

# Towards Effectively Test Report Classification to Assist Crowdsourced Testing

Junjie Wang<sup>1</sup>, Qiang Cui<sup>1</sup>, Qing Wang<sup>1,2</sup>, Song Wang<sup>3</sup>

<sup>1</sup>Laboratory for Internet Software Technologies, <sup>2</sup>State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China

<sup>3</sup>Electrical and Computer Engineering, University of Waterloo, Canada

{wangjunjie, cuiqiang, wq}@itechs.iscas.ac.cn, song.wang@uwaterloo.ca

## ABSTRACT

**Background:** Automatic classification of crowdsourced test reports is important due to their tremendous sizes and large proportion of noises. Most existing approaches towards this problem focus on examining the performance of different machine learning or information retrieval techniques, and most are evaluated on open source dataset. However, our observation reveals that these approaches generate poor and unstable performances on real industrial crowdsourced testing data. We further analyze the deep reason and find that industrial data have significant local bias, which degrades existing approaches.

**Aims:** We aim at designing an effective approach to overcome the local bias in real industrial data and automatically classifying *true fault* from the large amounts of crowdsourced reports.

**Method:** We propose a cluster-based classification approach, which first clusters similar reports together and then builds classifiers based on most similar clusters with ensemble method.

**Results:** Evaluation is conducted on 15,095 test reports of 35 industrial projects from Chinese largest crowdsourced testing platform and results are promising, with 0.89 precision and 0.97 recall on average. In addition, our approach improves the existing baselines by 17% - 63% in precision and 15% - 61% in recall.

**Conclusions:** Results imply that our approach can effectively discriminate *true fault* from large amounts of crowdsourced reports, which can reduce the effort required for manually inspecting the reports and facilitate project management in crowdsourced testing. To the best of our knowledge, this is the first work to address the test report classification problem in real industrial crowdsourced testing practice.

## Keywords

Crowdsourced testing; Report classification; Cluster

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

ESEM '2016 Ciudad Real, Spain

© 2016 ACM. ISBN 123-4567-24-567/08/06...\$15.00

DOI: 10.475/123.4

## 1. INTRODUCTION

Crowdsourced testing is an emerging trend in both the software engineering community and industrial practice. In crowdsourced testing, crowd workers are required to submit test reports after performing testing tasks in crowdsourced platform. A typical report contains description, screenshots, and an assessment as to whether the worker believed that the software behaved correctly or behaved incorrectly (i.e., *passed* or *failed*). In order to attract workers, testing tasks are often financially compensated, especially for these revealed failed behaviors [5]. In this context, workers may submit thousands of test reports due to financial incentive and other motivations. The Baidu crowdsourced testing platform delivers approximately 100 projects per month, and receives more than 1,000 test reports per day on average. Among those reports, more than 70% are reported as *failed*. However, these test reports often have many false positives, i.e., a test report marked as *failed* that actually described correct behavior or behavior that was considered outside of the studied software system.

Project managers or testers need manually inspect these failed test reports to judge whether they actually reveal faults—*true fault*. Inspecting 1,000 reports takes almost half a working week for a tester on average. Besides, only less than 50% of them are finally determined as true fault [5]. Hence, the process is time-consuming, tedious, and low-efficient.

Therefore, it would be beneficial to automatically classify *true fault* from the large amounts of test reports. There have been various existing researches focused on classifying issue reports of open source projects [13][16][22][24][25][27]. Nevertheless, crowdsourced reports are more noise than issue reports. There are two reasons. First, most of the test reports are submitted by non-specialized crowd workers. Second, crowd workers often work under financial incentives and tend to submit many reports without caring their quality. Hence, classifying them is more valuable, yet possesses more challenges. Furthermore, most existing approaches focus on examining the performance of different machine learning or information retrieval techniques, and most are evaluated on open source dataset.

However, our observation on real industrial crowdsourced testing data reveals that previous approaches generate poor and unstable performances on these industrial data. We further analyze the deep reason and find that industrial data have significant local bias, i.e., data are heterogeneous within dataset and their distributions are often different from one project to another (details are in Section 6.1).

To address such problem, we propose a cluster-based approach for effectively test report classification. Our idea behind is to cluster similar crowdsourced reports together and build machine learning classifiers based on most similar training clusters with ensemble method (details are in Section 3.3).

We experimentally investigate the feasibility of the proposed approach on 15,095 test reports of 35 projects from Chinese largest crowdsourced testing platform. Our approach demonstrates the advantages on both precision and recall, with 0.89 precision and 0.97 recall on average, which is 17% - 63% higher in precision and 15% - 61% higher in recall than existing baselines. This implies that our approach can effectively discriminate *true fault* from large amounts of crowdsourced reports, which can reduce the effort required for manually inspecting the reports and facilitate project management in crowdsourced testing. Additionally, we explore the relative contribution of different features (i.e., term, sentiment, worker experience) on report classification.

This study makes the following contributions:

- We investigate the problem of crowdsourced test report classification and reveal its local bias, which is different from traditional or open source issue report classification in prior researches. **To the best of our knowledge, this is the first work to address the test report classification problem in real industrial crowdsourced testing practice.**
- We propose a cluster-based classification approach to overcome the local dilemma.
- We evaluate our approach on 15,095 test reports of 35 projects from Chinese largest crowdsourced testing platform. The experimental results indicate its feasibility in terms of high precision and recall.

