Cross-Project Issue Classification based on Ensemble Modeling in a Social Coding World

Yarong Zeng, Yue Yu, Qiang Fan, Xunhui Zhang,
Tao Wang, Gang Yin, and Huaimin Wang

National Laboratory for Parallel and Distributed Processing
National University of Defence Technology
Changsha, China
{zengyarong16,yuyue,fanqiang09,zhangxunhui,
taowang2005,yingang,hmwang}@nudt.edu.cn

Abstract. The simplified and deformalized contribution mechanisms in social coding are attracting more and more contributors involved in the collaborative software development. To reduce the burden on the side of project core team, various kinds of automated and intelligent approaches have been proposed based on machine learning and data mining technologies, which would be restricted by the lack of training data. In this paper, we conduct an extensive empirical study of transferring and aggregating reusable models across projects in the context of issue classification, based on a large-scale dataset including 799 open source projects and more than 795,000 issues. We propose a novel *cross-project* approach which integrate multiple models learned from various source projects to classify target project. We evaluate our approach through conducting comparative experiments with the *within-project* classification and a typical *cross-project* method called *Bellwether*. The results show that our cross-project approach based on ensemble modeling can obtain great performance, which comparable to the *within-project* classification and performs better than *Bellwether*.

Keywords: Cross-project, Issue Classification, Transfer Learning, Ensemble Modeling;

1 INTRODUCTION

With the popularity of open source, more and more individual developers, research institutes or industrial companies prefer to host their software projects on social coding platforms, *e.g.*, BitBucket [1] and GitHub [2], to organize and manage the whole development life-cycle, which can catch more attentions from a large number of external developers and users. In order to simplify the collaborative process for both experienced and inexperienced contributors, a set of

[1] https://bitbucket.org
[2] https://github.com

lightweight tools are widely adopted, *e.g.*, pull request for code-patch submission [6] and Issue Tracking System (ITS) for development task management [4]. For example, by using the ITS on GitHub [3], a contributor only requires a short textual abstract containing a title and an optional description, when s/he wants to report a new found bug, ask an usage question or request an original feature to a software project [5]. On the one hand, those deformalized contribution mechanisms can make a collaborative project easy to collect more contributions from a wider range of the community. On the other hand, the increasing number of arbitrary, half-baked or undesirable contributions flow into the project along with high-quality ones, which poses a serious challenge for the core team in further maintenance, *e.g.*, project integrators are overburdened with evaluating excessive pull requests in time [28, 27].

To reduce the burden on the side of project core team, various kinds of automated and intelligent approaches have been proposed based on machine learning and data mining, such as prioritisation tools for recommending the high-priority bug reports [25] and pull requests [26]. In theory, the performances of the training-based approaches are highly correlated with the scale of training data [2, 17]. However, within a specific project, a comprehensive historical dataset which can cover the whole development life-cycle is often not available in practice. In this case, building a good training model is a challenging issue as collecting new data and labeling them is cost-expensive. Inspired by the theory and technique of transfer learning [20], *cross-project* approaches have been introduced and brought into focus to solve this problem. However, the existing cross-project work in software engineering, as discussed in Section 2.2, faces the challenges in terms of complicated training process and high computational cost. In this paper, we conduct an extensive empirical study of transferring and integrating reusable models across projects in the context of issue classification. We propose a novel approach to classify new issues in target projects by integrating multiple models learned from various source projects. And we evaluate our approach on a collection of 799 projects and more than 795,000 issue reports in GitHub. To the best of our knowledge, this is a first study of exploring cross-project approaches that categorizes issue reports, and also evaluating the cross-project learning topic in software engineering on such a large-scale dataset.

The rest of the paper is organized as follows. In Section 2, we discuss the backgroud and related work of issue classification and transfer learning applied in cross-project defect predction. In Section 3, we present the approach employed to address cross-project issue classification. In Section 4, we show the dataset used to evaluate our approach. Experiments and results are described in Section 5 and Section 6. In the end, we draw our conclusions in Section 7.

