Classification of Software Requirements from Online Feedback using Bio-inspired AI techniques

Abhishek Chandar
EECS Department
University of Ottawa
Ottawa, Canada
achan260@uottawa.ca

***Abstract*— Requirement engineering is a crucial but time-consuming task in any software engineering project. With the goal of helping software developers in the process of extracting requirements, an automated NLP pipeline is proposed that leverages language models to extract requirements from online user feedback data. This paper identifies the problem of appointing domain experts to perform qualitative coding manually to identify requirements from a particular source of data. To improve the efficiency of the requirement extraction process, this study focuses on automating the requirement elicitation process. Previous research works have extensively implemented traditional machine learning techniques to fetch requirements from a particular source of data such as Tweets and legal contracts. In this work, the performance of language models to perform text classification and thereby identifying requirements from online user feedback will be investigated. This problem is especially hard because of the lack of software engineering datasets that are labelled which is the reason for exploring transfer learning by implementing language models in this work. To automate the requirement extraction process, a well-established manual approach called Kyoryoku is considered and partially automated. In this work, language models namely BERT, ELMO and ULMFit were implemented. Based on the experimentation, it was found that BERT performed significantly better than ELMO and ULMFit for the dataset that we have with an average F1 score of 90%.**

Keywords—Requirement elicitation, Natural language processing, Transfer learning

I. Introduction

Requirement elicitation [1] is the first step in understanding the features that need to be built in a software engineering project. Usually, requirement elicitation is an iterative process where the requirements for new features are constantly observed so that new versions of the application are relevant to the end-users [2]. The purpose of this study is to explore the possibility of implementing deep learning techniques to automate the process of requirement elicitation from online feedback platforms. These include platforms such as Google Play Store and App Store in which users provide feedback on the applications that the users have used. Previous research work shows that crowdsourcing datasets such as Online feedback of applications are reliable sources of requirements [3]. Even though online feedbacks are valuable source of requirements, not all the feedbacks can be directly considered as requirements. Feedbacks can be considered as requirements only if they are constructive in nature. Therefore, it is important to develop a system to automatically extract the requirements i.e. Reviews that are *useful* from a corpus of online feedback. This study was undertaken to increase the efficiency of the requirement elicitation of requirements related to software engineering projects. Requirement gathering is already proven to be a crucial yet time-consuming task for any SE project [4]. Therefore, there is a need to improve the state-of-the-art automated techniques in extracting SE requirements from a particular source such as Online feedback. This area of research is relatively new with the implementation of traditional machine learning techniques to classify whether a sentence is a requirement or not. With the advancement in deep learning algorithms in recent times, Natural language processing (NLP) language models have not yet been completely experimented on software engineering datasets to extract requirements. This research work is an attempt to investigate whether language models can be implemented to extract software engineering requirements from online feedback textual data. The primary motivation of this research work is to automate a time-consuming software engineering task of identifying requirements. Further, this work also leverages application reviews as a viable source of requirements. Various strategies have already been proposed in the field of software engineering to effectively extract requirements from a source such as software engineering contracts [5]. One such effective strategy that is automated in this work is called Kyoryoku [6]. But these strategies are manual in nature. Thus, the research goal of this work is to automate the manual approach Kyoryoku that comprises of three main steps:

- Preliminary filtering (Filtering): In the first step, out of all the reviews from the dataset, only the useful reviews are extracted. The usefulness of a review is based on whether it contains phrases that represent requirements.
- Secondary filtering (Fragmenting): In the second step, each useful review is further broken down into individual sentences and each sentence is further filtered based on whether it represents a requirement or not.
- Categorization of requirements (Categorizing): Based on the output from the second step, the requirement phrases are further categorized into different categories such as Quality, Feature, Bug fix etc. The classes are usually related to software development so that it is easier for the developers to understand the type of requirement.