The rest of this paper is organized as follows. Section 2 provides background of crowdsourced testing. Section 3 describes our proposed approach. After showing the experimental setup in Section 4, we demonstrate the results and analysis in Section 5. Section 6 and Section 7 present discussion and related work respectively. We conclude this paper in Section 8.

## 2. BACKGROUND

In this section, we describe the background of crowdsourced testing to help better understand the challenges we meet in real industrial crowdsourced testing practice.

Our experiment is conducted with Baidu crowdsourced testing platform<sup>1</sup>. The general procedure of such crowdsourced testing platform is shown in Figure 1.

In general, testers in Baidu prepare packages for crowdsourced testing (software under test and testing tasks) and distribute them online using their crowdsourced testing platform. Then, crowd workers could sign in to conduct the task and are required to submit crowdsourced test reports<sup>2</sup>. Table 1 demonstrates the attributes of a typical crowdsourced

<sup>1</sup>Baidu (baidu.com) is the largest Chinese search service provider. Its crowdsourcing test platform (test.baidu.com) is also the largest one in China.

<sup>2</sup>We will simplify “crowdsourced test report” as “crowdsourced report” or “report”, potentially avoiding the confusion with “test set” in machine learning techniques.

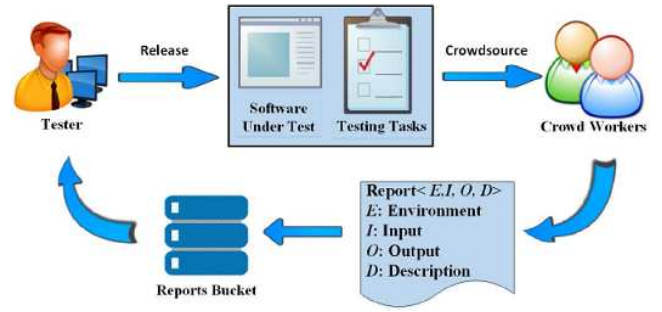



Figure 1: The procedure of crowdsourced testing [5]

Table 1: An example of crowdsourced test report

Attribute	Description: <i>example</i>
<b>Environment</b>	Phone type: <i>Samsung SN9009</i> Operating system: <i>Android 4.4.2</i> ROM information: <i>KOT49H.N9009</i> Network environment: <i>WIFI</i>
<b>Crowd worker</b>	Id: <i>123456</i> Location: <i>Beijing Haidian District</i>
<b>Testing task</b>	Id: <i>01</i> Name: <i>Incognito mode</i>
<b>Input and operation steps</b>	<i>Input "sina.com.cn" in the browser, then click the first news. Select "Setting" and then set "Incognito Mode". Click the second news in the website. Select "Setting" and then select "History".</i>
<b>Result description</b>	<i>"Incognito Mode" does not work as expected. The first news, which should be recorded, does not appear in "History".</i>
<b>Screenshot</b>	
<b>Assessment</b>	Passed or failed given by crowd worker: <i>Failed</i>

report<sup>3</sup>. The platform can automatically record the crowd worker information and environment information on which the test is carried on. A worker is required to submit the testing task s/he carried on and descriptions about the task including input and operation steps, results description and screenshots. The report is also accompanied with an assessment as to whether the worker believes that the software behaved correctly (i.e., *passed*) or incorrectly (i.e., *failed*).

In order to attract more workers, testing tasks are often financially compensated. Workers may then submit thousands of test reports due to financial incentive and other motivations. Usually, this platform delivers approximately 100 projects per month, and receives more than 1,000 test reports per day on average. Among those reports, more than 70% are reported as *failed*. Nevertheless, they have many false positives, i.e., a test report marked as *failed* that actually described correct behavior or behavior outside of the studied software system. This is due to the financial compensation mechanism, which favors reports revealing failed behaviors.

Testers need to manually inspect these failed test reports to judge whether they actually reveal faults—*true fault*. However, inspecting 1,000 reports manually could take almost half a working week for a tester. Besides, only less than

<sup>3</sup>Reports are written in Chinese in our projects. We translate them into English to facilitate understanding.

50% of them are finally determined as true fault. Obviously, such process is time-consuming, tedious, and low-efficient. In consequence, this motivates us to efficiently automate the classification of crowdsourced test reports.

### 3. APPROACH

As we mentioned that most existing issue classification approaches focus on examining the performance of different machine learning or information retrieval techniques, and most are evaluated on open source dataset [13][16][22][24][25][27]. However, our experiments show that these approaches generate poor and unstable performances on our industrial crowdsourced testing data (see details in Section 5.1), because of the significant local bias in industrial data, i.e., data are heterogenous within dataset and their distributions are often different from one project to another.

Hence, we propose a cluster-based classification approach to overcome the local dilemma. Figure 2 illustrates the overview of our approach. Firstly, we carry out feature extraction, which is to obtain features from crowdsourced report to train machine learning classifiers. Secondly, we cluster similar reports together to construct more homogeneous training dataset. Thirdly, we build classifiers based on most similar clusters and combine classification results from these classifiers. In the following subsections, we explain each step in detail.

#### 3.1 Extracting Features

The goal of feature extraction is to obtain features from crowdsourced reports which can be potentially used as input to train machine learning classifiers. We abstract these features from the following three dimensions: **textual**, **sentiment**, and **crowd worker experience**. Each dimension may contain a set of features.

