Recommendation System for Open Source Projects for Minimizing Abandonment

Sarah Sayce[†] **and Krishnendu Ghosh** [†]
[†]Department of Computer Science, College of Charleston,Charleston, SC.
ssayce0511@gmail.com,ghoshk@cofc.edu

Abstract

The rise in the creation and maintenance of Open Source Software have facilitated the software developers to contribute and prevent abandonment. Software developers often face a daunting task to select the open source projects that remain active. In the absence of any resource to guide in the selection of the Open Source Software projects, recommendation systems is way to provide guidance to open source contributors. In this work, we describe several approaches in creation of recommendation systems for Open Source Software projects. Experiments on a synthetic data set are performed to evaluate the performance of the recommendation system.

Introduction

Open Source Software (OSS) is becoming increasingly valuable in recent years. GitHub is one of the more popular OSS hosting sites. GitHub's ability to track developer's activities such as commits and pull-requests is valuable to managers but these metrics are not the sole indicators of project success. Surveyed developers have cited the quantity of contributions and contributor growth as measures of success significantly more than bug fixes or the quality of the software produced (McDonald and Goggins 2013).

Mann from the *Technology Review* critiques modern software development, dubbing it "chaotic" due to a lack of initial organization (Mann 2002). This lack of organization could take many forms. Unassigned issues, lack of communication between contributors, delays on pull requests approvals are all examples of ineffective OSS handling. The pitfalls of disorganization associated with OSS development are found in projects of all sizes, but can have fatal consequences for a project maintained by a handful of developers.

An at-risk project could be one not backed by an organization, a project with few contributors or no new commits. All of these factors can indicate future project failure due to inefficiencies in the development process or lack of knowledge and resources and measured by *truck factor* (Avelino et al. 2019). Truck factor (TF) is the minimal number of developers of a project that quit before the project is at-risk (Avelino et al. 2019). A truck factor developer is defined as someone who is a main contributor to a code stack, if they are hit by a truck and stop contributing to the project then there is a high risk of project abandonment. A project is abandoned when all truck factor developers have ceased contributing to the project.The study mentioned previously found key statistics relating to project abandonment: 65% of sampled projects had a TF of 2 or less. 2/3 of failed projects only had one core contributor at the time of abandonment (Avelino et al. 2019). A goal of this work is to facilitate newcomers into lasting maintainers of a project (particularly those at risk) so code-responsibility falls on more active and passionate developers. As an aid to the software developers, recommendation system provides options for selection of software projects.

There are thousands of contributors on GitHub, and turnover in OS development is somewhat unavoidable because developers are constantly wanting new objectives (Izquierdo-Cortazar et al. 2009). When one developer leaves a project and another takes over their *orphaned code*, there will always be some sort of knowledge loss. This loss is compounded if there is a mass exodus of contributors. We should be striving for efficiency when contributing to OSS. As a preventive measure of mass exodus of contributors from OSS, we create a recommendation system that aligns a given OSS with the interests of the contributors.

Our goal is to tackle the lack of personalization within recommendation systems for software engineering (RSSE). We propose a two prong approach, one to address projects at risk of failure by abandonment, and another to cater to the needs of developers with the aim of reducing project abandonment.. We create a hybrid based approach for our implementation of the recommendation system with the aim providing the user options for their preferences for selection of OSS projects.

Related Work

There is a body of work published in recommendation systems in software engineering (Robillard, Walker, and Zimmermann 2009). We focus on the recommendation systems on open source systems for the contribution of open source developers and related topics. ConRec (Zhang et al. 2017a) is a recommender system for contributors given a specific project. It leveraged user's historical activity to gauge their interests and created a commit network of GitHub users. Using this network a hybrid recommendation system, a combination of a text matching and collaborative-filtering algorithm, is created. A similar system, DevRec (Zhang et al.

2017b) was designed as a hybrid recommendation system which combines a development activity based system with a knowledge-activity based system to recommend proper developers to open source projects.

A study interviewing project mentors found that most new contributors face common challenges, one of the most prominent being a lack of self confidence and understanding their background in regards to the project (Balali et al. 2020). Many developers have skills that can be utilized but there are social barriers preventing them from taking advantage. A similar study analyzed the common roles present in a OSS ecosystem (Trinkenreich et al. 2020). One subsystem were labeled as *community centred roles*. Particularly an OSS Advocate is an individual that focuses on increasing contribution to the project by bringing new people in and making the community and inclusive and safe experience. The same study found that entry points and career pathways are fluid and non-linear. Users could be coders or non-coders within project or community centered roles. Researchers have explored methods to recommend relevant GitHub projects based on user behavior data (Zhang et al. 2014).

