Automatic Classification of Requirements Based on Convolutional Neural Networks

Jonas Winkler
Technische Universität Berlin, DCAITI, Germany
jonas.winkler@dcaiti.com

Andreas Vogelsang
Technische Universität Berlin, Germany
andreas.vogelsang@tu-berlin.de

***Abstract*—The results of the requirements engineering process are predominantly documented in natural language requirements specifications. Besides the actual requirements, these documents contain additional content such as explanations, summaries, and figures. For the later use of requirements specifications, it is important to be able to differentiate between legally relevant requirements and other auxiliary content. Therefore, one of our industry partners demands the requirements engineers to manually label each content element of a requirements specification as "requirement" or "information". However, this manual labeling task is time-consuming and error-prone. In this paper, we present an approach to automatically classify content elements of a natural language requirements specification as "requirement" or "information". Our approach uses convolutional neural networks. In an initial evaluation on a real-world automotive requirements specification, our approach was able to detect requirements with a precision of 0.73 and a recall of 0.89. The approach increases the quality of requirements specifications in the sense that it discriminates important content for following activities (e.g., which parts of the specification do I need to test?)**

***Index Terms*—Requirements engineering, convolutional neural networks, machine learning, quality assurance, classification**

I. Introduction

Requirements specifications are used in many requirements engineering (RE) processes to document results. The purpose of these documents is to define the properties that a system must meet to be accepted. Moreover, in contexts, where one company or department acts as a customer and another company acts as a supplier, the requirements specification also defines liability between the partners (i.e., what must be achieved to fulfill the contract). For this reason, requirements specifications should undergo a rigorous quality assessment process especially in industries where systems are created by a collaboration of many suppliers (e.g., automotive).

Besides the actual and legally binding requirements, requirements specifications usually contain auxiliary content (e.g., explanations, summaries, examples, and references to other documents). These content elements are not requirements, which must be fulfilled by the supplier but they may facilitate the process of understanding requirements and their context. To distinguish these auxiliary information from legally binding requirements, one of our industry partners annotates all content elements in their requirements specifications with specific labels for *requirements* and *information*. However, this manual labeling task is time-consuming and error-prone. By analyzing a set of requirements specifications from our partner, we observed that labels (i.e., *requirement* and *information*) are often not added when the content is created. This impedes the usage of these documents for following activities, such as creating a test specification based on a requirements specification. Adding the labels at a later stage is expensive since every content element has to be read and understood again.

In this paper, we present an approach to automatically classify content elements of a natural language requirements specification as *requirement* or *information*. This approach can be used either to classify content elements in documents that have not been classified before or to analyze already classified documents and support the author in identifying incorrectly classified content elements.

Our approach uses convolutional neural networks, a machine learning approach that has recently gained attention in natural language processing [1], [2]. To train the neural network, we used a set of 10,000 content elements extracted from 89 requirements specifications of our industry partner. By using 90% of the content elements as training data and 10% as test data, our approach is able to achieve a stable classification accuracy of $\approx 81\%$.

In a preliminary evaluation we applied our approach to an unknown requirements specification with 747 content elements. Our approach was able to classify *requirements* with a precision of 0.73 and a recall of 0.89 and *information* with a precision of 0.90 and a recall of 0.75.

We argue that an explicit differentiation between *requirements* and *information* increases the quality of a requirements specification because it facilitates the use of the document for following activities (see also [3]). An accurate differentiation between *requirements* and *information* is vital for these activities to be successful. For example, when a test specification is derived from a requirements specification, this differentiation defines for which content elements a test case has to be created. Also, this differentiation defines which content elements have to be implemented by a supplier.

II. Background

A requirements specification is a document that contains requirements and requirement related information for a specific scope and on a specific level of abstraction. Requirements specifications may serve different purposes, which influence the type and representation of information contained in the document. To cover all kinds of requirements specifications,

978-1-5090-3694-3/16 $31.00 © 2016 IEEE
DOI 10.1109/REW.2016.16

RE 2016 Workshops, Beijing, China
AIRE Workshop Paper

we characterize a requirements specification as a set of content elements (see [4]). Content elements are atomic parts of a requirements specification and usually contain a single sentence or a figure. Content elements can be associated with different attributes. For this paper, the following attributes are relevant:

- **Text:** This attribute contains the text body of the content element.
- **Type:** This attribute defines whether the content element is a *requirement*, which has to be satisfied by the supplier, or an *information*, which provides additional content that is legally not relevant. More types are used by our industry partner for special-purpose content elements.

