A Bird's Eye View on Requirements Engineering and Machine learning

Author1, Author2, and Authorn

fortiss GmbH. Guerickestraße 25, 80805 München author@fortiss.org

Abstract. Machine learning (ML) has demonstrated practical impact in a variety of application domains. Software engineering is a fertile domain where ML is helping in automating different tasks. In this paper, our focus is the intersection of software requirement engineering (RE) and ML. To obtain an overview of how ML is helping RE and the research trends in this area, we have surveyed a large number of research articles. We found that the impact of ML can be observed in requirement elicitation, analysis and specification, validation and management. Furthermore, in these categories, we discuss the specific problem solved by ML, the features and ML algorithms used, and datasets, when available. We outline lessons learned and envision possible future directions for the domain.

1 Introduction

This is an introduction [10]
RQ1: How is ML used inside RE tools?
RQ2: Is using ML inside RE tools beneficial?
RQ3: What ML task(s) is (are) used for which purpose in RE?
RQ4: Which model types are used to perform the RE tasks?
RQ5: How are ML features extracted/selected to guide RE tasks?

1.1 Machine Learning

Machine-Learning (ML) [**?**] is a range of algorithm to approximate functions and discover patterns in data. Historically, models and heuristics are human-built exhaustive prescriptions of how a system should behave. ML is grounded on different premises: rather than relying on humans to input all the possible cases the system can handle, the field attempts to extrapolate patterns from a representative set of examples that illustrate the expected behaviors. The way in which a learning algorithm operates attempts to emulate the way in which humans learn: from a set of examples, a general model for a behavior is induced.

Many learning algorithms exist, based on different visions of how learning happens in practice [10]. All these algorithms have in common the notion of *features*. Features correspond to characteristics of what is being learned and provide the grounds for the algorithm to abstract from the complexities of the real world. Assume for example that an algorithm should learn, based on a brain scan of a medical patient, to decide whether that patient has brain cancer or not. A number of *features* such as for example the "number of irregular objects in the scan", the "color of such objects", the "disposition of such objects" would be provided to the algorithm. Additionally, the algorithm is fed a number of brain scans together with decisions previously taken on them (cancer found / cancer not found) – the *training data*. The learning algorithm then undergoes a *training phase*. It attempts to find an internal model that allows it to map the decisions to the brain scans, given the training data. The model obtained from the training step is useful if it performs well (generalizes) when applied to new data from outside the training set – in our example, when it can accurately diagnose brain cancer for new brain scans. Such generalization is based on the premise that inputs that are "closer", in terms of the given *features*, should lead to "closer" outputs.

The formal notion of "closeness" is a characteristic of the learner algorithm being employed and determines how the algorithm generalizes the computation from the given examples. Achieving good generalizations is the cornerstone of machine learning and *overfitting* (performing very well on training inputs but very poorly on new inputs) is one of its major challenges. Levi ►cut these last couple of sentences◄

More formally, in textbooks, courses and articles, Machine Learning is often defined following the definition of Tom Mitchell [**?**]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Therefore, it is said that to classify some patients into classes (e.g healthy and unhealthy), the task T, one have to define an algorithm that provides a model, such as an artificial neural network. The quality of this model is quantified by a measure P, for instance its accuracy while predicting the classes. This measure is then sent back to the algorithm, a new experience E, in order to choose or improve the model. A machine learning tasks can be discussed and subdivided based on the elements of the following equation:

$$f(\mathbf{X}) = \mathbf{y} + \xi$$

where $\mathbf{X}$ is the $n \times d$ input matrix, containing n samples characterized by d features, $\mathbf{y}$ is the $n \times 1$ target vector containing the classes of the n samples and ξ is a $n \times 1$ vector representing the noise. The goal is to approximate f in order to provide the best mapping between $\mathbf{X}$ and $\mathbf{y}$, given some noise ξ. Indeed, the approximation has to map a $\mathbf{X}$ containing noise, to a $\mathbf{y}$ which may contain noise too. For instance, uncontrolled conditions such as the room temperature and the exposure time to this temperature can induce variations in the information contained in collected blood samples. Moreover, the ξ term also contains the approximation error when, for example, one tries to approximate a non-linear function with a linear function.