Methodology

Data for Recommendation System: The dataset for the evaluation of recommendation system is publicly available (Avelino et al. 2019). The dataset based on GitHub was used to study truck factor. This GitHub data set was developed by focusing on 6 programming languages most popular on GitHub and then selecting the top 500 repositories from each language (excluding forks). Using popular projects with common languages ensures the quality of the data and ensures the projects are relevant to the OSS community. The data set was slimmed down to 1,932 projects after researchers excluded projects that had not exclusively been hosted by GitHub, or had less than 2 years of development history (Avelino et al. 2019). Figure 1 displays the distribution of our quantitative data including the number of project commits, forks, developers, files, open issues, and a truck factor calculation for each project. Take note of the logarithmic scale. The quantity of forks, files, and developers per project each have relatively uniform distribution. These attributes are used to identify qualitative aspects of the projects for the knowledge based system. Figure 2 shows the frequency of the 20 most common words in the project descriptions. The descriptions of each project is utilized for text analysis in our content-based system.

System Architecture : Each type of recommendation system has their advantages and disadvantages. The aim of a hybrid recommendation system is to create a synergy between these two methods that make up for each other's shortcomings (Burke 2007). Due to the limitations of our data set, which only contains product data, a collaborative filtering technique would not be effective since it relies heavily on user ratings. While a content based recommendation system relies on product data (which we have an abundance of) it still suffers from the cold start problem. We have an abundance of products with no existing ratings and

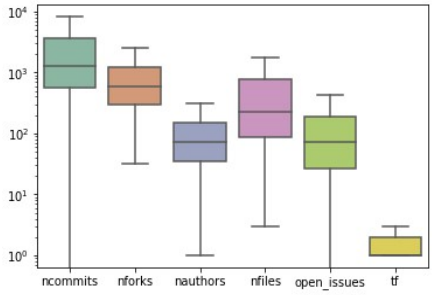

Figure 1: Box plots of numerical project data.

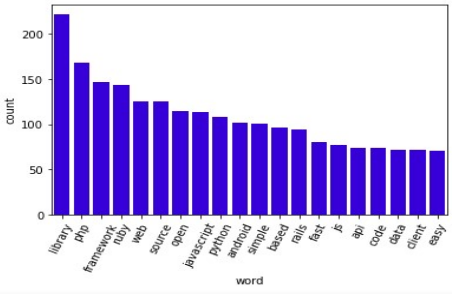

Figure 2: Word frequency in project descriptions.

new users. A knowledge-based recommendation system can overcome this problem. While it requires domain knowledge and in our case heavy influence from a knowledge engineer, a knowledge-based system focuses on the users immediate need. The utilization of a knowledge-based system also aids in the stability vs. plasticity problem (Burke 2007). Since this is a recommendation system for software engineering, users' preferences are expected to be static compared to the preferences for products. Our data set contains sufficient product data and there is an assumed acceptable level of domain knowledge surrounding OSS projects. Since our data set contains no user data, this knowledge source compiled of synthetic data that we produce to train and test our model.

In our implementation of recommendation systems, a *feature augmentation* strategy to created for a hybrid recommendation system. Feature augmentation hybrids allow the contributing recommendation system to augment the data with it's own recommendation logic before reaching the main recommendation system (Burke 2007). This strategy is employed when there is a desire to add additional knowledge sources to an existing recommendation system. Our contributing recommendation system will be knowledge based and augment the incoming data based on the user's preferences. This augmented data will be used in our main content based recommendation system.

Knowledge Based Recommendation System (KBRS):

The knowledge based recommendation system will generate a list of projects that match the developers needs. It is worth noting that aside from aiding developers, a second goal of this project is to prevent the failure of at risk projects. For this reason the recommendations from the knowledge

based system are sorted initially by the number of forks a project has. The less forks a project has implies there are less remote copies of this project and is less popular or used.

The third and final query to our user requires a Boolean response, and asks if they would be interested in contributing to an at risk project. If they respond yes, the recommendations will be sorted based on truck factor then by forks. A lower truck factor (TF) correlates to project instability and risk of complete abandonment. The majority of projects in our data set have a TF of 3 or lower, with only a handful of outliers so this sorting will most likely throw out any larger more established projects from our recommendation set. We have augmented the data with our query to the user and will take the top match as input for our content based recommendation system.

