirements rather than just exchanging a phrase in the original requirement. Due to the rigid structure of requirements many requirements pairs have a strong overlap despite these efforts. However we

Req1	Req2	Label	Phrase1	Phrase2
The Clarus system shall manage environmental data and metadata according to the Clarus data sharing agreements.	The Clarus system shall show users the data exchange conditions	CR	data sharing agreement	data exchange conditions
The Clarus system shall implement quality checking processes as soon as data become available.	The Clarus system shall implement a manual overwrite option for historical location data.	N		

Table 1: This is a small sample from the CCR dataset. The first column req1 (Original requirement) consists of the unaltered requirements from the public Clarus project [Fan+18]. The second column, req2, contains the requirements that have been manually constructed by our team. Each sentence pair is labeled to (CR) contain a coreference or (N) not to contain a coreference. The remaining two columns record the coreferent phrases for all sentence pairs labeled as CR.

Out of the 273 requirements we constructed a second requirement for 205 of the original requirements. We then reviewed the requirement pairs and removed any that violated the conditions set above. This left us with 152 requirement pairs out of which 72 contain a coreference and 80 do not. The 72 requirement pairs contain 144 coreferent phrases.

4.2 Evaluation

We want to check if the model picked the correct phrase in a sentence. In other words, we want to compare the predicted coreference pair from the model with the coreference pair recorded in the dataset. This presents a challenge that makes it difficult to apply conventional evaluation metrics. We want the comparison to be lenient towards predicted phrases that are a bit longer or shorter than the target phrase. Consider for example the second phrase *data exchange conditions* from Table 1. If the model chooses a slightly longer phrase such as ***the*** *data exchange conditions* we should still consider it a correct match. This disqualifies direct and fuzzy string comparisons such as traditional Levenshtein similarity [Lev66] or more modern methods like Mongue-Elkan [Jim+09a]. This

further prevents us from considering the evaluation as a multi-label classification problem and using cross-entropy. Instead, we compare character index range of the prediction with the character index range of the target phrase. We check for a overlap between predicted and goal phrase. We are aware that this method can generate false positives if the predicted phrase captures most or even the whole input sentence. This can be mitigated to some extend by disqualifying predicted phrases also based on their length relative to the target phrase and complete sentence. However in practice we have found this approach to work well since our phrase candidates do not span over the entire requirement.

5 Component Coreference Resolution Pipeline

We originally defined the CCR task in subsection 2.6. We will now describe our proposed pipeline resolve component coreference. As a reminder we take as input a sentences pair and output a coreference pair if the sentence pair contains any or none if they do not:

$$\textbf{Input:} \quad \text{Sentence pair}$$

$$\textbf{Output:} \quad \begin{cases} \text{Coreferent pair} & \text{if Input contains coreference} \\ \text{Empty} & \text{else} \end{cases}$$

We propose a simple end-to-end CCR pipeline that works with any Vector Space Model (VSM), Figure 9. By end-to-end we refer to the fact that our model does not require any inputs besides the sentences themselves. As we work on project requirements we will refer to the inputs as requirements rather than as sentences. The output will provide us with the phrase pair from the first and second requirement that is coreferent. Our pipeline will provide a single coreferent pair per input sentence. Unless the input sentences do not have any coreferent phrases the system should return no phrase pair. The first step in the pipeline is noun chunking which performs the coreference candidate selection. In other words, we select phrases that could potentially contain coreferences. As established in Finding 1 any terminology, independent of domain, makes heavy use of nouns. We use this insight to create grammatical patterns which ensure that each candidate phrase contains a noun, subsection 5.1. In the next step, we compute a vector representation of each phrase using a VSM subsection 5.2. One challenge here is that some Vector Space Models (VSMs) offers multiple possible vector embeddings with no clear evidence of which one is superior. This means we need to determine which embedding type is best suited for our CCR application. All considered VSM perform the vector embedding on token level. The number of tokens and by extension the number of vectors in a phrase embedding thus varies with the length of the phrase. However, to compare two phrase embeddings, the dimensionality of the vectors must be the same. This is achieved by a pooling operation that summarizes the different token embeddings into a single phrase embedding. Finally, we apply a vector metric to the pooled vectors that represent our phrases.

All phrases from two requirements will be compared in a pair-wise manner. This will result in a matrix of similarity scores. The columns represent the candidate phrases from the first sentence and the rows the candidate phrases from the other sentence. In the example Figure 9, *collected dataset* and *data samples* are the most similar phrases with a score of 0.9. However, since most sentence

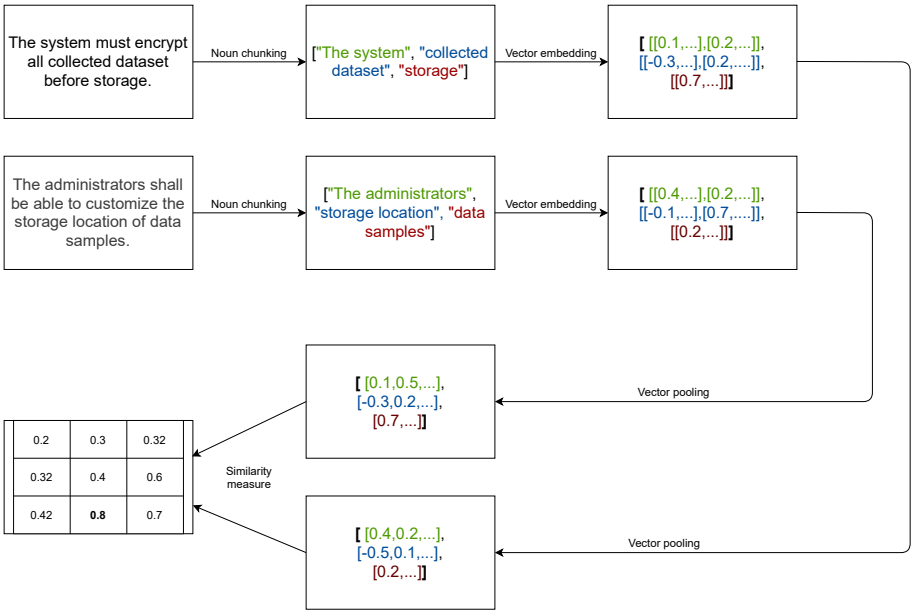


Figure 9: Our proposed end-to-end CCR pipeline. Each sentence is separately forwarded through until we obtain their phrase embeddings. These can then be compared to the phrases of another requirement.

pairs do not contain coreferences we can not just return the highest score in the matrix as a coreference. Instead, we need to define a threshold value for the similarity score. If none of the phrase pairs of two sentences have a similarity score above the threshold then we will consider the sentence pair as coreference free. If the score is above the threshold we can not only classify the sentence pair as coreference containing but also report the exact phrase pair. Choosing the best threshold depends on the model used but also on the final use case for CCR. Ideally one would like to achieve both high precision as well as high recall. In practice there is often a trade-off. A low threshold will result in a high recall but lower precision and a higher threshold in a high precision but lower recall. Thus if we have two models that have the same or very similar maximal recall/precision/F1 score we prefer the model that allows us to make this trade-off. A model where threshold vs precision is correlated.

5.1 Noun chunking and candidate selection

The goal of the candidate selection is to pick phrases from an input sentence that could be a project component. As seen in The goal of the candidate selection is

to pick phrases from an input sentence that could be a project component. As seen in subsubsection 2.6.1 we expect our project component terms to follow the same grammatical patterns as terms in established fields such as mathematics. As noted in Finding 1[page 15] we have noted that a vast majority (92.5%-99%) of these terms contain at least one noun. Thus we use noun chunking for the candidate selection. A noun chunk is a noun along with the surrounding words that further describe or define that noun. For example *autonomous cars* has *cars* as its root noun and then *autonomous* is referring to the root noun and should therefore be captured as part of the noun chunk. In order to select noun chunks we need to (1) have reliable grammatical annotation algorithms and (2) identify a grammatical pattern that captures the noun chunks we are interested in. Fortunately, for (1) we can rely on *spacy*[14], a standard industry framework for Natural Language Processing (NLP), to provide us with high-quality grammatical annotation. *Spacy* assigns each token, here each word, in a sentence a Part of Speech (POS) tag. Note that the tokenization during the POS tagging is completely independent of the tokenization performed by the Vector Space Model (VSM). While the tokenization of POS tagging assigns every word a separate token this is not the case for most VSM. The POS tags include nouns(N), adverbs(A), verb(V) or punctuation(P) for each word. A full list of all available tags can be found here[15]. To make pattern matching easier we convert a sentence into a list of abbreviated POS tags, as sketched in Listing 1 and applied in Listing 3 .

```
def map_sentence_to_pos_string(sentences):
    mapping=""
    for token in sentence:
        if token.pos is Punctuation and token is "–":
            mapping.append(B)
        elif token.pos is Verb and token.suffix is "ing":
            mapping.append(D)
        elif token is "and|for":
            mapping.append(J)
        elif token.pos is Noun:
            mapping.append(N)
        elif token.pos is Adjective:
            mapping.append(A)
        elif token.pos is Verb:
            mapping.append(V)
        else:
            mapping.append(X)
    return mapping
```

[14] https://spacy.io/
[15] https://github.com/explosion/spaCy/blob/master/spacy/glossary.py

Listing 1: Our approach of mapping and grouping POS tags to a simple string. Each character in the string represents the POS tag category of one token of the input sentence. The POS tags are calculated using *spacy*.

We noticed that some of the coreference entries in the dataset require information beyond usual POS tags. For example the phrase *environment-related measurement data* contains a punctuation *"-"* which will be considered its own token with the POS tag for punctuation. We want candidates to include punctuation's that bind words together but not punctuation's that separate words from each other such as *"."* We address this with an extra rule in Listing 1[line 4-5]. Similarly, we address verbs that are used to further describe a noun in lines 6-7.

```
1
2 >>>sentence="The Clarus system shall implement
3     quality checking processes as soon
4     as data become available."
5 >>>print(map_sentence_to_pos_string(sentence))
6 NNNXVNNNXXPNVAX
```

Listing 2: We can take a requirement and map it to its pos tag string.

We can then use traditional regular expressions to detect grammatical patterns. We defined the first pattern to match basic noun phrases, **??**[line 2]. This expression finds phrases with a root noun at the end and a number of possible POS tag combinations before that. The second pattern captures longer phrases that are joint together with an *and* such as *maintenance and construction vehicles*.

```
1 pos=map_sentence_to_pos_string(sentences)
2 short_matches=pos.regex(r"(D|A|NPP?|N|(P?B.)|NPX)*N",find_all=True)
3 long_matches=pos.regex(r"(NP|V|A)?(NJ)+N+",find_all=True)
```

Listing 3: We can take a requirement and map it to its pos tag string.

The matches from both regex queries then form the set of candidate coreference terms.

5.2 Vector embeddings and pooling

We already identified phrase candidates in the previous step, subsection 5.1. Now we want to map these phrases to a semantic vector space. For this mapping, a number of neural networks have been proposed and we presented the

general ideas in section 3. We will treat their exact inner workings and training procedures as a black box for this thesis. However, we will look at neural network structures that is relevant for this thesis. In the following, we will discuss which choices need to be made to extract a vector representation from a multi-layer network such as BERT[Dev+18]. We will consider each of these choices as a **parameter** for a model configuration. We will test all sensible permutations of model configurations on our dataset (section 4. A summary of all parameters along with the values we tested for each parameter can be found in subsubsection 5.2.6.

5.2.1 Models

We choose three different vector embedding methods to test on our CCR. First, we choose a shallow model with a single hidden layer that is akin to the word2vec model, subsection 3.2. Despite numerous proposed extensions the traditional word2vec model performs surprisingly well on semantic similarity benchmarks according to [PSP18]. However, most requirements in our test set contain domain-specific terms that are not present in the texts during the pretraining phase of the public model, see Out-of-Vocabulary in subsection 3.2. Training word2vec on our requirement sets would introduce the missing words to the word2vec vocabulary, however, we believe that the number of occurrences of these terms is too low to learn an accurate word embedding. Instead, we choose a model that can handle unknown words. We decided to test the **FastText** embedding, Figure 3.2 , due to its strong semantic performance among single layer unsupervised approaches, [PSP18].

As explained in subsection 3.4 the transformer architecture was a major breakthrough in natural language processing. This architecture was popularized by the BERT implementation that proposed the masked language modeling to train a stacked transformer architecture in an unsupervised manner. At the time of publication, the model managed to establish new state-of-the-art results in a number of benchmarks. Most interesting for us, it also set a new record in the STS (semantic similarity, subsection 3.5) benchmark [Cer+17] with a respectable 5% gap to the next runner up. To the best of our knowledge, any model that managed to improve on BERTs result is based on the same transformer architecture as BERT. Many not only use the same general architecture but even inherit the transformer layout and with it the bidirectional nature of BERT. One such model is the MP-NET[Son+20] developed and trained by Microsoft. They improved a bias in the random sampling of the Masked Language Modeling (MLM) and it is at the time of writing the best model on the

semantic similarity benchmark[16]. As such we choose **BERT** as our second and **MP-NET** as the third model to test.

5.2.2 Training dataset

In order to adapt the embedding models to our domain, we *fine-train* each model. We will give a rough idea of *fine-**tuning*** in contrast to *fine-**training*** later in subsubsection 5.2.3. Fine-tuning is usually performed directly on the task we are trying to solve. Since we do not have enough data to set up a supervised fine-tuning task we need a proxy task that we can use for fine-tuning. We choose to use the same unsupervised task that the public pre-training was performed with. For clarity we refer to the case where we adopt the model with the same task as used during pre-training as *fine-training* rather than *fine-tuning* from here on as proposed by [Das+21]. For fine-training we start with a publicly available model and then train it with our dataset. By fine-training, only on our datasets, we do introduce a bias to the training procedure. This bias is intentional and represents the adoption to our domain. We will only investigate if our models can benefit from this kind of fine-training and not concern ourselves with generalisation and transfer learning.

