Requirements Similarity and Retrieval

Muhammad Abbas[1,2] ✉, Sarmad Bashir[1,2], Mehrdad Saadatmand[1], Eduard Paul Enoiu[2], and Daniel Sundmark[2]

[1] RISE Research Institutes of Sweden, Västerås, Sweden, {first.last}@ri.se
[2] Mälardalen University, Västerås, Sweden, {first.middle.last}@mdu.se

Abstract. Requirement Engineering (RE) is crucial for identifying, analyzing, and documenting stakeholders' needs and constraints for developing software systems. In most safety-critical domains, maintaining requirements and their links to other artifacts is also often required by regulatory bodies. Furthermore, in such contexts, requirements for new products often share similarities with previous existing projects performed by the company. Therefore, similar requirements can be retrieved to facilitate the feasibility analysis of new projects. In addition, when a new customer requests a new product, retrieval of similar requirements can enable requirements-driven software reuse and avoid redundant development efforts. Manually retrieving similar requirements for reuse is typically dependent on the engineer's experience and is not scalable, as the set could be quite large. In this regard, applying Natural Language Processing (NLP) techniques for automated similarity computation and retrieval ensures the independence of the process from the human experience and makes the process scalable. This chapter introduces linguistic similarity and several NLP-based similarity computation techniques that leverage linguistic features for similarity computation. Specifically, we cover techniques for computing similarity ranging from lexical to state-of-the-art deep neural network-based methods. We demonstrate their application in two example cases: a) requirements reuse and b) requirements-driven software retrieval. The practical guidance and example cases presented in the chapter can help practitioners apply the concepts to improve their processes where similarity computation is relevant.

Keywords: Requirements similarity · Requirements retrieval · Requirements reuse · Software reuse · NLP

1 Introduction

Requirements Engineering (RE) is a crucial system/software engineering activity for eliciting and documenting stakeholder needs [37]. Enabling a smooth execution of the RE activities requires maintaining and managing requirements. Furthermore, most safety standards require demonstrating the traceability of requirements across various abstraction levels; therefore, the need to manage requirements is even more prevalent when engineering safety-critical systems.

Various RE activities, such as requirements retrieval and traceability link recovery, require finding similar requirements for reuse or impact analysis. Finding similar requirements manually is time-consuming and could be subject to fatigue effect [12].

In most settings, the requirements are predominantly expressed in a natural language. Therefore, linguistic similarity serves as a bridge to connect multiple aspects of natural language requirements, ranging from lexical attributes, syntactic structures, and semantic relations. These linguistic characteristics are architectural keystones for various NLP-based RE applications. From text classification to summarization and information retrieval to clustering, linguistic similarity enables understanding language nuances and interpreting meaningful insights from vast amounts of textual data. For instance, search engines rely on linguistic similarity measures to retrieve relevant documents by comparing their linguistic characteristics to a user query. Typically, intermediate representations of the input text are used to compute *similarity* to perform various tasks. Different similarity measures that quantify the degrees of closeness between two elements are used for similarity computation. In this context, multiple metrics such as cosine [42], Jaccard similarity index [52], and edit distance [44] can be employed to quantify the similarity among multiple textual fragments, e.g., requirements. In Section 2.3, we discuss them in detail with examples.

In the domain of RE, the similarity among requirements at a lexical or semantic level is a crucial enabler for supporting various RE activities, leveraging NLP and machine learning (ML) techniques. In particular, classification and content-based recommender systems are often exploited, such as identifying and allocating requirements [10,9,4] and recommending the reuse of existing requirements [1,3]. Moreover, requirements similarity is also exploited in other RE tasks, such as trace link recovery [32,49] for ensuring continuity, identification of equivalent requirements [29], and change impact analysis [15,6]. In this regard, Ilyas et al. [36] propose a framework *SimReq* to support design and code reusability based on similarity metrics such as Dice, Jaccard, and Cosine coefficient. In another work, Dag et al. [22] propose the ReqSimile tool based on recommender systems to retrieve previous requirements from a large industrial dataset for reuse. The tool is based on the Term Frequency-Inverse Document Frequency (TFIDF) model and cosine similarity measure [39]. Such example use cases benefit from applying NLP techniques and similarity computation methods to augment the RE process.

With the benefit of retrospective insights, this chapter aims to offer multiple approaches to address the problem of requirements similarity and retrieval, leveraging various NLP techniques. The chapter introduces similarity from the lens of linguistics and presents various approaches used to represent requirements for similarity computation in Section 2. In Section 3, we demonstrated the applicability of various requirements similarity pipelines in two relevant RE tasks inspired from real-word cases: a) requirements reuse and b) requirements-driven software reuse. Section 3 also presents the procedure for applying the similarity computation pipelines to perform the two considered tasks. In Section 4, we presents and discusses the obtained results. Finally, Section 5 concludes the chapter with future directions.

2 Linguistic Similarity

The scientific study of a language referred to as linguistics, is a complex and multifaceted field of study [33]. Conventionally, the linguistic field analyzes multiple components of the natural language, such as phonology, syntax, morphology, semantics, and pragmatics. In this context, phonology studies speech structure based on speech unit patterns and pronunciation rules to identify the words in a language [17]. Syntax and morphology focus on the arrangement and structure of meaningful linguistic elements, including words and morphemes. In particular, morphology studies morphemes that are the smallest building blocks and provide meaningful constituents for a linguistic expression [34]. It includes base words, such as *hat* and *swim*, alongside affixes, like *“inter”* in the term *interface*, the plural *“s”* or *“es”*, and the past tense *“ed”*. Furthermore, syntax refers to how individual words and their basic meaningful units—morphemes— are combined to form larger units, such as sentences and phrases, based on grammar rules and constraints [55]. The enrichment of a language makes it more challenging to comprehend the actual meaning conveyed, even with employed grammatical structures. Consequently, the sub-field semantics is concerned with understanding the underlying meaning within sentences or phrases [23]. In the context of speech, communicating sentences or phrases not only depends on structural and linguistic knowledge but also on the context of the utterance. The study of how sentences have been used in speech to convey their intended meaning is called pragmatics [13].

In linguistics, the notion of linguistic similarity serves as an essential catalyst to comprehend various facets of a natural language. As aforementioned, the natural language exhibits different linguistic dimensions incorporating a diverse set of structures, rules, and expressions. In essence, linguistic similarity analyzes the relationships and patterns within this set. It delves deeper into the attributes of a language to comprehend shared features, create associations, and indicate variations. Usually, similarity is quantified in textual data at a lexical or semantic level. They serve as a fundamental to multiple NLP-related applications [53]. Multiple methods are employed to learn the lexical characteristics, focusing on surface-level attributes of terms. A comparison between two text fragments shows lexical similarity when they share considerable common words or phrases. However, lexical methods do not attribute to the variability of word meanings or that different words can represent a similar concept. Other similarity methods delve deeper into natural language semantics to address this drawback. For example, given two requirements: *“The user should be able to login into the system”* and *“System must allow the user to login”* that are semantically related as they represent the same underlying concept: *enable user login.* Semantic similarity methods determine the ranking or degree of semantic relatedness among multiple data points [20]. Therefore, semantic approaches can capture syntactically different but semantically similar inputs like the example requirements described above.

The process of computing linguistic similarity for various NLP tasks, including requirements retrieval and reuse, can be categorized into the following steps: data pre-processing, data representation, similarity computation methods, and performance evaluation of the developed NLP pipelines. We discuss these steps below.