To better understand the different types of content contained in a requirements specification, we examined a set of requirements specifications from our industry partner, an automotive manufacturer. Although the documents were written by different people, the kind of content and the structure of the documents are quite similar. We made the following observations concerning the content:

- Most content elements contain natural language text. We observed that phrasing tends to be more precise for *requirements* compared with additional *information*.
- We identified unique phrasing and formatting within the content elements of individual specification documents. This is due to the fact that these documents are created by different authors with slightly different understandings about how these documents should be created.
- In addition to natural language content elements, many content elements are created using structured and semi-formal notations, such as enumerations, tables, diagrams, equations, logical expressions, and key-value pairs (e.g., "Maximum Voltage: 10mV").
- Certain content elements are always classified using the same label. For example, elements that represent references to external documents are always classified as *information*, whereas voltage range specifications are always classified as *requirement*.
- In some requirements specifications, the attribute *type* is not defined consistently. Remarks, which obviously are not requirements, are not labeled as *information* in some cases. Sometimes a classification was completely missing for the whole document.

In previous studies, we have shown that simple rule-based approaches are sufficient to classify content elements that follow a structured or semi-formal notation as the specific label can be derived from the used notation [5]. The difficult classification part resides in the content elements that contain unstructured text. Therefore, in this paper, we focus on the classification of content elements that contain unstructured text. In the following, we provide some examples for content elements of both categories.

- A reset counter must be implemented. *(Requirement)*
- The relay must not be deactivated in case of a voltage drop. *(Requirement)*
- Additional information can be found in Chapter A. *(Information)*
- The circuit diagram shows a schema of the component and its parts. *(Information)*

III. Convolutional Neural Networks for NLP

Classifying content elements of requirements specifications as either *requirement* or *information* is a two-class classification problem. Within the natural language processing community, many popular techniques exist to solve such a problem, including Naive Bayes [6] and support vector machines [7]. Although these techniques have limitations, such as ignoring word order, they have proved to be good enough for classification tasks such as sentiment analysis [8] or authorship attribution.

Convolutional neural networks (CNN) are a variation of classic feed-forward neural networks, which are widely used within the image recognition community [9] but have recently gained attention in natural language processing as well [1], [2]. These networks have several advantages compared with other classification techniques:

- Techniques such as Naive Bayes and support vector machines often rely on the bag-of-words approach to convert natural language sentences into machine understandable feature vectors. Information about the order of words in the sentence is lost in the process. CNNs for natural language processing operate on a sentence representation that keeps word order intact. Therefore, CNNs may learn and recognize patterns consisting of word sequences spanning multiple words in a sentence.
- To convert a natural language sentence into a machine understandable format, a word vectorization technique such as word2vec [10] or GloVe [11] is employed. This allows the network to recognize patterns even if the words used in the occurrences of the pattern 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. A more complete description of CNNs for natural language processing is provided by Zhang and Wallace [12]. For an introduction into the fundamentals of neural networks, refer to the book by Nielsen [13].

The first step is to transform an input sentence into a vector representation (1). This is called word embedding. We use word2vec 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 predefined set of filters (2) to the sentence matrix m. Each filter is a matrix $f \in \mathbb{R}^{n,o}$ of 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