[3] https://github.com/blog/831-issues-2-0-the-next-generation

2 BACKGROUND AND RELATED WORK

2.1 Issue Reports Classification

In project development process, both developers and end-users can submit issue reports to ITS when performance does not meet their expectations [5]. According to our statistics to investigate the issue's first response time from core team, for nearly 800 open source projects (see Section 4.1) in GitHub. We found that it takes a long time period for core team members to start addressing issues (the average of median response time is 5.2 days, while the average of mean response time is 44 days). Therefore, to reduce the issue analysis time and improve the overall management process, an automated mechanism to classify issue reports is urgently needed.

Categorization of issue reports using techniques from text mining and machine learning has been receiving increasing attention from the research community. Antoniol et al. [1] used three machine learning methods to distinguish bugs from other kinds of issues. They found the information contained in issues posted in ITS can be indeed used to classify such issues, distinguish bugs from other activities, with a correct decision rate as high as 82% (experiment on 3 case projects). There are also some ways to increase the information extracted from ITS to improve the performance of the model. Merten et al. [16] found meta-data from ITS can improve classifier performance through conducting empirical research on 4 open-source projects. Fan et al. [5] concluded that semantic perplexity of issue reports is a crucial factor that affects the classification performance. They experimented on 80 projects in GitHub and verified that model which considered semantic complexity do improve the issue classification performance.

Above classify methods rely on the data set within the project to train the classification model. However, Peters et al. [21] found within-projects predictors are weak for small data-sets. And Kitchenham et al. [9] discovered the problems with relying on within-project data-sets. They point out that the time required to collect enough project data can be prohibitive. In these situations, we propose a cross-project approach to address issues categorization problem of small dataset projects. In addition, those related works are based on a set of standard projects, there is no way to prove whether they are suitable for large-scale projects in actual. In this paper, we conduct a large-scale experiment to verify the effectiveness of our cross-project approach on issue classification.

2.2 Cross-project prediction

In practice, for new projects or projects with limited training data, It's a time-consuming and effort-intensive task to collect new data and label them. A possible way to solve this problem is using data collected from other projects, called cross-project prediction. Cross-project prediction has been receiving significant attention, and many studies have been proposed in research community.

Initial attempts for cross-project prediction always resulted in pessimistic performance. Zimmermann et al. [33] ran 622 cross-project predictions and found that only 3.4% actually worked. Menzies et al. [15] concluded that the local model were superior to the global model with experiment on four big datasets. Other researchers offer similar disappointing results [22, 3, 23]. The main reason that leads to the poor performance of cross-project prediction is the data distribution differences between the source and target project [18]. To reduce this difference, many optimized cross-project methods have been proposed. Zhang et al. [30] proposed context-aware rank transformations for predictors, built a universal model on the transformed data of 1398 projects, and found this model obtains prediction performance comparable to the within-project models. Then they proposed connectivity-based unsupervised classifier (via spectral clustering) and evaluated the feasibility for cross-project prediction by experimenting on three publicly available datasets [31].

Transfer learning techniques are often adopted for cross-project prediction to transfer lessons learned from source project S to the target project T in software engineering community. The specific application methods under cross-project scenarios can be divided into two types: data-level and feature-level transform. The data-level approaches directly transfer some subsets of data from source projects to the target project. Turhan et al. [24] collected similar source instances for target instances to train a prediction model using the nearest neighbour filter (NN filter) method. Ma et al. proposed Transfer Naive Bayes (TNB) [14] to predict target project. The data-level approaches use data in raw form, but they suffer from instability issues [12]. This prompted research on feature-level approaches. These approaches use projection to place the source and target data in a common latent feature space. Nam et al. [18] adapted the state-of-the-art transfer learning technique called Transfer Component Analysis (TCA) [19] and proposed TCA+ by adding decision rules to select proper normalization options.

