Ticket Tagger: Machine Learning Driven Issue Classification

Rafael Kallis
Department of Informatics
University of Zurich
Zurich, Switzerland
email: rk@rafaelkallis.com

Andrea Di Sorbo, Gerardo Canfora
Department of Engineering
University of Sannio
Benevento, Italy
email: {disorbo, canfora}@unisannio.it

Sebastiano Panichella
School of Engineering
Zurich University of Applied Sciences
Zurich, Switzerland
email: panc@zhaw.ch

***Abstract*—Software maintenance is crucial for software projects evolution and success: code should be kept up-to-date and error-free, this with little effort and continuous updates for the end-users. In this context, issue trackers are essential tools for creating, managing and addressing the several (often hundreds of) issues that occur in software systems. A critical aspect for handling and prioritizing issues involves the assignment of labels to them (e.g., for projects hosted on GitHub), in order to determine the type (e.g., bug report, feature request and so on) of each specific issue. Although this labeling process has a positive impact on the effectiveness of issue processing, the current labeling mechanism is scarcely used on GitHub. In this demo, we introduce a tool, called Ticket Tagger, which leverages machine learning strategies on issue titles and descriptions for automatically labeling GitHub issues. Ticket Tagger automatically predicts the labels to assign to issues, with the aim of stimulating the use of labeling mechanisms in software projects, this to facilitate the issue management and prioritization processes. Along with the presentation of the tool's architecture and usage, we also evaluate its effectiveness in performing the issue labeling/classification process, which is critical to help maintainers to keep control of their workloads by focusing on the most critical issue tickets.**

Tool Webpage: https://github.com/rafaelkallis/ticket-tagger

***Index Terms*—Software maintenance and evolution, Issue Processing, Unstructured Data Labeling**

I. Introduction

In the life cycle of software projects, maintenance is a crucial task, which implies several activities. First of all, the source code should be kept up-to-date and any potential flaws in terms of performance and correctness removed. On the other hand, a maintainer must invest as little time and effort (e.g., in understanding the relevant software artifacts [12], [11]) as possible to solve the mentioned tasks to keep the software maintenance costs low [6].

Issue tracking systems are important means for maintainers to enable rigorous yet effective software evolution tasks. In issue tracking systems maintainers report tickets or potential problems, manage them and keep track of their progress. But as useful they might be, many developers still end up with a rapidly growing workload and lose control of it [2], [12].

Differently from more traditional and structured issue tracking systems, such as *Bugzilla* or *JIRA*, GitHub, one of the most popular social coding platform, provides an integrated lightweight issue tracking system, in which issue submitters are only required to provide a short textual abstract, containing a title and an optional description to report a new issue to a project hosted on GitHub. While this simplified process of reporting issues decreases the barrier to entry and attracts more inexperienced external contributors, it complicates the work of the development teams for maintaining the software, as several hundreds of issues of different nature (e.g., asking questions, proposing features, signaling bugs) and quality are usually submitted [5]. To cope with these problems, GitHub also offers a customizable labeling system, which can be used by developers to mark and manage issue reports. In particular, labels can give immediate clues about the issues (*e.g.,* what sort of topic the issue is about, what development task the issue is related to, or what priority the issue has) and are also useful for project administrators, since they can serve both as classification and filtering mechanism, thus facilitating the managing of the project [8]. However, manually assigning labels to issues is a labor-intensive and time-consuming task for project managers [5]. Indeed, although labeling has a positive impact on the effectiveness of issue processing [10], the labeling mechanism is scarcely used on GitHub [3], [2]. The goal of our work is to help maintainers improving their issue processing effectiveness, by automating the issue labeling process on GitHub. To pursue this goal, we developed a tool, called *Ticket Tagger*.

Previous studies presented several approaches to automatically categorize issues posted in bug tracking systems. For example, Antoniol *et al.* [1] showed that machine learning models can be used in order to discriminate bugs from other kinds of issues. Herzig *et al.* [7] introduced six different issue categories – *bug, feature request, improvement request, documentation request, refactoring request*, and *others* – and demonstrated that often developers and maintainers assign the wrong issue category to the reports. To address this problem, Zhou *et al.* [14] combined structured data with unstructured free-text data to train a classifier able to predict with high accuracy if a bug report is actually a bug or another kind of issue. Clearly, no structured information could be found on GitHub issues, according to the GitHub issue tracking lightweight structure. Our tool enables, with high effectiveness, the automated labeling of GitHub issues, this by relying exclusively on the text contained in issue titles and descriptions.

This is important for developers (that often take a lot of time to handle new issues [8]), as this approach automatically categorizes issues immediately after they are reported.

