

# Learning to Categorize Bug Reports with LSTM Networks

Kaushikkumar D. Gondaliya

Jan Peters

Elmar Rueckert\*

Intelligent Autonomous Systems Lab  
 Technical University Darmstadt, Germany  
 email:kaushik.gondaliya1@gmail.com

Technical University Darmstadt  
 Max-Planck Institute Tuebingen  
 Darmstadt and Tuebingen, Germany  
 email:mail@jan-peters.net

Robotics and Cognitive Systems  
 University of Luebeck, Germany  
 \* Corresponding author

email:rueckert@rob.uni-luebeck.de

**Abstract**—The manual routing of bug reports to specialized expert teams is a time-consuming and expensive process. In this paper, we investigated how this process can be automated by training deep networks and state-of-the-art classifiers from thousands of real bug reports from a software company. Different combinations of the natural language processing methods *lemmatization*, *pos tagger*, *bigram* and *stopword removal* were evaluated in the classification algorithms *Linear Support Vector Machines* (SVMs), *multinomial naive Bayes*, and *Long Short Term Memory* (LSTM) networks. For feature processing we used the *Term Frequency-Inverse Document Frequency* (TF-IDF) method. Best results were obtained with a combination of the *bigram* method and linear SVMs. Similar prediction performance values were observed with LSTM networks that however promise to improve further with larger datasets. The bug triage tool was implemented in a microservice architecture using docker containers which allows for extending individual components and simplifies applications to other text classification problems.

**Keywords**—classification of text; bug reports; natural language processing; long short term memory networks; support vector machines.

## I. INTRODUCTION

The demands of quality software products have rapidly increased and a significant amount of cost is spent on support and maintenance. Although various testing methods are used to ensure a high quality of a software, it is almost never perfect and needs to be maintained continuously. As a result software developers and teams of experts are often confronted with a stream of service requests or bug reports [1, 2]. By a bug report, we refer to code errors or misbehaviour of a software based on an error in a computer program.

These reports have to be resolved by a selected team or expert. Due to the complexity of the reports, most software companies rely on human experts to assign them. This is an expensive task where for example the company who provided us with the data for this research received more than 6.8 million service requests for a single product in 2013. In a study of the *Eclipse project* conducted by Anvik et al. [3], it was found that on average 37 bugs are submitted per day. Moreover, three person-hours per day are required for performing the bug report assignment manually. Therefore, an automatic and efficient bug tracking tool is required to reduce human efforts or to make a bug tracking process less time consuming.

Existing learning approaches applied to such problems are based on training support vector machines or naive Bayes classifiers [4, 5]. While both approaches were shown to

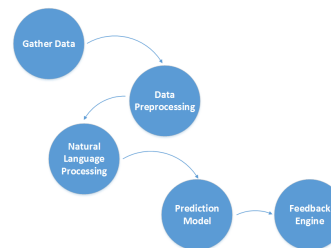


Figure 1. **Overview of the Framework:** Five docker containers are used for automated bug tracking with LSTM networks.

produce good results on small datasets [6] or in binary classification tasks [7], their application to large datasets or to online learning tasks is limited. For this problem domain, deep learning algorithms such as *Long Short Term Memory* (LSTM) networks [8, 9] are promising alternatives which have not been used for classifying bug reports so far. We choose LSTMs because they can be trained from small datasets with less than thousand samples in contrast to alternative deep learning approaches like [10]. This benefit was shown in our evaluation.

In this paper, we compare state-of-the-art bug assignment approaches to approaches based on LSTM networks. We present thorough results on datasets of real bug reports and validate the models' predictions with feedback collected from experts responsible for resolving the bug reports. Our findings and models can be easily extended and transferred to other text categorization problems.

In the following section, we review related work on automated bug tracking systems. In Section III, we discuss the used methods and In Section IV, we evaluate them on two datasets created from the bug reports provided by a software company. We conclude in Section V.

## II. RELATED WORK

Previous research investigated the detection of bug reports [11, 12], bug prioritization [13, 14], bug categorization [4, 5, 6, 7] and severity [13, 15]. In this report, we only focus on categorization and briefly review related approaches.

An important related by Xuan et al. [16] studied automatic bug triage systems based on feature selection and decision trees on the *Mozilla* [17] and the *Eclipse* [18] data sets. A good overview over data preprocessing strategies and feature modeling techniques is given. This work is based on a prior study from [19]. The discussed natural language processing techniques goes beyond the basic methods used here. However,

in contrast, we focused on the learning of the classifiers and compared LSTM networks to state of the art naive Bayes and SVMs.