We tested three different subsets of the PURE requirement dataset collection as training set for this domain adoption. Note that our test set is also based on one of the projects in this PURE collection, the Clarus project. Our first training set will contain the requirements extracted from the same Clarus project and the requirements that we manually constructed for the test set. This will probably set off alarm sirens in the reader's head for using the same data for training and testing. However, in this instance, we actually can use the test set requirements also for training. We are training only on a proxy task and we are not using the coreference information from the test set. Thus, we are not actually providing any of the CCR specific values to the system. In other words, we are only using the input of the CCR task for training. This training set is the most task-specific as it contains the same requirements we will use for testing and nothing more. As such it is also the smallest set and will be referred to as CLarus Requirements **(CLR)**. Next, we extend this dataset by also including the raw text that is part of the Clarus project specification document. This provides context information of the project that is not available in the requirements themselves. As such this allows us to investigate if any of these models can make use of this additional information. Perhaps even more interesting this dataset is essentially a pdf to text dump with minimal processing. If this performs comparably to the other

[16] https://www.sbert.net/docs/pretrained_models.html

sets it allows us to fine-train our model on any requirement specification without the need for labor-intensive requirement extraction. We refer to this subset as Clarus free-text**(CLF)**. Finally, the PURE requirements **(PR)** training set consists of the requirements from all projects in PURE collection but does not include any specification texts as in CLF. This will allow us to see if additional requirement samples lead to better understanding in form of precision.

We will not change the default training parameters of each model. We believe the training parameters have been sufficiently explored by transformer publications and the huggingface[17] community to provide a solid parameter set for most use cases.

5.2.3 Vector pooling

Now we have three different models and even training data to fine-train them. In subsection 3.2, we have seen how to extract token embeddings from models with a single hidden layer. We essentially used the weights from the one hidden layer as the token embedding. Both BERT and MP-NET use 12 stacked transformer layer and are not designed to directly provide a vector embedding. Rather they are deep neural networks that have been trained on basically a *fill in the blank* task, more formally know as Masked Language Modeling (MLM), subsection 3.3. These networks are given billions of sentences with a few tokens randomly masked/removed. The networks then need to fill these gaps by predicting a token that fits in the gap. In order to predict the correct word/tokens for a gap these models must learn both syntactic and semantic information from the training sentences. Thus the information we want in our word embeddings must be present in the weights of the hidden layers. The question is how to best extract that information. Traditionally one does not need to extract them at all. Instead the trained hidden layers are used as a base and then a small neural network according to the desired downstream task is added. In our case we could add a network with a single output which would predict if two input phrases are coreferent, this is analog to the models described in subsubsection 3.6.1. This would train the model in a supervised manner to pass relevant information down through all layers to the added downstream specific network. As such the manual extraction of word vectors is eliminated. This is referred to as *fine-tuning* as the weights of the original model are tuned to best serve the new task. While *fine-tuning* requires significantly fewer training samples than training a network from scratch it would probably still require at least one thousand training samples to achieve acceptable results. This type of

[17]Hugging face is a project that aims to make all kind of transformer model easy to use, compare and share.

fine-tuning would also learn how to compare two phrases for us, eliminating the need for a similarity metric. If we had even more samples we could also offload the candidate selection step to the neural network. Besides the fact that we do not have enough data for these supervised methods, the bigger issue would likely be performance. While supervised models compare phrases for us, it also means that the only way to compare two phrases is to forward them through these models. As noted in Finding 8[page 34] this lets model inferences grow quadratically with number of inputs. In our proposed pipeline, we only need to forward each phrase once through the network to get its vector representation. We can then do the comparisons between phrases by only using the vectors. As a result in our pipeline the number of network inferences grows linearly with the number of phrases instead quadratically.

Layer Pooling

This still leaves the question: If we do not let a downstream task decide what is relevant, how do we extract meaningful vectors from the twelve hidden layers? We could decide to just pick one layer to use for our token embedding. So for example we could always take the weights of the last layer and use that like the vector embedding from a model with just one layer. Instead of choosing the output of a single layer we can also combine the output of multiple layers, Figure 10. This is in principle the same process as for the phrase pooling in subsection 3.5. However, unlike the length of a phrase the number of layers we use will remain constant independent of the input. This allows us to also concatenate the layer vectors into one very large vector as the resulting vector will be the same size for every input. This would not work for phrase pooling as it would result in variable vector sizes depending on the number of tokens in a phrase. [Dev+18] found that layers closer to the input contain more syntactical information and the layers closer to the output contain more semantical and conceptual information. We will therefore focus our testing on the last layers. We will test three different layer selections choosing the: 1) **last**, 2) **last four** or 3) **all layers**. For **Phrase pooling** we will test 1) **mean** and 2) **max** pooling. Besides phrase and layer pooling there is a third pooling operation we need to consider, **subword token pooling**. In order to avoid out of vocabulary words BERT and its derivatives split words into multiple tokens. As such a single word may be represented by multiple token. Subword pooling decides how or if the different tokens of a word should be combined before layer pooling, Figure 10. [ÁKK21] showed that the choice of subword pooling is not only relevant but also task dependent. For subword pooling we test the some of pooling strategies presented in [ÁKK21] namely to select the 1) **first-token**, 2) **last-token**, 3)

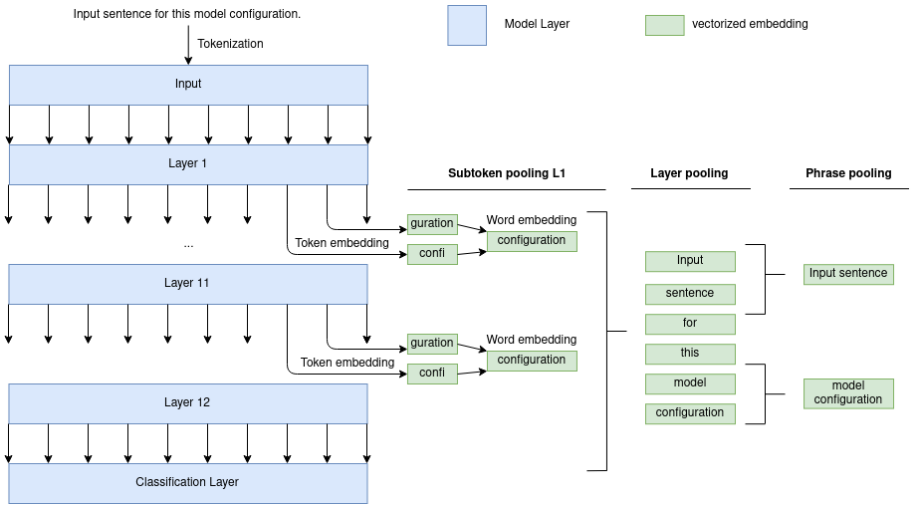


Figure 10: Different pooling stages for transformer architecture.

mean of tokens (mean-pooling) and finally add 4) max pooling. As noted in Finding 7[page 31] in order to apply set-similarities max-pooling is a good match for the set-similarity metric, subsubsection 5.2.4, we want to test.

In summary phrase pooling is necessary to get uniformly sized phrase vectors. The other two pooling stages (Subword and Layer) allow us to select which part of the network we want to use. This choice depends on the underlining task and optimization goals. These three pooling stages allow us to extract a phrase embedding from any deep language model.

Though using average layer pooling seems to be the most popular approach by far. Even the original BERT paper mentions that layer concatenation outperforms average pooling on average. Then [Zhe+19] advocates for max pooling instead by also introducing a new vector comparison that only works on max pooled vectors. Even more recently [ÁKK21] introduced subword pooling and showed that it makes a significant difference. Further they showed that the best subword pooling strategy for one task (e.g. Morphological task) can actually perform worse than no subword pooling in another task (e.g. POS tagging). We think it is beneficial to this project to test the different parameters described above.

5.2.4 Vector similarity

We are left with one vector representation for every coreference candidate. All vectors are of equal size. As noted in Finding 6[page 31] we can compare these vectors with each other to gai some word-similarity measure. As explained in subsection 3.5 cosine similarity is generally preferred over euclidean distance. Further, we will also test the two set similarities Jaccard index and the fuzzy set similarity DynaMax. These set methods mandate max-pooling during the phrase pooling and as such reduce the number of necessary configurations slightly. In summary we will test **cosine**, **jaccard** and **DynaMax** similarities.

5.2.5 Contextualization

Finally we need to decide whether we want to provide the network with context around the input phrases or not, see subsection 3.3. We can either forward only the actual phrase candidate through the network and the output will only contain token embeddings of that phrase. Or we can forward the whole sentence through the network and only pick the tokens from the output that correspond to the phrase we are interested in. The second approach makes use of the contextualization of BERT or MP-NET to differentiate between word sense of polysemus words forms. Recall the example from subsubsection 2.4.2:

> *to execute*
> He executed the diagnostic tools.
> The regime executed thousands of innocents.

If we forward each sentence through a BERT model we should get different vectors for the word form *executed* in the two setences. However if we only forward *executed* through the network without any context we would of course get the same vector twice. From this example it may seem like adding context information is always beneficial. However, in these models the context information will also shift vectors of monosemous words. Further contextualisation might lead to better semantic representation while removing syntactic informations. As such we test both contextualized and non-contextualized configurations.

5.2.6 Configuration parameter

While word embeddings are firmly established in the NLP domain word embeddings from transformer models is still under investigation. So far it seems like there is not one configuration that generally performs the best. Further we hope

to strengthen the evidence in favor of set-similarities, Finding 3.5 and subword-tokenization, subsubsection 5.2.3, to lend more credibly to their advances. In total this leaves us with the following variables to test on our CCR dataset:

- Model
  - [**fasttext**] FastText
  - [**bert-base**] Bert
  - [**mpnet-base**] MpNet
- Training dataset
  - [**clr**] Clarus requirements.
  - [**clf**] Clarus freetext.
  - [**pr**] Pure requirements.
- [**U**] Training duration in batch update steps
- [**L**] Layers to be combined
  - [**all**] considers all layers
  - [**-4**] considers the last four layers
  - [**-1**] only considers the last layer
- [**P**] Pooling (Layer pooling)
  - [**avg**] averages vectors
  - [**max**] takes the max value in each dimension out of all vectors
  - [**con**] concatenates the layer vectors together
- [**ST**] Subword pooling
  - [**avg**] averages vectors
  - [**max**] takes the max value in each dimension out of all vectors
  - [**first**] uses only the first token of each word
  - [**last**] uses only the last token of each word
- [**S**] Similarity metric
  - [**cos**] averages vectors, implies average phrase pooling
  - [**dyn**] DynaMax similarity as proposed by [], implies max phrase pooling
  - [**jac**] jaccard similarity for fuzzy sets, implies max phrase pooling
- [**C**] Contextualization
  - [**True**] makes use of contextualization.
  - [**False**] ignores context.

For the remainder of the thesis we will use an abbreviated notation to express configurations. The abbreviations for each parameter can be found in square brackets withing the above table. For example:

bert-base_clr_U48_Lall_Pcon_Sjac_STfirst_CTrue

would refer to the bert-base model trained on the Clarus requirements for 48 batch updates using contextualized word embeddings from layer concatenation

on all layers, first-token subword pooling and max phrase pooling since it uses the jaccard fuzzy set metric. Each configuration can then be evaluated on our test dataset. Note that the FastText model only has a single hidden layer and performs subword pooling implicitly. Thus, the parameter for layer choice, layer pooling and subword token-pooling are all fixed. As such all FastText configurations will contain *Lall,Pmean*.

6 Results

In order to compute recall, precision, and F1 metric we need to define a threshold similarity value. Given a threshold for our similarity score we can determine which phrase pairs are predicted to be coreferent. We can then compare the predicted to the actual coreference pairs recorded in the dataset to determine which are true positives or hits. Once we know the number of true positives we can compute recall, precision and F1 as follows:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{\text{True Positive}}{\text{Actual Positive}}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} = \frac{\text{True Positive}}{\text{Retrieved}}$$

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

The number of actual positives is the number of coreferences in the test set, section 4, in our case 72.

6.1 Noun chunking results

Using the candidate selection described in subsection 5.1 we were able to find 137 of the 144 (95%) coreference phrases of the test set among the candidates. This is in line with the findings of [JK95] we discussed earlier, subsubsection 2.6.1, that found 92.5%-99% of all terms to contain nouns. The 7 missing phrases are part of 5 different requirement pairs which will limit the recall of the overall pipeline to 67/72. This will give the remaining pipeline **a recall ceiling of $\approx$ 0.93**. Out of the 7 missing phrases we were unable to detect the correct descriptors for a noun in 3 cases. For example, our system found *schedule* but the phrase we were interested in was *configured schedule*. In addition, our system can only handle a single join keyword(J) in phrases like *data and information sharing*. However, there were 2 instances where the constructed phrases in the test set extend over multiple joining keywords. The two missing phrases are *guidelines for sharing and using data and information* and *data and information sharing and use policies*. Furthermore we had one instance of an enumeration *latitude, longitude, and elevation* as coreference for *geographic coordinates*. And finally, one typo that managed to pass the review process and was only detected

after all tests had been performed.

For most requirements we extract multiple coreferences candidates. In our pipeline, section 5 Figure 9, we compare all candidates in an pairwise manner. In the test set an average of ≈ 7.9 comparisons are necessary per requirement pair. As such we get a **precision baseline for our pipeline of $\approx$ 0.126** for selecting coreference pairs out of the candidates at random.

