Natural Language Processing and Artificial Intelligence Methods in Software Engineering

PhD Thesis

László Tóth

Supervisors

Tibor Gyimóthy, DsC
László Vidács, PhD

Doctoral School of Computer Science
Department of Software Engineering
Faculty of Science and Informatics
University of Szeged

Szeged
February 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The advance of technology is based on making it fit in so that you don't really even notice it, so it's part of everyday life."

Bill Gates

Over the past decade, we have witnessed an unprecedented acceleration in technological development. The evolution of microelectronics has enabled the general spread and expansion of programmable devices. Such appliances can now be found in all walks of life, the industry, the commerce, the financial sector, the healthcare, and even the household relying on the services provided by programmable devices and software-driven systems. As the computing power of programmable devices and computers has increased, so has the complexity of their tasks. The increase in computing power and the development of artificial intelligence, especially machine learning, have made it possible to solve tasks that only humans could previously do. Nevertheless, these services, such as face or speech recognition, are also included in modern smartphones' services.

As we entrust more and more of our life decisions to increasingly complex applications, often powered by artificial intelligence, the development of these applications is becoming increasingly complex, which cannot leave software development processes and technologies untouched. Since the days when the software was not just designed to automate a single complex calculation, the process of software development has become a series of sophisticated and well-thought-out activities for which various methodologies and technologies have also been worked out.

The fact that the development of software designed to solve increasingly complex problems is not simply a matter of cumulative use of algorithms designed for sub-problems is reflected in the concept of software crises coined by F.L. Bauer at the 1968 NATO conference in Garmisch, Germany [1], and often used since then to

express various problems originated from the complexity of software development. Initially, the way to increase the reliability and efficiency of software was almost exclusively dominated by the engineering approach, resulting in software development methodologies such as the waterfall model [2] or the V-model [3], which also considers the testing aspects. In parallel with these methodologies, there have been substantial developments in the expressiveness of programming languages and the emergence of programming paradigms optimized for specific tasks. These developments have helped solve the software crisis and opened up new possibilities for solving even more complex problems by software, aided by electronics development. At the same time, the increase in complexity and the volatility of the requirements of the problems solved by the software became a source of additional problems that engineering solutions alone could no longer adequately address [4].

To provide software solutions to the problems of a rapidly changing economic and social environment, the developers need to consider also the human factor besides the engineering aspects. Recognition of this problem led to the emergence, and then the rise, of agile development methodologies [5]. Rather than rigorous and detailed planning, agile accepts the need for change and designs and implements the software product in small steps during development. Small incremental developments allow changes to be implemented on time at a low cost while ensuring that the software is delivered to the customer in increments [6]. The latter is of particular importance because, prior to the advent of agility, a large proportion of project failures were caused by delays and cost increases that customers no longer accepted.

Agile methodologies have greatly improved software quality, but methodologies alone do not provide sufficient support to solve many problems, and there is a need for technology improvements to support developers in meeting requirements. As mentioned at the beginning of this chapter, software-driven systems are present in all areas of life, and therefore the expectations for their secure and efficient operation are also increased. However, these requirements are rarely available taxonomically and formally but are instead available to developers in the form of human interactions, expressed in natural language, often incompletely and imprecisely. The human factor and communication play a key role in software development processes, from requirements analysis through implementation to customer support.

Apart from the fact that communication may contain inaccuracies and omissions, it is also fraught with additional difficulties, which can be seen in the differences in meaning within the language used by the communication partners, determined by cultural factors [7, 8]. Natural language methods, equipped with machine learning tools, can help overcome communication barriers and support modeling requirements. The second chapter of this thesis provides the possibilities of classifying natural language requirements using machine learning. Furthermore, that chapter presents a procedure of creating semantic networks using a corpus of professional

discourse between stakeholders from different skilled backgrounds. The topic of the second chapter is the first thesis group of the dissertation, which is entitled Natural Language Processing and Artificial Intelligence Methods in Requirements Analysis.

As software has been used to focus on solving increasingly complex problems, ensuring the reusability of code developed for repetitive tasks has become an essential consideration for software developers. Reusability has also led to a rapid increase in development libraries for each programming language. At the same time, various frameworks have been developed which can significantly increase the efficiency of development if they are used with the right expertise. With the growth in developer tools, programmers can no longer keep up, so developers need to specialize in a particular discipline, programming language, or framework [9]. At the same time, specialization also means that developers in different disciplines need to cooperate more closely, highlighting the importance of communication and communication difficulties between developers working in different professional environments.

Over the last few years, online question answering (Q&A) forums such as Stack Overflow have become the essential knowledge repository for software engineers. Since the information gathered on the portal is tied to professional questions originating from practitioners or hobbyist programmers, these posts provide efficacious and practically applicable support to novice and experienced professionals [10, 11]. These Q&A platforms serve as a primary communication platform among the developers, so the study of communication using these channels is a popular research subject. With the growing popularity of Q&A platforms, there is increasing pressure on operators and moderators to maintain the professionalism of the platform, while at the same time strict rules make it much harder for less experienced users and their questions left unanswered or, in the case of a portal such as Stack Overflow, closed.

In the third chapter, the investigation of developer interactions on the Stack Overflow platform using natural language tools and deep learning and its results are presented. The chapter is also the second thesis group of the dissertation, entitled Investigating Developers Interactions Using Natural Language Processing And Deep Learning.

1.1 Contributions

The ideas, figures, tables, and results included in this thesis were published in scientific papers (listed at the end of the thesis). The thesis groups of the dissertation are presented in the second and third chapters. The author's contributions to the relevant chapters are given below.

Chapter 2.: The chapter presents the natural language processing and artificial intelligence methods in requirements analysis as the first thesis group of the dissertation. The principal findings of the research can be stated as the following theses:

- Based on the references to non-functional requirements in the text, requirements can be efficiently classified using linear models based on `tf-idf` vectorization. Based on the experiments, the best performance was achieved by the following three models: *SVM, Multinomial Naive Bayes, and Logistic Regression*.
- In the case of a significant number of learning examples, the *neural network* models classify non-functional requirements more efficiently than the traditional linear models.
- Based on the examination of the related literature and the investigation of the semantic relationships of the terms used in a specific and common semantic domain, it is confirmed that the meaning of the terms used in communication may differ significantly in the different semantic domains.
- Semantic networks are appropriate tools to resolve the confusion originating from the different meanings of the same terms.
- The semantic network that presents the hyperonym-hyponymy relationships based on the Stack Overflow posts has a specific structure. It is a *directed graph*; the degree distribution follows a *Box-Cox Exponential Distribution* in the hyponymy $\rightarrow$ hyperonymy direction and the *Power Law Distribution* in the opposite direction, respectively. Besides, the graph has a *very low clustering coefficient*, meaning that only a few concepts tend to form triadic closures.

The author is responsible for the following contributions in this thesis group:

- The author has developed preprocessing methods and a vectorization process applying the `tf-idf` representation form.
- The author has implemented scripts responsible for executing the classification experiments using various machine learning models implemented in the `scikit-learn` library.

- The author has implemented a simple neural network applied in the classification experiments based on the Stack Overflow samples.
- The author executed the experiments, compared the results of the classifiers, and identified the best classifiers.
- Based on the investigation of the semantics of the linguistic expressions, the author established a solid definition of semantic space and semantic networks, respectively.
- The author has implemented preprocessing steps to extract posts from the Stack Overflow database and separate them into proper sentences cleaned from the auxiliary characters and noise.
- The author has implemented a set of regular expressions based on the lexico-syntactic patterns representing the hyperonym-hyponym relationships found in the literature.
- The author has developed a phrase structure grammar and an automatization to recognize noun phrases in the text. To the best of our knowledge, the grammar provided by the author is the most general formalized solution available in the literature.
- The author has developed a simplified automatization for recognizing the noun phrases considering only a small set of the lexico-syntactic patterns.
- The author has built a semantic network based on the hyperonym-hyponym relationships, representing the semantic field of the software development community based on the Stack Overflow post utilizing the lexico-syntactic patterns.
- The author has investigated the structure of the resulting network and described its structure.
- The author compared the smaller network resulting from the mining process with the semantic network representing the common knowledge provided by WordNet.

Chapter 3.: Chapter three presents the investigation of developer interactions using natural language processing and deep learning as the second thesis group of the dissertation.

- Neural networks based on the recurrent neural models are adequate tools for classifying the quality of the questions published on Stack Overflow, leaning solely on the textual information encoded in the questions.

- Neural networks based on the recurrent neural models are adequate tools for predicting the likelihood of the closures of the questions published on Stack Overflow, leaning solely on the textual information encoded in the questions.
- Neural networks based on the recurrent neural models are proper tools for predicting the possible reasons for the closures of the questions published on Stack Overflow, leaning solely on the textual information encoded in the questions.
- In predicting the reasons for the closure, neural networks deliver more significant errors in cases where human decision also shows a more uncertain pattern.

The author is responsible for the following contributions in this thesis group:

- The author has developed a GRU-based deep learning model to classify the questions based on their quality posted to Stack Overflow, considering only the textual elements. The definition of the quality was taken from the literature for comparison purposes.
- The author executed the classification by applying different amounts of samples and vectorization processes.
- The author compared the results with each other and the results of the other researchers. When all inputs were used in the classification process, the model's performance outperformed the performance of the classifiers used by others, demonstrating that deep learning solutions can provide better results if there is enough input available.
- The author has developed three distinct GRU-based deep learning models to classify the likelihood of closing questions posted to the Stack Overflow, considering only the textual information available during the assembling of that question.
- The author executed the classifications and compared the results with the other results available in the literature. The classifier provided good performances outperforming the other results, which can also be used in practice to evaluate the possibility of the question's closure before posting it to Stack Overflow.
- The author modified the models to perform multi-class classification. In this case, the classifiers predict the possible closing reasons. To the best of our knowledge, this was the first classifier published to predict the closing reasons. However, a parallel study was in progress and published lately for the same purpose, but the classifiers made by the author have a slightly better performance. The classifiers provide a good result and are applicable in practice.

Table 1.1: *Correspondence between the thesis points and my publications.*

Publication	Thesis point																	
	I/1	I/2	I/3	I/4	I/5	I/6	I/7	I/8	I/9	I/10	I/11	I/12	II/1	II/2	II/3	II/4	II/5	II/6
[I]			•	•														
[II]		•		•														
[III]													•	•	•			
[IV]																•	•	•
[V]					•	•	•	•		•								
[VI]	•																	
[VII]						•			•	•	•	•						

Table 1.1 summarizes the relation between the thesis points and the corresponding publications.

Chapter 2

Natural Language Processing and Artificial Intelligence Methods in Requirements Analysis

"The hardest part of the software task is arriving at a complete and consistent specification, and much of the essence of building a program is in fact the debugging of the specification. "

Frederick P. Brooks

Like any other product, the software is created to meet socio-economic actors' needs. The software must fully meet the stakeholders' requirements to fulfill its purpose. These requirements are often not derived from a homogeneous medium but are provided by the various stakeholders through legislations or other regulations, business or technical documents, or orally using natural languages. From an often conflicting, inaccurate, or incomplete set of requirements, business analysts are tasked with producing a coherent and logically consistent, systematic set of requirements specifications that system analysts and developers can use to produce the appropriate designs and implementation. The quality of the requirements specification directly impacts the quality of the software and, not least, on the cost of development. A poorly formulated specification can lead to a flawed implementation, the subsequent correction of which, if at all possible, is a significant cost driver. However, even for experienced business analysts, producing a specification of sufficient quality from the often imprecise and contradictory information available in natural language is a real challenge. This task can be significantly aided by tools that can classify natural language requirements, check their completeness and consistency, or support the

resolution of semantic differences between business and technical language.

In the first thesis group of the dissertation, the results on the classification of natural language requirements are presented, focusing on the often neglected non-functional requirements. Besides, the investigation of semantic networks that support the resolution of semantic barriers between business and software development language is also discussed.

2.1 Introduction

Software systems are made based on specifications composed of functional and non-functional requirements. The needs and expectations of the customers are expressed by requirements primarily given in a natural language form. These textual documents are often ambiguous and sometimes also contain contradictions. To use the collected requirements for software design, they must be formalized, and any inconsistencies need to be removed. These tasks can be demanding and time-consuming. Tools supporting business analysts in requirements engineering provide templates, checklists, traceability, management, and reporting environment to facilitate their work, but these tools can support the processing of requirements given in textual form only to a limited extent [12].

The quality of requirements is the primary factor in a software project's success. Non-functional requirements (NFRs) are also crucial for software design [13]. The lack of a well-structured set of non-functional requirements can lead to an inappropriate software design and the project's failure. Many NFRs are out of the analysis, and those non-functional requirements considered during analysis are often weakly elaborated. Firesmith, in his article issued in the Journal of Object Technology in 2007, has collected the most common issues related to requirements engineering along with some practice to solve these problems [14]. Although steps have been taken to improve the quality of the specifications, unsuccessful software projects are still being attributed mainly to inadequate requirements engineering [15].

Requirements are originated from memos of interviews and other textual sources like regulations, laws, or reports. Regulations and laws are well-structured documents, but their structure reflects only the business viewpoints. The main challenge for processing business documents is managing texts and identifying information relevant to the application being made. On the other hand, memos created during interviews are often unstructured or semi-structured and frequently contain ambiguities or logical fallacies. Requirements are embedded into these textual contexts, and their extraction and classification are essential duties of business analysts. Non-functional requirements are mostly part of the text, but sometimes they are expressed superficially. Identifying and classifying requirements from this collection of different documents can be demanding and error-prone. Several investigations

have been made with remarkable results to support the identification and classification process using natural language processing and machine learning methods [16, 17, 18, 19, 20, 21, 22, 23]. Some researchers investigated the use of ontologies, which have been created based on the standards [17, 22]. Some researchers like Lu and Liang [21] or Abad et al.[16] utilized supervised learning methods, and others utilized semi-supervised learning techniques such as Expectation Maximization strategy [19]. These studies have shown that natural language processing and machine learning methods can be utilized successfully also for requirements engineering.

Learning examples have to be available to apply machine learning and natural language processing (NLP), but only a few labeled examples concerning the functional and non-functional requirements can be accessed via the Internet. The majority of researchers focusing on elaborating the non-functional requirements have used the Tera Promise NFR dataset, which the students of DePaul University created [24, 25]. This dataset contains 625 examples of 15 projects classified into 12 classes. Some classes contain very few examples that are detrimental to machine learning methods. One possible direction for overcoming the shortage of labeled examples is using semi-supervised methods, which can give better results at these conditions, as shown by Casamayor et al. [19]. Using ontologies as a background knowledge representation is another possible solution [17, 22], but building a proper ontology-based database is tedious and time-consuming. Rashwan et al. [22] have created an ontology based on the ISO/IEC 9126-1:2001[1]. Besides these strategies, a corpus of requirements also can be built. Software repositories make available various open-source software that can also provide sources of requirements related to the specific software [26, 27]. Extracting requirements from these repositories and building a corpus can support the usage of machine learning processes effectively. Q&A portals, like Stack Overflow, contain posts related to different aspects of software engineering, including requirements engineering. While these are not original requirements from business people, several posts related to non-functional requirements can be found and filtered using the appropriate tags. Stack Overflow posts can complement the scarce corpus of non-functional requirements since the terms used for related concepts form a standard basis. The potential of using a portal to identify non-functional requirements has been investigated and compared with the results obtained using the Tera Promise NFR dataset purely. Our results confirmed our hypothesis, i.e., that discourse resources in professional portals support the learning process for identifying non-functional requirements.

Applying NLP and machine learning methods for requirements engineering is based on the assumption that these methods can facilitate business analysts' work by reducing the amount of manual work, therefore reducing the time needed for

[1]This standard has been revised, and a new standard was published in 2011 as the ISO/IEC 25010:2011.

elicitation and the cost of the analysis. This assumption has also been confirmed by Groen et al. in their investigation [28].

As mentioned previously, the requirements come from business people of a particular professional domain. The duty of the business analysts is to organize these requirements and create a formal or semi-formal model suitable for software design for the development team. In order to perform this task at the appropriate level, business analysts need to have a thorough understanding of the requirements and their implications. The principal difficulty of this task is the usage of the same terminologies with different meanings and contexts and the related tacit knowledge of the business side, which is usually not articulated during the elicitation process. This dilemma exists between the developers and the customers in almost every communication situation.

The problem mentioned above lies in the specific nature of human communication. Information exchange is influenced by several factors, such as cultural background, social environment, the available communication channels, personality and mental state of the participants, and their communication intention, even when a common language is used. The cultural background and the social environment have paramount importance because they can affect the actual meanings of the words used in the communication. However, the participants can only understand each other if they use communication elements in the same sense.

Although communication disruptions can cause problems in everyday life, they may also have unforeseeable consequences in business life, especially if the communication within the participants does not go smoothly. Primarily in larger and sometimes in medium-sized companies, there is a remarkable diversification between the individual departments in the organizational culture, often reflected in their language usage. This phenomenon, called the communication silo [29], is increasingly present in communication between IT and the business area [30, 31]. The incomplete or insufficiently detailed requirements also mentioned by Firesmith [14] may be due to similar communication problems, despite most business analysts being experienced professionals with business knowledge.

Mapping the different semantic fields provides a possible solution for reconciling the different meanings and catching the corresponding tacit knowledge. The notions used in a particular domain often provide another or overplus meaning of the words denoting them. The semantic field is a set of lexemes describing a conceptual domain and its relationships [32, 33]. The semantic field can be represented by directed graphs where the nodes are the terms related to the specific notion, and the edges represent the relationship among the notions. These constructions are called semantic networks [34]. With the aid of these networks, terms from different semantic fields can be matched together either directly or via a formal upper ontology [35, 36]. This mapping process supports recognizing the different properties

of the meaning and catching the tacit knowledge among the participant of different domains.

The current thesis group presents our research results supporting requirements analysis and processing requirements. The discussion starts with the results on the classification of non-functional requirements using machine learning algorithms and natural language processing methods. The performance of the classifiers was investigated using both purely the Tera Promise NFR dataset and with the dataset extension from the Stack Overflow requirements related posts. Since Stack Overflow posts make the dataset size suitable for studying deep architectures, the set of models used in previous experiments was extended by studying the classification applying the most straightforward, fully connected neural network.

In the second part of this chapter, our research results related to the application of semantic networks presented that can be constructed in different semantic spaces to address the problems of the silo phenomenon that causes communication difficulties by modeling the software development semantic environment using Stack Overflow posts. In our research, an extensible semantic network was constructed for detecting and extracting hypernym relations in discourse in Stack Overflow posts.

Structure of the chapter: Section 2.2 summarizes the related works and briefly presents previous results. The research about the classification of non-functional requirements is covered in Section 2.3. The role of semantic networks and their construction from Stack Overflow posts as a representation of the software developer's semantic domain is covered in Section 2.4. Results and final thoughts are summarized in Section 2.5.

2.2 Related Works

The problem of processing requirements documents using natural language processing and machine learning methods has been a research topic for decades [18]. Although non-functional requirements are less dependent on the application domain, setting up a general list of NFR types is not a trivial problem. The types identified in the literature are widespread. For example, Chung et al. [37] identified 156 NFR categories, while Mairiza et al. [38] separated 114 different NFR classes in their work, on the contrary to the six high-level categories defined by the ISO/IEC 25010:2011 standard.

A fundamental study of NFR classification was published relatively lately in 2006 by Cleland-Huang et al. [25]. They used 14 NFR categories separated from functional requirements. More than 600 requirements from 15 projects were collected and manually categorized to train and test their categorization methods. Cleland-Huang et al. achieved high recall with the tradeoff of really low precision. Several researchers in the past reproduced this experiment [39, 40]. Casamayor et al.

[39] employed a multinomial Naive Bayes classifier coupled with an Expectation-Maximization algorithm. Rashwan et al. [22] applied the Support Vector Machine algorithm for extracting non-functional requirements from the specifications. The extraction process is based on ontologies composed by the authors. Balushi et al. [17] developed an ontology-based elicitation tool called ElicitO that helps capture precise non-functional requirements (NFRs) specifications during elicitation interviews. Li and Chen [41] and Alrumaih et al. [42] applied an ontology-based approach for classifying requirements. Kaiya and Saeki [43] used ontologies as domain knowledge to address the completeness issue in requirements engineering. Their early work of modeling the semantic field of a business domain provided proof of concepts in the applicability of the approach in the elicitation process. Sharma et al. [44] addressed the NFR extraction problem with a rule-based approach. They implemented a framework for NFR analysis, including DSL (Domain Specific Language). Sawyer et al [45] have focused on document archaeology and created an NLP-based tool called REVERE to support business analysts in investigating different documents containing requirements. This tool has utilized standard NLP techniques like part-of-speech tagging or semantic tagging and determination of modality.

Denger et al. [46] examined the ambiguity of requirements sentences and investigated language patterns for rewriting these requirements into less ambiguous sentences. The ambiguity of requirements is one of the most prominent issues in requirements engineering, which has to be resolved, as pointed out by Firesmith in his paper [14]. Kang and Park [47] also used the linguistics viewpoint to examine ambiguity, completeness, and conformity of requirements. The researchers developed a requirement-grammar and applied error patterns to develop a tool for parsing the manually annotated corpus built by system-engineering-related texts. The annotated corpus can then be used for the training error model, which can be applied to check the exactness of the requirements.

Requirements traceability is a related field where NLP and information retrieval techniques are frequently applied [48, 49, 50]. Hindle et al. [51] used topic modeling to link NFRs to topics found in commit messages. Falessi et al. [52] conducted a largescale experiment with various NLP techniques, including different algebraic models, term weightings, and similarity metrics to detect identical non-functional requirements. From the perspective of mining software repositories, Paixao et al. [53] investigated the relationship between built results obtained from continuous integration tools with non-functional requirements.

To overcome the scarcity of the labeled examples available via the Internet, extracting requirements-related information from social networks has become a frequently examined area. Portugal et al. [26] applied the pattern-search method and extracted requirements-related information from GitHub. They have also built a corpus of requirements based on the information retrieved from README files located

on GitHub repository [27]. Stack Overflow is also a popular repository among researchers. Zou et al. [54] examined the repository utilizing the Latent Dirichlet Allocation (LDA) to discover the main discussion topics. These topics are related to various NFRs, which the researchers also identified. The topics that are the developers' focus were also investigated by Barua et al. [55] using LDA on StackOverflow textual database.

Following the work presented in this thesis, research on the classification of non-functional requirements has continued, looking for more accurate solutions. Dias and Cordeiro [56] applied different vectorization methods along with the classic machine learning models for the expanded Promise NFR dataset (Promise_exp) and achieved excellent (91% F1) results. The best result was achieved using the SVM classifier, like in our experiments. Their results are between our results using the pure Promise NFR and the dataset mined from Stack Overflow.

Applying deep learning methods for classification non-functional requirements is also a recently investigated area. Kobilica et al. [57] investigated 22 supervised learning methods, among them two deep learning methods, the LSTM (Long Short Term Memory) and the Boosted Ensemble Method. They achieved 80-84% accuracy using the security-related requirements dataset called SecReq. Hey et al. [58] constructed a new classifier called NoRBERT based on the BERT (Bidirectional Encoder Representations from Transformers) model. Their model achieved 87-94% F1 value by classifying the Promise NFR dataset.

Applications of semantic networks in software engineering have recently received more attention. These structures have been investigated since the 1980s, but software engineering began to exploit the potential of these structures only in the 2000s. Steyvers and Tenebaum [59] investigated the structure of two famous semantic networks, WordNet and Roget's Thesaurus 4. They found that these databases have a small world structure and the distributions of the number of connections follow the power-law distribution. Seitner et al. [60] have extracted hypernym relations from the CommonCrawl web corpus. Du and colleagues [61] have developed a tool called OntoSpider to extract ontologies from a website and convert the pure HTML Web to Semantic Web. Martino et al. [62] conducted a comparison in terms of performance. The quality of NLP result, OWL (Semantic Web Language) completeness and richness between definite-clause formalism and the Watson Relationship Extraction service of IBM Cloud platform Bluemix were studied. Vizcaíno et al. [63] focused on establishing standard vocabularies among the development participants, whereas Wongthongtham et al. [64] considered the issues of the multisite developments. The research aimed to develop a common concept base and consistent information exchange in both cases.

Tian et al. [65], Howard et al. [66], Shridhara et al. [67], and Yang and Tan [68] investigated the semantic relationships of the terminologies used in software

source code. They found that in the specific semantic area of Software Engineering, the synonyms between the words are different from the regular usage of those words in English. The authors have worked out methods to extract these specific synonyms from source codes and the corresponding comments. The results can also help identify the methods used in software, which supports the developers in finding a specific function or method during maintenance.

Itto et al. [69] focused on subtracting meronymy relationships from texts created in product development and customer service relations, whereas Yildiz and his workmates [70] studied the meronym relationships in Turkish raw text, applying lexico-syntactic patterns. Futia and his workmates [71] have developed a tool called SeMi to build large-scale Knowledge Graphs from structured sources semi-automatically.

Holter developed methods based on NLP techniques to translate requirements given in natural language into a structured semantic database [72]. A systematic literature review of using machine learning methods and NLP in requirements engineering is presented in the paper of Ahmad et al. [73].

2.3 Classification of Non-Functional Requirements

2.3.1 Vectorization and dataset

The source of the requirements is the collection of business documents, laws, regulations, and records of interviews. These documents contain texts written in natural languages. In order to be able to classify these texts, they have to be preprocessed. This transformation process converts the raw text into a vectorized form used for machine learning procedures. The procedure involves standard natural language techniques such as tokenization and filtering. For vectorization, the *tf-idf* model is one of the most common representations, which was also used in our classification experiments. This representation measures the importance of a word in a given document and produces a sparse matrix representing the whole text. The measure of *tf-idf* is formulated as:

$$tfidf(t, d, D) = tf(t, d) * idf(t, D) \tag{2.1}$$

where *t* denotes terms (words in our case), *d* denotes the document (sentences from the NFR dataset, or Stack Overflow posts in our case) and *D* denotes the collection of documents (the set of sentences or set of posts in our case). The *tf(t,d)* is the term frequency in a given document. The *idf* (inverse document frequency) is formulated as:

$$idf(t, D) = log \frac{|D|}{1 + |\{d \in D : t \in d\}|} \tag{2.2}$$

Before the vectorization, punctuation characters and stop words were removed from the input, and sentences were tokenized. For tokenization, the Keras Tokenizer [74] was applied to filtering, lower case converting, and splitting words. Stemming was also executed before converting the words into *tf-idf* based vector.

Two different datasets were applied to our experiments. For the first, the Tera Promise NFR dataset was used, which was constructed by the MS students of DePaul University [24, 25]. This dataset contains 625 requirements collected from 15 different projects. The sentences containing the requirements are classified into 12 classes, from which one class represents functional requirements, and the other 11 classes correspond to various types of non-functional requirements. The statistics about the classification and the related projects are shown in Table 2.1 and Figure 2.1. As it can be seen, regarding the *Portability class*, the dataset contains only one example, so we removed that example before our experiments were executed.

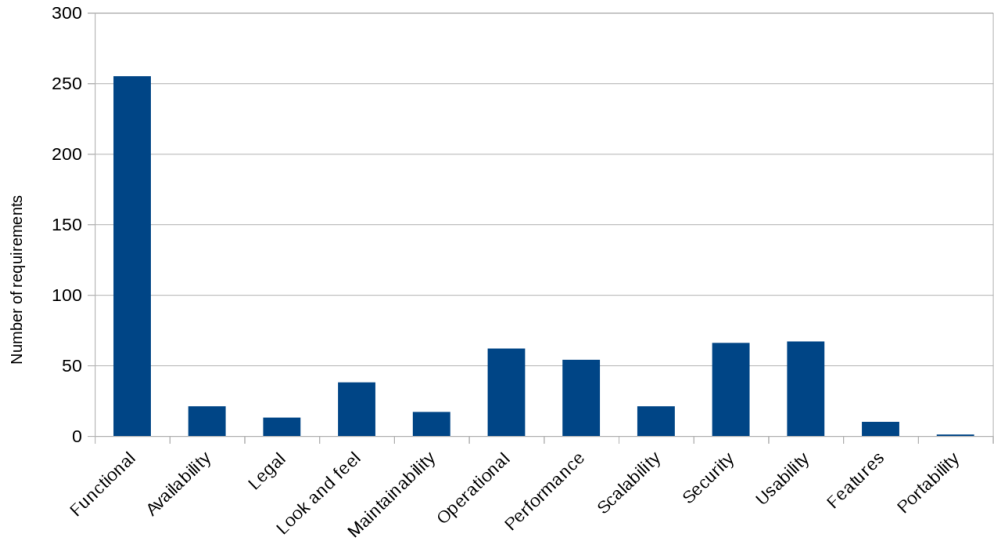
Table 2.1: *Requirement labels in the 15 projects of the Promise NFR dataset*

Requirement type		Project No															Total
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Functional	(F)	20	11	47	25	36	27	15	20	16	38	0	0	0	0	0	255
Availability	(A)	1	2	2	0	2	1	0	5	1	1	2	1	1	1	1	21
Fault tolerance	(FT)	0	4	0	0	0	2	0	2	0	0	0	2	0	0	0	10
Legal	(L)	0	0	0	6	3	0	1	3	0	0	0	0	0	0	0	13
Look and feel	(LF)	1	4	0	2	3	2	0	6	0	7	2	2	4	3	2	38
Maintainability	(MN)	0	0	0	0	0	4	0	2	1	0	1	3	2	2	2	17
Operational	(O)	0	0	7	6	10	15	3	9	2	0	0	2	2	3	3	62
Performance	(PE)	2	6	2	2	4	1	2	17	4	4	3	5	0	1	1	54
Portability	(PO)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Scalability	(SC)	0	3	4	0	3	4	0	4	0	0	0	1	2	0	0	21
Security	(SE)	1	3	10	10	7	5	2	15	0	1	3	3	2	2	2	66
Usability	(US)	3	6	8	4	5	13	0	10	0	2	2	3	6	4	1	67
Total NFRs		8	29	33	30	37	47	8	73	8	15	13	22	19	16	12	370
Functional		20	11	47	25	36	27	15	20	16	38	0	0	0	0	0	255
Total Requirements		28	40	80	55	73	74	23	93	24	53	13	22	19	16	12	625

The source of the second series of our experiments was a dataset queried from Stack Overflow. The dataset is composed of a sequence of tagged posts. Posts contain English texts, and sometimes code fragments occur. Stack Overflow is also an important source for research related to non-functional requirements [54, 55, 75, 76]. We have chosen posts tagged with *performance* or *test-related* labels for our experiments. *Testability* and *performance* are crucial factors for software quality, and these topics are also discussed thoroughly via Stack Overflow. The dataset was obtained from

the Stack Exchange Data Explorer[2]. This tool limits the overall size of the dataset queried at one time to 50,000 records. The obtained dataset contains posts where 20,166 records correspond to performance and 30,753 records to test. The Promise NFR dataset contains only one label for each example, but the dataset extracted from Stack Overflow is multi-labeled, so 919 posts are labeled both test and performance. We selected only two labels mentioned (performance and test) for extraction to obtain a well-balanced input. During the preprocessing, the labels of the dataset from Stack Overflow were also transformed using *one-hot encoding*, and the labels different from our interests were filtered out.