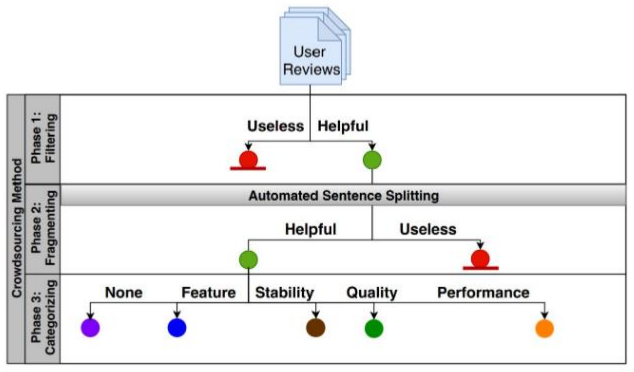

Fig. 1. Kyoryoku method of requirement extraction

The research questions that are addressed in this research are as follows:

RQ1: Can all the three steps of Kyoryoku be automated using NLP?

RQ2: How does the performance of language models compare with respect to the size of data?

RQ1 investigates the feasibility of the degree of automation that can be performed. On the other hand, RQ2 investigates the performance of different language models at a holistic level and specifically with respect to the size of data. The reason is because language models being a deep learning model requires large datasets for training. But it recent times, these models are being applied on small datasets to leverage the concept of transfer learning. So, RQ2 will be answered based on the experimental results.

II. LITERATURE REVIEW

In this area of research, previous research works focused more on apply traditional machine learning algorithms when compared to deep learning algorithms. Specifically, the application of language models for software engineering datasets is rare. Some of the significant works in this area is discussed below.

Mengmeng Lu and Peng Liang et al [7] proposed an automatic classification method to extract Non-functional requirements (NFRs) from App User reviews. Further, the requirements are classified into four types of NFRs. namely reliability, portability, usability, and performance. In this paper, classification techniques BoW, TF-IDF, CHI-squared and a novel method AUR-BoW were implemented along with machine learning classification algorithms Naive Bayes, Decision Trees and Bagging ensemble methods. Based on their experiments, it was concluded that the Bagging ensemble method performed the best.

Stanik, Christoph & Hearing et al. [8] applied traditional machine learning and deep learning techniques to classify user feedback that can be used as a valuable source of information to improve the application. The dataset consists of 10,000 English and 15,000 Italian tweets from Twitter support accounts and 6000 annotations of app reviews in English. The goal of their work was to implement both machine learning and deep learning techniques and provide a comparative analysis of different models to identify the best performing model. Traditional machine learning algorithms that were applied include Random Forest and Decision Tree. Convolutional neural networks were used to represent deep learning algorithms. For the datasets they used, it was concluded that traditional machine learning algorithms performed better than deep learning algorithms especially because the size of the training data was small.

T. Hey et al [9] took the deep learning path and proposed a novel deep learning architecture called NoRBERT which is based on the language model BERT which has been proven to display high performance in the field of NLP. NoRBERT was applied on PROMISE NFR dataset to classify non-functional requirement subclasses. It was found that there was a 15% improvement in the proposed approach with F1-score of 92% when compared to the baseline approach. Finally, they were also able to statistically prove at 95% level of significance that transfer learning-based techniques can be an alternative to traditional machine learning approaches. The PROMISE NFR dataset contains about 625 examples from 15 different projects classified into 12 different classes. This also proves the fact that language models can be implemented in small datasets as well which is the reason why it is highly suitable for software engineering related datasets which are usually small in size.

In terms of extracting requirements from different sources, previous works have used tweets from Twitter and legal contracts as the major source of requirement extraction. E. Guzman et al [10] proposed an automated NLP approach to extract requirements from legal contracts. Base on initial exploratory study of legal contracts, they found that sentences in contract are of either obligatory in nature or non-obligatory in nature. Further, it was also found that obligatory sentences have high probability of being a requirement. This assumption was leverages to perform a two-step binary classification to predict whether a sentence is obligatory or not and then the obligatory sentences were classified based on whether it is a requirement or not. They implemented both traditional ML algorithms and BERT to extract requirements from legal contracts and it was found that BERT performed the best when compared to traditional machine learning algorithms.