6.2 Pipeline results

As mentioned in the beginning of this section we need to decide on a similarity threshold before we can compute recall or precision. The ideal threshold will of course vary between models but also based on the use case. Instead of choosing a specific threshold recall, precision, and f1 are plotted against the threshold, Figure 11. The threshold starts at 0.1 and is increased by 0.05 until the upper bound of 1 is reached. We use these increments to create all metric-threshold graphs in the thesis. The similarity metrics[18] used throughout the different configurations can return values outside of the $[0.1, 1]$ range. However, our pipeline only returns the best match between two requirement pairs we pick only the largest similarity score of on average ≈ 7.9 comparisons. As such in practice no requirement comparison returned values below 0.1. In fact, for most model configurations the minimum similarity score returned is far higher. This leads to the plateau on the left of each of the three graphs in Figure 11. In general, the recall graph shows us how many of the coreferences within the test set we were able to identify. Keep in mind that our candidate selection limits us to 93% recall since we are missing the candidate phrases to even compare the rest. Moving to the precision graph we notice that it stays more or less constant even beyond the initial plateau. This indicates that our similarity score does not really manage to rate how similar two phrases are. No matter how we choose the threshold the proportion of correct to incorrect results stays more or less the same. This further implies that sorting our results by similarity would not show the best matches first. As we approach the right side of the graph the precision graph becomes erratic. This is caused by our rather small dataset. As we increase the threshold we reduce the number of pairs that are found. As the number of pairs is reduced even a single additional correct or incorrect pair will have a stronger and stronger effect on precision. Finally, the F1 graph displays the harmonic mean of the previous two graphs. This makes it easier to identify how the previous two graphs affect each other. Ideally, we would like to achieve a recall and a precision of one. Realistically there is often

[18] The cosine similarity for example returns values in the range $[-1, 1]$.

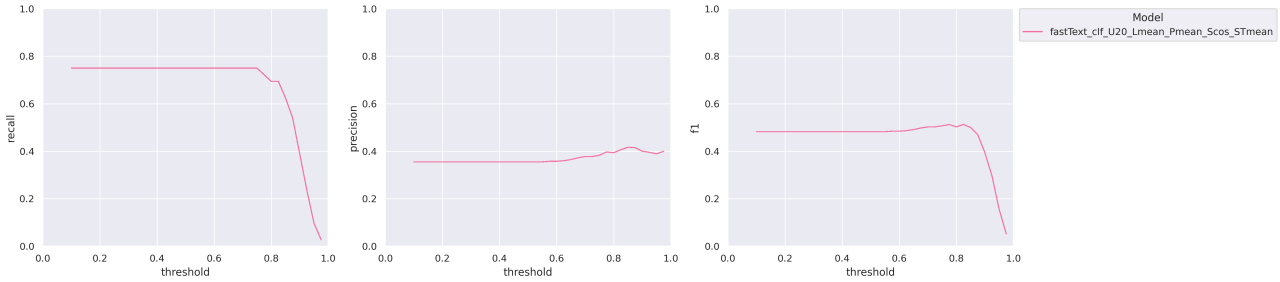


Figure 11: Our evaluation metrics for a single configuration of the FastText modell.

a trade-off between the two. Ignoring the candidate selection, a recall value of 1 could easily be achieved by reporting all possible combinations of phrases. This would of course return mostly useless results and precision close to the precision baseline of 0.126. As such we will always want to optimize for both to some extend and the F1 graph provides us with an indication for this tradeoff.

6.2.1 Recall and F1

Given the large number of parameter we are evaluating we have a total of around 4000 model configurations. As such we need a selection criterion to pick a few configurations at a time that we can evaluate. The selection criterion is our window into the data and will at the end define what we consider to be important. When choosing a selection criterion we have to be careful to only use operations that are threshold shift invariant. For example, when selecting for the maximal recall value in a graph it does not matter at what threshold value the maximum occurred. As such taking the maximum is invariant against shifts of the underlying similarity distribution. Selecting for the best mean value for example would benefit models that have the falloff at a higher threshold.

We will start by selecting for configurations with the largest recall. This will give us an idea of what the best recall that we can achieve with any of the tested models is. Then if we later change the selection criterion we know how much recall we are sacrificing. We find the highest recall overall threshold values for each configuration then select the ten configurations with the highest maximum, Figure 12. We note a max recall just under 0.8. We can see that the correlation between precision and threshold is very weak and in some cases even negative. If we increase the threshold and the precision stays constant it means that whether a phrase pair has a similarity of 0.3 or 0.9 does not affect how likely it is that

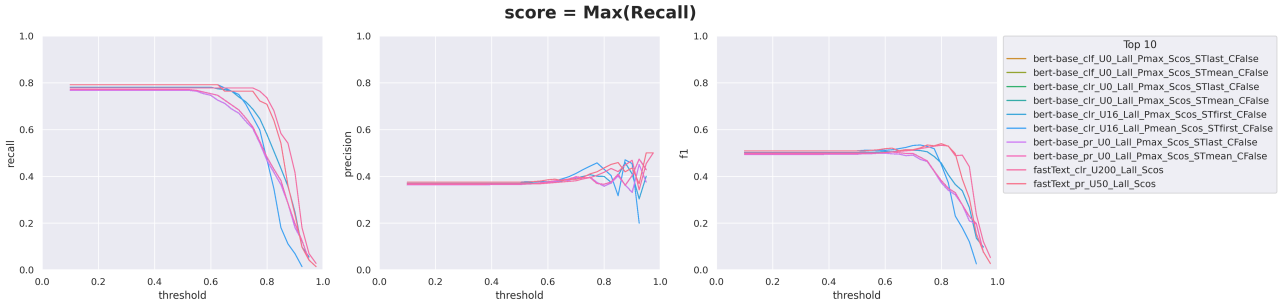


Figure 12

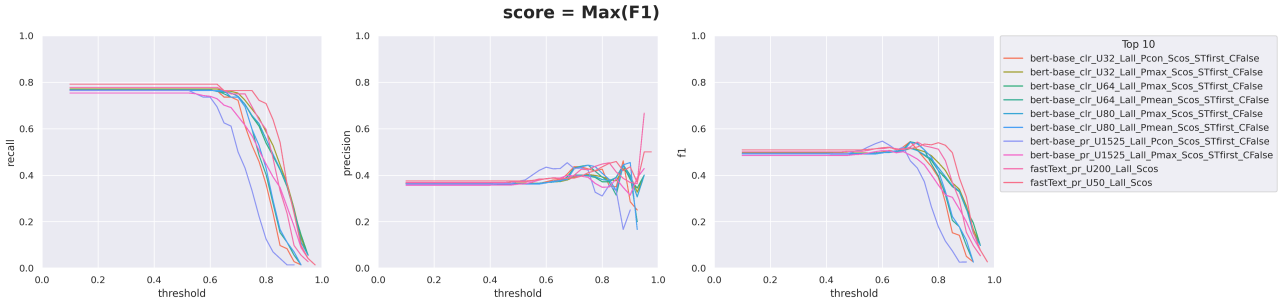
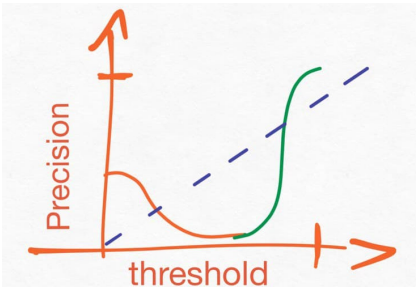


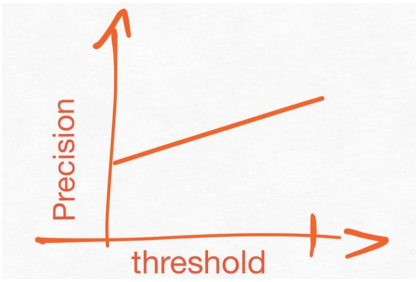
Figure 13

we found a coreference. We further note that all transformer models (BERT) in this top ten make use of all 13 layers. Surprisingly layer concatenation does not seem to provide a significant advantage despite producing vectors 13 times the size of the averaged vectors. FastText shows surprisingly strong performance and achieves the highest total recall of all models. It seems that our task-independent domain fine-training provided little to no benefit to the transformer models in our selection. The transformer models here all use either unmodified weights(U0) or very little training (U16 or about 1/3 epoch). FastText on the other hand seems to benefit from pretraining even when selecting for recall.

When selecting for *max*(F1), Figure 13, most observations remain the same to the previous graph. However, we note slightly higher and wider peaks in the precision graph and subsequently in the F1 graph. By making precision relevant to the performance we notice that all selected models seem to now benefit from our training. This is relevant to differentiating coreferences from random phrase pairs and can be measured by precision.



(a) Pearson correlation of 0.7 and max precision of 1.



(b) Pearson correlation of 1 and max precision of 0.8.

Figure 14: Graph (a) has a moderate correlation overall, despite having a near perfect correlation in the threshold range, we are actually interested in. This can cause graph (b) to have a higher combined score even though we might prefer graph (a).

6.2.2 Precision

So far we have only considered precision indirectly through the F1. Selecting for good precision is not as easy. There are a number of configurations that actually reach a precision of 1. Thus selecting for maximum precision is less useful. We have already analyzed the precision threshold correlation in the previous graphs. And if we want to be able to sort our CCR results by their similarity score we need a positive correlation between precision and threshold. If there is zero or very little correlation raising the threshold is equivalent to removing pairs at random. However, analyzing selected configurations for correlation and selecting configurations based on their correlation are two separate issues. We note that the correlation is also invariant to threshold shifts. Correlation itself does not select for high or low values so we will need to combine it with other criteria like maximal precision. For now, we need to make sure that selecting for correlation is not discarding wanted configurations.

Lets us consider the following example, Figure 14. The graph on the left, Figure 14a, first dips down and then rapidly grows to a precision of 1. Due to its initial dip its pearson correlation is lower lets say at 0.7 even though the correlation in the relevant region, in green, is near perfect. Regardless this gives us a combined score[19] of $\max(\text{precision}) \cdot \text{pear}(\text{precision}) = 0.7$. The graph on the right Figure 14b has a perfect correlation but maximal precision of 0.8 for a combined score of 0.8. As such adding a correlation criterion will slightly dis-

[19] We will consider how to combined criteria later. Here we simply multiply the maximum precision value with correlation for a combined score.

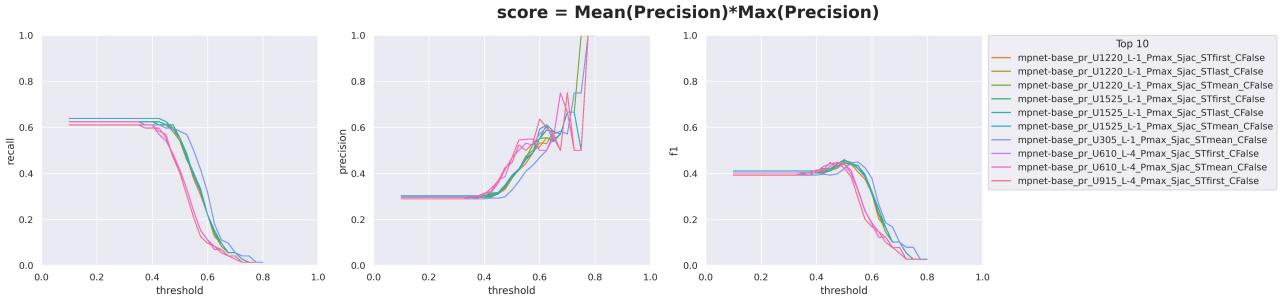


Figure 15

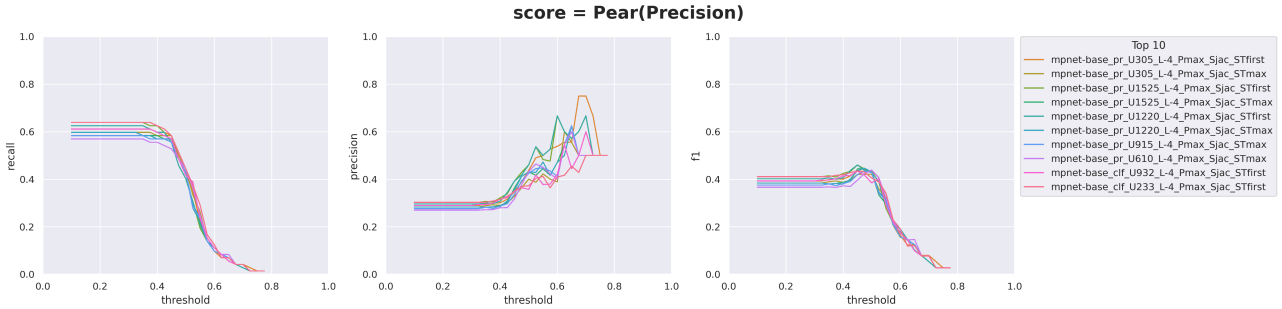


Figure 16

courage the graph on the left. However, this also encourages less erratic graphs that grow linearly. We have not seen any graphs that follow a shape as.

More importantly we are also discouraging, possibly removing, configurations that have low correlation but consistently high precision. As such we should check the highest precision configurations to see how much we are giving up to select to be able to properly sort our results. In Figure 15 we can see that the configurations with the highest precision scores all exhibit strong correlations. We can see that using correlation as the selection criterion will not cut as off from this models. When selecting for the best correlation, Figure 16, we notice very similar distribution in the majority of the precision graph. Though Figure 15 has a slightly stronger slope and rewards configurations that break out to precision of 1. With this we feel confident to use correlation as part of our selection criterion[20].

