Jonas Paul Winkler, Jannis Grönberg, Andreas Vogelsang

# Predicting How to Test Requirements: An Automated Approach

Accepted for 27th IEEE International Requirements Engineering Conference, http://re19.ajou.ac.kr/.

Winkler, Jonas Paul; Grönberg, Jannis; Vogelsang, Andreas (2019): Predicting How to Test Requirements: An Automated Approach. 27th IEEE International Requirements Engineering Conference (RE'19).

# Predicting How to Test Requirements: An Automated Approach

Jonas Winkler, Jannis Grönberg, Andreas Vogelsang
Technische Universität Berlin, Germany
{jonas.winkler, andreas.vogelsang}@tu-berlin.de, jannis.r.groenberg@campus.tu-berlin.de

***Abstract*—[Context] An important task in requirements engineering is to identify and determine how to verify a requirement (e.g., by manual review, testing, or simulation; also called *potential verification method*). This information is required to effectively create test cases and verification plans for requirements. [Objective] In this paper, we propose an automatic approach to classify natural language requirements with respect to their potential verification methods (PVM). [Method] Our approach uses a convolutional neural network architecture to implement a multiclass and multilabel classifier that assigns probabilities to a predefined set of six possible verification methods, which we derived from an industrial guideline. Additionally, we implemented a backtracing approach to analyze and visualize the reasons for the network's decisions. [Results] In a 10-fold cross validation on a set of about 27,000 industrial requirements, our approach achieved a macro averaged $F_1$ score of 0.79 across all labels. For the classification into test or non-test, the approach achieves an even higher $F_1$ score of 0.94. [Conclusions] The results show that our approach might help to increase the quality of requirements specifications with respect to the PVM attribute and guide engineers in effectively deriving test cases and verification plans.**

***Index Terms*—Requirements Engineering, Requirements Validation, Test Engineering, Machine Learning, Natural Language Processing, Neural Networks**

## I. Introduction

Verifiability is a quality characteristic for requirements that is mentioned in many normative quality standards such as ISO 29148 [1]. The IREB glossary defines *verifiability* (of requirements) as "The degree to which the fulfillment of a requirement by an implemented system can be checked, e.g., by defining acceptance test cases, measurements or inspection procedures." [2]. The standards argue that a requirement should be verifiable because it is pointless to specify requirements for which it is impossible to detect whether they are implemented correctly.

The first step towards assessing the verifiability of a requirement is to identify a potential verification method for a requirement. There are many viable methods to verify a requirement ranging from ad-hoc manual inspections over (automated) tests to formal analysis and process audits. The selection of a verification method has an effect on the verifiability of a requirement because verification methods have diverging demands on how requirements must be formulated in order to be verified. In addition, the selected potential verification methods may influence how the requirements are processed in the existing development process (cf. [3]).

One of our industry partners has therefore introduced an explicit requirements attribute called *Potential Verification Method (PVM)* that specifies in which ways a requirement must be verified. This attribute is the basis for several reporting activities and also a subject in reviews. Setting values for this attribute is a manual, time-consuming, and error-prone task. From our experience with requirements specifications of our industry partner, the PVM attribute is often set incorrectly or even neglected.

In this paper, we propose an automatic approach to classify natural language requirements with respect to their potential verification methods. The approach can be used to recommend PVMs for unlabeled requirements or to check labeled requirements for potential errors. Our method uses a convolutional neural network architecture to implement a multilabel and multiclass classifier that assigns probabilities to a predefined set of six possible PVM values derived from the guideline of our industry partner. We trained and validated the classifier on a set of about 27,000 requirements from our industry partner. In a 10-fold cross validation, we achieved $F_1$ scores between 0.65 and 0.97 for the different labels. According to our industry partner, it would be ideal to precisely predict all six labels. However, being able to discriminate between requirements that should be tested and requirements that should be verified by other means would also be very helpful. For this specific distinction, we evaluated a binary classifier, resulting in an $F_1$ score of 0.94.

To support requirements engineers in using our approach, we implemented a backtracing algorithm [4] that visualizes the reasons for the network's decisions on individual samples.

We conclude that our automated approach helps to increase the quality of requirements by detecting misclassified PVM attributes or automatically generating classification proposals for unlabeled requirements. Additionally, the analysis of the trained model provides insights into how requirements with specific verification methods are typically formulated. This resembles a new view on syntactic characteristics of *verifiable* requirements.

The remainder of the paper is structured as follows. Section II introduces the potential verification method attribute and describes its possible values. A quick introduction on convolutional neural networks is given in Section III. In Section IV, we elaborate on how we use neural networks to classify requirements regarding the PVM attribute. The evaluation using standard measures such as recall and precision is provided in Section V. Section VI discusses the results. That section also provides insights into how the classifier works and what its limitations are. Section VII points out related work. Section VIII concludes.

## II. The Potential Verification Method Attribute as Used by Our Industry Partner

One of our industry partners from the automotive domain decided to explicitly document the potential verification method for all requirements as a requirements attribute. A requirements engineer annotates requirements with a list of methods that he or she thinks can be used to verify the requirement. To assemble this list, the requirements engineer can choose from a set of six values that are listed in the following. The provided example requirements are taken from our dataset.

- **Review.** Manual inspection of development artifacts by experts (including code, engineering drawings, architectures). *Example: Exception handling may be turned on or off by setting a parameter.*
- **Simulation/Analysis.** Calculation of system properties by computational methods executed on development artifacts (including simulation, code analysis). *Example: The relay control output shall be controlled within the operating voltage range specified in the input/output table.*
- **Formal Verification.** Verification of system properties by means of mathematical methods. *Example: Consistency between type A memory and type B memory has to be ensured.*
- **Process audit.** Verification of process requirements by means of a process assessment. *Example: The contractor shall provide a project manager for this project who coordinates and supervises the various processes regarding the contractor.*
- **Test.** Verification of system properties by using the implemented system with the goal to verify all system properties. *Example: The signal of the button must be sent from the roof control unit to the component via the CAN bus.*
- **Production control.** Verification of system properties by assessing the production process. *Example: The weight of the component shall not exceed 1 kg.*

A requirements engineer can select more than one value from the list, which means that multiple verification methods are possible for a single given requirement. Specifying more than one verification method implies that a requirement may be verified using different methods. However, only one method is used to verify each requirement. The method that is actually employed is selected later in the requirements engineering process.

