Prioritizing Github Issues*

Akash Dhasade
Indian Institute of Technology, Tirupati
tee15b007@iittp.ac.in

K.S.S Bhargav
Indian Institute of Technology, Tirupati
tcs15b015@iittp.ac.in

Vanka Sai Sumanth
Indian Institute Of Technology, Tiruapati
tcs15b029@iittp.ac.in

Dr. Sridhar Chimalakonda
Indian Institute of Technology, Tirupati
ch@iittp.ac.in

ABSTRACT

Issues on github are a common mode of tracking bugs in software projects . Integrators face challenges with regards to prioritizing work in the face of concurrent issues. We present the design and implementation of Issue-Prioritizer, a tool to prioritize issues based on machine learning techniques. Issue-Prioritizer works like a priority inbox for issues, recommending top issues the user should focus on based on his/her own priorities which can be set through some parameters.

CCS CONCEPTS

• **Machine learning**; • **Supervised Learning**; • **Natural Language processing**; • **Software Engineering**; • **Software Architecture**;

KEYWORDS

Github, Issues, prioritisation, ensemble methods,latent dirichlet allocation, dynamic tracking

ACM Reference Format:
Akash Dhasade, K.S.S Bhargav, Vanka Sai Sumanth, and Dr. Sridhar Chimalakonda. 2018. Prioritizing Github Issues. In *Proceedings of ACM Conference (Conference'17),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Issues are user reports which are reported when a problem is faced. Issues are used to keep track of tasks, enhancements, and bugs for software projects on github [1]. Issues on github are characterized by their titles, description,labels, assignees and comments. Issues define some of the milestones of the project. At any given point of time, software repositories of considerable size have hundreds of open issues which the integrators need to inspect and resolve. For example, the tensorflow repository has more than 1000 issues open at any point in time.The github interface for issues allows developers to sort open issues on a multitude of criteria, ranging from the age of issue to recently updated issues. However, the integrators still face challenges with regards to prioritizing work in the face of concurrent issues. In this paper, we present the design and implementation of issue prioritization tool, the Issue-Prioritizer. The Issue-Prioritizer acts as a priority inbox for issues. It examines all open issues and presents the top issues that need immediate attention to the project integrators. Issue-Prioritizer is a dynamic tool that reacts to real time changes in the state of issues.

*Produces the permission block, and copyright information

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

2 PRIORITIZING ISSUES

2.1 Model

We model our tool similar to the pull request prioritization tool called the PRioritizer [2] and using a priority inbox approach. We not only look at static information with regards to issues but also take into account (the dynamics of) previous actions on issues. We prioritize issues based on three criteria:

- Issue Lifetime predicted via machine learning techniques
- Hotness of the issue
- Category of the issue

G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen [3] analyzed work practices and challenges in pull based development from an integrators's perspective and found out that integrators prioritize contributions by examining their criticality (in case of bug fixes), their urgency (in case of new features) and their size. Issue lifetimes are indicative of the criticality of fix and their urgency. We train a machine learning algorithm on historical data to build a model for predicting the closing dates of currently open issues.

PRioritizer [2], tool to prioritize pull requests does not take into account the liveness and asynchrony of pull requests. Essentially, high recent activity on the issue is indicative of the fact that the issue is currently trending and should be looked over (if not always). We take into account the recent activity on the issue by measuring the hotness of the issue following a model similar to YOUTUBE trending videos [4]. The method for measuring hotness and features used are explained in the following section.

Category of contributions plays an important role where integrators prefer looking at some categories of Issues over others[3]. Category of the Issue is identified by the title of the issue and its description i.e the first comment in the thread of the issue. If the issue is regarding a bug that is causing crashes it is more important than a issue which demands for a new feature to be added. The feature may be essential but it is certainly not as important as the former. So we need to identify to which category the issue belongs to. This is done using a tool in Natural Language Processing called Latent Dirichlet Allocation(LDA).

2.2 Features

The features set was extracted considering all possible data associated with an issue. These features are very similar to the features used for prioritizing pull requests (E.Van & et al., 2015). An overview of selected features is summarized in Table 1.