2.1 Data Pre-processing

To perform NLP-related tasks, it is essential to establish or learn a *language model* that provides a statistical representation of words within vector space [11]. Such language models consider words as real-valued vectors, later employed as features for NLP-related similarity tasks. Therefore, a preliminary crucial step is pre-processing the natural language-related artifacts in a structured manner for later analysis. This is because the raw textual data, such as textual requirements, are often unstructured and contain noisy elements like punctuation, special characters, and inconsistent capitalization. In many cases, commonly employed pre-processing techniques include the removal of stop words, also known as common words that have minimal semantic value to the analysis, such as “the”, “is”, etc, and the lemmatization technique that performs the morphological analysis and reverts the words to their base or dictionary form. In addition, part-of-speech (POS) tagging, word segmentation, and tokenization are other techniques that could be applied to NL requirements to achieve a desired format [30,8,43].

The structured format required from the pre-processing step depends on the subsequent data representation technique for analysis of the NLP task at hand. Particularly, the selection suite of pre-processing techniques depends on the *unit of analysis*, which could comprise words, sentences, phrases, and paragraphs. Based on the unit under analysis, the features are represented in a vector space, describing the unit’s characteristics.

2.2 Data Representation

A foundational aspect of enabling similarity computation for multiple NLP tasks is data representation in a format that ML algorithms can process and comprehend. In this regard, the data is often represented as “features” in a multi-dimensional vector space, enabling semantic and syntactic interpretation of the data [38]. For example, lexical approaches often use statistical measures to extract a representation vector from NL requirements. On the other hand, learning-based approaches such as word embeddings use neural networks to learn a “language model” from the corpus [43]. Such language representation techniques capture nuances beyond the lexical level and are classified as “semantic” approaches. First, we discuss some classical representation methods, followed by widely adopted techniques for different NLP-related requirements similarity tasks.

Categorical Representation. Such representation methods are considered straightforward because of their direct approach to transforming the textual data into vector formats using binary indicators “0” or “1”. The two common techniques are one-hot encoding and Bag-of-Words (BoW) models. Both are briefly discussed below.
One hot encoding is a traditional method of encoding words as spare binary vectors, where the dimension of the vector is the same as the vocabulary size. The resultant vectors are binary, meaning that each word vector comprises zeros and ones, where zero represent the absence and one represent the presence of a word in vocabulary. The one-hot encoding method does not capture the context or semantics between words of a text fragment.
Bag of Words (BoW) is an extension of one hot-encoding, which represents the words in a sentence as a collection and maintains the frequency of each word [57]. However, the BoW technique disregards the sequence of the words, grammar, and their semantic relationship. For example, consider the following two requirements: R1 = *“The system shall encrypt data.”* and R2 = *“The system must secure user data.”*. Utilizing a BoW model with the vocabulary of {*the, system, shall, encrypt, data, must, secure, user*}, the vector representations could be as follows: V1 = *[1,1,1,1,1,0,0,0]* and V2 = *[1,1,0,0,1,1,1,1]*, where the length of each individual requirement vector is equal to the vocabulary size, with one position in the vector to score each word. As a result, the vectors may become sparse because of the increased vocabulary size and possible large number of “0s”, maintaining no order of words in the requirements.

Weighted Representation. The weighted representation techniques capture more information than simple binary or count-based representations, like one-hot encoding and BoWs models, which assign equal weightage to each word. Weighted representation models allocate different importance to words based on specific defined criteria. A commonly used weighted word representation technique is discussed below.
Term Frequency-Inverse Document Frequency (TFIDF): A corpus-based approach that evaluates how relevant a word is in a document compared to the collection of documents [5]. The TFIDF technique operates in two parts: TF measures the occurrence of a word in a single document, which can be referred to as a text fragment or requirement. As a single word can appear multiple times in large requirements compared to smaller ones, the TF of a word is computed by dividing it by the total words in a requirement. For each word w in requirement r:

$$TF(w, r) = \frac{\text{Number of times } w \text{ appears in } r}{\text{Total number of } w \text{ in } r}$$

The second part of the technique is IDF, which reduces the effect of common words such as “the”, “and” etc. IDF values the rarity of the words in a corpus and assigns more weight to words that occur less frequently across the pool of documents. IDF is calculated across the entire corpus, therefore, for a given word

w and a corpus with N requirements:

$$IDF(w) = \log\left(\frac{N}{\text{Number of requirements containing } w}\right)$$

Given that, the TFIDF score for each word w in every requirement r is the product of TF and IDF:

$$TFIDF(w, r) = \text{TF}(w, r) \times \text{IDF}(w)$$

The TFIDF algorithm considers the lexical aspects of the input requirements and derives the term matrix. Based on the feature matrix, feature vectors for individual requirements can be extracted. It is important to highlight that the TFIDF approach can not capture words' semantics and syntactical information but rather is only weighting the terms.

Distributed Representation. The conventional feature learning techniques, such as categorical and weighted, do not capture the semantics and syntactic meaning between the words and encounter issues with the high dimensionality of vector space. This challenge prompted researchers to explore distributed word representations in a lower-dimensional space. The distributed representation of words, commonly known as *word embeddings*, can capture the global context and semantic relationships between terms of a text fragment [51]. Such representation of words in a vector space captures linguistic regularities and patterns that have demonstrated exceptional performance in NLP tasks. The fundamental concept behind distributed representations is that words with similar meanings must appear closer to each other in vector space. For instance, a well-known example of this is deriving a resultant vector by computing vec("king") - vec("man") + vec("woman"), where the closest vector typically corresponds to vec("queen") among all the other vectors in an embedding space [41]. Below, we discuss seminal distributed word representation techniques.

Continuous Bag of Words (CBOW) language model aims to predict the target word vector based on its context that constitutes preceding and proceeding words. The CBOW architecture consists of a shallow feed-forward natural network consisting of three layers. The first layer consists of the context words that result in an average word vector of fixed length, regardless of the context size, and is projected to the middle hidden layer. Finally, the last layer correlates the output and improves the representation based on the backpropagation of the error to predict the middle word based on its context words.
Skip-Gram (SG) model for representing words as feature vectors are opposite in approach to CBOW but similar in the architecture. The SG language model estimates the context words given the current word as input. The first input layer represents the current word and the second projection layer for predicting a range of context words. This range is defined as the window size representing the words to be considered for prediction. The window slides across the given input sentence to calculate the likelihood of context words given a specific target

word [40].

The CBOW and SG are Word2Vec models that capture the semantic relationship between words but struggle to comprehend the polysemy—same word with multiple meanings—of words. For example, unless specifically trained as a single vector, both CBOW and SG models treat words like *"Windows 11"* as distinct vectors *"Windows"* and *"11"*, which results in only taking into account the local context of the words. As a result, this will lead to less accurate predictions and poor performance on NLP tasks. To address such issues, the researchers proposed two enhanced distributed representation algorithms, i.e., GloVe and FastText, which are commonly used in multiple NLP tasks. Below, we discuss them briefly.

GloVe language model is based on unsupervised learning to capture the global statistical information of words. It is designed to combine global matrix factorization and local context window-based methods to leverage the best of both approaches in creating word embeddings [45]. GloVe creates a co-occurrence matrix consisting of statistics of words, where each element X_{i_j} represents how often word i appears in the context of the word j in an appropriate context window. This is followed by the factorization to get the word vectors. GloVe pre-trained embeddings[3] are available in different dimensions, which are trained on huge amounts of the corpus.
FastText (FT) is an extension of the SG language model that enriches the dense word vectors with sub-word information. FT learns the structural information of words to provide efficient word representation [14]. It treats each word as a composition of n-grams, capturing the morphology of words and generating embeddings for those words not seen during training, commonly referred to as out-of-vocabulary (OOV) words. For example, given the word *"system"* and window size of 3, FT model represents the word as <*sy, sys, yst, ste, tem, em*> character n-grams, including special boundary symbols. In this way, the FT model can better capture the internal structure of words and understand both suffixes and prefixes. In addition, FT treats each word as another feature, and the final representation of the word is the sum of its n-grams vectors and word vectors. This ensures that both the morphological and semantic properties of words are captured. FT library[4] provides word representation models for different languages and is widely used in NLP-related similarity and classification tasks.