The company's requirements specification guideline states: "The purpose of the PVM attribute is to indicate relevant requirements for the test specification. The attribute defines potential verification methods." The guideline also contains a hint stating that the values within the PVM attribute are only recommendations. The actually performed verification activities may differ from the PVM values.

The PVM attribute is subject to manual review of the requirements specification. The reviewers check whether the specified PVM values fit the specified requirement. Furthermore, the reviewers must assure that the PVM attribute is set for all safety relevant requirements. For requirements that are not safety relevant, setting the PVM attribute is not mandatory but highly recommended. In early stages of the requirements engineering process, a PVM attribute with multiple values may indicate that the requirement is not yet specific enough and contains several aspects that are verified by different verification methods.

The specified values of the PVM attribute have an influence on several reporting activities. For example, the company's traceability report contains information about requirements and related test cases. However, this is only reported for requirements where the PVM attribute contains the value *Test*. When test case specifications are derived from requirements, a test engineer usually starts by filtering the requirements with respect to the PVM attribute.

## III. Convolutional Neural Networks for NLP

Predicting the proposed PVM attribute of requirements within requirements specifications is a multiclass-multilabel classification problem because each element may have multiple labels from a set of predefined classes. Within the natural language processing community, many popular techniques exist to solve such a problem, including Naive Bayes [5] and support vector machines [6]. Although these techniques have limitations, such as ignoring word order, they proved to be good enough for classification tasks such as sentiment analysis [7] or authorship attribution [8].

Convolutional neural networks (CNN) are a variation of classic feed-forward neural networks. CNNs are widely used within the image recognition community [9] but gained attention in natural language processing as well [10], [11]. These networks have several advantages compared to other classification techniques. They keep word order intact to be used for finding patterns, whereas other approaches often use *Bag of Words* techniques and information about word order is lost. Furthermore, due to the use of word embeddings, they are able to detect patterns even if words vary slightly.

The organization and functionality of CNNs as applied in this paper is illustrated in Fig. 1 and will be described briefly in the following section. CNNs for text classification in general are described in detail by Zhang and Wallace [12].

The first step is to transform an input sentence into a vector representation (1). This is called word embedding. We use word2vec [13] for this step. Word2vec maps a single word to a vector $v \in \mathbb{R}^n$, where $n$ is called the *embedding size*. One remarkable property of word2vec is that the vector distance of two given words is small if these two words are used in similar contexts whereas it is large if the words are not related at all. Sentences are transformed into a matrix $m \in \mathbb{R}^{n,l}$, where $l$ is the number of words in the sentence.

The first layer in the network applies a set of filters (2) to the sentence matrix $m$. Each filter is a matrix $f \in \mathbb{R}^{n,o}$ of

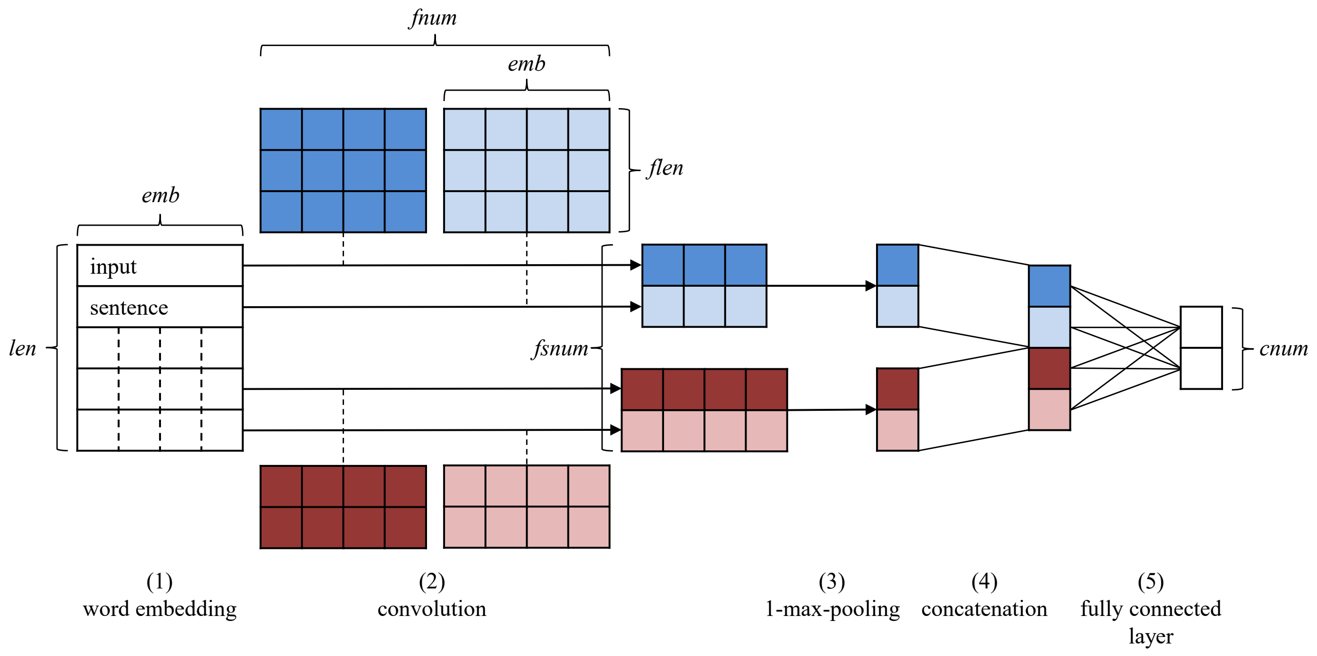

Fig. 1: Convolutional neural network architecture (simplified) as proposed by [12]

trainable parameters, where $n$ is the embedding size and $o$ is the length of that particular filter. Number and sizes of the filters are hyper parameters and as such manually defined prior to training. In Fig. 1, two filters of length 3 and two filters of length 2 are illustrated. Filters are applied to a sentence matrix by moving them as a sliding window over the sentence matrix, producing a single value at each position using an activation function such as a rectifier or sigmoid function (2). This step is called convolution. Each filter learns to recognize a specific word pattern (e.g., a filter of size 2 might learn to recognize the pattern “function must”).