The problem presented by the above equation is called *supervised learning*, and can be roughly subdivided in two popular problems: *classification* and *regression*. When the target vector $\mathbf{y}$ is composed of categorical values (i.e. classes), then we have a classification problem. The goal is to learn how to link instances or samples in $\mathbf{X}$ to a certain class (e.g. healthy patient or unhealthy patient). However, if the target vector contains continuous values, we face a regression problem (e.g. predict the body temperature of a patient given some clinical features of the patient).

In some cases, $\mathbf{y}$ is not given and we have to find patterns in $\mathbf{X}$ "blindly." This is called an *unsupervised* learning problem. Finding clusters in $\mathbf{X}$, i.e. finding a $\mathbf{y}$ that has never been given, is such a problem. For instance, one may want to group patients based on symptoms they have.

Reinforcement Learning can be seen as an intermediate problem where $\mathbf{y}$ is not given but the procedure is guided nevertheless. In RL, an agent has to find a sequence of actions leading to a success. The fact that the sequence leads to a success, i.e what would be in $\mathbf{y}$, is not known in advance, but rewards are given to the agent in order for him to know if it follows a path to success. In other words, the goal is, by providing rewards along the way, to find the sequence of actions leading to the desired state. Typical examples can be found in gaming, where an agent receives a reward when he wins the game. From the different chains of actions that led him to a reward, the agent must generalize to find how to win that game.

1.2 Requirements Engineering

Software systems are developed over millions of lines of code, number of modules and documents. The primary goal of the software system is to satisfy users by developing the software that can meet their needs and expectations. This goal is achievable by applying different methodologies and engineering techniques. One of the key factor is to understand and identify the needs of users, also known as, software requirements. Software requirement engineering is the process that helps to determine the requirements in a systematic way to know what functionalities the targeted system should have to fulfil user's needs. Formally RE is defined as [46]:

> "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

Software requirements play a key role in the success of a project. In the USA, a survey was conducted over 8380 projects by 350 companies to know the project failure rates. The report overall results showed only 16.2% projects were completed successfully and one-half (52.7%) of projects met with challenges and

were completed with partial functionalities, time delays and over budget. Almost 31% of the projects were never completed. The main cause told by the executive managers was the poor requirement. The major problems were the lack of user involvement (13%), requirements incompleteness (12%), changing requirements (11%), unrealistic expectations (6%), and unclear objectives (5%).[**?**] [6]
Software requirement engineering has mainly four phases; requirement elicitation, requirement analysis, requirement documentation and requirement verification [26]. Requirement elicitation [6,47] helps to understand the stakeholders needs, e.g. what features he wants in the software. Requirement elicitation techniques are mostly derived by the social sciences, organizational theory, knowledge engineering and practical experience. For requirements elicitation, different techniques exist in the literature that include interviews, questioners and ethnography etc. Requirement analysis [31] is the next step after requirement elicitation. In this phase, software requirements are analyzed to check conflicts and consistency of requirements. It is also makes sure that the requirements are clear, complete and consistent. Furthermore, the agreed requirements are documented. This documentation has a clear and precise definition of the system functionalities. It also acts as an agreement between stakeholders and developers. These functionalities and requirements are documented usually as diagrams, mathematically formulae or natural languages. These documents are used and iterated until the end of the projects. System requirements are classified into businees requirments, user requirments, functional requirements (FR) and non-functional requirements (NFR). Functional requirements are the system requirements that include the main features and characteristics of the desired system. Non-functional requirements are the system properties and constraints [7]. NFRs set the criteria for judging the operation of the system e.g. performance, availability and reliability etc. Levi ►what about the business and user requirements. Do we consider them?◄

2 Contributions

Levi ►summarize preprocessing steps and feature calculation◄ Levi ►disclaimer: sometimes features are presented sometimes not◄

2.1 Requirements Elicitation and Discovery

The manual process of requirement elicitation is time consuming and required efforts. The project success majorly depends upon right identification of stakeholder expectations and requirements. The latest trend for identification of user requirement is mining social media like twitter, google store, and app store data. Levi ►add citation◄ These user reviews are not structured requirement and contained other information too such as praise, dislikes, bug report etc. The manual analysis of these large user reviews dataset and identify the user requirements hard. The automated analysis of requirement elicitation is helpful and can significantly reduce time and cost. This is ML classification task: give the set of information and identifying it as requirement or not. Levi ►we need to speak about other sources of requirements other than social media◄