Contextual Representation. While contextual embedding models fall within the scope of distributed representation, they are discussed separately because, unlike methods such as FastText and GloVe that produce static word vectors, the models based on deep learning and attention mechanism yield word representation, which vary depending on surrounding context or the specific usage of the word. Context-aware embedding methods create vectors that more ef-

[3] `https://nlp.stanford.edu/projects/glove/`
[4] `https://fasttext.cc/`

fectively capture a wide range of linguistic features, including polysemy and complex syntax. Below, we discuss the most seminal of the context-aware embedding techniques.
Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained transformer network based on encoder architecture, which considers positional and contextual information of words to capture semantics better. The pertaining objective of BERT is masked language modeling and next-sentence prediction. BERT is trained on BooksCorpus and the English Wikipedia with 2,500M words. It achieved state-of-the-art results in multiple NLP downstream tasks such as question-answering and sentence classification. However, a significant drawback of the BERT network is that it does not compute independent sentence-level embeddings. Researchers have adopted another approach to overcome this limitation: providing an input sentence to the BERT network and obtaining a fixed-sized sentence-level vector. This can be achieved by averaging the outputs—similar to averaging word embeddings—or using the output of a special [CLS] token. The [CLS] stands for *classification token*, which is added at the start of the input sentence, and its final hidden state is often used as a sentence-level representation for classification tasks [21].
Sentence-BERT(SBERT) is a variant of BERT, which directly generates sentence-level embeddings from the network. The SBERT's architecture is based on the siamese network, which is fine-tuned to ensure semantically similar sentences are closer together in the embedding space [48]. A mean-pooling operation is added to the network to get fixed-size sentence embeddings. SBERT is trained on Standford Natural Language Inference (SNLI) [16] and Multi-Genre SLI [56] datasets, which consist of one million sentences combined with the labels neutral, contradiction and entailment. The ability of SBERT to generate meaningful sentence embeddings makes it well-suited for various NLP tasks such as sentence-level similarity and information retrieval. SBERT has shown state-of-the-art performance compared to the other sentence embedding models when tested on semantic textual similarity datasets, i.e., STS benchmark and SemEval STS tasks 2012-2016.
Universal Sentence Encoder (USE) is based on Deep Averaging Network (DAN) architecture to represent the text as vectors and capture both the semantic and contextual meaning. It has been trained on large and diverse datasets, which include different sources on the web, including books, articles, and discussion forums [19]. USE encodes sentences or phrases and is widely used for text classification and semantic similarity tasks.

Takeaway: Data Representation

Representing data as feature vectors is essential for NLP tasks involving similarity computation. Contextual embedding techniques provide considerable advantages for tasks where linguistic context plays an important role. However, the selection of embedding techniques depends on the trade-off between linguistic representation and computational efficiency. Moreover, it is also important to evaluate simpler static embedding techniques that may suffice for projects with lower resources and can also be considered as baselines for comparison.

2.3 Similarity Measures

In the context of textual requirements, similarity refers to the degree to which two or more entities exhibit proximity in specific characteristics. Characterizing similarity measures for different RE tasks depends on how the data is represented as features. Therefore, various similarity metrics are available and can be applied to quantify the degree of closeness at the syntactic or semantic level. This chapter only uses the cosine similarity metric in its pipelines. Nevertheless, for the readers, we provide a brief overview and comparison of some of the most seminal similarity metrics below.

- *Dice* is a non-parametric similarity measure used to compare the overlap between two sets of vectors. The Dice metric is particularly useful when comparing sets with significantly different sizes. It normalizes for vector lengths by dividing the sum of non-zero entries and filtering out empty elements that do not contribute to understanding the similarity between two sets [25].
- *Edit distance* is a class of similarity metrics based on the dissimilarity between input data points. Edit distance-based metrics could be applied to raw text data points and work based on quantifying the number of edits required to make the two inputs similar [24]. A widely used metric based on edit distance within software engineering is Levenshtein distance [44]. It is often used for text similarity and spell-checking tasks.
- *Jaccard Similarity Index (JSI)* can work both on the syntactic level and with representation vectors. However, unlike edit distance, JSI is based on the ratio of common terms over their union. JSI is commonly used in document clustering and information retrieval tasks for measuring the similarity between given data elements [52].
- *Euclidean distance* is a metric based on quantifying a straight-line distance between two data points in multi-dimensional feature space. It is often used for tasks like dimensionality reduction [27] and recommender systems to measure the distance between items and users to identify similar items for personalized recommendations [7].
- *Cosine similarity* metric is a widely used distributional measure among multiple NLP applications [42]. It calculates the cosine angle formed between

Table 1: Comparison of different similarity computation metrics

Metric	***Range***	***Pros***	***Cons***
Dice	[0, 1]	1. Effective for binary and sparse datasets. 2. Emphasize the relative importance of shared data items.	1. Less effective for continuous or non-binary data. 2. Sensitive to duplicate elements.
Edit distance	[0, 1]	1. Effective for comparing two sets where the order of elements is essential. 2. Offer a granularity measure of similarity.	1. Less effective for comparing numerical vectors. 2. Ignore semantics between data items.
JSI	[0, 1]	1. Effective for sparse data. 2. Focuses on data items overlap ratio.	1. Sensitive to the size of data items. 2. Ignores the order of data items.
Euclidean distance	[0, ∞]	1. Captures both magnitude and direction differences between vectors. 2. Suitable where each sentence vector dimension equally affects the distance.	1. Sensitive to high dimensional vectors. 2. Can be influenced by outliers in high-dimensional vector space.
Cosine Similarity	[-1, 1]	1. Invariant to the magnitude of vectors. 2. Less sensitive to individual outliers. 3. Suitable for high-dimensional vectors.	1. Do not capture differences in magnitude. 2. Assumes all dimensions are independent.

two vectors in multi-dimensional feature space and does not depend on their magnitude. The similarity score between two non-zero vectors ranges from -1 to 1, with -1 representing opposite vectors, 0 representing no similarity (orthogonal vectors), and 1 indicating perfect similarity (proportional vectors). In the case of non-negative vectors, the cosine similarity ranges between 0 to

1. Notably, the efficacy of similarity computed through cosine relies on the employed language model for measuring feature vectors.

Table 1 compares the above-mentioned similarity metrics. The defined ranges are generic and can vary depending on the data types and normalization techniques.

Read and `execute Jupyter Notebook` to explore different similarity metrics on an example.

2.4 Performance Measures for Evaluation

In the context of RE, particularly for requirements similarity and retrieval tasks, evaluating NLP pipelines involves analyzing specific datasets comprising numerous requirement artifacts. Each NLP approach could recommend multiple requirement candidates based on the similarity computation techniques. Therefore, it is important to determine the effectiveness of NLP pipelines indicating superior performance. This could be achieved by adopting several metrics based on the problem characterization. We provide a brief overview below.

Filtering: For the task of filtering/retrieving requirements, especially in the scenario of requirements reuse, NLP pipelines are often evaluated on precision, recall, and F1 score metrics. This is because such performance measures provide a balanced view of the developed approach's effectiveness in identifying relevant requirements [28]. *Precision* calculates the ratio of correctly identified requirements to the total number of requirements flagged as relevant, whereas *Recall* evaluates the ratio of correctly identified relevant requirements to the total set of relevant requirements *F1 score* is the harmonic mean of *precision* and *recall* metric, which is useful to find a trade-off between retrieving maximum relevant requirements—achieving high *recall*—while ensuring the relevance of retrieved requirements—maintaining high *precision*—in NLP pipelines. In this chapter, we provide a detailed analysis of the requirement reuse pipelines based on precision, recall, and F1 score.