On the other hand, Leonidis, Asterios & Baryannis et al [11] suggested ALERTme which is an approach to perform classification, grouping and ranking of tweets based on software applications. Machine learning algorithms namely Random Forest, Naïve Bayes classifier were implemented on tweets to perform the same. Their dataset consisted of a total of 68,108 tweets from three different applications and the performance of different ML algorithms were studied. Finally, it was found that Multinomial classifier performed better than Random Forest on F1-score.

Therefore, based on the previous research attempts in this area, it can be concluded that online user feedback has proven to be an effective source of requirements. Although some works have compared both traditional Machine learning and Deep Learning algorithms, this work does not focus on that comparison since there are not empirical guidelines to compare ML and DL algorithms. On the other hand, this work will perform a comparative study on the performance of different language models since the overall approach of language models remains the same and comparison of language models will be on a fair ground unlike a comparison between a machine learning algorithm with a pretrained language model.

III. BACKGROUND

Before discussing the methodology that is proposed in this work, there are some important concepts that form the background for the methodology and the results following it. In order to provide some context to the language models and the dataset, related concepts are explained at a high level below.

In terms of the language models, BERT [12], ULMFit [13] and ELMO [14] are implemented for both the preliminary as well as secondary classification. The dataset used in this research is derived from the work of Groen et al. Further, Van Vliet et al created a sample of 1000 online users reviews from Play Store and App stores related to 8 different apps from the work of Groen et al.. It is important to note that the dataset sample was bootstrapped to have an equal distribution of good and bad reviews. In this case, *useful* reviews means that the review was constructive, and requirements can be extracted from those feedbacks and *useless* reviews implies that it is not useful for our requirements. The dataset introduced by Groen et al was originally used to identify software product quality characteristics which has broader scope than considering only the online user feedback for requirement engineering. Therefore, the dataset proposed by Van Vliet et al is used in this research which completely focuses on user reviews of different apps.

Language models in the field of Natural language processing implements the concept of transfer learning over textual data. These models are usually trained on millions of text data and therefore can understand the semantics and structure of English text out of the box. The idea is to implement these pre-trained models that can perform any NLP task on the target dataset to perform a specific task such as Text classification, Text summarization and Text generation. In this work, BERT, ULMFit and ELMO are implemented to perform text classification at review level and finally sentence level as well. The language models that are applied for text classification in this work are further explained below:

BERT: Bidirectional Encoder Representations from Transformers (BERT) is a deep learning model that leverages a bidirectional transformer to generate a model. It is one of the state-of-the-art language models used for various Natural language processing tasks. In this work, BERT is implemented to perform text classification. More specifically, BERT is implemented to classify useful and useless sentences in the preliminary classification and further used to classify each sentence from useful reviews into requirements or not in secondary filtering. Training a BERT model is especially different when compared to other language models in that it uses the previous word and the next word to predict the masked words in a sentence. On top of that, it also uses attention by implementing two techniques. They are called “Masking” and “Next sentence prediction” and these techniques are used to learn the semantic context of the word. Previous research has also proven that BERT performs well with unseen vocabulary words which will be an advantage in this case since the different people write online feedback in different ways. Usually, BERT is developed using two main steps. The first step is referred to as “Pre-training” in which Masking and Next sentence prediction is carried out to learn the language model. The second step is called “Fine-tuning” and in this step, the language model is used to perform a specific NLP task for our dataset.

ELMO: “Enough, Let's Move On” model is a pre-trained language model that is trained on Google 1-Billion Words dataset [15] and tokenized by Moses tokenizer. By default, just like any other language model, ELMO can understand the semantics of textual data because it is already pre-trained on a huge corpus of data. Further, to enable ELMO to perform classification on our dataset to extract requirements from application reviews, we fine-tune the model on our dataset. For this, the initial weights are set as the weights of the pretrained model and then training is performed from that checkpoint. This will make sure that the model understands the nuanced details of our dataset on top of the previous information from the pretraining stage. Finally, the language model will be trained to classify the requirements by providing the labelled dataset as input and further tested on an evaluation set.