All values produced by a filter are reduced to a single value by applying 1-max-pooling (3). The max-pooled values indicate whether the pattern learned by a filter is present within a sentence. All resulting values are concatenated and form a feature vector (4). This vector is connected to the output layer using a standard fully connected layer and an appropriate set of trainable parameters (5). The fully connected layer is used to associate certain patterns with an output class. The sigmoid function is applied to each output to create probabilities between 0 and 1 for each class. Whenever this probability is above a predefined threshold (usually 0.5), the label corresponding to the output is assigned to the input example.

## IV. Approach

To build a classifier that is able to assign potential verification methods to a requirement, we followed the Knowledge Discovery in Databases (KDD) process presented in [14]. The process consists of nine steps that help creating reasonable knowledge from raw data using data mining techniques, while pointing out what risks to be aware of.

1) **Developing an understanding of the application domain.** We thoroughly analyzed how the *potential verification method* is used at our industry partner (see Section II).
2) **Selecting and creating a dataset on which discovery will be performed.** The analysis helped us to create a dataset that reflects our understanding of the domain. This will be discussed in Section IV-A.

3) **Preprocessing and cleansing.** We applied a few preprocessing steps in order to minimize error sensitivity. This will also be discussed in Section IV-A.
4) **Data transformation.** We used the word2vec [13] word embedding technique. This transforms the data into word vectors usable by neural networks. Word2vec word embeddings retain the context as well as the relationships between words.
5) **Choosing the appropriate data mining task.** Choosing one or more potential verification methods for a requirement is a multiclass and multilabel classification task.
6) **Choosing the data mining algorithm.** We used CNNs as they are straightforward to train and easy to explain by using back tracing approaches [4]. Another key reason is that CNNs perform very well on related tasks [15], [11].
7) **Employing the data mining algorithm.** Details on how to build and train the classifier will be presented in Section IV-B.
8) **Evaluation.** We evaluated the results with standard metrics such as precision and recall. This is described in Section V.
9) **Using the discovered knowledge.** This will be discussed in Section VI. Possible applications and benefits are specifically discussed in Section VI-B.

### *A. Creating the Dataset*

To build our dataset, we collected a large number of requirements specification documents available at our industry partner, containing many real-world requirements. The requirements specifications were available in multiple languages, some in English and some in German. We focused on the language in which the majority of the specifications were written in (German). Furthermore, the requirements originated from two types of requirements specifications:

- *System requirements specifications* contain requirements that are used for the development of a vehicle system (e.g., airbag system).
- *Component requirements specifications* contain requirements that are used for the development of individual components (e.g., control units, sensors, actuators, etc).

Documents of both types usually include *default requirements*. Default requirements are predefined in specification templates and define requirements independent of any specific component or system, such as legal and process requirements.

We wanted to recreate the real usage of the PVM attribute as accurately as possible. In order to do so, we selected requirements specifications regardless of the number of elements with PVM attributes or the PVM values they held. Moreover, we did not exclude default requirements but included only one instance of each in the dataset.

We excluded all requirements that did not contain any of the six previously presented PVM values (e.g., requirements that were marked as “to be done” and requirements on which no PVM had been specified at all). In addition, we excluded requirements that contained only one word or contained only numbers or special characters because these requirements did not contain relevant information.

TABLE I: Dataset for multilabel/multiclass classification task

| **PVM value** | **Number of reqs with label** | **Ratio** |
|---|---|---|
| Review | 3,483 | 12.91% |
| Simulation/Analysis | 646 | 2.40% |
| Formal verification | 832 | 3.09% |
| Process audit | 788 | 2.92% |
| Test | 23,529 | 87.25% |
| Production control | 289 | 1.07% |
| **Total number of requirements** | 26,966 | |
| **Label cardinality**[1] | 1.096 | |
| **Label density** | 0.183 | |

Th retain as much information of the input requirement as possible. Therefore, we decided to neither remove stop words nor to use a stemming method. Consequently, no information about the order of words, the dependency of words or information about the conjugation of words was lost.

Last but not least, we eliminated duplicate requirements occurring in more than one specification. The resulting dataset consists of 26,966 requirements that specify automotive software systems from the interior of a car. These requirements are written in natural language and do not follow any specific form of writing. An overview of the distribution of examples with respect to the different labels is provided in Table I.

About 87% of all requirements carry the PVM value *Test*. The least frequently specified PVM value is *Production control*, which is only associated with a little more than 1% of all requirements. The label cardinality shows that 1.096 labels are attached to each requirement on average. The label density (*label cardinality divided by number of labels*) of 0.183 indicates that a requirement covers 18.3% of all labels on average.

The labels are not independent from each other. Some labels have a higher chance to be selected together with certain other labels. Table II provides an overview of the probability of occurrence of the labels, depending on the occurrence of another label. For example, 95% of requirements with the PVM value *Simulation/Analysis* also have the PVM value *Test*. A possible explanation is that simulation software such as MATLAB Simulink is well suited for virtual testing and that simulation regularly goes hand in hand with virtual tests. Due to that, both PVM values are set.

The PVM values *Process Audit* and *Production Control* are very often accompanied by the PVM value *Review*. In both cases, an assessment by an expert could be essential. If this is required, a *Review* should be mandatory.

Another observation can be taken from the last row: The PVM value *Formal Verification* is frequently set with additional PVM values. Usually, *Formal Verification* requires a lot of resources. Sometimes, a *Test*, a *Review*, or a *Process Audit*

[1]*Label cardinality* is the average number of labels per example in the set. *Label density* is the number of labels per sample divided by the total number of labels, averaged over the samples. [16]

TABLE II: Label dependencies

| then / if | Simulation/ Analysis | Test | Review | Production Control | Process Audit | Formal Verification |
|---|---|---|---|---|---|---|
| **Simulation/Analysis** | - | 0.9505 | 0.0294 | 0.0000 | 0.0000 | 0.0046 |
| **Test** | 0.0261 | - | 0.0233 | 0.0003 | 0.0145 | 0.0142 |
| **Review** | 0.0055 | 0.1573 | - | 0.0787 | 0.1947 | 0.1270 |
| **Production Control** | 0.0000 | 0.0277 | 0.9481 | - | 0.1073 | 0.2491 |
| **Process Audit** | 0.0000 | 0.4327 | 0.8604 | 0.0393 | - | 0.4264 |
| **Formal Verification** | 0.0036 | 0.4014 | 0.5313 | 0.0865 | 0.4038 | - |

beforehand is inevitable in order to reduce time and costs during the formal verification process.

Ultimately, the label dependencies show that sometimes a single potential verification method is not enough to verify a requirement or that it is possible to choose a different one. This is more often the case for more detailed verification methods. On the contrary, *Test* as a single requirement verification method is sufficient in more than 99% of the cases where it is used.