Figure 2.1: *Distribution of the Promise NFR dataset*



2.3.2 The experiments

There were two different experiment series conducted. In the first series, we accomplished classification processes using algorithms implemented in the *scikit-learn* library [77] on the dataset of Promise NFR and compared the results with each other, which we then also compared to the results obtained by Cassamayor et al. in their experiments [19]. The objective of these experiments was to determine the best classification algorithm implemented in the *scikit-learn* library for requirements classification tasks. The classification was performed using the *one-versus-rest* strategy for each class. In this strategy, the examined class is fitted against the other classes; therefore, the binary version of the classifiers was used if the classifier defined both the multi-class and binary versions. It has to be noted that some algorithms, like label propagation, do not have specific binary versions.

[2]https://data.stackexchange.com/

The classification algorithms used in this experiment were the following:

1. ***Naive Bayes:*** The Naive Bayes Classifiers are conditional probability models. Given an instance from the problem set, denoted by $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, the classifier calculates a conditional p probability for every C_k class from a finite class set. Calculating this conditional probability, called posterior probability, is difficult if the feature set of the instance vector is large. The calculation, however, can be simplified using the *Bayes' theorem*:

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})} \tag{2.3}$$

The $p(C_k)$ is the prior probability, the assumption of the classes' distribution, whereas $p(\mathbf{x}|C_k)$ is the likelihood, and the denominator is the evidence which is effectively a constant. As the denominator does not depend on the classes, it is omitted from applying the algorithm. When the numerator is expanded and using the *chain rule*, it takes the following form:

$$p(x_1, ..., x_n, C_k) = p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k)...p(x_n|C_k)p(C_k) \tag{2.4}$$

The method's name is naïve, which derives from the fact that an additional assumption is made, namely the hypothesis of conditional independence, i.e., it is assumed that the features defining the dimensions of $\mathbf{x}$ are mutually independent, conditional on category C_k. In this case $p(x_i|x_{i+1}, ..., x_n, C_k) = p(x_i|C_k)$. Under the principle of conditional independence, the original conditional probability formula takes the following form:

$$p(C_k|\mathbf{x}) = \frac{1}{Z}p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \tag{2.5}$$

where Z is the scaling factor, the evidence $Z = p(\mathbf{x})$.

The algorithm uses the *maximum a posteriori decision rule*. The Bayes classifier assigns a class label like the following:

$$\hat{C} = \underset{k\in\{1,...,K\}}{\arg\max}\, p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \tag{2.6}$$

- *Multinomial Naive Bayes:* This version of the algorithm assumes that the likelihood probabilities follow the multinomial distribution:

$$p(x_i|C_k) = \frac{N_{C_k i} + \alpha}{N_{C_k} + \alpha n} \tag{2.7}$$

where $N_{C_k i} = \sum_{\mathbf{x} \in T} x_i$ is the number of times feature i appears in a sample of C_k in the training set T, and $N_{C_k} = \sum_{i=1}^{n} N_{C_k i}$. The $\alpha \geq 0$ is the smoothing factor that prevents the zero probabilities in the calculations. The n is the number of features.

- *Gaussian Naive Bayes:* This version of the classifier assumes that the likelihood probabilities follow the Gaussian distribution:

$$p(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{C_k}^2}} \exp\left(-\frac{(x_i - \mu_{C_k})^2}{2\sigma_{C_k}^2}\right) \tag{2.8}$$

The mean μ_{C_k} and the variance σ_{C_k} are estimated using the ML method.

- *Bernoulli Naive Bayes:* In this case, the assumption is that the likelihood follows multivariate Bernoulli distribution. The dataset vectors can contain multiple features, but these features are assumed to be binary-valued.

$$p(x_i|C_k) = p(i|C_k)x_i + (1 - p(i|C_k))(1 - x_i) \tag{2.9}$$

where i is the indicator for class C_k.

2. ***Support Vector Machine with linear kernel:*** The Support Vector Machine constructs hyperplanes that separate instances of the training data given in the high dimensional space into two classes so that the distance between the two classes is maximized. Linear SVM calculates two parallel hyperplanes for the separation. The region between the two hyperplanes is called a margin. This task can be described as the following optimization problem:

$$\begin{gathered} \text{minimize } ||\mathbf{w}|| \\ \text{subject to } y_i(\mathbf{w}^T\mathbf{x_i} - b) \geq 1 \; for \; i = 1, ..., n \end{gathered} \tag{2.10}$$

The value of b can be either 1 or -1 denoting the two classes. Every $\mathbf{x_i}$ $i = 1, ..., n$ is a k-dimensional vector representing the instances of the training set.

The $\mathbf{w}$ defines the distance between the two hyperplanes as $\frac{2}{||\mathbf{w}||}$. After solving the optimization problem, the classifier can use the following equations:

$$\mathbf{x} \rightarrow sgn(\mathbf{w}^T\mathbf{x} - b) \tag{2.11}$$

where $sgn(x)$ is the sign function.

If the data are not linearly separable, the minimization of the following formula can be applied for the data separation:

$$\lambda||\mathbf{w}||^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x_i} - b))\right] \tag{2.12}$$

where λ determines the trade-off between the margin size and placing the instance on the right side of the margin.

3. ***Linear Logistic Regression:*** In logistic regression, also called maximum-entropy classification, the probabilities of the outcomes of a given instance can be described using the sigmoid function:

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w^T x}+b}} \tag{2.13}$$

where $\mathbf{x}$ is the given instance, the $\mathbf{w}$ is a column vector that determines the importance of the features of $\mathbf{x}$, b is the bias. The weight vector and the bias are determined during the learning phase, by minimizing the loss function, in this case, the cross-entropy loss function.

$$L(\hat{y}, y) = -[y \log \sigma(\mathbf{w^T x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w^T x} + b))] \tag{2.14}$$

The $\hat{y} = \sigma(\mathbf{w^T x} + b)$ denotes the calculated class, whereas y denotes the class labeled in the training set. As the loss function is convex, it has one minimum. In practice, the minimum value is usually calculated using the gradient descent algorithm.

4. ***Label Propagation:*** Label propagation is a semi-supervised graph inference algorithm. The algorithm builds a similarity graph on all elements of the input data set.

- The algorithm initially generates a connected graph. In order to reduce the memory consumption of the process, the number of neighbors of a point can be limited.
- The algorithm determines the weight of the edges. Those edges that connect closer points have higher weight values than the edges connecting farther points.
- The algorithm performs a random walk from the unlabeled points to find a probability distribution of reaching the labeled points. During the walk, edges with a higher weight are more likely to be chosen. This phase includes iterations while all paths are explored or the probabilities do not change.
- The unlabeled point is assigned the point label at the end of the maximum probability path.

5. ***Label Spreading:*** Label Spreading is a modified version of the Label Propagation algorithm. The algorithm iterates on the modified version of the original graph and normalizes the edge weights by computing the normalized graph Laplacian matrix. Besides, Label Propagation performs a hard label clamping, i.e., the labels of the labeled nodes cannot be modified. In contrast, Label Spreading applies soft clamping, meaning the labeled nodes' labels can be changed with a given probability parametrized by $\alpha \geq 0$.

   Note: The Laplacian matrix is calculated as $L = D - A$, where D is the degree matrix of the graph and A is the adjacency matrix. Normalization can be performed in multiple ways; for example, a random walk normalized graph can be calculated as $D^{-1}L$.

6. ***Decision Tree:*** A Decision Tree is a non-parametrized supervised learning procedure that builds a tree structure iteratively based on selecting features with the maximum information gain and thus separating the instances on the actual branch based on the value of the selected feature. The prediction is based on the value of the features of the given instance by selecting the tree's branches until reaching one of the leaves.

7. ***Extra Tree:*** The Extra Tree algorithm is an ensemble method that uses randomized decision trees to make the decision based on averaging the decision of the participant's trees. The trees used in the ensemble are built on both the randomly selected instances from the training set. The splitting points of these trees are determined by randomly drawing for each candidate feature and selecting the best of these randomly-generated values.

8. ***K-Nearest Neighbour:*** K-NN is a supervised learning algorithm presumes that the instances with the same label are close in the feature space. The algorithm works through the following steps:

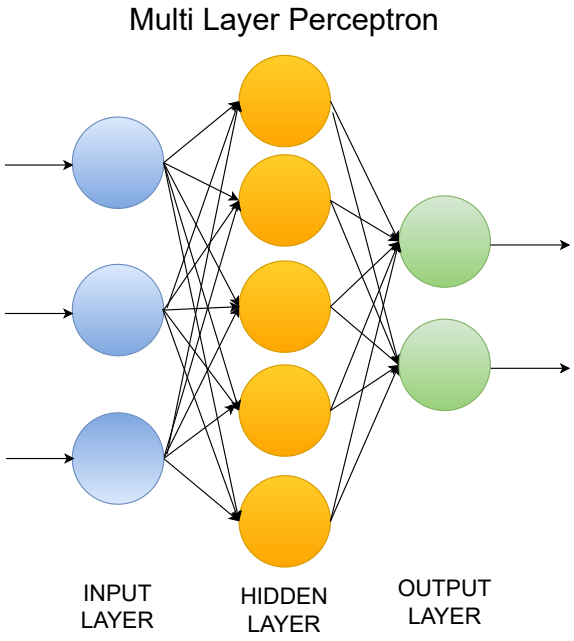
   - An integer K is selected to determine the number of neighbors.
   - The distances (usually the Euclidean distance) between the examples and the selected test point in the feature space are calculated.
   - The label for the selected point will be the most common label from the labels of the K nearest points.

9. ***Multi Layer Percepton:*** The perceptron model is a binary classifier inspired by the function of the neurons. The first model of an artificial neuron was the *Threshold Logic Unit* proposed by McCulloch and Pitts in 1943 [78]. The *perceptron* is a significant improvement of this model created by Frank Rosenblatt and presented in 1957 [79]. The model can be described using the following formula:

$$\sigma(\mathbf{w}^{\mathbf{T}}\mathbf{x} + b) = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathbf{T}}\mathbf{x} + b \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

   The $\mathbf{x}$ is the input vector, $\mathbf{w}$ are the weights of the features, b is the bias, and σ is the activation function.

Figure 2.2: *Multi Layer Perceptron with one hidden layer*



Multi-Layer Perceptron is a simple feedforward neural network where the perceptrons are organized into layers. Three layer types can be distinguished: one input, one output, and any number of hidden layers. An example of the model can be seen in Figure 2.2. The example provides a multi-layer perceptron with three neurons in the input layer and two in the output (classification) layer. It has only one hidden layer with five neurons. The network is fully connected, meaning that every neuron in a layer is fed by every output of the previous layer weighted by a learnable (w) parameter and modified by the bias (b) (*the figure does not show*).

The model is trained by minimizing a proper loss function, usually using the *backpropagation algorithm* [80] to calculate the weights and biases.

In the experiments, the default settings of the classifier were used, but the number of neurons in the hidden layer was set to 25, and the maximum number of iterations to 50. During learning, *Early Stopping* was enabled to avoid overfitting.

The experiments were executed using the *repeated K-Fold cross-validation* method. Both the number of groups and the repetition number were set to 10. The classification was performed using the *one-versus-rest* strategy for each class. Precision, recall, and F-measure were calculated for each test, each class by the corresponding *scikit-learn* method, and the results were averaged. The averaged variance was calculated as well, whereas the F-measure value was computed for every classifier using the averaged precision and recall because the averaged F-measure does not hold any useful information.

The Stack Overflow based experiments applied the Multinomial-, the Gaussian-, and the Bernoulli Naive Bayes, the Support Vector Machine with linear kernel, the Linear Logistic Regression, the Decision Tree, the Extra Tree, and the K-Nearest Neighbour classifiers from the *scikit-learn* library, and the experiments were supplemented with a *Tensorflow* and *Keras* based Fully Connected Neural Network implementation. The MLP (Multilayer Perceptron) model implemented in the *scikit-learn* was removed from these experiments because the *Tensorflow* implementation provides better scalability than the *scikit-learn* one, and the applied strategy is similar in the two cases. The Label Propagation and the Label Spreading models were also removed because a performance issue was detected when applying them on the Stack Overflow dataset.

The original dataset extracted from Stack Overflow contains 50,000 examples split into a train- and a test-set. The train set contains 35,000 train posts, whereas the test set contains 15,000 posts. The classifiers from *scikit-learn* were applied using the *one-versus-rest* strategy. The precision, recall, and the F1 measure were calculated for each class, and the averages were determined as described previously.

The Fully Connected Network was constructed using *Keras* libraries and the *Tensorflow* backend. The implemented model contains 1024 units in its first layer. The number of units and the input shape can be varied, but the shape has to be aligned with the dimension of *tf-idf* vectors. For regularization purposes, the *dropout* technique was applied. The *dropout* regularization selects a given proportion of the units randomly and inactivates them during the learning phase. The neurons' ratio to be inactivated is a hyperparameter that can be tuned during the experiments. The activation function of the first layer is the ReLU (Rectified Linear Unit) function. This function produces 0 for every negative input and any positive x returns with x:

$$f(x) = \max(0, x) \tag{2.16}$$

The second layer is a softmax layer whose output size is the same as the number of classes under investigation. The softmax function can be written as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \tag{2.17}$$

where K is the number of classes, $z_i = \mathbf{w_i^T x} + b_i$, $\mathbf{w_i}$ is the weight vector for the ith output neuron representing the ith class, and b_i is the bias for this neuron. The formula provides a probability value for every class used in the model. The network was trained using the categorical cross-entropy as a loss function, and the Adam optimizer was applied. The equation of categorical cross-entropy is:

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} \mathbf{1}_{z_i \in C_j} \log p_{model}[z_i \in C_j] \tag{2.18}$$

where N is the number of observations, K is the number of classes, C denotes the set of classes. The term $\mathbf{1}_{z_i \in C_j}$ is the indicator function of the ith observation that belongs to the jth class. The $p_{model}[z_i \in C_j]$ is the probability predicted by the model for the ith observation belonging to the jth class.

Adam (Adaptive Moment Estimation) is a gradient-based optimizer for stochastic objective functions that applies adaptive estimation of lower-order moments, which helps the algorithm not stuck in a local optimum. The optimizer combines the advantages of AdaGrad and RMSProp methods [81].

The update rule of Adam is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.19}$$

where $\hat{m}_t$ and $\hat{v}_t$ are calculated as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.20}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.21}$$

where m_t and v_t are the first- and second-order moments which are updated as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.22}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.23}$$

For β_1, 0.9 is proposed by the authors as the default value whereas for β_2, 0.999 and for ϵ 10^{-8}. The default values were used in the experiment. g_t is the value of the gradient calculated in tth step.

In addition to the performance of classifiers, the execution time was also measured during the experiments.

For evaluation, the precision, recall, and F1 metrics were applied. The precision is formulated as:

$$precision = \frac{tp}{tp + fp} \tag{2.24}$$

where tp is the true positive, which is the number of correct positive classification; fp denotes the false positive, which is when the classifier accepts the example but it still has to be rejected. This measure is also called a Type I error.

Recall can be formulated as:

$$recall = \frac{tp}{tp + fn} \tag{2.25}$$

where fn denotes the false negative, which is when the classifier rejects the example, but it has to be accepted. This measure is also called a Type II error.

The recall is also an important metric for our purpose because one of the possible usages of the models might be the identification of non-functional requirements from their textual context, and in this case, it is crucial to find them in the input. However, precision is just as important as recall. The adequate metric, in this case, is the F1. F1 is formulated as:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.26}$$

Note: F1 is a special case of F-measure with $\beta = 1$ value. The F-measure formula:

$$F_\beta = (1 + \beta^2) * \frac{precision * recall}{\beta^2 * precision + recall} \tag{2.27}$$

During the experiments using the Stack Overflow posts, the metrics were calculated using classification reports provided by the *scikit-learn* library. The report used those calculation methods applied in the experiments with the Promise NFR dataset. The averaged measures were calculated manually for the Promise NFR dataset, whereas in the case of Stack Overflow experiments, the classification report produced the average values. The report calculated four types of averages: the micro, macro, weighted, and sample averages.

Let y_l denote the set of label l predicted by a classifier and $\hat{y}_l$ the set of classes with true label l. Let $y = \cup y_l$ and $\hat{y} = \cup \hat{y}_l$, where $l \in L$ and L is the set of labels. Then the micro average of precision can be given as

$$P(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y|} \tag{2.28}$$

The formula for calculating the micro average of the recall:

$$R(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|\hat{y}|} \tag{2.29}$$

For macro averages, the formulas are the following:

$$P(y, \hat{y})_{macro} = \frac{1}{L} \sum_{l \in L} P(y_l, \hat{y}_l) \tag{2.30}$$

$$R(y, \hat{y})_{macro} = \frac{1}{L} \sum_{l \in L} R(y_l, \hat{y}_l) \tag{2.31}$$

Similar formulas can be used for sample averages, where S denotes the set of samples:

$$P(y, \hat{y})_{samples} = \frac{1}{S} \sum_{s \in S} P(y_s, \hat{y_s}) \tag{2.32}$$

$$R(y, \hat{y})_{samples} = \frac{1}{S} \sum_{s \in S} R(y_s, \hat{y_s}) \tag{2.33}$$

For weigted averages, the following formulas can be applied:

$$P(y, \hat{y})_{weighted} = \frac{1}{\sum_{l \in L} |\hat{y_l}|} \sum_{l \in L} |\hat{y_l}| P(y_l, \hat{y}_l) \tag{2.34}$$

$$R(y, \hat{y})_{weighted} = \frac{1}{\sum_{l \in L} |\hat{y_l}|} \sum_{l \in L} |\hat{y_l}| R(y_l, \hat{y}_l) \tag{2.35}$$

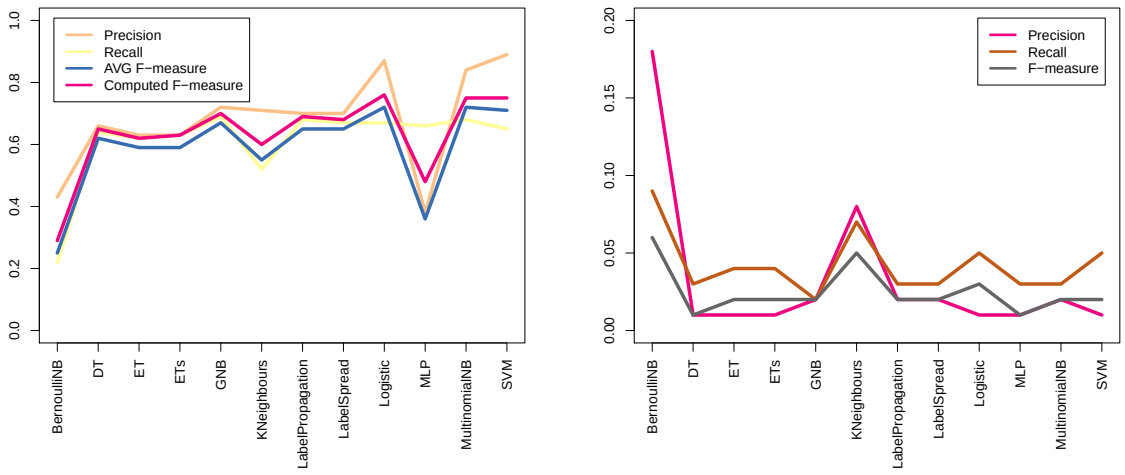
2.3.3 Results

Table 2.2: *Precision, recall, and F-measure of the classification of the Promise NFR*

Classifier		Average		Comp		Variance	
	P	R	F	F	P	R	F
BernoulliNB	0.43	0.22	0.25	0.29	0.18	0.09	0.06
DT	0.66	0.64	0.62	0.65	0.01	0.03	0.01
ET	0.63	0.62	0.59	0.62	0.01	0.04	0.02
ETs	0.63	0.63	0.59	0.63	0.01	0.04	0.02
GNB	0.72	0.69	0.67	0.70	0.02	0.02	0.02
KNeighbours	0.71	0.52	0.55	0.60	0.08	0.07	0.05
LabelPropagation	0.70	0.68	0.65	0.69	0.02	0.03	0.02
LabelSpread	0.70	0.67	0.65	0.68	0.02	0.03	0.02
Logistic	0.87	0.67	0.72	0.76	0.01	0.05	0.03
MLP	0.38	0.66	0.36	0.48	0.01	0.03	0.01
MultinomialNB	0.84	0.68	0.72	0.75	0.02	0.03	0.02
SVM	0.89	0.65	0.71	0.75	0.01	0.05	0.02

In Table 2.2, the measured averages of Precision, Recall, and F-measure related to the experiments on the Tera Promise NFR dataset with their averaged variance are presented. As mentioned in the previous section, averaging F-measure does not hold valuable information, so F-measure was computed based on average precision and recall. The averages of precision, recall, and F-measure values and their variance are illustrated in Figure 2.3. For comparison purposes, the results are illustrated using line diagrams. The results show that the SVM has produced the best precision value; however, the recall is only 65% which is the median value of the results. The Multinomial Naive Bayes and Logistic Regression have produced the best F1 values, but the F1 value of the SVM is only 1% lower. As one can see in the chart, the MLP has produced the worst result. The dataset size is too small for applying the Multilayer Perceptron classifier, and this result can be explained with the underfitted model.

Figure 2.3: *Precision, recall, F1 (left), and their variance (right) on the Promise NFR*



The variance of the measures can be seen on the right side of the chart. As one can see, the Bernoulli Naive Bayes and the K-Nearest Neighbours have resulted in the most significant variance, but the top 3 classifiers (Logistic Regression, Multinomial Naive Bayes, and SVM) have a minor variance. The average variance is relatively low; the values are lower than 5% for every metric.

As the results of the experiments present, regarding the precision, the Multinomial Naive Bayes Classifier, Support Vector Machine with linear kernel, and Linear Logistic Regression have produced the best values. The Naive Bayes Classifier was also found in former research to be the best classifier for the classification of requirement sentences compared to other classifiers such as the *tf-idf* classifier [19], the k-Nearest

Neighbour [16, 19], the Bittern Topic Model (BTM), or the Latent Dirichlet Allocation (LDA) [16]. Lu and Liang have found that the SVM classifier has performed best during their research [21].

The second series of experiments were executed on the dataset obtained from Stack Overflow. This dataset is large enough for machine learning purposes. While the Promise NFR contains 625 examples, this dataset contains 50,000. The main drawback regarding the Stack Overflow dataset is its noisy nature. Noisiness means, in our case, that there are several false or off-topic posts and code-fragments presented in the database.

Table 2.3: *Precision, recall, and F1 values on the Stack Overflow dataset*

Classifier	micro average			macro average		
	P	R	F	P	R	F
BernoulliNB	0.91	0.91	0.91	0.90	0.90	0.90
GaussianNB	0.82	0.81	0.81	0.84	0.84	0.81
MultinomialNB	0.92	0.91	0.92	0.92	0.92	0.92
DecisionTree	0.91	0.91	0.91	0.91	0.90	0.90
ExtraTree	0.84	0.84	0.84	0.83	0.83	0.83
LogisticRegr	0.95	0.95	0.95	0.95	0.95	0.95
KNeighbours	0.80	0.80	0.80	0.82	0.76	0.77
SVM	0.95	0.94	0.94	0.94	0.94	0.94
FullyConnected	0.96	0.94	0.95	0.96	0.94	0.95

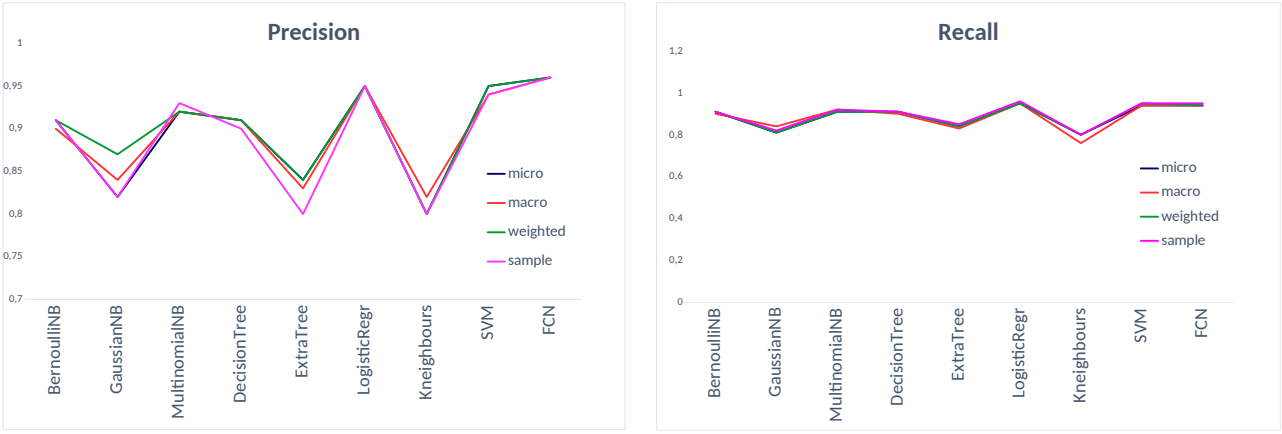
Table 2.4: *Precision, recall, and F1 values on the Stack Overflow dataset*

Classifier	weighted average			samples average		
	P	R	F	P	R	F
BernoulliNB	0.91	0.91	0.91	0.91	0.91	0.91
GaussianNB	0.87	0.81	0.81	0.82	0.82	0.81
MultinomialNB	0.92	0.91	0.92	0.93	0.92	0.92
DecisionTree	0.91	0.91	0.91	0.90	0.91	0.90
ExtraTree	0.84	0.84	0.84	0.80	0.85	0.80
LogisticRegr	0.95	0.95	0.95	0.95	0.96	0.95
KNeighbours	0.80	0.80	0.79	0.80	0.80	0.80
SVM	0.95	0.95	0.94	0.94	0.95	0.94
FullyConnected	0.96	0.94	0.95	0.96	0.95	0.95

The results of the experiments (Stack Overflow) are presented in Tables 2.3 and 2.4, respectively. The results are also illustrated in Figures 2.4 and 2.5. According

to the results, the Logistic Regression, the SVM, and the FCN (Fully Connected Network) have produced the best values. The FCN has yielded the highest values of precision and F-measure in all calculated averages. However, regarding the F-measure, the results of Logistic Regression are the same as those of the FCN. In the case of the recall, the Logistic Regression has outperformed the other classifiers; however, the results of the FCN are only 0.1% lower.

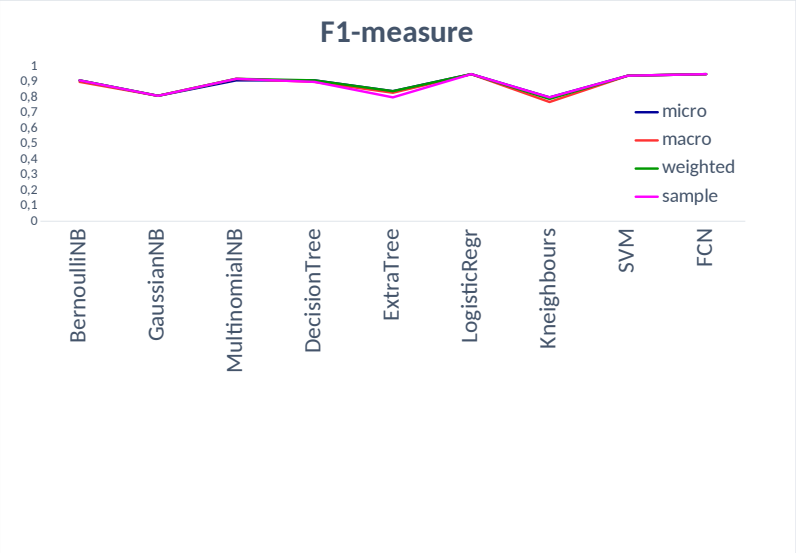
Figure 2.4: *Average precision and recall values on Stack Overflow dataset*



The averages were calculated using the micro, macro, weighted, and sample methods. The suitability of these methods depends on the input. Macro averages take the values calculated for every class and calculate their average. In the case of a well-balanced dataset, this average can be useful, but when the size of the classes varies, this method is not recommended. The micro average can be applied for any case, and it is the best choice if the size of the classes varies. If the size of the classes is also an important factor, weighted averages can be proposed. Sample averages calculate the averages of the values of the instances. As one can see in Figures 2.4 and 2.5, these values differ but can be used in a similar way to characterize classifiers.