One of the most critical parameters in prioritizing issues is the author of the issue. The author could be the project owner, a project member, project contributor or any other third person. The owner of the repository raising an issue is more important than a third person raising one. Developers also deem the age of the issue important: we measure it as the elapsed time in days since the issue was created. Number of comments and labels on the issues such as 'awaiting response' are indicative of recent activity and current state of issue respectively. The title and description are used to analyze category of the issue. Number of assignees is indicative of the urgency of fix.

We classify the features as being static and dynamic. The features whose values once specified do not change (or hardly change) with time are called as static features while the features whose values keep on changing (or frequently change) with time are called as dynamic features. Examples of each category are assignees and comments respectively. Assignees once assigned by the developer look over the issue throughout its lifetime but might change over time in rare cases. Comments are a good measure of recent activity on the issue and keep changing frequently with time.

3 DESIGN AND IMPLEMENTATION

The Issue-Prioritizer is implemented in Java and forms a loosely coupled architecture with the following measure components: (1) Watcher (2) Analyzer (3)Predictor and (4) Visualizer. The architecture diagram can be seen in Figure 1. Each of the components is detailed next.

3.1 Watcher

It is responsible for keeping track of new issues and state changes in the old issues. It maintains a time stamp of last updated time and SQL tables corresponding to issues, comments and labels for a repository. This storage of data is crucial for quick retrieval of issues in later stages of calculation and it takes place when Issue-Prioritizer is setup for the first time. Next time when watcher is called, it scrapes the only those issues which have been updated/added/closed after the last time stamp. All data retrieval is done using github API v3.

3.2 Analyzer

It is responsible for extracting the required data from the SQL database of issues and evaluating metrics required for priority calculation.

3.3 Predictor

The metrics evaluated by analyzer and other required data are then passed to the predictor which uses the trained machine learning model to predict close dates. It then evaluates the final prioritizes of all open issues passes the list of prioritized issues to the visualizer.

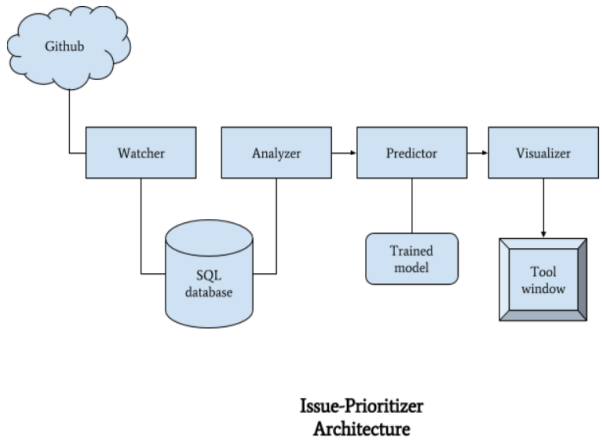

Figure 1: Issue-Prioritizer Architecture

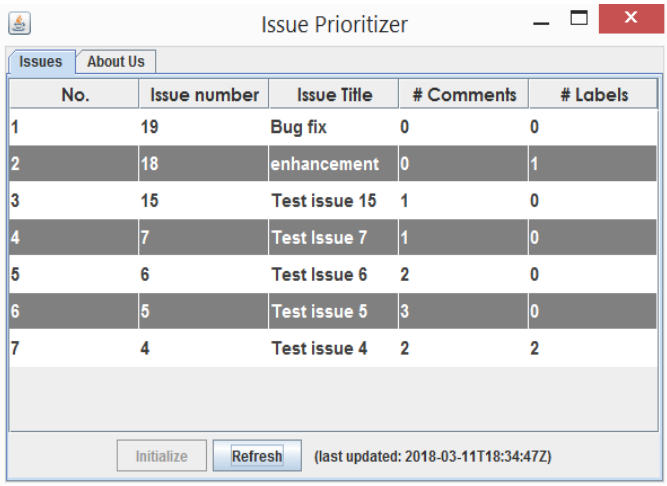

Figure 2: First view of Issue-Prioritizer

3.4 Visualizer

Visualizer presents the list of issues, enriched with information extracted from the analysis and prediction phases. A screenshot of the visualizer (and the tool) can be seen in Figure 3.