## B. Building and Training the Classifier

Our classifier takes a requirement and assigns labels to it, which represent the different PVM values. There are six different PVM values and since more than one PVM value can be assigned to a requirement, our classifier has to perform a multiclass and multilabel classification. This is achieved by having 6 different output values at the end of the CNN and using `sigmoid` as the activation function.

In order to counteract the problem of class imbalance as seen in Table I, we weighted the different labels [17]. We determined their individual number of occurrences in relation to the number of occurrences of the most frequent label *Test*. Thus, we determined weight factors for our six PVM values. These *class weights* were used to give more credit to underrepresented classes during training.

Additional parameters that we had to determine include the *number of epochs*, which indicates how often the classifier is trained on all examples in the dataset. Each epoch is divided into multiple iterations. The number of samples that the classifier processes in each iteration is defined by the *batch size*. We chose these parameters empirically to minimize both training time and the negative effects on gradient computation due to too large batches. In addition, *early stopping* was used to save time.

Furthermore, we had to choose additional parameters of the network. The *embedding size* parameter defines how many dimensions are used to represent words using word2vec. Using more dimensions results in a more fine-grained model, which captures more semantic information from the dataset. Also, we may choose to use static embeddings (constant pre-trained vectors for each word). Non-static embeddings are handled as weights during training and may be changed to better fit the classification task.

The *filter sizes* parameter defines the shape of the filters. Using filters of varying length allows the network to learn word patterns of different lengths. For each filter size, the *count per size* parameter defines how many filters are used. Using more filters allows the network to learn more word patterns in parallel, which usually increases the performance.

The word embedding used for the classifier was created based on all available requirements specifications and was specifically trained for use with our classifier. We opted against using a public pre-trained model since these models do not contain many of the domain-specific words used in the specifications.

We trained and evaluated our classifier using 10-fold cross validation. The folds were produced by first shuffling the training data and then creating 10 parts of equal size. Shuffling ensures that the distribution of all labels across the folds is roughly the same. We considered using stratification as well. however, this prove to be difficult due to the multilabel setting [18].

TABLE III: Dataset for binary classification task

| PVM value | Number of reqs with label | Ratio |
|---|---|---|
| Test | 23,529 | 87.25% |
| No Test | 3,437 | 12.75% |
| **Total number of requirements** | 26,966 | |

## C. Building a Binary Test/No-Test Classifier

In addition to the classifier as described in the previous subsection, we also constructed a binary classifier discriminating between Test and other verification methods (*No Test*). The reason is that our industry partner told us that, in fact, the distinction between requirements that are verified by testing and requirements verified by other means is most important to them because subsequent development steps are fundamentally different.

The labels in the dataset were modified using the following method: We discarded all labels except *Test*. For all requirements without any label, we assigned the new *No Test* label. Table III shows an overview of the modified dataset. The classifier was modified slightly to fit the altered scenario: The final layer now has 2 output neurons and uses `softmax` as its activation function.

The remainder of the procedure is largely equal to the procedure used for the multilabel and multiclass classifier: We determined hyper-parameters manually, used class weights to counteract the problem of class imbalance, and used 10-fold cross validation to assess the performance of the model.

## V. Evaluation

In order to evaluate our model using standard evaluation metrics such as precision, recall, and $F_1$ score, we have created a baseline using the ZeroR classifier. This classifier simply assigns the most frequent class in the dataset to all input examples without inspecting any features of the input examples. Any other classifier should perform better than ZeroR.

For all configurations and the baseline acquired using the ZeroR classifier, the following measures are compared:

- **Accuracy.** Multilabel accuracy on the test splits.
- **Perfect Match Ratio.** This score is the percentage of test examples on which all classes were predicted correctly. This is similar to accuracy in a non-multilabel classification task.
- **Macro-F1.** Macro-Averaged $F_1$ score. This score is particularly useful, since it is more sensitive to the performance of the classifier on the underrepresented classes.
- **Micro-F1.** Micro-Averaged $F_1$ score.

The results for the baseline as well as for different configurations of the CNN can be found in Table IV. The ZeroR baseline has high values for *Accuracy*, *Exact Match Ratio*, and *Micro-F1* due to the fact that the class *Test* is the most frequent class in our dataset.

We used grid search to test various network configurations and search for the set of network parameters that works best for our dataset, also shown in Table IV.

We used various combinations of embedding sizes, filter sizes, and filter count, and also allowed the classifier to fine-tune the embedding parameters (non-static embeddings). We were able to achieve better results by increasing the number of filters per size and the embedding size, by using filters of more sizes and by allowing the network to also change the embedding parameters. The number of filters per size affects the network performance most, since more filters allow the network to detect more word patterns. Increasing the size of the word embeddings from 32 to 128 dimensions also results in consistent improvements. Allowing the network to also change word embeddings parameters increases the performance when the word embedding has few dimensions (32). Gains in network performance vary when larger word embeddings are used. Adding filters with length 4 does not affect the performance at all.

### A. Evaluation of the Multilabel/Multiclass Classifier

To further assess the performance of the network on individual classes, we selected the configuration with the highest macro-F1 score (word embedding layer: 256 dimensions, non-static embeddings, filters of length 2 and 3, 64 each), since this score is a good indicator for network performance averaged over all classes. Table V shows detailed results for this configuration.

We are able to achieve 97% recall and precision for the *Test* label. These results show that our classifier performs well in discerning requirements validated using a test on the actual system versus requirements validated using other methods.

Considering that our dataset probably contains some incorrectly classified items due to how the data was collected,

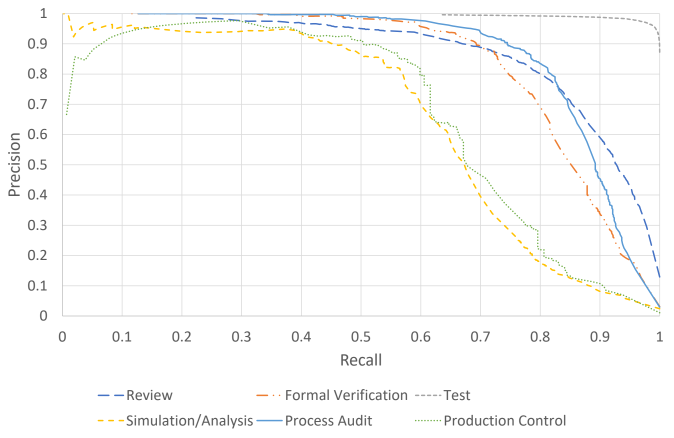

Fig. 2: Recall-precision graph

our classifier performs reasonably well on the *Review* and *Process audit* classes, yielding an $F_1$ score of 80.9% and 82.4% respectively. The $F_1$ score on the remaining classes (*Simulation/Analysis*, *Formal verification*, *Production control*) is comparatively low. Recall on *Simulation/analysis* and *Production control* is especially low: The classifier essentially misses to assign these labels for almost every second requirement.