In [4], the authors proposed an automatic routing system to classify incoming bug reports. The goal was to develop a continuously running router with a low misclassification error. The authors gathered around 6000 reports from a large software system which were classified into eight different categories by human experts. They compared several classification approaches like *naive Bayes*, *Support Vector Machines* (SVMs), *classification trees* and *k-nearest neighbor classification*. Their empirical results showed that the probabilistic models, i.e., the *k-nearest neighbor classification* and the SVMs outperformed the others. Furthermore, the accuracy improvement with an increasing amount of training data. However, for natural language processing only the *stopword removal* and *stemming method* were evaluated.

In [5] SVMs were used to train classifiers from two datasets, i.e., the open-source *Eclipse* [18] and open *Firefox* [17] bug report collections. These datasets contained 8.655 and 9.752 bug reports. For the *Firefox* dataset, the developer who submitted the last patch was used for labelling the bug reports. For the *Eclipse* dataset, the developer's name was used for labelling the bug reports, one who marked the bug report as "resolved". In a case of duplicates, a name of the developer who resolved the original bug report was taken for Eclipse and Firefox. They achieved an accuracy of 64% for the *Firefox* dataset and a precision of 58% for the *Eclipse* dataset. However, they obtained only a recall value of 3% for the *Firefox* dataset and 10% for the *Eclipse* dataset. For feature selection only the *bag of words method* and the *stopwords removal* approach were evaluated.

In a related approach, [6] evaluated the approaches *naive Bayes*, SVMs, *Radial Basis Function* (RBF) networks and *Random Forest* on a dataset from *Mozilla* [17] containing 1.983 bug reports. They used a *Term Frequency - Inverse Document Frequency* (TF-IDF) weighting scheme for feature extraction. The best result obtained was a classification accuracy of 44.4%, a precision of 30% and a recall value of 28%.

In [7], the authors proposed using the TF-IDF feature extraction method in combination with a *naive Bayes* classifier. They evaluated their approach on a dataset containing 10.000 reports and classified them into security or non-security related reports. On this binary classification problem, the authors achieved an accuracy of 93.9% and a precision of 92.5%. In this work however, we focus on multi-class assignments and the results are not directly comparable.

[6] and [7] showed that the TD-IDF method improves the results for classification tasks. Therefore we build on these results and used the TD-IDF method as well. However, in addition we trained *Long Short Term Memory* (LSTM) networks [8, 9] and tested other natural language processing methods such as *lemmatization*, *pos tagger* and *bigram*.

### III. METHODS

In this section, we discuss the necessary parts of an automated bug assignment system. These parts are (i) natural language processing techniques that transform and normalize text into a machine learning friendly form, (ii) feature extraction approaches that convert text into vectors of numbers,

and (iii) classification algorithms. An overview of how these methods are used for computing predictions of bug report assignments is sketched in Figure 2.

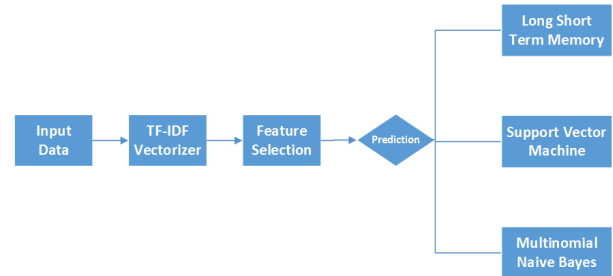


Figure 2. **Prediction model:** Features are generated from the raw text using the TF-IDF vectorizer method. Three classification algorithms, i.e., LSTM, SVM and *naive Bayes* as prediction model.

#### A. Natural Language Processing Methods

*Natural Language Processing* (NLP) is used to analyze, understand, and process meaning from human language. NLP is used to perform tasks, such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, or topic segmentation. Some of the most important NLP methods, which are also used in this study are discussed here. For a more detailed review, we refer to [19, 16].

*Tokenization.:* Text is treated as a string that is chopped into the pieces called tokens. For example, the string 'winter is coming' is tokenized into the terms 'winter', 'is', 'coming'.

*Word Boundary.:* It removes the extra white spaces or punctuation from the text. However, this removal entirely depends on the domain and for that reason in most cases regular expressions are used. For example, punctuation in terms like in 'M.Sc.' might contain information depending on the context.