4 PRIORITIZATION CRITERIA

We prioritize issues based on three criteria: lifetime of the issue predicted via machine learning techniques, hotness of the issue and category of the issue. The process of calculation of each of them is presented next.

4.1 Lifetime of the Issue

We predict the lifetime of the issue in days using three of the above features: number of assignees, number of comments and author association of the issue. Choice of these features from all available set of features is intuitive from the fact that more the number of

Table 1: Issue features and their nature

Feature	Description	Type
Author of issue	Is the author a project member?	Static
Comments	Number of discussion comments	Dynamic
Assignees	No of assignees to look over the issue	Static
Title and description	The text associated with issue title and description	Static
Labels	The type and number of labels assigned to the issue	Static
Age	Days between creation of issue and current time	Dynamic

assignees, faster will the issue be closed. Likewise, an issue with author as a project member is more likely to get closed than the issues with authors who are non members. The choice of these features is not exclusive but is subject to trial.

The closing time of the issue was categorized into 5 classes having analyzed the plot of number of issues vs closing time of issues for the tensorflow repository on github [5]. The plot is presented in Figure 2. We observed that a lot of issues get resolved within first two days, while the maximum that any issue was not resolved was around 80 days. The five classes for closing time with associated number of days are presented in Table 2.

The accuracies of the models trained are presented in Table 3. The issue dataset comprised of 1039 closed issues corresponding to the tensorflow repository on github. The decision tree classifier and Support Vector Machine performed better than other classifiers with accuracy of 54%. A soft voting ensemble was made out of them which achieved an accuracy score of 55.769%.

4.2 Hotness of the issue

The model used for measuring hotness of the issues resembles the one used to find out trending videos on YOUTUBE[4]. Table 4 compares the parameters used in both models.

We use four features for measuring hotness: comments, assignees, author-association and label. We deteriorate the weight of each instance of a feature with time such that the contribution of that instance in determining total priority decreases as the age of feature instance increases. Static features are the features which stay intact throughout the life time of issue while dynamic features keep changing with time. Hence, we deteriorate the contribution from static features slowly as compared to that of dynamic features in hotness calculation. Log with base 2 deteriorates static features while log base 10 in used for deteriorating dynamic features. The formula for measuring hotness can be stated as:

$$\textbf{hotness} = \sum \frac{w_{comment}}{ln(2+(t-t_0))} + \sum \frac{w_{assignees}}{log_{10}(2+(t-t_0))} + \frac{authAssoc}{log_{10}(2+(t-t_0))} + \sum \frac{w_{label}}{log_{10}(2+(t-t_0)))}$$

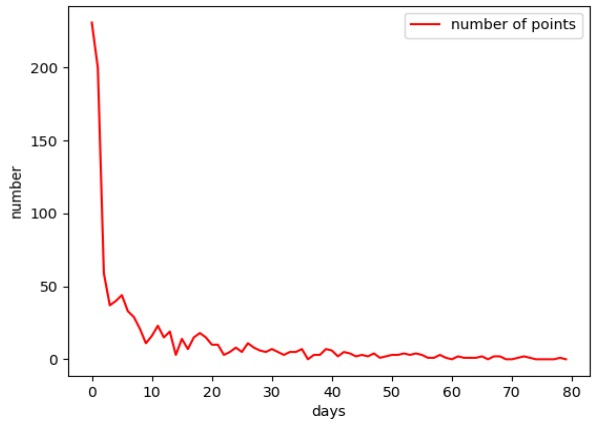

Figure 3: Lifetime vs Frequency of Issues

Table 2: Close time class

Days Class	Number
0 - 2	Class 0
3 - 7	Class 1
8 - 14	Class 2
15 - 28	Class 3
> 28	Class 4

