Determining Domain-specific Differences of Polysemous Words Using Context Information

Daniel Toews
Fraunhofer FKIE
Wachtberg 53343
daniel.toews@fkie.fraunhofer.de

Leif Van Holland
Fraunhofer FKIE
Wachtberg 53343
leif.van.holland@fkie.fraunhofer.de

Abstract

In this paper we describe our early work with determining domain-specific differences of polysemous words. The goal is to determine whether a word has different meanings in two different text corpora. Previous work tried to identify such words by training word embeddings specific for those domains and then comparing the resulting vectors. Our use case, however, does not contain enough text to train word embeddings. Thus, we present a novel method that is able to make use of context information and determine domain-specific differences of words without generating word embeddings.

1 Introduction

When managing multiple related projects with different stakeholders, it is difficult to maintain a consistent usage and interpretation of words. Requirement sets are elicited over a long time span by multiple different people, each using different words, or the same words with different meanings. Additionally, if the projects are related, similar words may be used in each of the requirement sets, but may have slightly different meanings. Such words are called polysemous words. To prevent confusion, maintaining a good overview over such a project structure is critical. We were tasked to design automated software that is able to identify polysemous words.

This confronted us with one of the biggest problems in natural language processing, which is the inherent ambiguity of natural language. It is difficult for software to determine the intention of one sentence, as well as the differentiating nuances when comparing similar requirements, as they strongly depend on the context. Words that are different can have similar meanings, while one word can have multiple interpretations.

An important contribution to the field of NLP are word embeddings. Popularized by Mikolov et al. [MCCD13] [MSC$^+$13], machine learning algorithms can generate such word embeddings, which are able to place words into a vector space in a meaningful manner. Words that are similar to each other will be close to one another, while words that are not similar will be further apart. This allows for more accurate distance calculations between requirements that use similar, but unequal words. Thus, they are able to incorporate information on how words are generally used.

However, word embeddings, as well as other methods [DDF$^+$90], fail to capture words that have multiple interpretations. Those polysemous words, like *group*, which can be used to describe a criminal gang, a political party, a musical band or nearly any other collection of related objects, will be poorly represented as they will be a mixture of their different meanings. In our use case, this could cause confusion about the connection of different elements between projects, as the same words are used, but not the same part of the project is meant.

In other cases it could lead to miscommunication during requirement elicitation [FDG17], or low accuracy for requirement tracing [WNLN18].

This problem can be avoided when using domain-specific texts for training, which match the domain of the requirements, as that would generate a more specific representation of problematic words. Yet, in certain cases it is unrealistic to train word embeddings for niche (sub)domains (in which requirements are often described, as they are highly specific), since collecting enough suitable text material is often infeasible. For example, the word *server*, which would be part of the computer science domain, can even have two meanings in this domain. First, the server can be a service on the internet that provides a response to user requests. Second, the server can be a physical device composed of processor units and other hardware. And even though, both instances of the word can refer to the same object, both aspects will have different types of requirements that have to be solved by different people. Segregating those two meanings with specific word embeddings means collecting and curating enough texts containing different interpretations of this word. We can assume that the related projects mentioned above are all part of the technical or computer science domain. However, the computer science domain is too broad to allow for a precise interpretation of requirements. In our case, one requirement set describes infrastructure to distribute information in areas without internet connection, while the second set describes applications that are built upon this infrastructure. Curating a text corpus to train representative word embeddings for those two cases would require laborious work from experts.

We propose a new method, which is able to calculate the domain- and subdomain-specific differences of words without using different word embeddings. This provides a more flexible and application-friendly solution. The remainder of the paper is structured as follows. Section 2 provides a brief overview of the related work, as well as an explanation of the previous research that is used for our method. In section 3, we present our proposed algorithm. The algorithm is verified in section 4. In section 5, we describe the advantages and shortcomings of our method and how it fits into our application. Section 6 concludes the paper with a summary and a short outlook into future work.

2 Related Work

Today, sophisticated word embedding algorithms, like word2vec [MCCD13], GloVe [PSM14] and fastText [BGJM16], are able to place words in a vector space that contains semantic information about the words. Similar words will be placed close to each other and form groups of words. Therefore, the distance between two words in this space can describe how semantically similar they are. For example, synonyms of a certain word will be placed close to each other, while non-related words will have a higher distance. Additionally, the words have a fixed dimensionality (generally orders of magnitude lower than the number of different words in the corpus). Previous work in requirement analysis discusses the possibility of integrating Word Embeddings [FDE+17] or integrates them in classification approaches [WV16]. Additionally, Lucassen et al. also use the approach for clustering requirements [LDvdWB16].