[4], [24], [3]

Guzman etal [17] proposed ALERTme approach for classifying, grouping and ranking tweets in software evolution process. For identification of evolutionary requirement supervised classification performed on tweets as improvement request or not using Naïve Bayes algorithm. It was the first kind of this study that finds user requirements from tweets for software. Williams etal Levi ►use the *et al.* command◄ [44] performed a similar study on tweets for classifying as user requirement or not. The study gave state of the art results using Naïve Bayes and VSM algorithm. Different techniques were implemented for feature extraction that includes stopword removing, sentimental analysis, stemming Levi ►bring to machine learning section◄ , and Bag of words. However, results showed that software tweets are neutral in nature and sentiment analysis did not influence ML algorithm. Levi ►instances are manually classified tweets◄

Research on mining user reviews in mobile application (app) stores has remarkably advanced in the past few years. Jiang etal [22] presented an optimized method for discovering the evolutionary requirements for developers. This method clustered opinion expressions in form of a macro network topology, and combine polarized sentimental analysis with the economic factors to decide evolutionary requirements. The dataset used POS tagger and parser with additional defined rules for feature extraction. Douglas S. etal [28] mapped software requirement elicitation process to an existing military tool skiweb that used for making the decision. The proposed methodology used supervised Naive Bayes to classify text document to find related requirements. Furthermore, recommendation system used topic modeling to identify the key stakeholder for which that requirement is important and allow for the analysis. Jha etal [21] Classified application store reviews and discover user requirement as feature request. The data was classified into three categories bugs, features, and junk. For classification, SVM and Naïve Bayes algorithm was used. The study used frame semantic analysis for feature identification, this technique produced slower Levi ►slower?◄ dimensional model with smaller number of features. [47].

Prioritization of Requirements

Text mining Levi ►don't understand this sentence◄ Kaiya etal [24] proposed a tool to improve the ontology of domain knowledge for requirement elicitation by using web mining and NLP technique. It helped to mine the general concepts to ontology for elicited requirements. Hollis etal [19] proposed an initial study to automate requirement elicitation in Agile environment. By providing the list of words and loosely formatted list of requirements. The proposed methodology applied text mining technique on recorded conversation of the stakeholder and developer conversation. Dong etal [11] also applied text mining on different form of document and resources from internet for gathering requirement. The system applied data preprocessing as word segmentation and stop words removal before building up the VSM model. Levi ►not clear what the ML contribution is here. Model, features and instances are missing.◄

2.2 Requirements Specification and Analysis

Non-Functional Non-functional requirements may not be explicitly mentioned in a formal specification requirements documents even though, all systems have them [39]. Moreover, freeform documents like interview notes, meeting minutes and scattered requirements specifications include non-functional requirements which need to be detected and classified. In order to support analyst in the error-prone task of manually discovering and classifying NFRs machine learning can be useful. Automatic detection can be used to quickly and more effectively analyze large and complex documents for searching the NFRs[5]. This is a classification problem as from a set of requirements we want to decide a class membership.

One of the studies is by Slankas *et al.* [39] where they automatically identified and classified sentences in natural language from use agreements, install manuals, regulations, request for proposals, requirements specifications, and user manuals output into 14 different NFRs categories: Access Control (AC), Audit (AU), Availability (AV), Legal (LG), Look and Feel (LF), Maintenance (MT), Operational (OP), Privacy (PR), Recoverability (RC), Performance and Scalability (PS), Reliability (RL), Security (SC), Usability (US) Levi ►only introduce acronyms if they are used further ahead in the paper◄ . Their two-step process: 1) parse natural language and turn sentences into graphs 2) classify sentences into categories with k-nearest neighbor algorithm led them into finding 20 keywords for each category of NFRs for their classifier. They trained the NFR classifier with a wide variety of open and closed source EHRs (Electronic Health Record), various industry standards (HL7, CCHIT), governmental regulations, and other document sources exist to elicit documentation.
Cleland-Huang *et al.* [5] provided the same approach and used k-nearest neighbor classification for grouping non-functional requirements: availability, look-and-feel, legal, maintainability, operational, performance, scalability, security, and usability. For training their classifier they used 15 requirements specifications developed as term projects by master students at DePaul University.