In scenarios like retrieving software for reuse through similar requirements and assessing the performance of the NLP pipeline in light of human judgment, evaluating the significance of their interrelationship is critical. In this regard, correlation methods like Pearson and Spearman's rank coefficient can facilitate quantitative similarity analysis in refining and evaluating the retrieval process [59]. Pearson correlation quantifies the linear relationship between the ranks of retrieved items and ground truth ranks, while Spearman correlation assesses the monotonic relation between the ranked order of retrieved items and their ground truth, regardless of exact values. In this chapter, we demonstrate the use of correlation coefficients on multiple datasets for different cases, with further details available in the upcoming section.

Ranking. After filtering the requirements, the volume of potential requirement pairs could be significantly large; therefore, it is important to prioritize or rank the requirements based on their equivalence. In this regard, *Lag* metric can quantify the effectiveness of ranking requirements in NLP-based retrieval pipelines. In a nutshell, the *Lag* metric calculates, on average, the quantity of non-equivalent requirement pairs ranked higher in similarity than the true equivalent pairs [29]. A lower *Lag* value indicates the higher effectiveness of a retrieval pipeline. This chapter does not focus on ranking.

Takeaway: Performance Measures for Evaluation

Different metrics serve different purposes. Therefore, selecting performance measures for evaluation depends on the NLP application. Considering the data and task characteristics to evaluate NLP approaches is important.

3 Considered Cases and Procedure

This chapter leverages the application of similarity analysis in two RE example cases—presented in Sub-Section 3.1—to demonstrate their relevance in the field. These cases are performed using various similarity measuring pipelines presented in Sub-Section 3.2. The cases are demonstrated on data collected from industry and the Semantic Textual Similarity (STS) benchmark dataset [18], presented in Sub-Section 3.3.

3.1 Example Cases

Companies often do not develop products from scratch but reuse existing components from existing projects. In contexts like these, when a new project is to be delivered to a new customer, finding reuse opportunities for existing software based on similar requirements could save time and boost confidence in the end products. Reusing existing software increases confidence in the end product because they have already been tested and proven in other products before [2]. In addition, it reduces the development, certification, and testing time. In such cases, to aid reuse, new requirements are compared to requirements from existing projects to recommend reuse and avoid redundant development efforts. This chapter demonstrates similarity-driven requirements retrieval for two cases with the RE domain as follows.

– *Case 1: Requirements reuse* is focused on identifying similar requirements. This case is focused on demonstrating the applicability of automated similarity computation pipelines on a public dataset. In addition, this case demonstrates how to evaluate the computed similarity in light of human-rated similarity using relevant metrics, such as precision, recall, and their harmonic mean (F1 score).

- *Case 2: Requirements-driven software retrieval for reuse* goes beyond identifying similar requirements and looks into the relevance of the retrieved software. This case demonstrates how similar requirements could be used to retrieve software for reuse. Furthermore, this case evaluated the similarity computation pipelines in light of the relevance of the retrieved software by correlating requirements similarity to software similarity. This case is demonstrated on both an industrial and a public dataset.

3.2 Similarity Computation Pipelines

In this chapter, we employ various pipelines comprising data representation techniques for similarity computation and evaluation of RE tasks, i.e., requirements reuse and requirements-driven software retrieval for reuse. These techniques range from weighted representation (TFIDF) to distributed methods (GloVe, FastText) and extend to contextual models (USE, BERT, SBERT). We consider the following data representation approaches (discussed in Section 2) with cosine similarity metric to construct pipelines for similarity computation:

- *Lexical and Weighted:* From the traditional lexical approaches, we consider the string-level JSI and TFIDF-based pipelines
- *Distributed (word2vec):* We consider two seminal distributed representation pipelines, GloVE (GLV) and FastText (FT) from the word2vec family.
- *Contextual:*
  - *Vanilla BERT*: We consider pipelines that utilize representation obtained from both the BERT with averaging strategy (BERT-Avg) and BERT with sentence token embedding (BERT-CLS).
  - *Sentence BERT (SBERT)*: We considered two specific variants of the SBERT model, namely *stsb-roberta-base-v2*[5] and *all-MiniLM-L6-v2*[6], which we have labeled as ST-Roberta and Mini-LM, respectively. Both variants are explicitly designed for clustering and semantic search tasks.
  - *Universal Sentence Encoder (USE)*: We also consider pipelines based on USE, which is trained on semantic textual similarity tasks.

As pre-processing might impact the pipeline's performance, we also consider two pre-processing configurations of the pipelines. In particular, we consider computing similarity using each similarity pipeline with no, basic[7], and full pre-processing[8]. In the rest of this chapter, pipeline variants with basic and full pre-processing are distinguished with names starting with "b" and "p", respectively.

[5] STSb-roberta-base-v2, available online at Hugging Face

[6] Mini-LM, available online at Hugging Face

[7] In basic pre-processing, the text is converted to lower case, and special characters are replaced with a blank space that is followed by lemmatization.

[8] Full pre-processing uses basic pre-processing and also removes stop-words.

Table 2: Considered datasets (Pre-processing (P.))

Dataset		Rows	No P.		Basic P.		P.	
			Words	AVG. Words	Words	AVG. Words	Words	AVG. Words
STS	Dev. + Test	2173	52646	12.11	52652	12.12	30733	7.07
Industrial	Project A	112	5935	52.99	5976	53.35	3553	31.72
	Project B	142	10878	76.61	11343	78.88	6805	47.92

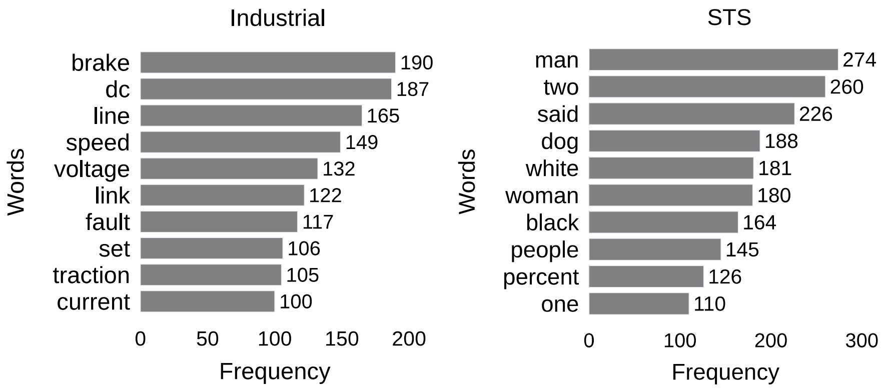

Fig. 1: Top-10 frequently occurring words in the data

3.3 Data Collection

To demonstrate the applicability of various similarity pipelines, we considered two example cases inspired by the current challenges of a large-scale railway company. Due to confidentiality agreements, we cannot share industrial data. Therefore, we apply the same pipelines on the public dataset to support replication. We consider one public dataset—the *STS benchmark*—from the NLP domain that helps demonstrate both example cases. In addition, for the requirements-driven software reuse, we also consider an industrial dataset from a larger railway vehicle manufacturing company.

The *STS benchmark* dataset contains pairs of natural language phrases that human subjects have rated from 0 (no similarity) to 5 (highly similar). As shown in Table 2, for this chapter, we consider development (Dev.) and Test set rows that include sentences with more than 30 characters. The considered *industrial* dataset contains 250+ requirements originating from two projects (Project A and Project B) that are already delivered to customers, as shown in Table 2. The requirements in both projects describe a similar safety-critical sub-system within a railway vehicle responsible for the train's propulsion control. All the requirements in both projects have traceability to implementation models that are used to generate code.