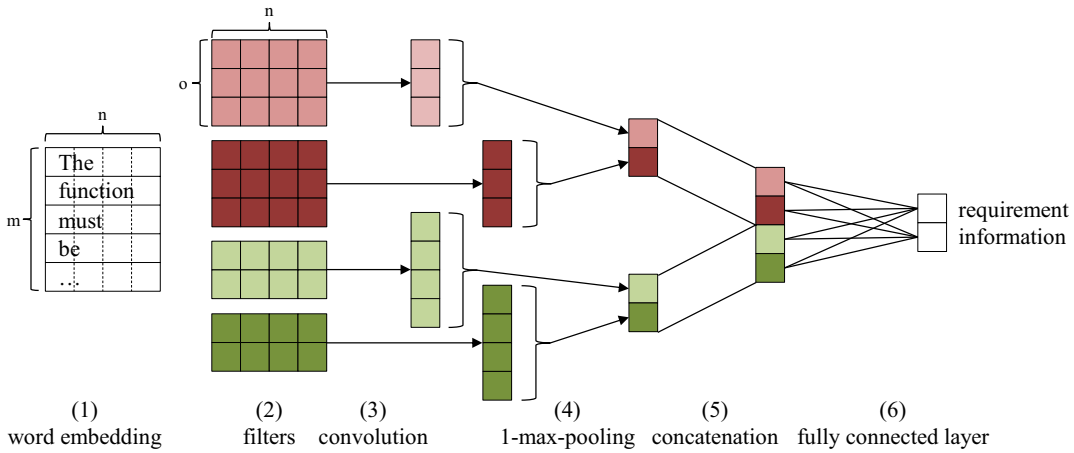

Fig. 1. Convolutional neural network architecture (simplified) as proposed by [12]

using an activation function such as a rectifier or sigmoid function (3). 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 then reduced to a single value by applying 1-max-pooling (4). 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 (5). This vector is connected to the output layer using a standard fully connected layer and an appropriate set of trainable parameters (6). The fully connected layer is used to associate certain patterns with an output class (e.g., the network might learn to associate the pattern "must be" with the class "requirement"). A softmax layer is finally used to create a true probability distribution.

IV. Approach

To construct a classifier that is able to distinguish information from requirements in natural language sentences, we employed the Knowledge Discovery in Databases (KDD) process as described by Maimon and Rokach [14]. This process describes necessary steps and risks to be aware of when attempting to create knowledge from raw data using data mining techniques. The process contains the following 9 steps:

1) **Understanding the application domain.** We did a thorough analysis of our data to gain a better understanding about *requirements* and *information* (see Section II).
2) **Creating a dataset.** We have created a dataset using the knowledge gained in the previous step. Details will be described in Section IV-A.
3) **Preprocessing and cleansing.** We applied preprocessing steps to the dataset to remove noise. Details will be described in Section IV-A as well.
4) **Data transformation.** We transformed the data into a format appropriate for training a machine learning algorithm. We decided to use the word2vec word embedding technique [12].
5) **Choosing the appropriate data mining task.** The problem of determining the type (*requirements* or *information*) of a given content element is a classification problem.
6) **Choosing the data mining algorithm.** We selected CNNs for approaching our problem due to their recent success in many common natural language problems [12].
7) **Employing the data mining algorithm.** The procedure of selecting hyper parameters and training of the model is described in Section IV-B.
8) **Evaluation.** We evaluated our approach by applying it to a requirements specification from industry. The results are presented in Section V.
9) **Using the knowledge.** Possible ways to incorporate the created model into a tool for quality assurance will be discussed in Section VI.

A. Constructing the Dataset

The DOORS document database of our industry partner contains all requirement documents of the company. For training the convolutional neural network, we selected 89 documents from that database based on the following criteria:

- The document must be a specification describing a single electronic vehicle component. We excluded multi-component specifications and auxiliary documents because these are written on a different level of abstraction using different phrasing and terminology and thus might negatively affect the training process.
- All documents must be written in the same language. In our case, most documents were written in German. Therefore, we only selected documents written in German.
- To be able to train the classifier, the content elements of the selected documents must be classified as *information* or *requirement*. Therefore, we selected only documents where most content items were manually classified before.
- The classification of the content elements within the document must be reliable. We inspected each document and assessed the classification quality. We excluded

documents in which classification was questionable (e.g., all content elements classified as requirement)

We extracted all content elements from the resulting document set that are either classified as *information* or *requirement*. We ignored content elements copied from templates during this process because the classification of these elements is specified by the templates. We also ensured that the resulting set of content elements does not include any duplicates.

Since we want to build a classifier that determines the class of natural language sentences, we filtered the dataset to only include content elements containing natural language sentences. We specifically discarded content elements containing headings, figures, tables, math expressions, logical expressions, itemizations, and other structured content. We used the Stanford Parser [15] to identify sentences.