Since missing requirements of a particular label is worse than assigning too many requirements to that label, tuning the classifier for recall at the expense of precision may increase its usefulness. In order to assess the losses in precision, we created recall-precision graphs for all classes by choosing different thresholds for the network output (see Fig. 2). By increasing recall to 90%, precision drops to roughly 60% to 10%, depending on the label.

### B. Evaluation of the Binary Classifier

Our binary classifier performs well in discerning requirements that require tests from requirements that do not require tests. The results are displayed in Table VI. Precision on the Test class is close to 100%, meaning that almost all requirements classified as Test actually do require a test. Recall is reasonably good as well on both classes. The low precision on the No Test class (83%) indicates that the classifier favors the No Test class over the Test class for ambiguous requirements. Depending on the application, this may be an issue and can be adjusted using a different threshold.

### C. Evaluation with Document-based Cross Validation

When applying the classifier in a real world scenario, it will be trained on complete documents and applied to completely unseen documents. However, in a 10-fold cross validation with shuffling, certain requirements of a single given document will end up in a training set whereas the remaining requirements will end up in the test set. To make an evaluation that is closer to the potential application scenario, we also employed a document-based cross validation in addition to the 10-fold cross validation.

For this validation, the folds were constructed as follows. We used as many folds as we had documents available. For

TABLE IV: Results for the multiclass/multilabel classifier with different configurations

| **Embeddings** |  | **Filters** |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| **Size** | **Static** | **Sizes** | **Count per size** | Accuracy | Perfect Match Ratio | Macro-$F_1$ | Micro-$F_1$ |
| **ZeroR baseline** |  |  |  | 0.8477 | 0.8293 | 0.1553 | 0.8323 |
| 32 | yes | 2, 3 | 8 | 0.8979 | 0.8682 | 0.5818 | 0.9003 |
| 32 | yes | 2, 3 | 64 | 0.9099 | 0.8824 | 0.7131 | 0.9153 |
| 32 | yes | 2, 3, 4 | 8 | 0.9016 | 0.8718 | 0.6119 | 0.9042 |
| 32 | yes | 2, 3, 4 | 64 | 0.9177 | 0.8930 | 0.7331 | 0.9226 |
| 32 | no | 2, 3 | 8 | 0.9130 | 0.8856 | 0.7127 | 0.9177 |
| 32 | no | 2, 3 | 64 | 0.9266 | 0.9044 | 0.7644 | 0.9316 |
| 32 | no | 2, 3, 4 | 8 | 0.9140 | 0.8872 | 0.7179 | 0.9177 |
| 32 | no | 2, 3, 4 | 64 | 0.9275 | 0.9070 | 0.7706 | 0.9331 |
| 128 | yes | 2, 3 | 8 | 0.9057 | 0.8766 | 0.6705 | 0.9098 |
| 128 | yes | 2, 3 | 64 | 0.9234 | 0.9019 | 0.7641 | 0.9286 |
| 128 | yes | 2, 3, 4 | 8 | 0.9055 | 0.8768 | 0.6954 | 0.9104 |
| 128 | yes | 2, 3, 4 | 64 | 0.9249 | 0.9048 | 0.7696 | 0.9314 |
| 128 | no | 2, 3 | 8 | 0.9220 | 0.8986 | 0.7523 | 0.9265 |
| 128 | no | 2, 3 | 64 | **0.9313** | **0.9122** | 0.7809 | 0.9361 |
| 128 | no | 2, 3, 4 | 8 | 0.9260 | 0.9049 | 0.7623 | 0.9307 |
| 128 | no | 2, 3, 4 | 64 | 0.9304 | 0.9119 | 0.7549 | 0.9360 |
| 256 | yes | 2, 3 | 8 | 0.9082 | 0.8800 | 0.6948 | 0.9121 |
| 256 | yes | 2, 3 | 64 | 0.9274 | 0.9067 | 0.7753 | 0.9332 |
| 256 | yes | 2, 3, 4 | 8 | 0.9130 | 0.8876 | 0.7157 | 0.9173 |
| 256 | yes | 2, 3, 4 | 64 | 0.9270 | 0.9068 | 0.7707 | 0.9329 |
| 256 | no | 2, 3 | 8 | 0.9224 | 0.9002 | 0.7518 | 0.9281 |
| 256 | no | 2, 3 | 64 | 0.9310 | 0.9115 | **0.7904** | **0.9368** |
| 256 | no | 2, 3, 4 | 8 | 0.9251 | 0.9043 | 0.7671 | 0.9307 |
| 256 | no | 2, 3, 4 | 64 | 0.9261 | 0.8995 | 0.6960 | 0.9308 |

TABLE V: Results for multiclass/multilabel classifier

| **PVM value** | **Support** | **Precision** | **Recall** | **$F_1$** |
|---|---|---|---|---|
| Review | 3,483 | 0.818 | 0.800 | 0.809 |
| Simulation/Analysis | 646 | 0.755 | 0.563 | 0.645 |
| Formal verification | 832 | 0.910 | 0.692 | 0.786 |
| Process audit | 788 | 0.865 | 0.787 | 0.824 |
| Test | 23,529 | 0.970 | 0.977 | 0.973 |
| Production control | 289 | 0.856 | 0.599 | 0.705 |
| **Micro-Averaged** |  | 0.944 | 0.930 | 0.937 |
| **Macro-Averaged** |  | 0.862 | 0.736 | 0.790 |

TABLE VI: Results for test/no-test binary classifier

| **PVM value** | **Support** | **Precision** | **Recall** | **$F_1$** |
|---|---|---|---|---|
| Test (ZeroR) | 23,529 | 0.872 | 1 | 0.932 |
| No Test (ZeroR) | 3,437 | undefined | 0 | undefined |
| Test | 23,529 | 0.997 | 0.969 | 0.983 |
| No Test | 3,437 | 0.833 | 0.981 | 0.901 |
| **Micro-Averaged** |  | 0.975 | 0.971 | 0.972 |
| **Macro-Averaged** |  | 0.916 | 0.975 | 0.942 |

TABLE VII: Results for the binary classifier with document-based cross validation

| **PVM value** | **Support** | **Precision** | **Recall** | **$F_1$** |
|---|---|---|---|---|
| Test | 23,529 | 0.948 | 0.962 | 0.955 |
| No Test | 3,437 | 0.437 | 0.366 | 0.399 |
| **Micro-Averaged** |  | 0.915 | 0.916 | 0.916 |
| **Macro-Averaged** |  | 0.692 | 0.664 | 0.677 |

each fold, we used one document for testing and all other documents for training. This way, we can ensure that each document will be used for testing exactly once and that the requirements of a single document will be either used entirely for training, or entirely for testing.