ULMFit: Universal Language Model Fine-tuning for Text Classification is a language model that can be applied for various NLP text classification. It consists of 3 layers of AWD-LSTM architecture. This model is different in the fine-tuning process as it implements discriminative fine-tuning. According to this process, ULMFit gradually unfreezes the model from the final layer. Each layer from the one closer to the output layer to the first layer is unfreezed after evey epoch cycle and is iteratively fine-tuned until all the layers are tuned and convergence is reached. This model is known for performing significantly well on small sized datasets. The performance of ULMFit on the dataset used in this work will be used to analyze whether the model performs significantly better than other language models for small sized datasets.

IV. METHODOLOGY

The overall idea of this research is to automate Kyoryoku approach in which the first step is used to filter useless reviews. Therefore, this can be translated into a binary classification problem in which the target classes to predicted will be “Useful review” and “Useless review”. In this case, the reviews that do not contain any requirements are referred to as useless. In the second step, the sentences of each useful review are split into individual instances and subject to further classification of whether the sentence is a requirement or not. By this approach, filtering is performed to extract requirements at the review level, as well as sentence level of each useful review. Due to the lack of labelled dataset, only step 1 and 2 of Kyoryoku is automated in this work.

Fig.2 depicts the overall workflow of the proposed NLP pipeline to automate requirement extraction. The process starts from top starting from preprocessing of application reviews to the final step of evaluation the language models based on evaluation metrics.

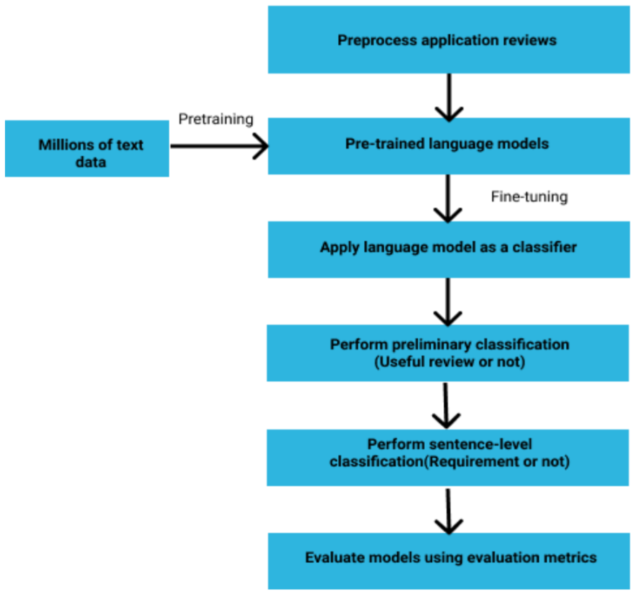
Preprocess application reviews
Millions of text data
Pretraining
Pre-trained language models
Fine-tuning
Apply language model as a classifier
Perform preliminary classification (Useful review or not)
Perform sentence-level classification(Requirement or not)
Evaluate models using evaluation metrics

Fig. 2. Proposed NLP pipeline

A. *Preprocessing*

The textual data from the dataset can often be filled with characters that do not contribute to the classification task that is necessary. For example, online reviews can have emojis which needs to be eliminated. Therefore, it is important to perform some NLP preprocessing so that the model does not get ambiguous data as input. The steps taken in the preprocessing step are as follows:

- Conversion of text into lowercase
- Removal of stop words
- Filtering unnecessary hyperlinks, special characters and emojis

Apart from performing preprocessing to clean the data, another set of preprocessing techniques were performed to transform the textual data into a feature matrix that can be used as an input to the language model. For example, adding special tokens at the beginning and end of each text data with padding so that the size of each input vector is equal.

B. *Fine-tuning language models*

All the language models are trained on large corpus of textual data so that the models can understand the overall semantics and structure of the sentences in the dataset. To localize the model to the dataset that is used in this work and perform a specific NLP task, fine-tuning of these language models is required. Fine-tuning process involves using the pretraining language models i.e. Using the last weights of those pretrained model and then perform further training on the dataset of our choice on top of that. In other words, the language models are continued training on the dataset used in this work to understand the vocabulary and the sentence structure in the dataset and ultimately build embeddings that understand the overall semantics of the sentences in the dataset. This embedding is further used to build a classifier. This classifier is the preliminary level of classification will classify whether the review is useful and not. Therefore, the input of the first classification is individual reviews, and the output will be a classification of whether it is useful or not. In the second level of filtering, the classifier is used to identify sentences that represent requirements from the useful reviews. In this case, the input is individual sentences of useful reviews and the output will be classification of whether the sentence represents requirement or not. Finally, the evaluation metrics that were used and the results are explained the next section.