The significance of the difference was tested using the first five classifiers, the FCN, the SVM, the Logistic Regression, the Multinomial Naive Bayes, and the Decision Tree Classifier. For the significance test, we have applied the one-way repeated measures ANOVA test. The *p-value* of the test is 0.000575, which is significant at $p < 0.05$. Because the assumptions of ANOVA as the samples are drawn from a normal distribution, and the variance homogeneity is not held, we also performed a repeated-measures Friedman Test [82]. The *p-value*, in this case, is 0.10739, which is not significant at $p < 0.05$. As mentioned before, the F-measure value is the same as the FCN and the Logistic Regression, and these results also support the outcome of the Friedman Test.

Figure 2.5: *Average F1 values on Stack Overflow dataset*



The variance was also calculated; however, only two classes were included in the classification. The results of the calculation are shown in Table 2.5 and in Figure 2.6. The results show that Gaussian Naive Bayes has produced the biggest variance regarding the precision, whereas the K-NN yielded the biggest value regarding the recall. Those classifiers, which presented the best values, also produced a small variance.

The results of the experiments on the Stack Overflow dataset confirmed the results on the Tera Promise NFR as the SVM, and the Logistic Regression has produced the best values using *scikit-learn* classifiers. Multinomial Naive Bayes has also given good results, but its performance appeared to be a bit worse than the top three results. These values suggest that the linear classifiers can be applied to classify, label, or tag the requirements given in natural language. The result of the FCN shows that if there are enough examples available, the neural network models can outperform the classic models of machine learning.

During the FCN test, we applied a simple, fully connected network with *dropout* regularization and the Adam optimizer. The hyperparameters such as the *number of neurons*, the *dropout keep ratio*, the *batch size*, or the *number of epochs* were also adjusted, and the results were checked. The best results were achieved using 1024 neurons in the first layer, 0.1 value for the dropout parameter and 100 for batch size. The training phase contained three epochs. The maximum difference between the worst and the best results was only 0.2% during the parameter changes.

As mentioned in the previous sections, the execution time was also measured for every classifier during both experiments. Execution time is also an essential factor for

Table 2.5: *Variance statistics on the Stack Overflow datase*

Classifier	Variance		
	P	R	F
BernoulliNB	0.045	0.045	0.045
GaussianNB	3.200	3.645	0.005
MultinomialNB	0.320	0.020	0.045
DecisionTree	0.045	0.080	0.045
ExtraTree	0.405	0.500	0.405
LogisticRegr	0.020	0.020	0.020
KNeighbours	0.320	6.480	1.125
SVM	0.045	0.320	0.320
FullyConnected	0.020	0.020	0.020

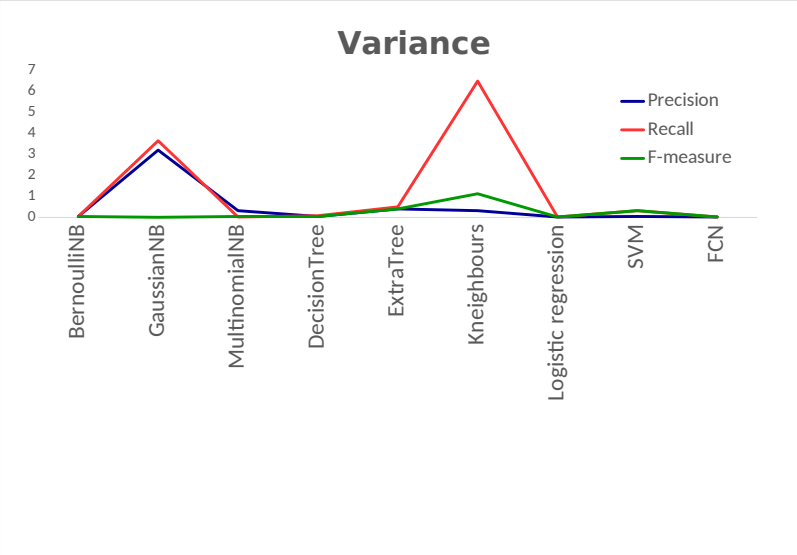
practice which can help business analysts choose the appropriate tool for requirement elicitation. The values of the execution time of the second series of our experiments are presented in Table 2.6. The two experiments were performed in a different hardware context; therefore, the results cannot be compared. However, the second series provides valuable information about the expectable execution time using a standard desktop environment with Intel I7-3770 3.9 GHz CPU and 8 GB RAM with 240 GB SSD. The operating system is Windows 10 64 bit.

Table 2.6: *Execution time of the learning processes on the Stack Overflow dataset*

Method	Execution time (s/epoch)
BernoulliNB	3.80
GaussianNB	6.67
MultinomialNB	0.99
DecisionTree	162.26
ExtraTree	3.33
LogisticRegression	11.18
KNeighbours	4290.83
SVM	10.91
FCN	50.92

The execution times are given in seconds. However, the actual execution time depends on the dataset under investigation, though these results can give us a hint at the choice if the running time is an essential factor. The best result has been achieved using Multinomial Naive Bayes. The K-Nearest Neighbours classifier has produced the worst result. This epoch lasted 4290.83 seconds which is very high compared to other classifiers.

Figure 2.6: *Variance statistics on the Stack Overflow dataset*



2.4 Mining Hyperonym Relations from Stack Overflow

Communication between people does not always go smoothly. The participants' cultural and professional backgrounds, their mental state, the communication channel(s) they use, and the purpose of the communication significantly influence the way they communicate and the set of terms they use. The communication partners may interpret expressions used in communication, particularly polysemous words, allegorical or metaphorical expressions, differently. The cultural and professional context is particularly relevant to the actual meaning. If the context is not clear, these expressions may confuse the understanding.

Misunderstandings in business communication can have tangible consequences, not just in terms of inconvenience but also in making the wrong decisions, leading to inefficiencies or loss of business results. Communication breakdowns in the business and corporate environment are significantly influenced by the terminology used in different professional fields and departments, often using the same terms with different meanings. Since information tools nowadays drive business and technology processes, it would be crucial that communication between business actors and software developers is seamless. However, the above mentioned issues are prevalent in the communication between these two areas even when an experienced business analyst is assessing the information system's requirements.

This chapter presents the background to the different semantic contexts, along with the possible solutions to resolve the resulting communication difficulties in the communication between the businesses and IT. In this research, a knowledge graph

representing the semantic environment of software development was constructed, and the meaning of terms used was examined in this environment compared to the everyday meaning of those terms. The results show that the use of semantic graphs can help determine the actual meaning of terms with different meanings, and thus their application can be an effective aid to support requirements engineering processes.

2.4.1 Introduction

The most frequent issues of software projects are requirements comprehension and establishing a common understanding among the different domain experts. The principal difficulty is the usage of the same terminologies with different meanings and contexts, along with the corresponding tacit knowledge, which is usually not articulated during the elicitation process. This dilemma exists between the developers and the customers in almost every communication situation.

The problem exists not only between different organizations but also internally in a team, leading to silo effects that have a detrimental impact on productivity and efficiency. Communication silos are well-studied phenomena in various fields of science, from organizational psychology to engineering [83, 84, 85, 86]. One of the biggest challenges in practical software engineering is to combat the communication silos in projects with multiple participants from different organizations and domains, not only in the requirements engineering but in every communication between the customers and the development team [87].

To reconcile the different meanings and catch the corresponding tacit knowledge, a mapping of the different semantic fields is needed. The notions used in a particular domain often provide another or overplus meaning of the words denoting them. The semantic field – also called the lexical field – is a set of lexemes describing a conceptual domain and its relationships with each other [32, 33]. The semantic field can be represented by directed graphs where the nodes are the terms related to the specific notion, and the edges represent the relationship among the notions. These constructions are called semantic networks [34]. Semantic networks can also be used as a psychological model of the notions of the world in the human mind [88, 89]. With the aid of semantic networks, terms from different semantic fields can be matched together either directly or via a proper upper ontology [35, 36]. This mapping process supports recognizing the different properties of the meaning and the catching of the tacit knowledge among the participant of different domains.

This chapter presents a method that extracts the principal notions used by software developers and creates the corresponding semantic networks based on one of the most common semantic relationships called hyperonyms (*'is a'* relationship). The dataset used for extraction is the collection of Stack Overflow (SO) posts. The SO

community has been making a significant effort to maintain the quality and professionalism of the site. On the one hand, Stack Overflow is a community portal for programmers and, on the other hand, a knowledge repository aimed to provide professional support for programmers in their everyday work. Consequently, the language and the conversations on SO reflect the semantic field common among software developers making the textual data of posts suitable for this research.

2.4.2 Theoretical background

The formation of the conceptual system of humans depends on several factors. One such important factor is the presence of common characteristics of perception. These properties influence the order of language acquisition and affect the usage of various languages [90]. The very first linguistic elements acquired by young children relate to nouns denoting objects and verbs connecting to simple movements. These lexical elements carry the same meaning for everyone within a given language; using them in communication does not cause misunderstandings. The more abstract concepts are built from already known lexical elements, and many are context-dependent. Similarly, the actual meaning of multi-meaning words is determined by their context. When the context is asymmetric in the communication, context-dependent words can introduce misunderstandings.

The meaning of an abstract concept in a given context marked by a particular term is determined by its relation to other concepts valid in the same context. Processing concepts marked with their terms by a computer program requires awareness of these relationships, which, in turn, can help clarify the actual meaning of a given term.

Concepts are abstract elements referring to the things of the world and their relationships. Concepts are mental objects, and according to Frege [91], they are abstract objects and can be organized in a hierarchical structure. This structure can be defined in various ways depending on the relationships considered. These relationships can represent linguistic relationships and other semantic relations like geographical or other connections.

The human brain is an efficient information processing system considering its flexibility and adaptability. Human memory, particularly long-term memory, can be modeled in an organized form called semantic network [88, 90, 92, 93]. Semantic networks are graph structures where nodes represent the concepts via their terms, and edges represent the links. In this research, the semantic network is considered a definitional network [34], which is also closely parallel with the semantic memory in the human brain [89, 90]. The relationship examined in this research is a model of super-subordinate relations or, as called in linguistics, hyperonym-hyponym relations. This bond is also typical in object-oriented analysis and design, the generalization-specification relationship, also called inheritance.

Definition 1 *Semantic network is a*

$$G = (C, R, \Sigma_1, \Sigma_2, s, d, l_1, l_2)$$

labeled directed multigraph, where C is the set of nodes representing the concepts of a given domain, and R is the set of edges representing the relationships between the elements of C. Σ_1 and Σ_2 are the alphabets of the labels corresponding to the nodes and edges, respectively. The $s, d : R \to C$ are the source and destination functions:

$$\forall r \in R,\ \exists (X, Y \in C) : r = (X, Y) \wedge X = s(r), r \wedge Y = d(r).$$

Similarly, $l_1 : \Sigma_1 \to C$ and $l_2 : \Sigma_2 \to R$ are the two labeling functions for the nodes and the edges, respectively.

The connection between the world and the concepts has a third component, without which communication would be impossible. This part is the marker associated with concepts, most often a linguistic phrase. This marker is called a term or, in other modalities, a symbol. The relationships among the objects of the world, the concepts, and the symbols can be represented via the semantic triangle shown in Figure 2.7 [94].

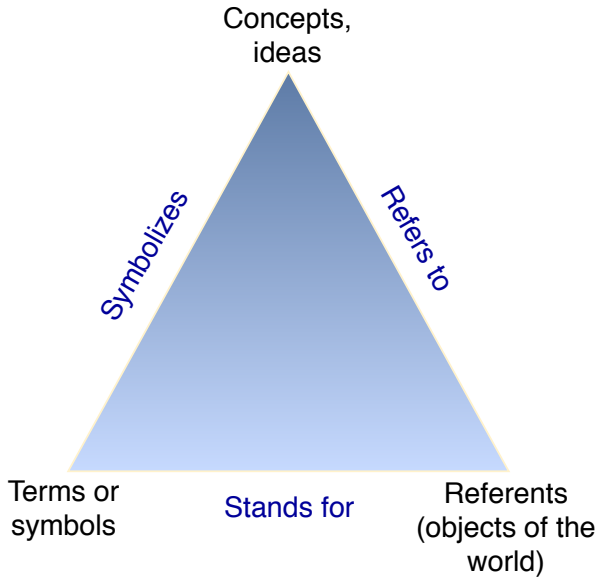


Figure 2.7: *Semantic (Odgen/Richards) Triangle*

Ideally, the relationship between concepts and their linguistic markers in a given language is injective, but in reality, this is not the case. Injection exists only in a narrower context called *semantic space*.

Definition 2 *Let C be an n-element set of concepts and $C_k \subseteq C$ its k-element subset $(k \leq n)$. Let*

$$S = (c_1, f_1), (c_2, f_2), ..., (c_k, f_k)$$

$c_i \in C_k$, $f_i \in F(i = 1, ..., k \leq n)$ be a series, where

$$F := \{f | f : C \times C_k \rightarrow [0, 1]\}$$

is a set of membership functions designating those $c_j \in C_k$ $(j = 1, ..., k)$ concepts that participate in defining a particular $c_i \in C$ concept $(i = 1, .., .n)$. Let $V = \boldsymbol{R}^k$ be a vector space over $\boldsymbol{R}$ and $\Psi : C \rightarrow V$ surjective mapping as follows:

1. $\forall c \in C : \Psi(c) = \boldsymbol{v}(v_1, v_2, ..., v_k) \ \in V$.

2. $v_i = f_i(c, c_i) \in [0, 1], \ (i = 1, ..., k)$*, where* $(c_i, f_i) \in S$ *and* $(c, c_i) \in C \times C_k$*, a fuzzy relationship between* c *and* $c_i, \ (i = 1, ..., k)$.

Given S, the vector space V is called the semantic space of C if the mapping Ψ is bijective.

Semantic networks restricted to various semantic spaces provide a comprehensive and tractable way for examining the semantic relationships among the concepts denoted by various word phrases. These relationships can be defined in various ways. We focus on the generalization and specialization, which are called *hypernymy* and *hyponymy* relations in linguistics.

Since computer processing requires clearly defined concepts, the following is a precise definition of the terms used in this study.

Definition 3 *Let C be the set of concepts and let L be a natural language with the alphabet Σ. Let Σ^* be the constraint of the set of words over L for the valid words, and word phrases of L. Let $X, Y \in \Sigma^*$, and let $M : \Sigma^* \rightarrow C$ be a mapping from the word phrases to the set of concepts. Let P be the set of properties describing the objects of the world, and let $\Pi : C \rightarrow P$ a mapping from the set of concepts to the set of properties. Hyponymy relation between X and Y is defined as*

$$Hyponym(X, Y) \iff \Pi(M(Y)) \subset \Pi(M(X)).$$

Hyperonymy relation, denoted as $Hyperonym(X, Y)$ is the inverse relation of the hyponymy.

Note: Other linguistic relationships, such as *meronymy*, *holonymy*, *synonymy*, and *antonymy* can be defined similarly.

2.4.3 Mining linguistic relationships from Stack Overflow posts

Stack Overflow is an extensive knowledge repository in software development and engineering. The phrases found in its posts might serve as a base to build a semantic network for the software development domain. Nevertheless, some critical issues need to be considered upon using this dataset. The posts often contain code fragments or unique character strings used in programming. Besides, many posts are written by non-English speakers; therefore, they might contain smaller or bigger grammatical errors.

In terms of content value, tacit knowledge in Stack Overflow posts can be trusted to be high quality because of the site's strict community control. Posts that fail to meet requirements established by the SO community are to be closed and eventually deleted. The SO community has been making a significant effort to maintain the quality and the professionality of the site [10]; therefore, posts with positive scores can be considered trustworthy.

Hyponymy and hyperonymy relations can be extracted from free texts using *lexico-syntactic patterns* proposed by Hearst [95]. These relationships represent the canonical *"is-a"* relationship, which can also be interpreted as the specialization and the generalization in object-oriented modeling. Since Hearst established the base models, the collection of *lexico-syntactic patterns* has been expanded. The momentum of this expansion comes from the rapid spread of web-based text mining. Our work adopted the patterns introduced in the article of Seitner et al. [60] The patterns presented in the paper were used in a slightly modified form, aiming to avoid confusion with extracting noun phrases from the text and writing more compact code.

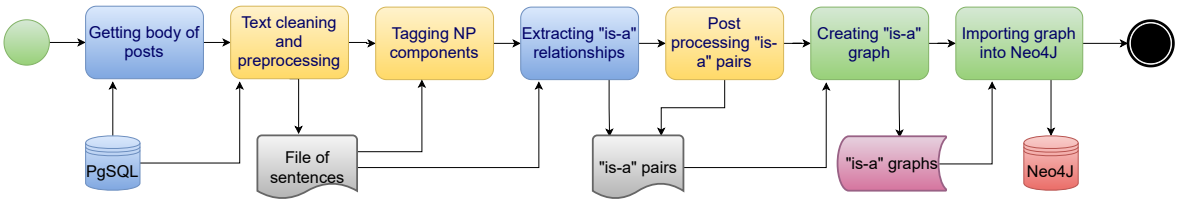


Figure 2.8: *Creating semantic network*

The whole process applied in the research is presented in the Figure 2.8. Posts used in the mining process are extracted from the Stack Overflow data dump created on March 4, 2019. The dump was migrated into a PostgreSQL 10.10 database and used for further processing. The total number of imported posts is 43,872,992, and from these, the amount of questions is 17,278,709.

The objective of the preprocessing phase is to provide a cleaned input text sliced into sentences for the mining process. The raw text contains elements that have to be eliminated or simplified. Besides, it has to be ensured that the processed text contains only relevant and accepted questions and answers. Therefore only posts that

obtained non-negative scores from the Stack Overflow community were considered. The preprocessing phase consisted of the following steps:

- Stack Overflow posts with non-negative scores were collected.
- Code blocks were replaced with the string `code example`.
- Hyperlinks were replaced with the string `link`.
- All HTML tags were removed from the text.
- `C++`, `C#`, and `F#` as well as their lower case counterparts were replaced with `cplusplus`, `csharp`, and `fsharp`, respectively.
- Some special characters were replaced with white space characters, but punctuation was retained.
- Multiple white spaces were combined into a single space.
- The text was then split into sentences.

In computer-based language processing, we need to formally define the examined linguistic structures, which approximates the set of the structures used in reality. For this purpose, the phrase structure grammar [96] is a suitable choice due to its algorithmic manageability.

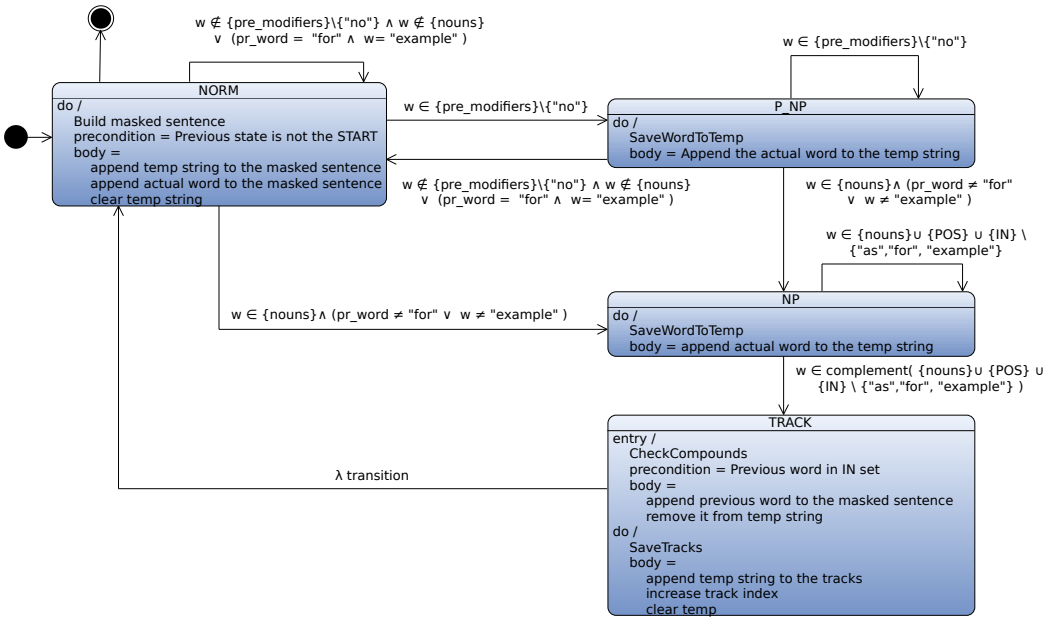


Figure 2.9: *Extracting noun phrases*

For matching *lexico-syntactic patterns* to the input text, *identifying noun phrases* [97] in the original text is required. The mining and the experiments were conducted twice. The first experiment examined the applicability of using semantic networks for collating different meanings of terms applied in the different semantic domains. For this purpose, a sample of questions containing 1.3 M sentences was selected randomly from the original dataset after the preprocessing phase. In this experiment, we applied simplified automation for recognizing noun phrases, as shown in Figure 2.9. The simplified automation could be used as we selected only the two most frequent lexico-syntactic patterns to extract the relationships:

- $\{NP_t\}\ (which)\ \{is|was|are|were\}\ \{NP_h\}$
- $\{NP_h\}\ \{for\ example|e.g.|i.e.\}\ \{NP_t\}$

NP_t denotes the hyponym part of the pattern, whereas NP_h means the hypernym member. The parentheses indicate optional components of the pattern, and pipe (|) signifies a choice among more than one constituent.

The graph obtained from Stack Overflow posts was compared with the WordNet[3] database for testing its applicability in practice. WordNet is an extensive lexical-semantic database of English. The different words are grouped into sets of synonyms called *synsets*. In WordNet, the synsets build those symbols mapped to the set of concepts. The definition of the given concepts is also provided. There are 117,000 synsets, and each of them is connected to other synsets based on various types of relations, like super-subordinate relations, also called hyperonymy and hyponymy among the noun-phrases, meronymy (*and implicitly the holonymy*), which represents the part-whole relationships among the synsets and antonymy by which adjectives are organized. For the complete view, we refer to [98].

WordNet is a semantic network based on a directed hypergraph in which the nodes are synsets, and edges represent the relations mentioned above. The semantic network obtained in this present study can be linked to WordNet by implementing a proper mapping of the set of nouns in the noun phrases (*NP*) vertices of our semantic network to the synsets of WordNet containing the particular noun. Using this mapping, we can discover an extended semantics of a particular term using the relationships from the corresponding synsets. WordNet, therefore, can serve as an interface among the different semantic fields.

After testing the concept, a more complete graph was built on the whole dataset, using a larger set of lexico-syntactic patterns and a precise and detailed definition of the grammar and automation that recognizes noun phrases. For this purpose, a phrase structure grammar was developed, which is shown in Figure 2.10. The $\langle . \rangle$ symbols denote the non-terminals, whereas the symbols without brackets denote

[3]https://wordnet.princeton.edu/

$$
\begin{aligned}
\langle \text{NP} \rangle &\models (\text{PDT})\ (\text{DET}\ (\text{CD}) \mid \text{PRP\$}) \\
&\quad (\langle \text{ADJPS} \rangle)\ \langle \text{HEAD} \rangle\ (\langle \text{GERS} \rangle)\ (\langle \text{PPS} \rangle)\ (\langle \text{REL} \rangle) \\
\langle \text{ADJPS} \rangle &\models (\langle \text{ADVS} \rangle)\ \langle \text{ADJ} \rangle\ (\langle \text{CONJ} \rangle\ \langle \text{ADJ} \rangle)^* \\
\langle \text{ADVS} \rangle &\models \langle \text{ADV} \rangle\ (\langle \text{CONJ} \rangle\ \langle \text{ADV} \rangle)^* \\
\langle \text{ADJ} \rangle &\models \text{JJ} \mid \text{JJR} \mid \text{JJS} \\
\langle \text{ADV} \rangle &\models \text{RB} \mid \text{RBR} \mid \text{RBS} \\
\langle \text{HEAD} \rangle &\models \text{NN(POS)} \mid \text{NNP(POS)} \mid \text{NNPS(POS)} \mid \\
&\quad \text{NNS(POS)} \mid \text{PRP} \mid \text{SYM} \mid \text{FW} \\
\langle \text{GERS} \rangle &\models \langle \text{GER} \rangle\ (\langle \text{CONJ} \rangle\ \langle \text{GER} \rangle)^* \\
\langle \text{GER} \rangle &\models \text{VBG}\ (\langle \text{NP} \rangle) \\
\langle \text{PPS} \rangle &\models \text{IN}\ \langle \text{NP} \rangle \mid \text{TO VB}\ (\langle \text{CONJ} \rangle\ \text{VB})^* \\
\langle \text{CONJ} \rangle &\models , \mid \text{CC} \\
\langle \text{REL} \rangle &\models \text{WDT} \mid \text{WP} \mid \text{WP\$} \mid \text{WRB}\ \langle \text{VP} \rangle
\end{aligned}
$$

Figure 2.10: *Grammar defined for NP recognition*

the terminals based on the POS (part of speech) tags defined in The Penn Treebank [99]. Parentheses denote the optional elements. Although the relative clauses are presented in the grammar, they are omitted from the current implementation.

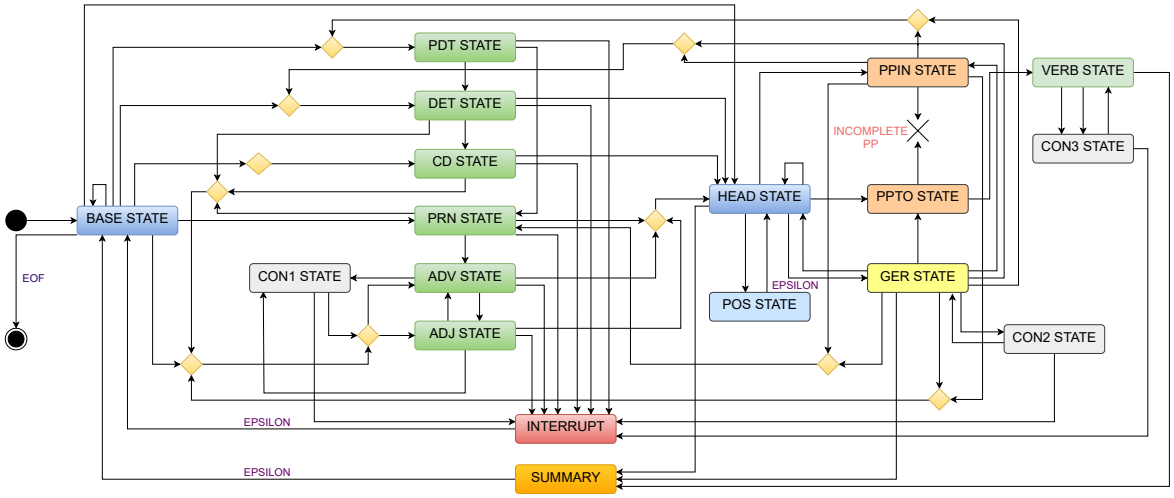


Figure 2.11: *NP Extractor Automation*

The sentences need further processing steps before the noun phrase (NP) recognition can be performed. All characters were converted into lowercase. Besides, the recognition process needs the part-of-speech tags; therefore, the sentences were converted into a series of *(word, POS tag)* tuple pairs.

The NP parser is based on the automation presented in Figure 2.11. The automation utilizes the POS tags to compute the proper transition. Although the recognition

of NPs and lexico-syntactic patterns is a separate procedure, the lexical elements *(e.g.: some, any, kind, sort, etc.)* used in lexico-syntactic patterns should be considered when recognizing NPs.

The POS tagging does not always produce the appropriate tag, which requires manual work to correct. In the experiments, two important cases were detected and corrected. In the expression of *sound/s like*, the POS tag of *sound* was corrected to VB (verb) and in the case of the phrase of *operating system*, the POS tag of *operating* was corrected to JJ (adjective).

By running the procedure described above, the noun phrases recognized by the algorithm are tagged using <NP > and </NP > tags in the string containing the original tokens. For example, the sentence *"Flour or any other grain can be found in the kitchen"* is transformed to *"<NP > flour </NP > or any other <NP > grain </NP > can be found in <NP > the kitchen </NP >"* series of tokens.

The token series obtained in the NP recognition process was the input for the extraction of relations fitted to lexico-syntactic patterns presented in Table 2.8 . The extractor was written using regular expressions carefully designed to identify the patterns. The pairs of related NPs were written in a CSV file from the recognized terms. The first part of the pair is the more specific NP *(hyponym part)*, and the second part is the more general one *(hyperonym part)*. The pairs were only saved if neither of the two parts of the pairs was a single pronoun.

The lexico-syntactic patterns used in the extraction process assume an input text written with proper English grammar. However, Stack Overflow posts are often written by non-English speaking users who sometimes make grammatical errors. These errors might result in a wrong relation extracted from the text. Therefore, a few post-processing steps have been introduced to reduce the number of mismatched pairs.

During the post-processing, pairs in which one part is empty or contains a single character other than *c*, *f*, and *b*, and those in which both parts are the same, have been deleted. If only one part of the relation is a proper noun, that part was considered the hyponymy part. If both parts of the relationship are proper nouns, or neither of them, the relationship was checked against the WordNet, and was corrected according to it.

After the post-processing phase, the extracted pairs are ready for graph building. These pairs provide the set of edges of the semantic graph. Unfortunately, duplications can also occur among these pairs, which must be removed during the graph building process. For building the graph, the `networkx` Python package was utilized. The resulting graph was then imported into a Neo4J[4] database.

[4]https://neo4j.com/

2.4.4 Analysing the extracted semantic net

The resulting pairs of the extraction are the edge list of the semantic network under investigation. This network is represented as a directed graph G. The direction of the edges represents the Hyperonymy relations: $(A, B) \in \{Hyperonymy(X, Y) : X, Y \in NP\}$ represented as $A \rightarrow B$. Let us consider the (*'a salt'*, *'a string'*) pair. In this case, it denotes the chunk *'the salt is a string'* , which means the *salt* is a specific *string*, a random data in the cryptography used as an additional input for hashing functions, hence the *string* is a generalization of the *salt* in this context.