However, this approach also introduces issues regarding the evaluation: Only very few documents contain the majority of certain underrepresented labels. By using this cross validation approach, we cannot ensure that the samples of these underrepresented labels will be appropriately distributed among training and test sets.

Table VII shows the performance of our binary classifier with the document-based cross validation. The performance of the classifier drops significantly for both classes. We assume this effect is caused by the diversity in styles introduced by varying authorship. Requirements originating from one document may be written in a similar manner, whereas requirements originating from different documents may be quite different in terms of writing and terminology. Additionally documents may contain domain specific words which are exclusively used in that particular document. Therefore, detecting the correct label in completely unseen documents is very challenging.

We also considered applying the document-based cross validation to the multilabel/multiclass classifier. However, the examples of the underrepresented classes were distributed very unevenly among all documents (i.e., a very small fraction of documents contained almost all examples labeled with *Production Control*). This means that either almost all examples are used for training and none for testing, or vice versa, leading to $F_1$ scores close to 0 on multiple classes.

## VI. Discussion

In this section, we discuss the results presented in Section V and present possible applications of our approach.

### A. What Words Say About Verification

Winkler and Vogelsang have proposed a technique that allows tracing back decisions made by convolutional neural networks for text classification and visualizing them on the input sentences [4]. For any trained model, this technique reveals the reasons (i.e., important word structures in an input example) due to which the classifier chose the assigned labels for an example. The approach essentially reverses the operations performed by a neural network and creates a *Document Influence Matrix*, indicating which inputs in a particular input example were especially significant. This technique may be used to create ranked lists of particularly important words for each class as well as visualizations of individual input examples in which important words are highlighted.

Table VIII displays example sentences from each class. The highlighting shows individual words that are especially important for the classification process in this particular sentence. The sentences were translated from German to English. This has only been done for a better understanding in this paper and we tried to keep them as close as possible to their respective German original. Please note that the importance of single words is not universal but differs between examples depending on the context (e.g., the word "be" has a high influence value in the first example because it is used in a context "must be activated", whereas in the sixth example "must be agreed", the word "be" is not as relevant for the classification). Since these visualizations were created based on what the classifier learned, their expressiveness largely depends on the accuracy of our classifier.

We examined several examples in our dataset and also created a list of words with the highest average influence values for each class. Based on that, we were able to characterize the different classes with respect to certain words and phrases.

*Simulation/Analysis* is often used for requirements that either define functionality depending on the state of the ignition lock, specify precise values (i.e., voltage, dimensions, time, etc.), or deal with functionality of other components (names of other components have a high influence value).

*Process Audit* is a verification method used to verify requirements describing engineering processes rather than system properties. These requirements usually define what rights and liabilities a *contractor* and *client* have, what they *shall* do, and what *requirements* they need to fulfill. Consequently, the words emphasized in the last sentence have a high impact value.

*Production Control* is used for requirements defining how certain parts will be produced. As such, requirements labeled with this label usually contain words regarding parts and components, time constraints for production, or physical properties of the component.

*Test* is the class used for almost all other requirements at our industry partner. These are functional requirements of the respective system or component and are verified using various test methods involving the system or component itself (i.e., hardware in the loop, integration test, etc.) Words such as "will" or "shall" that are typically used to formulate functional requirements have high influence values for this class.

*Review* is used whenever no test is required to verify the requirement. Rather than performing a test, the fulfillment of these requirements is verified manually. When comparing examples from this class and the *Process Audit* class, it is worth noting that similar words are highlighted as important (e.g., *contractor*). This is due to the fact that whenever *Processs Audit* is used for verifying a requirement, *Review* is used very often as well (cf. Table II).

We were unable to draw any conclusions about requirements verified using the method *Formal Verification*. We expected that this method is used whenever certain safety-critical requirements are specified, however this was not the case. Requirements labeled with *Formal Verification* were almost indistinguishable from requirements labeled with *Test*.

### B. Application of the Approach

In previously published works [15], we used the same techniques as presented in this paper to solve a similar classification problem. We used CNNs to separate requirements from non-requirement content elements in requirements specifications (i.e., additional information such as examples, references, and explanations). Similar to the PVM attribute, requirements engineers are expected to manually set an attribute called *Object Type* to either *information* or *requirement*.

We built a tool using a trained CNN model that displays warnings where the actual Object Type differs from the predicted Object Type. In a preliminary case study [19], we showed that such a tool provides several benefits, such as increased rate of error detection, decreased review time and decreased error introduction rate. These benefits may also be transferred to the problem described in this paper.