On closer inspection of Figure 16, we notice that almost all variables that we investigate throughout the different configurations have change compared to the recall/F1 selection in Figure 12. Selecting for precision has not left a single con-

[20]For future evaluations we recommend to use a top-n true positive rate ranking rather than the evaluation performed in this thesis.

figuration in the top ten that makes use of all layers. This is in line with previous observations that found that later layers contain more semantical information. However, the perhaps most exciting change happens for model and similarity choice. Both MP-NET and the Jaccard index were almost non-existent in the previous evaluations and now dominate the field exclusively. This is especially noteworthy as it goes against the almost exclusive use of cosine similarity within NLP literature. With our selection criterion, we were able to find at least ten configurations with a strong correlation in the precision graph. Let us reiterate how crucial a positive correlation between precision and threshold is.

Next, we combine multiple criteria to find those model configurations that both exhibit a precision-threshold correlation and a high F1 score. We could simply take the average criterion, e.g. $\frac{1}{2}(max(\text{F1}) + pear(\text{Precision}))$. This would allow a model configuration to completely ignore one component if the other is sufficiently large. Instead, we decided to multiply the different criteria. Conceptually this considers the area enclosed by both variables and is commonly used to create objective functions over multiple objectives[SD94]. This way we look for models that consider both criteria. Unfortunately, we struggle to find any models that strike a good balance between our selected goals. In this graph Figure 17, we can see that the fastText models are clearly seperated from the other models. Generally, we found the variability to be between different FastText configurations to be less pronounced. As FastText only uses a single hidden layer two of our model parameters are eliminated. Or rather they are implicitly set for us. We assume this to be the source of the reduced variability. While the selected FastText configurations exhibit a slightly positive precision-threshold correlation score the correlation is weaker.

In our test set about half of all requirement pairs contain a coreferent phrase. In an actual requirement set, this ratio is likely to be drastically lower. Further, every requirement will need to be compared to every other requirement. As such the number of comparisons will be significantly larger. This implies that precision will be much lower on a real requirement set. Nonetheless, configurations that perform better on our test set should also perform better on real datasets. However, this does imply that we should put a stronger emphasis on precision when selecting configurations in our test set. The pearson correlation we used before does not select for a high precision only for a strong correlation. This means that we are only relying on the F1 score to also select for high precision values. In order to select for higher precision values we can either multiply $max(\text{Precision})$ or $mean(\text{Precision})$ to our selector expression. As discussed before both options have their drawbacks.

We choose to present the $max(\text{Precision})$ in Figure 18. However, using mean

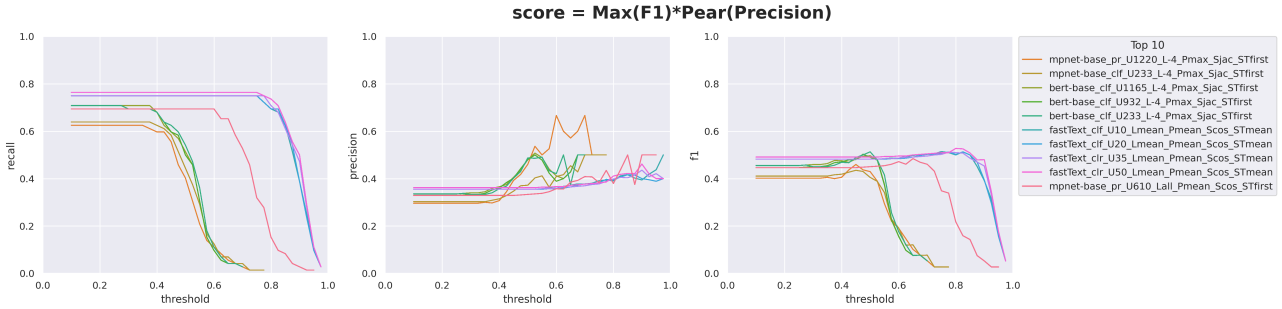


Figure 17

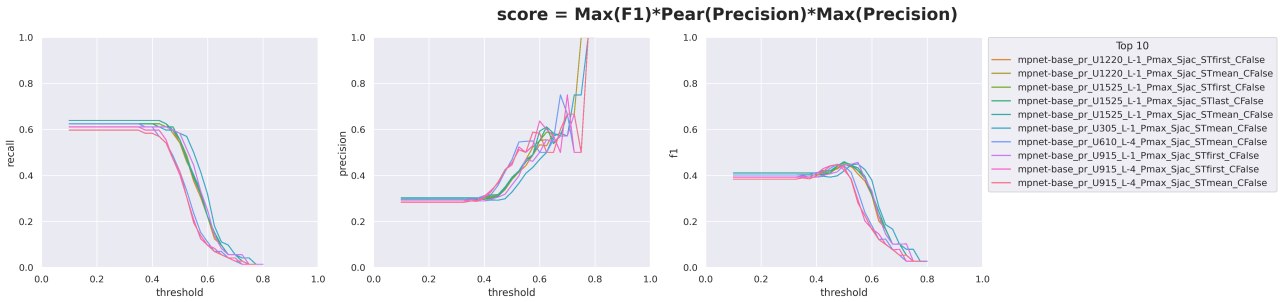


Figure 18

gives a very similar graph. We can see that a stronger emphasis on precision once again selects networks that make use of the last or last four layers and Jaccard similarity. While we lose about 0.1 max F1 compared to the highest F1 in Figure 13 we believe the trade-off is necessary. The selected configurations have a precision of around 0.6 at the peak F1 threshold. As such more than half of the results our pipeline produces are correct. Keep in mind that most requirements have 4 or 5 phrases that have to be compared. A random guess would therefore achieve a precision of $\frac{1}{4^2}$ or $\frac{1}{5^2}$.

6.3 Parameter investigation

We believe that this final selection criterion puts a sufficient focus on precision without completely neglecting recall. We already made some observations based on the top ten results in each selection. We will now further investigate our findings by plotting all configurations in the space spanned by our final selection criterion. To collapse the three components of our selection criterion into two dimension for plotting we consolidate $(max(\text{Precision}) * pear(\text{Precision}))$ into our combined precision measure. This combined precision measure is be plotted

on the y-axis. The x-axis will show the max(F1) and give us a representation of the overall performance of each configuration. Thus the graph will show the precision performance vs overall performance for each configuration. We can then color the configurations based on which parameter they use. We plotted the graph Figure 19 multiple times[21] and just changed which configuration parameter we display.

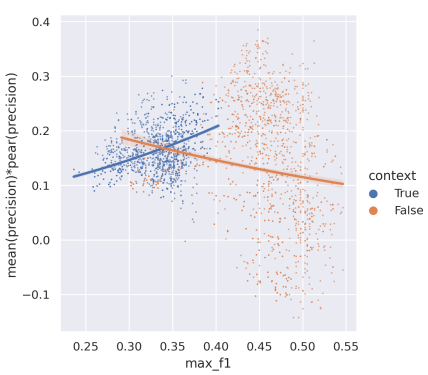
6.3.1 Contextualization

First we investigate the contextualization. For this we color every configuration that uses contextualization blue and color every configurations that isolate the phrase in orange, Figure 19a. The graph shows a strong separation between contextualized and context-free configurations. Contextualized embeddings have a correlation between F1 and precision score. At the same time, the contextualization seems to severely limit the recall performance. We speculate that the contextualization embeddings form more unique vectors that depend less on syntactic features. As we shall see approaches that only rely on syntactic features reach max-F1 scores of 0.41, subsubsection 6.3.7. As such removing syntactic information without a more robust semantic embedding will lead to a worse F1 score. Further, we note that no contextualized configuration has a combined precision score under 0. This suggests that contextualized embeddings provide more consistent semantic embedding with a positive Pearson correlation between precision and threshold. The context-free configurations that only consider a phrase without the requirement around it can clearly separate themselves from the contextualized on the F1 axis. Our other parameters seem to have a stronger influence on the precision performance as we see a stronger variance in the context-free configurations. Despite the negative regression line, we can find a number of configurations that outperform all contextualized configurations in precision while maintaining a significant lead in F1 score.

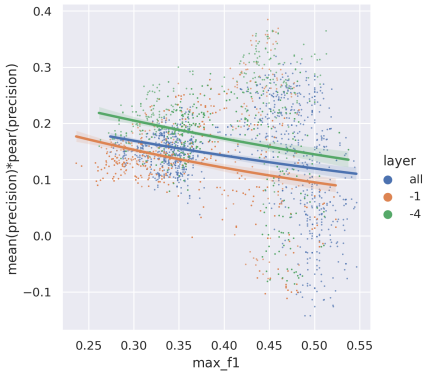
6.3.2 Layer Pooling

Moving to the layer choice, Figure 19b, the plot is far less segregated. The regression lines suggest that using the last four layers leads to the best precision score followed by all layers and finally choosing only the last layer. However, looking at the strongest precision configurations at the top of the graph we notice that both the last as well as the last four layers seem to float above the

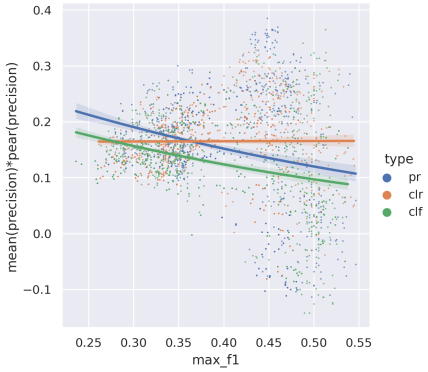
[21] The graphs will vary slightly. For example, when exploring contextualization we will only plot those models that support contextualization.



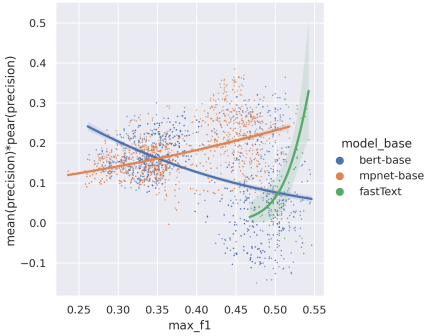
(a) Contextualization



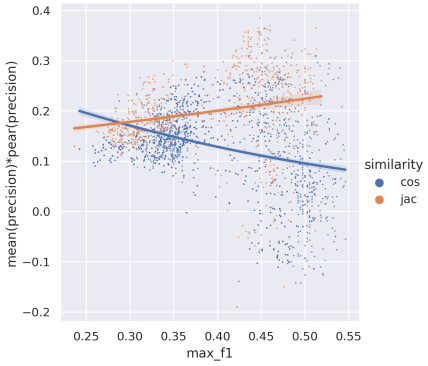
(b) Layer choice



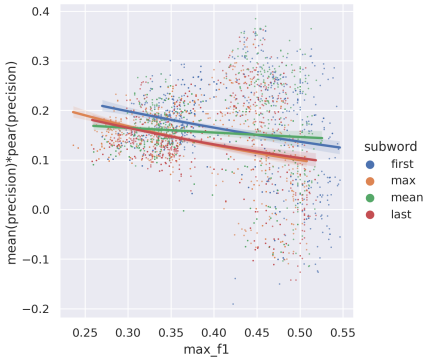
(c) Trainingset



(d) Model architecture



(e) Similarity metric



(f) Subword pooling

Figure 19: Plot of all tested configurations based on their general performance(F1 on x-axis) against their precision performance measured by (max(Precision) $*$ $pear$(Precision)) on the y-axis. The plot remains the same throughout the figure but the distribution of different configuration parameter is shown in color.

configurations that use all layers. Since this is consistent across the whole F1 range we do not think that these are outliers. Instead, we believe that the vector that results from choosing only the last layer is more dependent on similarity choice. Whereas choosing the last four layers is a bit more robust across different similarity and pooling strategies. This confirms the dominant recommendation to use the last four layers. However, when we switch our focus on the F1 we notice that a majority of configurations the right edge (F1$>$ 0.5) of the graph are using all layers.

6.3.3 Trainingset

In Figure 19c we investigate the trainingsets used for domain adaption fine-training. The choice of training set does not seem to have a strong impact on the max F1 distribution (x-axis). However, we can clearly see that using the unprocessed pdf dump (CLF) gives in general the worst precision performance. This is to be expected as this contains texts that are not directly relevant to the requirements. Further, the pdf dump contains many incorrect splits which result in incomplete sentences. Surprisingly some of the strongest configurations in the top right have also been achieved with the CLF set. These configurations also earned the CLF trainingset 3 out of the 10 spots in our top 10 ranking, Figure 18. Unlike the goal-specific requirement set (CLR) that did not secure any spots. Despite not breaking into the top ten CLR shows the strongest precision performance among the three trainingsets on average. If we ignore the results from the contextualized embeddings (F1$<$ 0.4). However, it fails to establish any configuration at the very top right of the graph. Using all available public requirements (PR) we find a patch of configurations that towers above the rest in terms of combined precision performance ($0.425 < x < 0.475, y > 0.332$). This cluster allowed the PR set to take the majority of spots in our top ten ranking. We believe the PR set offers stronger precision since it is larger and contains a larger variety of requirements. From this data, we would recommend future projects to start out using the raw pdf dumps as these are the easiest to acquire. Then if available fine-training with a large requirement set might offer some performance benefits though it requires manual effort to establish such a set. Using only the requirements that we later evaluate provides significant performance improvements over models without domain adoption.