*Lemmatization or Stemming.:* The document could consist the same words in many forms such as infection, derivation, etc. Due to grammatical reasons *lemmatization* and *stemming* are used to convert different word variants into similar canonical forms. For example, the different forms of the words 'work', 'works', 'worked' and 'working' share a same stem 'work'.

*Stopword Removing.:* Commonly used words like 'a', 'the', 'of' etc. normally do not contain any meaningful information. So it is beneficial to remove these words from the text.

*Part-of-Speech Tagging.:* The *POS tagger method* reads the texts from the document and applies labels like 'noun', 'verb', etc. to them. For example, the string 'Heat water in a large vessel' will create the tags '(heat,VB), (water,NN), (in,IN), (a,DT), (large,JJ), (vessel,NN)'.

*N-gram.:* Sometimes, a group of words is more beneficial than just a single word. Here, N is the number of words in a group. When N=1 the approach is called unigram, with N=2 it is called bigram, and with N=3 it is called trigram. An example for a bigram of the string 'winter is coming' is 'Winter is', 'is coming'.

We evaluated two basic pre-processing strategies commonly used in the literature [19, 16]. These two approaches

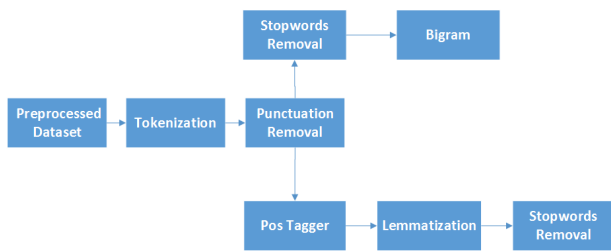


Figure 3. **Features:** We evaluated two sequences of text pre-processing techniques commonly used in text categorization.

are shown in Figure 3. The first strategy terminates the pre-processing with the *bigram* method. In the second sequence of applied NLP techniques, we end with the *stopword removal* method. For the remaining part of this paper, we will denote words in the pre-processed text by the variable  $w$ .

### B. Feature Extraction

In NLP, the most widely used extraction methods are the *Term Frequency - Inverse Document Frequency* (TF-IDF) [20], the *word 2 vector* method [21] and the *count vectorizer* approach [22]. We used the TF-IDF approach which was shown to outperform the other techniques in related applications [6, 7].

The term *TF* in the TF-IDF approach denotes the frequency count of keywords  $t$  which is weighted by its importance denoted by the term *IDF*. Thus, a feature in the TF-IDF approach results from the product of the TF count with the IDF weight, i.e.,  $\text{TF-IDF}(t) = \text{TF}(t, d) \cdot \text{IDF}(t)$ . The term-frequency TF is defined as

$$\text{TF}(t, d) = \sum_{w \in d} f(w, t),$$

where  $t$  denotes a keyword and  $w$  a word in text  $d$ . The function  $f$  returns 1 for a match ( $t = x$ ) and zero otherwise. Now, given a dataset of  $K$  documents  $D = \{d_1, d_2, \dots, d_K\}$ , the inverse document frequency (IDF) is defined as

$$\text{IDF}(t) = \log \frac{K}{1 + |\{d : t \in d\}|}.$$

Here the term  $|\{d : t \in d\}|$  denotes the cardinality of documents, i.e., when the condition  $\text{TF}(t, d) \neq 0$  is satisfied. Note that 1 is added to the denominator to avoid a divide by zero situation.

Finally, for each bug report in dataset  $D$  a feature vector for classification  $\mathbf{x} = [x_1, x_2, \dots, x_N]^T$  is computed by evaluating  $x_n = \text{TF-IDF}(t)$  for each of the selected keywords denoted by  $t$ .

### C. Classification Algorithms for text

In this section, we discuss the commonly used *naive Bayes* classifier, *Linear Support Vector Machines* and *Long Short Term Memory* (LSTM) networks for text classification.

1) *Naive Bayes Classifier:* The *naive Bayesian classifier* is a generative linear model based on the *Bayes theorem* [23, 24]. Important to note is that it relies on the assumption that all the features are independent.