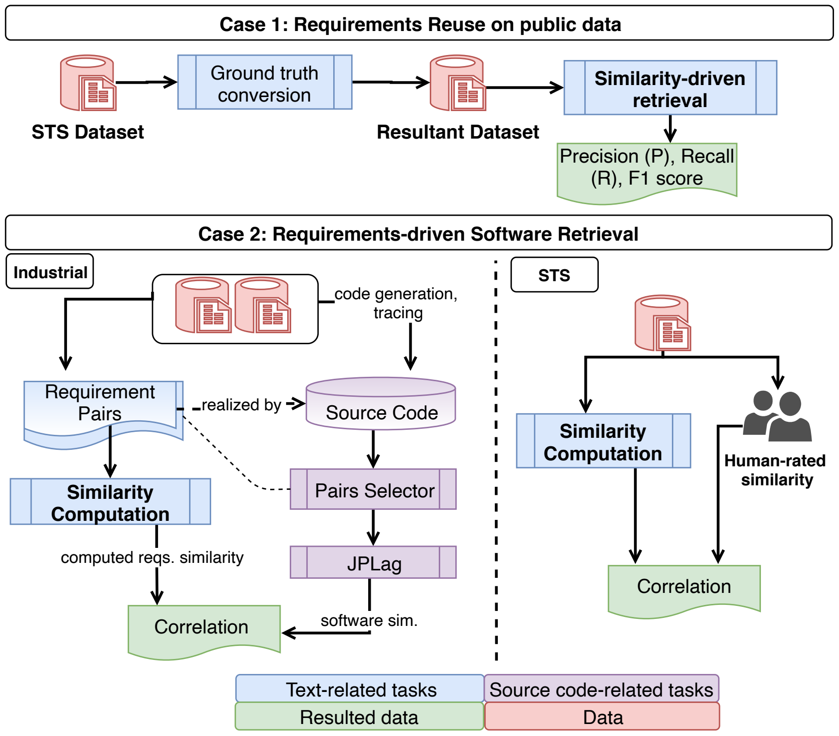

Fig. 2: Data collection procedure

In Table 2, we also present the total number of words in the datasets and the average number of words per requirement/phrase in three cases, i.e., unprocessed (`No P.`), basic pre-processing with lemmatization (`Basic P.`), and full pre-processing with stop words removal (`P.`). In addition, to provide some insights into the data, Figure 1 presents the top ten most frequently occurring words in both datasets.

Procedure, Applicability and Evaluation. We compute requirements similarity with some seminal textual similarity computation pipelines in the two cases. Below, we detail the procedure and evaluation metrics for each case.

Case 1: Requirements Reuse. The STS dataset comes in pairs of sentences with human-rated similarity values (0 to 5) as a ground truth. To demonstrate and evaluate the applicability of the similarity pipelines, we first converted the human-rated similarity values into percentages. In addition, to allow the computation of robust metrics for performance evaluation, we converted the human-rated similarity values to either similar or non-similar, as shown in the top part of Figure 2. In particular, with random trails, we considered a human-rated similarity value of more than 60% to be classified as similar. This threshold value was selected based on its effects on data imbalance between the similar and non-similar groups of pairs. In other words, the chosen 60% threshold resulted in a nearly perfect balanced dataset. As shown in Case 1 of Figure 2, the resultant data was subjected to similarity computation with the selected pipelines. In particular, the first set of sentences in the pairs are used as queries to retrieve the most similar sentences from the second set of sentences based on computed similarity. Since the ground truth of most similar sentences to query sentences is already available and the pairs are already grouped into similar and non-similar pairs, we can calculate relevant standard metrics for performance evaluation, such as precision, recall, and F1 score. In this context, True Positives (TP) are requirements correctly identified as the most similar and match the ground truth. False Negatives (FN) are instances where requirements are similar (ground truth = True), but not identified as the most similar by the pipeline. Similarly, False Positives (FP) represent requirement instances misidentified as the most similar when they are actually not the most similar ones (ground truth = False).

Read and `execute Jupyter Notebook` to explore different NLP pipelines for requirements reuse.

Case 2: Requirements-driven Software Retrieval. To mimic real-world software retrieval scenarios, we considered Project A from the industrial case as a “query” project as it was done later in time than Project B. As shown in the bottom left part of Figure 2, we use various similarity measures to retrieve the most similar requirement from Project B for each requirement in Project A to create requirement pairs and compute similarity among them. As typical in the NLP community, the computed similarity is often evaluated in light of its association with human-rated similarity [48]. However, in the context of requirements-driven retrieval of software for reuse, we can use the software similarity as a ground truth to evaluate the significance of the computed requirements similarity in a software retrieval context.

In this chapter, we use a string tiling algorithm to compute software similarity using JPLag [46]. JPLag was initially designed to detect plagiarism in students' assignments and thus can detect semantically similar code. In addition, it ignores code comments and white spaces and scans and parses the input programs to convert them into comparable string tokens. JPLag then uses a greedy version of the string tiling algorithm to compute the similarity between the tokens of the source code. The similarity number is the percentage of similar tokens in the pair of source codes.

As shown in the bottom left part of Figure 2, we trace the implementation models and then the generated code for each requirement to compute similarity among their software using JPlag. That way, the association between the computed requirements similarity and their software similarity could be used to evaluate the applicability of such approaches in a software retrieval context. The association is quantified using correlation analysis with a correlation coefficient (ρ or rho) as an indicator of the association's strength.

The correlation coefficient is a value between -1 and 1 that shows a negative or positive association between two variables. Hinkle *et al.* [35] propose to interpret the correlation coefficient value based on the ranges presented in Table 3. In requirements-driven software retrieval for reuse context, a high positive correlation between software and requirements similarity is favored and could be used as a means of evaluating such pipelines in a retrieval context.

Table 3: Interpretation of Correlation coefficient (rho OR ρ) as per Hinkle *et al.* [35]

Correlation coefficient	**Interpretation**
Between 0.9 and 1.0	Very High
Between 0.7 and 0.9	High
Between 0.5 and 0.7	Moderate
Between 0.3 and 0.5	Low
Between 0.0 and 0.3	Negligible

As shown in the bottom right part of Figure 2, we also compute the similarity among the pairs of phrases in the STS dataset using the same similarity measuring pipelines. We then use the association between the computed similarities and human-rated similarities as a means of evaluation. The STS dataset is quite similar to our industrial dataset as the phrases could be used to represent requirements, and the scaled human-rated similarity could be used to represent software similarity. Therefore, both tasks could be performed on both of the considered datasets and will enable replication. We provide our replication package with the source code and dataset [9] to allow replication and support future research on the topic.

Read and `execute Jupyter Notebook` to explore different NLP pipelines for similarity computation.