6.3.4 Similarity metric

Figure 19e shows a good separation between the cosine similarity and Jaccard. The Jaccard index consistently provides better precision and worse F1 scores. This confirms both our previous observations and the claims in [Zhe+19]. They did also propose their own similarity measure DynaMax based on fuzzy set theory. As such it is necessary to establish a universe set to compute the DynaMax similarity. In theory, this universe contains all possible word vectors. In practice, this would not only be too expensive to establish but also too large to store. They proposed to use the embedding matrix of a static word embedding method as the universe set. While not discussed in the paper the code[22] that was published alongside the paper suggests using the vectors of the two phrases that are being compared as the universe. This may work for full sentences but we found a universe of 4-10 vectors as is typical with our phrase lengths to be insufficient. As such we excluded DynaMax similarity from further testing. If one can find a reasonable universe set choice for transformer models it might outperform the Jaccard index in terms of precision as was the case for static embedding models in their publication. One could collect the vectors of all phrases or even better all requirements before computing similarities and construct a universe set from that. Since these vectors are computed anyways this should incur much computational overhead. The large universe set will however increase the computational cost of computing the similarity itself. However, this will need to be tested in future works. Overall we feel confident in recommending Jaccard Index for tasks that focus on semantic similarity such as CCR.

6.3.5 Model architecture

Figure 19d follows a similar distribution as the similarity graph. MP-NET has a much stronger capability to differentiate between coreferences and random phrases as seen by its higher precision score. Along with its leading position on the STS benchmark, we feel confident in recommending MP-net for use in requirement engineering. While it has a lower F1 score than competing models it makes up for it by a strong precision score and a similarity metric that allows to rank results.

[22] https://github.com/babylonhealth/fuzzymax

Figure 20

6.3.6 Subword tokenization

Finally, Figure 19f explores the choice of subword token pooling. On average choosing only the first token of a word and discarding the rest performs on par with mean pooling. For contextualized embeddings, it performs the best out of all pooling strategies. This might seem like discarding information leads to better results. We believe that the cause for the performance gain comes from avoiding the suboptimal pooling strategies like max- or mean-pooling. Seeing the last token strategy perform the worst in this task is unsurprising as the ending of words usually contains grammatical hints and carries less semantic meaning. Surprisingly max-pooling performs just as badly. Even more surprising is the location of mean pooling configurations at the top of the precision scores (x$>$ 0.4,y$>$ 0.3). If we check in the previous graph Figure 19e we find that most of the configurations in that region use Jaccard similarities. With our intuition gained from the theoretical discourse on set theory, Finding 3.5, we would expect this mean pooling to perform poorly when paired with Jaccard-Index. Yet a majority of the strongest configurations use precisely this combination.

6.3.7 Baseline

Based on our scoring function we pick the two best configurations per model architecture and plot them, Figure 20. We then compare these to two traditional syntactic text similarities.

First we picked the Levenshtein similarity. It is still widely used and is based on the number of editing operations needed to convert one string into the other. We can see from the graph that this offers very poor performance. A much more recent approach splits each input sequence into tokens much like our neural networks would. The tokens from each input sequence are then compared

with each other rather than the whole phrase at once. For each token in the first phrase, the most similar token in the second phrase is found. Then only the similarity for the most similar pairs contributes to the overall similarity of the two phrases. The method is agnostic to the similarity method used to compare tokens. This approach is known as Mongue-Elkan and has been recently been generalized to make use of partially assigned pair similarity rather than only counting strongest pair, [Jim+09b]. We use this generalized method with Levenshtein similarity as the inner similarity between tokens. We can see that its recall performance is on par with the transformer-based model(Bert, MP-NET). Its max precision does trail all tested models and also exhibits a negative pearson correlation. Regardless it is impressive how close this simple approach gets to the tested methods. Or the other way around it is disappointing that despite the massive computational cost of tested models the gap in the max F1 score is rather small with 8-15%. The saving grace of the transformer models lays in the precision performance which as argued above, section 6, is more relevant to the CCR task as well as any other search task. We have also seen in Figure 19a that our model loses recall if syntactic information is confuscated by contextualization. As such these models still rely heavily on syntactic information instead of deriving and relying on semantics as we had hoped. While the precision curve reveals the promising progress made with the help of neural networks overall these methods do not yet offer solid semantic embeddings.

6.4 Supervised phrase similarity

The best models for Semantic Textual Similarity (STS) are supervised models. That means they have been trained on sentence pairs that have been manually scored for their similarity. There are both test and training sets available for the STS task that are based on sentence pairs. At the time of writing the model with the strongest performance on these test sets is a MpNet[23] that has been fine-tuned on cosine similarity. This means that if we want to continue fine-tuning on our domain we would need a training set with manually collected similarity scores. If we had enough coreference pairs we might be able to fine-tune them instead. Since this is not the case we do not have the ability to adapt the model. Further, since it has been fine-tuned with pooling operation and similarity metric locked in place all our configuration parameters we tested previously are set and baked into this model. In Figure 21 we graph both the supervised model that has been evaluated on our test set as well the strongest recall, precision[24] and F1 configuration of the unsupervised models we discussed

[23] https://www.sbert.net/docs/pretrained_models.html

[24] The best precision is selected as before by $(max(\text{Precision}) * pear(\text{Precision}))$.

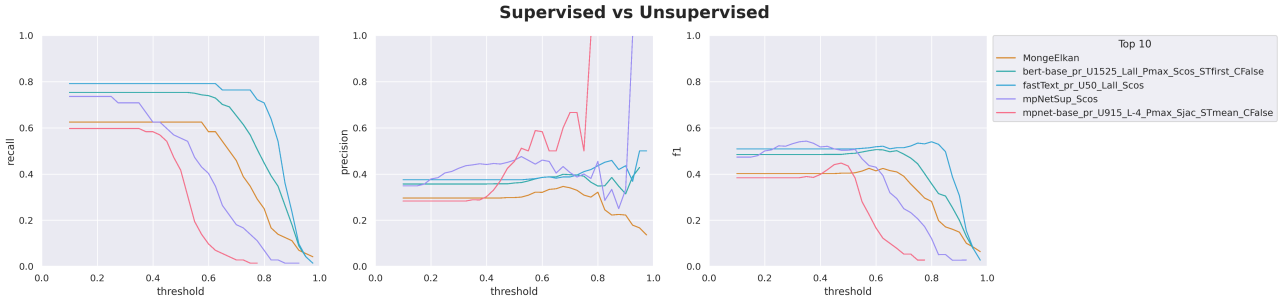


Figure 21: Publicly available supervised MpNet (mpNetSup) compared to best recall, precision and F1 unsupervised fine-trained model configurations from our test. Best recall is achieved by the FastText model, best precision by the unsupervised MpNet and best F1 by the Bert model.

above.

We notice that the supervised model can not outperform the unsupervised and domain fine-trained models, in any metric. Its max recall is below both FastText and Bert. It has a strong precision across the whole threshold range. However, it struggles to differentiate coreferences from random phrases on larger similarity values as seen by the drop in precision ($0.5 < x < 0.85$). The max F1 score is just slightly below the peak of the BERT graph. Overall the supervised model performs better than the syntactical algorithm MongeElkan. The supervised model is a solid choice especially since no training is required for this performance. Our fine-trained MpNet does have a significantly higher precision-threshold correlation and overall precision performance. As such we conclude that unsupervised domain fine-training can yield significant advantages over semi-task specific[25] supervised models that rely entirely on transfer learning[26].

6.5 Qualitative evaluation

For a qualitative evaluation we apply mpnet_clf_U15_L-4_Pmax_Sjac_STfirst model configuration to a list of requirements. We compare all possible pairs with our pipeline and record the resulting coreferences. We then manually inspect these results.

[25]This model is trained on STS task while similar to CCR it is not the same so we refer to it as semi-task specific.

[26]Transfer learning here refers to the fact that the supervised model was trained on a different domain and will apply the learned knowledge to CCR task on requirements. As such we implicitly test how well this model transfers knowledge from the training domain to requirement engineering.

Ideally, we would like to apply our pipeline to a collection of requirements that have not yet been worked through by a requirement engineer. Unfortunately, all requirement specifications we could find are from finished projects. These specification documents had months if not years to mature. As such it is unlikely that many relevant coreferences remain. Nevertheless, analyzing the output on such a dataset can give us insights into false positives. This can help to develop filter rules we can apply to the output to improve performance on real requirement sets.

For the evaluation, we choose the model configuration and the *RiskPrevention* dataset [SNK18] which is not part of the PURE requirement collection. First, we remove phrase duplicates, which means phrase pairs that are either duplicated or permutated. As we do not use contextualization the assigned similarity will be the same regardless of the context(requirement) they are used in. While it might be useful for a human evaluator to view all instances of a critical coreference pair most will not require additional examples. In the remaining results, Appendix C, we noticed three patterns that lead to false positives.

The first group consists of slight spelling variations of phrase pairs for example "*course time-table page*" and "*course time-tabl**es** page*". These types of errors or grammatical variations are not what we are trying to address with CCR. As such these kinds of results might confuse the end-user. We should be able to differentiate these spelling errors from other results by computing the Levenshtein distance between phrases. The Levenshtein distance counts how many characters need to be changed to transform one phrase into the other. It stands to reason that any phrase pair that is only one or two character changes away from the other is just a gramatical variation of the other. Depending on the application some of these results might be typos and as such still useful. Though as already mentioned they should probably be presented to the end-user separately.

The second group consists of phrase pairs that are identical in all but one word. For example "*reports on **bed availability** about the following information*" and "*reports on **patients** about the following information*". We can see that both phrases have a strong overlap which will improve their similarity score. Also, it seems our noun chunking was a bit too eager. Finally *patient* and *bed* are in the same general domain of *health* so it makes sense if their embeddings would be close to each other in at least a few dimensions. We should be able to mitigate these false positive to some extend by isolating the core of the phrases that actually differs. This core needs to be retested. Whether it's best to reuse the same VSM or use a different model remains to be seen. If the phrase core is a single word we also have the option to explore Knowledge Base (KB),

subsection 2.7. This coincides with the third pattern and will be explained in the next paragraph.

The third group consists of single nouns that are not coreferent. For example *http* and *chat* or *server* and *transaction*. Most of these word pairs are used in the same domain. We can probably find a lot of examples where *server* and *transaction* occur in the same sentence. As by the distributional hypothesis section 3 that all VSM are based upon these words should be closer to each other. As such it will be difficult to use any VSM to differentiate between these types of word pairs and actual coreferences. Fortunately, since these are single words we now actually have the option to use a Knowledge Base (KB) like WordNet. As discussed in subsection 3.6 WordNet records the relation between word senses, not phrases. So while we can not use it to compare phrases it might help us to make sense of these word pairs. For example, if a synonym relation between the words is recorded we can confirm the coreference. If an antonym relation is recorded instead we can be sure that the pair is not coreferent. This only works if we can find both words in WordNet. As explained this is unlikely to succeed for all words but should work for 90+% of cases. We could apply the same method to differing phrase cores from the previous paragraph as long as both cores contain only a single word.

We leave the actual implementations and exploration of the output filtering to future works.

7 Future works

7.1 Fuzzy set similarity

We have established that the Jaccard index outperforms cosine similarity in our tests.[Zhe+19] proposed their own similarity measure DynaMax based on fuzzy set theory. Unfortunately, the DynaMax similarity proposed by the same paper failed in our tests. Since the transformer models do not provide a simple embedding matrix that we can use as a universe set, we instead used a drastically reduced universe set that only contains the vectors of the two input phrases. Unsurprisingly, this similarity measure does not perform well without a proper universe matrix. Exploring the extraction of such a universe set could add another 3-5% precision if similar improvements over Jaccard as reported in the paper can be attained. As mentioned in subsubsection 6.3.4 we leave the exploration of universe sets to future works.

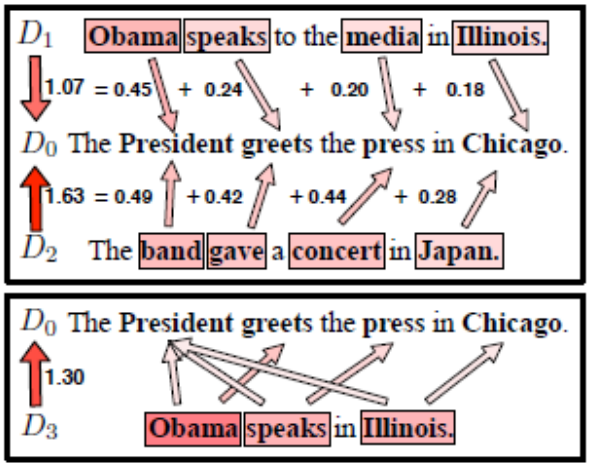


Figure 22: Each arrow depicts the amount of work to go from one word in phrase to a word in another phrase. If the word count does not match some words need to contribute to multiple words from the other phrase. Figure from [Kus+]

7.2 Token based phrase similarity

We saw that similarity metrics on token rather than phrase-level can offer significant advantages as was the case when switching from Levenshtein to Mongue-Elkan. For our vector similarities, we instead merge the token representations with pooling. Since Mongue-Elkan leaves the choice of the inner similarity measure between tokens open, it should be possible to use both cosine and Jaccard similarity here.

A similar idea has been implemented by [Kus+]. Their perspective on the task is based on the Earth Mover Distance (EMD). Conceptually, EMD imagines each input as piles of earth in a domain. It then computes the minimal amount of effort to turn the piles from the first input into the piles of the second input. Of course, this requires both inputs to contain the same amount of dirt. Or in other words, the inputs need to be normalized. They consider each token as a pile of dirt and the euclidean distance between word vectors as the cost to turn one pile into the other, Figure 22 upper half. As we assume the amount of dirt to be the same in both phrases this also works for phrases of differing length. Then the dirt needs to be collected from multiple piles. As such, they also avoid phrase pooling, Figure 22 lower half. They only evaluate their approach on classification and sentiment analysis. Testing this approach on semantic similarity benchmarks could be a valuable contribution.

7.3 Extended trainingset

One surprising result from our training was the performance of some configurations fine-trained with the Clarus free-text dataset. Despite the lack of processing some configurations performed among the best. It would be interesting, to collect the text specifications from all PURE projects and compare the performance to that of the PR set. This would extent PR with context information just like CLF extended CLR, subsubsection 5.2.2. We would expect slightly higher precision scores and slightly lower F1 max values, if it follows the same pattern as switching from CLR to PR.