For **textual** dimension, we first collect different sources of text description together (input and operation steps, result descriptions). Then we conduct **word segmentation**, as the crowdsourced reports in our experiment are written in Chinese. We adopt ICTCLAS<sup>4</sup> for word segmentation, and segment descriptions into words. We then remove **stop-words** (i.e., “the”, “am”, “on”, etc.) to reduce noise. Note that, workers often use different words to express the same concept, so we introduce the **synonym replacement** technique to mitigate this problem. Synonym library of LTP<sup>5</sup> is adopted.

Each of the remaining word token corresponds to a feature. For each feature, we take the frequency it occurs in the description as its value. We used the TF (term frequency) instead of TF-IDF because the use of the inverse document frequency (IDF) penalizes terms appearing in many reports. In our work, we are not interested in penalizing such terms (e.g., “break”, “problem”) that actually appear in many reports because they can act as discriminative features that guide machine learning techniques in classifying reports.

These **textual** features describe what the report is and play a significant role in revealing true fault. Most of the previous researches would utilize text related features to conduct the report classification problem in open source soft-

<sup>4</sup>ICTCLAS (<http://ictclas.nlpir.org/>) is widely used Chinese NLP platform.

<sup>5</sup>LTP (<http://www.ltp-cloud.com/>) is considered as one of the best cloud-based Chinese NLP platforms.

Table 2: Features used in this work

Dimension	Feature	Num
Textual	Words segmented from description excluding stop words and replaced with synonym	2262 in our experiment
Sentiment	Positive score Negative score Combined score	3
Crowd worker experience	Number of reports Number of true faults Percentage of true faults	3

ware [16][22][27]. This is why we bring these features in our classification model.

For **sentiment** dimension, we apply SentiStrength [21], which has been successfully used in app review of English corpus, to crowdsourced report for sentiment analysis. For each report, SentiStrength assigns one negative sentiment score in a scale of -5 to -1 and one positive score in a scale of 1 to 5. The English Sentiment Words List is substituted with Chinese ones provided by HowNet<sup>6</sup>. We retrieve three features: the positive score, negative score and combined score for each crowdsourced report.

Intuitively, crowdsourced reports usually reflect users’ positive or negative emotions. For example a true fault will probably include a negative sentiment. Prior researches for classifying app reviews [12][18] also leverage sentiment analysis to improve their results. As both app reviews and our crowdsourced reports are written by non-specialized people, we assume sentiment related features might contribute to report classification.

For **crowd worker experience** dimension, we capture three features to measure crowd worker’s experience: the number of reports submitted, the number of true faults submitted, and the percentage of true faults among one’s submitted reports.

We extract these features based on the hypothesis that if the crowd worker often reported high reliable crowdsourced reports, he might continue submitting true faults. Note that, some researches also consider the name of reporter or assignee of a issue report as a feature to assist the classification problem [24][27]. However, simply using the crowd worker itself might not work well in our context, because numerous workers only submit very few reports. Thus, currently we model worker’s experience only by these three features.

We organize all the features under investigated (Table 2) into a **feature vector**. For each crowdsourced report, we extract the value for each feature and prepare the feature vector for building machine learning classification models.

#### 3.2 Clustering Reports

For mitigating the local bias of our industrial crowdsourced data, we perform clustering techniques to ensure the homogeneity within training data. Intuitively, the crowdsourced reports within the same cluster tend to homogeneous with each other. Specifically, we apply K-means algorithm [1] to separate all training data into different clusters. Note that, we break the boundary exerted by projects and reorganize all available crowdsourced reports into clusters.

Given feature vectors for crowdsourced reports of all train-

<sup>6</sup>HowNet (<http://keenage.com/>) is the largest Chinese knowledge database.

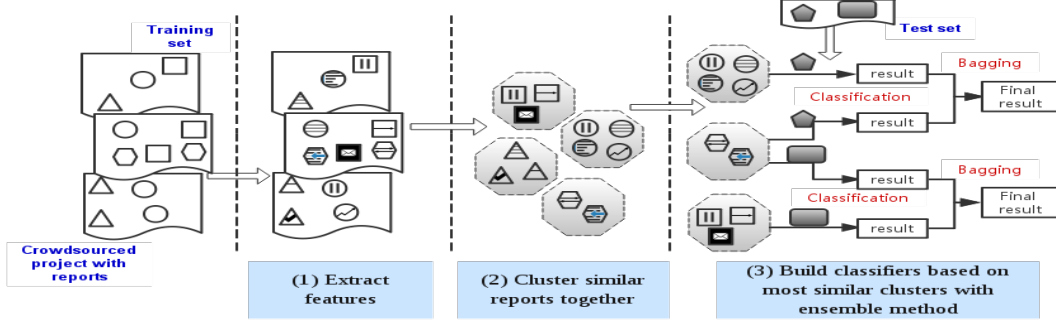


Figure 2: Overview of our cluster-based classification approach

ing data, number  $K$  of desired clusters, K-means first selects one centroid for each cluster, and then associates each feature vector with the nearest centroid, thus identifying clusters. We apply cosine similarity to measure the distance because prior study showed that it performs better for high-dimensional text documents than other distance measures (e.g., euclidean distance) [6].