To reduce the complexity of cross-project prediction, Krishna et al. [12] proposed a simple but effective cross-project approach. They chose the prior project which offered most accuracy predictions that generalize across other projects as "bellwether", then used the "bellwether" to generate quality predictors on new project data. This research is very instructive for our work. It proved that directly transferring model learned from the source project to predict the target project is feasible. However we think that the cross-project approach will be more effective [13] when taking the integration of models into consideration. In this work, we propose a new cross-project approach by taking into account the cost of computation and the reusability of cross project resources.

3 APPROACH

Cross-project prediction may suffer from two limitations. First, conclusion instability proposed by Krishna [12], that is when learning from all available data, prediction model may undergo constant changes whenever new data arrives. Second, the learner is opaque and the cost of calculation is expensive. When

encountering new test data, the model needs to be retrained. To ease these two restrictions, we propose a new cross project prediction approach based on **Ensemble Modeling**, which consider the cross-project problem from the perspective of integration with reference to *Ensenmble Learning*. The key idea of our approach is using multiple models of source projects to make a classification decision for the target project. The source project is a project which have sufficient training samples to train a local model. And the target project is a project with limited training data. Comparing with the existing cross-project approaches, doing transfer at model level in our approach can greatly reduce the computational cost, because it can directly reuse the existing trained models. In addition, compared to *Bellwether* mentioned in Section 2, which transfer only one model to predict target predict, our approach can achieve better generalization performance than it by combining multiple models. We apply following two operators to classify the new target project P_T. The source project set is denoted as C.

1. *CLASSIFY:*
   - Randomly choose K models of source projects.
   - Integrate these K models to predict the target project P_T using the majority voting rule.
2. *MONITOR:*
   - For source projects, update classification model on time to add the newly arrived training data.
   - For target projects, if enough training data have been accumulated over time, the local model will be used for classification.

For the selection of K value, we test its impact on cross-project classification in Section 5, we will elaborate how to choose K value in Section 6.

Pay attention to the simplicity of this approach, each model of project in C can be trained in advance. For a new target project, we only need to randomly select models and integrate voting. We can directly reuse the knowledge of other projects to reduce the cost of calculation. And the *MONITOR* operator can relieve conclusion instability.

4 DATASET

4.1 Data Collection

We compose a sample of GitHub projects that each project contains a sufficient number of labeled issues. Using the 05/2017 GHTorrent [4] dump, we get projects that have at least 100 issues in total, including 12,797 projects and 6,414,872 issues. Then, we remove the non-existent projects online and collect issues data for each project through GitHub public API in consideration of data missed in Ghtorrent. The ITS in GitHub uses a labeling system to help organize and prioritize issues. To get a pre-labeled training set, we need to extract category

[4] http://ghtorrent.org/downloads.html

information from the user-defined label system. In this work, we use category extraction method[5] based on the big issue datasets to extract bug-prone labels, such as "[Type] Bug", "Defect", and non-bug-prone label, such as "type: enhancement", "feature request". Through this process, we extract 4,222 labels and about 2,436,308 issues have these labels. Via the statistics of these labels, we found that some labels appeared in only one project. Considering the universality of the labels, we choose the first 40 labels which cover most of the issues(90.88%) and manually determine the issue type "bug" or "non-bug" according to the tendencies of the issue labels. Finally, we get 10,564 projects and 2,214,117 labeled issues.

4.2 Filtering

Repositorys hosted on GitHub have a variety of uses[8], some projects that use GitHub for issue tracking but not for code hosting or reversely. In order to avoid analyzing non-software projects and ensure the effect of classification model, we filter the dataset using the following rules:

- Project must not be forks of existing GitHub projects.
- Project must not be a pure documentation project. This criterion guarantees project have enough contextual information exploring cross project relevance.
- The issue text of project must be in English. This criterion avoids language deviation when classified the issue of other projects.
- Project must have at least 500 labeled issues. This criterion guarantees that the classification models have enough training and testing data.