Thus, the contribution of this paper is two-fold:

- we release Ticket Tagger, a new GitHub app that can be integrated with any new or existing software repositories hosted on GitHub. When creating a new ticket on the repository's issue tracker, Ticket Tagger automatically assigns a relevant label to it, relying on the title and the description of the issue;
- we evaluate the issue classification effectiveness of Ticket Tagger. In particular, we conduct an assessment of the classification performance of the machine learning model implemented in Ticket Tagger, using a dataset that comprises about 30,000 issues extracted from heterogeneous GitHub projects.

Paper structure. The paper is organized as follows: Section II explains the approach behind Ticket Tagger, Section III illustrates the tool's architecture and its usage, while in Section IV we discuss the classification results achieved by the proposed approach.

II. Method

Ticket Tagger classifies issues by performing the following steps: (i) processing of title and body of the issues; (ii) vectorial representation of issues; (iii) classification of issues.

Choice of the Machine Learning Model. Different methods can be used to analyze issues information to determine/predict the label(s) to assign to them. Machine learning models require a lot of time for training and use a lot of memory. Ideally, we look for a model that achieves good results while remaining portable, i.e., is not memory and storage intensive, as we desire to deploy the model on low-end server hardware[1]. Linear classifiers often obtain near state-of-the-art text classification performance despite their simplicity[9], [4], [13]. To perform this task, we decided to use *fastText*, a tool that was open sourced by Facebook AI Research in 2016, which uses linear models with a rank constraint and fast loss approximation [9]. Joulin *et al.* [9] showed that whilst fastText's accuracy is competitive against several deep learning based models, it is several orders of magnitude faster for training and evaluation.

Tickets Pre-processing. The issues pre-processing steps strictly depends on the method selected to perform the automated classification of issues. The fastText linear classifier is typically trained with sentences represented as a bag of words and a bag of n-grams to partially embed information on word order. This because local word order information tends to improve the text classification performance of the linear model. Thus, each issue submitted to the issue tracker is passed to a fastText based classifier. We extract the title and body of the issue and concatenate them into a single textual paragraph. We then tokenize the resulting text and derive the bag of words representation from the tokenized text.

[1]AWS EC2 `t2.nano` (1 vCPU, 512 MB RAM, 20 GB SSD)

Vectorial Representation of Issues. From the extracted bag of words representation, each word is represented by a vector of *character n-grams*, which represents the desired input of fastText.

Issues Classification. To classify issues through fastText the main objective is to minimize the following objective function over N (possible tickets) labels:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n))$$

where x_n is a bag of features, A represents the weight dictionary of the average text embeddings, B is the weight dictionary that converts the embedding to pre-softmax values for each class, and f is the hierarchical *softmax function* used to minimize computational complexity. Due to the memory constraints of our server hardware, we do not capture word n-gram features in our model[2]. For the same reason, our model only contains words that occur at least 14 times in the dataset[3]. With a minimum word count of 14, the trained model requires less than 5 MB of disk space. For the setting of fastText we used the default values for all other parameters[4]. Most notably, this includes $2,000,000$ buckets, a minimum character n-gram length of 3, a sampling threshold of 0.0001, a learning rate of 0.05, word vector size of 100, context-window size of 5 and a *skipgram negative sampling* loss function with 5 negatives sampled.

It is important to mention that, currently, Ticket Tagger classifies issues with a multi-class classifier according to three categories describing the *intent* of the writer: *bug report*, *enhancement*, and *question*. We chose these labels as they are included by default in every issue tracker on GitHub repositories and they encode the vast majority of available issue tickets [3]. Clearly, the approach is designed to work for any label used on GitHub. Indeed, retraining a multi-class classifier does not require too much effort, making our model easily adaptable to specific projects' needs. In addition, a multi-class classifier can assign the more relevant label (*i.e.*, the one reflecting the main *intent* of the reporter) to any new opened issue, avoiding to assign multiple tags that may be contradictory [3]. When a developer submits a new issue to a software repository, Ticket Tagger is activated and a relevant label is automatically assigned to the new issue.

In Section IV we discuss our evaluation of the proposed approach, presenting the accuracy of the trained fastText model given a training-set consisting of pre-labeled issue tracker tickets found on GitHub.

III. Ticket Tagger in Action

As reported in Section I, our goal is to support developers during issue processing activities. We, therefore, integrated the pre-trained fastText model (see Section II) into a GitHub app,

[2]`wordNgram` parameter

[3]`minCount` parameter

[4]Complete list of parameters can be found at https://fasttext.cc/docs/en/options.html

namely Ticket Tagger, that we developed. The Ticket Tagger app is able to automatically (i) gather issues information from a GitHub repository, and (ii) assign labels to newly created issue tickets.

Ticket Tagger is based on Node.js, de-facto server-side javascript runtime, and uses the fastText implementation[5].