Furthermore, we utilize the top  $para\_F$  descriptive features to represent the cluster, denoted as **descriptive vector**. The descriptive features are determined by selecting the features that contribute the most to the average similarity between the objects of each cluster.

One of the challenges of K-means algorithm is to estimate the number of clusters that should be created. To identify the best solution, we use the **element silhouette** [19]. The silhouette of an element is the measure of how closely the element is matched to the other elements within its cluster, and how loosely it is matched to other elements of the neighboring clusters. When the value of the silhouette of an element is close to 1, it means that the element is in the appropriate cluster. If the value is close to -1, instead, it means that the element is in the wrong cluster. Thus, to identify the best solution, we compute the average of the elements' silhouette for each solution using  $K$  as the number of clusters, and we select the solution whose average silhouette was closest to 1. In our experiment, candidate  $K$  is ranged from 1 to 1000.

### 3.3 Building Classifiers

To further mitigate the influence of local bias in our classification model, we choose the training data from the most similar clusters to build classifiers. In addition, it is natural that classifiers learned from different clusters place different emphasis on predictive features. So to combine the knowledge from multiple sources, we adopt the bagging ensemble method to combine the classification results from these classifiers.

We have noted that reports from the same crowdsourced project can be dramatically different in their descriptions. This can result in different reports corresponding to different set of most similar clusters. For this reason, our approach would find most similar clusters and build classifiers for each crowdsourced report, not the whole project.

When a new test set is coming, for each crowdsourced report, we first retrieve the feature vector. Then we compute the similarity with each descriptive vector of corresponding cluster in training set. We select the top  $para\_C$  most simi-

lar clusters and with similarity value higher than  $thres\_sim$ . Note that, we restrict the selected clusters with both number and similarity value. This is because we try to avoid the classification results from quite dissimilar clusters. For each selected cluster, we build machine learning classifier based on feature vectors of the crowdsourced reports in that cluster. The classifiers are then used to classify the report and classification results are combined using bagging.

Bagging is essentially a method to eliminate the potential prediction bias caused by each individual model through integrating their predictive power [3]. For the crowdsourced report under test, voting score is computed as follows:

$$score = \frac{\sum_{i=1}^p score_i}{p}$$

where  $score_i$  is 1 if the classification result from  $i_{th}$  model claims *true fault*, otherwise  $score_i$  is 0;  $p$  is number of selected models. If voting score is bigger than  $thres\_vot$ , the final classification result of that report will be set as "true fault".

## 4. EXPERIMENT SETUP

### 4.1 Research Questions

To systematically evaluate our cluster-based classification approach, we set four research questions.

- **RQ1:** How effective of our cluster-based classification approach in classifying crowdsourced reports?

We additionally conduct five comparison experiments (Section 4.3) to demonstrate the effectiveness of our approach.

- **RQ2:** Which classification algorithm works better?

We employ four machine learning algorithms to conduct classification (Section 4.4). This question aims at investigating whether these classification algorithms behave differently on model performance.

- **RQ3:** What is the impact of cluster number on model performance?

We have applied element silhouette to determine the cluster number  $K$ . To investigate whether there is alternative  $K$  which can achieve significant better results, we conduct experiments with  $K$  ranging from 1 to 1,000.

- **RQ4:** What is the effect of each dimension of features on model performance, used in isolation and in combination?

We utilize features involved in three dimensions (textual, sentiment, crowd worker experience shown in Table 2) to conduct the classification. To investigate the relative effect of features in each dimension, we build classifiers using features in isolation and in combination. We aim at investigating which dimension of features contribute more to the classification results.

## 4.2 Data Collection

Our experiment is based on crowdsourced reports from the repositories of Baidu crowdsourced testing platform. We collect all crowdsourced testing projects closed between Oct. 20th 2015 and Oct. 30th 2015. There are totally 35 projects covered 8 categories. Table 3 provides more details with total number of crowdsourced reports submitted ( $\#$ ), number of *failed* reports ( $\#Fa$ ), number and ratio of reports judged as *true fault* ( $\#TF$ ,  $\%TF$ ). Due to commercial consideration, we replace detailed project names with serial number. The serial number is determined based on the chronological order of its closed time. These projects are organized into categories corresponding to those in the platform.

Note that, our classification is conducted on *failed* reports, not the complete set. This is because through talking with testers in the company, we find that workers tended to label correct behavior as *failed* report, while almost none of the true fault was labeled as *passed*, due to financial incentives (failed reports can often receive higher rewards).

We use the judgement attributes (Table 1) to construct the ground truth of classification. We additionally verify its validity through randomly sampling and relabeling. Specifically, we randomly select one project from each category, and sample 10% of crowdsourced reports from each selected project. A tester from the company is asked to relabel the data, without knowing previous label. We then compare the difference between prior label and new label. The number of different labeling for each projects are all below 4%.

## 4.3 Experimental Design with Baselines

For the 35 projects under investigation (Table 3), we divide them into 5 folds according to the chronological order, i.e., P1 to P7 in 1st fold, P8 to P14 in 2nd fold, p29 to p35 in 5th fold. When classifying a project, the candidate training data are these projects from all its preceding folds. To be more specific, if we want to classify the crowdsourced reports in P8 (*P30*), we can only utilize the data from P1 to P7 (*P1 to P28*). In this way, we only present the classification results of P8 to P35, as projects in 1st fold did not have any training data.