Looking closer at word2vec, the models are generated by reading large sources of text, such as Wikipedia, and trying to determine the word positions of word x by looking at words used around x. In short, the model is trained by looking at n words before and after x to estimate x (other methods like the Skip-Gram Model [MSC+13], GloVe [PSM14] or fastText [BGJM16] prove to be more precise, but would be too extensive in this report). Thus the position of x in the vector space is determined by the words often used with x. The assumption is, that similar words are used with the same surrounding words. Because of this training, we assume that the model trained with an appropriate data set contains finer nuances of words like *server*, as the surrounding words will catch the different meanings and contexts.

When trying to determine the polysemy of words in requirement sets, different methods were proposed [FDG17], [WNLN18], [FEG18]. In the first method, domain-specific word embeddings of two different domains are merged into one corpus. Certain hand chosen words are marked with a suffix to distinguish the domains. All instances of chosen word a will be replaced with $a_{domain1}$ in the texts from the first domain and with $a_{domain2}$ in the second domain. The word a will receive two different vector representations, one for each domain, which can be compared. The higher the resulting distance, the more different are the interpretations of the word in the domains. However, this means that controversial words need to be known beforehand, as they have to be marked manually. Another method [FEG18] tries to identify polysemous words by looking at a words and its neighbors in a certain domain and then comparing it to the neighbors of the same word in another domain. This means that no preprocessing of words is needed, but it requires one completely trained word embedding for each domain that is involved.

In [DvdSL18], Dalpiaz et al. describe a method to determine how similar words are used in a set of user stories. The goal is to improve the quality of the data set by ensuring only clearly defined terms are used and they are independent from each other. For example, the words *broadcast* and *transmit* can both be used similarly in the computer science domain. User stories for one project, generated from different sources, can contain both words with identical meanings. Using both words however, may cause problems, as the precise nature of user stories, as well as requirements, would suggest that when both words are used, they mean different things. Otherwise only one of those words would be needed. To determine whether words a and b might potentially be interchangeable, the word embedding distance of both words is considered, as well as the contextual distance. For this, the usage of words a and b in the data set are determined by looking at the other words in the same sentence. The more the words in the sentences differ, the higher the resulting contextual difference is. Following formula was used:

Given two terms t_1 and t_2:

$$ambig_{t_1,t_2} = \frac{2 * sim_{t_1,t_2} + simc_{t_1,t_2}}{3},$$

with sim_{t_1,t_2} being the similarity between t_1 and t_2 and $simc_{t_1,t_2}$ being the context similarity. For the context similarity, all sentences that contain t_1 are collected and compared against the sentences that contain t_2. If those collections use similar words, the *simc* of t_1 and t_2 is high.

3 Approach

As described above, word embeddings, as well as *simc* as presented by Dalpiaz et al. [DvdSL18], provide a higher distance between words that have differing contexts. In word embeddings this happens during training, which results in vector representations that are further apart. This means, using *simc*, in comparison to calculating the vector distance on the same data set as the word embedding was trained upon, would give different values, but the ordering and relative values will strongly correlate. However, if *simc* is used on texts that differ from the trained word embedding, it indicates whether the word was used differently in this specific text. Furthermore, it is also possible to use *simc* on one word across different texts. For example, using *simc*, two texts from different domains will result in a lower similarity if the word is used differently in those domains.

The semantic folding theory [DSW15] is another method to place words in a semantic vector space [DvdSL18]. In our approach we use word embeddings, but the theoretical idea behind it and the work of Dalpiaz et al. applies to both methods.