Functional Software requirements specifications are usually stated in informal, imprecise and ambiguous natural language, thus analyzing them is a challenging task. However, for requirements reuse in Software Product Line Levi ►why do you introduce SPLs here?◄ analyzing them zis a vital task. Automatically extract structured information of functional requirements from Software Requirements Specifications and grouping them into different categories is a machine learning classification task[41].

Wang *et al.* [41] applied a combination of machine learning, natural language processing, and semantic analysis methods for automatically extract functional requirements and classify them into 10 different cases: Agentive, Action, Objective, Agent mode, Objmod, Locational, Temporal, Manner, Goal, Constraint Levi ►the class names are not informative◄ . Their framework has four steps: corpus construction, NLP preprocessing, feature extraction and EFRF (Extended Functional Requirements Frame) functional cases extraction. which for NLP processing they did tokenization, lemmatization, part-of-speech tagging (POS tagging) and dependency parsing. They trained their bi-directional LSTM-CRF network which is a variant of Recurrent Neural Networks(RNN) architecture model with E-commerce requirements dataset and test it on requirements of automaker systems. Ultimately, they showed that their trained model on E-commerce requirements dataset can be used to extract semantic information from the requirements of automaker systems Levi ►which kind of semantic information?◄ .

Functional and Non functional Requirements The success of a system does not solely depend on its functional requirements. Like functional requirements, it also significantly depends upon the adherence to non-functional requirements. However, the primary focus is generally more towards identification and specification of FRs. NFRs are usually identified and specified in later development stages that can increase the risks in development life cycle. FRs tend to be more straightforward e.g. store and retrieve transaction. On the other side, NFRs are complicated and challenging to implement e.g., making the design to meet NFRs or design test case for them. Different types of requirements analyzed in a different way, and it is useful to have a separate division to look at one particular division. That is why it is necessary to distinguish between FR and NFR and categorize NFRs into subcategories. This distinction helps to manage changes in requirements. The manual division is difficult and time consuming. Machine learning can be used for reducing the effort and categorizing the requirements based on the text segment analysis. This is ML classification task: give the set of requirements and identifying its category.
Mengmeng Lu *et al.* [29] automatically classify the user review text mobile from application (app) stores into FR, NFR, and others. It further classified NFRs into four categories including reliability, usability, portability, and performance. It used supervised machine learning algorithm (bagging) for training the classifier. The text trimming used stopwords elimination, lemmatization, stemming, and sentences split. Word2vec was used for augmenting the user review, which is a two layer neural network model to process text for finding the word embedding. Deoxadez *et al.* [9]performed semi-supervised classification techniques for automated classification of FRs and NFRs on user reviews from the app store. This study dealt with two problems: 1) minimize annotation effort or label the big dataset of user reviews, and 2) classification of FRs and NFRs. The proposed

solution to solve the first problem used semi-supervised self-labeling algorithm. Self-labeling algorithms required a small amount of dataset to get comparable results as supervised techniques. Naïve Bayes algorithm was selected because of high performance results for classification problem. Features were obtained by applying standard text mining technique and additional attribute embellishment. For text mining technique, included features were Inverse Document Frequency (IDF) transform, Term Frequency (TF) transform, lowercase transformation, minimum term frequency, stemmer, and number of words. The second stage involved removing numbers, 2- letter words and other symbolic characters. Kutranvoic *et al.* [27]performed automated analysis on software requirements and performed classification on FRs, NFRs and subcategories of NFRs using supervised machine learning algorithm(support vector machine). Some NFR had negligble requiremtnst in dataset due to this they were ignored. The additional dataset of user comments from Amazon was integrated into the main dataset to avoid the data imbalance problem in NFR. The predictor used text-preprocessing techniques such as removal of punctuations, removal of stop words, and lemmatization for feature extraction.All experiments were performed on dataset given by RE datatrack. Abad *et al.* [1] targeted two problems: first is classifying the FR and NFR and second classification of NFR into categories. Preprocessing of text performed by applying: POS tagging, Entity tagging and temporal tagging .Fas a next step feature co-occurrence and regular expression used to increase the weight of influential words in the datset. J.48 DT used for the classifying the FR and NFR. For achieving second goal topic Modeling unsupervised algorithm LDA and BTM applied. For topic generation differnt algorithms such as clustering, k-means, hybrid clustering and k-means, BNB were applied.The results showed BNB worked better out of clustering, k-means, LDA, BTM. learnt decision tree to classify the FRs and NFRs and improved results from 89% to 95% by analyzing the requirements in natural language. For NFRs subclassifcation, Binarized Naïve Bayes (BNB) achieved highest results. The preprocessing of text involved, Part of Speech (POS) tagging, entity tagging, and temporal tagging.Garzoli [14] classified data into five types: FRs, NFRs, design and construction constraints, operator requirements, performance requirements.The datset was taken from combat management system of a Naval Combat System in generall complex system domain. However, the primary goal was to come up with a general architecture for large-scale and adaptive requirement analysis. For classification multi VSM model used and for lexical and grammatical features BoW + N-words + N-POS achieved highest accuracy.