We set parameters *para\_F* as 30, *para\_C* as 5, *thres\_sim* as 0.2, *thres\_vot* as 0.5 based on our experimental outcomes.

We compare our **CIUster-based REport claSsification approach (CURES)** to five baselines.

The first baseline relates with the classification without considering the local bias of dataset, which is the common practice in prior researches [22][24].

**Classification based on all available training data (U\_Total).** It represents a very intuitive prediction setting: combine all available crowdsourced reports into a big training set without any data selection operation, and then learn a classifier from it.

Table 3: Projects under investigation

	#	#Fa	#TF	%TF		#	#Fa	#TF	%TF
<b>Tools</b>									
<b>P1</b>	321	267	183	68.5	<b>P2</b>	874	717	144	20.1
<b>P8</b>	688	647	253	39.1	<b>P15</b>	217	157	46	29.3
<b>P22</b>	802	672	177	26.3	<b>P23</b>	466	402	102	25.3
<b>P29</b>	391	317	253	35.2	<b>P30</b>	390	334	87	26.0
<b>P31</b>	495	417	83	19.9					
<b>Entertainment</b>									
<b>P3</b>	302	168	125	74.4	<b>P9</b>	1094	727	447	61.4
<b>P16</b>	423	272	250	91.9	<b>P17</b>	556	398	251	63.1
<b>P24</b>	1414	1034	500	48.3	<b>P25</b>	1502	1152	583	50.1
<b>P32</b>	832	754	465	61.7					
<b>Efficiency</b>									
<b>P4</b>	216	144	31	21.5	<b>P10</b>	504	394	157	39.8
<b>Finance</b>									
<b>P13</b>	436	232	165	71.1					
<b>Music</b>									
<b>P5</b>	683	486	51	10.4	<b>P11</b>	320	163	61	37.4
<b>P18</b>	815	667	262	39.3	<b>P26</b>	342	181	76	41.9
<b>News</b>									
<b>P6</b>	492	455	280	61.5	<b>P19</b>	307	180	122	67.8
<b>P27</b>	824	503	131	26.0	<b>P33</b>	806	522	199	38.1
<b>Photo and Video</b>									
<b>P12</b>	637	537	220	40.9	<b>P20</b>	632	545	183	33.5
<b>P21</b>	580	323	112	34.7	<b>P34</b>	358	93	49	52.7
<b>Read</b>									
<b>P7</b>	403	304	25	8.2	<b>P14</b>	297	223	129	57.8
<b>P28</b>	524	424	163	38.4	<b>P35</b>	452	284	58	20.4
<b>Summary</b>									
<b>#</b>		<b>#Fa</b>		<b>#TF</b>		<b>%TF</b>			
<b>20,395</b>		<b>15,095</b>		<b>6,423</b>		<b>42.5</b>			

The second baseline relates with the classification using a straightforward data selection method to overcome local bias.

**Classification based on training data within category (U\_Catg).** It represents combing all available crowdsourced reports in the same category with test set, and then learning a classifier from it. It is designed considering that projects within same category tend to be more similar with each other so as to improve the classification performance. Note that, for projects without satisfied training data (e.g., P12, P13), we adopt the average values under this scenario as their performance.

The third and fourth baselines are about the classification with data selection, but using more straightforward data organization methods than cluster-based approach.

**Classification based on random-partition and data selection (S\_Rand).** It randomly separates the training set into K parts and then conducts classification as step 3 of cluster-based approach. K is set accorded with cluster-based approach. This setup is to compare the performance of random separation and cluster-based separation of training data.

**Classification based on project-partition and data selection (S\_Prj).** It separates the training set just as it was by projects and then conducts classification as step 3 of cluster-based approach. This setup is to examine whether the pre-existing boundary of training data can work as well as cluster-based separation.

The fifth baseline concerns with the most common validation method in previous researches [16][27], which is actually unattainable in real practice.

**Cross validation within project (CV\_Prj).** It represents within project validation scenario, i.e., 10-fold cross validation on test set. It is often treated as the theoretical best performance.

## 4.4 Classification Algorithms

We employ four machine learning algorithms to construct classifiers. They are Naive Bayes (NB), Decision Tree (J48), Support Vector Machine (SVM), and Logistic Regression (LR).

Naive Bayes is a very popular algorithm, which is based on applying Bayes’ theorem with strong independence assumptions between the features [1]. Decision Tree learning is another popular classification algorithm [11]. It builds the tree structure from the training data by using information entropy. Leaves in the tree structure represent classifications and branches represent judgment rules. The SVM learner performs classification by finding the optimal hyper-plane that maximally separates samples in two different classes [11]. The Logistic Regression measures the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution [1].

## 4.5 Evaluation Metrics

To evaluate our proposed approaches, we use three metrics: *precision*, *recall*, and *F – Measure*. These three metrics are widely adopted to evaluate issue report classification techniques [16][22][25][27]. To explain these measures, we use the following contingency table.

		Observed	
		True fault	Not true fault
Predicted	True fault	TP	FP
	Not true fault	FN	TN

Precision  $TP/(TP+FP)$  denotes the percentage of crowd-sourced reports observed as true faults that are classified correctly.

Recall  $TP/(TP+FN)$  denotes the percentage of crowd-sourced reports classified as true faults that are classified correctly.