V. RESULTS AND DISCUSSION

The NLP pipeline proposed in this work is experimented on a server with Nvidia A40 GPU, 500GB RAM and 7.7TB disk space. The language models were trained on the dataset first to classify whether the review is useful or not. This is considered as preliminary filtering. Based on the predictions from the preliminary filtering, a new dataset is created for the secondary classification. This dataset is a subset of the first dataset that contains only the useful reviews classified by the language models. The second dataset is further subject to sentence level classification using the same set of language models. The training phase of all the language models for both preliminary and secondary classification was performed 5 times and the average values for the evaluation metrics are reported. The evaluation metrics used for both the tasks are precision, recall, accuracy and F1 score. Additionally, the training time required for each model on the dataset is also recorded. To maximize the performance of each language model, hyperparameters of each model were tuned and the best performing model’s hyperparameters are specified in the table below.

The hyperparameter settings for the language models used are displayed in Table I. Before the final step, discriminative learning is implemented on top of the encoder generated in which one layer from the last is unfrozen in each iteration.

Finally, all the layers are completely unfrozen, and the model is fine-tuned one last time. Using the encoder, the learning rate and number of epochs for each iteration are mentioned below:

- First iteration (Last layer is unfrozen) - Epoch = 4, Learning rate = 5e-2
- Second iteration (Last layer and Penultimate layers are unfrozen) - Epoch = 4, Learning rate = 1e-3
- Third iteration (Third last, Penultimate, and last layers) - Epoch = 4, Learning rate = 5e-3
- Fourth iteration (All layers are unfrozen) - Epoch = 10, Learning rate = 1e-3

The evaluation metrics used for comparing the performance of language models are defined below:

- **Precision:**
  Precision is the ratio between the True Positives and all the Positives
- **Recall:**
  The recall is the measure of our model correctly identifying True Positives
- **Accuracy on test data:**
  Accuracy is the number of correctly predicted data points out of all the data points in the test data
- **Time taken for training:**
  The time taken for a model to complete its training phase.

While implementing the language models and tuning them with an aim of maximizing the performance of the models, the goal was to maximize the recall as much as possible. The reason is because a low recall results in a high number of False positives. Having many False positives in this case means that the model has failed to identify requirements from review sentences. This is more dangerous when compared to a case where the precision is less where the model has wrongly predicted a sentence as a requirement. While the wrong requirement can be identified in the latter, the requirements that were missed by the model in the former case cannot be identified. This is the reason why recall is especially important in this application of language models.

Table II below show the performance of BERT, ELMO and ULMFit on classifying requirements from online user feedback. The first table represents the performance of the models on the preliminary filtering and the second table represents the performance on the model on secondary filtering.

Based on the results from Table II, it can be observed that BERT performed the best when compared to ELMO and ULMFit with a recall of 92% and overall F1-score of 88%. It can also be understood that BERT also managed to commit multiple False Positive (FP) which is the reason for BERT's precision to be similarly poor when compared to ELMO and ULMFit.

TABLE I. HYPERPARAMETER SETTINGS

	BERT	***ELMO***	***ULMit***
Epochs	20	25	10
Learning rate	2e-5	2e-4	1e-3
Epsilon	1e-7	N/A	N/A
Batch Size	16	16	16

TABLE II. PRELIMINARY FILTERING - RESULTS

	BERT	***ELMO***	***ULMFit***
Precision	0.85	0.82	0.82
Eval set accuracy	0.91	0.82	0.84
Recall	**0.92**	0.82	0.83
F1-score	0.88	0.81	0.83
Train time (s)	108	186	203