Table 3: Machine learning models and their accuracies

Classifier	Accuracy Score (%)
Logistic Regression	52.88
Decision tree (max depth 2)	54.80
Random Forest	49 - 51
SVM	54.807
Ensemble	54.807

Table 4: Model similarity - Youtube trending videos & Issue-Prioritizer

Youtube trending vidoes	Issue-Prioritizer
Age of video	Age of issue
Growth rate of views	Growth rate of comments
Where are view s coming from?	Author of comment
View count	Comment and label count

4.3 Category of the issue

We need to identify the category of the issue, as some category of issues are more crucial to resolve immediately than others. For example an issue regarding a bug that is causing crashes would need

Table 5: Category wise priority assignments

Category of the Issue	Priority
fix	3
error	2
update	1

an immediate attention. Where as an issue demanding a feature addition is atleast not as important as the former. Hence it is necessary to identify the issue category. We can apply machine learning to do this. But what tool in machine learning is an elegant way? If we have to do supervised learning, we need a labeled dataset which is not available. We could just hand label the issues which is possible but is a clumsy and tiring way. So we decided upon using unsupervised learning to do this. We used latent dirichlet allocation technique of natural language processing to do this. Latent dirichlet allocation or LDA scans through a corpus of the documents as many times as the number of iterations parameter and outputs the topics in the corpus. LDA is a bag of words model, meaning it disregards grammar, word order but keeps the multiplicity of each word. Before running LDA, the stop words, punctuation marks are removed from the corpus. Then the words are lemmatized .Thus corpus is made into a clean document term matrix which is passed into LDA. Each document is made of topics which are to be discovered. Each topic is made of words which occur with a certain probability in that topic. The following categories have been identified by running LDA on tensorflow issues.

$$(0,' 0.044 * "fix" + 0.008 * "mkl"') \tag{1}$$

$$(1,' 0.017 * "branch" + 0.014 * "update"') \tag{2}$$

$$(2,' 0.025 * "tensorflow" + 0.017 * "error"') \tag{3}$$

The priority assigned to each category is shown in Table 5. We observe that some of the topics turned out to be tensorflow specific. This is because LDA was trained on tensorflow issues. We need to fix this by training on a larger dataset of issues.

4.4 Final priority calculation

The final priority is the weighted sum of its components. It is given by

$$\textbf{priority} = \frac{W_{closeTime}}{(closeTime + 1)} + W_{hotness} * hotness +$$

$$W_{Category} * CategoryPriority$$

We leave the choice of assigning these weights to the user because strictly speaking, priority is a subjective thing. One may feel that an issue which is hot must be given high priority while other might think the issue which closes first needs to be attended first. Hence the user can adjust the weights based on whats more important to him. Thus the tool can be personalized based on the requirement of the user.

5 FUTURE WORK

This can be considered as being only the first stage of the tool. A lot more need to be done. Firstly we need to train the machine learning model on a larger dataset. We also need to our machine learning model to be dynamic and adapt itself so that it becomes repository specific. We need to build an online model so that it gets trained and adapts as and when a new issue enters a repository. For this we need to run the model on a server. We are also planning on building a github extension for our tool so that it can reach to a large number of audience.

6 CONCLUSION

Our tool the issue prioritizer considers almost everything that need to be considered regarding thee issue. Priority is a subjective thing. But here we give the user the option to assign weights to each of the components that make the total priority. Hence it reduces the burden of manual sorting the issues by members to a large extent. We are hopeful that our tool would turn out to be of great use to developers in the near future.

REFERENCES

[1] https://guides.github.com/features/issues
[2] Automatically Prioritizing Pull Requests, Erik van de Veen, Georgios Gousios and Andy Zaidman in the Proceedings of 12th Working Conference on Mining Software Repositories, 2015
[3] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, âĂIJWork practices and challenges in pull-based development: The integrator âĂŹs perspective,âĂİ in Proceedings of the 37th International Conference on Softw are Engineering (ICSE), 2015.
[4] https://support.google.com/youtube/answ er/7239739?hl=en
[5] https://github.com/tensorflow /tensorflow