In [DvdSL18], no exact method is given on how to calculate *simc* (a method provided by cortical.io[1] is used in the paper), so we devised our own formulation for word embeddings. Let D_1 and D_2 be text corpora from two different domains. Given a term $t \in D_1 \cap D_2$, its contexts $c_1 \subset D_1$ and $c_2 \subset D_2$ consist of all words from sentences that contain t. Furthermore, let v_w be the vector representation of a term w in a given embedding. We calculate the context similarity of t as

$$simc_t = \frac{center(c_1) \cdot center(c_2)}{\|center(c_1)\| \cdot \|center(c_2)\|}, \quad \text{where} \quad center(c) = \frac{1}{|c|} \sum_{w \in c} \mathrm{IDF}_D(w) \cdot v_w.$$

In other words, we calculate the cosine similarity of the mean vectors of all words from c_1 and c_2. This method of determining a sentence representation was previously shown to be effective for capturing and distinguishing semantic meaning [BCM+14]. In addition to this, each word w is weighted by its inverse document frequency $\mathrm{IDF}_D(w)$ from the corpus D the context was taken from.

Even though this method requires some word embedding to be trained, it is not necessary to use a word embedding trained on either D_1 or D_2, as the specific nuances of the word in those domains are represented by the formula.

4 Verification

To evaluate the described approach, we tested it on corpora that were retrieved analogously to the work of Ferrari et al. [FDG17]. We crawled topic pages of the English Wikipedia to retrieve pages from six categories: Computer Science (CS), Electronic Engineering (EEN), Mechanical Engineering (MEN), Literature (LIT), Medicine (MED) and Sports (SP). We also trained a word2vec embedding on a dump of the English Wikipedia and the created Computer Science corpus using the implementation from Gensim [ŘS10]. Then we calculated the similarity

[1] https://www.cortical.io/compare-text.html

scores of common nouns that are mentioned in [FDG17] using our method. More specifically, we used the CS corpus as D_1 and each one the other fetched corpora as D_2. Additionally, we reimplemented the approach proposed in [FDG17] to be more flexible in comparing our results with those reported by Ferrari et al. The goal in this chapter is to verify if, using the same conditions as in the work provided by Ferrari et al., our method provides a comparable ordering of terms with respect to their similarity value. This would suggest that the two approaches provide the same results.

4.1 Wikipedia crawling

The retrieval of the six corpora was done using a Wikipedia API for Python[2] by fetching pages of a category and subsequently the subcategories, until a total of 10,000 pages was reached or no all pages were found. The raw texts of every page were concatenated to create a corpus for the given category. Apart from lowercasing the text and removing all stopwords and punctuation (except for sentence boundaries), no further preprocessing was applied to the corpora. The overlap between the corpora is on average lower than 0.1% and thus negligible.

4.2 word2vec training

Using a publicly available dump of the English Wikipedia, we trained a word2vec model on the raw text that was preprocessed like the corpora described above. This resulted in a total of 74,526 unique terms. We chose a dimension of 200 and 15 epochs to train the model. Other parameters were the default values used in the implementation of Gensim [ŘS10]. We also trained a similar word embedding on the 10,000-page corpus for the topic "Computer Science".

4.3 Reimplementation

We reimplemented the approach proposed in [FDG17] with two important differences. First, we used the corpora described above, which means no lemmatization was used. On top of that, we used the 100 most common nouns for which Ferrari et al. reported similarity values, instead of determining the most frequent nouns ourselves.

4.4 Scoring

To calculate the scores of the 100 nouns, we used the formula described in chapter 3. The scores represent the calculated similarity score between the CS corpus and each of the other corpora. Table 1 shows the ten terms with highest and lowest similarity scores for each of the five corpora EEN, LIT, MEN, MED and SP, compared to CS. It is important to note, that there is no clear threshold when words are used ambiguously or not. For our examples in chapter 5 we used relative values and the rankings of the words as suggestions which words seem to be used polysemous and thus are problematic.

For example the results show that the term *file* is used in similar contexts in the CS and the EEN corpus, whereas the similarity value is much lower for non-technical categories (LIT, MED, SP). This may point to the fact that EEN and MEN refer to files of a file system in a computer, contrary to physical documents. Likewise, the term *web* has higher similarity values in LIT, MED, SP than in the more technical categories (EEN, MEN), which could be interpreted as *web* being a more nuanced term in technical areas compared to non-technical descriptions. Furthermore, the word *software* is in the ten most similar words across all corpora.