Content-Based Recommendation System (CBRS)

The content-based recommendation system in our hybrid is the main recommendation system. The content-based recommendatiom system will analyze the descriptions of our software projects using a *term frequency-inverse document frequency* (TF-IDF) matrix. A TF-IDF calculation tells us the importance or relevance of a word in a particular document, in our case the project descriptions. Using the project names as indices, this system will take an existing project name from the data set and return the top 10 most similar projects. Since our knowledge-based recommendation system is our accompanying system, the input for our content recommendation system will be the top 10 matches on the users query. Our content recommendation system will provide the most textually similar project to each item. This final set of 20 will be ranked ascending by number of forks and presented to the user. We choose number of forks here since there is not much variability within truck factor left in our augmented data set. While our content recommendation system is our main system, the knowledge based recommendation system aided in our text analysis of projects by providing domain knowledge otherwise unavailable to the system alone. The content based recommendation system provides items that were otherwise out of the users query.

Evaluation & Results

The dataset consisted of only project data. However, to evaluate our system we had to augment the dataset using synthetic user data. The synthetic data we created contains 50 users each assigned 3 features corresponding to the three queries (preferred coding language, experience, and willingness to help others). Since our user data is synthetic, we shall have to add a column for the actual outcome of our recommendation results. A satisfaction column has been added to our users data frame. A users satisfaction is on a scale of 0-10 . The generation of the satisfaction ratings for our synthetic users we add a variable to the user profiles that represents the users taste. A user's taste is not random, but influenced by other values in the user's profile. More experienced users will be labeled as "optimists" and have a distribution of ratings skewed higher, while less experienced users will be labeled as "demanding" thus their ratings will be skewed to lower values. The reasoning behind assigning these tastes to particular users is to generate more realistic data that is somewhat aware of the context in which these ratings are given. We can use probability distribution functions to help create context-aware synthetic data (Pasinato et al. 2013). It is assumed that the less experienced the user, the more guidance and accurate recommendations needed. Experienced OSS developers can be assumed to have low expectations or be more willing to work on anything rather than satisfy a niche. A users assigned taste will be a scalar value 1-3 and correlate to the users experience. 1 being a pessimist with a skewed distribution to lower values and 3 being an optimist with a skew to higher values. Those with an indicated level 2 experience will have a normal distribution. Each user will be assigned a PDF or probability distribution function that governs it's rating behavior. To manipulate the distribution of ratings we will only need to manually change the mean and standard deviation of our random variable. While all user "types" will have ratings of a normal (Gaussian) distribution and a standard deviation of 2. Average users will have a randomly generated distribution of ratings with a mean of 5. Pessimists will have an average of 3 and optimists an average of 7. Now each user will assign 10 scores to their respective 10 results. These scores are randomly generated following their assigned PDF. These 10 scores will be averaged for a users final satisfaction score. This will be the metric we will use to evaluate our model.

Train/Test Split	MAE	MSE	RMSE
50/50	0.592	0.452	0.672
70/30	0.467	0.358	0.598
80/20	0.461	0.388	0.623
90/10	0.592	0.604	0.777

Table 1: Performance Measure of Recommendation System with Mean absolute error(MAE), Mean square error (MSE) and Root Mean Squared Error (RMSE)

Statistical accuracy metrics evaluate accuracy of a filtering technique by comparing the predicted ratings directly with the actual user rating. In our case, the users "actual" rating was the context-aware satisfaction score we just generated for each user. We are implementing a train / test split to create and evaluate our model. Multiple models were created with the corresponding training / testing intervals using polynomial regression, predicting satisfaction outcome with experience level. Root Mean Squared Error (RMSE) value for our initial model was 0.834, so we can infer around 80% of the variance here is explained by our model. This is due to the context-aware attributes we established earlier that governs the "users" behavior. Table I shows the mean absolute error, as well as the mean squared and root mean squared error for each of our three models (Isinkaye, Folajimi, and Ojokoh 2015). These metrics reveal the accuracy of our model, the lower the value the more accurate our predictions. Decision Accuracy Metrics are another way to measure the effectiveness of the system. These metrics help users in selecting items that are of very high quality out of the available set of items. The metrics view prediction procedure as a binary operation (Isinkaye, Folajimi, and Ojokoh

Train/Test Split	Recall	Precision	F1
50/50	0.350	0.149	0.204
70/30	0.500	0.322	0.388
80/20	0.333	0.155	0.208
90/10	0.333	0.222	0.250

Table 2: Performance Measure of Recommendation System with Mean absolute error(MAE), Mean square error (MSE) and Root Mean Squared Error (RMSE)

2015). The relative contribution of precision and recall to the F1 score are equal. Table 2 displays these metrics along with the respective models.