4 Results and Discussions

This section presents and discusses the results of the two considered example cases where the similarity analysis is relevant. In particular, we first present and discuss the performance of the pipeline in requirements reuse based on standard metrics like precision, recall and F1 score. Note that for the requirements reuse, we manipulated the STS benchmark for demonstration, and therefore, the results may vary in other cases. We also present the software retrieval performance of

[9] Replication package, `https://github.com/a66as/ReqSim/`

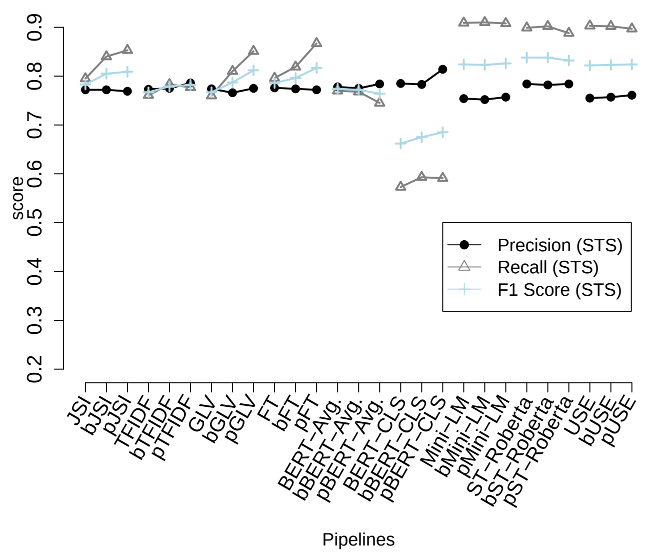

Fig. 3: Pipelines performance in requirements reuse on public dataset

the considered pipelines in terms of the association between their computed requirements similarity and the actual software similarity values.

Considering the importance of visualization in facilitating comprehension of the results, we provide Jupyter Notebook to explore and create different visualization graphs.

4.1 Case 1: Requirements Reuse

We apply the pipelines presented in Section 3.2 to the STS dataset, as described in Section 3.3. To demonstrate the applicability of the pipelines in the context of requirements reuse, we present standard metrics that evaluate the performance of the various pipelines, shown in Figure 3. Below, we discuss the results briefly.

Lexical and Distributed representation-based pipelines. As shown in Figure 3, the string-level JSI (with F1 score = 0.81) and the weighted representation-based TFIDF (with F1 score = 0.78) pipelines follow the word2vec distributed representation-based pipelines (with F1 score of 0.81 for both FT and GLV) closely in terms of F1 score. In addition, the recall (0.85) for the simple string-level JSI with pre-processing is slightly higher than the TFIDF-based pipelines

(0.78). The shorter length of the sentences and common vocabulary in the STS dataset could explain this. On the other hand, the GLV and FT-based pipelines with pre-processing tend to perform slightly better than TFIDF in terms of recall and F1 score. However, we observe that the TFIDF-based pipeline with pre-processing achieved a slightly higher precision. It is important to consider that recall may take precedence over precision (or vice versa) in some scenarios.

Contextual representation-based pipelines. For the vanilla BERT, the BERT-Avg-based pipeline (with best F1 score = 0.77) tends to perform better than BERT-CLS (with best F1 score = 0.69). However, both of the vanilla BERT variants generally tend to have comparatively lower performance in similarity-driven tasks. The subpar performance of vanilla BERT could be explained by its training objectives. BERT's primary pre-training objective is mask word and next sentence prediction and, therefore, might not perform well in similarity-driven tasks.

On the other hand, USE (with F1 score = 0.82) closely follow the performance SBERT variants (with best F1 score = 0.84). The SBERT and USE seem to perform slightly better than all other considered pipelines. In particular, the ST-Roberta performs slightly better than all other pipelines, with an F1 score of 0.84. Its training data could explain this slightly better performance of SBERT and USE. The STS benchmark dataset— which we consider to demonstrate these pipelines — is part of the pre-training datasets of these models.

Surprisingly, the SBERT variants and USE show a slight improvement in performance when pre-processing is applied. In general, minimal/no pre-processing of datasets is required for contextual representation models because of their ability to capture a wide range of linguistic features directly and the use of stop words for learning context. However, as observed from the results, pre-processed input could have marginally better performance in some cases. This showcases the ability of such models to capture the nuances of data effectively, even when pre-processed.

Takeaway: Requirements Reuse

The presented performance results of these pipelines demonstrate their relevance and applicability in requirements reuse and similarity analysis tasks. However, it is important to consider the interplay between data pre-processing and representation techniques to optimize the outcome of the NLP task at hand.

4.2 Case 2: Requirements-driven Software Retrieval

We also apply the selected pipelines for requirements-driven retrieval of software. In this case, we use both the STS benchmark and an industrial dataset from a big railway company. To evaluate the performance of the computed requirements

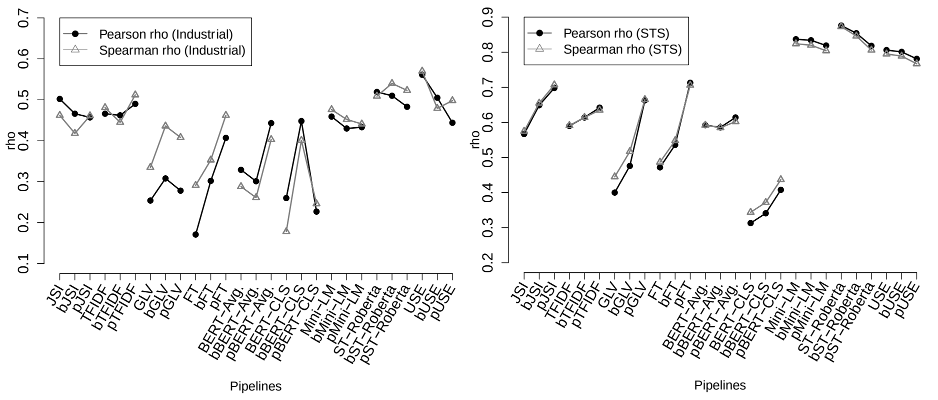

Fig. 4: Similarity pipelines Vs. Correlation coefficient

similarity in the context of software retrieval, we report the correlation coefficient (ρ) to quantify the association between the computed similarities with the similarity of their software. Below, we discuss these results briefly.

Lexical and Distributed representation-based pipelines. As shown in Figure 4, lexical approaches (JSI with $\rho = 0.50$ and pTFIDF with $\rho = 0.49$) that are based on syntax tend to perform slightly better than the pre-trained word2vec-based approaches (GLV with $\rho = 0.30$ and FT with $\rho = 0.30$) in the industrial case. However, the same trend is not apparent in the benchmark dataset. In the STS benchmark, pGLV (with $\rho = 0.66$) and pFT (with $\rho = 0.71$) perform slightly better than the lexical approaches. This could be explained by the vocabulary used in the industrial requirements. The vocabulary used in the industrial documents contains domain-specific words that are often out-of-vocabulary (OOV) for the pre-trained word2vec models. In particular, on average, 925 out of 2370 unique words of the industrial requirements are OOV in the wor2vec-based models. In contrast, the vocabulary used in the public STS benchmark is commonly used in public articles that are typically used to pre-train such models. This is also apparent in the most frequently occurring words in both datasets, shown in Figure 1. Therefore, traditional approaches could perform better in a software retrieval context where requirements are domain-specific and use complex and non-public vocabulary. There has been some work on adapting/fine-tuning such models for software engineering [26,54]; however, the efforts are limited to generic software engineering datasets and tasks.

We also observed a drop in the performance in terms of the correlation coefficient for the string-level JSI-based pipeline when pre-processing is applied for the industrial case. In contrast, the TFIDF and the word2vec-based pipelines show an increase in the correlation coefficient when pre-processing is applied in both cases. On the other hand, the word2vec-based GLV pipeline tends to produce the best results only with basic pre-processing, while FT favors full pre-processing. Nevertheless, both lexical and word2vec-based pipelines show improved performance with pre-processing in the STS benchmark case. As mentioned before, evaluating these models in different pre-processing configurations is important for obtaining the best performance in a specific case.