We also calculated the F-Measure, which is the harmonic mean of precision and recall, providing a single accuracy measure.

## 5. RESULTS AND ANALYSIS

### 5.1 Addressing RQ1

RQ1: How effective of our cluster-based classification approach in classifying crowdsourced reports?

Figure 3 illustrates the performance of our cluster-based classification approach (CURES), as well as five baselines.

For all four classification algorithms, CURES achieves best performance in terms of precision and recall. For models using Naive Bayes, our approach can achieve 0.89 of precision and 0.97 of recall on average. In other words, among the crowdsourced reports we classified as true fault, 89% of them are really true faults and they can cover 97% of total true faults. Furthermore, results of Naive Bayes show that CURES improves the baselines by 17% - 63% in precision and 15% - 61% in recall. However, as the recall does not reach 100%, indicating our method might have missed few vulnerabilities. We will investigate how to improve the recall value without sacrificing much in precision, in our future work.

**Analysis of the projects with low performance.** We have carefully examined the projects with low performance

values, e.g., P11, P15 and P23. Project P11 is a new-released cloud music service provider, with immature music database. An alarming number of *failed* reports are about failing to find particular songs, which is actually not fault. There are also a large part of *failed* reports related with the interruption when playing or downloading particular songs, which is actually outside the scope of this software (mainly due to the slow network). But the diverse words from song names, which rarely show up in other projects, bring noises for the classification. The low performance values of other two projects are originated from similar reason. To summarize, we assume the decline in performance values is due to the data drift problem in learning tasks [23]. We plan to introduce other techniques (e.g., transfer learning) to overcome the data drift problem.

**Comparison of CURES with U\_Total and U\_Catg.** CURES conducts careful data selection process, while U\_Total and U\_Catg do not. Intuitively we can see that, performance of U\_Total produce very bad classification results in terms of precision and recall. This leads to the finding that learning from all available data without data filtering will lead to bad performance. The low and unstable performances of U\_Catg further confirm that building classifier without careful data selection is not a good practice.

We carefully examine the data and find that the bad performance of U\_Total and U\_Catg might due to the local bias (i.e., data heterogeneity) in crowdsourced report. More specifically, the term appearance and distribution in different crowdsourced categories may differ a lot. For example, classification features in entertainment category (such as “play”, “load”) are often irrelevant to the fault behavior of projects in tool category. Even in the same category, as there are different projects (e.g., ranging from animation to adventure game in entertainment category), this can also happen quite often. When data from multiple projects are combined, the cumulative effect of these extreme cases on overall model would increase significantly.

**Comparison of CURES with S\_Rand and S\_Prj.** CURES clusters similar reports together, while S\_Rand and S\_Prj organize reports in more straightforward ways. For all four classification algorithms, CURES achieves higher and more stable performances than S\_Rand and S\_Prj. This implies that through clustering similar reports together, our approach can build better classifier based on more homogeneous training data. Moreover, separating training data randomly or as projects themselves cannot effectively organize training data, thus potentially reduce classification performance.

**Comparison of S\_Prj with CV\_Prj.** We also note that S\_Prj produces better performance results than CV\_Prj. This maybe because combining results produced by classifiers of nearest projects can potentially avoid the prediction bias caused by the classifier of each project. For classification within project, some crowdsourced projects can contain several different testing tasks, which might be the root of bad performance.

**Impact of training data size on performance.** We additionally present the statistics of performance values under Naive Bayes for each fold in Table 4. Results reveal that performances tend to go higher as available training data increases, especially for precision and F-Measure. But this does not always hold true, especially for recall. We will investigate the influence of training data size on model perfor-