In the context of text classification, the probability that text  $T$  belongs to a class  $C$  can be expressed as  $P(C|T)$ , where

$$P(C|T) = \frac{P(C)P(T|C)}{P(T)}.$$

The distribution  $P(C)$  denotes the prior probability of class  $C$ ,  $P(T)$  is the prior probability of text  $T$  and  $P(C|T)$  is posterior probability that we need to compute. Given vectorized feature representations of text of the form  $\mathbf{x} = [x_1, x_2, \dots, x_N]^T$  the above equation can be interpreted as:

$$\begin{aligned} P(C|T) &= P(C|\mathbf{x}), \\ &= P(C|x_1, x_2, \dots, x_N), \\ &= \frac{P(C)P(x_1, x_2, \dots, x_N|C)}{P(x_1, x_2, \dots, x_N)}. \end{aligned}$$

Assuming independent features for a given class the posterior distribution above factorizes to  $P(C|T) \propto P(C) \prod_{k=1}^N P(x_k|C)$ .

For a new document in a test dataset, we can compute class labels using the maximum a posteriori decision rule [25],

$$C_{map} = \underset{x \in C}{\text{argmax}} P(C) \prod_{k=1}^N P(x_k|C),$$

where  $P(C)$  is the prior probability of class which can be estimated as follows:

$$P(C) = \frac{\# \text{ of instances in this class}}{\# \text{ of instances in all classes}}.$$

The presented *naive Bayes* approach is used for multi-class classification in our experiments.

2) *Linear Support Vector Machine:* SVMs are powerful and popular supervised learning approaches [26]. They can be used for both classification and regression problems though they are mostly used for classification problems. SVMs are applicable for both linear and nonlinear classification problems using kernels. However, according to related work, see for example the discussion in [20], linear support vector machines are sufficiently powerful for text classification problems.

For samples  $\mathbf{x}_k$ , with class labels  $c_k \in \{-1, 1\}$  for  $k = 1, \dots, N$ , we compute a hyperplane which satisfies  $\mathbf{v} \cdot \mathbf{x}_k + b = 0$ . Here,  $\mathbf{v}$  is a vector orthogonal to the hyperplane and  $b$  is a perpendicular distance of the hyperplane to the origin. The canonical hyperplane is observed when the condition  $c_k(\mathbf{v} \cdot \mathbf{x}_k + b) \geq 1 \forall k$  is met.

For classification problems with more than two categories the most frequent approaches are the *One-vs.-Rest* method and the *One-vs.-One* method. In this paper, the *One-vs.-Rest* classifier is used for multi-class classification.

### D. Long Short Term Memory Networks

LSTMs are recurrent neural networks which were proposed by Hochreiter et al. [8] to overcome the vanishing gradient problem. The ability to capture temporal correlations over long time horizons was exploited in many speech processing and deep learning applications [9]. For text categorization however, we are not aware of any study using LSTM networks for computing multi-class label predictions.

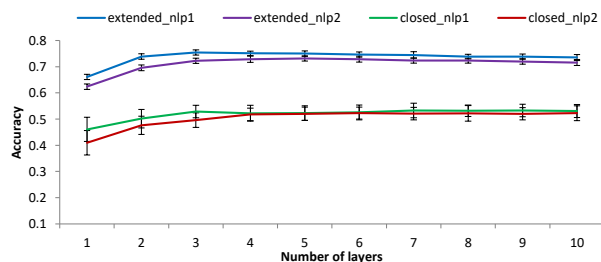


Figure 4. We tested different numbers of hidden layers for two pre-processing strategies. Results are shown for a small dataset of 1215 bug reports and on an extended one with 7346 reports.

For text classification, the LSTM network is trained on a sequence of feature vectors  $x_1, x_2, \dots, x_N$  that update the internal hidden state denoted by  $h_k$ . The corresponding output is trained to fit  $c_k$ . In this paper, we used sigmoid activation functions for the input layer. We optimized for the optimal number of hidden layers, i.e., 3 and 6 layers for the two evaluated datasets as shown in the right panel in Figure 4.

For the experiments in this paper, we used an open-source framework implementation (*Keras*) with a training batch size of 32. The neural network model had two hidden layers with 64 neurons in the first layer and seven units with soft-max activation functions in the output layer. A dropout regularization of 25% was used to reduce the models complexity and to prevent over-fitting. For training the *categorical cross entropy loss* function was used.

#### IV. EXPERIMENTAL RESULTS

In this section, we present results of the classification models on two datasets created from the bug reports provided by a software company. We prepared a small dataset with 1215 reports and a large dataset with 7346 samples, which is illustrated in Figure 5. The bug reports in the small dataset were already processed by the expert teams and had therefore labels or class assignments. For the large dataset we had to generate labels. We denote this manual labelling process by the *keyword algorithm* for the remainder of this paper.