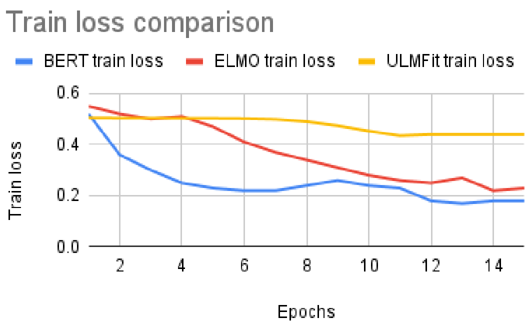

Fig. 3. Train loss comparison of BERT,ELMO and ULMFit

TABLE III. SECONDARY FILTERING - RESULTS

	BERT	***ELMO***	***ULMFit***
Precision	0.92	0.82	0.8
Eval set accuracy	0.9	0.82	0.79
Recall	**0.89**	0.83	0.78
F1-score	0.9	0.83	0.79
Train time (s)	110	147	190

Fig.3 depicts the train loss of all the three language models implemented for classification. It can be observed from the diagram that the learning curve of the language model BERT is better when compared to the other models that have the learning curve to be relatively more linear in nature. It is important to note that this corresponds with the high F1-score by BERT in both the classifications.

Based on the experiments performed, we can answer the research questions of this work. The answer to RQ1 is that the Kyoryoku process cannot be *completely* automated due to lack of labelled dataset for multi class classification. To answer

RQ2, experimentation involves three state-of-the-art language models, and it can be concluded that BERT performs better when compared to the other two language models.

In order to validate the statistical different empirically, statistical tests are further performed, and the results are discussed in the next section.

VI. STATISTICAL ANALYSIS

Statistical analysis is performed to empirically proven the performance difference of different language models used for classification. Therefore, Friedman test is performed over the F1-scores of each language model. More specifically, since each model was trained 5 times on the data and the average F1-score was reported, the actual F1-scores that was obtained in each training iteration is used to perform Friedman test. The performance of each language model over 5 training iterations is shown below. For this case, the hypothesis of Friedman test is as follows:

NULL HYPOTHESIS: All language models perform similar to each other in preliminary classification

ALTERNATE HYPOTHESIS: All language models perform significantly different from each other

TABLE IV. F1-SCORE OVER 5 TRAINING ITERATIONS

Iteration	***BERT***	***ELMO***	***ULMFit***
1	0.86	0.82	0.82
2	0.88	0.82	0.84
3	0.87	0.82	0.82
4	0.88	0.81	0.83
5	0.85	0.83	0.83
P-Value	**.01925**		
Statistic Value	**7.9**		

Based on the results from Table IV, it can be concluded that the null hypothesis can be reject at 95% level of significance. Therefore, the alternative hypothesis holds true in this case. This means that BERT, ELMO and ULMFit perform significantly different from each other.

Similar to the first hypothesis testing, another set of hypotheses was considered for the secondary classification and the hypothesis are as follows:

NULL HYPOTHESIS: All language models perform similar to each other in performing secondary classification

ALTERNATE HYPOTHESIS: All language models perform significantly different from each other

The results from the second hypothesis testing are shown in Table V below.

TABLE V. F1-SCORE OVER 5 TRAINING ITERATIONS

Iteration	***BERT***	***ELMO***	***ULMFit***
1	0.9	0.83	0.79
2	0.91	0.83	0.82
3	0.90	0.84	0.82
4	0.89	0.83	0.80
5	0.9	0.83	0.80
P-Value	**. 0067**		
Statistic Value	**10**		

From Table V, it can be concluded that similar to the previous hypothesis testing, null hypothesis can be rejected at 95% level of significance. Therefore, BERT, ELMO and ULMFit perform significantly different from each other in which BERT performs the perform with ~90% F1-score in the secondary classification task.