4.5 Correlation

In a second step, we compared the values yielded by our reimplementation of the method proposed by Ferrari et al. with the values they reported in [FDG17] and our results using different embeddings. More precisely, we calculated the Spearman rank correlation comparing the reimplementation with the results found in [FDG17] (Table 2a) and our results on the CS-embedding (Table 2b), the Wikipedia embedding without using IDF-weighting (Table 2c) and with the weighting enabled (Table 2d). The rank correlation coefficient ρ denotes the strength and direction of the monotonic relationship, values > 0 representing a monotonous increasing and values < 0 a monotonous decreasing relationship. The p-value represents the probability of the correlation being non-existent ($\rho = 0$), therefore $p < 0.05$ is commonly considered a desired result.

Values in Table 2a show a moderate but significant correlation between the original values and our reimplementation. Differences in the calculated values are presumably caused by not applying lemmatization while

[2] https://pypi.org/project/Wikipedia-API/

Electr. Engineering		Literature		Mech. Engineering		Medicine		Sports	
science	0.9954	**conference**	0.9799	university	0.9894	project	0.9921	science	0.9789
code	0.9899	software	0.9786	**award**	0.9852	software	0.9874	computer	0.9699
security	0.9898	data	0.9755	**conference**	0.9835	google	0.9858	software	0.9651
memory	0.9874	user	0.9750	program	0.9835	web	0.9856	**research**	0.9647
file	0.9873	computer	0.9737	programming	0.9828	interface	0.9856	human	0.9645
language	0.9870	research	0.9731	science	0.9823	computer	0.9773	data	0.9608
algorithm	0.9867	program	0.9718	algorithm	0.9822	server	0.9766	input	0.9591
database*	0.9864	google*	0.9718	model	0.9812	university	0.9765	work	0.9586
software	0.9859	database*	0.9716	data	0.9804	user	0.9763	device	0.9578
user	0.9858	web	0.9682	**software**	0.9803	security	0.9743	web	0.9572
⋮		⋮		⋮		⋮		⋮	
see	0.9619	feature	0.9137	interface	0.9439	technique	0.9321	**window**	0.8995
structure	0.9617	point	0.9103	field	0.9430	**solution**	0.9320	solution	0.8980
type	0.9605	device	0.9068	machine	0.9430	include	0.9300	**non**	0.8931
input	0.9581	solution	0.9055	**support**	0.9429	line	0.9282	network	0.8916
reference	0.9559	**non**	0.9051	include	0.9413	set	0.9261	security	0.8902
source	0.9529	algorithm	0.9032	**text**	0.9378	function	0.9254	**field**	0.8895
technology	0.9465	**set**	0.8961	input	0.9316	type	0.9249	**programming**	0.8854
game	0.9458	file	0.8915	**type**	0.9302	link	0.9241	server	0.8837
field	0.9447	**window**	0.8838	web	0.9292	**non**	0.9129	microsoft	0.8632
non	0.9238	security	0.8828	**non**	0.9190	file	0.9012	file	0.8498

Table 1: Highest and lowest context similarity scores per category, calculated by our proposed method using the Wikipedia embedding to determine word vectors. High scores mean that the word is used similar in both domains. Bold items also appear in the ten highest / lowest rated words w.r.t. the results of Ferrari et al. Items marked with * are ranked highest (lowest) by our method, but are ranked lowest (highest) by Ferrari et al.

corpus	(a) Ferrari et al.		(b) CS + IDF		(c) Wiki		(d) Wiki + IDF	
	ρ	p	ρ	p	ρ	p	ρ	p
Electronic Engineering	0.5882	<0.0001	0.1151	0.2564	0.4703	<0.0001	0.6079	<0.0001
Literature	0.4717	<0.0001	0.4028	<0.0001	0.5844	<0.0001	0.5921	<0.0001
Mechanical Engineering	0.4385	<0.0001	-0.0452	0.6569	0.5442	<0.0001	0.5736	<0.0001
Medicine	0.4877	<0.0001	0.1760	0.0829	0.5429	<0.0001	0.6164	<0.0001
Sports	0.4684	<0.0001	0.3395	0.0007	0.3839	0.0001	0.4710	<0.0001

Table 2: Spearman correlation between our reimplementation of [FDG17] and different methods on the generated corpora. For every combination, the Spearman rank correlation coefficient ρ and its p-value is given.

generating our corpora as well as by fetching pages from Wikipedia in a different order, so that our corpora might considerably differ from the ones created by another fetching method. Also, as seen in [CDLRL18], our choice of word2vec hyperparameters may have influenced the training significantly, resulting in dissimilar inter-corpora embeddings.