In the experiment aimed at examining the applicability of the graph, the resulting set of edges was extended using the WordNet database. The mapping process applied the head of the noun phrase as a key for searching the generalization of that particular phrase in the WordNet. All generalizations based on WordNet were determined for these nodes.

Using the above example, the head of the source is *salt*, which has many generalizations in WordNet, such as *compound, chemical_compound, flavorer, flavourer, flavoring, flavouring, seasoner, seasoning, taste, taste_sensation, gustatory_sensation, taste_perception, gustatory_perception*. In this case, edges from the node *'salt'* to every element above were added to the list of edges.

The graph representing the semantic network was created from the set of edges using the `networkx` Python module[5]. This graph contains edges extracted from Stack Overflow and the mapped nodes from WordNet. The network has 41,501 nodes and 70,698 edges. Among the nodes, there are 5,001 that contain self-loops.

Figure 2.12 presents a small subnetwork with edges representing a connection between Stack Overflow and WordNet. The wealthy meaning of a particular word can be recognized if the different semantic domains are linked. For example the node *salt* can be connected to node *compound* which can be either the *compound key* from the domain of software development, or an *acid* from the domain of chemistry.

Table 2.7: *Differences between relations in Stack Overflow and WordNet*

General term	Stack Overflow	WordNet
default	consistent across browsers	[delinquency]
weak password hashes	des	[ ]
a defect	a bug	[birth defect, congenital anomaly, congenital defect, congenital disorder, . . .]
an accidental complexity	the clunkier syntax	[complicatedness, complication, knottiness, tortuousness, elaborateness, . . .]
a one-way operation	hashing	[commission, idle, running, rescue operation, access, memory access, . . .]

The Stack Overflow semantic network links software engineering terms together, sometimes representing deep domain knowledge. On the contrary, WordNet captures the general meaning of terms. The difference of the captured relations is demonstrated in Table 2.7. The first column contains the general term, and for each of

[5]https://networkx.github.io

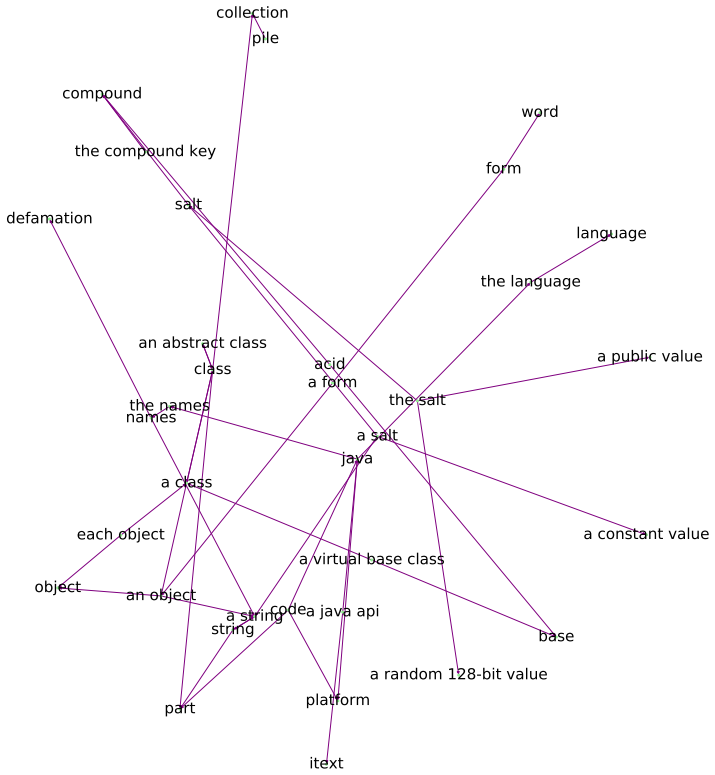


Figure 2.12: *Subgraph of the resulting semantic network*

these, we present specific terms from the two different sources in the respective columns. Well-known notions in software engineering like a defect or hashing are linked to more specific terms than WordNet. Making connections between the two networks would facilitate the common understanding of communicating with parties from different backgrounds.

In the second experiment, the whole imported database was used for building the semantic graph. The total number of imported posts from the dump created on March 4, 2019, is 43,872,992. Only posts with non-negative scores were considered for further processing, which is 42,160,482; this is 96.1 % of the original posts. From this dataset, 137,440,998 sentences were created during the sentence-based tokenization process, and 7,583,195 hyponymy and hyperonymy relations were extracted. The ratio is quite small; approximately only 5.5 % of the sentences contain the studied relations. It is important to note that the amount of relations imported into the semantic graph is smaller due to the filtering and merging method applied during the graph creation.

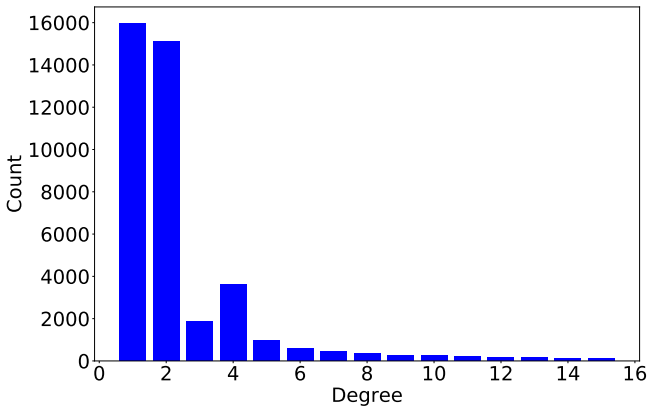


Figure 2.13: *Degree distribution of the network*

Table 2.8 presents the distribution of the extracted relations from this set. According to the distribution, the following three lexico-syntactic patterns are used the most dominantly in Stack Overflow posts: $NP_t\ is|are|was|were\ (a|an)\ NP_h$, $NP_t\ as\ NP_h$ and $NP_h\ like\ NP_t$. In turn, the occurrence of the following patterns is marginal: $NP_h\ example\ of\ this\ is|are\ NP_t$ and $NP_t\ or\ the\ many\ NP_h$.

The statistical results were compared to those of Seitner et al. [60] They applied a similar lexico-syntactic pattern-based mining on the dataset obtained from CommonCrawl[6] using a slightly different grammar for NP identification and, therefore, a slightly different set of patterns. Despite the differences, we found that patterns followed a similar trend. Interestingly, according to Seitner et al., the most commonly occurring pattern is a sub-pattern of the most frequent pattern found in our study ($NP_t\ is|are|was|were\ (a|an)\ NP_h$). The incidence of the other patterns is similar, although there are some differences. For example, the frequency of the pattern $NP_t\ as\ NP_h$ is significant in our case, while in Seitner's study, this is not typical.

The resulting semantic graph contains 3,926,617 nodes and 7,413,639 relationships. The considered *"is-a"* relationship is directed. The order of the input pairs defines the natural direction and points from the more specific term to the more general one. The natural degree of a given node indicates how many examples of different higher-order concepts can be represented with that same node. In the opposite direction, the degree of a node indicates the number of possible subtypes related to the generality or specificity of the concept denoted by that node.

[6]https://commoncrawl.org

The distribution of the degree in the normal direction is presented in Figure 2.14. Looking at the relationship in the opposite direction, we get a similar distribution. It can be seen in the figure that the degree of the node *code example* excels from the distributions. The difference between the degree of this node and that of the second-highest is 106,653 in the normal *(hyponymy → hyperonymy)* direction and 208,263 in the reverse *(hyperonymy → hyponymy)* direction.

Table 2.8: *Distribution of the patterns in the extraction results*

Lexico-syntactic pattern	Number of cases
$NP_t\ is\|are\|was\|were\ (a\|an)\ NP_h$	3,490,476
$NP_t\ as\ NP_h$	2,143,334
$NP_h\ like\ NP_t$	1,033,497
$NP_h\ such\ as\ NP_t$	278,078
$NP_t\ and\|or\ (any\|some)\ other\ NP_h$	207,079
$NP_h\ especially\|esp(.)\|including\|inc(.)\ NP_t$	117,151
$NP_h\ for\ example\ NP_t$	66,968
$NP_h\ except\ NP_t$	44,766
$NP_h\ e.g.\|i.e.\ NP_t$	44,295
$NP_h\ other\ than\ NP_t$	33,956
$NP_t\ (is)\ one\ of\ the\|these\|those\|this\|that\ NP_h$	30,996
$NP_t\ which\ look(s)\|sound(s)\ like\ NP_h$	28,589
$such\ NP_h\ as\ NP_t$	14,969
$compare\ NP_t\ with\ NP_h$	10,992
$NP_h\ compared\ to\ NP_t$	8,106
$NP_h\ which\ is\|are\ similar\ to\ NP_t$	6,385
$NP_h\ in\ particular\ NP_t$	5,671
$NP_h\ mainly\|mostly\|notably\ NP_t$	3,866
$NP_h\ particularly\|principally\ NP_t$	3,035
$NP_h\ which\ is\ called\|named\ NP_t$	2,924
$NP_h\ whether\ NP_t\ or$	1,957
$NP_t\ is\|was\|are\|were\ a\ kind\ of\|kinds\ of\ NP_h$	1,936
$NP_t\ like\ other\ NP_h$	1,813
$NP_t\ is\|was\|are\|were\ a\ form\ of\|forms\ of\ NP_h$	799
$NP_t\ is\|are\ example(s)\ of\ NP_h$	623
$NP_t\ is\|was\|are\|were\ a\ sort\ of\|sorts\ of\ NP_h$	620
$examples\ of\ NP_h\ is\|are\ NP_t$	276
$NP_t\ or\ the\ many\ NP_h$	21
$NP_h\ example\ of\ this\ is\|are\ NP_t$	17

The node with the second highest degree in the normal direction is the node *below* with a score of 148,435. This result seems to be a mistake because the word *below* can be a preposition or an adverb. The expected results, however, should only be

NPs. This particular word, *below*, however, can sometimes behave like a noun *(as well as an adjective)* [100]. Consider the following example from a Stack Overflow post: *"and below is the python interpreter setting on pycharm..."*. In this case, below is recognized by the NLTK POS tagger as a noun; therefore, the NP extractor also recognizes it as a noun phrase. The extracted relationship is *"the python interpreter setting on pycharm | below"* in this case. The culprit is the phrase *"below is"*. The tagger recognizes all similar occurrences as nouns.

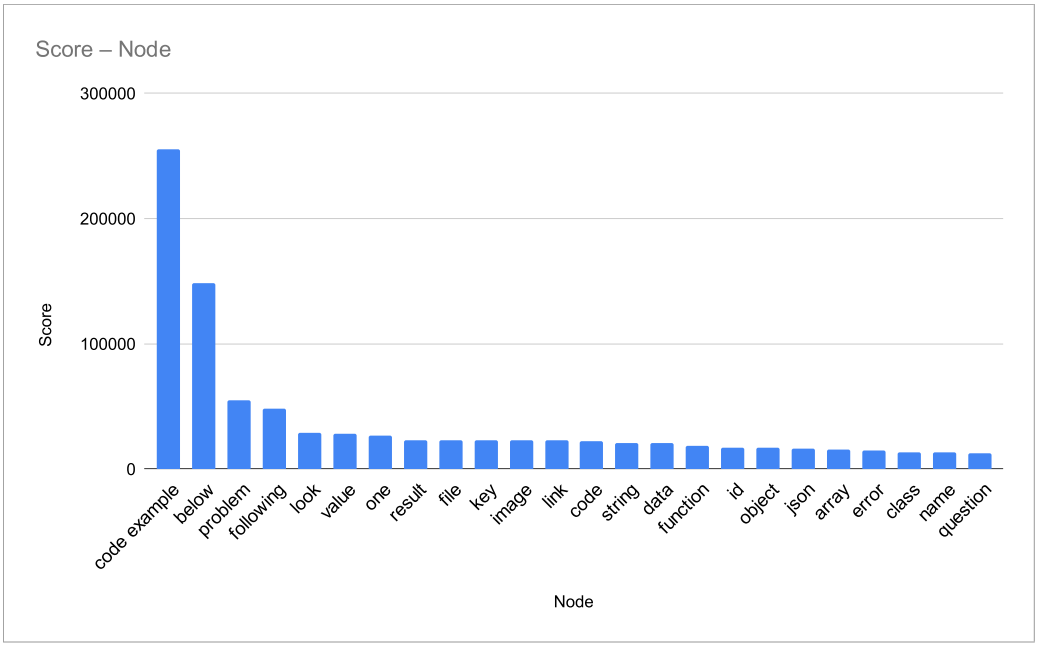


Figure 2.14: *Distribution of degree in hyponym-hyperonym direction*

The node *below* and other similar adverbs, such as *before, after, following, next*, and *above* are not interpretable without their actual context. Therefore, nodes representing these words can be removed securely from the semantic network, including their counterparts with either definite or indefinite articles. Similarly, placeholders, such as the phrase *code example* or the word *link* are replacing specific objects – code snippets or hyperlinks, respectively – and they neither provide a refinement nor a more general meaning of any terms. These nodes can also be discarded from the semantic network. After discarding the aforementioned nodes, 3,926,609 nodes and 6,059,017 relationships remained in the resulting graph. The Figure 2.15 presents a subgraph of the resulting semantinc network.

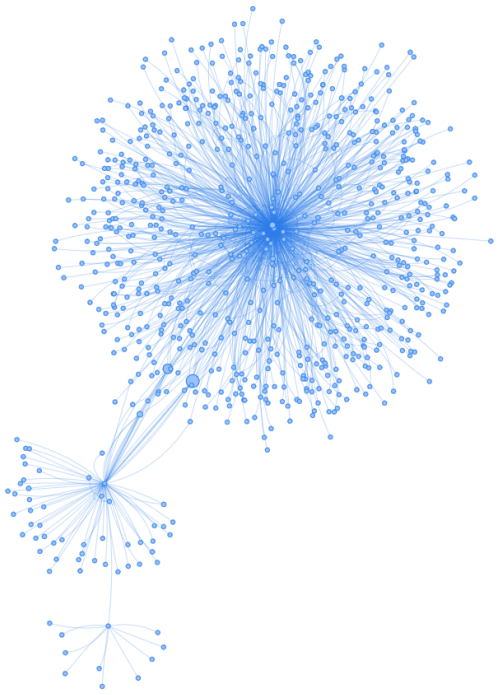


Figure 2.15: *Subgraph of the resulting semantic network*

The degree distribution of the resulting graph was also examined. The dataset was tested using the *Goodness of Fit method* [101]— based on the *Kolmogorov-Smirnov distance* [102] against some well-known distributions, such as power-law, Weibull, and lognormal distribution; however, the result of the fitting was not convincing in the case of hyponymy $\rightarrow$ hyperonymy direction. In the light of the negative result, a deeper analysis was performed using the `fitdistrplus` [103], `GAMLSS` [104] and `poweRlaw` [105] packages for R, and the result was tested using the *Goodness of Fit method* based on the *Kolmogorov Smirnov statistics*. Examining the distance in the square of skewness and kurtosis space of the actual distribution from some theoretical distributions indicates that a specialized gamma distribution can be fitted. The result of the analysis shows that in the case of the hyponymy $\rightarrow$ hyperonymy direction, the distribution follows the *Box-Cox Power Exponential Distribution* [106], according to which the probability density function is given by the following formula:

$$f_Z(z) = \frac{\tau}{c^{2^{1+\frac{1}{\tau}}}\Gamma(\frac{1}{\tau})} exp(-0.5|\frac{z}{c}|^{\tau}) \tag{2.36}$$

for $-\infty < z < \infty$ and $\tau > 0$, where $c^2 = 2^{\frac{-2}{\tau}}\Gamma(\frac{1}{\tau})\Gamma(\frac{3}{\tau})^{-1}$. This distribution is specified through the transformed random variable Z, where the original random variable is Y.

$$Z = \begin{cases} \frac{1}{\sigma\nu}\left[(\frac{Y}{\mu})^{\nu} - 1\right] & \text{if } \nu \neq 0 \\ \frac{1}{\sigma} log(\frac{Y}{\mu}) & \text{if } \nu = 0 \end{cases} \tag{2.37}$$

for $0 < Y < \infty$, where $\mu > 0, \sigma > 0$ and $-\infty < \nu < \infty$. The parameter values given by the fitting process are $\mu = 1094.857, \sigma = 0.5237, \nu = -1.074, \tau = 150.69$. The *Kolmogorov-Smirnov distance* is 0.038, which is significant using the confidence with $\alpha = 0.05$ and the sample size $N = 1000$. The *p-value* of the fitting process is 0.47.

In the case of the opposite direction, the distribution follows the *Power-Law Distribution*, according to which probability density function can be described using the following formula:

$$f(x) = \frac{\beta - 1}{x_{min}} \left(\frac{x}{x_{min}}\right)^{-\beta} \tag{2.38}$$

where $\beta > 0$. The *Kolmogorov-Smirnov distance* is 0.03, which is significant at $\alpha = 0.2$ with the sample size $N = 1000$. The *p-value* is 0.32 meaning that the result is acceptable (the null hypotheses cannot be discarded).

The average clustering coefficient is 0.017, meaning that only 1.7% of the concepts tend to form triadic closures. A possible explanation of this small number can be that large degree nodes connect the communities formed in the knowledge graph. This phenomenon can be interpreted as the concepts being defined based on a few core concepts. Further studies are needed to confirm this conjecture.

2.5 Conclusions and future works

Identifying and classifying non-functional requirements is a crucial duty of business analysts, which can be demanding and error-prone without using a proper tool. The collection of the requirements expressed using natural language form originates from customers and stakeholders. Natural Language Processing techniques and Machine Learning can support processing the requirements given in the textual form.

Several studies have been carried out to identify appropriate machine learning methods for requirement classification tasks to support business analysts in their elicitation process. Classification is crucial in the selection process, supporting the identification of non-functional requirements that are often not sufficiently highlighted by stakeholders. This research investigated the classification of non-functional requirements given in natural language using natural language processing tools and machine learning. Various machine learning methods were examined along with the transformation of the textual input into an appropriate form which is processable

by implemented machine learning algorithms. The methods have been compared using precision, recall, and F1 metrics, and a significance test has been performed. Two series of experiments were executed. The small-sized Tera Promise NFR dataset was applied for the first series, and a large dataset had been extracted from Stack Overflow for the second experiment.

The experiments show that the linear classification algorithm produced the best values using both the small and the large datasets. The winners are Multinomial Naive Bayes, Support Vector Machine, and Logistic Regression, but Decision Tree also produced satisfactory results. Using the dataset from Stack Overflow, the Fully Connected Network produced the best values and outperformed the other classifiers.

The performance metrics and execution times present that using NLP and machine learning techniques provides a reasonable solution for the base technology of tools supporting business analysts in the requirement elicitation process. The precision and recall obtained can be above 95 %, a sound starting point for practical usage.

Our results on the classification were presented at the International Conference on Computational Science and Its Applications in 2018 [II] and the Information Technology and Control journal in 2019, issue 48(3) [I].

Over the last few years, the linguistic modeling provided by transformers has opened up new horizons in the classification of requirements, allowing semantic relations to be taken into account. The possibilities for deep learning have been limited due to the small amount of tagged data available, but transformers are usually pretrained using various language corpora, and refining them for the current task can be done with much fewer examples. Our future research plans will focus on using transformers that also model semantic relations to not only classify but also produce requirements in a canonicalized form, including both functional and non-functional requirements.

A significant proportion of communication difficulties among people and organizations stem from distinct interpretations of the linguistic elements used in communication. The only way to deal with semantic differences caused by different contexts is to clarify the meanings of terms used, for which semantic networks are suitable tools.

This research examined the semantic network structure and gave a precise definition of both semantic networks and the concept of semantic space. The sound definition is necessary to handle semantic relations algorithmically. In order to extract semantic relations from natural language text, lexico-syntactic patterns were used in the form of regular expressions. Pattern recognition also requires the identification of noun phrases for which a phrase-structure grammar was developed.

There is a wide range of possibilities for representing semantic relations; this research focused on hypernym-hyponym relations. The interaction in Stack Overflow

posts was used to model the semantic domain of the software development environment, and the hyponym-hyperonym relationships were extracted from these discourses. Two experiments were performed; the first one compared specific concepts used in the software development domain using a smaller set of relations extracted from Stack Overflow with their counterparts used in the general semantic environment, that WordNet models. The second experiment extracted discourse from the entire Stack Overflow database using a more complete set of lexico-syntactic samples and examined the resulting graph.

The investigation of the yielded semantic graph shows that relatively few sentences from the available textual corpus contained structures fitting the relationships examined. The distribution of patterns also shows that the users prefer using particular patterns to others. The semantic network structure suggests that the concepts in the studied environment are built from a small number of basic concepts. This phenomenon can be seen on the graph; several nodes are connected to nodes representing the basic concepts. This phenomenon, however, needs further investigation.

Our results related to the usage of semantic networks were presented at The First International Workshop on Knowledge Graphs for Software Engineering (colocated with ICSE 2020) [VII] in 2020 and The Seventh International Conference on Fundamentals and Advances in Software Systems Integration 2021 [V].

As a continuation of the research, the meronymy-holonymy relationship was extracted, and the investigation of deep learning methods in extracting the relationships is in progress. The extended procedures will be used to extract semantic relationships from the discourse of other semantic domains.

The author of this Ph.D. thesis group is responsible for the following contributions presented in this chapter:

I/1. The author has developed preprocessing methods and a vectorization process applying the `tf-idf` representation form.

I/2. The author has implemented scripts responsible for executing the classification experiments using various machine learning models implemented in the `scikit-learn` library.

I/3. The author has implemented a simple neural network applied in the classification experiments based on the Stack Overflow samples.

I/4. The author executed the experiments, compared the results of the classifiers, and identified the best classifiers.

I/5. Based on the investigation of the semantics of the linguistic expressions, the author established a solid definition of semantic space and semantic networks, respectively.

I/6. The author has implemented preprocessing steps to extract posts from the Stack Overflow database and separate them into proper sentences cleaned from the auxiliary characters and noise.

I/7. The author has implemented a set of regular expressions based on the lexico-syntactic patterns representing the hyperonym-hyponym relationships found in the literature.

I/8. The author has developed a phrase structure grammar and an automatization to recognize noun phrases in the text. To the best of our knowledge, the grammar provided by the author is the most general formalized solution available in the literature.

I/9. The author has developed a simplified automatization for recognizing the noun phrases considering only a small set of the lexico-syntactic patterns.

I/10. The author has built a semantic network based on the hyperonym-hyponym relationships, representing the semantic field of the software development community based on the Stack Overflow post utilizing the lexico-syntactic patterns.

I/11. The author has investigated the structure of the resulting network and described its structure.

I/12. The author compared the smaller network resulting from the mining process with the semantic network representing the common knowledge provided by WordNet.

Chapter 3

Investigating Developers Interactions Using Natural Language Processing And Deep Learning

"In theory, there is no difference between theory and practice. But, in practice, there is."

Jan L. A. van de Snepscheut

Software development has grown into the leading industry of the last decades. Numerous devices used in everyday situations are based on software-driven systems. Whether looking at communication, learning, transport, entertainment, industrial production, or healthcare, software-driven systems can be seen everywhere. Nevertheless, such widespread technologies inevitably imply the need for specialization in the various branches of the given technology. Software developers cannot be familiar with all programming languages, libraries, frameworks, and platforms; it is usually only the tools they use in everyday work that they know in sufficient depth to develop effectively. However, it is often the case that when project tasks change a developer might have to deal with less familiar technologies. In the age of the Internet, it is less efficient to consult books, often out of date, to find solutions to problems encountered in day-to-day work, but more efficient to look for solutions to the issues raised via the Internet. There are many professional articles, blogs, and tutorials available that help developers in their work, as well as developer community sites, which aim to facilitate communication between developers to help them solve their daily tasks effectively.

The software development community's most popular Question and Answer (Q&A)

site is Stack Overflow[1], launched in 2008 by Joel Spolsky and Jeff Atwood [107]. The site aims to support both professional and enthusiast developers in their everyday labor activities. It is important to emphasize that the site was created to support the work of actual practitioners, so, initially, it was not meant to support mentoring the beginners, participating in the solution of student assignments, or providing a platform for either educational or informal conversations. Since the beginning, the portal's operability and quality have been primarily maintained by experienced users, who devote their free time to volunteering.

As the community grew in size and the less experienced users became more active on the site, the moderator's duties became increasingly demanding. However, the community remained determined to maintain the professionalism and quality of the site, and often closed issues that did not meet the requirements established by the community. At the same time, less experienced users found it increasingly difficult to adapt to the strict quality requirements, which can be daunting for newcomers, and increasingly encountered the closure of questions without response, which they experienced as frustration. The reason for closing questions is not always apparent, which has led to disapproval from not only novices but also the proficient users [108, 109].

In this thesis group, research results are presented that support the work of both the moderators and the questioners by predicting the quality of the question, its potential closure, and the possible reasons for closure. All these predictions are made solely based on textual information provided by the questioner, using natural language processing tools and deep learning models. Based on the research results, a proper tool can be created that examines the possible fate of the user's question by checking it in advance, thus preventing or at least reducing the chances of closure. Such a tool would also effectively support moderators, reducing the manual work they have to do.

3.1 Introduction

Stack Overflow (SO) is probably the most influential community question answering site for the software engineering and development society. With more than 11 million registered users and over 6,500 new questions posted on the site every day, SO has undoubtedly become an essential technical knowledge repository in programming. It is no exaggeration to say that both hobbyists and professional programmers and even leading software projects depend heavily on this site. Furthermore, SO is also a widely used basis for various information science research, and a vast number of studies have been published since the site's launch in 2008.

[1]https://stackoverflow.com/

SO's immense popularity clearly shows the ever-growing impact of social media in software engineering [110, 111], which, however, has its certain drawbacks. In particular, the competition between the quantity and quality of questions increases as the number of posts rises, leading to issues in maintaining the site's professionalism. Moderator work is undoubtedly an elaborate and laborious task, given the extreme posting frequency. At the same time, moderation is the key to avoiding the issue of quality decline, which is highly important for the sustainability of the service SO provides [112]. Dropping quality is a serious present-day issue that raised vivid discussions among the users,[2] *inter alia*, on the Meta site[3] of SO (MSO). Briefly, to maintain a reasonable level of quality, supporting poor questions should be avoided, and closing or deleting them is often inevitable.

Questions posted on the portal should be relevant, unambiguous, and comprehensible, meaning that every question has to be related either to specific development issues or methods such as using a particular API, algorithmic problems, tools, or methodology. Questions that cannot be answered suitably and straightforwardly, e.g., open-ended, off-topic, or chatty questions, which may reduce the quality of the portal, are not supported. The primary purpose of Stack Overflow is to solve well-defined technical issues; therefore, subjective posts or questions raising never-ending, opinion-related discussions should also be avoided. In addition, questions need to be formulated explicitly to show their purpose to the potential respondents, and friendly behavior towards other members is also expected. The last expectation is reflected in the reputation of the members, since whose personality is more open and supportive are more likely to get positive votes, as has been demonstrated by Bazelli et al. in their investigation [113]. As a result, positive attitudes play a prominent role in evaluating the questions or answers.

Currently, there are five reasons for closing a question:

- due to *duplication*;
- the question is *off-topic*;
- it is *unclear* what the user is asking;
- the question is *too broad* and cannot be answered straightforwardly;
- the question is primarily *opinion-based* leading to subjective discussions.

The closing procedure is a manual task relying on a voting system. Moderators or privileged users with 3,000 or more reputation points can cast a vote if they find a question inappropriate, and five such votes would end up closing the question. A previously closed question can be deleted from the site if it receives at least three

[2]https://meta.stackoverflow.com/questions/252506
[3]https://stackoverflow.com/help/whats-meta

votes from members within the 10,000 or higher reputation range. Reputation points and different badges reflect both the experience and the activity of the users on Stack Overflow. These achievements can be earned by asking good quality questions or providing good quality answers, both receiving high scores from the community. Higher reputation allows more elevated privileges such as the right to vote for closing or deleting a post.

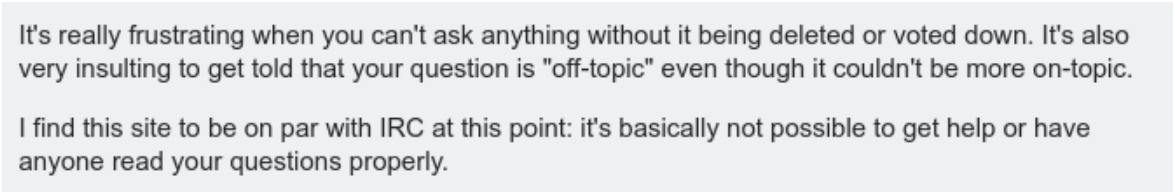


Figure 3.1: *Frustration for closing and deleting a question*[4]

Although the rules for posting a question on SO are fairly self-contained, and the portal itself provides support for the user compiling proper questions[5], the exact reason for closing a given post is not always clear to the user. Accordingly, *"Why was my question closed?"* type discussions appear quite frequently on the SO Meta site. The ambiguity in closing questions could easily lead to debates and frustrations that often manifest themselves in negative behavior in comments or answers. This, eventually, could easily result in hurting the user and make the SO community apparently hostile and unsupportive, especially for users with less experience[6]. This is well illustrated by a recently deleted post on MSO shown in Figure 3.1. This behavior, according to the comments, happens regularly.

Our research was motivated by the duality of the problem mentioned in the introduction. On the one hand, maintaining the quality of the portal currently requires a significant amount of manual work, which can be replaced by natural language processing tools and machine learning. On the other hand, it is also useful for users to be able to assess in advance the appropriateness of the questions they wish to submit to the portal, thus avoiding its closure.

Structure of the chapter: Section 3.2 summarizes the related works and briefly presents previous results. Section 3.3 presents the statistical description of the dataset along with the experiments executed on it. These experiments are designed to evaluate the quality of the questions posted to the site, their possible closure, and the reason for that closure. Section 3.4 presents the results and the discussion of the experiments, and the final thoughts are presented in Section 3.5.