[9], [27], [17], [1], [8], [35], [29], [18], [44], [14], [2], [42], [20], [22], [21], [33]

Security Requirements Due to the orthogonal character of their impact on a system, *security* requirements are notoriously difficult to identify, objectify and quantify []. Also during requirement specification, it very often happens that security requirements are masked by functional requirements (but can be deduced from the context of the domain the system operates in) [38]. Because of this, it often happens in practice that security requirements are only marginally tackled during system construction [], paving the way to potentially catastrophic consequences. Machine learning can be of use here by aiding in the identification of segments of text that describe security requirements. This is a *classification* problem: given a text, identify which parts of it correspond to which type of security issues.

Jindal*et al.* [23] automatically learn decision trees that can be used to classify security requirements as *authentication*, *access control*, *encryption* or *data integrity*. The *features* used are relevant terms found in the text. Such relevant terms are obtained by the following sequence of actions: 1) removing stop words from the text; 2) stemming the remaining words; and 3) ranking the stemmed words by their *info-gain* measure.

Riaz and her colleagues [38] use the k-nearest neighbors algoritm to classify sentences in requirements documents as *confidentiality*, *integrity*, *authentication*, *availability*, *accountability* or *privacy* requirements. In order to find adequate sentences and provide context to the classifier, the authors start by finding a type for each sentence among the possibilities *title*, *list start*, *list element* or *normal sentence*. For the classification the authors use a modified version of the Levenshein distance [] based on the number of word transformations needed to go from one term in one sentence to a term in another sentence. The classifier is trained using requirements sentences from the healthcare domain that are manually classified. A particularity of the approach is that each security requirement type is associated to a template that helps in translating the security requirements into functional requirements in order to ease during the implementation of the final system.

[25]

Contextual Requirements

2.3 Requirements Validation

Traceability Validation is to guarantee that requirements are reflecting stakeholders' needs, confirm the quality of the system, consistency, and traceability. One of the definitions for requirements traceability is given by [16]:

> "Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)."

Based on this definition the emphasis is on the ability to track the life of requirements and their established links within other artifacts. However, the main barrier assures traceability is the needed effort for building and maintaining the links between those artifacts. That is why many research has tried to apply machine learning and automated tools for facilitating the establishment of links[15].

Traceability tackled in the research mainly by the use of machine learning classification methods Levi ►only classification?◄ .

Wieloch *et al.* [43] introduced the hurdle of traceability methods that enhance the accuracy of tracing requirements which occur frequently across multiple projects and/or domains. In the paper, they present Trace by Classification (TBC), a machine learning approach in which a classifier is trained to generate Levi ►a classifier cannot *generate*◄ trace links for requirements and/or other kinds of software artifacts which occur relatively frequently across different projects. The first step in their approach is preprocessing which is to eliminate common stop words. Then, there is training phase that a set of indicator terms is identified for each NFR category and the classifier will be trained by the set of identified weighted indicator terms that can then be used for the last step which is to classify target documents into specific types Levi ►didn't get this or the relation with traceability◄ .

