Poster: Predicting components for issue reports using deep learning with information retrieval

Morakot Choetkiertikul
University of Wollongong, and
Mahidol University
morakot.cho@mahidol.ac.th

Hoa Khanh Dam
University of Wollongong
hoa@uow.edu.au

Truyen Tran
Deakin University
truyen.tran@deakin.edu.au

Trang Pham
Deakin University
phtra@deakin.edu.au

Aditya Ghose
University of Wollongong
aditya@uow.edu.au

ABSTRACT

Assigning an issue to the correct component(s) is challenging, especially for large-scale projects which have are up to hundreds of components. We propose a prediction model which learns from historical issues reports and recommends the most relevant components for new issues. Our model uses the deep learning Long Short-Term Memory to automatically learns semantic features representing an issue report, and combines them with the traditional textual similarity features. An extensive evaluation on 142,025 issues from 11 large projects shows our approach outperforms alternative techniques with an average 60% improvement in predictive performance.

ACM Reference Format:
Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, and Aditya Ghose. 2018. Poster: Predicting components for issue reports using deep learning with information retrieval. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194952

1 INTRODUCTION

The *component* is the most important information in an issue report, which software engineers rely on in order to determine the location of a bug [1]. Assigning the correct component(s) to an issue would help expedite its resolution, e.g. locating the relevant parts in the software codebase which may need modifying. As a project evolves over time, the number of components also increases significantly. For example, the Moodle[1] project now has 89 components while larger projects may have hundreds of components (e.g. Atlassian's JIRA project has 169 components). This makes assigning new issues to the correct component(s) become a highly challenging task [2]. Hence, there is an important need to provide some (semi-) automated support in assigning components to a new issue [1, 3]. In this paper, we offer a means for providing this support in the form of a prediction machinery which takes as input the *textual* description of an issue (e.g. its title and description), and *recommends* a list of components that are most likely relevant to the issue. The prediction machinery is trained using a number of examples (i.e. issues with known components). The trained system is then used to automatically predict the component(s) for new issues. Our prediction system is referred to as a **Deep** learning model for **Soft**ware **C**omponent recommendation (***DeepSoft-C***).

[1]https://moodle.org

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194952

2 DEEPSOFT-C

DeepSoft-C takes as input the textual description (title and description) of a (new) issue, and recommends a list of k components[2] that are most relevant to the issue. The architecture of *DeepSoft-C* is provided in Figure 1. We leverages Long-Short Term Memory (LSTM), an highly effective method for modelling sequential data such as natural language text. The LSTM model can automatically learn and generate semantic features for representing an issue report. Moreover, textual similarity features (which are based on term frequencies) are however robust in bug/issue report classification. Hence, our model also makes use of those textual similarity features. For each issue report, we generate another set of features that reflect the similarity distance from the issue to each component label. Both semantic and textual similarity features are combined and fed into a simple single-layer neural network which has been modified to support multi-label classification. This classifier is trained using all issues for a given project in the training set.

2.1 Generating semantic features

LSTM can automatically train itself using many sequences of words in the corpus built from our dataset. During training, for every word in a sequence $\langle w_1, w_2, ..., w_n \rangle$, we know the true next word. We use this information to learn the model parameters which maximize the accuracy of our next word predictions. Once the training phase has been completed we use the learned LSTM model to generate an output state vector for any words in the corpus where it contains information from other words that come before. The semantic feature vector for the issue report is computed by aggregating all the word states in the same sequence (i.e. pooling) so that all information from the start to the end of the issue's description is accumulated.

[2]The number of top k components is specified by the user.