After the above filterings, the obtained dataset contains 779 projects, and 795,284 issues. The smallest project has 502 issues and the largest project has 13,793 issues in total. The mean number of issues per project is 1020 and the median is 672.

4.3 Preprocessing

Each issue, which have be labeled as "bug" or "non-bug", has two main textual information sources – title and description. We concat the two parts of information together as issue text data. In order to train an automatic classifier, we create a feature vector for each text information through the following preprocessing steps. The first step is data preprocessing, a series of operations (lowercasing, tokenizing, stop-word removing, and stemming) is used to remove noise data and standardize text vocabulary. The issue text is segmented into different terms through this step. The second step is text quantizing. Using *TF-IDF* to calculate term weight, and then each issue can be represented as a weighted vector. For each project, a collection of raw issues text data can be converted to a matrix of *TF-IDF* vectors, which can be used as training data to get a classification model.

5 EXPERIMENT

5.1 Model Training

In this paper, we address the issue classification problem using binary classification. For each issue of the target project, we aim to determine the issue type as "bug" or "non-bug". We train a classification model based on the past data that takes the temporal information of issues into account. The temporal information reflects how such classification models can be used in practice. We split the issue data into two clusters at October $1th$, 2016 (According to statistics, this division can get more training and testing projects.). All issues before that time point are for training and issues after that time point are for testing. This divides 673,490 issues into training set and 121,794 issues into testing set.

We use the state-of-the-art two-stage classifier [5] to deal with issue classification problem. The first-stage uses SVM to predict the probability of bug-prone and extracts perplex information of sentences from free issue text. The second-stage combines the output of the first-stage and the developer information as training input, then uses logistic regression to predict issue type. In order to make supervised text-based classification performs the best and avoid the influence of unbalanced dataset, we train the classification model using projects with more than 500 labeled issues and the bug rate of which are between 20% and 80%. When validating the performance of the model, we use the projects with more than 50 issues as test projects so as to ensure the authenticity of the classification results.

5.2 Experimental Methodology

To verify our approach, we compare our cross-project approach against *Within-project* and *Bellwethers* approach.

Within-project:

- Using local labeled data of the project to train a classification model.
- Using the model to classify new issue data of project.

Bellwether:

- Traversing cross-project pairs that are independent of the target project and choose the most prior project which offers the highest accuracy value of prediction as "bellwether".
- Using the "bellwether", generate classifier on new issue data of target project.

We use the f_{avg} (See Equation 1) to evaluate classification performance of the above methods. Statistical analysis is used to verify our conclusions about the difference between the three methods. For the comparison between multiple groups, we use multiple contrast test procedure $\tilde{T}$ [10] in this study. We implement the procedure $\tilde{T}$ by nparcomp[5] package in R to evaluate the classification performance of all the approaches by using our dataset. We set the contrast

[5] https://cran.r-project.org/web/packages/nparcomp/index.html

type to *Tukey* (all-pairs) to compare all groups pairwise. For each pair of groups, the 95% confidence interval is analyzed to test whether the corresponding null hypothesis can be rejected. The null hypothesis of $\tilde{T}$ test is that there is no difference between classification result of the two approaches. If $p.Value < 0.05$, it means that the null hypothesis is significantly rejected at 5% level of significance. The *Estimator* denotes the relative effect between two sets of data. If the lower boundary of the interval is greater than 0.5 for groups A and B, it's means that B tends to be larger than A. Similarly, if the upper boundary of the interval is less than 0.5 for groups A and B, it's means that A tends to be larger than B. Finally, if the lower boundary of the interval is less than 0.5 and the upper boundary is greater than 0.5, it's means that the data does not provide enough evidence to reject the null hypothesis [11].

5.3 Experimental Design

According to the model training methods mentioned in the 5.1, we get a projects set S ($|S| = 465$) which have sufficient issues data (at least 500 issues which before October $1th$, 2016) to train a good classification model within the project, and a projects set T ($|T| = 500$) which have a certain amount of data (at least 50 issues which after October $1th$, 2016) to verify the performance of the cross classifier. We conduct experiments with the setting below.