7.4 User Interaction

Detecting coreferences is crucial. Detection is of little practical use if we do not also offer a way to resolve them. As we have seen, the pipeline presented in this thesis offers good results but is far from fail-safe. As such the resolution of coreferent phrases will require the help of humans for the foreseeable future. Presenting an author with an unsorted list of potential coreferences is probably not the most effective approach. In this thesis, we have laid the groundwork for a successful user interface by placing a strong emphasis on precision-threshold correlation. As previously explained this allows us to sort results by similarity score. Thus enabling to show the most relevant coreferences first. Besides the coreferent phrases, the user needs to see the requirements that the phrases originated from. In some cases, the user might even need to look at surrounding requirements or the section title. The interface might also provide the option to automatically replace the less common with the more common phrase. This would allow the user to apply the resolution with a single click. A well-structured interface that provides the user with the necessary information is crucial for the practical application of CCR to requirements engineering.

7.5 Reoccurring phrases

In this thesis, we only compare two requirements at a time. This means we treat each occurrence of a phrase separately. Recall that we assumed identical phrase forms to be trivially coreferent. In other words, we assume that identical phrases refer to the same component within one project. We do not need to resolve these different occurrences. However, we could use the different requirements that these phrases occur in to extract multiple contextualized embeddings for the same phrase. This could provide a more complete representation of the

component.

The obvious problem is that this requires contextualized embeddings. Otherwise, we would only feed the phrase form through the network and get the same embedding independent of the requirement the phrase was cut out of. While we believe that contextualized embeddings are a necessary step to handle polysemy, we have seen, subsubsection 6.3.1 that for now, foregoing contextualization offers significant F1 score improvements. However, once models that can handle contextualization more robustly are available, considering multiple occurances should offer precision score improvements.

For now, however, we only use static embeddings as such we do not need to compare requirement pairs and can instead compare phrase pairs. This also allows us to reduce the number of phrases that need to be compared by removing duplicates.

8 Conclusion

In this thesis, we established a basic overview of the linguistic challenges of word similarity and synonym detection. We then presented the general ideas current natural language models use to tackle these challenges. We condensed the abstract task of synonym detection to a better-defined task of detecting coreferences in project components specifically for requirements engineering. We propose the novel term *component coreference resolution* (CCR) for this task. While we are not the first to tackle this task, to the best of our knowledge, no term has been established or proposed until now. Further, we are the first to publish a test dataset on the task. We proposed a fully unsupervised approach that tackles the whole CCR task from phrase detection to semantic comparison.

The core of our CCR pipeline lays in the semantic vector embedding of phrases. The idea of using vector embeddings created by neural networks is well established. However, for more recent models it is not clear what the ideal way to extract, combine and compare vectors is. We tested the most common and some promising new approaches. We showed that all models benefit from unsupervised domain fine-training. We also demonstrated that even training with a minimally prepossessed pdf extraction can provide substantial performance gains. From our results, we would recommend to develop and test models on pdf extractions to avoid the semi-manual requirement extraction. This should be sufficient to evaluate the different systems. To achieve peak performance, a training-dataset with more pre-processing might be necessary.

We have found contextualized embeddings to provide more consistent results at the cost of a substantially lower recall. Our tests indicate that contextualization might generally result in worse phrase similarity measures for the current generation of transformer models. Another surprising result is the sub-par performance of the almost exclusively used cosine similarity in the NLP community. Instead, we investigated the Jaccard Index and found its precision score to outperform cosine in most instances. Further Jaccard similarity seemed to provide a stronger precision threshold correlation. This is essential for tasks beyond CCR. For example, a semantic search would want to rank the search results based on their similarity. This is only sensible if the model configuration exhibits a strong positive correlation between precision and threshold. We feel confident in recommending the use of Jaccard Similarity and avoidance of contextualization for phrase similarity for the models tested in this thesis. [ÁKK21] proposed subword pooling but only considered cosine similarity in their experiments. To our knowledge, this thesis is also the first publication to evaluate both subword token pooling and similarity metrics together. While our tests are not conclusive one surprising finding is that subword average token pooling paired with Jaccard similarity performs on par with other configurations.

References

[AA21] Assad Alzayed and Ahmed Al-Hunaiyyan. "A Bird's Eye View of Natural Language Processing and Requirements Engineering". en. In: *International Journal of Advanced Computer Science and Applications* 12.5 (2021). ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2021.0120512. URL: http://thesai.org/Publications/ViewPaper?Volume=12&Issue=5&Code=IJACSA&SerialNo=12 (visited on 06/12/2021).

[ABS15] Eneko Agirre, Ander Barrena, and Aitor Soroa. "Studying the wikipedia hyperlink graph for relatedness and disambiguation". In: *CoRR* abs/1503.01655 (2015). URL: http://arxiv.org/abs/1503.01655.

[AJ19] Israa Abdurrahman and Istabraq Jawad. "The ability of eFL students to differentiate between homonymy and polysemy". In: *Journal of Language Studies* 1.1 (2019), pp. 119–153. DOI: 10.25130/lang.v1i1.6. URL: http://jls.tu.edu.iq/index.php/lang/article/view/6.

[ÁKK21] Judit Ács, Ákos Kádár, and Andras Kornai. "Subword Pooling Makes a Difference". en. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, 2021, pp. 2284–2295. DOI: 10.18653/v1/2021.eacl-main.194. URL: https://aclanthology.org/2021.eacl-main.194 (visited on 10/28/2021).

[ALS14] Eneko Agirre, Oier López de Lacalle, and Aitor Soroa. "Random Walks for Knowledge-Based Word Sense Disambiguation". en. In: *Computational Linguistics* 40.1 (Mar. 2014), pp. 57–84. ISSN: 0891-2017, 1530-9312. DOI: 10.1162/COLI_a_00164. URL: https://direct.mit.edu/coli/article/40/1/57-84/1454 (visited on 06/01/2021).

[ALS18] Eneko Agirre, Oier López de Lacalle, and Aitor Soroa. "The risk of sub-optimal use of Open Source NLP Software: UKB is inadvertently state-of-the-art in knowledge-based WSD". In: *Proceedings of workshop for NLP open source software (NLP-OSS)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 29–33. DOI: 10.18653/v1/W18-2505. URL: https://www.aclweb.org/anthology/W18-2505.

[Bak+21] Özge Bakay et al. “Turkish WordNet KeNet”. In: *Proceedings of the 11th global wordnet conference*. University of South Africa (UNISA): Global Wordnet Association, Jan. 2021, pp. 166–174. URL: https://www.aclweb.org/anthology/2021.gwc-1.19.

[BF08] Alex Baron and Marjorie Freedman. “Who is Who and What is What: Experiments in cross-document co-reference”. In: *Proceedings of the 2008 conference on empirical methods in natural language processing*. Honolulu, Hawaii: Association for Computational Linguistics, Oct. 2008, pp. 274–283. URL: https://www.aclweb.org/anthology/D08-1029.

[Boj+16] Piotr Bojanowski et al. “Enriching word vectors with subword information”. In: *CoRR* abs/1607.04606 (2016). URL: http://arxiv.org/abs/1607.04606.

[Bro+20] Tom B. Brown et al. “Language models are few-shot learners”. In: *CoRR* abs/2005.14165 (2020). URL: https://arxiv.org/abs/2005.14165.

[Cer+17] Daniel M. Cer et al. “SemEval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation”. In: *CoRR* abs/1708.00055 (2017). URL: http://arxiv.org/abs/1708.00055.

[Cer+18a] Daniel Cer et al. “Universal sentence encoder”. In: *CoRR* abs/1803.11175 (2018). URL: http://arxiv.org/abs/1803.11175.

[Cer+18b] Daniel Cer et al. “Universal sentence encoder for English”. In: *Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 169–174. DOI: 10.18653/v1/D18-2029. URL: https://aclanthology.org/D18-2029.

[CH90] Kenneth Ward Church and Patrick Hanks. “Word association norms, mutual information, and lexicography”. In: *Computational Linguistics* 16.1 (1990), pp. 22–29. URL: https://www.aclweb.org/anthology/J90-1003.

[Cho+21] Hyunjin Choi et al. “Evaluation of BERT and ALBERT Sentence Embedding Performance on Downstream NLP Tasks”. en. In: *arXiv:2101.10642 [cs]* (Jan. 2021). URL: http://arxiv.org/abs/2101.10642 (visited on 07/31/2021).

[CLJ19] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. “On the relationship between self-attention and convolutional layers”. In: *CoRR* abs/1911.03584 (2019). URL: http://arxiv.org/abs/1911.03584.

[CV14] Agata Cybulska and Piek Vossen. “Using a sledgehammer to crack a nut? Lexical diversity and event coreference resolution”. In: *Proceedings of the ninth international conference on language resources and evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), May 2014, pp. 4545–4552. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/840%3Csub%3EP%3C/sub%3Eaper.pdf.

[Das+21] Souvick Das et al. “Sentence Embedding Models for Similarity Detection of Software Requirements”. In: *SN Computer Science* 2.2 (Feb. 2021), p. 69. ISSN: 2661-8907. DOI: 10.1007/s42979-020-00427-1. URL: https://doi.org/10.1007/s42979-020-00427-1.

[Dev+18] Jacob Devlin et al. “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *CoRR* abs/1810.04805 (2018). URL: http://arxiv.org/abs/1810.04805.

[Dom20] Pedro Domingos. “Every model learned by gradient descent is approximately a kernel machine”. In: *CoRR* abs/2012.00152 (2020). URL: https://arxiv.org/abs/2012.00152.

[DSL18] Fabiano Dalpiaz, Ivor van der Schalk, and Garm Lucassen. “Pinpointing Ambiguity and Incompleteness in Requirements Engineering via Information Visualization and NLP”. en. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Erik Kamsties, Jennifer Horkoff, and Fabiano Dalpiaz. Vol. 10753. Cham: Springer International Publishing, 2018, pp. 119–135. ISBN: 978-3-319-77242-4 978-3-319-77243-1. DOI: 10.1007/978-3-319-77243-1_8. URL: http://link.springer.com/10.1007/978-3-319-77243-1_8 (visited on 05/23/2021).

[DSS21] Philipp Dufter, Martin Schmitt, and Hinrich Schütze. “Position information in transformers: An overview”. In: (2021).

[DW15] Sourav Dutta and Gerhard Weikum. “Cross-document co-reference resolution using sample-based clustering with knowledge enrichment”. In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 15–28. DOI: 10.1162/tacl0119. URL: https://www.aclweb.org/anthology/Q15-1002.

[EP07] Eneko Agirre and Philip Edmonds. *Word Sense Disambiguation Algorithms and Applications*. Springer Netherlands, 2007.

[Fan+18] Alessandro Fantechi et al. “Hacking an ambiguity detection tool to extract variation points: An experience report”. In: *Proceedings of the 12th international workshop on variability modelling of software-intensive systems*. VAMOS 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 43–50. ISBN: 978-1-4503-5398-4. DOI: 10.1145/3168365.3168381. URL: https://doi.org/10.1145/3168365.3168381.

[FSG18] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. *PURE: a Dataset of Public Requirements Documents*. Zenodo, Sept. 2018. DOI: 10.5281/zenodo.1414117.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[Gil96] Brendan S. Gillon. “The lexical semantics of English count and mass nouns”. In: *Breadth and depth of semantic lexicons*. 1996. URL: https://www.aclweb.org/anthology/W96-0307.

[Har54] Zellig S. Harris. “Distributional structure”. In: *Papers in structural and transformational linguistics*. Dordrecht: Springer Netherlands, 1954, pp. 775–794. ISBN: 978-94-017-6059-1. DOI: 10.1007/978-94-017-6059-16. URL: https://doi.org/10.1007/978-94-017-6059-1%E2%82%836.

[Hav+02] Taher H. Haveliwala et al. “Evaluating strategies for similarity search on the web”. In: *Proceedings of the 11th international conference on world wide web*. WWW ’02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 432–442. ISBN: 1-58113-449-5. DOI: 10.1145/511446.511502. URL: https://doi.org/10.1145/511446.511502.

[Hey+20] Tobias Hey et al. “NoRBERT: Transfer Learning for Requirements Classification”. en. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. Zurich, Switzerland: IEEE, Aug. 2020, pp. 169–179. ISBN: 978-1-72817-438-9. DOI: 10.1109/RE48521.2020.00028. URL: https://ieeexplore.ieee.org/document/9218141/ (visited on 06/13/2021).

[HLL20] Ruijuan Hu, Zhufeng Li, and Jian Li. “Research on entity coreference resolution technology oriented to military knowledge graph”. In: *Journal of Physics: Conference Series* 1624 (Oct. 2020), p. 052003. DOI: 10.1088/1742-6596/1624/5/052003. URL: https://doi.org/10.1088/1742-6596/1624/5/052003.

[Hov+06] Eduard Hovy et al. “OntoNotes: The 90% solution”. In: *Proceedings of the human language technology conference of the NAACL, companion volume: Short papers*. NAACL-Short ’06. USA: Association for Computational Linguistics, 2006, pp. 57–60.

[Jim+09a] Sergio Jimenez et al. “Generalized mongue-elkan method for approximate text string comparison”. In: *Computational linguistics and intelligent text processing*. Ed. by Alexander Gelbukh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 559–570. ISBN: 978-3-642-00382-0.

[Jim+09b] Sergio Jimenez et al. “Generalized mongue-elkan method for approximate text string comparison”. In: vol. 5449. Mar. 2009, pp. 559–570. ISBN: 978-3-642-00381-3. DOI: 10.1007/978-3-642-00382-0_45.