Column b shows that using an embedding only trained on the CS corpus leads to weak and non-significant correlations compared to the results of Ferrari et al., so the quality of the results might be sub-par, seemingly because the embedding is unable to map semantic differences sufficiently. In contrast, correlations shown in column c and d are significant, which underlines the importance of using a general-purpose word embedding. Furthermore, the IDF-weighting improved the correlation across all corpora, as can be seen in column d.

4.6 Bigger dataset

To examine the stability of our solution regarding the size of the corpora, we repeated the experiment using a limit of 100,000 pages (and a maximum subcategory crawling depth of 5) and compared the results with our findings above. Table 3 show the resulting values. We also determined the correlation between this data and the results from Table 1 (Table 4a), the results of our reimplementation on the smaller corpora (Table 4b) and the bigger corpora (Table 4c).

While Table 4 shows that the overall trend of the results is similar to that of Table 1, the correlation is much weaker for corpora SP and MEN than for MED. For example the term *text* was ranked as one of the most similar

Electr. Engineering		Literature		Mech. Engineering		Medicine		Sports	
algorithm	0.9983	google*	0.9925	game	0.9994	server	0.9954	algorithm	0.9817
game	0.9982	user	0.9914	google	0.9985	game	0.9929	google	0.9676
device	0.9965	ibm	0.9887	server	0.9979	software	0.9911	version	0.9608
database*	0.9965	software	0.9873	algorithm	0.9967	google	0.9906	software	0.9599
version*	0.9964	server	0.9864	**software**	0.9966	ibm	0.9847	science	0.9593
code	0.9959	**game**	0.9844	**image**	0.9963	database	0.9826	**user**	0.9563
ibm	0.9957	interface	0.9844	text*	0.9954	intelligence	0.9821	human	0.9558
tool	0.9954	device	0.9809	ibm	0.9945	engineering	0.9820	**research**	0.9544
project	0.9944	application	0.9792	database*	0.9941	design	0.9818	intelligence	0.9529
window	0.9942	engineering	0.9768	language	0.9936	user	0.9817	ibm	0.9513
⋮		⋮		⋮		⋮		⋮	
conference*	0.9879	order	0.9349	company	0.9685	system	0.9473	process*	0.8826
source	0.9871	known	0.9327	list	0.9669	case	0.9468	state*	0.8808
solution	0.9868	point	0.9325	tool	0.9668	**code**	0.9460	known	0.8806
process	0.9868	text	0.9279	**application**	0.9667	link	0.9458	point	0.8797
model	0.9864	**set**	0.9243	search	0.9648	**solution**	0.9454	conference	0.8769
type	0.9848	**support**	0.9223	machine	0.9604	window	0.9431	**form**	0.8679
theory	0.9843	language	0.9162	design	0.9593	type	0.9340	**set**	0.8678
structure	0.9828	search	0.9097	**support**	0.9550	function	0.9303	server	0.8637
field	0.9822	**memory**	0.9067	source	0.9544	memory	0.9269	application	0.8591
programming	0.9703	**window**	0.8934	**type**	0.9538	**support**	0.9247	**field**	0.8584

Table 3: Highest and lowest similarity scores per category, calculated on corpora from up to 100,000 Wikipedia pages. Bold items also appear in the ten highest / lowest rated words w.r.t. the results of Ferrari et al. Items marked with * are ranked highest (lowest) by our method, but are ranked lowest (highest) by Ferrari et al.

corpus	(a) 10k Wiki + IDF		(b) 10k Ferrari		(c) 100k Ferrari	
	ρ	p	ρ	p	ρ	p
Electronic Engineering	0.4606	<0.0001	0.2727	0.0063	0.2668	0.0076
Literature	0.5088	<0.0001	0.4699	<0.0001	0.7027	<0.0001
Mechanical Engineering	0.4090	<0.0001	0.4903	<0.0001	0.7057	<0.0001
Medicine	0.7396	<0.0001	0.5867	<0.0001	0.6835	<0.0001
Sports	0.3596	0.0003	0.2850	0.0047	0.6284	<0.0001

Table 4: Spearman correlation between the results from Table 3 and different methods. For every combination, the Spearman rank correlation coefficient ρ and its p-value is given.

words in MEN for the smaller corpus size, while it is one of the least similar for the 100,000 page corpus. This could be caused by the smaller corpora representing the categories only partially and using more Wikipedia pages led to a more general representation of the individual topics. Interestingly, our results on the EEN corpus are notably different from the ones our reimplementation produced (Table 4b-c). For EEN, a weaker correlation was determined, whereas a strong monotonic relationship was found for the other corpora, especially with the results of our reimplementation on the bigger corpora. These results can suggest the instability of word embeddings in general or word embeddings with smaller corpus size. In conclusion, this creates the need for a more thorough investigation in the future.