To improve the quality of the dataset, we applied a single-link text clustering algorithm [16] to the dataset. We identified several groups of content elements with very similar content (e.g., only component names, numbers or few individual words varied) but inconsistent classification: most were classified as requirement and some were classified information, or vice versa. We manually adjusted the classification of wrongly classified items within large clusters.

All content elements were preprocessed using standard preprocessing steps such as converting text to lower case and removing stop words.

The resulting dataset was imbalanced, containing approximately 5 times more requirement content elements than information content elements. This is a major problem, since a classifier trained on an imbalanced dataset would be heavily biased towards the majority class [17]. To deal with this problem, undersampling was applied to the dataset. Instead of using random undersampling, we decided to use a clustering algorithm on the dataset again and repeatedly removed random content elements from large clusters until the dataset was balanced. This ensures that important content elements (i.e., elements with rarely used phrasing) are not removed from the dataset. It also helps to reduce overfitting because only very similar or even partially identical content elements are removed from the dataset.

B. Building the Classifier

We built the convolutional neural network using the guidelines provided by Zhang and Wallace [12]. For implementing the actual network, we used the Tensorflow[1] library.

The word2vec word embedding was trained once using the dataset created as described in the previous section. This created both a dictionary of frequent words and a mapping of each frequent word to a vector. These are used to convert sentences into matrices.

The hyper parameters of the model such as embedding size, filter sizes and filter count per size were chosen using an iterative process. We started with small and few filters and increased both the count and the size of the filters until the performance of the network stopped to increase. We achieved best performance using embedding size 128, filters of size 1, 2, and 3 and 64 filters per size (192 filters total).

[1]https://www.tensorflow.org/

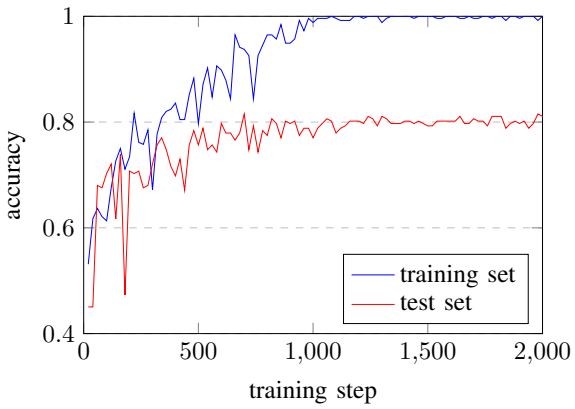

Fig. 2. Development of the accuracy of the CNN

To train the network, the dataset was randomly split into 90% training data and 10% test data. The network was trained on the training data using stochastic gradient descent.

V. Evaluation

We evaluated the performance of the trained network using standard measures. This includes the network accuracy on the training and test set, as well as precision, recall and f1-score of both target classes on the test set.

Fig. 2 visualizes the performance on the training set and test set during training. After roughly 1,000 training steps (approx. 10 epochs), the network reaches 100% accuracy on the training set. After training completes, test set performance is 81%, meaning that about 4 out of 5 examples from the test set were classified correctly.

TABLE I
Performance Evaluation

Class	Precision	Recall	f1
Requirement	0.733	0.885	0.802
Information	0.896	0.754	0.819

The precision and recall values of both target classes in Table I reflect this result. Overall, the network prefers to classify elements as *requirement*. This is indicated by the low precision and high recall of the class *requirement*. Precision on the class *information* is relatively high, although only 75% of all information elements within the test set were correctly identified as such.

To further assess the quality of the predictions provided by the network, we identified two indicators. The first indicator uses the output values of the network. The network outputs one value for each class, which are trained to be 0 for the false class and 1 for the true class. The absolute difference between both values before softmax application is correlated