Interestingly the recall, precision, and F1 score indicate the 70/30 split model performs better in those aspects than the 80/20 model. This is also reflected in the model's respective receiver operating characteristic (ROC) curves. An ROC curve shows the relationship between recall over time. More acutely, it is the plot between the false positive rate and the true positive rate.

Conclusion

The hybrid recommendation system produced was a feature augmentation hybrid of a knowledge based and content based recommendation system. In terms of statistical accuracy the model did well, but it had limited effectiveness regarding precision and recall. These shortcomings are most likely due to the small number of ratings from our synthetic user data. The content based half of the hybrid system would have more effectiveness with a larger corpus of documents. This recommendation system could be improved upon in the future with a larger data set of GitHub user and project data. Our system was effective in promoting projects that were "at risk" and deemed to have a low truck factor regardless of user input. The design of this recommendation system could be built on in the future to increase the personalizing among software recommendation systems. Further research into developer motivations and behaviors would be useful for similar knowledge based systems. Rigorous evaluation of the recommender system using open source software data will be conducted. This user-centered recommendation system can serve as a preliminary model for future developer-focused software recommendation engines.

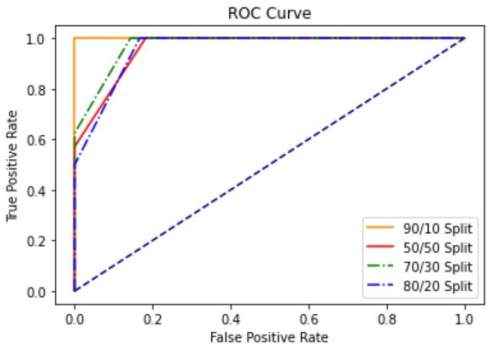

Figure 3: ROC Curves of Different Training-Test Data

References

Avelino, G.; Constantinou, E.; Valente, M. T.; and Serebrenik, A. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–12. IEEE.

Balali, S.; Annamalai, U.; Padala, H. S.; Trinkenreich, B.; Gerosa, M. A.; Steinmacher, I.; and Sarma, A. 2020. Recommending tasks to newcomers in oss projects: How do mentors handle it? In *Proceedings of the 16th International Symposium on Open Collaboration*, 1–14.

Burke, R. 2007. Hybrid web recommender systems. *The adaptive web* 377–408.

Isinkaye, F. O.; Folajimi, Y.; and Ojokoh, B. A. 2015. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal* 16(3):261–273.

Izquierdo-Cortazar, D.; Robles, G.; Ortega, F.; and Gonzalez-Barahona, J. M. 2009. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *2009 42nd Hawaii International Conference on System Sciences*, 1–10. IEEE.

Mann, C. C. 2002. Why software is so bad. *Technology Review* 105(6):33–38.

McDonald, N., and Goggins, S. 2013. Performance and participation in open source software on github. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. 139–144.

Pasinato, M.; Mello, C. E.; Aufaure, M.-A.; and Zimbrao, G. 2013. Generating synthetic data for context-aware recommender systems. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, 563–567. IEEE.

Robillard, M.; Walker, R.; and Zimmermann, T. 2009. Recommendation systems for software engineering. *IEEE software* 27(4):80–86.

Trinkenreich, B.; Guizani, M.; Wiese, I.; Sarma, A.; and Steinmacher, I. 2020. Hidden figures: Roles and pathways of successful oss contributors. *Proceedings of the ACM on Human-Computer Interaction* 4(CSCW2):1–22.

Zhang, L.; Zou, Y.; Xie, B.; and Zhu, Z. 2014. Recommending relevant projects via user behaviour: An exploratory study on github. In *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies*, CrowdSoft 2014, 25–30. New York, NY, USA: Association for Computing Machinery.

Zhang, X.; Wang, T.; Yin, G.; Yang, C.; and Wang, H. 2017a. Who will be interested in? a contributor recommendation approach for open source projects. In *SEKE*, 363–369.

Zhang, X.; Wang, T.; Yin, G.; Yang, C.; Yu, Y.; and Wang, H. 2017b. Devrec: a developer recommendation system for open source repositories. In *International Conference on Software Reuse*, 3–11. Springer.