5 Discussion

Comparing the method from [FDG17] to the method presented in this work, the new method proves to be more flexible and needs less initial effort. This stems mainly from the fact that no separate word embeddings need to be trained for specific domains. The results in chapter 4 suggest that the best results can be achieved with a general purpose word embedding as a basis. Thus, in an application it would be sufficient to provide the two texts from different domains or sources and the domain-specific differences can be calculated. Notable are the following advantages:

1. Since there is no need for word embeddings, the differences can be calculated even on small data sets that would not provide enough data for own word embeddings. This means differences in niche domains,

P_1 vs. P_3		P_1 vs. P'_3		P_1 vs. P_2		P_3 vs. P'_3		P_2 vs. P_3		P_2 vs. P'_3	
bieten	0.9874	bieten	0.9884	system	0.9717	verbund	0.9940	bieten	0.9705	ermöglichen	0.9696
möglichkeit	0.9874	möglichkeit	0.9881	bieten	0.9690	service	0.9938	möglichkeit	0.9705	möglichkeit	0.9693
nutzer	0.9771	nutzer	0.9770	möglichkeit	0.9690	möglichkeit	0.9933	nutzer	0.9691	bieten	0.9689
fähig	0.9422	fähig	0.9622	nutzer	0.9639	bieten	0.9932	ermöglichen	0.9584	bereitstellen	0.9685
chat	0.9397	konfigurieren	0.9618	stanag	0.9588	nutzer	0.9931	bereitstellen	0.9510	nutzer	0.9655
⋮		⋮		⋮		⋮		⋮		⋮	
mission	0.9177	anzeigen	0.9052	ermöglichen	0.9343	informationen	0.9251	services	0.9008	maximal	0.9101
automatisch	0.8941	durchzuführen	0.9012	bereitstellen	0.9258	endgerät	0.9148	gemäß	0.8966	beim	0.8966
informationen	0.8637	entsprechend	0.8961	informationen	0.9250	clients	0.8703	mobilen	0.8911	services	0.8929
service	0.8625	nutzung	0.8899	nato	0.9070	planning	0.8701	informationen	0.8730	priorisierung	0.8717
durchzuführen	0.8540	service	0.8750	service	0.8978	durchzuführen	0.8436	plattform	0.8621	plattform	0.8269

Table 5: Highest and lowest context similarity scores of a pairwise comparison of four requirement datasets P_1, P_2, P_3 and P'_3, where the latter two originate from a single project with requirements split in two parts.

subdomains as well as in different requirements sets can be compared to each other.

2. The word differences can be calculated for every word. In [FDG17] the words have to be marked before the word embeddings are calculated. This means that either the most frequent words are taken (as is done in the paper), or they have to be marked manually. However, this would require previous knowledge about which words may have different interpretations in those domains. Additionally, the more words are marked, the more tokens will be different between the domains. Thus, less words will be common during training and the domains will completely separate from each other, resulting in less expressive distances. This is not desirable.

One disadvantage we found while comparing the two methods is the calculation time. Determining the distance of a word in two domains in [FDG17] requires two look-ups in the word embedding model and a simple distance calculation between two vectors. The presented method, however, needs to determine the whole context of the word in both domains, meaning all instances of the word have to be found and the context words in the sentence have to be collected. This then requires multiple look-ups in the model, the calculation of the average vector and a distance calculation. This results in approximately 120 ms per call when comparing two 10,000-page corpora from above, and approximately 1800 ms for the bigger 100,000-page corpora, but only about 4 ms in requirement texts with around 800 requirements. In comparison, the method devised in [FDG17] is able to calculate the distance in less than a millisecond.