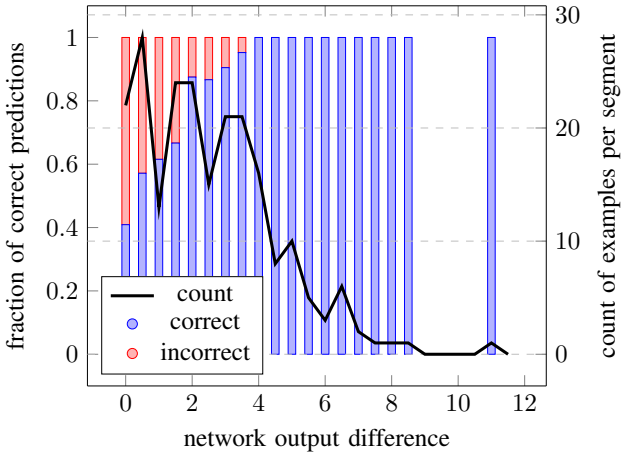

Fig. 3. Network output difference to network accuracy correlation

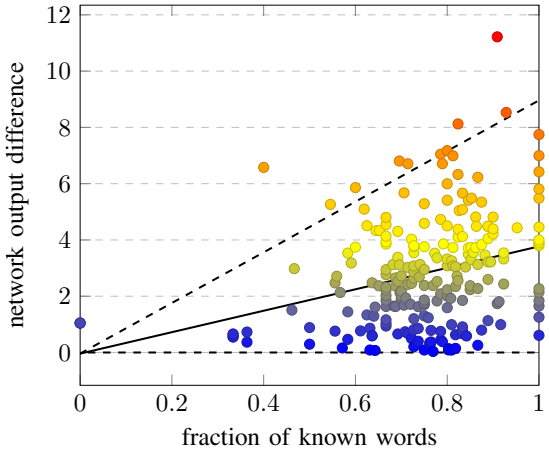

Fig. 4. Fraction of known words to network output difference correlation

to the probability of a correct prediction, i.e., a classification result with an *information* value of -2.5 and a *requirement* value of 2.2 is more likely to be correct than a classification result with an *information* value of 0.3 and a *requirement* value of 0.5.

Fig. 3 visualizes this correlation. We classified a set of examples that have not been part of the training process using a trained network and sliced the examples into multiple segments. Each segment contains the examples whose output difference is within a specified output difference interval. We used an interval length of 0.5. The figure shows that the higher the output difference, the higher the fraction of correctly classified examples within that segment. The figure also shows that only very few examples yielded a very high output difference, whereas most of the examples (75%) yielded an output difference of 4 or less.

Examples with a difference of 1 or less are very likely to be classified incorrectly. Since the network produces incorrect predictions for approximately half of these examples, we assume that these examples do not contain any decisive features correlated with either of the two output classes. On the other hand, 25% of the examples yielded an output difference of 4 or more and were always classified correctly. This suggests the definition of a threshold below which results will be regarded as possibly incorrect. How such a threshold might be useful will be discussed in Section VI.

The second indicator uses the fraction of known words within an example. The word embedding part of the neural network uses a dictionary to create sentence matrices. Words not found in the dictionary are marked as unknown during sentence matrix creation and as such do not contribute to the classification process. We assume that the higher the amount of known words within an example is, the higher the likeliness of a correct prediction will be.

Fig. 4 visualizes this correlation. Each example is represented by a dot, indicating its fraction of known words and its output difference yielded by the network. The fraction of known words is 0 if the example contains no known words and 1 if all words of the example are known. A linear regression on the data points (solid line) reveals that the higher the fraction of known words, the higher the average output difference (and thus also the likeliness of a correct prediction as shown in Fig. 3) of the neural network.

Although there are very few examples with a fraction of known words below 0.4 (due to the fact that the test data was derived from the same source as the training data), the diagram shows that a low fraction of known words leads to a smaller output difference and thus is a cause for wrong classification of a particular content element. A high fraction of known words does not necessarily imply that a prediction is correct.

VI. Discussion

Using the results gained during evaluation, we present some notable examples from our test set to demonstrate the abilities and limitations of the neural network. Table II shows a set of example sentences, their respective true class, the fraction of known words, and the output provided by the network. Unknown words within the sentences are marked in gray.

The first row shows a requirement that was classified correctly by the network. Since the difference between the output values is reasonably high, it is relatively safe to assume that the classification of the network is correct (Fig. 3 shows that almost 90% of the examples with an output difference of 2.5 were classified correctly).

In sentence 2, all words are known and the output difference is exceptionally high. This sentence contains words and phrasings that are very common for information content elements, such as “described in detail”.