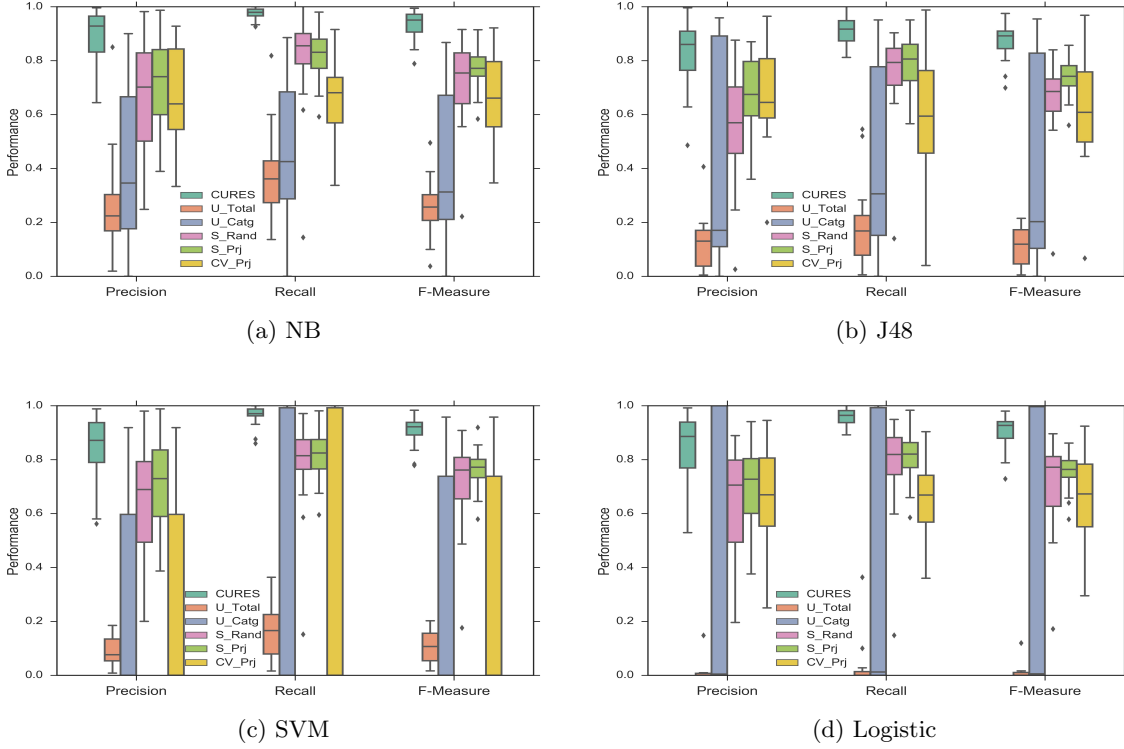


Figure 3: Classification performance for RQ1&RQ2

Table 4: Statistics of performance using NB for each fold

		Min	Max	Mean	Median
Precision	2nd fold	0.64	0.89	0.81	0.83
	3rd fold	0.70	<b>0.99</b>	0.90	0.95
	4th fold	0.72	0.96	0.88	0.94
	5th fold	<b>0.91</b>	<b>0.99</b>	<b>0.96</b>	<b>0.98</b>
	all	0.64	<b>0.99</b>	0.89	0.92
Recall	2nd fold	0.93	<b>1.00</b>	<b>0.97</b>	<b>0.98</b>
	3rd fold	0.94	0.99	<b>0.97</b>	<b>0.98</b>
	4th fold	0.92	<b>1.00</b>	0.96	0.97
	5th fold	<b>0.96</b>	<b>1.00</b>	<b>0.97</b>	0.97
	all	0.92	<b>1.00</b>	<b>0.97</b>	0.97
F-Measure	2nd fold	0.78	0.94	0.89	0.90
	3rd fold	0.84	<b>0.99</b>	0.94	0.96
	4th fold	0.84	0.97	0.92	0.94
	5th fold	<b>0.94</b>	<b>0.99</b>	<b>0.96</b>	<b>0.97</b>
	all	0.78	<b>0.99</b>	0.93	0.95

mance in more detail, and try to recommend the minimum volume of training data for effective classification.

## 5.2 Addressing RQ2

RQ2: Which classification algorithm works better?

Figure 3 shows that classification models built using Naive Bayes produce highest precision and recall, followed by SVM and Logistic Regression, with Decision Tree producing lowest performance values. Besides, results produced by SVM are more unstable than others. Similar with prior machine learning researches [7], our results imply that the choice of classification techniques has an impact on the classification performance. Therefore, given the ease of access to such techniques nowadays in the machine learning toolboxes (e.g., Weka and R), exploring various techniques is encouraged. In following section, we will only present results of

Naive Bayes for space limit, as the performance accords with the observation presented here.

## 5.3 Addressing RQ3

RQ3: What is the impact of cluster number on model performance?

Due to limited space, we only present classification performance under certain representative K values in Figure 4. Note that, *best* in Figure 4 denotes the optimal K value determined by element silhouette. *Best* are respectively 58, 95, 122, 146 for 2nd to 5th fold in our context. By demonstrating classification performance under K values of *best-60*, *best-40*, *best-20*, *best*, *best+20*, *best+40*, *best+60*, we want to illustrate the trend of performance values with different cluster number. Note that, for the 2nd fold, as *best-60* falls to negative number, we substitute it with 1.

We can find that the optimal K values suggested by our approach can produce best and most stable performance results. Too small or large number of clusters will lower the performance in both precision and recall. This may because small number of clusters can not separate different topics of crowdsourced reports thoroughly, while too many clusters would result in the overfitting of the classification models.

We further conduct experiment for models under K values between *best-20* and *best+20*. For each experimental project, we compare the F-Measure value between models under *best* and experimental K. Figure 5 gives an overview of results, where *Win* indicates that performance under best K is better than experimental K (F-Measure is bigger), *Tie* means no difference observed (absolute difference of F-measure is smaller than 0.05), and *Loss* means experimental K is bet-



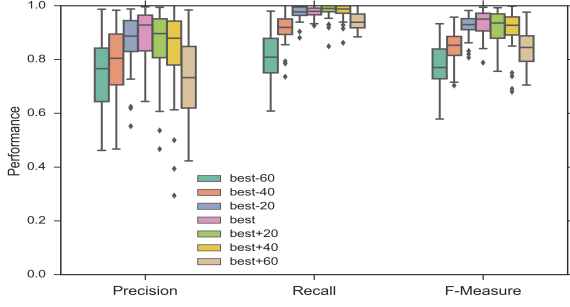


Figure 4: Classification performance for RQ3

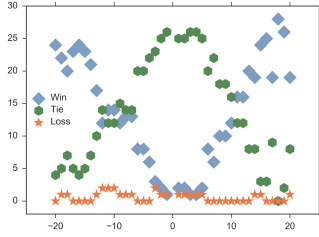


Figure 5: Comparison of model performance for RQ3

ter.