*Manual data labelling:* To generate a larger dataset, a *keyword algorithm* was used to manually generate labels. As first step, experts assigned keywords to just submitted and therefore unprocessed bug reports. The algorithm looks for matches to fixed sets of keywords dedicated to each of the specialized support teams. Labels were thereafter created based on the maximum number of occurrences of experts' keywords. In case of equal counts, a random team assignment was generated.

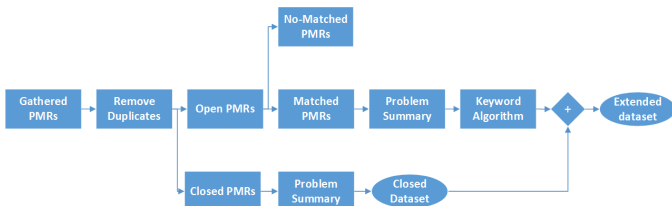


Figure 5. **Data preprocessing:** The execution of a process on gathered data results in 'closed' and 'extended' dataset.

We first present results evaluating the effect of the number of features. Subsequently we discuss the effect of different

combinations of pre-processing and classification techniques in the two datasets. We conclude by verifying the assumption of the creation of the additional labels for the large dataset. For that human experts rated the predictions of our models. All presented statistics were obtained through running 100 experiments with 79% of the samples randomly selected as training set and 21% as test set.

##### A. The effect of the number of used features

We used the *Term Frequency-Inverse Document Frequency* (TF-IDF) method which is discussed in Section III-B for feature selection. In Figure 6, we show the classification performance values for the three classification methods SVM, LSTM and naive Bayes (NB) for an increasing number of used features. The *stopwords removal* and the *bigram* methods were used for the pre-processing of the bug reports.

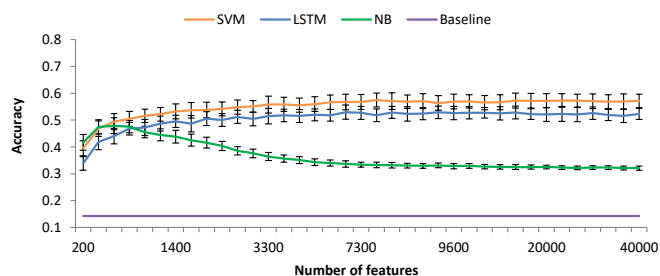


Figure 6. Evaluation of the effect of the used number of features in three classification methods. The baseline accuracy for a 7-class classification problem is 1/7. The classifiers were trained on the small dataset.

According to Figure 6, we observe that a linear SVM outperforms the *multinomial naive Bayes* and the LSTM network in general. The best accuracy values for *multinomial naive Bayes*, linear SVM and the LSTM network obtained were 0.479, 0.574 and 0.529 with 600, 7900 and 6700 features respectively.

##### B. Comparison of the Prediction Models

On the small dataset all three classification methods correctly classified 47.9 → 57.2% of the test samples. Note that these results are significantly better than random assignments (14.3% for seven classes). For the large dataset the performances ranged from 68.6 → 77.6%. For this experiments the optimal number of features for each classifier was used as evaluated in the previous subsection.

We also evaluated the effect of two commonly used pre-processing approaches that were discussed in Subsection III-B. With the term 'nlp1' we denote the application of the *stopwords removal* and the *bigram* methods and with the term 'nlp2' we denote the the combination of the *pos tagger*, *lemmatization* and *stopword removal* methods. As shown in Figure 7, no significant difference in the classification performance could be found. However, the orchestration of methods denoted by 'nlp2' is favoured because of computational reasons.