VII. CONCLUSION & FUTURE SCOPE

In this research work, an automated NLP pipeline that leveraged the concept of transfer learning from language models to automatically extract requirements from online user feedback. More specifically, the Kyoryoku process of manually extracting requirements was partially automated to the point where the pipeline can identify not only the useful reviews but also filter the sentences that represent the requirements. These requirements can further be used to either improve an already existing software engineering project or be used as a basis for building features on a new project. An empirical comparison of three state-of-the-art language models was performed and it was also concluded that BERT performed the best out of all algorithms with an average F1-score of 90% for filtering useful reviews and an average F1-score of 87% for filtering requirement sentences from useful reviews. The advent of language models in the field of NLP have had a significant impact in the automation process that could've not been imagined before. Moving forward, this work can be extended further by exploring latest language models such as XLNET which has gained some attention in recent times. In the future, this work can form a basis to completely automate the Kyoryoku process for requirement extraction. With the data being scarce in the field of Software engineering, it will be interesting to perform unsupervised clustering techniques to cluster the requirements into different classes.

REFERENCES

[1] Ivan, Garcia & Pacheco, Carla & Reyes, Myriam. (2018). Requirements elicitation techniques: A systematic literature review based on the

maturity of the techniques. IET Software. 12. 365-378. 10.1049/iet-sen.2017.0144. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Chuanyi Li, Liguo Huang, Jidong Ge, Bin Luo, Vincent Ng, Automatically classifying user requests in crowdsourcing requirements engineering, Journal of Systems and Software, Volume 138, 2018, Pages 108-123, ISSN 0164-1212,

[3] T. Gemkow, M. Conzelmann, K. Hartig and A. Vogelsang, "Automatic Glossary Term Extraction from Large-Scale Requirements Specifications," 2018 IEEE 26th International Requirements Engineering Conference (RE), 2018, pp. 412-417, doi: 10.1109/RE.2018.00052.

[4] Ferrari, A., Esuli, A. An NLP approach for cross-domain ambiguity detection in requirements engineering. Autom Softw Eng 26, 559–598 (2019). https://doi.org/10.1007/s10515-019-00261-7

[5] Mahmoud, A., Williams, G. Detecting, classifying, and tracing non-functional software requirements. Requirements Eng 21, 357–381 (2016). https://doi.org/10.1007/s00766-016-0252-8M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6] van Vliet M., Groen E.C., Dalpiaz F., Brinkkemper S. (2020) Identifying and Classifying User Requirements in Online Feedback via Crowdsourcing. In: Madhavji N., Pasquale L., Ferrari A., Gnesi S. (eds) Requirements Engineering: Foundation for Software Quality. REFSQ 2020. Lecture Notes in Computer Science, vol 12045. Springer, Cham. https://doi.org/10.1007/978-3-030-44429-7_11

[7] Mengmeng Lu and Peng Liang. 2017. Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17). Association for Computing Machinery, New York, NY, USA, 344–353. DOI:https://doi.org/10.1145/3084226.3084241

[8] Stanik, Christoph & Haering, Marlo & Maalej, Walid. (2019). Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning. 220-226. 10.1109/REW.2019.00046. Mahmoud, A., Williams, G. Detecting, classifying, and tracing non-functional software requirements. Requirements Eng 21, 357–381 (2016). https://doi.org/10.1007/s00766-016-0252-8M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[9] T. Hey, J. Keim, A. Koziolek and W. F. Tichy, "NoRBERT: Transfer Learning for Requirements Classification," 2020 IEEE 28th International Requirements Engineering Conference (RE), 2020, pp. 169-179, doi: 10.1109/RE48521.2020.00028.

[10] E. Guzman, M. Ibrahim and M. Glinz, "A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution," 2017 IEEE 25th International Requirements Engineering Conference (RE), 2017, pp. 11-20, doi: 10.1109/RE.2017.88.

[11] Leonidis, Asterios & Baryannis et al. Tesseris, George & Voskakis, Emmanouil & Bikakis, Antonis & Antoniou, Grigoris. (2009). AlertMe: A Semantics-Based Context-Aware Notification System. 2. 200 - 205. 10.1109/COMPSAC.2009.134.

[12] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv:1810.04805 [cs.CL]

[13] Deep contextualized word representations, Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer.NAACL 2018

[14] Universal Language Model Fine-tuning for Text Classification arXiv:1801.06146 [cs.CL]

[15] One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling ,arXiv:1312.3005 [cs.CL]