Gervasi *et al.* [15] investigate what can be learned from links that are already established. They build classifiers as a mean to develop models of tracing that can then be interpreted by humans to understand how requirement tracing is done in practice. Their purpose is to revise the existing models of hard-coded traceability tools such as VSM. They used a publicly-available dataset of requirements with traceability information, originally based on the CM-1 project by the NASA Metrics Data Program. Their approach has 5 steps: 1) tokenize and stem requirements and removing stopwords 2) derive two features from each term t in the vocabulary, one for the occurrence of t in a high-level requirement and one for the similar occurrence in a low-level requirement 3) transfer requirements into a vector of features 4) derive set of classification cases Levi ►what does this mean?◄ 5) finally, use the dataset to train and test two different classifiers from the WEKA collection, a Naive Bayesian classifier and the J48 decision-tree classifier.

Sultanov *et al.* [40] finds traceability candidates from high-level to low-level requirements by the use of reinforcement learning Levi ►so there is some reinforcement learning going on in traceability. This should also be mentioned at the top of the section…◄ . They used textual high and low-level requirements documents as an input and try to find the candidate traces. Their technique demonstrated statistically significantly better results than the Information Retrieval technique.

2.4 Requirements Management

Visualization Natural language requirement documents can be hard to comprehend and analyze. Similarly, stakeholders have to review and understand requirements for large and complex systems. In these scenarios, basic information visualizations, like charts and graphs have been used in requirements engineering. These visualizations are usually applied to improve textual requirements with summarization that combined large amounts of information into a single representation for quick consumption by stakeholders[37]. Machine learning is of great value in discovering visualized groups of large numbers of requirements artifacts Levi ►I think you mean "machine learning is useful in grouping requirements for visualization purposes"◄ . Levi ►which kind of machine learning taks? Clustering?◄

ReCVisu (Requirements Clustering Visualization) tool is presented in Reddivari *et al.* [37] paper. ReCVisu, an exploration tool based on quantitative visualizations helps requirements engineers understand the nature of the requirements in a visual form. In ReCVisu, the dependence graph consists of requirements artifacts as nodes and the textual similarities as edges. The automatic grouping of requirements into clusters can help in areas such as uncovering the requirements structure, navigating around the requirements space, modularizing crosscutting concerns, and understanding requirements interactions and evolution.

Pinqui *et al.* [34] recognize the enormous volume of requirements as big data with which companies struggle to make strategic decisions early on. Therefore, they built a complete visual framework to filter requirements from stakeholders in a way that architects can make better insightful decisions. They suggest training a multi-class SVM model from domain-specific (mechanics, electronics, etc.) dictionaries and handbooks. Overall, the paper proposes a framework to go from management-oriented requirements to architecture-oriented requirements in which SVM is only applied in a small part of it.

Software requirements are mostly stated in natural text notations such as user stories which is making it hard for people to develop an accurate mental image of the most relevant entities and relationships. Lucassen *et al.* [30] introduced an automated method for visualizing requirements at different levels of granularity. Their visualization method from user stories consists of 1) the generation of an overview which provides a general context for understanding the dataset:

- Extract a set of relevant concepts from the user stories and their relationships
- Calculate the semantic similarity by using skip-gram implementation word2vec
- Utilize Ward's clustering algorithm to group all the concepts according to their similarity
- Identify the concept which is most similar to the collection of concepts in a cluster
- Generate inter-cluster relationships matrix
- Visualization Drawing

2) zooming in and out mechanisms and 3) filtering techniques to reduce the complexity of the data presentation. Possible anticipated applications of this visualization are: discovering missing relationships between clusters that may result in further user stories, teaching system functionality by exploring simplified, manageable chunks, and analyzing expected system changes after introducing new sets of user stories.

Structuring Documents Requirements of the system are usually presented in natural language documents. These documents require to be properly structured for a better overall understanding of the requirements. For this purpose, the document should be organized with independent sections which each one contains conceptually connected requirements[13]. Moreover, technical review is a usual way to guarantee the quality in natural language specifications. However, extensive and comprehensive specifications make it problematic for reviewers to find defects, especially consistency or completeness ones. Therefore, use of machine learning algorithms can support reviewers with their work by automatically classifying and clustering the information that is spread over many sections of many documents [32].