Sentence 3 was classified incorrectly. The network was simply unable to deal with this example, as indicated by the low fraction of known words, the low output difference and the fact that neither of the two output values is close to or greater than 1.

Sentence 4 was classified incorrectly as well, although we do not know why. The fraction of known words is reasonably

TABLE II
EXEMPLARY SENTENCES, THEIR RESPECTIVE TRUE CLASS, FRACTION OF KNOWN WORDS, AND THE OUTPUT PROVIDED BY THE NETWORK.

Sentence	Class	Known words[2]	Inf	Req	Diff
The transformer must switch to *self-protection mode* according to the *derating strategy*.	req	0.727	-1.655	0.986	2.641
This function is described in detail in the trunk lid system description.	inf	1	3.386	-4.359	7.745
Internal short-circuits due to *willful intrusion* are *allowed* to *lead* to *destruction*.	inf	0.364	-0.919	-0.551	0.368
The *contributions* for the controller are described in the *door opener* system description.	inf	0.727	-0.898	0.302	1.200

high and the sentence seems to be similar to sentence 2. Insight into the network's learned features would help to understand this output.

A. Limitations of the Approach

The last example highlights a fundamental limitation of our approach. The network does not provide any insight into what it learns and why a certain output is produced. This problem is well known within the neural network community [18]. Many approaches have been proposed to deal with this problem, both generic approaches such as fuzzy rule extraction from neural networks [19] as well as domain specific approaches (i.e., visualizing weights of deep image recognition networks [20]). A technique to trace back decisions through the network to identify relevant patterns in the input sentence would certainly be important to real users, especially when incorporating our approach into a tool.

We also identified that the applicability of our approach might be limited to the documents of the industry partner whose documents we used to train the network. We used the trained network to classify content elements of documents provided by a different industry partner and received inferior results. The most likely reason for that is that most of these content elements had a fraction of known words below 0.4.

B. Threats to Validity

Although the dataset was created with great care towards quality and common threats such as class imbalance and leakage of information into the test set, the quality of the dataset might still impact the presented results in a negative way. During evaluation, we identified several examples in the dataset that were either classified incorrectly or were poorly written. These examples might affect the training process, restraining the network from learning relevant patterns.

Another issue that commonly arises with machine learning techniques is overfitting. Our network might be heavily biased towards specific and often reoccurring words and patterns in our training set and therefore might not be applicable to other documents. We still need to analyze whether this is an issue with our model.

C. Applications in Industry

To apply our approach in industry, we plan to integrate a pre-trained CNN into a tool. This tool shall assist the requirements engineer in three different scenarios:

[2]The values correspond to the German versions of the given examples since the German version was actually analyzed by our approach.

The requirements engineer may use the tool to identify misclassified content elements in a document in which content elements were already classified. The tool will analyze each content element and issue warnings if a misclassified item is identified. We are considering to define thresholds on the network output difference and the fraction of known words as presented in Section V to prevent issuing false positives.

The tool may also be used to create an initial classification for all content elements of a document in which content elements were not classified.

We are currently investigating whether our approach is able to detect content elements with low quality. Low output difference values and low fractions of known words imply the use of infrequent words and phrasing and thus could be indicators for content elements which do not conform to established guidelines and maybe need to be revised. The tool may issue remarks for those content elements.

We argue that introducing a tool that supports these scenarios, authors are able to identify misclassified items faster and are encouraged to write higher quality (i.e., easier to classify) content elements. Ideally, the tool would also provide explanations why a certain warning or remark is issued by highlighting specific parts of a content element.

VII. RELATED WORK

Automatic requirements classification and application of machine learning approaches in the context of RE has gained attention since natural language processing and computing power enables the automation of tasks that have traditionally been performed manually.

Hayes et al. [21] introduced a tool for analyzing requirements, which includes a set of components for automatic requirements classification. The tool can, for example, differentiate between temporal and non-temporal requirements. Huang et al. [22] described an approach to detect and classify non-functional requirements automatically. The approach iteratively trains a classifier of non-functional requirements.