When started, our app exposes an endpoint which is called by a GitHub hook every time an issue is submitted on repositories Ticket Tagger was installed on. Fig. 1 illustrates a sequence diagram of the protocol. The flow is initiated when a user opens a new issue, after which a procedure is started on GitHub asynchronously. GitHub calls the hook endpoint exposed by Ticket Tagger, including a reference to the newly created issue ticket in the request body. Ticket Tagger then runs a prediction over the ticket. Before assigning the predicted label, temporary credentials for the issue ticket's repository are fetched. More specifically, an access token is issued from GitHub and it is consumed when finally assigning the predicted label to the issue ticket.

Our Ticket Tagger application is freely accessible to any developer and can be effortlessly integrated into existing GitHub repositories. Indeed, after going to the Ticket Tagger app page[6], the repository administrator has to simply click on the *"Install"* button and select the repository to integrate. When this is done, Ticket Tagger is successfully integrated with the specified repositories.

Fig. 2 depicts the standard tagging process. As mentioned above, each time a user submits a new issue on a software repository, the application is triggered and runs a classification. In Fig. 2, Ticket Tagger correctly identifies the issue ticket as a bug report and automatically assigns the *"bug"* label to it. Once labeled, the code owners can (i) immediately react to any urgent tickets, (ii) postpone less important issues such as feature or enhancements requests, or (iii) transmit questions to the responsible persons.

IV. Performance Evaluation

In this section, we assess the classification performance achieved by the fastText model integrated into Ticket Tagger

[5]https://github.com/facebookresearch/fastText/

[6]https://github.com/apps/ticket-tagger

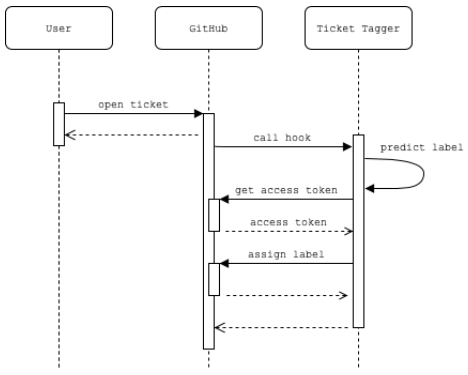

Fig. 1. Sequence diagram of Ticket Tagger.

TABLE I
Model Benchmark; precision, recall and F-measure of "bug", "enhancement" and "question" labels for the fastText model after a 10-fold cross validation.

	Bug	*Enhancement*	*Question*
Precision	82.2%	89.4%	78.1%
Recall	84.1%	76.3%	87.4%
F-measure	83.1%	82.3%	82.5%

(see Section II). To conduct our assessment, we collected a dataset containing 30,000 issues[7]: (i) 10,000 of them had the *"bug"* label assigned, (ii) 10,000 issues had the *"enhancement"* label assigned, and (iii) 10,000 issues had the *"question"* label assigned. Tickets with the *"enhancement"* label refer to improvements and new features. The dataset was collected by querying the GitHub Archive[8] using Google BigQuery[9]. Collected issues were drawn at random from the set of all GitHub issues closed during January 2018 and containing a label matching to one of the strings: "bug", "enhancement" or "question". More specifically, the issues were collected from $12,112$ heterogeneous projects. On average, 2.48 issues for each project (median $= 1$ and standard deviation $= 15.78$) were extracted.

We performed a 10-fold cross validation using the dataset containing the aforementioned pre-labeled issues. More precisely, the collected dataset was partitioned into 10 equal size subsamples. The cross-validation process was repeated ten times so that each time a different subsample was used as the testing set while the remaining nine subsamples were used as training data. During each iteration, we recorded the *precision*, *recall* and *F-measure* metrics achieved for each class. At the end of this process, we had exactly ten values for each considered metric and each issue category. Such results were then averaged in order to produce a single estimation for each considered metric and each category.

As shown in Table I, we observe that our fastText model obtained F-measure values above 80% for each considered label, confirming the practical usefulness of the proposed approach for improving the issue management practices on GitHub. However, while for issues belonging to the *Bug* class both precision and recall are above 80%, our model produces higher numbers of false positives for the *Question* category (*i.e.,* a lower precision is achieved for this class) and higher amounts of false negatives for the *Enhancement* class (*i.e.,* for this category a lower recall is achieved). Indeed, the strong use of function words (such as *how* or *what*, that typically introduce questions) in the issue title or description could lead the classifier to erroneously assign the *"question"* label to issues that actually belong to different classes and, consequently, this degrades the precision results achieved for the *Question* category. In addition, the lower recall obtained for the *Enhancement* class could be connected with the many different ways through which users (and developers)