##### C. Verification of the automatically labeled bug reports

The 'large dataset' relied on the assumption that correct labels could be automatically created using a keyword algorithm. The keywords were provided by experienced support engineers who were also asked to validate the predictions of our three trained classifiers. A feedback engine was implemented and

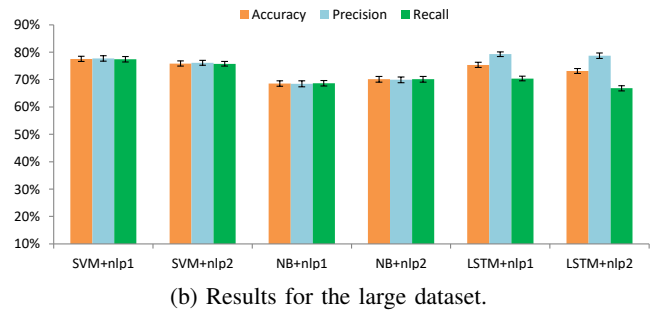
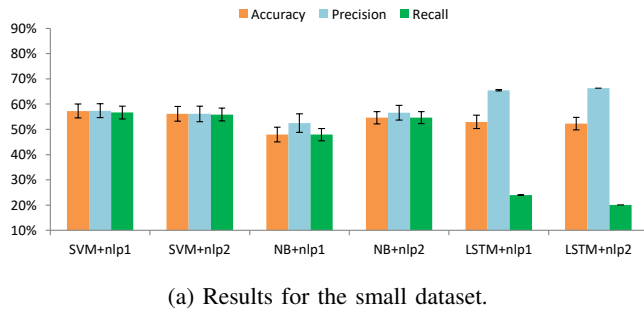


Figure 7. Shown are the classification performances for the combination of *linear SVM*, *multinomial naive Bayes*, and *LSTM* networks with different pre-processing approaches. The terms 'nlp1' and 'nlp2' denote two commonly used preprocessing strategies, see the text for details.

the experts rated our models' predictions of additional 132 bug reports which were not used for training.

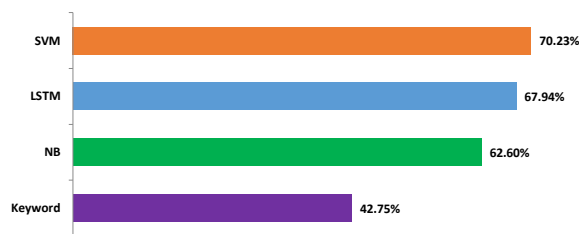


Figure 8. The experts' feedback values for the *linear SVM*, *multinomial naive Bayes*, and *LSTM* networks and 'keyword algorithm'.

The results are shown in Figure 8. For SVMs 70.23% of these new reports were correctly assigned to one of the seven teams. For LSTMs 67.94% of the assignments were correct. The naive Bayes approach generated 62.6% of correct labels. Note that the predictions of the keyword algorithm could only classify 42.75% of the reports correctly. These predictions and results were perceived as very helpful and will save substantial resources in the future for performing bug report assignments.

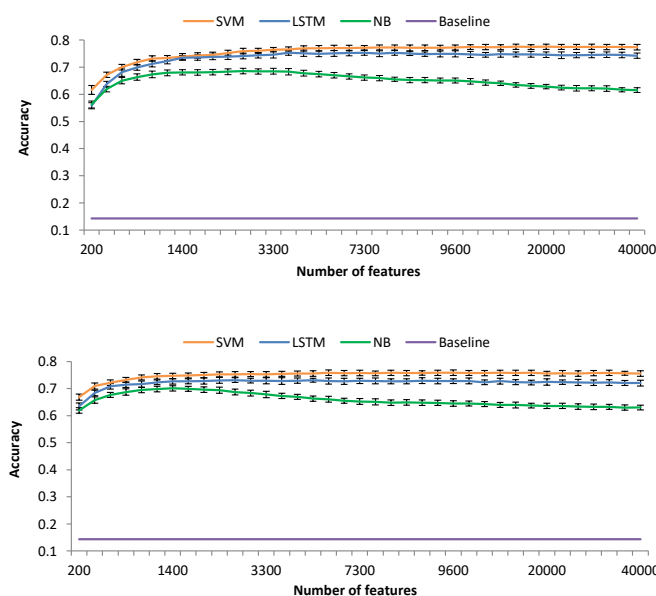


Figure 9. The prediction accuracy improves with the number of used features. Left: 'nlp1' and right 'nlp2' preprocessing approach.

#### D. Specific Feature Selection.

We also evaluated accuracy values for different number of features by applying *Term Frequency-Inverse Document Frequency* (TF-IDF) method. For the 'large dataset' the results are shown in Figure 9.

#### V. CONCLUSION

Every year software companies receive millions of service requests that have to be resolved by specialized expert teams. The assignment of the requests is traditionally done by human experts. First attempts in automatic routing of service requests are based on training *Support Vector Machines* (SVMs) and *naive Bayes* multi-class classifier [4, 3, 7]. However, these approaches are computationally and memory demanding for large datasets and thus of limited practical use for large software companies.