[13]
[36]
[32]
[45]
[12]

3 Discussion

	Themes	Contributions	ML Task	Ml Model Types	Datasets Used
E	Theme 1				
	Theme 2				
	Theme 3				
S	Theme 1				
	Theme 2				
	Theme 3				
V	Theme 1				
	Theme 2				
	Theme 3				
M	Theme 1				
	Theme 2				
	Theme 3				

Legend: (+) improves the state of the art; (-) comparable to or worse than state of the art; (○) no information on how the approach relates to the state of the art

Table 1: Contributions and ML tasks related to each theme within each RE approach.

Levi ►internal vs external◄

4 Threats to Validity

The validity of the study might be affected by the coverage of the search results, bias on study selection, and inaccuracy of data extraction.

Study Coverage. Some relevant studies could be missing in our study due to inadequate search strings or missing databases. To cope with this threat, the data preparation was based on a systematic method.

Study Selection Bias. Study assessment might be biased by researchers. To mitigate for this threat, a set of include and exclude criteria was predefined and researchers assessed the title and abstract of the papers based on them to steer the assessment.

Inaccuracy of Data Extraction. Also the data extraction process might be biased by researchers. To mitigate for this threat, the selection of data items was strictly driven by the research questions. Moreover, assignments were marked by the researchers depending on their confidence level. Low-confidence assignments were discussed between the authors until a consensus was reached.

5 Conclusion

References

1. Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider. What works better? A study of classifying requirements. *CoRR*, abs/1707.02358, 2017.
2. A. Casamayor, D. Godoy, and M. Campo. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Inf. Softw. Technol.*, 52(4):436–445, Apr. 2010.
3. C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. A recommender system for requirements elicitation in large-scale software projects. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1419–1426, New York, NY, USA, 2009. ACM.
4. J. Cleland-Huang and B. Mobasher. Using data mining and recommender systems to scale up the requirements process. In *Proceedings of the 2Nd International Workshop on Ultra-large-scale Software-intensive Systems*, ULSSIS '08, pages 3–6, New York, NY, USA, 2008. ACM.
5. J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, Apr 2007.
6. J. Coughlan and R. D. Macredie. Effective communication in requirements elicitation: A comparison of methodologies. *Requir. Eng.*, 7(2):47–60, June 2002.
7. A. M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
8. A. Dekhtyar and V. Fong. Re data challenge: Requirements identification with word2vec and tensorflow. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 484–489, Sept 2017.
9. R. Deocadez, R. Harrison, and D. Rodriguez. Automatically classifying requirements from app stores: A preliminary study. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 367–371, Sept 2017.
10. P. Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, 2015.
11. L. Dong, X. Zhang, N. Ye, and X. Wan. Research on user requirements elicitation using text association rule. In *Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on*, pages 357–359. IEEE, 2010.
12. C. Duan and J. Cleland-Huang. A clustering technique for early detection of dominant and recessive cross-cutting concerns. In *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design*, EARLYASPECTS '07, pages 1–, 2007.

13. A. Ferrari, S. Gnesi, and G. Tolomei. Using clustering to improve the structure of natural language requirements documents. In J. Doerr and A. L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, pages 34–49, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
14. F. Garzoli, D. Croce, M. Nardini, F. Ciambra, and R. Basili. Robust requirements analysis in complex systems through machine learning. In A. Moschitti and B. Plank, editors, *Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, pages 44–58, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
15. V. Gervasi and D. Zowghi. Mining requirements links. In D. Berry and X. Franch, editors, *Requirements Engineering: Foundation for Software Quality*, pages 196–201, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
16. O. C. Z. Gotel and C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, Apr 1994.
17. E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20, Sept 2017.
18. J. H. Hayes, W. Li, and M. Rahimi. Weka meets tracelab: Toward convenient classification: Machine learning for requirements engineering problems: A position paper. In *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 9–12, Aug 2014.
19. C. Hollis and T. Bhowmik. Automated support to capture verbal just-in-time requirements in agile development: A practitioner view. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 419–422, Sept 2017.
20. I. Hussain, O. Ormandjieva, and L. Kosseim. Lasr: A tool for large scale annotation of software requirements. In *2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 57–60, Sept 2012.
21. N. Jha and A. Mahmoud. Mining user requirements from application store reviews using frame semantics. In P. Grünbacher and A. Perini, editors, *Requirements Engineering: Foundation for Software Quality*, pages 273–287, Cham, 2017. Springer International Publishing.
22. W. Jiang, H. Ruan, L. Zhang, P. Lew, and J. Jiang. For user-driven software evolution: Requirements elicitation derived from mining online reviews. In V. S. Tseng, T. B. Ho, Z.-H. Zhou, A. L. P. Chen, and H.-Y. Kao, editors, *Advances in Knowledge Discovery and Data Mining*, pages 584–595, Cham, 2014. Springer International Publishing.
23. R. Jindal, R. Malhotra, and A. Jain. Automated classification of security requirements. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2027–2033, Sept 2016.
24. H. Kaiya, Y. Shimizu, H. Yasui, K. Kaijiri, and M. Saeki. Enhancing domain knowledge for requirements elicitation with web mining. In *2010 Asia Pacific Software Engineering Conference*, pages 3–12, Nov 2010.
25. E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens. Supporting requirements engineers in recognising security issues. In D. Berry and X. Franch, editors, *Requirements Engineering: Foundation for Software Quality*, pages 4–18, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
26. G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998.