We extract 50 projects from the project set S as test projects $Test_{prj}$, which have enough issue data to train a local model, and a certain amount of test issue data to verify the performance (make sure can compare with the *Within-project* method). For the rest of S and the project set T, we construct a set of cross-project pairs as training samples which used to choose prior project "bellwether" in *Bellwethers* approach. The cross-project pair is defined as, for project $A \in S$ and project $B \in T$, we consider a cross-project pair (A, B) to train a model from A to predict B if and only if $A \neq B$. Next, Referring to the steps mentioned in Section 3, 5.2, we use the three approaches (our method, *Bellwether*, *Within-project*) to classify issue data for each project in $Test_{prj}$. We repeated the above process 10 times to enhance the robustness of the experiment and make the conclusions more convincing. Moreover, In order to verify the effect of K value on classification performance, we test the classification performance of our approach with different K values in experiment, within 100 intervals, the step is 1.

5.4 Evaluation Metrics

Using *Precision*, *Recall*, *F-measure* metrics as evaluation criteria is a common procedure in related work [14, 18, 33, 7, 29]. *Precision* is used to measure the exactness of the prediction set, while *Recall* evaluates the completeness. *F-measure* represents the harmonic mean of *Precision* and *Recall*. In [32, 5], in order to give an overall performance evaluation, the average *F-measure* is used to evaluate the classification model. This metric uses the weighted average value of *F-measure* for both categories by the proportions of instances in that category [32].

In this paper, considering the performance of both categories (bug or non-bug), we use average *F-measure* to assess the accuracy of the classification. Equation (1) describes the formula to derive the average *F-measure*. The average *F-measure* is denoted as f_{avg} , *F-measure* of bug (nonbug) as f_{bug} (f_{nonbug}), and number of bug (nonbug) as n_{bug} (n_{nonbug}).

$$f_{avg} = \frac{n_{bug} * f_{bug} + n_{nonbug} * f_{nonbug}}{n_{bug} + n_{nonbug}} \tag{1}$$

6 RESULTS

Figure 1 provides a summary of the comparison steps. It shows the median values of f_avg with three approach based on 500 projects. The horizontal axis is the K value, which is mainly used to show the classification performance of our approach with different K values in the experiment. The *Within-project* and the *Bellwether* method are not affected by the K value. From the figure, we can find our approach obtains classification performance comparable to the *Within-project* method when K is large enough. And our approach out-performs *Bellwether* when integrating a few models. The preliminary result provide a evidence that the idea of model integration would achieve ideal performance in cross-project classification. In addition, it can be seen from the figure that there are some fluctuations between adjacent K values when K is greater than 25, which may be related to the randomness of the model selection.