The successful use of machine learning techniques for supporting RE tasks has been shown for many cases [21]. Several different machine learning techniques have been applied for different purposes. Ott [23], for example, uses Naive-Bayes and Support Vector Machines to classify requirements with respect to a common topic to improve the review process. Hayes et al. [24] use the C4.5 decision tree algorithm to predict the testability of requirements based on a set of textual features. Perini et al. [25] proposed an approach for deciding software

requirement priority using machine learning. This approach takes into account both the requirements ordering generated using machine learning techniques and the stakeholders' preference. The combination of both types of information facilitates the task of requirements prioritization.

VIII. Conclusions

In this paper, we proposed an automatic approach for classifying content elements of natural language requirements specifications as *requirement*, which are legally relevant or *information*, which only provide additional explanations or references. The approach can be used to classify content elements in documents that have not been classified before or to analyze already classified documents and pinpoint the author to possibly incorrect classifications of content elements.

The presented approach uses convolutional neural networks that we trained on a set of existing requirements specifications. After training the neural network, the approach is capable of classifying new requirements documents with an accuracy that is comparable to the accuracy of CNNs applied to other tasks [12]. The accuracy of the network may be further improved by increasing the amount of training data (i.e., by including documents of other types as well, such as multi-component system specifications) and by increasing the quality through careful application of semi-automated techniques to filter out bad and incorrectly classified content elements.

Besides improving the effectiveness of the approach, we currently work on integrating a pre-trained network into a requirements management tool with the goal to pinpoint the requirements engineer to incorrectly classified content elements. Despite the fact that precision and recall is not exceptionally high, our evaluation showed that indicators exist which help to minimize the amount of false positive warnings.

An additional challenge is to provide the user not only with the actual findings but also with an explanation why a content element is classified incorrect. Especially with neural networks, it is not easy to provide these explanations since the learned decision process within the network can be quite complex. We are currently investigating ways to provide these explanations.

References

[1] 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.

[2] 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.

[3] H. Femmer, J. Mund, and D. Méndez Fernández, "It's the Activities, Stupid!: A New Perspective on RE Quality," in *2nd International Workshop on Requirements Engineering and Testing (RET'15)*, 2015.

[4] D. Méndez Fernández, B. Penzenstadler, M. Kuhrmann, and M. Broy, "A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering," in *Model Driven Engineering Languages and Systems*, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds. Springer Berlin Heidelberg, 2010, pp. 183–197.

[5] J. P. Winkler, "Automatische Klassifikation von Anforderungen zur Unterstützung von Qualitätssicherungsprozessen," in *INFORMATIK 2016. Lecture Notes in Informatics (LNI).*, H. C. Mayr and M. Pinzger, Eds., Bonn, 2016.

[6] 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.

[7] 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.

[8] 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.

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

[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[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] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[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] D. Klein and C. D. Manning, "Accurate Unlexicalized Parsing," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ser. ACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 423–430.

[16] C. C. Aggarwal and C. Zhai, "A Survey of Text Clustering Algorithms," in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer US, 2012, pp. 77–128.

[17] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview: Data Mining and Knowledge Discovery Handbook," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Boston, MA: Springer US, 2010, pp. 875–886.

[18] J. M. Benítez, J. L. Castro, and I. Requena, "Are Artificial Neural Networks Black Boxes?" *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1156–1164, 1997.

[19] C. J. Mantas, J. M. Puche, and J. M. Mantas, "Extraction of similarity based fuzzy rules from artificial neural networks," *International Journal of Approximate Reasoning*, vol. 43, no. 2, pp. 202–221, 2006.

[20] J. Yosinski, J. Clune, A. M. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization," *arXiv preprint*, vol. abs/1506.06579, 2015.

[21] 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*, ser. AIRE, 2014, pp. 9–12.

[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] D. Ott, "Automatic Requirement Categorization of Large Natural Language Specifications at Mercedes-Benz for Review Improvements," in *Proceedings of the 19th International Conference on Requirements Engineering: Foundation for Software Quality*, ser. REFSQ. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 50–64.

[24] J. H. Hayes, W. Li, T. Yu, X. Han, M. Hays, and C. Woodson, "Measuring Requirement Quality to Predict Testability," in *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2015, pp. 1–8.

[25] A. Perini, A. Susi, and P. Avesani, "A Machine Learning Approach to Software Requirements Prioritization," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 445–461, 2013.