Contextual representation-based pipelines. For BERT, the results indicate a slight variation across the BERT with averaging strategy (BERT-Avg) and BERT with sentence token embedding (BERT-CLS). In the industrial case, the bBERT-CLS pipeline produced a slightly higher correlation with the similarity of the software it retrieved than BERT-Avg. However, an opposite trend could be observed for the STS benchmark. Generally, the results also indicate that BERT performs well in a retrieval context when pre-processing is applied. It is also important to note that, like the results of requirements reuse, both vanilla BERT variants perform slightly worse than the lexical or word2vec-based pipelines in the industrial and benchmark cases, respectively. As mentioned before, this is because the BERT's primary pre-training objective is mask word and next sentence prediction. It is not specifically tailored for similarity-related tasks and, therefore, may not perform very well in tasks where similarity computation is relevant. In general, on average, BERT-Avg performed slightly better than the BERT-CLS.

Both SBERT variants, Mini-LM and ST-Roberta, tend to perform the best in the benchmark case. The performance in terms of correlation is also only slightly lower than that of USE in the industrial case. It appears that pre-processing has a negative impact on the performance of the SBERT-based variants. This is because these language models were fine-tuned with whole sentence similarity tasks in mind by generating semantically rich sentence-level embeddings and optimizing them for the association of the computed similarity with human-rated similarity. Also, the SBERT network relies on contextual information, and when pre-processing is applied, such as stop word removal or lemmatization, the semantic information is degraded, and the word order is disrupted. As mentioned in the requirements reuse case's results, the training set used to train them could explain the good performance results (moderate and high correlation in industrial and STS cases, respectively) of the SBERT-based pipelines in the benchmark. The benchmark itself was used in the fine-tuning of the ST-Roberta variant. Nevertheless, since these models were trained for sentence similarity, the industrial case also reflects better performance. USE closely followed SBERT variants' performance in industrial and benchmark cases. In the industrial case, the performance of USE without pre-processing appears to be slightly better than S-BERT variants. This could be explained by the fact that USE is sensitive to data pre-processing, and the results also reflect that it has a negative impact on the performance of the USE-based pipelines. However, the impact of pre-processing varies in different variants of language models because of the characteristics of domain-specific datasets, and it should be assessed with a range of standard pre-processing techniques.

Takeaway: Requirements-driven Software Retrieval for Reuse

In a requirements-driven software retrieval context, the considered NLP approaches (particularly, the SBERT variants and USE) show better results in terms of the association between the computed requirements similarity and its software similarity ranging from moderate positive (in the industrial case) to high positive (in the public dataset). The results demonstrate the relevance of such approaches in requirements-driven software retrieval but also call for more research in the area.

5 Future Directions and Conclusions

In this section, we first present areas of future research and then conclude the chapter with a summary.

5.1 Future Directions

Pre-processing and similarity. In NLP for RE, we mainly borrow standard pre-processing pipelines from text mining and the NLP community. These borrowed pre-processing pipelines for textual requirements often use domain-generic part-of-speech (POS) and entity tagging that guides lemmatization and other tasks to produce input for similarity eventually. However, current similarity-driven tasks in the field rarely consider software engineering-related named entity recognition (such as classes, components, and parameters) or other meta information (such as input and conditions in the requirements) for similarity computation. A recent study also suggests that engineers perceive two requirements to be similar if they share similar input processing and have similar conditions [1]. Such additional information is not extracted for similarity and, if done, could guide the similarity computation in the right direction.

Pre-trained models for representation. We observed that quite a big portion of railway industry-specific words from our data are not seen by most of the considered pre-trained models. While newer approaches for embedding and representation of textual data have ways to deal with such out-of-vocabulary words, we believe that the performance may improve significantly if they are pre-trained on domain-specific data. Literature suggests some efforts towards software engineering-specific pre-trained models [26,54]. However, the efforts are limited to shallow language models or non-RE-related tasks. Studying the domain adaptability, its challenges, and pre-training large language models specifically for RE tasks from scratch could elevate RE processes in practice with more accurate requirements retrieval and similarity analysis. In the future, we plan to pre-train large language models for the railway domain to achieve various downstream to assist requirements engineers in requirements elicitation, specification, and formalization.

Cross-domain retrieval. As noted by the recent secondary study [58], requirements search and retrieval are less explored in NLP for RE. Having requirements-driven software search and retrieval for reuse could reduce waste in the development processes and could reduce the time-to-market for upcoming projects. The typical assumption of such retrieval approaches is often that requirements similarity could be used as a proxy for similarity in other domains, such as models or software. This assumption allows requirements-driven reuse of artifacts from other domains, but the current language models are not tailored and optimized for those domains. For example, language models could be re-trained with the objective of optimizing the correlation between the similarity it computes with the similarity of software. In addition, requirements could be supplemented with information from other domains, such as tokens from their code, to represent better other domains in the training process of language models. We believe considering cross-domain information for training language models may produce better results in cross-domain recommendation tasks.

More applications to explore. Requirements similarity and text classification could be leveraged to perform tasks beyond finding similar requirements or classifications. For example, conditionals in requirements could be extracted, and test cases could be generated to validate requirements [31]. Additionally, traceability links could be used as an enabler for consistency checking between requirements and their implementation, models, and test cases.

Furthermore, requirements representation and similarity could also be leveraged in deriving sub-requirements, requirements completion, and in-Editor support for requirements writing. In particular, the next words/phrases could be suggested to engineers while writing requirements based on search and retrieval, as done in other domains (e.g., [47]). Furthermore, having domain-specific pre-trained models could also enable requirement assistants that use emerging generative approaches for reviewing requirements and requirements summarization.

5.2 Conclusions

Requirements similarity is a crucial enabler for RE recommenders, traceability link recovery, reuse recommendation, and many other RE activities. In this chapter, we presented linguistic similarity, data representation approaches, and similarity metrics used to compute textual similarity. Furthermore, we demonstrated the applicability of various similarity computation pipelines in requirements reuse and requirements-driven software retrieval for reuse. We also outline future directions in similarity computation and retrieval for reuse.

Acknowledgements

This work has been supported by and received funding from the ITEA SmartDelta [50] and the KDT AIDOaRT projects.

References

1. Abbas, M., Ferrari, A., Shatnawi, A., Enoiu, E., Saadatmand, M., Sundmark, D.: On the relationship between similar requirements and similar software: A case study in the railway domain. Requirements Engineering **28**(1), 23–47 (2023)
2. Abbas, M., Jongeling, R., Lindskog, C., Enoiu, E.P., Saadatmand, M., Sundmark, D.: Product line adoption in industry: An experience report from the railway domain. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A. SPLC '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3382025.3414953
3. Abbas, M., Saadatmand, M., Enoiu, E., Sundamark, D., Lindskog, C.: Automated reuse recommendation of product line assets based on natural language requirements. In: International Conference on Software and Software Reuse. pp. 173–189. Springer (2020)
4. Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Traynor, M.: Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empirical Software Engineering **25**, 5454–5497 (2020)
5. Aizawa, A.: An information-theoretic perspective of tf–idf measures. Information Processing & Management **39**(1), 45–65 (2003)
6. Arora, C., Sabetzadeh, M., Goknil, A., Briand, L.C., Zimmer, F.: Change impact analysis for natural language requirements: An nlp approach. In: 2015 IEEE 23rd International Requirements Engineering Conference (RE). pp. 6–15. IEEE (2015)
7. Arsan, T., Köksal, E., Bozkus, Z.: Comparison of collaborative filtering algorithms with various similarity measures for movie recommendation. International Journal of Computer Science, Engineering and Applications (IJCSEA) **6**(3), 1–20 (2016)
8. Balazs, J.A., Velásquez, J.D.: Opinion mining and information fusion: a survey. Information Fusion **27**, 95–110 (2016)
9. Bashir, S., Abbas, M., Ferrari, A., Saadatmand, M., Lindberg, P.: Requirements classification for smart allocation: A case study in the railway industry. In: 31st IEEE International Requirements Engineering Conference (September 2023)
10. Bashir, S., Abbas, M., Saadatmand, M., Enoiu, E.P., Bohlin, M., Lindberg, P.: Requirement or not, that is the question: A case from the railway industry. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 105–121. Springer (2023)
11. Bengio, Y., Ducharme, R., Vincent, P.: A neural probabilistic language model. Advances in neural information processing systems **13** (2000)
12. Berry, D.M.: Empirical evaluation of tools for hairy requirements engineering tasks. Empirical Software Engineering **26**(6), 1–77 (2021)
13. Birner, B.J.: Introduction to pragmatics. John Wiley & Sons (2012)
14. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the association for computational linguistics **5**, 135–146 (2017)
15. Borg, M., Wnuk, K., Regnell, B., Runeson, P.: Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. IEEE Transactions on Software Engineering **43**(7), 675–700 (2016)
16. Bowman, S.R., Angeli, G., Potts, C., Manning, C.D.: A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326 (2015)
17. Bybee, J.: Phonology and language use, vol. 94. Cambridge University Press (2003)
18. Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., Specia, L.: Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. arXiv preprint arXiv:1708.00055 (2017)