[7]Raw data available at: https://tinyurl.com/y23kgdro

[8]https://gharchive.org

[9]https://cloud.google.com/bigquery

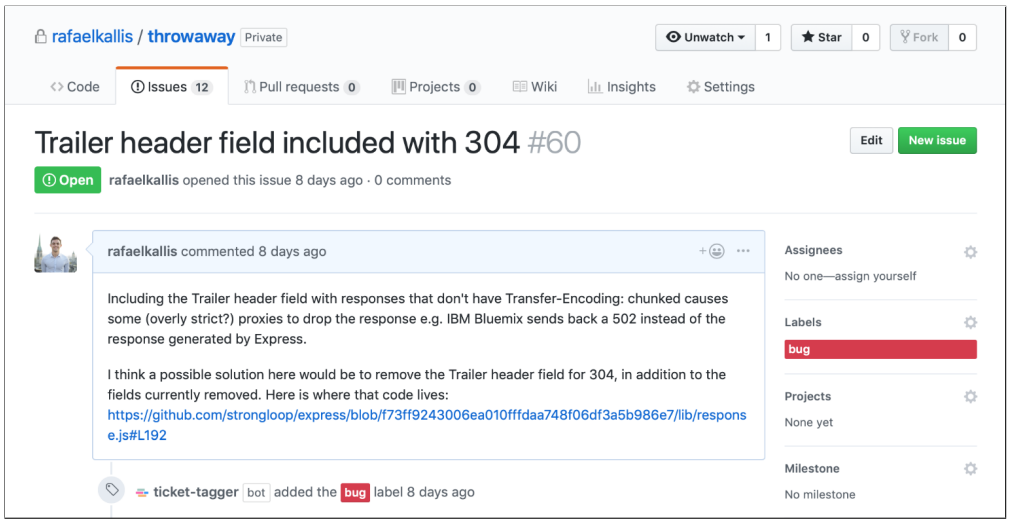

Fig. 2. Ticket Tagger automatically assigns the "bug" label to a newly created issue on GitHub.

request improvements or new features [13], making it hard to learn all the patterns that could lead to the assignment of this label.

V. Conclusion

In this work, we presented Ticket Tagger, an app that we released on the GitHub marketplace, that automatically assigns suitable labels to issues opened on GitHub projects. In particular, we presented the tool's architecture and provided all the necessary information for installing and using our app on custom GitHub repositories. Indeed, fellow developers who desire to improve the issue maintenance process through the automated classification enabled by the tool, can easily integrate Ticket Tagger into their repositories.

The core of Ticket Tagger is represented by a machine learning model that analyzes the title and the textual description of the issue in order to determine whether such issue can be labeled as a bug report, a feature request or a question. Thus, with the aim of assessing the classification performance achieved by our tool, we conducted an experimental evaluation of the model on a dataset comprising 30,000 issues. The results of such evaluation showed that our classifier allows to automatically assign labels with appreciable levels of precision and recall for all the three categories, confirming the practical usefulness of Ticket Tagger for improving the issue management practices on GitHub.

Future work will be aimed (i) at comparing Ticket Tagger's accuracy and functionality with other existing solutions, as well as (ii) at investigating its usefulness through the analysis of direct feedback from end-users.

References

[1] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the Centre for Advanced Studies on Collaborative Research, October 27-30, 2008, Richmond Hill, Ontario, Canada*, 2008, p. 23.

[2] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 2013, pp. 188–197.

[3] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 550–554.

[4] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "DECA: development emails content analyzer," in *International Conference on Software Engineering, ICSE 2016 - Companion Volume*, 2016, pp. 641–644. [Online]. Available: https://doi.org/10.1145/2889160.2889170

[5] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in *International Symposium on Empirical Software Engineering and Measurement, ESEM 2017*, 2017, pp. 121–130. [Online]. Available: https://doi.org/10.1109/ESEM.2017.19

[6] P. Floris and H. Vogt Harald, "How to save on software maintenance costs, omnext white paper," *SOURCE 2 VALUE*, 2010.

[7] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *International Conference on Software Engineering, ICSE '13, 2013*, 2013, pp. 392–401. [Online]. Available: https://doi.org/10.1109/ICSE.2013.6606585

[8] J. L. C. Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, and J. Cabot, "Gila: Github label analyzer," in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015*, 2015, pp. 479–483. [Online]. Available: https://doi.org/10.1109/SANER.2015.7081860

[9] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[10] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[11] S. Panichella, "Summarization techniques for code, change, testing, and user feedback (invited paper)," in *VST@SANER*. IEEE, 2018, pp. 1–5.

[12] S. Panichella, G. Bavota, M. D. Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes," in *ICSME*. IEEE, 2014, pp. 251–260.

[13] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can I improve my app? Classifying user reviews for software maintenance and evolution," in *International Conference on Software Maintenance and Evolution, ICSME*, 2015, pp. 281–290. [Online]. Available: https://doi.org/10.1109/ICSM.2015.7332474

[14] Y. Zhou, Y. Tong, R. Gu, and H. C. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016. [Online]. Available: https://doi.org/10.1002/smr.1770