[JK95] John S. Justeson and Slava M. Katz. “Technical terminology: some linguistic properties and an algorithm for identification in text”. en. In: *Natural Language Engineering* 1.1 (Mar. 1995), pp. 9–27. ISSN: 1351-3249, 1469-8110. DOI: 10.1017/S1351324900000048. URL: https://www.cambridge.org/core/product/identifier/S1351324900000048/type/journal_article (visited on 06/23/2021).

[JM20] Dan Jurafsky and James H. Martin. *Speech and Language Processing (3rd ed. draft)*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2020. ISBN: 978-0-13-187321-6 0-13-187321-0. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

[Jos+19] Mandar Joshi et al. “BERT for coreference resolution: Baselines and analysis”. In: *CoRR* abs/1908.09091 (2019). URL: http://arxiv.org/abs/1908.09091.

[Kam+03] Erik Kamsties et al. “From contract drafting to software specification: Linguistic sources of ambiguity”. In: parallelism (Dec. 2003).

[Kam05] Erik Kamsties. “Understanding Ambiguity in Requirements Engineering”. en. In: *Engineering and Managing Software Requirements*. Ed. by Aybüke Aurum and Claes Wohlin. Berlin/Heidelberg: Springer-Verlag, 2005, pp. 245–266. ISBN: 978-3-540-25043-2. DOI: 10.1007/3-540-28244-0_11. URL: http://link.springer.com/10.1007/3-540-28244-0_11 (visited on 05/20/2021).

[KGS20] Jenna Kanerva, Filip Ginter, and Tapio Salakoski. “Universal lemmatizer: A sequence to sequence model for lemmatizing universal dependencies treebanks”. In: *Natural Language Engineering* (2020), pp. 1–30. DOI: 10.1017/S1351324920000224. URL: http://dx.doi.org/10.1017/S1351324920000224.

[Kim+15] Yoon Kim et al. “Character-aware neural language models”. In: *CoRR* abs/1508.06615 (2015). URL: http://arxiv.org/abs/1508.06615.

[Kiy+08] Nadzeya Kiyavitskaya et al. “Requirements for tools for ambiguity identification and measurement in natural language requirements specifications”. In: *Journal of requirements* 13 (Sept. 2008), pp. 207–239. DOI: 10.1007/s00766-008-0063-7.

[KR20] Sopan Khosla and Carolyn Penstein Rosé. “Using type information to improve entity coreference resolution”. In: *CoRR* abs/2010.05738 (2020). URL: https://arxiv.org/abs/2010.05738.

[Kus+] Matt J Kusner et al. “From Word Embeddings To Document Distances”. en. In: (), p. 10.

[LC99] Xia Lin and Lois Mai Chan. “Personalized knowledge organization and access for the web”. In: *Library & Information Science Research* 21.2 (1999), pp. 153–172. ISSN: 0740-8188. DOI: https://doi.org/10.1016/S0740-8188(99)00007-9. URL: https://www.sciencedirect.com/science/article/pii/S0740818899000079.

[Lee+17] Kenton Lee et al. “End-to-end neural coreference resolution”. In: *CoRR* abs/1707.07045 (2017). URL: http://arxiv.org/abs/1707.07045.

[Lev66] Vladimir Iosifovich Levenshtein. “Binary codes capable of correcting deletions, insertions and reversals.” In: *Soviet Physics Doklady* 10.8 (Feb. 1966), pp. 707–710.

[LG14] Omer Levy and Yoav Goldberg. “Neural word embedding as implicit matrix factorization”. In: *Proceedings of the 27th international conference on neural information processing systems - volume 2*. NIPS’14. Cambridge, MA, USA: MIT Press, 2014, pp. 2177–2185.

[Liu+19] Yinhan Liu et al. “RoBERTa: A robustly optimized BERT pretraining approach”. In: *CoRR* abs/1907.11692 (2019). URL: http://arxiv.org/abs/1907.11692.

[LJ19] Daniel Loureiro and Alípio Jorge. “Language modelling makes sense: Propagating representations through WordNet for full-coverage word sense disambiguation”. In: *Proceedings of the 57th annual meeting of the association for computational linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 5682–5691. DOI: 10.18653/v1/P19-1569. URL: https://www.aclweb.org/anthology/P19-1569.

[Mar+19] Marco Maru et al. “SyntagNet: Challenging Supervised Word Sense Disambiguation with Lexical-Semantic Combinations”. en. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, 2019, pp. 3532–3538. DOI: 10.18653/v1/D19-1359. URL: https://www.aclweb.org/anthology/D19-1359 (visited on 05/30/2021).

[Mar11] G.R.R. Martin. *A game of thrones*. Fantasy (bantam books). Bantam Books, 2011. ISBN: 978-0-553-38679-0. URL: https://books.google.de/books?id=hXNvadj27ekC.

[Mik+13] Tomas Mikolov et al. *Efficient estimation of word representations in vector space*. 2013.

[Mil+90] George A. Miller et al. “Introduction to WordNet: An on-line lexical database”. English. In: *Journal of Lexicography* 3.4 (1990), pp. 235–244. URL: ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.pdf.

[Mil95] George A. Miller. “WordNet: A lexical database for english”. In: *Communications of the ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748.

[Mit99] Ruslan Mitkov. *Anaphora resolution: the state of the art*. Citeseer, 1999.

[MM02] Rada Mihalcea and Dan Moldovan. “eXtended WordNet: progress report”. In: (Aug. 2002).

[MR19] John P McCrae and Alexandre Rademaker. “English WordNet 2019 – An Open-Source WordNet for English”. en. In: (2019), p. 8.

[Nav18] Roberto Navigli. “Natural language understanding: Instructions for (present and future) use”. In: *Proceedings of the twenty-seventh international joint conference on artificial intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 5697–5702. DOI: 10.24963/ijcai.2018/812. URL: https://doi.org/10.24963/ijcai.2018/812.

[Ng17] Vincent Ng. “Machine learning for entity coreference resolution: A retrospective look at two decades of research”. In: *Proceedings of the thirty-first AAAI conference on artificial intelligence*. AAAI’17. AAAI Press, 2017, pp. 4877–4884.

[NP12] Roberto Navigli and Simone Paolo Ponzetto. "BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network". en. In: *Artificial Intelligence* 193 (Dec. 2012), pp. 217–250. ISSN: 00043702. DOI: 10.1016/j.artint.2012.07.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0004370212000793 (visited on 05/31/2021).

[ORD19] Katrin Ortmann, Adam Roussel, and Stefanie Dipper. "Evaluating off-the-shelf NLP tools for german". In: *Proceedings of the 15th conference on natural language processing (KONVENS 2019): Long papers*. Erlangen, Germany: German Society for Computational Linguistics & Language Technology, 2019, pp. 212–222.

[Pet+18] Matthew Peters et al. "Deep contextualized word representations". In: *Proceedings of the 2018 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://www.aclweb.org/anthology/N18-1202.

[PLM15] Fabian Pittke, Henrik Leopold, and Jan Mendling. "Automatic detection and resolution of lexical ambiguity in process models". In: *IEEE Transactions on Software Engineering* 41.6 (2015), pp. 526–544. DOI: 10.1109/TSE.2015.2396895.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global vectors for word representation". In: *Empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[PSP18] Christian S. Perone, Roberto Silveira, and Thomas S. Paula. "Evaluation of sentence embeddings in downstream and linguistic probing tasks". en. In: *arXiv:1806.06259 [cs]* (June 2018). URL: http://arxiv.org/abs/1806.06259 (visited on 10/28/2021).

[Qi+18] Peng Qi et al. "Universal dependency parsing from scratch". In: *Proceedings of the CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 160–170. URL: https://nlp.stanford.edu/pubs/qi2018universal.pdf.

[Rad+18] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". en. In: (2018), p. 24.

[RG19] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese BERT-Networks". In: *CoRR* abs/1908.10084 (2019). URL: http://arxiv.org/abs/1908.10084.

[Rob+16] Marcel Robeer et al. “Automated Extraction of Conceptual Models from User Stories via NLP”. en. In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. Beijing: IEEE, Sept. 2016, pp. 196–205. ISBN: 978-1-5090-4121-3. DOI: 10.1109/RE.2016.40. URL: https://ieeexplore.ieee.org/document/7765525/ (visited on 05/24/2021).

[SD94] N. Srinivas and Kalyanmoy Deb. “Muiltiobjective optimization using nondominated sorting in genetic algorithms”. In: *Evolutionary Computation* 2.3 (1994), pp. 221–248. DOI: 10.1162/evco.1994.2.3.221.

[SJ15] Unnati S. Shah and Devesh C. Jinwala. “Resolving ambiguities in natural language software requirements: A comprehensive survey”. In: *SIGSOFT Softw. Eng. Notes* 40.5 (Sept. 2015), pp. 1–7. ISSN: 0163-5948. DOI: 10.1145/2815021.2815032. URL: https://doi.org/10.1145/2815021.2815032.

[SNK18] Zain Shaukat, Rashid Naseem, and Muhammad Zubair Khan. “A dataset for software requirements risk prediction”. In: Oct. 2018, pp. 112–118. DOI: 10.1109/CSE.2018.00022.

[Son+20] Kaitao Song et al. “MPNet: Masked and permuted pre-training for language understanding”. In: *CoRR* abs/2004.09297 (2020). URL: https://arxiv.org/abs/2004.09297.

[Spa88] Karen Sparck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Document retrieval systems*. GBR: Taylor Graham Publishing, 1988, pp. 132–142. ISBN: 0-947568-21-2.

[SPN20] Bianca Scarlini, Tommaso Pasini, and Roberto Navigli. “With More Contexts Comes Better Performance: Contextualized Sense Embeddings for All-Round Word Sense Disambiguation”. en. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020, pp. 3528–3539. DOI: 10.18653/v1/2020.emnlp-main.285. URL: https://www.aclweb.org/anthology/2020.emnlp-main.285 (visited on 05/23/2021).

[SWY75] G. Salton, A. Wong, and C. S. Yang. “A vector space model for automatic indexing”. In: *Communications of the ACM* 18.11 (Nov. 1975), pp. 613–620. ISSN: 0001-0782. DOI: 10.1145/361219.361220. URL: https://doi.org/10.1145/361219.361220.

[Tak+18] Kazuma Takaoka et al. “Sudachi: a Japanese tokenizer for business”. In: *Proceedings of the eleventh international conference on language resources and evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://www.aclweb.org/anthology/L18-1355.

[TH19] Daniel Toews and Leif Van Holland. “Determining domain-specific differences of polysemous words using context information”. In: *Joint proceedings of REFSQ-2019 workshops, doctoral symposium, live studies track, and poster track co-located with the 25th international conference on requirements engineering: Foundation for software quality (REFSQ 2019), essen, germany, march 18th, 2019*. Ed. by Paola Spoletini et al. Vol. 2376. CEUR workshop proceedings. CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2376/NLP4RE19_paper02.pdf.

[TP10] Peter D. Turney and Patrick Pantel. “From frequency to meaning: Vector space models of semantics”. In: *CoRR* abs/1003.1141 (2010). URL: http://arxiv.org/abs/1003.1141.

[Vas+17] Ashish Vaswani et al. “Attention is all you need”. In: *CoRR* abs/1706.03762 (2017). URL: http://arxiv.org/abs/1706.03762.

[Wal05] Christopher Walker. *ACE 2005 Multilingual Training Corpus*. 2005.

[Wan+13] Yue Wang et al. “Automatic detection of ambiguous terminology for software requirements”. In: *Natural language processing and information systems*. Ed. by Elisabeth Métais et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 25–37. ISBN: 978-3-642-38824-8.

[Wan+20] Yawen Wang et al. “A Deep Context-wise Method for Coreference Detection in Natural Language Requirements”. en. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. Zurich, Switzerland: IEEE, Aug. 2020, pp. 180–191. ISBN: 978-1-72817-438-9. DOI: 10.1109/RE48521.2020.00029. URL: https://ieeexplore.ieee.org/document/9218208/ (visited on 06/13/2021).

[Wen17] Lilian Weng. “Learning word embedding”. In: *lilianweng.github.io/lil-log* (2017). URL: https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html.

[Wu+16] Yonghui Wu et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *CoRR* abs/1609.08144 (2016). URL: http://arxiv.org/abs/1609.08144.

[You+20] Muhammad Younas et al. “Extraction of non-functional requirement using semantic similarity distance”. en. In: *Neural Computing and Applications* 32.11 (June 2020), pp. 7383–7397. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-019-04226-5. URL: http://link.springer.com/10.1007/s00521-019-04226-5 (visited on 06/13/2021).

[Zha+20] Liping Zhao et al. “Natural language processing (NLP) for requirements engineering: A systematic mapping study”. In: *CoRR* abs/2004.01099 (2020). URL: https://arxiv.org/abs/2004.01099.

[Zhe+19] Vitalii Zhelezniak et al. “Don’t Settle for Average, Go for the Max: Fuzzy Sets and Max-Pooled Word Vectors”. en. In: *arXiv:1904.13264 [cs]* (Apr. 2019). URL: http://arxiv.org/abs/1904.13264 (visited on 07/07/2021).

[ZZS20] Hongming Zhang, Xinran Zhao, and Yangqiu Song. “A Brief Survey and Comparative Study of Recent Development of Pronoun Coreference Resolution”. en. In: *arXiv:2009.12721 [cs]* (Sept. 2020). URL: http://arxiv.org/abs/2009.12721 (visited on 06/16/2021).