[4]https://meta.stackoverflow.com/questions/388076
[5]https://stackoverflow.com/help/asking
[6]https://bit.ly/2oFsfKz

3.2 Related Works

Classifying the questions based on their quality has become an important research area in recent years. One of the pioneering studies from this area is the paper by Treude et al. [114], where the authors investigated whether a question received an accepted answer or remained unanswered. They introduced self-defined question categories and studied the number of posts receiving accepted answers in each category. It was found that some categories such as how-to-like instruction questions have been answered more frequently than other categories such as questions from novice programmers. Asaduzzaman et al. [115] investigated several factors that have remarkable effects on whether or not a particular question is answered. Their research found that the most crucial factor is that the question should be interesting for experienced users. Given the distribution of unanswered questions and the reputation points of the users, the researchers found that the proportion of unanswered questions was significantly lower among those with higher reputations.

Correa et al. [10] studied the effects of 19 features extracted from Stack Overflow posts using machine learning methods, including Support Vector Machine, Naive Bayes, Logistic Regression, and Stochastic Gradient Boosted Trees, and they have achieved an accuracy of 73% in identifying the prediction of closing. In another series of experiments, Correa and Sureka [116] achieved a 66% estimation accuracy with the help of the Decision Tree and Adaboost classifiers during the investigation of the deleted Stack Overflow posts. Ponzanelli et al. [11] studied various quality metrics of the questions submitted onto Stack Overflow. In their experiments, the Decision Tree classifier and genetic algorithms were used to predict the values of the quality metrics, and precision of 62.1% to 76.2% was achieved. Yao et al.[117] investigated the correlation of the quality of the questions and answers applying a co-prediction method based on both regression and a classification algorithm, which have been elaborated for non-linear optimization problems. They found a strong correlation regarding the quality between the questions and answers. Baltadzhieva and Chrupala [118] studied the linguistic features of the questions and investigated the extent to which these features influence the number of answers and the score a question receives. They found that the inclusion of linguistic information improves the prediction of the models' accuracy in their experiments.

Although the general prediction of the exact closing reasons has been less studied, one case, the duplicate question detection, is an exception. Several studies have been conducted to detect duplications [119, 120, 121, 122], and Stack Overflow now offers an automatic and real-time solution to list some similar posts while the user types the title of a new question. Predicting the possibility of closing a question due to the exact reasons other than duplication is a less popular area of research, and only a few papers can be found in the literature about this type of experiment.

Recently, Roy and Singh [123] studied the problem of predicting the exact closing reason of SO posts. They trained several machine learning- and deep learning-based classifiers to predict one of the five classes: *(i)* open; *(ii)* off-topic; *(iii)* not a real question; *(iv)* too constructive; and *(v)* too localized. This taxonomy corresponds to the former official closing policy of SO. They used features automatically extracted from the question body for machine learning models and pre-trained word embedding vectors for deep learning models. The authors performed many experiments with the various machine and deep learning approaches using both imbalanced and balanced datasets. The best result, an average precision of 47%, was obtained with a recurrent neural network-based model using the balanced dataset originating from the imbalanced one via oversampling the minority classes.

3.3 Experiments

This section presents the experiments focusing on the quality and the possible closing of questions posted to Stack Overflow. The research is simultaneously motivated by two different aspects. Firstly, the frequent question posting causes a significant workload for moderators to evaluate their quality and decide to retain or close the actual question. Second, assembling questions that meet the strict quality requirements is difficult, especially for beginners. Evaluating the question before posting it to the portal can help the users create their questions more easily. The prediction provided by the models presented in this research is based solely on the textual information; the prediction does not apply to any other information but the texts provided by the questioners during the assembling of those questions.

3.3.1 Description of the data

The dataset for this study was taken from the SO data dump[7] created on March 4, 2019. Both open and closed questions posted before June 2013 – the introduction of the currently available closing policy and reasons – were filtered. Questions closed due to duplication were also excluded from our dataset because of two reasons: *i*) this problem has already been well examined previously [119, 120, 122], and *ii*) finding a duplicate would imply the knowledge of the previously posted questions.

Preparing the dataset for quality based experiments

The first study focused on the classification of the quality of the questions. In defining high and low quality, we relied on the definition given in Ponzanelli et al. [11], which allowed us to compare our results with theirs. We consider those questions

[7] https://archive.org/details/stackexchange

high quality that have at least one accepted answer, their score is higher than zero, and they are neither closed nor deleted. Those questions that own a score less than zero and were closed or deleted in their final states were considered low-quality. As defined in the way mentioned, the distribution of the questions is presented in Figure 3.2. The distribution reflects the state queried on 21 April 2019 using Stack Exchange Data Explorer (SEDE)[8]. We have to highlight that neither the online database nor the dump contains permanently deleted questions. Those data can only be obtained for the moderators. From this fact, it can be concluded that the distribution is highly distorted; there are many more low-quality questions than available in the database. Nevertheless, the distribution presented here is essential because our models are trained on the available data. When evaluating performance, it should be taken into account that the accuracy achieved is likely to be lower than would be obtained using the complete dataset.

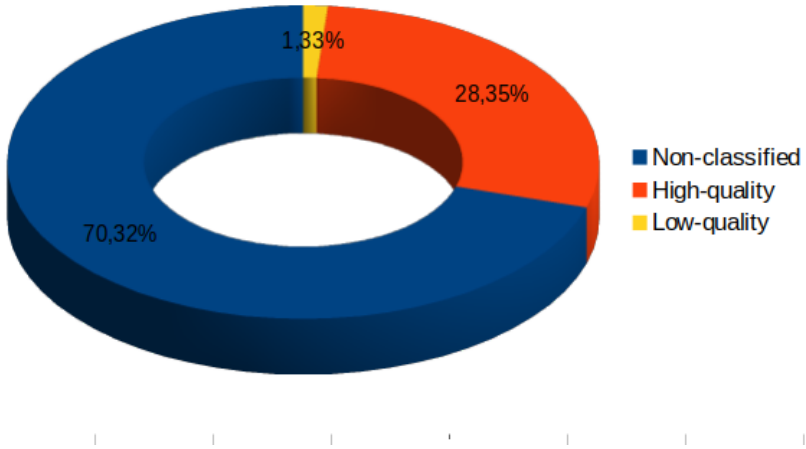


Figure 3.2: *Distribution of the questions based on quality*

Based on the quality definition and the distribution obtained from the online database, a significant number of questions cannot be classified mainly due to the vast amount of questions that have scored equal to zero (8,062,685), and even questions with positive scores that do not have any accepted answers (3,226,399).

The results of the quality-based experiments were intended to compare with the results of the model of Ponzanellis' who had used the dump of the Stack Overflow created in 2013 September. That dump is not available anymore; therefore, we have selected the corresponding period using the dump created on the 4th of March 2019 for our quality-based classifier. The test set was established from that period with two disjunct classes: high-quality and low-quality questions employing the definition mentioned earlier. Those questions that their originator edited after posting it to

[8]https://data.stackexchange.com/

Stack Overflow were excluded. We have also not considered those questions that hold zero scores. The team of Ponzanelli had also applied these filtering conditions, and we applied them for comparison purposes.

The test dataset was compiled using balanced subsets of high-quality and low-quality questions. The whole test set contains 110,547 posts from both subsets. We have chosen 30% of examples *(66,328 posts)* from this dataset as a final test set like the team of Ponzanelli had done.

We trained our quality-classification model using the whole question dataset, applying the same selection criteria for the two classes as we did with the test set. The items that occurred in the test set were excluded from the training set; hence for evaluation of the model, we used questions that have never been seen before by our model.

Our objective was to decide on the potential rejection regarding a question to be posted to Stack Overflow using only the linguistic features; therefore, the training and the test set only contain the question-body, the title, and the tags attached to the given post. The quality class and the `Id` of the posts were also included. Two separate experiments were executed using two different vectorization processes.

For the first experiments, the following input processing steps were performed. The selected sets were transformed using *Spacy* [9] document vector generation based on the `en_core_web_md` model which is a general linguistic model based on text from blogs, news, and comments. Before the vectorization, we had performed text cleaning procedures: code blocks, HTML tags, and non-textual characters were removed, and the tokenization process was also performed using *Spacy's* `nlp` method. The vectorized form using document vectors was also provided by *Spacy*.

Considering also the particular language used on technical forums, a 200 dimensional *Doc2Vec* model [124] on Stack Overflow questions using the *Gensim* [10] library was constructed for the second experiment. This model is based on the *Word2Vec* model [125] with the addition of the feature vector related to the whole document. Code blocks, preformatted text, and numbers were not removed but changed into the string *"example code snippets"*, *"number"*, and *"preformatted text"*, respectively. This method preserves the semantics of the document without considering the unnecessary details. Some specific characters such as #, + (see C#, C++), or glyph were also retained during the preprocessing.

Preparing the dataset for prediction of the likelihood of closing

The dataset was prepared differently to classify the likelihood of closing and determine the closing reasons. The dump used for the classification was the same as the

[9] https://spacy.io/

[10] https://radimrehurek.com/gensim/apiref.html

previous case, but the filtering and preparation processes differed. Both open and closed questions posted before June 2013 – the introduction of the currently available closing policy and reasons – were filtered. Questions closed due to duplication were also excluded. As already mentioned, neither the online database nor the dump contains the deleted posts, which are only accessible through moderation tools. The deleted data naturally have an impact on accuracy, as the application of deep learning requires a significant amount of data to achieve the possible accuracy that the model provides. The dataset applied for the classification is unbalanced containing 98,242 off-topic, 46,389 unclear, 46,026 too broad, and 16,383 opinion-based questions. The final dataset consisted of 207,040 closed questions, and – as usual in machine learning – we balanced our dataset by randomly sampling the same amount of open questions for the binary *(open vs. closed)* classification. Next to the likelihood of closing, the reason for the closing was also predicted in this research. For this purpose, the downsampling strategy was applied for balancing the dataset. The size of the classes is based on the smallest class, i.e., the opinion-based class with 16,383 questions. For the five-class classifier, 16,383 questions were sampled randomly for each label. This resulted in a dataset of 81,915 questions, out of which 2,730 were retained for testing, and the rest were used for training.

As mentioned earlier, our models rely only on textual information that is known by the user at the time of editing the question, particularly the text of the title, the body, and the tags. Several text-preprocessing procedures based on conventional NLP methods were performed applying the following steps:

- Transforming words to their lowercase format
- Removing non-ASCII and special characters, not in `[a-zA-z0-9+#]`
- Removing all new line and carriage return characters
- Removing stop words
- Replacing code blocks by a QCODE token
- Replacing numbers by a QNUMBER token
- Cleaning the text from HTML tags
- Performing tokenization and stemming using *Porter Stemmer*
- Performing part-of-speech tagging

The transformations applied were intended to reduce the size of the dictionary without losing essential information. POS tagging was used to preserve the grammatical information lost in the stemming procedure. Each individual token in the

body and the title text was associated with a POS tag to form a tuple. Therefore, the input for the vectorization process contained the `Id` of the given question, the transformed body, title, tags, and the actual class label.

During the vectorization, the elements of the tuples, together with items in the comma-separated list of question tags, were transformed into integers based on the dictionary index of the particular word and the corresponding POS tag. The size of the dictionary was determined based on the distribution of the prevalence of words in the input. Those words that are inside of the 99.5% frequency of occurrence were part of the dictionary. An additional QUNKN token was also added to the dictionary and substituted for words out of the dictionary.

After the transformation, the transformed tokens of the body, the title, and tags were concatenated. The input size of the resulting vector was also determined using the same procedure as determining the size of the dictionary. A longer input vector was cut to size, while a shorter one was padded with zeros. The resulting vector was exported to different *NumPy* arrays along with the `Id`s of the original posts and the corresponding labels. The resulting vectors were fed into the *Embedding* layer of *TensorFlow*[11] and *Keras*[12], which produced the actual embedding.

3.3.2 Quality-based experiments

The whole process of the quality-based classification experiments is presented in Figure 3.3. The preprocessing steps described in the previous subsection were executed both on the test and training set, beginning with the test set. The training set was selected so that questions contained by the test set do not come into the training set; therefore, those items have never been seen before by the model.

Our classifier is constructed using the *Keras* library for deep learning computations, with the *Tensorflow* backend. The model consists of one *Gated Rectifier Unit (GRU) [126]*, five *Dense* and one *Flatten* layers. The size of the GRU layer corresponded to the dimension of the input vector, which is 200 in this case. The next five *Dense* layers were constructed using the number of neurons corresponding to the input size, the half of the input size, the quarter, and the octet of the input size, respectively.

The GRU (see Figure 3.4) is a gated unit of recurrent neural networks designed to address the long-term dependencies. To do that, GRU applies a gating function using the previous hidden state and the bias in various combinations. The state values calculated by the unit are given in the following formulas *(in the formulas $*$ denotes the Hadamard product)*:

[11] https://www.tensorflow.org/
[12] https://keras.io

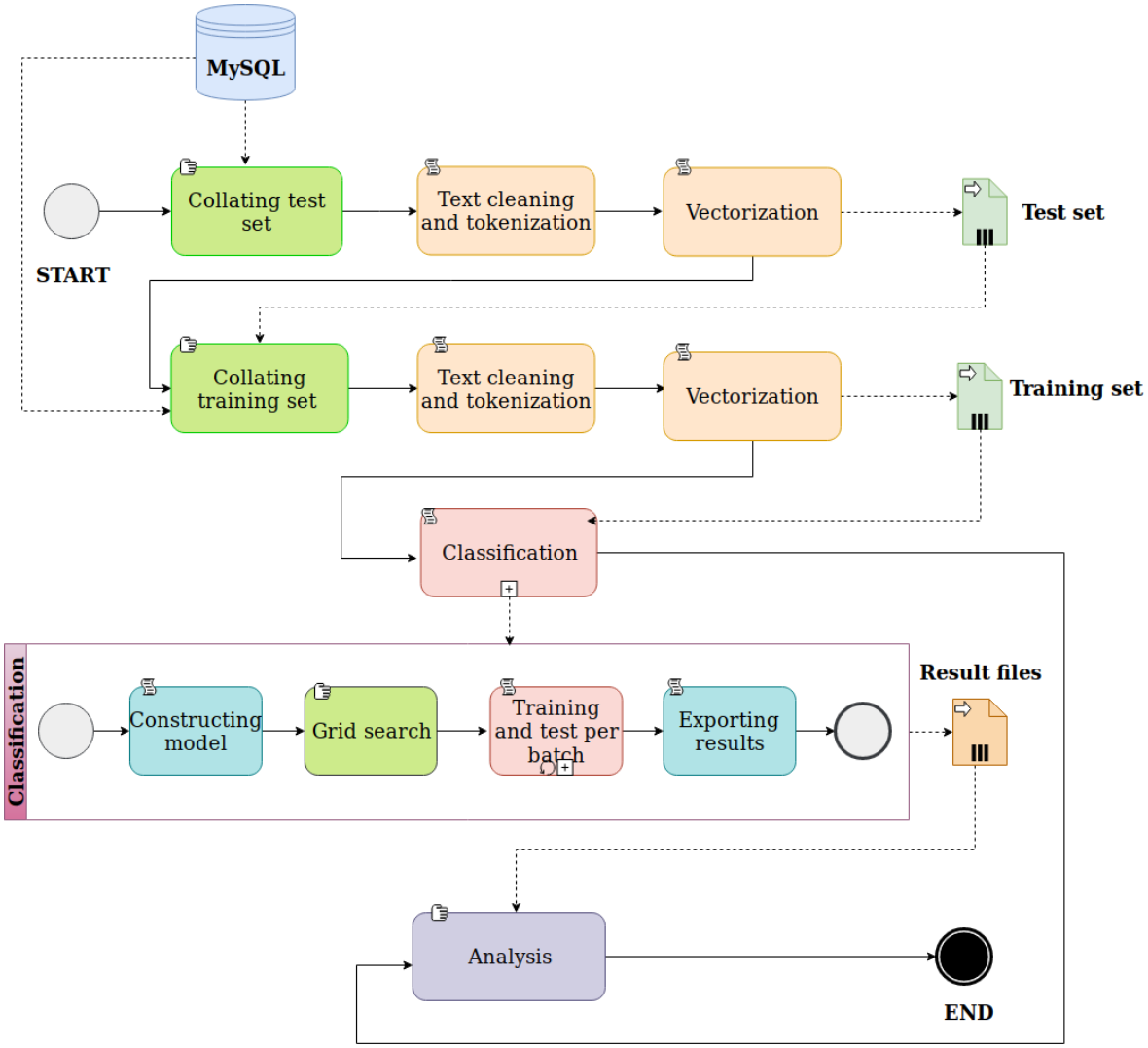


Figure 3.3: *Workflow of the experiments*

$$\mathbf{z_t} = \sigma(W_z[\mathbf{h_{t-1}}, \mathbf{x_t}]) \tag{3.1}$$

$$\mathbf{r_t} = \sigma(W_r[\mathbf{h_{t-1}}, \mathbf{x_t}]) \tag{3.2}$$

$$\mathbf{h'_t} = \tanh(W[\mathbf{r_t} * \mathbf{h_{t-1}}, \mathbf{x_t}]) \tag{3.3}$$

$$\mathbf{h_t} = (\mathbf{1} - \mathbf{z_t} * \mathbf{h_{t-1}}) + \mathbf{z_t} * \mathbf{h'_t} \tag{3.4}$$

We applied the *batch normalization* process [127] to decrease the variation of the distribution of inputs in the internal layers of the deep neural networks. This method has numerous benefits, such as stabilizing the network, accelerating learning, and reducing the dependence of gradients on the scale of the parameters. We utilized a

regularization technique called *dropout* to avoid overfitting. During the learning process, this regularization technique deactivates the neurons with a given probability provided by a hyperparameter called `dropout_rate`.

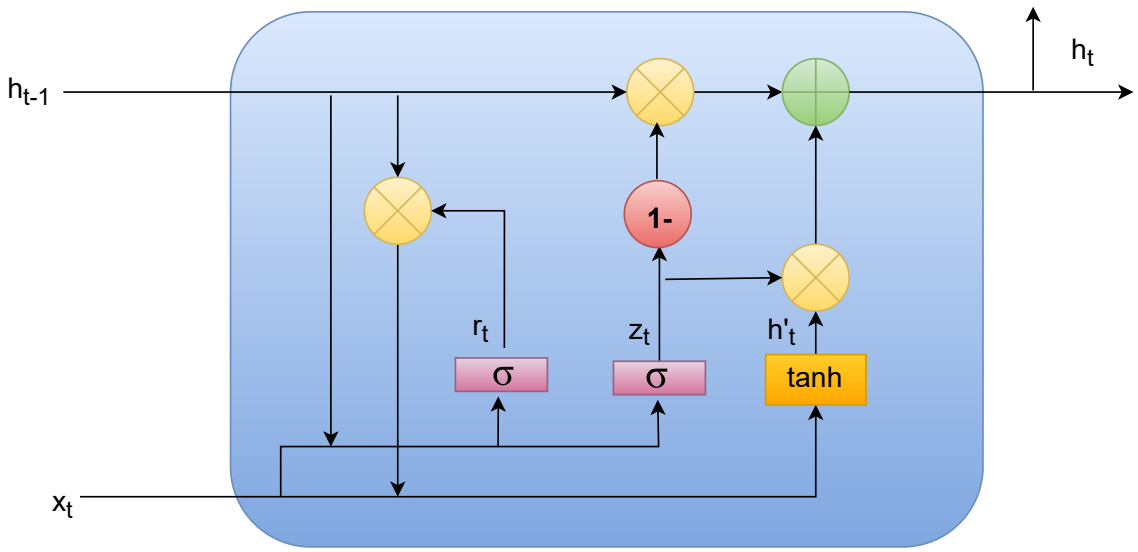


Figure 3.4: *Gated Recurrent Unit*

We chose the ReLU (Rectified Linear Unit) (Formula 2.16) activation function for the inner layers. This function helps avoid the vanishing gradient problem related to commonly applied activation functions, such as the sigmoid (Formula 2.13) and tangent hyperbolic (Formula: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$) in the inner layers of the deep neural networks[128]. For the last layer, which performs the classification, we chose the softmax function (Formula 2.17). As a loss function, the Binary Cross-Entropy (Formula 2.14) was selected.

For training the model, we applied Nesterov Stochastic Gradient Descent with momentum [129]. In this case, the value of the loss function is considered not in the actual position but in the near future, which can be adjusted using the momentum parameter. Besides, stochastic variation of the gradient descent performs the parameter update for each training example, which causes higher fluctuation of the objective function in the training process. Nevertheless, the higher fluctuation enables the objective function to jump to new, potentially better local minima. The parameter update using this algorithm obeys the following formula:

$$\mathbf{v_t} = \gamma \mathbf{v_{t-1}} + \eta \nabla_\theta J(\theta - \gamma \mathbf{v_{t-1}}) \tag{3.5}$$

$$\theta = \theta - \mathbf{v_t} \tag{3.6}$$

where θ denotes the parameter vector of the model, γ is the value of the momen-

tum, η the learning rate. The symbol $\mathbf{v_t}$ denotes the update vector *(velocity)* of the tth step.

After preparing the dataset, a grid-search procedure was performed using a random subset of the transformed input to find optimal hyperparameters for our model. Based on the grid search, we set the *learning rate* to 0.175 and the *momentum value* to 0.9. The *dropout rate* was set to 0.2, the *batch size* to 1000 and the *number of epochs* was set to 150. Based on these parameters, the experiments were executed, and the results were evaluated as presented in Section 3.4.

3.3.3 Experiments for closing prediction

The objective of the research is to provide the user with a practical tool capable of pre-evaluating the questions to be posted on SO and determining whether the question will be marked for closure by the community after submission. Figure 3.5 shows an overview of the experiments.

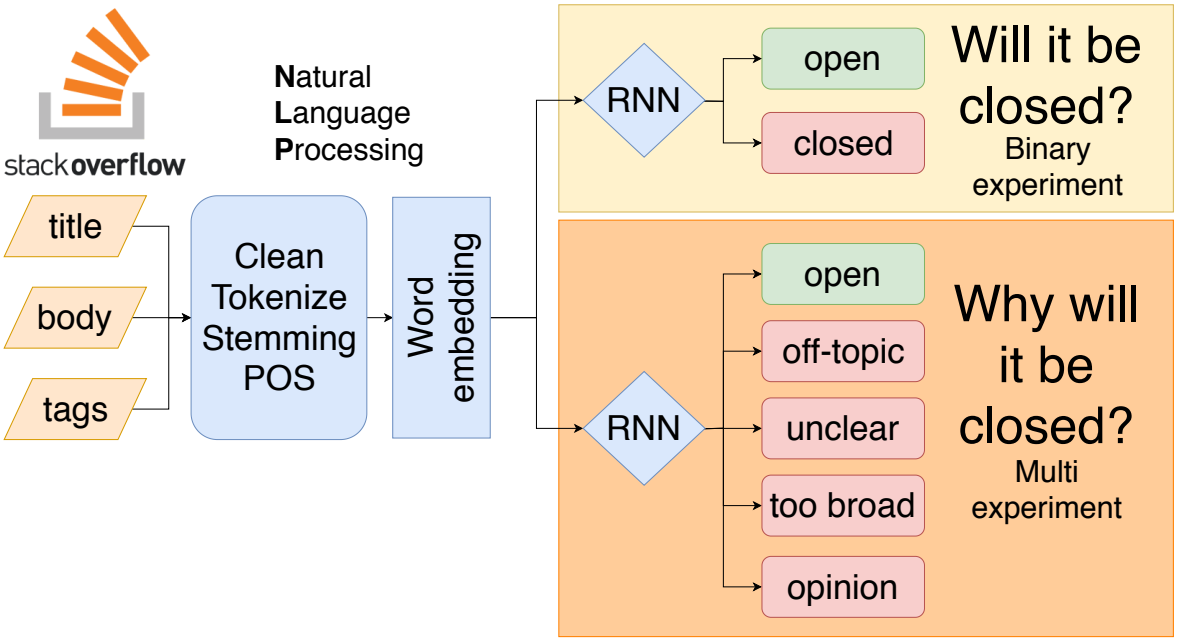


Figure 3.5: *The outline of the experiments*

After the preprocessing phase explained in Subsection 3.3.1, the embedding process was executed. In this experiment, we did not use an external library to perform the embedding, but the *Embedding* layer that is part of the neural network, was responsible for the actual vectorization given the dictionary size and dimension size, meaning that the actual embedding is learned parallel with the other model parameters. The output dimension was set to 50 for every *Embedding* layer.

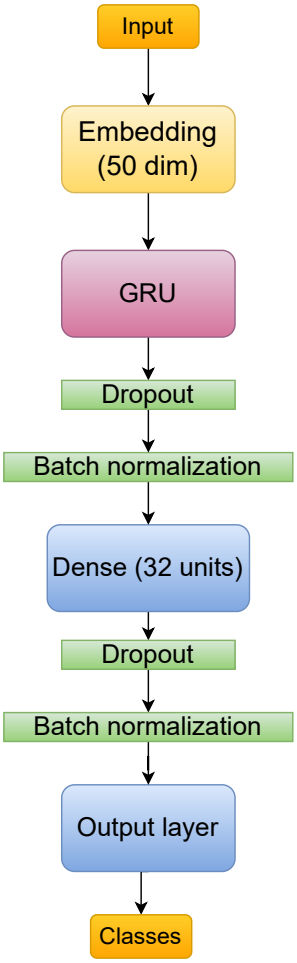


Figure 3.6: *The UNI classifier*

The binary classification's objective is to decide between the open versus closed questions, i.e., questions that meet the community rules versus questions that violate these rules. The second series is a five-class classification predicting the different closing reasons *(off-topic, unclear, too broad, opinion-based)* versus the open state. In both experiments, three different recurrent neural network (RNN) models denoted as UNI, BID, and COMP *(composite)* were defined and used. Each model is based on an embedding layer that creates a dense vector representation of the input followed by a GRU unit. The topology of the UNI (see Figure 3.6) and the BID model is almost the same; the only difference is the second layer which is unidirectional GRU in the case of the UNI, and bidirectional GRU in the case of the BID model. Both models apply one fully connected layer with 32 neurons and an output layer with the number of neurons corresponding to the classes under classification. The GRU contains the same number of units as the input dimension, in this case, fifty. In the case of the hidden layer, no activation function was applied. The output layer applied

sigmoid and softmax activation functions for binary and multiclass classification, respectively. The GRU applied hyperbolic tangent function as its activation function. The regularization dropout layer was applied after every layer, along with the batch normalization function.

Note: The bidirectional GRU consists of two GRU layers. One of them takes the input in the forward direction, and the other in the backward direction. The output of each position is a combination of the two values of the given position resulting from the forward and the backward computation.

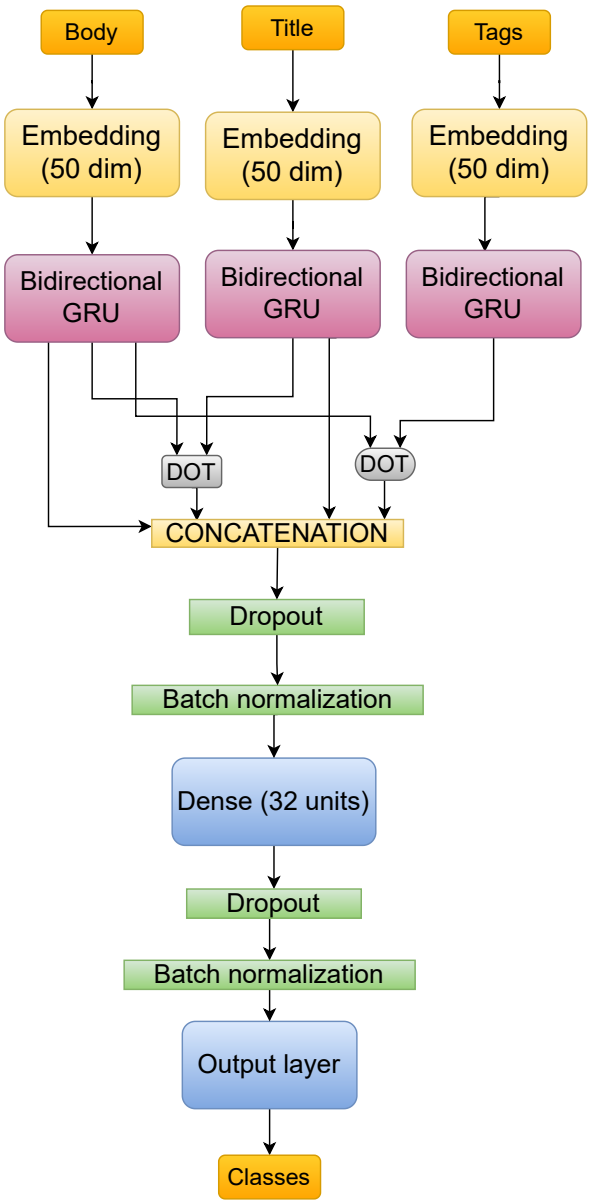


Figure 3.7: *The COMP classifier*

The COMP model (see Figure 3.7) applies three distinct *Embedding* layers along with a bidirectional GRU layer for every input *(body, title, tag)*. The transformed vector of the body has been multiplied together to form a dot product with the transformed title vector and the transformed tags vector, respectively. The body vector, the title vector, and the resulting dot products were concatenated, and the resulting vector was fed into a dense layer. The architecture contains the dense layer mentioned and also an output layer. The layers contained the same number of neurons as the previous cases. The regularization dropout layer was also applied after every dense layer and the concatenation, along with the batch normalization function. This model applies an activation function after every layer. The GRU layers use the hyperbolic tangent function, the dense layer applies ReLU, and the output layer applies sigmoid and softmax activation functions for binary and multiclass classification, respectively.