19. Cer, D., Yang, Y., Kong, S.y., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al.: Universal sentence encoder. arXiv preprint arXiv:1803.11175 (2018)
20. Chandrasekaran, D., Mago, V.: Evolution of semantic similarity—a survey. ACM Computing Surveys (CSUR) **54**(2), 1–37 (2021)
21. Clark, K., Khandelwal, U., Levy, O., Manning, C.D.: What does bert look at? an analysis of bert's attention. arXiv preprint arXiv:1906.04341 (2019)
22. och Dag, J.N., Regnell, B., Gervasi, V., Brinkkemper, S.: A linguistic-engineering approach to large-scale requirements management. IEEE software **22**(1), 32–39 (2005)
23. Davidson, D., Harman, G.: Semantics of natural language. Philosophy of language: The central topics pp. 57–63 (2008)
24. Deza, E., Deza, M.M., Deza, M.M., Deza, E.: Encyclopedia of distances. Springer (2009)
25. Dice, L.R.: Measures of the amount of ecologic association between species. Ecology **26**(3), 297–302 (1945)
26. Efstathiou, V., Chatzilenas, C., Spinellis, D.: Word embeddings for the software engineering domain. In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR). pp. 38–41 (2018)
27. Elmore, K.L., Richman, M.B.: Euclidean distance as a similarity metric for principal component analysis. Monthly weather review **129**(3), 540–549 (2001)
28. Falessi, D., Cantone, G., Canfora, G.: A comprehensive characterization of nlp techniques for identifying equivalent requirements. In: Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement. pp. 1–10 (2010)
29. Falessi, D., Cantone, G., Canfora, G.: Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. IEEE Transactions on Software Engineering **39**(1), 18–44 (2011)
30. Faruqui, M., Dodge, J., Jauhar, S.K., Dyer, C., Hovy, E., Smith, N.A.: Retrofitting word vectors to semantic lexicons. arXiv preprint arXiv:1411.4166 (2014)
31. Fischbach, J., Frattini, J., Vogelsang, A., Mendez, D., Unterkalmsteiner, M., Wehrle, A., Henao, P.R., Yousefi, P., Juricic, T., Radduenz, J., et al.: Automatic creation of acceptance tests by extracting conditionals from requirements: Nlp approach and case study. Journal of Systems and Software **197**, 111549 (2023)
32. Guo, J., Cheng, J., Cleland-Huang, J.: Semantically enhanced software traceability using deep learning techniques. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 3–14. IEEE (2017)
33. Halliday, M.A.K., Webster, J.J.: On Language and Linguistics: Volume 3. A&C Black (2003)
34. Haspelmath, M., Sims, A.: Understanding morphology. Routledge (2013)
35. Hinkle, D.E., Wiersma, W., Jurs, S.G.: Applied statistics for the behavioral sciences. Houghton Mifflin, 5th ed edn. (2003), `https://cir.nii.ac.jp/crid/1130012535369496890`
36. Ilyas, M., Kung, J.: A similarity measurement framework for requirements engineering. In: 2009 Fourth International Multi-Conference on Computing in the Global Information Technology. pp. 31–34. IEEE (2009)
37. Kotonya, G., Sommerville, I.: Requirements engineering: processes and techniques. Wiley Publishing (1998)
38. Latif, S., Bashir, S., Agha, M.M.A., Latif, R.: Backward-forward sequence generative network for multiple lexical constraints. In: Artificial Intelligence Applications

and Innovations: 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, June 5–7, 2020, Proceedings, Part II 16. pp. 39–50. Springer (2020)

39. Manning, C.D.: An introduction to information retrieval. Cambridge university press (2009)
40. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
41. Mikolov, T., Yih, W.t., Zweig, G.: Linguistic regularities in continuous space word representations. In: Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies. pp. 746–751 (2013)
42. Mohammad, S.M., Hirst, G.: Distributional measures of semantic distance: A survey. arXiv preprint arXiv:1203.1858 (2012)
43. Naseem, U., Razzak, I., Khan, S.K., Prasad, M.: A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. Transactions on Asian and Low-Resource Language Information Processing **20**(5), 1–35 (2021)
44. Navarro, G.: A guided tour to approximate string matching. ACM computing surveys (CSUR) **33**(1), 31–88 (2001)
45. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
46. Prechelt, L., Malpohl, G., Philippsen, M., et al.: Finding plagiarisms among a set of programs with jplag. J. UCS **8**(11), 1016 (2002)
47. Raychev, V., Vechev, M., Yahav, E.: Code completion with statistical language models. In: Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation. pp. 419–428 (2014)
48. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 (2019)
49. Rodriguez, D.V., Carver, D.L.: Comparison of information retrieval techniques for traceability link recovery. In: 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT). pp. 186–193. IEEE (2019)
50. Saadatmand, M., Abbas, M., Enoiu, E.P., Schlingloff, B.H., Afzal, W., Dornauer, B., Felderer, M.: Smartdelta project: Automated quality assurance and optimization across product versions and variants. Microprocessors and Microsystems **103**, 104967 (2023). https://doi.org/10.1016/j.micpro.2023.104967
51. Schnabel, T., Labutov, I., Mimno, D., Joachims, T.: Evaluation methods for unsupervised word embeddings. In: Proceedings of the 2015 conference on empirical methods in natural language processing. pp. 298–307 (2015)
52. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval, vol. 39. Cambridge University Press Cambridge (2008)
53. Sunilkumar, P., Shaji, A.P.: A survey on semantic similarity. In: 2019 International Conference on Advances in Computing, Communication and Control (ICAC3). pp. 1–8. IEEE (2019)
54. Tabassum, J., Maddela, M., Xu, W., Ritter, A.: Code and named entity recognition in StackOverflow. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 4913–4926. ACL (2020)
55. Van Valin, R.D.: An introduction to syntax. Cambridge university press (2001)
56. Williams, A., Nangia, N., Bowman, S.R.: A broad-coverage challenge corpus for sentence understanding through inference. arXiv preprint arXiv:1704.05426 (2017)

57. Zhang, Y., Jin, R., Zhou, Z.H.: Understanding bag-of-words model: a statistical framework. International journal of machine learning and cybernetics **1**, 43–52 (2010)
58. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., Batista-Navarro, R.T.: Natural language processing for requirements engineering: A systematic mapping study. ACM Computing Surveys (CSUR) **54**(3), 1–41 (2021)
59. Zhao, Y., Scholer, F., Tsegay, Y.: Effective pre-retrieval query performance prediction using similarity and variability evidence. In: Advances in Information Retrieval: 30th European Conference on IR Research, ECIR 2008, Glasgow, UK, March 30-April 3, 2008. Proceedings 30. pp. 52–64. Springer (2008)