In this work, we investigated the performance of *Long Short Term Memory* (LSTM) networks which can be trained from millions of samples. We evaluated different combinations of natural language processing methods such as *lemmatization*, *pos tagger*, *N-gram* and *stopword removal* and tested different numbers of features ranging from 200 to 40.000. We found that for small datasets with 1215 reports, SVMs achieved the best classification performance. Out of 256 bug reports of a test set,  $57.2 \pm 0.028\%$  were correctly assigned to one of the seven expert teams. In contrast with LSTMs only  $52.9 \pm 0.026\%$  were correctly assigned. Here, the  $\pm$  symbol denotes the standard deviation computed from 100 runs.

Interestingly, with an increasing number of training samples LSTM networks achieved similar classification results. This was shown in training from a larger dataset with 7346 samples. Here SVMs correctly classified  $77.6 \pm 0.009\%$  and LSTM  $75.3 \pm 0.009\%$ . Given that LSTM networks were used with millions of samples in deep learning approaches we expect them to outperform SVMs with larger datasets. This assumption will be tested in near future as the framework is constantly used to collect new service requests, i.e., during the time writing this report 2000 additional request were obtained. Moreover, we plan to compare our pre-processing methods, feature extraction strategies and deep network classifier to different text categorization problems.

## ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements No 713010 (GOAL-Robots) and No 640554 (SKILLS4ROBOTS). The authors also thank the software company who provided the data.

## REFERENCES

- [1] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," *Procedia Computer Science*, vol. 70, pp. 632 – 639, 2015.
- [2] B. Beizer, *Software Testing Techniques*. Dreamtech, 2003. [Online]. Available: <https://books.google.co.in/books?id=Ixf97h356zcC>
- [3] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 937–940. [Online]. Available: <http://doi.acm.org/10.1145/1134285.1134457>
- [4] G. A. Di Lucca, M. Di Penta, and S. Gradara, "An approach to classify software maintenance requests," in *Software Maintenance, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 93–102.
- [5] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 361–370. [Online]. Available: <http://doi.acm.org/10.1145/1134285.1134336>
- [6] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine," in *Proceedings of the 2009 Fourth International Conference on Software Engineering Advances*, ser. ICSEA '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 216–221. [Online]. Available: <http://dx.doi.org/10.1109/ICSEA.2009.92>
- [7] D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, Feb 2014, pp. 294–299.
- [8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>
- [9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.
- [10] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.
- [11] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 461–470. [Online]. Available: <http://doi.acm.org/10.1145/1368088.1368151>
- [12] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *Proceedings of the 2010 Asia Pacific Software Engineering Conference*, ser. APSEC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 366–374. [Online]. Available: <http://dx.doi.org/10.1109/APSEC.2010.49>
- [13] K. Chaturvedi and V. Singh, "Determining bug severity using machine learning techniques," in *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*. IEEE, 2012, pp. 1–6.
- [14] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 495–504. [Online]. Available: <http://doi.acm.org/10.1145/1806799.1806871>
- [15] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, May 2010, pp. 1–10.
- [16] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE transactions on knowledge and data engineering*, 2014.
- [17] I. mozilla.org contributors, "Bugzilla dataset," <https://bugzilla.mozilla.org/>, 19982017, retrieved: Aug., 2018.
- [18] T. E. Foundation, "Eclipse dataset," <http://www.eclipse.org>, retrieved: Aug., 2018.
- [19] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 25–35.
- [20] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, ser. ECML '98. London, UK, UK: Springer-Verlag, 1998, pp. 137–142. [Online]. Available: <http://dl.acm.org/citation.cfm?id=645326.649721>
- [21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, vol. 14, 2014, pp. 1532–43.
- [22] A. Tripathy, A. Agrawal, and S. K. Rath, "Classification of sentimental reviews using machine learning techniques," *Procedia Computer Science*, vol. 57, pp. 821–829, 2015.
- [23] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*. New York: John Wiley, 1973.
- [24] P. Langley, W. Iba, and K. Thompson, "An analysis of bayesian classifiers," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, ser. AAAI'92. AAAI Press, 1992, pp. 223–228. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1867135.1867170>
- [25] C. D. Manning, P. Raghavan, and H. Schütze, "Text classification and naive bayes," *Introduction to information retrieval*, vol. 1, p. 6, 2008.
- [26] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.