All networks apply the Nesterov Adam optimizer [130] together with binary and categorical cross-entropy (Formula 2.14) as loss functions for binary and multiclass classifications, respectively. The hyperparameters used in the training of every model are presented in Table 3.1. Micro and macro averages of the precision and recall (Formulas: 2.28,2.29, 2.30,2.31) pairs were applied; F1 measures were calculated a posteriori based on the averaged precision and recall values.

Table 3.1: *Parameter values of the models*

Model	Learning rate	Dropout rate	Batch size	Number of epochs	Beta1	Beta2	Decay
UNI BINARY	0.008	0.1	64	20	0.9	0.999	0.004
BID BINARY	0.008	0.1	64	20	0.9	0.999	0.004
COMP BINARY	0.009	0.5	64	10	0.9	0.999	0.004
UNI MULTI	0.008	0.1	64	20	0.9	0.999	0.004
BID MULTI	0.008	0.1	64	20	0.9	0.999	0.004
COMP MULTI	0.005	0.3	64	30	0.9	0.999	0.004

The update rules of the Nesterov Adam algorithm are given in the following formulas:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.7}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.8}$$

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{3.9}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{3.10}$$

$$\overline{m_t} = \beta_1 \hat{m_t} + (1 - \beta_1) g_t \tag{3.11}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\overline{m_t}}{\sqrt{\hat{v_t} + \epsilon}} \tag{3.12}$$

where g_t is the gradient calculated as

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta L(x^{(i)}, y^{(i)}, \tilde{\theta}]) \tag{3.13}$$

m is the first-order momentum, whereas v is the second-order momentum. The first and second-order momentum effects are regulated by β_1 and β_2, respectively. θ is the parameter vector, η is the learning rate.

The input data was split into train and test sets applying the *stratified k-fold cross-validation strategy* with k = 30 yielding 30 different train and test sets. We have selected k = 30 instead of the typical 10 to obtain a more reliable statistical analysis. In each training process and the corresponding evaluation, a distinct arbitrary random seed was chosen to ensure both the stochasticity allowing the statistical analysis of the results and the reproducibility of the experiments.

3.4 Results And Discussion

3.4.1 Results of the quality based experiments

As mentioned in the previous section, two series of experiments were performed. In the first series, a vectorized form of the input was applied using document vectors provided by *Spacy*. Only 425,097 random samples from the training set were used in the first experiment and obtained 65.5% precision, 65% recall, and 65% accuracy. For evaluation, the weighted average method provided by *scikit-learn* was applied. The size of the test set was 66,328.

The previous experiments were replicated and used 764,443 samples for training. The parameters and the input structure were the same as in the previous experiment. The size of the test set was also the same. We obtained 73.3% precision, 69.2% recall, and 69.3% accuracy in this case. After that, the experiment was replicated again, and the cosine similarity metrics as additional features were used to extend the input features. The result has not changed significantly; 73.1% precision, 67.8% recall, and 67.8% accuracy were obtained. This result can be explained as the dimension of the original vectors is 300, extending the features with three similarities, namely *body-title*, *body-tags*, and *title-tags*, did not significantly contribute to the result.

We applied the Stack Overflow specific *Doc2Vec* representation of the input for the second series of experiments. The test set was chosen from the period as we had done it before, and its size was 66,328. The size of the training set was expanded; we used 1,031,998 samples for training. The hyperparameters were also retained. Changing these parameters *(based on grid search)* did not produce better results,

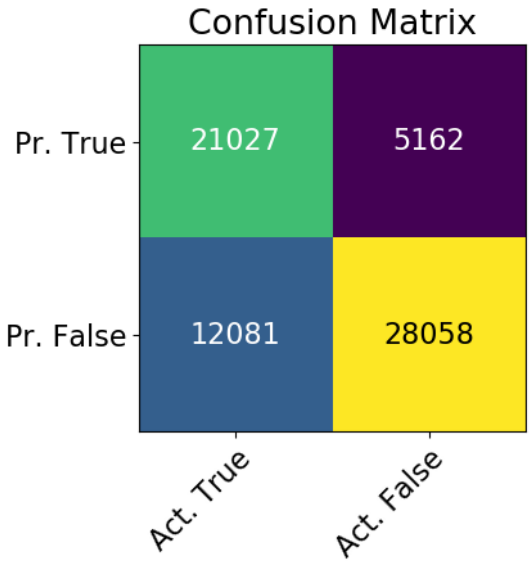


Figure 3.8: *Confusion matrix*

which is interesting. We assume this is due to the size of the training set, but this assumption is to be checked empirically. Using the hyperparameters described in Subsection 3.3.2, we got 75% precision, 74% recall, and 74% accuracy. Figure 3.8 presents the confusion matrix corresponding to this later experiment. The results of the experiments are summarised in Table 3.2.

Table 3.2: *Performance measures of the experiments*

Exp. number	Precision	Recall	Accuracy
Experiment 1.1	65.5%	65.0%	65.0%
Experiment 1.2	73.3%	69.2%	69.3%
Experiment 1.3	73.1%	67.8%	67.8%
Experiment 2.1	75.0%	74.0%	74.0%

Ponzanelli et al. [11] used various metrics for classification. They established three categories of metrics such as *common-metrics, readability-metrics*, and *popularity-metrics*. Although, the last category is related to the questioners, not the questions themselves. The authors utilized a decision tree and a genetic algorithm for training a quality function expressing a measure of quality using different combinations of the metrics. They obtained a precision between 61.2% and 66.3% using the decision tree algorithm combined with the common- and the readability-metrics. Using the quality function, they obtained 73.3% precision regarding the low-quality question based on common metrics. When authors used only the readability-metrics, the results were worse; however, by applying popularity-metrics, they obtained exquisite precision on the right tail of the dataset (up to 90.1%). The main

conclusion is that popularity metrics can be considered a better feature in deciding the quality of a given question.

Our result was obtained using the overall dataset, and therefore we obtained 75% precision, which is better than Ponzanellis'. However, this result is below their result if also considering the popularity metrics. Therefore, we can confirm that those features that correspond to user expertise are better forecasters considering the question quality. On the other hand, the 75% precision and 74% recall assure that natural language methods can be applied for pre-filtering the questions before posting them to Stack Overflow.

The threats to the experiments' validity are based mainly on the definitions chosen for measuring the quality. The scores do not always reflect the actual quality of the questions, as not every question is scored. The existence of accepted answers relates to the activity of the questioner; therefore, there might be questions with satisfactory answers, but the questioners did not use the acceptance possibility provided by the site.

Another issue is that the quality of the questions does not always indicate the decision to close. There might be cases where low-quality questions remain open, especially when the importance of the topic justifies it. Moreover, there might be high-quality questions, and they will be closed because of another issue like off-topic. Recommendation systems should apply a different approach to address these exceptions.

3.4.2 Results of the experiments predicting the closing

The fine-tuning of the hyperparameters for RNNs was accomplished to ensure the appropriate performance and avoid overfitting observed in some cases in the early training phases. As presented in Subsection 3.3.3, only a few parameters differ among the models.

Table 3.3: *Average performance measures (and variances) for the binary classification experiment*

Metric	**UNI**	**BID**	**COMP**
Micro precision	71.87 (0.06)%	71.00 (0.04)%	70.84 (0.08)%
Micro recall	71.87 (0.06)%	71.00 (0.04)%	70.84 (0.08)%
Micro F1	*71.87 (0.06)%*	*71.00 (0.04)%*	*70.84 (0.08)%*
Macro precision	73.78 (0.01)%	73.33 (0.01)%	73.66 (0.01)%
Macro recall	71.87 (0.06)%	71.00 (0.04)%	71.00 (0.08)%
Macro F1	*72.81 (0.02)%*	*72.14 (0.02)%*	*72.22 (0.02)%*
Accuracy	**71.87 (0.06)%**	**71.00 (0.04)%**	**70.84 (0.08)%**

The evaluation metrics of the binary classification experiment are shown in Ta-

ble 3.3. The Table presents the averages over the results obtained for the test sets in the *30-fold calculations*. The values in parentheses are the corresponding variances also given as percentages. As simply averaging F1 measures does not hold any information, these values in Table 3.3 were calculated from the average precision and recall numbers. Italics letters indicate this *a posteriori* nature.

Table 3.4: *Results of the K-S tests of the binary experiment*

Values of metrics	UNI	BID	COMP
Micro precision			
d-value	0.1684	0.1315	0.1336
p-value	0.3256	0.6302	0.6111
Distribution	*normal*	*normal*	*normal*
Micro recall			
d-value	0.1684	0.1315	0.1336
p-value	0.3256	0.6302	0.6111
Distribution	*normal*	*normal*	*normal*
Macro precision			
d-value	0.1507	0.0936	0.1987
p-value	0.4588	0.9334	0.1630
Distribution	*normal*	*normal*	*normal*
Macro recall			
d-value	0.1684	0.1315%	0.1336
p-value	0.3256	0.6302%	0.6111
Distribution	*normal*	*normal*	*normal*

The normality of the results for every model was checked using the *Kolmogorov-Smirnov (K-S) tes*t [131] with a significance level of 0.05. The difference statistics *(d-values)* along with the *p-values* and the decision of these K-S tests are presented in Table 3.4. The choice of Kolmogorov-Smirnov to perform the normality test is justified by the fact that this test can be used successfully with smaller sample sizes [132]. In this case, the samples were the results of the experimental runs, which number was 30 for each model.

As the normality is held for the samples, *the one-way ANOVA test* with a significance level of 0.05 was utilized to compare the results obtained with the models. According to the tests, the results of the binary classifiers follow a normal distribution and do not differ significantly from each other at the applied significance level of 0.05.

For comparison purposes, the last row of Table 3.3 also lists the accuracies of the models. As only the pre-submission textual information of posts was used as input, and the classification of the questions into closed versus open groups was performed directly, the number of related studies in the literature is scarce.

Our study presented in the previous subsection that focused on the quality classification also used textual information exclusively and obtained an accuracy of 74% for classifying SO posts into good versus weak categories using a deep neural network approach. The definition of the category labels as well as the filtering approach for input creation was adopted from the study of Ponzanelli *et al.* [11] and involved the use of post-submission information during the learning phase, including score and existence of an accepted answer. Ponzanelli *et al.* defined more than 40 different metrics for quality classification categorized in three distinct sets: *readability, user popularity*, and *common* textual metrics. Using two different classification algorithms, they provided a broad selection of precision values in the range of 60-90%. The highest precisions in the 80-90% range were obtained by including the user popularity metrics in their feature set. However, with the user-independent features, i.e., the readability and common metrics, they achieved a precision of 66.3% with the decision tree classifier. Precision values in the 60-70% range were also reported for these two metrics obtained with the genetic algorithm and quality functions for their most extensive datasets. Correa and Sureka [10], similar to our work, predicted question closing based on various predictive features including post-submission information and obtained an accuracy of 70.3%. The obsolete *(pre-June 2013)* closing policy was considered in their experiments, and questions from their collection, either closed or open, are not included in our dataset used in this study. In our experiments, in contrast to post-submission or user-related information, we considered only textual features irrespective of metrics such as the experience level of the user or question popularity.

Table 3.5: *Average performance measures (and variances) for the five-class classification experiment*

Metric	**UNI**	**BID**	**COMP**
Micro precision	48.55 (0.02)%	47.88 (0.04)%	47.38 (0.03)%
Micro recall	48.55 (0.02)%	47.88 (0.04)%	47.38 (0.03)%
Micro F1	*48.55(0.02)%*	*47.88 (0.04)%*	*47.38 (0.03)%*
Macro precision	50.42 (0.03)%	50.86 (0.02)%	49.04 (0.03)%
Macro recall	48.55 (0.02)%	47.88 (0.04)%	47.38 (0.03)%
Macro F1	*49.47 (0.02)%*	*49.32 (0.03)%*	*48.20 (0.03)%*
Accuracy	**48.55 (0.02)%**	**47.88 (0.04)%**	**47.39 (0.03)%**

The second series of experiments addressed the central question of why exactly a given post will be closed. To this end, a five-class classifier using the same pre-submission textual information as the binary model above is designed with the labels *off-topic*, *unclear*, *too-broad*, *opinion-based*, and *open*. The original dataset is heavily unbalanced, containing 98,242 off-topic, 46,389 unclear, 46,026 too broad, and

16,383 opinion-based questions. In order to balance this set for training, a downsampling strategy was applied, i.e., 16,383 questions were randomly sampled for each label resulting in a dataset of 81,915 questions out of which 2730 were retained for testing, and the rest were used for training in each loop of the 30-fold cross-validation setting.

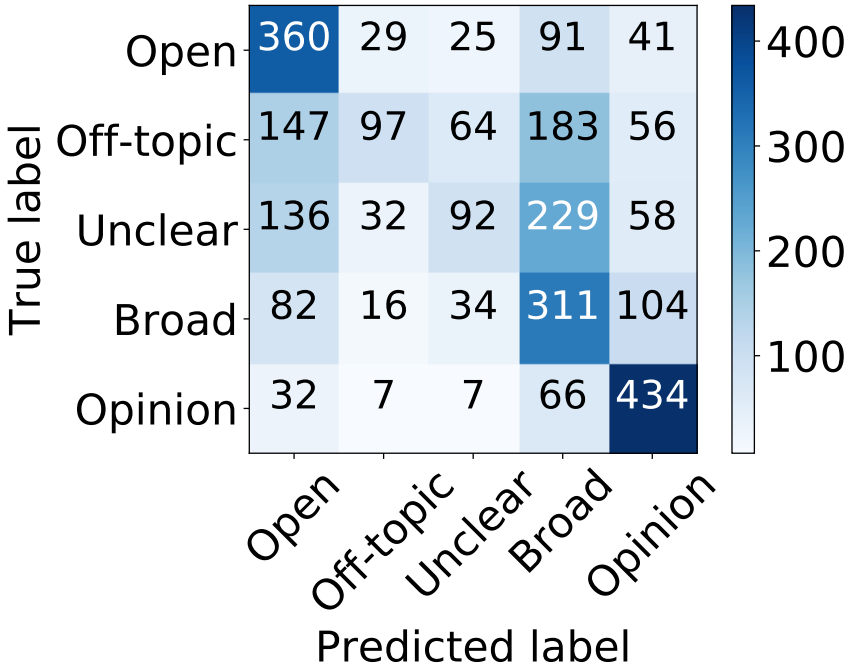


Figure 3.9: *Confusion matrix for the five-class classification*

The performance measures on the test set are presented in Table 3.5. Again, F1 measures written in italic are obtained from the average precision and recall values. The statistical analyses *(Kolmogorov-Smirnov test and the one-way ANOVA)* are the same as in the binary case: values resemble a normal distribution, and the models do not differ significantly.

As can be seen, the five-class classifier's performance is far superior to random guessing. There is, however, room for further improvement. One limiting factor to achieving higher accuracy is in distinguishing particular closing reasons. This issue is best represented by an example confusion matrix measured on one of the test sets from the 30-fold run and shown in Figure 3.9, which displays that the labels off-topic and unclear are often misclassified.

Although the SO Help Center gives a brief description of the closing reasons, distinguishing them in specific cases can be challenging for the machine and the human. This assumption is well supported by a study published on the SO Meta

site:[13] According to the study, there are many closings without a proper consensus amongst voters, and the efficacy of the voting system is questioned.

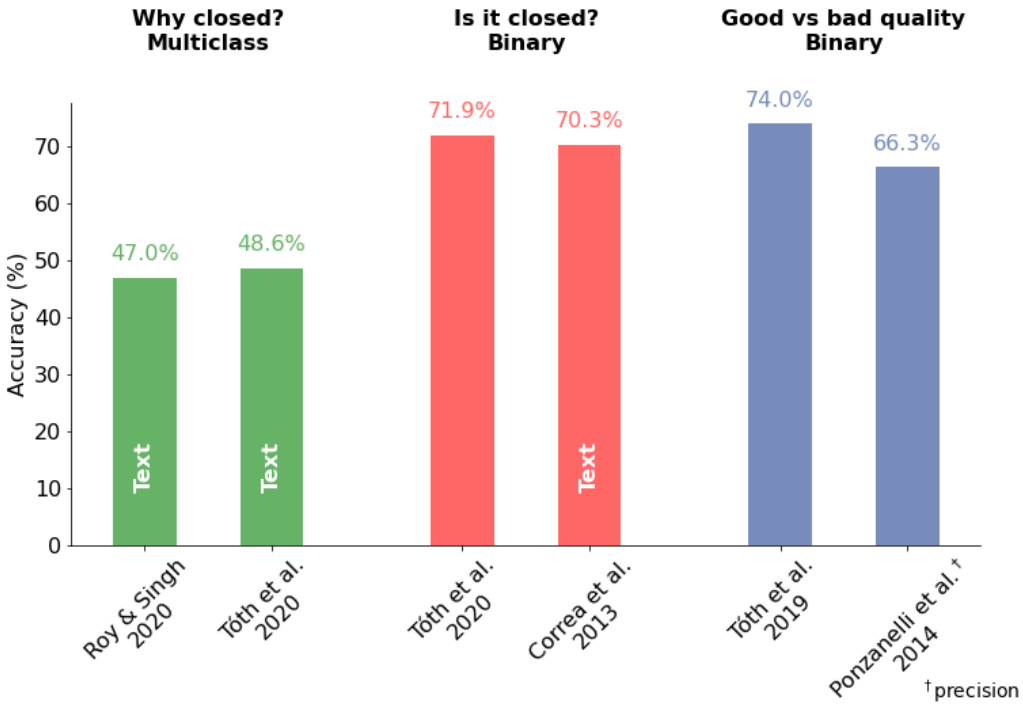


Figure 3.10: *Comparison of the results with selected literature values*

The other limiting factor to higher accuracy is the unbalanced dataset of the closing reasons. In contrast to the binary experiment, where all closed questions were considered together, the cardinality of the separate sets in the five-class case is significantly different: 98,242 for off-topic, 46,389 for unclear, 46,026 for too broad, and 16,383 for opinion-based questions corresponding to the original distribution. The smallest set contains 16,383 examples, while the largest closed set contains 98,242. In order to improve our model, more data and a better balancing strategy are necessary. One possible way to achieve this is by involving deleted questions in the training set. According to our hypothesis, these posts contain essential characteristics highlighting the features of closed questions. These, however, are not available in the public SO data dumps.

Interestingly, the best results were obtained in our binary and five-class classification experiments with the simplest RNN model, i.e., the unidirectional GRU classifier denoted UNI. We found only one study in the literature that directly attempts to predict closure reasons based on textual information. This research was conducted in parallel with our work, and its results became known to us after the publication

[13]https://meta.stackoverflow.com/questions/390083

of our paper. The authors have carried out several experiments using various machine learning, including deep learning procedures. They obtained the best results on balanced data by oversampling the minority class with their neural network-based models. Their result was an average precision of 47%. In our experiments, 48.6% precision was achieved by the UNI model, and both the BID and the COMP model produced slightly better results than Roys' models. Although a direct comparison of the results is not possible due to different datasets, a side-by-side comparison of the results, including the binary and multiclass results, gives an overview in Figure 3.10.

3.5 Conclusion and future works

Maintaining the quality of Q&A portals like Stack Overflow is a challenging task. While the popularity of these websites is increasing, the quality of their posts and the quality of the questions submitted to them is decreasing. Checking questions and removing those considered unsuitable will place a growing burden on moderators. Efforts to automatize this task have produced moderate results. Hence, there is still a need to develop procedures that can efficiently support this task.

Two aspects have been taken into account in this research. On the one hand, the reduction of the moderator burden mentioned above and on the other hand, the support of users in the preparation of their questions were both motivating factors. The research started by classifying the quality of the questions, where the concept of quality was interpreted in the same way as in the literature. The prediction was performed using a deep neural network, which despite the limited number of examples available, performed well, outperforming the results of known solutions in the literature.

Our research related to classifying the quality of the questions published to Stack Overflow was presented at the 14th International Conference on Software Technologies in 2019 [III].

The model relies only on textual information to classify quality but considers the number of scores and the number of accepted responses in the training process and the selection of posts, i.e., the definition of quality. In addition, closing a question and defining the quality of a question do not always coincide. For example, the case when an otherwise good quality question is asked, but it does not entirely fit the topic of the portal and is closed for off-topic reasons.

Considering the above, we have created a classifier that directly predicts closure. In this case, only textual information was used, known when the questions were compiled. The classifiers are based on recurrent deep neural networks and have been used to obtain results that can also be used in practice.

Moving on from binary classification, we attempted to predict the causes of closures by modifying the models. At the time of our research, no similar experiment

had been published, but a similar study was carried out in parallel with our work. Ultimately, our results proved to be better than the results obtained in the competing study. The multiclass classifier produced moderately good results, but this may be due to the much smaller training set and the uncertainty that otherwise exists for human classification. The results are nevertheless suitable for practical use.

Our research related to predicting the likelihood of the question closure along with the reasons for the closure was presented at the 42nd International Conference on Software Engineering at New Ideas and Emerging Results section in 2020 [IV]. This conference is the leading event in the field of software engineering.

We continued to study the closures, modified the models' structure, and included a new model, BERT *(Bidirectional Encoder Representation from Transformers)*, in the experiments. The dataset was also updated, using the dump data of 7 June 2021 for the new experiments. As expected, the performance of all models improved, with BERT achieving the best results, with an accuracy of 77.9% in the binary case and 57.67% in the multiclass case. We also validated the results on an independent set, which yielded 1% weaker results in both cases. The research results have not yet been published when writing this thesis and are in preparation for publication.

To put the research results into practice, we have created an online demonstration application that works similarly to the duplicate check on Stack Overflow, but in this case, it can be used to predict other reasons for closure. The input to the application is a user-specified question, which is vectorized online, and the vectorized input is evaluated by a trained model selected by the user. The application is planned to be made available to users in parallel with the publication of the new research results.

The author of this Ph.D. thesis group is responsible for the following contributions presented in this chapter:

II/1. The author has developed a GRU-based deep learning model to classify the questions based on their quality posted to Stack Overflow, considering only the textual elements. The definition of the quality was taken from the literature for comparison purposes.

II/2. The author executed the classification by applying different amounts of samples and vectorization processes.

II/3. The author compared the results with each other and the results of the other researchers. When all inputs were used in the classification process, the model's performance outperformed the performance of the classifiers used by others, demonstrating that deep learning solutions can provide better results if there is enough input available.

II/4. The author has developed three distinct GRU-based deep learning models to classify the likelihood of closing questions posted to the Stack Overflow, con-

sidering only the textual information available during the assembling of that question.

II/5. The author executed the classifications and compared the results with the other results available in the literature. The classifier provided good performances outperforming the other results, which can also be used in practice to evaluate the possibility of the question's closure before posting it to Stack Overflow.

II/6. The author modified the models to perform multi-class classification. In this case, the classifiers predict the possible closing reasons. To the best of our knowledge, this was the first classifier published to predict the closing reasons. However, a parallel study was in progress and published lately for the same purpose, but the classifiers made by the author have a slightly better performance. The classifiers provide a good result and are applicable in practice.

Bibliography

[1] Peter Naur and Brian Randell, editors. *Software engineering : report on a conference sponsored by the NATO Science Committee*, 1968. NATO Science Committee. Chairman: Professor Dr. F. L. Bauer.

[2] Winston W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 328–338. IEEE Computer Society Press, 1987. ISBN: 0897912160.

[3] Kevin Forsberg and Harold Mooz. The relationship of systems engineering to the project cycle. *Engineering Management Journal*, 4(3):36–43, 1992. doi: 10.1080/10429247.1992.11414684.

[4] Khaled El Emam and A. Günes Koru. A replicated survey of IT software project failures. 25(5):84–90, 2008. ISSN: 1937-4194, doi: 10.1109/MS.2008.107.

[5] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. URL `http://www.agilemanifesto.org/`.

[6] Sheetal Sharma, Darothi Sarkar, and Divya Gupta. Agile processes and methodologies: A conceptual study. 4, 2012.

[7] Colin Cherry. *On human communication: A review, a survey, and a criticism, 3rd ed*. On human communication: A review, a survey, and a criticism, 3rd ed. The MIT Press, 1978. ISBN: 9780262030656.

[8] Loretta L. Pecchioni, Hiroshi Ota, and Lisa Sparks. Cultural issues in communication and aging. In *Handbook of Communication and Aging Research*. Routledge, 2 edition, 2004. isbn: 9781410610171.

[9] Jarosław Kuchta. Framework reuse - heaven or hell, 2014. URL `https://mostwiedzy.pl/pl/publication/framework-reuse-heaven-or-hell,129822-1`.

[10] Denzil Correa and Ashish Sureka. Fit or unfit: Analysis and prediction of 'closed questions' on stack overflow. In *Proceedings of the First ACM Conference on Online Social Networks*, COSN '13, pages 201–212, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2084-9, doi: 10.1145/2512938.2512954.

[11] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, and Michele Lanza. Understanding and Classifying the Quality of Technical Forum Questions. In *14th International Conference on Quality Software*, pages 343–352. IEEE, oct 2014. ISBN: 978-1-4799-7198-5, doi: 10.1109/QSIC.2014.27.

[12] J. M. Carrillo de Gea, J. Nicolás, J. L. F. Alemán, A. Toval, C. Ebert, and A. Vizcaíno. Requirements Engineering Tools. *IEEE Software*, 28(4):86–91, 2011. doi 10.1109/MS.2011.81.

[13] M. Glinz. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26, 2007. doi 10.1109/RE.2007.45.

[14] Donald Firesmith. Common Requirements Problems, their Negative Consequences, and The Industry Best Practices to Help Solve Them. *Journal of Object Technology*, 6(1):17–33, 2007. doi 10.5381/jot.2007.6.1.c2.

[15] Azham Hussain, Emmanuel Mkpojiogu, and Fazillah Kamal. The Role of Requirements in The Success or Failure of Software Projects. *EJ Econjournals*, 6 (7S):6–7, 2016.

[16] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. What Works Better? A Study of Classifying Requirements. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pages 496–501, 2017. doi 10.1109/RE.2017.36.

[17] Taiseera Hazeem Al Balushi, Pedro R. Falcone Sampaio, Divyesh Dabhi, and Pericles Loucopoulos. ElicitO: A Quality Ontology-Guided NFR Elicitation Tool. In *Requirements Engineering: Foutndation for Software Quality*, pages 306–319. 2007. doi 10.1007/978-3-540-73031-6_23.

[18] V. Ambriola and V. Gervasi. Processing Natural Language Requirements. In *Proceedings 12th IEEE International Conference Automated Software Engineering*, pages 36–45, 1997. doi 10.1109/ASE.1997.632822.

[19] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Identification of Non-Functional Requirements in Textual Specifications: A Semi-supervised Learning Approach. *Information and Software Technology*, 52(4):436–445, 2010. doi 10.1016/J.INFSOF.2009.10.010.

[20] Yang Li, Emitza Guzman, Konstantina Tsiamoura, Florian Schneider, and Bernd Bruegge. Automated Requirements Extraction for Scientific Software. *Procedia Computer Science*, 51:582–591, 2015. doi 10.1016/J.PROCS.2015.05.326.

[21] Mengmeng Lu and Peng Liang. Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE'17*, pages 344–353, 2017. doi 10.1145/3084226.3084241.

[22] Abderahman Rashwan, Olga Ormandjieva, and Rene Witte. Ontology-Based Classification of Non-Functional Requirements in Software Specifications: A New Corpus and SVM-Based Classifier. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 381–386, 2013. doi 10.1109/COMPSAC.2013.64.

[23] Américo Sampaio, Neil Loughran, Awais Rashid, and Paul Rayson. Mining Aspects in Requirements. In *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, page 15, 2005.

[24] B Caglayan, E Kocaguneli, J Krall, Fayola Peters, and Burak Turhan. The PROMISE Repository of Empirical Software Engineering Data, 2012.

[25] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated Classification of Non-Functional Requirements. *Requirements Engineering*, 12(2):103–120, 2007. doi 10.1007/s00766-007-0045-1.

[26] Roxana Lisette Quintanilla Portugal and Julio Cesar Sampaio do Prado Leite. Extracting Requirements Patterns from Software Repositories. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 304–307, 2016. doi 10.1109/REW.2016.056.

[27] Roxana Lisette Quintanilla Portugal, Hugo Roque, and Julio Cesar Sampaio Do Prado Leite. A Corpus Builder: Retrieving Raw Data from Github for Knowledge Reuse in Requirements Elicitation. In *CEUR Workshop Proceedings*, pages 48–54, 2016.

[28] Eduard C. Groen, Jacqueline Schowalter, Sylwia Kopczynska, Svenja Polst, and Sadaf Alvani. Is There Really a Need for Using NLP to Elicit Requirements? A Benchmarking Study to Assess Scalability of Manual Analysis. In *CEUR Workshop Proceedings*, 2018.

[29] Steve Bundred. Solutions to Silos: Joining up Knowledge. *Public Money & Management*, 26(2):125–130, 2016. ISSN 0954-0962 doi 10.1111/j.1467-9302.2006.00511.x.

[30] H. Enquist and N. Makrygiannis. Understanding Misunderstandings [in Complex Information Systems Development]. In *Proceedings of the thirty-first Hawaii International Conference on System Sciences*, volume 6, pages 83–92, 1998. doi 10.1109/HICSS.1998.654762.

[31] Tshidi Mohapeloa. Effects of Silo Mentality on Corporate ITC's Business Model. *Proceedings of the International Conference on Business Excellence*, 11 (1):1009–1019, 2017. doi 10.1515/picbe-2017-0105.

[32] Peter Rolf Lutzeier. The Notion of Lexical Field and Its Application to English Nouns of Financial Income. *Lingua*, 56(1):1–42, 1982. ISSN 0024-3841 doi 10.1016/0024-3841(82)90048-1.

[33] Macarena Navarro-Pablo. The Lexical Field, a Key to Semantics. *Cauce: Revista Internacional de Filologia, Comunicacion y sus Didacticas*, (22):539–548, 1999. ISSN 2603-8560.

[34] John F. Sowa. *Semantic Networks*. 1987.