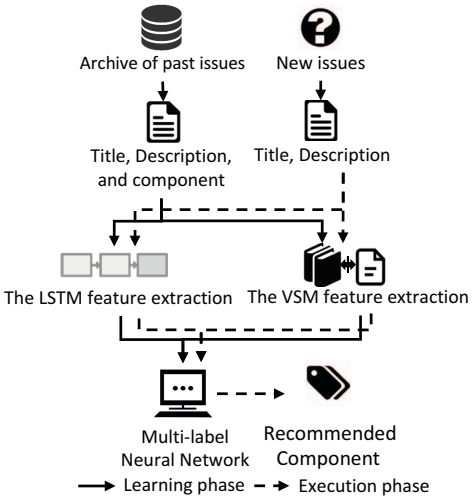

Figure 1: The architecture of *DeepSoft-C*

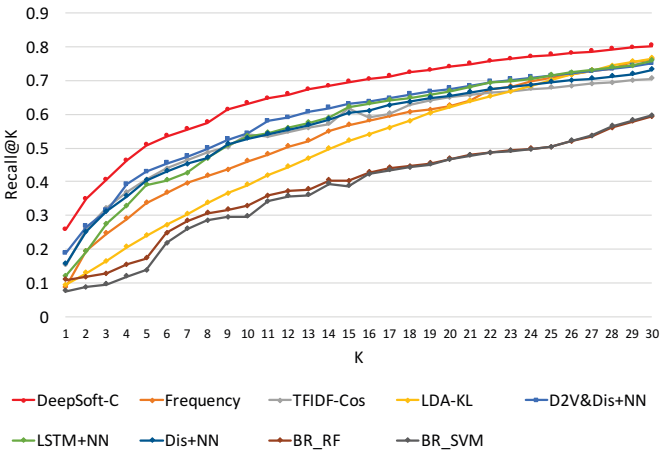

Figure 2: The performance of DeepSoft-C

2.2 Textual similarity features

The textual similar features of an issue report is generated as follows. First, for each component label, we collected the descriptions of all the issues (in the training set) which were assigned to that component. *We concatenate all these descriptions to form the description of a component.* The cosine similarity between component C_i and issue report S_j is then computed as: $Similarity(C_i, S_j) = \frac{\vec{V}_{C_i} \bullet \vec{V}_{S_j}}{\|\vec{V}_{C_i}\| \|\vec{V}_{S_j}\|}$ where $\vec{V}_{C_i}$ and $\vec{V}_{S_j}$ are a vector of term weights for the descriptions of C_i and S_j respectively.

2.3 Prediction model

Both semantic and textual similar features are combined into a single feature vector representation for an issue report. Our prediction model is built as a single hidden layer neural network which takes as input vector $\boldsymbol{x} = [x_1, x_2, ..., x_m]$. Assume that $\{C_1, C_2, ..., C_p\}$ are the set of available components. At the output layer, the model generates an output vector $\hat{\boldsymbol{c}} = [\hat{c_1}, \hat{c_2}, \hat{c_3}, ..., \hat{c_p}]$, where each $\hat{c_1}$ represents the probability of the issue report being assigned to component C_i. The model can be expressed as a function $\hat{\boldsymbol{c}} = f_\Theta(\mathbf{x})$, which is defined as: $f_\Theta(\mathbf{x}) = sigmoid(\mathbf{W}_o relu(\mathbf{W}_h x + \mathbf{b}_h) + \mathbf{b}_o)$ where $\Theta = \{\mathbf{W}_h, \mathbf{b}_h, \mathbf{W}_o, \mathbf{b}_o\}$ are parameters, and *sigmoid* is the activation function in the output layer, and *relu* (i.e. ReLU unit) is the activation function at the hidden layer.

3 EVALUATION RESULT

We evaluate our approach on 142,025 issues collected from 11 open source projects, namely HBase, HIVE, Infrastructure, Cordova, Hadoop, Fedora CloudSync, Islandora, JIRA software, Confluence, Bamboo, and Moodle. Figure 2 shows the evaluation result of our DeepSoft-C and the other methods: the Most Popular method (Frequency), the method using TF-IDF combined with cosine similarity score (TFIDF-Cos), the method using LDA combined with KL (LDA-KL), the combination of Doc2vec and the textual similarity features (D2V&Dis+NN), the method using only-LSTM features (LSTM+NN), the method using only the textual similarity features (Dis+NN), and the methods using Random Forests and Support Vector Machine in Binary Relevance (BR_RF and BR_SVM, respectively) using Recall@1 to 30, averaging across 11 projects. Overall, our DeepSoft-C consistently outperforms the others in all cases.

4 CONCLUSION AND FUTURE WORK

We have proposed a novel approach to recommend the most relevant components for an issue report. There are several important novel aspects in our proposal: automatic semantic feature learning, combining semantic features with traditional textual similarity features, and adapting a multi-label neural network classifier for component prediction. Our future work would involve expanding our study to commercial software projects and other large open source projects to further evaluate our proposed method. We would also like to evaluate the impact of our prediction system for recommending software components in practice by project mangers and/or software developers.

ACKNOWLEDGMENTS

This research project was partially supported by Faculty of Information and Communication Technology, Mahidol University. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

REFERENCES

[1] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM Press, New York, USA, 308–318.

[2] George Kakarontzas, Ioannis Stamelos, Stefanos Skalistis, and Athanasios Naskos. 2012. Extracting components from open source: The component adaptation environment (COPE) approach. In *Proceedings of the 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. 192–199. https://doi.org/10.1109/SEAA.2012.39

[3] Ashish Sureka. 2012. Learning to Classify Bug Reports into Components. *Objects, Models, Components, Patterns: 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings* (2012), 288–303. https://doi.org/10.1007/978-3-642-30561-0_20