A tool that includes a model trained to classify requirements regarding their PVM attribute may provide the following benefits over manual inspection of the PVM attribute:

- Users of the tool may be able to easily identify requirements within a requirements specification where the defined potential verification method differs from what is normally used in similar requirements, because requirements with warnings are visually highlighted.
- As a side effect, users may also be able to identify requirements where a potential verification method has not yet been set.
- Users of such a tool might be able to find and fix more errors regarding the PVM attribute than users without the tool on the same document due to the issued warnings. Thus, using the tool can help to increase the quality of requirements specifications regarding the PVM attribute.
- Since the tool shifts the attention of the users towards elements with warnings, the users are less inclined

TABLE VIII: Visualizing the importance of parts of the input sentence for the classification decision.

| **Class** | **Example Sentences** |
|---|---|
| Test | The actuators and switches must be activated separately within the control unit . |
| Test | The function is triggered whenever a definable rain intensity is detected for a definable time . |
| Test | The function must be initiated immediately if the switch is pressed and released within 100 ms . |
| Review | The contractor must agree on a suitable test platform and test plan with the department . |
| Review | When mesh is used in front of the ultrasonic sensors the compliance with world-wide legal and insurance requirements has to be ensured |
| Review | All can signals must always carry meaningful values and must be agreed on with the contractor . |
| Production control | All materials and processes must comply with current legal regulations regarding regulated substances and reusability . |
| Production control | The weight of the remote control in operational state must be between 100 and 200 g . |
| Formal verification | Consistency between nvram and noinitram has to be ensured . |
| Simulation/Analysis | In order to ensure availability of the bus systems , the bus systems must never be impaired or even blocked . |
| Simulation/Analysis | Since an electronic horn has a down time of aproximately 50 ms after initiation , the horn must be triggered for at least 100 ms . |
| Process audit | The contractor must fulfill the requirements regarding logistics according to the description in the separate logistics specifications . |
| Process audit | The contractor must consider and demonstrate weight-reducing measures during the entire development cycle . |

to change the PVM attribute of requirements without warnings. Therefore, less errors might be introduced when the tool is used.

- Since the tool issues warnings only on a fraction of all requirements within a requirements specification and the users focus on elements with warnings, they might be able to complete the review of a specification in shorter time than without the tool.
- Due to using a tool that assists requirements engineering practitioners in their tasks, they might be more motivated to fix issues within their documents.

A separate case study has to be performed in order to prove these claims. However, since we have already performed a case study on a similar classification problem, we may get similar results.

As stated in the case study on requirements and non-requirements in [19], using a tool also carries certain limitations and risks. Since 100% accuracy on the classification task at hand is not achievable, such a tool will never help requirements engineers to achieve perfect requirements specification documents regarding the PVM attribute.

Furthermore, since the focus of users is shifted towards requirements with visual warnings, users tend to inspect requirements *without* warnings less carefully and are thus more likely to miss defects in these requirements. This was shown in the case study as well: Users without the tool found and fixed more *unwarned defects* than users with the tool. However, the number of missed defects compared to defects fixed was still small and the authors were able to achieve improvements by introducing the tool.

As these benefits and limitations have only been proven for the task of discerning requirements from non-requirements, further experiments have to be conducted to determine whether and which benefits transfer to the PVM classification task, especially since the usefulness of such a tool will largely depend on the accuracy of the underlying classifier.

## C. *Limitations of the Approach*

In this section, we will outline the various limitations of our approach.

The classifier was trained to discern between six different types of requirements verification methods that we extracted from guidelines at our industry partner. Other companies may use a different set of verification methods. Also, even if the same set of potential verification methods is used, other companies' processes regarding requirements verification may be different from the processes practiced at our industry partner. Therefore, the trained model might not be directly applicable. To counteract this issue, the model needs to be trained again using company-specific data.

As shown in Section VI-A, the model learned to recognize certain company-specific words and uses these words to decide between the output classes. These words are present in the dictionary of the embedding layer, since the word embedding was also created from documents at the same company. However, other companies might use different terminology not present in the dictionary, severely hindering the performance of the model. As with the previous limitation, the model might need to be trained again.

The trained model may as well be domain-specific since it has been trained on data acquired from a company in the automotive domain. In other domains, such as rail and aerospace, requirements may be verified using different techniques (e.g., formal verification is used much more in the aerospace domain compared to the automotive domain).

As already discussed in the previous section, achieving 100% accuracy is impossible and therefore, using our approach will not lead to perfect requirements specification documents regarding the PVM attribute.

Furthermore, after training a model and deploying it in a tool, the weights and vocabulary of the model are usually fixed. Should the company alter their methods and guidelines for writing requirements or introduce new potential verification methods, the model will not adapt to these changes and its usefulness will decrease. To counter this issue, active learning approaches (see [20]) have to be implemented. Active Learning will allow users to gradually adapt the model by correcting incorrect predictions. However, this is not an easy task due to *catastrophic forgetting*: when retraining neural networks with different input data, they tend to forget information about previously learned examples by using important weights for learning new information [21].

### D. Threats to Validity

In this section, we describe the threats to the validity of our evaluation.

The most severe threat is the way we collected the data for training and evaluation. We used requirements specifications available at our industry partner that have already gone through several iterations of quality checks. However, these documents may still contain misclassified items. Due to the large amount of data required to train neural networks, we were unable to manually check the correct labeling of all examples in the dataset.

A related issue arises since many different people from multiple departments worked on the requirements specifications used to create our dataset. Despite company-wide guidelines on how requirements should be verified, there might be differences between departments. This may result in similar requirements in our dataset being labeled with different potential verification methods.

A label cardinality of 1.096 shows that a majority of the requirements only have one specified PVM value. This may indicate that the addressed problem is, in fact, not a multilabel classification problem. Our industry partner confirmed that the company's guideline allows selecting multiple PVM values, however, they also confirmed that they have a closer look at requirements with more than one PVM value because this may indicate that the requirement is not yet specific enough and covers multiple aspects. Therefore, it is possible that solving the classification problem with a single-label classifier that only suggests the most promising label may be perceived as more useful by the engineers.

Overfitting may also be an issue. As seen in Section V, using more network parameters tends to yield better results. However, increasing the network size too much results in only minor improvements. At that point, the network may learn specifics about individual examples and does not generalize on the dataset anymore. This is especially an issue, since our dataset is relatively small (compared to other datasets used in natural language machine learning applications).

Furthermore, our dataset was randomly split into 10 folds for performing 10-fold cross validation. Due to that, we could not ensure that the distribution of samples per class is approximately the same across all 10 pairs of training and test sets. Also, applying stratified sampling in a multilabel classification task is not a trivial task [18].

The classification was performed using only the text of the requirement in question. However, deciding the PVM of a requirement may require more information than just the text, such as other attributes or the text of surrounding requirements in the specification. Using more information may result in better classification performance, especially in the multilabel case. However, the label of a requirement may also depend on information that is not available in the specification documents, such as domain knowledge of the requirements engineers.

## VII. Related Work

Automatic classification on content elements of requirements specifications has been performed before for many purposes and using varying techniques.