[35] Adam Pease and Ian Niles. IEEE standard upper ontology: a progress report. *The Knowledge Engineering Review*, 17(1):65–70, mar 2002. ISSN 0269-8889 doi 10.1017/S0269888902000395.

[36] Roberto Poli, Michael Healy, and Achilles Kameas, editors. *Theory and Applications of Ontology: Computer Applications*. Springer Netherlands, 2010. ISBN 978-90-481-8846-8 doi 10.1007/978-90-481-8847-5.

[37] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[38] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An Investigation Into The Notion of Non-Functional Requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, page 311. ACM Press, 2010. ISBN 9781605586397 doi 10.1145/1774088.1774153.

[39] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Functional Grouping of Natural Language Requirements for Assistance in Architectural Software Design. *Knowledge-Based Systems*, 30:78–86, jun 2012. ISBN 0950-7051 doi 10.1016/j.knosys.2011.12.009.

[40] John Slankas and Laurie Williams. Automated Extraction of Non-Functional Requirements in Available Documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 9–16, 2013. doi 10.1109/NAturaLiSE.2013.6611715.

[41] Tong Li and Zhishuai Chen. An Ontology-Based Learning Approach for Automatically Classifying Security Requirements. *Journal of Systems and Software*, 165, jul 2020. ISSN: 0164-1212 doi 10.1016/j.jss.2020.110566.

[42] H. Alrumaih, A. Mirza, and H. Alsalamah. Domain Ontology for Requirements Classification in Requirements Engineering Context. 8:89899–89908, 2020. ISSN: 2169-3536 doi 10.1109/ACCESS.2020.2993838.

[43] H. Kaiya and M. Saeki. Using Domain Ontology as Domain Knowledge for Requirements Elicitation. *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 189–198, 2006. ISBN 0-7695-2555-5 doi 10.1109/RE.2006.72.

[44] Vibhu Saujanya Sharma, Roshni R. Ramnani, and Shubhashis Sengupta. A Framework for Identifying and Analyzing Non-functional Requirements from Text. In *Proceedings of the 4th International Workshop on Twin Peaks of Requirements and Architecture*, pages 1–8, 2014. doi 10.1145/2593861.2593862.

[45] Pete Sawyer, Paul Rayson, and Roger Garside. REVERE: Support for Requirements Synthesis from Documents. *Information Systems Frontiers*, 4(3):343–353, 2002. doi 10.1023/A:1019918908208.

[46] C. Denger, D. M. Berry, and E. Kamsties. Higher Quality Requirements Specifications through Natural Language Patterns. In *Proceedings 2003 Symposium on Security and Privacy*, pages 80–90, 2003.

[47] Juyeon Kang and Jungyeul Park. Generating a Linguistic Model for Requirement Quality Analysis. In *Proceedings of the 30th Pacific Asia Conference on Language, Information and Computation (PACLIC 30)*, pages 439–447, 2016.

[48] Anas Mahmoud and Grant Williams. Detecting, Classifying, and Tracing Non-Functional Software Requirements. *Requirements Engineering*, 21(3):357–381, sep 2016. ISSN 1432010X doi 10.1007/s00766-016-0252-8.

[49] Anas Mahmoud. An Information Theoretic Approach for Extracting and Tracing Non-Functional Requirements. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 36–45. IEEE, aug 2015. ISBN 978-1-4673-6905-3 doi 10.1109/RE.2015.7320406.

[50] Xuchang Zou, Raffaella Settimi, and Jane Cleland-Huang. Improving Automated Requirements Trace Retrieval: A Study of Term-Based Enhancement Methods. *Empirical Software Engineering*, 15(2):119–146, 2010. doi 10.1007/s10664-009-9114-z.

[51] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. Automated Topic Naming. *Empirical Software Engineering*, 18(6):1125–1155, 2013. 10.1007/s10664-012-9209-9.

[52] Davide Falessi, Giovanni Cantone, and Gerardo Canfora. 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 - ESEM '10*, pages 1–10, 2010. doi 10.1145/1852786.1852810.

[53] Klerisson V.R. Paixao, Cricia Z. Felicio, Fernanda M. Delfim, and Marcelo De A. Maia. On The Interplay between Non-Functional Requirements And Builds on Continuous Integration. In *IEEE International Working Conference on Mining Software Repositories*, pages 479–482, 2017. doi 10.1109/MSR.2017.33.

[54] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang. Which Non-Functional Requirements Do Developers Focus On? An Empirical Study on Stack Overflow Using Topic Analysis. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 446–449, 2015. doi 10.1109/MSR.2015.60.

[55] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. What Are Developers Talking About? An Analysis of Topics and Trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, 2014. doi 10.1007/s10664-012-9231-y.

[56] Edna Dias Canedo and Bruno Cordeiro Mendes. Software requirements classification using machine learning algorithms. *Entropy*, 22(9), 2020. ISSN 1099-4300 doi 10.3390/e22091057.

[57] Armin Kobilica, Mohammed Ayub, and Jameleddine Hassine. Automated identification of security requirements: A machine learning approach. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20, page 475–480, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450377317 doi 10.1145/3383219.3383288.

[58] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F. Tichy. Norbert: Transfer learning for requirements classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 169–179, 2020. doi 10.1109/RE48521.2020.00028.

[59] Mark Steyvers and Joshua B. Tenenbaum. The Large-Scale Structure of Semantic Networks: Statistical Analyses And a Model of Semantic Growth. *Cognitive Science*, 29(1):41–78, 2005. ISSN: 0364-0213 doi 10.1207/s15516709cog2901_3.

[60] Julian Seitner, Christian Bizer, Kai Eckert, Stefano Faralli, Robert Meusel, Heiko Paulheim, and Simone Paolo Ponzetto. A Large DataBase of Hypernymy Relations Extracted from the Web. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation {LREC} 2016, Portorož, Slovenia, May 23-28, 2016*. European Language Resources Association {(ELRA)}, 2016.

[61] Timon C. Du, Feng Li, and Irwin King. Managing knowledge on the Web Extracting ontology from HTML Web. *Decision Support Systems*, 47(4):319–331, nov 2009. ISSN 0167-9236 doi 10.1016/J.DSS.2009.02.011.

[62] Beniamino di Martino, Antonio Esposito, Salvatore D'Angelo, Alessandro Marrazzo, and Angelo Capasso. Automatic Production of an Ontology with NLP: Comparison between a Prolog Based Approach and a Cloud Approach Based on Bluemix Watson Service. In *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, pages 537–542. IEEE, jul 2016. ISBN 978-1-5090-0987-9 doi 10.1109/CISIS.2016.98.

[63] Aurora Vizcaíno, Felix García, Mario Piattini, and Sarah Beecham. A validated ontology for global software development. *Computer Standards & Interfaces*, 46(C):66–78, may 2016. ISSN 09205489 doi 10.1016/j.csi.2016.02.004.

[64] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville. Development of a Software Engineering Ontology for Multisite Software Development. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1205–1217, aug 2009. ISSN 1041-4347 doi 10.1109/TKDE.2008.209.

[65] Y. Tian, D. Lo, and J. Lawall. Automated Construction of a Software-Specific Word Similarity Database. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 44–53, Feb 2014. doi 10.1109/CSMR-WCRE.2014.6747213.

[66] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker. Automatically Mining Software-Based, Semantically-Similar Words from Comment-Code Mappings. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 377–386, 2013. ISSN 2160-1852 doi 10.1109/MSR.2013.6624052.

[67] G. Sridhara, E. Hill, L. Pollock, and K. Vijay-Shanker. Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools. In *2008 16th IEEE International Conference on Program Comprehension*, pages 123–132, 2008. ISSN 1092-8138 doi 10.1109/ICPC.2008.18.

[68] J. Yang and L. Tan. SWordNet: Inferring Semantically Related Words from Software Context. In *Empirical Software Engineering*, number 19, pages 1856–1886, 2014. doi 10.1007/s10664-013-9264-x.

[69] Ashwin Ittoo, Gosse Bouma, Laura Maruster, and Hans Wortmann. Extracting Meronymy Relationships from Domain–Specific, Textual Corporate Databases. In *Natural Language Processing and Information Systems*, pages 48–59, 2010. ISBN: 978-3-642-13881-2.

[70] Tuğba Yıldız, S1avaş Yıldırım, and Banu Diri. A Study on Turkish Meronym Extraction Using a Variety of Lexico–Syntactic Patterns. In *Human Language Technology. Challenges for Computer Science and Linguistics*, Lecture Notes in Computer Science, pages 386–394. Springer International Publishing, 2016. ISBN: 978-3-319-43808-5 doi 10.1007/978-3-319-43808-5_29.

[71] Giuseppe Futia, Antonio Vetrò, and Juan Carlos De Martin. SeMi: A Semantic Modeling machIne to build Knowledge Graphs with graph neural networks. *SoftwareX*, 12, jul 2020. ISSN: 2352-7110 doi 10.1016/j.softx.2020.100516.

[72] Ole Magnus Holter. Semantic Parsing of Textual Requirements. In *The Semantic Web: ESWC 2020 Satellite Events*, pages 240–249. Springer International Publishing, 2020. ISBN: 978-3-030-62327-2 doi 10.1007/978-3-030-62327-2_39.

[73] Arshad Ahmad, Chong Feng, Muzammil Khan, Asif Khan, Ayaz Ullah, Shah Nazir, and Adnan Tahir. A Systematic Literature Review on Using Machine Learning Algorithms for Software Requirements Identification on Stack Overflow, jul 2020. ISSN: 1939-0114 doi 10.1155/2020/8830683.

[74] François Chollet. Keras, 2015. URL `https://github.com/fchollet/keras`.

[75] Le An, Ons Mlouki, Foutse Khomh, and Giuliano Antoniol. Stack Overflow: A Code Laundering Platform? In *SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, pages 283–293, 2017. doi 10.1109/SANER.2017.7884629.

[76] Jie Zou, Ling Xu, Mengning Yang, Xiaohong Zhang, and Dan Yang. Towards Comprehending the Non-functional Requirements Through Developers Eyes. *Information and Software Technology*, 84(C):19–32, 2017. doi 10.1016/j.infsof.2016.12.003.

[77] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[78] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[79] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386, 1958.

[80] Hinton G. Rumelhart, D. and R. Williams. Learning representations by back-propagating errors. *Nature*, (323):533–536, 1986.

[81] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations*, 2015.

[82] Janez Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets, 2006.

[83] Christine A. Bevc, Jessica H. Retrum, and Danielle M. Varda. New Perspectives on The "Silo Effect": Initial Comparisons of Network Structures Across Public Health Collaboratives. *American journal of public health*, 105, 2015. doi 10.2105/AJPH.2014.302256.

[84] Hossein Vatanpour, Atoosa Khorramnia, and Naghmeh Forutan. Silo Effect a Prominence Factor to Decrease Efficiency of Pharmaceutical Industry. *Iran J Pharm Res*, 12:207–216, 2013. ISSN 1735-0328.

[85] Brent Gleeson. The Silo Mentality: How To Break Down The Barriers, 2013. URL `https://www.forbes.com/sites/brentgleeson/2013/10/02/the-silo-mentality-how-to-break-down-the-barriers/`.

[86] Daniel E. The Silo Effect | Case Study On The Silo Effect The Strategic CFO, may 2018. URL `https://strategiccfo.com/silo-effect/`.

[87] Xiaohua Wang, Zhi Wu, and Ming Zhao. The Relationship between Developers and Customers in Agile Methodology. In *2008 International Conference on Computer Science and Information Technology*, pages 566–572, 2008. doi 10.1109/ICCSIT.2008.9.

[88] Alan Baddeley, Michael W. Eysenck, and Michael C. Anderson. *Memory*. Memory. Psychology Press, New York, NY, US, 2009. ISBN 978-1-84872-001-5 978-1-84872-000-8.

[89] Marie Poirier, Jean Saint-Aubin, Ali Mair, Gerry Tehan, and Anne Tolan. Order Recall in Verbal Short-Term Memory: The Role of Semantic Networks. *Mem Cognit*, 43(3):489–499, apr 2015. ISSN 1532-5946 doi 10.3758/s13421-014-0470-6.

[90] Csaba Pléh and Ágnes Lukács. *Pszicholingvisztika*. Akadémiai kiadó Zrt., 2009. ISBN 9789630594998.

[91] Gottlob Frege, P. T. Geach, and Max Black. On Concept And Object. *Mind*, 60 (238):168–180, 1951. ISSN: 0026-4423.

[92] John R. Anderson and Robert Milson. Human Memory: An Adaptive Perspective. *Psychological Review*, 96(4):703–719, 1989. ISSN 1939-1471(Electronic),0033-295X(Print) doi 10.1037/0033-295X.96.4.703.

[93] Prahlad Gupta. A Computational Model of Nonword Repetition, Immediate Serial Recall, and Nonword Learning. In *Interactions between short-term and long-term memory in the verbal domain*, pages 108–135. Psychology Press, New York, NY, US, 2009. ISBN 978-1-84169-639-3.

[94] IA Richards and CK Ogden. *The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism*. Harvest/HBJ, 1989. ISBN 0-15-658446-8.

[95] Marti A Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In {*COLING*} *1992 Volume 2: The 15th International Conference on Computational Linguistics*, 1992. url: https://www.aclweb.org/anthology/C92-2082.

[96] Noam Chomsky and David W. Lightfoot. *Syntactic Structures*. Berlin, Boston, 2009. ISBN: 978-3-11-021832-9 doi 10.1515/9783110218329.

[97] David Crystal. *A Dictionary of Linguistics and Phonetics, Sixth Edition*. 2008. ISBN: 9781444302776 doi 10.1002/9781444302776.

[98] Princeton University. About WordNet, 2010. URL `https://wordnet.princeton.edu/`.

[99] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The Penn Treebank: An Overview. In *Treebanks: Building and Using Parsed Corpora*, Text, Speech and Language Technology, pages 5–22. Dordrecht, 2003. ISBN: 978-94-010-0201-1 doi 10.1007/978-94-010-0201-1_1.

[100] Merriam-Webster. Merriam-Webster dictionary: bellow, 2021. URL `https://www.merriam-webster.com/dictionary/below`.

[101] D. Mavridis, I. Moustaki, and M. Knott. 7 - Goodness-of-Fit Measures for Latent Variable Models for Binary Data. In Sik-Yum Lee, editor, *Handbook of Latent Variable and Related Models*, Handbook of Computing and Statistics with Applications, pages 135 – 161. North-Holland, Amsterdam, 2007. ISSN 18710301 doi 10.1016/B978-044452044-9/50010-0.

[102] Marvin Karson. Handbook of Methods of Applied Statistics. Volume I: Techniques of Computation Descriptive Methods, and Statistical Inference. Volume II: Planning of Surveys and Experiments. I. M. Chakravarti, R. G. Laha, and J. Roy, New York, John Wiley; 1967, $9.00. *Journal of the American Statistical Association*, 63(323):1047–1049, 1968. doi 10.1080/01621459.1968.11009335.

[103] Marie Laure Delignette-Muller and Christophe Dutang. fitdistrplus: An R Package for Fitting Distributions. *Journal of Statistical Software*, 64(4):1–34, 2015. URL `https://www.jstatsoft.org/v64/i04/`.

[104] R. A. Rigby and D. M. Stasinopoulos. Generalized additive models for location, scale and shape,(with discussion). *Applied Statistics*, 54:507–554, 2005.

[105] Colin S. Gillespie. Fitting Heavy Tailed Distributions: The poweRlaw Package. *Journal of Statistical Software*, 64(2):1–16, 2015.

[106] Robert A. Rigby and D. Mikis Stasinopoulos. Smooth Centile Curves for Skew and Kurtotic Data Modelled Using The Box-Cox Power Exponential Distribution. *Statistics in Medicine*, 23(19):3053–3076, 2004. ISSN: 0277-6715 doi 10.1002/sim.1861.

[107] Jeff Atwood. What does Stack Overflow want to be when it grows up?, oct 2018. URL `https://blog.codinghorror.com/`.

[108] Aleksi Aaltonen and Sunil Wattal. Rejecting and Retaining New Contributors in Open Knowledge Collaboration: A Natural Experiment in Stack Overflow Q&A Service. In *ECIS 2020 Research Papers*, volume 183, page 16, 2020.

[109] Jay Hanlon. Stack Overflow isn't very welcoming. It's time for that to change, 2018. URL `https://medium.com/@jayhanlon/welcome-wagon-dd57cbdd54d9`.

[110] T. Lopez, T. Tun, A. Bandara, M. Levine, B. Nuseibeh, and H. Sharp. An Anatomy of Security Conversations in Stack Overflow. In *41st International Conference on Software Engineering: Software Engineering in Society*, pages 31–40, 2019. doi 11.1109/ICSE-SEIS.2019.00012.

[111] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (R)Evolution of Social Media in Software Engineering. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 100–116, 2014. ISBN 978-1-4503-2865-4 doi 10.1145/2593882.2593887.

[112] I. Srba and M. Bielikova. Why is Stack Overflow Failing? Preserving Sustainability in Community Question Answering. *IEEE Software*. doi 10.1109/MS.2016.34.

[113] Blerina Bazelli, Abram Hindle, and Eleni Stroulia. On the Personality Traits of StackOverflow Users. In *2013 IEEE International Conference on Software Maintenance*, pages 460–463. IEEE, sep 2013. ISBN 978-0-7695-4981-1 doi 10.1109/ICSM.2013.72.

[114] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How Do Programmers Ask And Answer Questions on The Web? In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 804, New York, New York, USA, 2011. ACM Press. ISBN 9781450304450 doi 10.1145/1985793.1985907.

[115] Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K Roy, and Kevin A Schneider. Answering Questions About Unanswered Questions of Stack Overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 97–100, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1 doi 10.1109/MSR.2013.6624015.

[116] Denzil Correa and Ashish Sureka. Chaff from the Wheat: Characterization and Modeling of Deleted Questions on Stack Overflow. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 631–642, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2744-2 doi 10.1145/2566486.2568036.

[117] Tao Xie Leman Akoglu Feng Xu Yuan Yao, Hanghang Tong and Jian Lu. Want a Good Answer? Ask a Good Question First! *CoRR*, abs/1311.6876, 2013.

[118] Antoaneta Baltadzhieva and Grzegorz Chrupała. Predicting the Quality of Questions on Stackoverflow. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 32–40, 2015.

[119] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Mining Dduplicate Questions in Stack Overflow. In *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*, pages 402–412. ACM Press, 2016. ISBN 9781450341868 doi 10.1145/2901739.2901770.

[120] Rodrigo F. G. Silva, Klerisson Paixao, and Marcelo de Almeida Maia. Duplicate Question Detection in Stack Overflow: A Reproducibility Study. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 572–581. IEEE, mar 2018. ISBN 978-1-5386-4969-5 doi 10.1109/SANER.2018.8330262.

[121] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. Multi-Factor Duplicate Question Detection in Stack Overflow. *Journal of Computer Science and Technology*, 30(5):981–997, sep 2015. ISSN 1000-9000 doi 10.1007/s11390-015-1576-4.

[122] Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, and Ermyas Abebe. Detecting Duplicate Posts in Programming QA Communities via Latent Semantics and Association Rules. In *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, pages 1221–1229. ACM Press, apr 2017. ISBN 9781450349130 doi 10.1145/3038912.3052701.

[123] Pradeep Kumar Roy and Jyoti Prakash Singh. Predicting Closed Questions on Community Question Answering Sites Using Convolutional Neural Network. *Neural Computing and Applications*, 32(14):10555–10572, Jul 2020. ISSN 1433-3058 doi 10.1007/s00521-019-04592-0.

[124] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. *CoRR*, abs/1405.4053, 2014.

[125] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[126] Kazuki Irie, Zoltán Tüske, Tamer Alkhouli, Ralf Schlüter, and Hermann Ney. LSTM, GRU, highway and a bit of attention: An empirical overview for language modeling in speech recognition. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, volume 08-12-Sept, 2016. ISBN 9781510810587 doi 10.21437/Interspeech.2016-491.

[127] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[128] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[129] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On The Importance of Initialization And Momentum in Deep Learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[130] Amitabh Basu, Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence Guarantees for RMSProp And ADAM in Non-Convex Optimization And Their Comparison to Nesterov Acceleration on Autoencoders. *CoRR*, abs/1807.06766, 2018.

[131] Frank J. and Massey Jr. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951. doi 10.1080/01621459.1951.10500769.

[132] Jürgen Janssen and Wilfried Laatz. *Statistische Datenanalyse mit SPSS für Windows*. Springer, Berlin, Heidelberg, 2007. ISBN 978-3-540-72977-8 doi 10.1007/978-3-540-72978-5.

Summary

The Ph.D. dissertation presents research aimed at developing solutions that support software developers in performing their specific tasks based on natural language processing and artificial intelligence-based methods.

Software development is a complex activity that requires, in addition to professional knowledge and technical skills, an analytical approach to the phenomena of the world around us. For software to perform its task at the appropriate level requires a thorough understanding and analysis of the phenomenon it is modeling, based on the functional and non-functional requirements formulated by the experts and stakeholders of the problem and detailed in the business analysis. The requirements are predominantly available in natural language and are formulated using the particular language of the business area for which the software is being developed. In addition, a significant part of the requirements is not only conveyed in the form of verified business documents and policies but also in the form of ideas and concepts expressed verbally during interviews. The latter are often ambiguous, contradictory, or incomplete. In addition, the use of business terminology also implies tacit knowledge that is self-evident to the stakeholders but often unknown to the development team.

The use of natural language processing methods and artificial intelligence tools can help the development team automate processing requirements given in natural language forms. The present research has focused on classifying requirements and developing tools for reconciling and managing semantic obstacles arising from different language usage by the stakeholders and the developers. Based on the results of the research presented in this thesis, it can be stated that natural language processing methods, as well as artificial intelligence tools, machine learning, and semantic networks, are suitable to achieve the declared objectives of supporting the requirements engineering tasks of software developers, which can be extended to other cases of interactions between the development and business domains, such as the end-user support processes.

Software development has undergone a significant transformation in recent decades. Nowadays, no one is expected to have universal knowledge of the vast array of frameworks, programming languages, and tools. At the same time, development is increasingly relying on reusable elements that have been previously built and made

available to the development communities. Because of the high specialization, it can be stated that it is increasingly important for developers to share their knowledge efficiently and to ask specialists from different professional areas for help in solving problems they encounter in their daily work. Addressing the problem that arose from the specialization, a number of Q&A sites have been created on the Internet to share the experiences of the developers and help them solve problems they encounter in their work. The best known such sites is Stack Overflow, which is dedicated to supporting developers' work by creating a repository of knowledge of their solved problems.

Maintaining the quality of a Q&A site while its traffic is growing is a significant challenge for moderators, but at the same time, formulating questions that meet the expectations of the community of that particular site is not always straightforward. Our research focused on predicting the quality of the questions asked on Stack Overflow using natural language processing tools and deep learning, respectively, and the prediction of the likelihood for their subsequent closure. The research also investigated the possible reasons for the question closures. The results have led to models that can be used in practice to check the compliance of questions under assembling by the user with the requirements of the Stack Overflow, thus reducing both the moderator workload and the likelihood of the subsequent closures thus indirectly preserving the professional quality of the portal.

The dissertation consists of two major parts. Chapter 1 introduces the research about the application of natural language processing and artificial intelligence methods in requirements engineering. Chapter 2 presents the research investigating developer interactions using natural language processing and deep learning methods. The target communication platform is Stack Overflow, and the research focused on predicting the quality and the likelihood of the closure of the questions that developers ask on the site.

Thesis Group 1: Applying NLP And Artificial Intelligence Methods in Requirements Engineering

Requirements elicitation is an essential part of software development; a successful software project would not exist without it. Requirements are available in various legal, technical, and business documents, and a significant part of them are collected during interviews with the stakeholder conducted by the business analyst. The requirements are articulated using a natural language. In addition, those requirements collected from the interviews are often ambiguous, incomplete, and contradictory, thanks to the different expectations of the stakeholders. The non-functional requirements, which are also an essential part of the collection of the requirements, are

often not expressed explicitly, but the software cannot fulfill its purpose without considering them. Besides, the collected requirements are expressed in the particular business language, and the terms used in that business domain usually differ semantically from those that the developers use in their everyday communication. In order to resolve the different semantic interpretations, in many cases, it is also necessary to have a general tacit knowledge of the business domain, which is not always available in requirements analysis.

In Chapter 2, the research aimed to support the process of requirements engineering is discussed. The research consists of two parts. In the first part, the classification of the non-functional requirements from the requirements documents is discussed. During the research, two separate experiments were conducted. First, a relatively small labeled dataset, the Tera Promise NFR dataset, was applied. This dataset contains 625 requirements collected from 15 different projects and classified into 12 classes, from which one class represents the functional requirements. During the experiments, various machine learning algorithms were compared on the vectorized form of the input. For the vectorization, the *tf-idf* format was applied. The number of records collected in the Promise NFR is rather scarce for machine learning tasks, although the result of the experiments produced quite good precision values. The best result was achieved by the Logistic Regression (precision: 87%, recall: 67%), the Multinomial Naive Bayes (precision: 84%, recall: 68%), and the SVM classifier (precision: 89%, recall 65%). The experiment was replicated using specifically chosen posts from Stack Overflow. In this experiment, 50,000 examples were collected related to the *usability* and the *testability* requirements. Using this dataset, every model produced better results. The best results have been achieved using Fully Connected Network (precision: 96%, recall: 94%), SVM (precision: 95%, recall: 94%) and Logistic Regression classifiers (precision: 95%, recall: 95%). The results of the experiments strongly suggest that the use of natural language processing methods and supervised learning algorithms can be an effective and practical support for the requirements analysis processes.

In another line of research, we looked for algorithmic solutions to overcome communication difficulties to support communication between the business domain and the software development community. A significant proportion of communication difficulties are caused by, among other things, semantic differences between the terms used, which is referred to in organizational psychology as the communication silo phenomenon. Communication silos are also a significant obstacle in software development, where software is used to model a particular business phenomenon, and a thorough understanding of that particular phenomenon is an essential criterion for building its model. The difference in meaning between the terms needs to be explored and resolved to enable communication; thus, requirements analysis can happen at the right level.

In the research, the factors determining the meaning of the terms used in communication and the relationship between terms were explored. The semantic network used for modeling and the concept of a semantic space that determines unambiguous meaning were also defined. Stack Overflow posts were used to model the semantic environment as confined by the software development community. The most common relationships between noun phrases in the discourses of Stack Overflow were mined, the hyperonym-hyponym relationships. For the detection of these relationships, lexico-syntactic patterns were utilized. A phrase-structured grammar and a deterministic automaton were constructed and applied to recognize the essential parts of these relationships, the noun phrases. Based on the hyperonym-hyponym relationship, a semantic network modeling the semantic space of the software developer was constructed, and the specific conceptual meanings defined by this network were compared with the meaning defined by WordNet, which is built based on the general semantic environment. From the results of these studies, it can be concluded that semantic networks are suitable tools for capturing specific meanings, thus allowing the recognition of tacit knowledge and thus supporting the relevant software development processes (requirements management and end-user support).

Thesis Group 2: Investigating Developers' Interactions Using NLP and Deep Learning

The proliferation of programming languages, frameworks, and specific tools used in software development also requires a high degree of specialization of software developers. This specialization is not limited to the target domain but also significantly impacts the technology used in software development. However, in the course of their practical tasks, developers inevitably encounter technologies that they are not familiar with or are less familiar with, and which, due to the accelerated development cycle, they are less able to study in-depth, thus relying more on the support of the software development community as a whole, where the necessary knowledge is available.

In response to these needs, several Q&A platforms have been created on the Internet, where developers can share their experiences and knowledge and ask for help in solving problems they encounter in their everyday work. These platforms have been set up primarily to support professional work, with little or no explicit aim to educate or mentor newcomers. Taking into account the professional support aspects leads to high-quality expectations. The community of portals has established specific rules to maintain professionalism and quality. In addition to the rules, these portals are moderated, i.e., the questions and answers submitted are checked by experienced members of the community or by professional moderators.

One of the best known such professional portal built around supporting software development is Stack Overflow, founded by Joel Spolsky and Jeff Atwood in 2008. From the beginning, the purpose of the portal was to support professional and hobbyist programmers and create a knowledge base to help developers solve problems they encounter in their daily work. The portal sets strict quality standards for users, and those questions that do not comply with the rules set by the community are closed or occasionally deleted.

The popularity of the portal has steadily increased since its inception, and in parallel, more and more questions have been received that did not meet the quality requirements of the community. This increase in popularity also resulted in an increased workload for the moderators. An increasing number of issues have been closed, accompanied by increased user dissatisfaction. The fact that the portal changed and simplified the applied rules did not change this discontent. It seems that it is a serious challenge not only for novice users but also for experienced users to compile questions that meet the expectations, which could be helped by a tool that could determine the quality of the questions and the chances of them getting closed before they are sent to the portal.

In Chapter 3, research focusing on the prediction of the question quality and the potential closing is presented. We focused on reducing the moderator workload with the help of the solutions we found, and at the same time, providing support to users in compiling questions of decent quality that meet the expectations of the portal. When compiling the questions, no information other than the content of the question itself, the title, and the tags that the questioner assigns to the question to thematize it are known. In other words, only textual information is available at the prediction time.

Two experiment series were conducted to achieve the designated goal. In the first series, the questions' quality was evaluated using the criteria of quality found in the relevant literature. A deep neural network was created based on a Gated Rectifier Unit layer following five fully connected and flattened layers to determine the questions' quality. The textual input was preprocessed and vectorized using two different tools. In the first case, the vectorization was executed using *Spacy*'s document vector generation process, and in the second case, the embedding was executed based on a specific *Doc2Vec* representation created on the Stack Overflow dataset. The experiments were executed using a growing number of examples in the training set, 425,097; 764,443; and 1,031,998 random samples, respectively. The results were promising; the best results were achieved in the last case, where the precision was 75% and the recall 74%, respectively. The results of the experiments surpassed the results of similar research.