A Word Sense Disambiguation Algorithm

A.0.1 Synset

WordNet also enables us to extract subgraphs. For us, the most interesting of these subgraphs are the synsets. A synset contains synonyms as a set of lemmas and their common word sense description (gloss). Using the above example $\{$*trash.n.1, rubbish.n.1, scrap.n.2, waste.n.1*$\}$*(worthless material that is to be disposed of)* would be a synset. This follows the syntax for synsets from [SPN20], $\{l_1, \ldots, l_n\}(g)$. So all lemmas $l_1, ..., l_n$ are synonym to each other if their word sense corresponds to the gloss g in a given context. Unfortunately, these synsets can not express different degrees of synonymity.

A.0.2 SyntagNet

SyntagNet [Mar+19] is a new LKB that records cooccurrence between glosses. For example $\{$*run*$\}$*(to carry out a process)* and $\{$*programs*$\}$*(a sequence of instructions)* occur more often in the same sentence than words with unrelated concepts. It contains 88K manually constructed coocurrances linking about 20K WordNet nodes. When trying to identify the correct word sense the entries in syntagNet provide a strong indication for the correct word sense. While it is difficult to benchmark LKB directly we can benchmark other tasks with methods that make use of an LKB. The authors demonstrated this by combining SyntagNet with UKB, subsection A.1, to by achieving near State of the Art (SOTA) results on a WSD benchmark [ALS14]. Note that the WSD task can not be solved directly with WordNet or SyntagNet but the recorded relations can be useful for solving the WSD task.

A.1 UKB graph-based WSD

UKB is a collection of graph-based WSD approaches. [ALS14] proposed to apply PageRank random walk algorithm [Hav+02] to a LKB; here WordNet. For this they only consider word senses as nodes and then connect them if any type of relation exists, disregarding the type of relation. Given this graph, the main idea behind the PageRank is that we randomly walk through a graph and whenever we go from a node v_i to a node v_j we increase the rank of the destination node v_j. Instead of only measuring first-order connection the amount of gained rank of a node v_j depends on the rank of the outgoing node v_i. This propagates

the importance of nodes to their immediate surroundings. Conceptually this random walk is continued until we achieve an equilibrium of the ranks relative to each other. This will result in a graph that represents the distance between two word senses. In Figure 23 we presented a small portion of the resulting graph. Every word sense from the original LKB is present and connected to its related word senses. Now if we want to resolve the word sense of *trash* in the sentence *I throw trash into the bin* we can do so by finding the shortest path between each word sense of *trash* and the context words like *trash* or *bin*. Of course, we do not know the word senses of the surrounding words so we are looking for the shortest path to any word sense of the surrounding words. The word sense with the shortest combined path to its neighboring words in the sentence is then returned as word sense.

Fortunately, this random walk can be reformulated as a linear system and solved analytically. So far we have static ranks and simply assigned the highest rank to the dominant word sense, assuming that the authors of WordNet spend more effort on the dominant sense of a word.

In subsequent investigations [ALS18; ABS15] propose optimized parameters which achieved an average F1 score of 71% just 6% less than the current SOTA [SPN20].

To perform WSD we need to take the context into account. We take each word in the context and determine their lemma. Then for each lemma, we create a new node in the graph that is connected to all possible word senses as recorded in WordNet. Then we artificially increase the rank of these new nodes. The new nodes effectively act as a rank source. After reapplying the PageRank algorithm we iterate all possible senses of the target word and choose the sense node with the highest rank.

A.2 ARES context AwaRe Embedding of Sense

[SPN20] propose a new method to automatically extract occurrences of a word senses from a general text. They aim to create an ideal vector for every lemma in a synonym set. To create this vector for a particular lemma l in a synset s they first collect all occurrences of that lemma in the corpus. Each occurrence comes with its own context. All of these sentences are then forwarded through a BERT model and its last layers are averaged to give us a separate vector for each occurrence of the lemma. Since these vectors likely include instances where the sense of the lemma is different from the sense s the instances that actually correspond to s need to be identified. The set of vectors is clustered

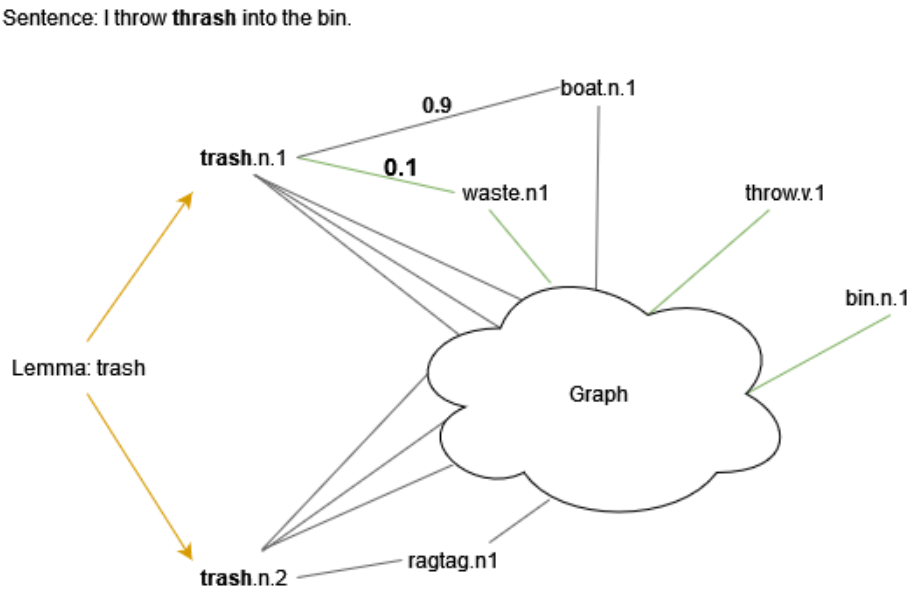


Figure 23: Overview of the graph used for the UKB algorithm. Every word sense in WordNet forms a node. The nodes are connected based on word sense relations. In the figure we abstracted most of the graph into the cloud icon. Every word sense on the perimeter has a number of connections to the nodes inside the rest graph. The PageRank algorithm also provides us with a distance between two connected nodes. The correct word sense is then selected by estimating the shortest distance from each word sense of *trash* to the word sense in its context *[throw, bin, into, ...]*.

with K-NN where K is set to the number of word senses recorded for lemma l. From each cluster, they take the most relevant words and use UKB to identify the most likely cluster for s. They use SyntagNet to increase the number of samples available for each lemma and use an additional concatenation of word embeddings as described in the exceptionally well-written paper [LJ19]. Finally, the clusters can be averaged to create a reference vector for each word sense. The idea of extending the limited sense annotated data with samples from an unlabelled text builds on the previous works of [Pet+18].

As of Jun 2021, they hold the current SOTA on the average SemEval benchmark sets with an F1 of 77.9. Despite this impressive performance, their results do not provide any evidence that the created word sense vectors actually encode semantics better than the underlying BERT model. It is likely that they simply added more anchor points to the semantic space provided by the BERT model. This is akin to overfitting to the WSD task rather than improving the semantic

mapping that is necessary to identify synonyms beyond those listed in a synset.

B Dataset Construction Guidelines

Hello everyone.

As discussed today I want to create a test set for coreference resolution. For our application you can think of coreference resolution as a special kind of synonym resolution. However we focus on domain terminology rather than general synonyms. Thus we want to create a dataset where we refer to the same project component with different words. For this we construct requirement pairs that contain coreferences from an existing dataset. For example given the following req1:

> The Clarus system shall be able to implement quality checking rules for each environmental parameter.

you will have to:

> [CR] construct a requirement req2 that uses a different formulation for one component e.g.
>
> *The Claurus system shall display unused environmental variables.*
>
> - or -
>
> [N] construct a requirement that does not contain any non trivial coreferences. Trivial coreferences are those where we reference the same component with the same formulation. So it is fine to reuse "The Claurus system" for this type of requirement. Example:
>
> *The Clarus system shall collect, quality check, and disseminate environmental data.*

The source project wants to collect, filter, quality check and save weather and environmental data to predict effects on transportation of goods and road conditions. When constructing coreferences you have to be mindful of this context. So while system variable might be coreferent to environmental parameter when talking about operating systems it will not work here. I attached the original pdf, please take a few minutes to read page 90-93, APPENDIX A - DEFINITIONS, ACRONYMS, AND ABBREVIATIONS.

Guidelines:

- Make sure coreferences match in the project domain
- Pick project components not verbs to construct coreferences
- Avoid constructing requirement duplicates. Don't just exchange a phrase from the source requirement.
- Avoid repeating previous coreference pairs
- Skip requirements if you can not find a good coreferences
- Go through skipped requirements and construct coreference free requirements until u get to roughly 50/50
- You can check the first 50 as further examples.

C Quantitative Evaluation Output

Table 2: Top 20 results from applying our pipeline with unsupervised MpNet on a requirement dataset.

Phrase1	Phrase2	similarity	Requirement1	Requirement2
the course time-table page	the course time-tables page	0.817921	The Course time-Table page will provide general course information without any need of registration.	The Course Time-Tables page basically displays information (timings, instructor, class size, pre-requisites) of all the courses offered at UVic.
the transaction processing system software system	the transaction processing system system	0.812852	There are many types of 0.25 as such supported by the Transaction Processing System software system namely; User 0.25, Software 0.25 and Hardware 0.25.	The Transaction Processing System system shall communicate with CRM system to provide support.
the student accounts page	the student account page	0.790	The Student Accounts page will display a student's personal and tuition (monetary) information. It will also contain access to their Netlink account.	The Student Account page contains information like a student's address, tuition account information, Netlink account details, and PinChg.

Continued on next page

Table 2: Top 20 results from applying our pipeline with unsupervised MpNet on a requirement dataset. (Continued)

the transaction processing system	the transaction processing system system	0.769	The Transaction Processing System shall communicate with the content manager to get the product specifications, offerings and promotions.	The Transaction Processing System system shall communicate with export regulation system to validate export regulations.
the student main page	the records main page	0.766	If a student signs in they are taken to the Student Main Page. If a Records employee signs in they are taken to a Records Main Page.	The Records main page would allow a records employee to process Official Transcript orders.
reports on bed availability about the following information	reports on patients about the following information	0.747	The HPIMS shall generate reports on bed availability about the following information: ward name, bed number, occupied/unoccupied.	The HPIMS shall generate reports on patients about the following information: patient's PHN, patient's name, ward name, bed number and the doctor's name which was assigned.
server	reservation	0.744	System can participate in chat with user and SERVER.	at time of pickup the system will allow the employee to bring up the specified reservation and to print out a rental agreement.
registration window for each users	login window for each user	0.741	Web site contains Registration window for each users.	Web site contains login window for each user.

Continued on next page

Table 2: Top 20 results from applying our pipeline with unsupervised MpNet on a requirement dataset. (Continued)

offerings	transaction	0.739	The Transaction Processing System shall communicate with the content manager to get the product specifications, offerings and promotions.	In the event of a system crash during a transaction, the current transaction will either completed or not completed. This will be handled by an already existing database.
reservation	requests	0.737	at time of pickup the system will allow the employee to bring up the specified reservation and to print out a rental agreement.	The Employee is directed to a page that allows them to process Student Record related requests.
transaction	requests	0.736	In the event of a system crash during a transaction, the current transaction will either completed or not completed. This will be handled by an already existing database.	The Employee is directed to a page that allows them to process Student Record related requests.
administrators	compliant	0.733	Administrators shall be able to view and modify all information in HPIMS.	Web site contains Compliant issued dialog box for every user.
server	transaction	0.732	System can participate in chat with user and SERVER.	In the event of a system crash during a transaction, the current transaction will either completed or not completed. This will be handled by an already existing database.

Continued on next page

Table 2: Top 20 results from applying our pipeline with unsupervised MpNet on a requirement dataset. (Continued)

compliant	transaction	0.731	Web site contains Compliant issued dialog box for every user.	In the event of a system crash during a transaction, the current transaction will either completed or not completed. This will be handled by an already existing database.
the web site	the web portal	0.727	User should provide a valid user id and password to access the web site.	The system will provide customers to login on the web portal and view existing reservation in the system.
compliant	http	0.726	Web site contains Compliant issued dialog box for every user.	The protocol used shall be HTTP.
http	chat	0.725	The protocol used shall be HTTP.	With the information provided by administrator user can directly contact with system or he can contact during their chat.
the payments	the reviews	0.724	The Transaction Processing System system shall communicate with billPay system to identify available payment methods , validate the payments and process payment.	The system shall display the reviews and ratings of each product, when it is selected.

Continued on next page

Table 2: Top 20 results from applying our pipeline with unsupervised MpNet on a requirement dataset. (Continued)

server	administrators	0.723	System can participate in chat with user and SERVER.	Administrators are responsible for all of the scheduling and updating day/night employee shifts.
transaction	administrators	0.722	In the event of a system crash during a transaction, the current transaction will either completed or not completed. This will be handled by an already existing database.	Administrators are responsible for all of the scheduling and updating day/night employee shifts.

Rheinische
Friedrich-Wilhelms-
Universität Bonn

Prüfungsausschuss Informatik

universität **bonn** · Institut für Informatik · 53012 Bonn

Vorsitzender des Prüfungsausschusses

Prof. Dr. Thomas Kesselheim

Prüfungsamt:
Judith König
Tel.: 0228/73-4418
Fax : 0228/73-4788
pa@informatik.uni-bonn.de
Postanschrift: 53115 Bonn
Endenicher Allee 19a

www.informatik.uni-bonn.de

Erklärung über das selbständige Verfassen einer Abschlussarbeit/ Declaration of Authorship

Titel der Arbeit/Title:

.....

.....

.....

Hiermit versichere ich, _____, _____,
Name / name, Vorname / first name

dass ich die Arbeit – bei einer Gruppenarbeit meinen entsprechend gekennzeichneten Anteil der Arbeit – selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

…… Bonn, den ……..
Unterschrift/signature date