Cleland-Huang et al. [22] built a classifier to distinguish between functional and non-functional requirements. Their dataset contains about 350 functional and 350 non-functional requirements. The non-functional requirements are further subdivided into NFR types such as availability, usability, security, etc. The classifier achieved an average recall of 80% and precision of 20%. The authors suspect that achieving 100% recall and a reasonably good precision at the same time may be impossible. They argue that their classifier may be used to aid requirements engineers reviewing requirements specifications and detecting non-functional requirements.

Hayes et al. [23] present a tool that incorporates the Weka toolset to perform automatic classification on requirements. The tool is evaluated using two different datasets: a dataset containing functional and non-functional requirements and a dataset containing temporal and non-temporal requirements. The authors argue that by providing easy-to-use tools, approaches incorporating machine learning techniques may be more likely to be adopted by requirements engineering practitioners.

In [24] and [25], the authors train and compare different classification techniques (Multinomial Naive Bayes, Support Vector Machines) for classifying requirements into different topics (such as temperature, velocity, voltage). They argue that by grouping requirements by topics, requirements engineers are able to review requirements much faster and more consistently.

## VIII. Conclusions & Future Work

In this work, we presented our approach towards classifying requirements regarding their potential verification method. We used convolutional neural networks adapted to text classification and trained this network on a dataset created from requirements specifications at our industry partner.

Overall, we achieved mixed results. Inspecting the performance of our model reveals that it does not work particularly well on certain classes, most likely due to imperfect training data. However, due to very good results on the *Test* class, our classifier is very well fit to separate requirements within and requirements outside that class.

As shown in previous studies on a related classification task [19], integrating the classifier into a tool to support review processes may provide certain benefits such as shorter review time and increased number of errors fixed. In the future, we plan to evaluate this using our PVM classification model.

However, certain improvements have to be made beforehand: First of all, a better and possibly larger dataset has to be created. This is particularly challenging since acquiring large amounts of high quality and consistently labeled data without manually labeling each sample is difficult.

Dividing the class *Test* into multiple subclasses may also be beneficial, since this verification method is very generic and many different test strategies exist, such as hardware in the loop and tests on the actual vehicle. Our industry partner hinted that they consider to subdivide the class *Test* into more refined classes such as *Test (component)*, *Test (integration)*, and *Test (vehicle)*.

Due to the points laid out in Section VI-C, the trained model is most likely not applicable in other companies or other domains. Whether it is possible to create a model that works across different domains and companies has to be 6investigated. We believe that for each domain and company, a specialized model has to be trained in order to be useful.

## References

[1] "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, pp. 1–94, 2011.

[2] M. Glinz, "A Glossary of Requirements Engineering Terminology: Version 1.6."

[3] J. Eckhardt, A. Vogelsang, and D. Méndez Fernández, "On the Distinction of Functional and Quality Requirements in Practice," in *Product-Focused Software Process Improvement*, P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen, Eds. Cham: Springer International Publishing, 2016, pp. 31–47.

[4] J. P. Winkler and A. Vogelsang, "'What Does My Classifier Learn?' A Visual Approach to Understanding Natural Language Text Classifiers," in *Proceedings of the 22nd International Conference on Natural Language & Information Systems*, ser. NLDB, 2017, pp. 468–179.

[5] C. C. Aggarwal and C. Zhai, "A Survey of Text Classification Algorithms," in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer US, 2012, pp. 163–222.

[6] J. T. Y. Kwok, "Automated Text Categorization Using Support Vector Machine," in *In Proceedings of the International Conference on Neural Information Processing (ICONIP*, 1998, pp. 347–351.

[7] V. Narayanan, I. Arora, and A. Bhatia, "Fast and Accurate Sentiment Classification Using an Enhanced Naive Bayes Model," in *Intelligent Data Engineering and Automated Learning – IDEAL 2013*, H. Yin, K. Tang, Y. Gao, F. Klawonn, M. Lee, T. Weise, B. Li, and X. Yao, Eds. Springer Berlin Heidelberg, 2013, pp. 194–201.

[8] J. Diederich, J. Kindermann, E. Leopold, and G. Paass, "Authorship attribution with support vector machines," *Applied Intelligence*, vol. 19, no. 1-2, pp. 109–123, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1023824908771

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, Pereira, Fernando C. N., C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc, 2012, pp. 1097–1105.

[10] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1746–1751.

[11] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, pp. 655–665.

[12] Y. Zhang and B. C. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," *arXiv preprint*, vol. abs/1510.03820, 2015.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint*, vol. abs/1301.3781, 2013.

[14] O. Maimon and L. Rokach, "Introduction to Knowledge Discovery and Data Mining," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Boston, MA: Springer US, 2010, pp. 1–15.

[15] J. P. Winkler and A. Vogelsang, "Automatic Classification of Requirements Based on Convolutional Neural Networks," in *3rd IEEE International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2016, pp. 39–45.

[16] F. C. Bernardini, R. B. d. Silva, R. M. Rodovalho, and E. B. M. Meza, "Cardinality and Density Measures and Their Influence to Multi-Label Learning Methods," *Learning & Nonlinear Models*, vol. 12, no. 1, pp. 53–71, 2014.

[17] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, Eds., *Training Deep Neural Networks on Imbalanced Data Sets: 2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.

[18] K. Sechidis, G. Tsoumakas, and I. Vlahavas, "On the Stratification of Multi-label Data," in *Machine Learning and Knowledge Discovery in Databases*, D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 145–158.

[19] J. P. Winkler and A. Vogelsang, "Using Tools to Assist Identification of Non-Requirements in Requirements Specifications - A Controlled Experiment," in *Requirements Engineering: Foundation for Software Quality: 24th International Working Conference*, 2018.

[20] B. Settles, "Active Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[21] A. Robins, "Catastrophic Forgetting, Rehearsal and Pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[22] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[23] J. H. Hayes, W. Li, and M. Rahimi, "Weka meets TraceLab: Toward Convenient Classification: Machine Learning for Requirements Engineering Problems: A Position Paper," in *1st IEEE International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, ser. AIRE, 2014, pp. 9–12.

[24] E. Knauss and D. Ott, "(Semi-) automatic Categorization of Natural Language Requirements," in *Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings*, C. Salinesi and I. van de Weerd, Eds. Cham: Springer International Publishing, 2014, pp. 39–54. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05843-6_4

[25] D. Ott, "Automatic Requirement Categorization of Large Natural Language Specifications at Mercedes-Benz for Review Improvements," in *Requirements Engineering: Foundation for Software Quality: 19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11, 2013. Proceedings*, J. Doerr and A. L. Opdahl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 50–64.