In the second series of experiments, the likelihood of the closing was predicted using three different deep neural networks. The applied neural networks were based

on a unidirectional Gated Rectifier Unit, bidirectional Gated Rectifier Unit, and a composed model based on three bidirectional Gated Rectifier Units. In this case, only the body, the title, and the tags were considered in the prediction and the training process, while the quality-based training also utilized the score and the number of accepted answers. The embedding process was executed applying the *Keras Embedding* layer after vectorizing the input. During the vectorization, the size of the dictionary was reduced appropriately while the grammatical information was retained using a part of the speech tagging process.

Two experiments were executed in this series. In the first experiment, only the likelihood of the closing was predicted, while in the second case, the possible reason for closing was also determined. At the time of the experiments, no similar experiment was known in the literature; a similar study was performed in parallel with this work; however, based on the results of the competing experiments, the accuracy was higher for the models we used. The prediction results measured by accuracy were 71.87%, 71%, 70.84% of the unidirectional, bidirectional, and composite models, respectively, in the binary experiment. Similarly, the accuracy was 48.55%, 47.88%, 47.39% of the unidirectional, bidirectional, and composite models, respectively, in the multiclass experiment.

Based on the results, especially on those obtained for binary classifiers, it can be said that the objective of the research is achievable. Models applying natural language processing methods and deep learning algorithms can be used to create a tool that effectively helps users pre-evaluate the adequacy of their questions. The experiments were continued, better classifiers were created with more promising results in binary and multiclass cases, and a prototype application was also developed to evaluate the questions during their editing. The latter research is currently ready for publication.

Contributions of the thesis

In the **first thesis group**, the contributions are related to applying a Natural Language Processing and Artificial Intelligence Method in software development, especially in requirements engineering. A detailed discussion can be found in Chapter 2.

I/1. The author has developed preprocessing methods and a vectorization process applying the `tf-idf` representation form.

I/2. The author has implemented scripts responsible for executing the classification experiments using various machine learning models implemented in the `scikit-learn` library.

I/3. The author has implemented a simple neural network applied in the classification experiments based on the Stack Overflow samples.

I/4. The author executed the experiments, compared the results of the classifiers, and identified the best classifiers.

I/5. Based on the investigation of the semantics of the linguistic expressions, the author established a solid definition of the semantic space and the semantic networks, respectively.

I/6. The author has implemented preprocessing steps to extract posts from the Stack Overflow database and separate them into proper sentences cleaned from the auxiliary characters and noise.

I/7. The author has implemented a set of regular expressions based on the lexico-syntactic patterns representing the hyperonym-hyponym relationships found in the literature.

I/8. The author has developed a phrase structure grammar and an automatization to recognize noun phrases in the text. To the best of our knowledge, the grammar provided by the author is the most general formalized solution available in the literature.

I/9. The author has developed a simplified automatization for recognizing the noun phrases considering only a small set of the lexico-syntactic patterns.

I/10. The author has built a semantic network based on the hyperonym-hyponym relationships, representing the semantic field of the software development community based on the Stack Overflow post utilizing the lexico-syntactic patterns.

I/11. The author has investigated the structure of the resulting network and described its structure.

I/12. The author compared the smaller network resulting from the mining process with the semantic network representing the common knowledge provided by WordNet.

In the **second thesis group**, the contributions are related to the investigation of developers' interaction using natural language processing and deep learning. The research focused on evaluating the questions posted to the Stack Overflow portal. A detailed discussion can be found in Chapter 3.

II/1. The author has developed a GRU-based deep learning model to classify the questions based on their quality posted to the Stack Overflow considering only the textual elements. The definition of the quality was taken from the literature for comparison purposes.

II/2. The author executed the classification by applying different amounts of samples and vectorization processes.

II/3. The author compared the results with each other and the results of the other researchers. When all inputs were used in the classification process, the model's performance outperformed the performance of the classifiers used by others, demonstrating that the deep learning solutions can provide better results if there is enough input available.

II/4. The author has developed three distinct GRU-based deep learning models to classify the likelihood of closing questions posted to the Stack Overflow considering only the textual information available during the assembling of that question.

II/5. The author executed the classifications and compared the results with the other results available in the literature. The classifier provided good performances outperforming the other results, which can also be used in practice to evaluate the possibility of the question's closure before posting it to the Stack Overflow.

II/6. The author modified the models to perform multi-class classification. In this case, the classifiers predict the possible closing reasons. To the best of our knowledge, this classifier was the first published to predict the closing reasons. However, a parallel study was in progress and published lately for the same purpose, but the classifiers made by the author have a slightly better performance. The classifiers provide a good result and are applicable in practice.

Összefoglalás

Jelen Ph.D. értekezés olyan kutatást mutat be, amelyek célja természetes nyelvfeldolgozási módszerek és mesterséges intelligencia eljárások alkalmazásával a szoftverfejlesztők munkájának támogatása feladataik ellátásában.

A szoftverfejlesztés olyan összetett tevékenység, amely a szakmai ismeretek és a technikai készségek mellett a körülöttünk lévő világ jelenségeinek analitikus szemléletét is megköveteli. Ahhoz, hogy egy adott szoftver megfelelő szinten lássa el feladatát, szükség van az általa modellezett jelenség alapos megértésére és elemzésére, a probléma szakértői és az érintettek által megfogalmazott, és az üzleti elemzésben részletezett funkcionális és nem funkcionális követelmények alapján. A követelmények túlnyomórészt természetes nyelven állnak rendelkezésre, és a fejlesztés alatt álló szoftverre vonatkozó követelmények az érintett üzleti terület sajátos nyelvén fogalmazódnak meg. Mindezek mellett a követelmények jelentős része nemcsak ellenőrzött üzleti dokumentumokban és szabályzatokban, hanem az interjúk során szóban megfogalmazott ötletek és koncepciók formájában jelenik meg. Az utóbbiak sokszor pontatlanok, ellentmondásosak vagy hiányosak lehetnek. Ezenkívül az üzleti terminológia használata olyan hallgatólagos tudást is feltételez, amely magától értetődő az érintettek számára, de jellemzően nem ismert a fejlesztőcsapat tagjai számára.

A természetes nyelvfeldolgozó módszerek és a mesterséges intelligencia eszközök alkalmazása segítheti a fejlesztőcsapatot a természetes nyelven megfogalmazott követelmények feldolgozásának automatizálásában. Jelen kutatásban a követelmények osztályozására, valamint az érintettek és a fejlesztők különböző nyelvhasználatából adódó szemantikai eltérések egyeztetésére és kezelésére szolgáló eszközök kidolgozására összpontosítottunk. A disszertációban bemutatott kutatási eredmények alapján megállapítható, hogy a természetes nyelvfeldolgozó módszerek, valamint a mesterséges intelligencia eszközök, a gépi tanulás és a szemantikai hálózatok alkalmasak a kitűzött célok elérésére, a követelmény-feldolgozási folyamatok támogatására, illetőleg kiterjeszthetők a fejlesztői és üzleti interakciók egyéb eseteire is, például a felhasználói támogatási folyamatokra.

A szoftverfejlesztés az elmúlt évtizedekben jelentős átalakuláson ment keresztül. Ma már nem várható el senkitől, hogy egyetemleges ismeretekkel rendelkezzen a keretrendszerek, programozási nyelvek és eszközök hatalmas tárházáról. A fejlesztés ugyanakkor egyre inkább olyan újrafelhasználható elemekre támaszkodik, amelyeket

korábban már elkészítettek és a fejlesztő közösségek rendelkezésére bocsátottak.

A magas szintű specializáció alapján kijelenthető, hogy egyre fontosabbá válik, hogy a fejlesztők hatékonyan osszák meg tudásukat, és tudjanak segítséget kérni a különböző szakterületen dolgozó szakemberektől a napi munkájuk során felmerülő problémák megoldásához. A specializációból adódó problémák kezelése érdekében számos kérdezz-felelek portált hoztak létre az interneten, amelyek célja a fejlesztői tapasztalatok megosztása, valamint a fejlesztők támogatása a napi munka során felmerülő problémák megoldásában. Ezek közül a legismertebb a Stack Overflow, amelynek dedikált célja, hogy támogassa a fejlesztői munkát azáltal, hogy böngészhető tudástárat hoz létre a már előzetesen megoldott problémák gyűjteményéből.

Egy kérdezz-felelek oldal minőségének megőrzése a forgalom növekedésével jelentős kihívássá válik a moderátorok számára, ugyanakkor a közösségi elvárásoknak megfelelő kérdések megfogalmazása nem mindig egyszerű. Jelen kutatás a Stack Overflow-ra feltett kérdések minőségének meghatározására, illetőleg azok későbbi lezárásának becslésére összpontosított a természetes nyelvfeldolgozó eszközök, illetve a mélytanulás alkalmazásával. A kutatás része volt az esetleges kérdéslezárások okainak vizsgálata is. A kutatási eredményekből olyan gyakorlatban is használható modellek születtek, amelyek segítségével a felhasználók ellenőrizhetik kérdéseik követelményeknek való megfelelését, csökkentve ezzel mind a moderátorok leterheltségét, mind az utólagos lezárások lehetőségét, közvetve megőrizve a portálok szakmai színvonalát is.

A dolgozat két nagy részből áll. Az 1. fejezet a természetes nyelvfeldolgozás és a mesterséges intelligencia módszerek a követelmények feldolgozásában történő alkalmazási lehetőségeinek kutatását mutatja be. A 2. fejezet a fejlesztői interakciókat a természetes nyelvi feldolgozás és a mélytanulás módszereivel vizsgáló kutatásával mutatja be. A kommunikációs célplatform a Stack Overflow, és a kutatás a portálon feltett kérdések minőségére és azok lezárásának predikciójára koncentrált.

1. tézis: A természtes nyelvfeldolgozás és a mesterséges intelligencia módszereinek alkalmazása a követelménytervezésben

A követelmény-feltárás a szoftverfejlesztés elengedhetetlen része; egy sikeres szoftverprojekt nem létezhet enélkül. A követelmények különböző jogi, műszaki és üzleti dokumentumokban kerülnek meghatározásra, azonban jelentős részüket az üzleti elemző az érintettekkel folytatott interjúk során gyűjti össze. A követelményeket természetes nyelven fogalmazzák meg. Az interjúk során összegyűjtött követelmények viszont gyakran hiányosak és ellentmondásosak, köszönhetően az érintettek eltérő elvárásainak. A nem funkcionális követelmények, amelyek szintén lényeges részét

képezik ennek a halmaznak, gyakran nincsenek is határozottan megfogalmazva, viszont ezek figyelembevétele nélkül a szoftver nem tudja betölteni a feladatát. Mindezek mellett az összegyűjtött követelményeket az érintett terület üzleti nyelvét felhasználva fogalmazzák meg, és az adott szakterületen használt kifejezések szemantikája sokszor eltér azoktól, amelyeket a fejlesztők használnak a napi kommunikációjuk során. Az eltérő szemantikai értelmezések feloldásához sok esetben az érintett üzleti területet érintő általános ismeretekre is szükség lehet, amely a követelményelemzés során nem mindig áll rendelkezésre.

A 2. fejezetben a követelmény-menedzsment folyamatok támogatását célzó kutatás eredményeit mutattuk be. A kutatási program két részből tevődött össze. Az első részben a nemfunkcionális követelmények osztályozását vizsgáltuk a követelmény-dokumentumok alapján. A kutatás során két külön kísérletet végeztünk. Először egy kisebb címkézett adatkészletet, a Tera Promise NFR-t használtuk fel. Ez az adatkészlet 625 követelményt tartalmaz, amelyeket 15 különböző projektből gyűjtöttek össze, és 12 osztályba soroltak, amelyek közül egy osztály képviseli a funkcionális követelményeket. A kísérletek során különböző gépi tanulási algoritmusokat hasonlítottunk össze a bemeneti adatok vektorizált formáját alkalmazva. A vektorizáláshoz a *tf-idf* formátumot használtuk. A Promise NFR-ben gyűjtött rekordok száma meglehetősen szűkös a gépi tanulási feladatokhoz, bár a kísérletek elég jó pontossági értékeket eredményeztek. A legjobb eredményt a logisztikus regresszió (pontosság: 87%, visszahívás: 67%), a Multinomial Naive Bayes (pontosság: 84%, visszahívás: 68%) és az SVM osztályozó (pontosság: 89%, visszahívás: 65%) érte el. A kísérletet megismételtük a Stack Overflow kiválasztott bejegyzéseit használva. Ebben a kísérletben 50 000 példát választottunk ki, amelyek a használhatósági és tesztelhetőségi követelményekkel állnak kapcsolatban. Ezzel az adatkészlettel minden modell jobb pontossági eredményt adott. A legjobb eredményt a Fully Connected Network (pontosság: 96%, visszahívás: 94%), az SVM (pontosság: 95%, visszahívás: 94%) és a logisztikus regresszió osztályozók (pontosság: 95%, visszahívás: 95%) használatával értük el. A kísérletek eredményei határozottan azt sugallják, hogy a természetes nyelvfeldolgozó eljárások és a felügyelt tanulási módszerek alkalmazása hatékony és a gyakorlatban is alkalmazható támogatást tudnak nyújtani a követelmény-elemzési folyamatokhoz.

A kutatás másik vonalán algoritmikus megoldásokat kerestünk a kommunikációs nehézségek áthidalására annak érdekében, hogy támogatást nyújtsunk az üzleti terület és a szoftverfejlesztő közösség közötti kommunikációhoz. A kommunikációs nehézségek jelentős hányadát egyebek mellett a használt kifejezések közötti szemantikai különbségek okozzák, amit a szervezetpszichológia kommunikációs siló jelenségként emleget. A kommunikációs silók jelentős akadályt képeznek a szoftverfejlesztésben is, ugyanis a szoftver egy adott üzleti jelenség modellezésére használt megoldás, és egy modell elkészítésének elengedhetetlen eleme a vizsgált jelenség alapos megértése.

A felhasznált kifejezések jelentésbeli különbségét fel kell tárni, illetőleg fel kell azokat oldani annak érdekében, hogy a kommunikáció és a követelmények elemzése megfelelő színvonalon történjen.

A kutatás során feltártuk a kommunikációban használt fogalmak jelentését meghatározó tényezőket és a fogalmak közötti kapcsolatoknak a jelentés meghatározásában betöltött szerepét. Definiáltuk a modellezéshez használt szemantikai hálózat és az egyértelmű jelentést meghatározó szemantikai tér fogalmát. A kutatásban a Stack Overflow bejegyzéseket a szoftverfejlesztő közösség szemantikai környezete modelljének tekintettük. Ezekből a diskurzusokból kivonatoltuk a főnévi kifejezések közötti leggyakoribb kapcsolatokat, a hiperním-hiponím viszonyokat. Ezen összefüggések felismerésére lexikális-szintaktikus mintákat alkalmaztunk. A kapcsolatok lényeges részei, a főnévi kifejezések kinyerésére alkalmas kifejezés-struktúrájú nyelvtant és determinisztikus automatát definiáltunk, illetőleg használtunk fel. Az így kinyert összefüggések alapján építettük fel a szoftverfejlesztői közösség szemantikus terét modellező szemantikus hálózatot, és hasonlítottuk össze az e hálózat által meghatározott konkrét fogalmi jelentéseket az általános szemantikai környezetre épülő WordNet által meghatározott jelentéssel. E vizsgálatok eredményeiből megállapítható, hogy a szemantikus hálózatok alkalmas eszközök egy konkrét környezetben a pontos fogalmi jelentések megragadására, így lehetővé teszik a hallgatólagos tudás felismerését, és ezzel támogatják a releváns szoftverfejlesztési folyamatokat (követelménykezelés, végfelhasználói támogatás).

2. tézis: Fejlesztői interakciók vizsgálata természetes nyelvi feldolgozó eljárások és mélytanulás segítségével

A programozási nyelvek, keretrendszerek és a szoftverfejlesztésben használt speciális eszközök elterjedése a szoftverfejlesztőktől is magas fokú specializációt igényel. Ez a specializáció nem korlátozódik a szakmai célterületre, hanem jelentős hatással van a szoftverfejlesztésben használt technológiákra is. A fejlesztők a gyakorlati feladataik során elkerülhetetlenül találkoznak olyan technológiákkal, amelyeket nem vagy kevésbé ismernek, és amelyeket a felgyorsult fejlesztési ciklus miatt kevésbé tudnak elmélyülten tanulmányozni, ezért gyakrabban kell támaszkodniuk a szoftverfejlesztő közösség egészének támogatására, ahol a szükséges tudás rendelkezésre áll.

Ezekre az igényekre reagálva több kérdezz-felelek platform is létrejött az interneten, ahol a fejlesztők megoszthatják egymással tapasztalataikat, tudásukat, illetve segítséget kérhetnek a munkájuk során felmerülő problémák megoldásához. Ezeket a platformokat elsősorban a professzionális munka támogatására hozták létre, és csak kevésbé vagy egyáltalán nem céljuk a kezdők oktatása vagy mentorálása. A szakmai támogatás szempontjainak figyelembe vétele magas színvonalú elvárásokhoz

vezet. A portálok közössége szabályokat határozott meg a szakmaiság és a minőség megőrzése érdekében. A szabályokon túl ezek a portálok moderáltak, azaz a beküldött kérdéseket és válaszokat a közösség tapasztalt tagjai vagy foglalkoztatott moderátorok ellenőrzik.

A szoftverfejlesztés támogatására épülő szakmai portálok közül az egyik legismertebb a Stack Overflow, amelyet Joel Spolsky és Jeff Atwood 2008-ban alapított. A portál célja kezdettől fogva az volt, hogy támogassa a professzionális és a hobbi programozókat, valamint olyan tudásbázist hozzon létre, amely segíti a fejlesztőket a mindennapi munkájuk során felmerülő problémák megoldásában. A portál szigorú minőségi követelményeket támaszt a felhasználókkal szemben, a közösség által felállított szabályoknak nem megfelelő kérdéseket lezárják, illetőleg alkalomadtán törlik is őket.

A portál népszerűsége fennállása óta folyamatosan nő, ezzel párhuzamosan pedig egyre több olyan kérdést posztolnak, amely nem felel meg a közösség által elvárt minőségi követelményeknek. Ez a népszerűség-növekedés a moderátorok munkaterhének növekedését is eredményezte. Ugyanakkor egyre több kérdés kerül lezárásra, ami a felhasználók elégedetlenségének növekedésével jár együtt. Az a tény, hogy a portál megváltoztatta és egyszerűsített a szabályokon, érdemben nem eredményezett változást. Úgy tűnik, nemcsak a kezdő, hanem a tapasztalt felhasználók számára is komoly kihívást jelent az elvárásoknak megfelelő kérdések összeállítása, amiben segíthet egy olyan eszköz, amely képes meghatározni a kérdések minőségét és a kérdés lezárásának esélyét a portálra történő küldését megelőzően.

A 3. fejezetben a kérdések minőségének és lehetséges lezárásának előrejelzésére irányuló kutatásunk eredményeit mutatjuk be. A célunk az volt, hogy a kutatás során talált megoldások segítségével egyrészt a moderátorok terheltségét is csökkentsük, ugyanakkor támogatást is nyújtsunk a felhasználóknak a megfelelő minőségű, a portál elvárásainak megfelelő kérdések összeállításában. A kérdések összeállítása során magának a kérdésnek a tartalmán, a címén és a hozzárendelt címkék halmazán kívül, amelyeket a kérdező a kérdésének tematizálásához rendel, más információ nem ismert. Más szóval, csak szöveges információ áll rendelkezésre az előrejelzés időpontjában.

A kitűzött cél elérése érdekében két kísérletetsorozatot hajtottunk végre. Az első sorozatban a kérdések minőségét a szakirodalomban megtalálható minőségi kritériumok alapján értékeltük. E célből egy mély neurális hálózatot készítettünk Gated Rectifier Unit rétegre építve, amelyet öt teljesen összekapcsolt és egy a reprezentált tenzor dimenzióját csökkentő réteg követett. A szövegalapú bemenetet két különböző eszközt használva dolgoztuk fel, illetőleg vektorizáltuk. Az első esetben a vektorizálás a *Spacy* dokumentumvektor generátorával, a második esetben a Stack Overflow adatkészleten létrehozott *Doc2Vec*-reprezentáció segítségével valósult meg. A kísérleteket a tanítókészletben szereplő, egyre növekvő számú 425 097, 764 443 és

1 031 998 véletlenszerű mintát választva hajtottuk végre. Az eredmények ígéretesek voltak; a legjobb eredményt az utolsó esetben értük el, ahol a pontosság 75%, a visszahívás 74% volt. A kísérletek eredményei a korábbi kutatások eredményeinél jobbnak bizonyultak.

A második kísérletsorozatban a lezárások valószínűségét három különböző mély neurális hálózat segítségével vizsgáltuk. Az alkalmazott neurális hálózatok egy egyirányú Gated Rectifier Unit rétegre, kétirányú Gated Rectifier Unit rétegre, valamint három különböző Gated Rectifier Unit rétegen alapuló kompozit modellre épültek. Ebben a kísérletben csak a szövegtörzset, a címet és a címkéket vettük figyelembe az előrejelzési és a tanítási folyamatban is, míg a minőség alapú tanításnál a pontszám és az elfogadott válaszok száma is felhasználásra került. A beágyazási folyamat a *Keras Embedding* réteg alkalmazásával történt a bemenet vektorizálása után. A vektorizálás alatt a felhasznált szótár méretének csökkentése és a nyelvtani információk szófaji meghatározás segítségével történő megőrzése kiemelt szempont volt.

A sorozatban két kísérletet hajtottunk végre. Az első kísérletben csak a lezárás valószínűségére helyeztük a hangsúlyt, míg a második esetben a lezárás lehetséges okát is meghatároztuk. A kísérletek időpontjában más hasonló kutatás nem volt ismert a szakirodalom alapján, azonban ezzel a munkával párhuzamosan készült hasonló vizsgálat, viszont a konkurens kísérletek eredményei az általunk használt modellek eredményeitől elmaradtak. A bináris osztályozási kísérletben az egyirányú, kétirányú és kompozit modellek 71,87%, 71%, 70,84% pontosságot eredményeztek. Hasonlóképpen, a pontosság 48,55%, 47,88%, 47,39 % volt az egyirányú, kétirányú és kompozit modelleknél a többosztályos kísérletben.

Az eredmények, különösen a bináris osztályozókra kapott eredmények alapján elmondható, hogy a kitűzött célok elérhetők. A természetes nyelvfeldolgozás eszközei és a mélytanulás modelljei segítségével olyan eszközt lehet létrehozni, amely hatékonyan segíti a felhasználókat kérdéseik megfelelőségének előzetes felmérésében. A kísérleteket tovább folytattuk, és sikerült a fenti eredményeknél jobb osztályozókat készítenünk bináris és többosztályos esetekben is, valamint egy prototípus-alkalmazást is fejlesztettünk a kérdések előzetes értékelése céljából. Ez utóbbi kutatás megvalósult, jelenleg publikálást megelőző szakaszban van.

A disszertáció tézisei

A **első téziscsoport** a természetes nyelvi feldolgozás és a mesterséges intelligencia módszereinek a szoftverfejlesztésben, különösen a követelményelemzésben történő alkalmazásához kapcsolódik. Részletes tárgyalása a 2 fejezetben található.

I/1. A szerző `tf-idf` reprezentációra épülő előfeldolgozó és vektorizáló eljárást dolgozott ki és implementált.

I/2. A szerző a `scikit-learn` library-ben implementált gépi tanuló modellekre épülő osztályozó kísérleteket futtató scriptet készített.

I/3. A szerző a Stack Overflow mintákat használó kísérletekhez egy egyszerű neurális hálózatot is készített.

I/4. A szerző elvégezte az osztályozási kísérleteket, a kapott eredményeket összehasonlította és azonosította a legjobb eredményeket adó modelleket.

I/5. A nyelvi kifejezések szemantikájának vizsgálata alapján a szerző pontos definíciót adott a szemantikus tér és a szemantikus hálózat fogalmáról.

I/6. A szerző előfeldolgozó eljárást készített a Stack Overflow posztok kivonatolására, amelyek segítségével eltávolította belőlük a zajt, valamint az egyéb karaktereket, illetőleg elvégezte a mondatokra tagolást.

I/.7 Az irodalomban talált, hiperním kapcsolatokat felismerő lexiko-szintaktikus mintákhoz a szerző reguláris kifejezéseket készített.

I/8. A szerző a főnévi kifejezések felismeréséhez egy kifejezés struktúrájú nyelvtant, valamint a kapcsolódó determinisztikus automatát definiálta. Legjobb tudásunk szerint a szerző által adott nyelvtan az irodalomban ismert legáltalánosabb megoldás a főnévi kifejezések felismeréséhez.

I/9. A szerző egy egyszerűsített automatát is definiált a főnévi kifejezések felismeréséhez egy kisebb számosságú lexiko-szintaktikus mintákat tartalmazó halmaz figyelembevételével.

I/10. A szerző a szoftverfejlesztés szemantikus környezetének reprezentálására hiperním-hiponím kapcsolatokat ábrázoló szemantikus hálózatot készített Stack Overflow posztokból, lexiko-szintaktikus mintákat használva.

I/11. A szerző megvizsgálta és dokumentálta a létrejött szemantikus hálózat szerkezetét.

I/12. A szerző a folyamat eredményeképpen létrejött kisebb méretű hálózatot összehasonlította a WordNet által reprezentált szemantikus hálózattal.

A **második téziscsoport** kontribúciói a természetes nyelvi feldolgozást és a mélytanulást alkalmazó kutatáshoz kapcsolódnak, amelyek célja a fejlesztői interakciók vizsgálata. A kutatás a Stack Overflow portálra feltett kérdések értékelésére összpontosított. Részletes tárgyalás a 3 fejezetben található.

II/1. A szerző GRU alapú mélytanuló modellt készített a Stack Overflowra publikált kérdések minőségének szöveges információn alapuló osztályozásához. A minőség definícióját a kapcsolódó szakirodalomból vette összehasonlítás céljából.

II/2. A szerző elvégezte az osztályozási kísérleteket különböző számosságú mintát és vektorizálási eljárást alkalmazva.

II/3. A szerző összehasonlította az eredményeket egymással, valamint a szakirodalomban publikáltakkal. Abban az esetben, ha minden input felhasználásra került, akkor a modellek teljesítménye a szakirodalomban publikált teljesítményt felülmúlta demonstrálva ezzel, hogy megfelelő számú tanítópélda esetében a mélytanuló modellek által adott megoldások jobb eredményt adnak.

II/4. A szerző három GRU alapú modellt készített a Stack Overflowra feltett kérdések lezárásának prediktálására, kizárólag a szöveges információkra alapozva.

II/5. A szerző elvégezte a kísérleteket, összehasonlította az eredményeket egymással és az irodalomban rendelkezésre álló eredményekkel. Az osztályozók jó eredményeket adtak, az irodalomban ismert eredményeket felülmúlták, valamint a gyakorlatban is használhatók a Stack Overflowra feltett kérdések lezárási valószínűségének előrejelzésére, mielőtt azokat ténylegesen elküldenék a portálra.

II/6. A szerző módosította a modelleket, hogy többosztályos osztályozásra is alkalmasak legyenek. Ebben az esetben a modellek a lezárási okok prediktálását végezték. Legjobb tudásunk szerint ezek a modellek az első publikált modellek, amelyek célja a lezárások okainak prediktálása. Jelen munkával párhuzamosan zajlott egy kutatás hasonló céllal, azonban a szerző által adott modell valamelyest jobb eredményt mutat a párhuzamos kutatásban fejlesztett modell teljesítményénél. A modellek elfogadható eredményeket adtak, amelyek a gyakorlatban is alkalmazhatók.

Publications

Journal publications

[I] **László Tóth** and László Vidács Comparative Study of The Performance of Various Classifiers in Labeling Non-Functional Requirements. *Information Technology and Control*, 48(3), 432-445, 2019.

Papers in conference proceedings

[II] **László Tóth** and László Vidács Study of various classifiers for identification and classification of non-functional requirements. In *Computational Science and Its Applications – ICCSA 2018*, Springer, 492-503, 2018.

[III] **László Tóth**, Balázs Nagy, Dávid Janthó, László Vidács, Tibor Gyimóthy Towards an Accurate Prediction of the Question Quality on Stack Overflow using a Deep-Learning-Based NLP Approach. In *Proceedings of the 14th International Conference on Software Technologies*, Institute for Systems and Technologies of Information, Control and Communication, 631-639, 2019.

[IV] **László Tóth**, Balázs Nagy, Tibor Gyimóthy, László Vidács Why Will My Question Be Closed? NLP-Based Pre-Submission Predictions of Question Closing Reasons on Stack Overflow. In *2020 ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results - ICSE-NIER* Association for Computing Machinery (ACM), 45-48, 2020.

[V] **László Tóth** and László Vidács Analyzing Hyperonyms of Stack Overflow Posts. In *The Seventh International Conference on Fundamentals and Advances in Software Systems Integration – FASSI 2021*, IARIA, 1-6, 2021.

Further related publications

[VI] **László Tóth** Preliminary Concepts for Requirements Mining and Classification using Hidden Markov Model. In *11th Conference of PhD Students in Computer Science*, 110-113, 2018.

[VII] **László Tóth**, Balázs Nagy, Tibor Gyimóthy, László Vidács Mining Hypernyms Semantic Relations from Stack Overflow. In *2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops - KG4SE* Association for Computing Machinery (ACM), 360-366, 2020.

Acknowledgments

First of all, I would like to thank my supervisors, László Vidács and Tibor Gyimóthy, for directing my Ph.D. studies and for helping me get through the emerging difficulties. I would like to thank the language lector, Edit Szűcs, for the grammatical control of the dissertation. I would also like to thank my colleagues and friends who helped me realize the results presented here and made me enjoy the period of my studies. Last but not least, I wish to thank my wife and family for their constant love, patience, and support.