Turning to our previously described example, we compared the distances of words in four different requirement sets. As all those requirement sets were related to each other, but discussed different parts of connected projects, there should be multiple words that are shared between those sets, while also containing some differences in their usage. The notation in Table 5 shows the results between those four data sets. P_1 describes the overarching military project that will connect multiple sub projects, like P_2 and P_3. The requirements in P_2 describe the infrastructure for a mobile network that has to be able to connect multiple devices in areas without internet connection. P_3 describes devices that will work in the infrastructure provided by P_2. Additionally, P'_3 is a small part in P_3 that was focused separately from the rest of P_3.

The results show that in most comparisons the word *service* has one of the lowest similarity scores. Upon further investigation of the requirement sets, some differences can be seen. In P_1 the word is mainly used on which services should actually be used and supported for the project, while P_2 describes which features the services shall integrate. The requirements for P_3, however, specify which features for the user shall be provided and how they should be able to use them. This indicates that P_1 and P_2 are emphasizing the technical aspects and details of the services more, while in P_3 the user interaction aspect of services are emphasized. In P_2, the system uses the services, while in P_3 humans use the services. The word *clients* (which in this context was used as a term for a computer device) had a high difference between P_3 and P'_3, as one part was focused on what features shall be provided by the client, while the other part specified details as the maximal weight or battery runtime. In one case the client was used as a pure hardware product, while the other part specifies the software part of the client. The same semantic difference was found with the word *plattform* which had a low similarity between P_2 and P_3.

However, the results provided in this experiment may have some shortcomings. In section 4 we were able to show that our method provides the same results as a previously established method on text corpora that are big in comparison to this experiment. There was no qualitative study on how good the results are on smaller texts, or whether smaller texts even contain enough information to show discrepancies in word usage.

6 Conclusion

In this paper we discussed the potential problems that may arise from polysemous words in requirements. Those different interpretations of a word can lead to confusion when working with requirement sets. Different methods were suggested that solve this problem by comparing domain specific word embeddings. However, it is not always possible to train word embeddings due to text limitations. The method proposed in this work is able to calculate domain specific differences by comparing the two different text corpora without training word embeddings. This results in a more flexible method that needs less initial work. We verify our work by comparing it to a previously established method in this field and prove that our method, albeit a bit slower in calculation, provides similar results. In our experiment we show the usefulness of our method, as it is able to calculate word differences even on requirement sets that contain less than 1000 sentences. In the future we would like to conduct a qualitative analysis of the usefulness of our results. This means the method needs to be integrated into a tool for users to interact with. Additionally we want to see how users can implement this information to find differences in the interpretation of polysemous words.

References

[BCM+14] Carmen Banea, Di Chen, Rada Mihalcea, Claire Cardie, and Janyce Wiebe. Simcompass: Using deep learning word embeddings to assess cross-level similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 560–565. Association for Computational Linguistics, 2014.

[BGJM16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[CDLRL18] Hugo Caselles-Dupré, Florian Lesaint, and Jimena Royo-Letelier. Word2vec applied to recommendation: Hyperparameters matter. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 352–356, New York, NY, USA, 2018. ACM.

[DDF+90] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[DSW15] Francisco De Sousa Webber. Semantic folding theory and its application in semantic fingerprinting. *arXiv preprint arXiv:1511.08855*, 2015.

[DvdSL18] Fabiano Dalpiaz, Ivor van der Schalk, and Garm Lucassen. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 119–135. Springer, 2018.

[FDE+17] Alessio Ferrari, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi, and Stefania Gnesi. Natural language requirements processing: A 4d vision. *IEEE Software*, 34(6):28–35, 2017.

[FDG17] A. Ferrari, B. Donati, and S. Gnesi. Detecting domain-specific ambiguities: An nlp approach based on wikipedia crawling and word embeddings. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 393–399, Sep. 2017.

[FEG18] Alessio Ferrari, Andrea Esuli, and Stefania Gnesi. Identification of cross-domain ambiguity with language models. In *2018 5th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 31–38. IEEE, 2018.

[LDvdWB16] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *International Conference on Conceptual Modeling*, pages 463–478. Springer, 2016.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[ŘS10] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[WNLN18] Wentao Wang, Nan Niu, Hui Liu, and Zhendong Niu. Enhancing automated requirements traceability by resolving polysemy. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 40–51. IEEE, 2018.

[WV16] Jonas Winkler and Andreas Vogelsang. Automatic classification of requirements based on convolutional neural networks. In *Requirements Engineering Conference Workshops (REW), IEEE International*, pages 39–45. IEEE, 2016.