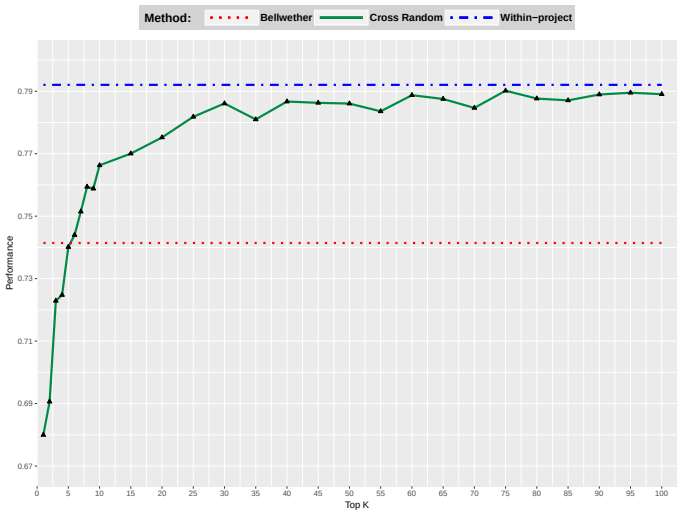

Fig. 1. The median values of f_avg with three approach based on 500 projects

The experimental results in Figure 1 give us a intuitive view of how well our cross-project approach performs. In addition, we use multiple contrast test to verify whether the conclusion is correct, through comparing the overall f_avg of 500 projects (10 tests, 50 projects at a time) for different approaches(with different K value). Table 1 shows the result of contrast test. And we reveal some of the significant turning points with different K value in performance comparison. The *p.Values* are over 0.05 in row $2, 5$ (gray background in the table), and there are flips between lower boundaries ($Lower < 0.5$) and upper boundaries ($Upper > 0.5$) in these rows. It means there is no statistical significant difference between each group pair. A significant difference at 5% level occurs in row $1, 3, 4, 6$. The lower boundaries and upper boundaries are both greater than 0.5 or less than 0.5, which means that there is a significant difference between the two sets of comparison data, one of which is better than the other. And we bold the name of the better method in the figure.

Therefore, we can draw the following conclusions. First, our approach out-perform the *Bellwether* method when integrating greater or equal to 9 models of source projects. Second, our cross-project approach can achieve the performance of the *Within-project* method when integrating greater or equal to 25 models.

Table 1. Results of multiple contrast test procedure

Comparison		Estimator	Lower	Upper	Statistic	p.Value
Ensemble_4	vs. **Bellwether**	0.552	0.509	0.594	2.823	0.133e-01
Bellwether	vs. Ensemble_5	0.503	0.460	0.545	0.151	9.893e-01
Bellwether	vs. **Ensemble_9**	0.559	0.509	0.608	3.234	1.197e-02
Ensemble_24	vs. **Within**	0.563	0.501	0.624	3.462	4.488e-02
Ensemble_25	vs. Within	0.556	0.493	0.618	3.070	1.490e-01
Bellwether	vs. **Within**	0.664	0.603	0.720	8.859	0.000e+00

7 CONCLUSION

When historical data is not available, engineers often use data from other projects. In this paper, we propose a new cross-project approach to address issue categorization problem in the case of inadequate historical data. We evaluate our approach through an empirical study on open source projects in GitHub, which compares with the *Bellwether* and the *Within-project* method. The comparison result shows that our approach out-performs the *Bellwether* and can achieve the accuracy of the *Within-project* method. This means that the idea of integration is encouraging in cross-project classification. In future work, we plan to investigate the effectiveness of our cross-project approach in different software engineering scenarios, such as defect prediction or effort estimation.

References

1. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.G.: Is it a bug or an enhancement?: a text-based approach to classify change requests. In: Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds. p. 23. ACM (2008)
2. Batista, G.E., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter 6(1), 20–29 (2004)
3. Bettenburg, N., Nagappan, M., Hassan, A.E.: Think locally, act globally: Improving defect and effort prediction models. In: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on. pp. 60–69. IEEE (2012)
4. Bissyandé, T.F., Lo, D., Jiang, L., Réveillere, L., Klein, J., Le Traon, Y.: Got issues? who cares about it? a large scale investigation of issue trackers from github. In: Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. pp. 188–197. IEEE (2013)
5. Fan, Q., Yu, Y., Yin, G., Wang, T., Wang, H.: Where is the road for issue reports classification based on text mining? In: Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on. pp. 121–130. IEEE (2017)
6. Gousios, G., Pinzger, M., Deursen, A.v.: An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering. pp. 345–355. ACM (2014)
7. He, P., Li, B., Ma, Y.: Towards cross-project defect prediction with imbalanced feature sets. arXiv preprint arXiv:1411.4228 (2014)
8. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining github. In: Proceedings of the 11th working conference on mining software repositories. pp. 92–101. ACM (2014)
9. Kitchenham, B.A., Mendes, E., Travassos, G.H.: Cross versus within-company cost estimation studies: A systematic review. IEEE Transactions on Software Engineering 33(5), 316–329 (2007)
10. Konietschke, F., Hothorn, L.A., Brunner, E., et al.: Rank-based multiple test procedures and simultaneous confidence intervals. Electronic Journal of Statistics 6, 738–759 (2012)
11. Konietschke, F., Placzek, M., Schaarschmidt, F., Hothorn, L.A.: nparcomp: an r software package for nonparametric multiple comparisons and simultaneous confidence intervals (2015)
12. Krishna, R., Menzies, T., Fu, W.: Too much automation? the bellwether effect and its implications for transfer learning. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. pp. 122–131. ACM (2016)
13. Lan, L., Tao, D., Gong, C., Guan, N., Luo, Z.: Online multi-object tracking by quadratic pseudo-boolean optimization. In: IJCAI. pp. 3396–3402 (2016)
14. Ma, Y., Luo, G., Zeng, X., Chen, A.: Transfer learning for cross-company software defect prediction. Information and Software Technology 54(3), 248–256 (2012)
15. Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., Cok, D.: Local vs. global models for effort estimation and defect prediction. In: Automated Software Engineering. pp. 343–351. IEEE (2011)
16. Merten, T., Falis, M., Hübner, P., Quirchmayr, T., Bürsner, S., Paech, B.: Software feature request detection in issue tracking systems. In: Requirements Engineering Conference (RE), 2016 IEEE 24th International. pp. 166–175. IEEE (2016)

17. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th international conference on Software engineering. pp. 452–461. ACM (2006)
18. Nam, J., Pan, S.J., Kim, S.: Transfer defect learning. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 382–391. IEEE Press (2013)
19. Pan, S.J., Tsang, I.W., Kwok, J.T., Yang, Q.: Domain adaptation via transfer component analysis. IEEE Transactions on Neural Networks 22(2), 199–210 (2011)
20. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22(10), 1345–1359 (2010)
21. Peters, F., Menzies, T., Marcus, A.: Better cross company defect prediction. In: Mining Software Repositories. pp. 409–418 (2013)
22. Posnett, D., Filkov, V., Devanbu, P.: Ecological inference in empirical software engineering. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. pp. 362–371. IEEE Computer Society (2011)
23. Premraj, R., Herzig, K.: Network versus code metrics to predict defects: A replication study. In: Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on. pp. 215–224. IEEE (2011)
24. Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J.: On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering 14(5), 540–578 (2009)
25. Uddin, J., Ghazali, R., Deris, M.M., Naseem, R., Shah, H.: A survey on bug prioritization. Artificial Intelligence Review 47(2), 145–180 (2017)
26. Van Der Veen, E., Gousios, G., Zaidman, A.: Automatically prioritizing pull requests. In: Proceedings of the 12th Working Conference on Mining Software Repositories. pp. 357–361. IEEE Press (2015)
27. Yu, Y., Wang, H., Filkov, V., Devanbu, P., Vasilescu, B.: Wait for it: Determinants of pull request evaluation latency on github. In: Mining software repositories (MSR), 2015 IEEE/ACM 12th working conference on. pp. 367–371. IEEE (2015)
28. Yu, Y., Wang, H., Yin, G., Wang, T.: Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? Information and Software Technology 74, 204–218 (2016)
29. Zanetti, M.S., Scholtes, I., Tessone, C.J., Schweitzer, F.: Categorizing bugs with social networks: a case study on four open source software communities. In: Proceedings of the 35th International Conference on Software Engineering. pp. 1032–1041. IEEE (2013)
30. Zhang, F., Mockus, A., Keivanloo, I., Zou, Y.: Towards building a universal defect prediction model. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 182–191. ACM (2014)
31. Zhang, F., Zheng, Q., Zou, Y., Hassan, A.E.: Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the 38th International Conference on Software Engineering. pp. 309–320. ACM (2016)
32. Zhou, Y., Tong, Y., Gu, R., Gall, H.: Combining text mining and data mining for bug report classification. Journal of Software: Evolution and Process 28(3), 150–176 (2016)
33. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B.: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 91–100. ACM (2009)