27. Z. Kurtanović and W. Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 490–495, Sept 2017.
28. D. S. Lange. Text classification and machine learning support for requirements analysis using blogs. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, pages 182–195, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
29. M. Lu and P. 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, New York, NY, USA, 2017. ACM.
30. G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, and M. Saeki, editors, *Conceptual Modeling*, pages 463–478, Cham, 2016. Springer International Publishing.
31. B. Nuseibeh and S. Easterbrook. Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.
32. D. Ott. Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In J. Doerr and A. L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, pages 50–64, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
33. R. Pinquié, P. Véron, F. Segonds, and N. Croué. A collaborative requirement mining framework to support oems. In Y. Luo, editor, *Cooperative Design, Visualization, and Engineering*, pages 105–114, Cham, 2015. Springer International Publishing.
34. R. Pinquié, P. Véron, F. Segonds, and N. Croué. A collaborative requirement mining framework to support oems. In Y. Luo, editor, *Cooperative Design, Visualization, and Engineering*, pages 105–114, Cham, 2015. Springer International Publishing.
35. A. Rashwan. Semantic analysis of functional and non-functional requirements in software requirements specifications. In *Proceedings of the 25th Canadian Conference on Advances in Artificial Intelligence*, Canadian AI'12, pages 388–391, Berlin, Heidelberg, 2012. Springer-Verlag.
36. R. Rauf, M. Antkiewicz, and K. Czarnecki. Logical structure extraction from software requirements documents. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 101–110, Aug 2011.
37. S. Reddivari, Z. Chen, and N. Niu. Recvisu: A tool for clustering-based visual exploration of requirements. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 327–328, Sept 2012.
38. M. Riaz, J. King, J. Slankas, and L. Williams. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 183–192, Aug 2014.
39. J. Slankas and L. 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, May 2013.

40. H. Sultanov and J. H. Hayes. Application of reinforcement learning to requirements engineering: requirements tracing. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 52–61, July 2013.
41. Y. Wang and J. Zhang. Experiment on automatic functional requirements analysis with the efrfs semantic cases. In *2016 International Conference on Progress in Informatics and Computing (PIC)*, pages 636–642, Dec 2016.
42. Y. Wang and J. Zhang. Experiment on automatic functional requirements analysis with the efrf's semantic cases. In *2016 International Conference on Progress in Informatics and Computing (PIC)*, pages 636–642, Dec 2016.
43. M. Wieloch, S. Amornborvornwong, and J. Cleland-Huang. Trace-by-classification: A machine learning approach to generate trace links for frequently occurring software artifacts. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 110–114, May 2013.
44. G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10, Sept 2017.
45. J. Winkler and A. Vogelsang. Automatic classification of requirements based on convolutional neural networks. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 39–45, Sept 2016.
46. P. Zave. Classification of research efforts in requirements engineering. *ACM Comput. Surv.*, 29(4):315–321, Dec. 1997.
47. D. Zowghi and C. Coulin. *Requirements Elicitation: A Survey of Techniques, Approaches, and Tools*, pages 19–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.