Results reveal that models under nearby K values demonstrate similar performance, denoting with high tie number. Models under the optimal K values suggested by our approach can produce relatively best performance, with 32 tie, 2 win and 1 loss under worse condition.

## 5.4 Addressing RQ4

RQ4: What is the effect of each dimension of features on model performance, used in isolation and in combination?

Figure 6 illustrates the performance values under different feature dimensions. Note that, because of the quite low performance values and space limit, we do not present the performance of these models built with features of sentiment and crowd worker experience, alone or combined.

Intuitively, models built on features combining all three dimensions achieve the highest precision and recall. The difference among other combinations can not be observed easily. Hence, we employ Mann-Whitney U test to compare the performance of different combination of feature dimensions. Table 5 gives an overview of the significance level.

Compared with models building solely on textual features, adding one dimension of features (*Text vs. TextExp*, *Text vs. TextSnt*) can not help improve the classification performance. Furthermore, for textual features, performance promotion by adding each of the other two feature dimensions (*TextExp vs. TextSnt*) exert no significant difference. However, combining all three dimensions of features together can significantly improve the performance (*Text vs. All*, *TextExp vs. All*, *TextSnt vs. All*, except recall value for *TextExp vs. All*).

That is to say, text related features almost act as the dominate role in report classification. This is in accordance with the observation in previous researches [12][18]. From cost efficiency point of view, as there are only little differences

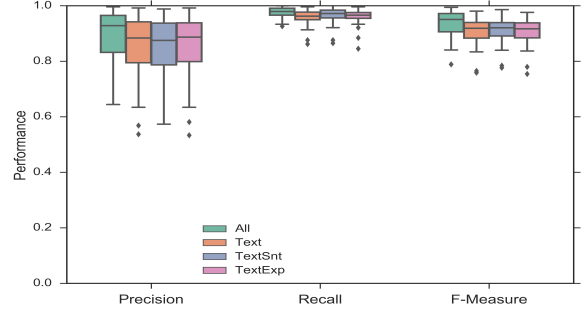


Figure 6: Classification performance for RQ4

between performance of *Text* and *All* (0.90 vs. 0.93 in F-measure), one can utilize textual features alone to conduct the classification.

Table 5: Significance level of MannWhitney U test for classification performance of different combination of feature dimensions

Feature dimension	Precision	Recall	F-Measure
Text vs. TextExp	0.747	0.091	0.865
Text vs. TextSnt	0.930	0.846	0.883
Text vs. All	0.008**	0.004**	0.001**
TextExp vs. TextSnt	0.869	0.137	0.778
TextExp vs. All	0.005**	0.100	0.001**
TextSnt vs. All	0.008**	0.005**	0.005**

Note: Significance level at 0.01 (p-value <0.01) are marked by \*\*

## 6. DISCUSSION

### 6.1 Local Phenomenon

The local phenomenon has been investigated by Menzies et al. [14][15] in defect prediction and effort estimation. They investigated the local treatment (apply local data in the adjacent cluster for modeling) versus global treatment (apply all global data for modeling). Experimental results revealed that the treatments from local regions were different and superior to the global treatment. They then suggested a general framework to overcome the local dilemma: 1) ignore any existing local divisions into multiple sources, 2) cluster across all available data, 3) then learn from the cluster that are nearest to the test data. They believed this conclusion can translate to other software engineering tasks and encouraged researchers to conduct exploration.

This paper reveals the local phenomenon in crowdsourced testing context experimentally. We further propose a cluster-based approach to overcome the local problem. Our approach extends and refines their framework in three ways. First, they adopted FASTMAP as clustering algorithm, which did not work well in our high dimension text classification context. So we apply K-means to conduct clustering. Second, they did not explicitly present how to determine the cluster number, which is crucial as it can influence model performance. Instead our approach includes the algorithm to decide the optimal cluster number and experimentally proves its efficiency. Third, their work only chose the nearest cluster to conduct prediction, while our classification is based on several most similar clusters with ensemble method. This can effectively reduce the prediction bias caused by each individual model.



## 6.2 Threats to Validity

The external threats concern the generality of this study. First, our experiment data consists of 35 projects covering 8 categories collected from Chinese largest crowdsourced testing platform. We can not assume a priori that the results of our study could generalize beyond this environment in which it was conducted. However, the various categories and size of data relatively reduce this risk. Second, all crowdsourced reports investigated in this study are written in Chinese, and we cannot assure that similar results can be observed on crowdsourced projects in other language. But this is alleviated as we did not conduct semantic comprehension, but rather simply tokenize sentence and use word as token for learning.

Regarding internal threats, we do not tune parameters of *para\_C*, *para\_F*, *thres\_sim*, *thres\_vot* in an experimental way, which may influence our results. But we have conducted experiment with several different values for each parameter. For example, if the number of clusters for bagging (*para\_C*) is set as 2, the classification would often fall into local regions so as to influence its performance. While *para\_C* is set 10, the classification requires more computational resources, while its outcomes remain unchanged. Nonetheless, there is need to conduct more carefully designed controlled experiment to investigate the influence of these parameters on model performance. In addition, we do not adopt feature selection process and use all the features to build classifier. This maybe the reason for the unstable performance of some baseline experiments with SVM. But the performance achieved by our approach is satisfying. Anyhow, we will conduct feature selection and investigate its influence on model performance.

Construct validity of this study mainly questions the data processing method. We rely on the judgement attribute of crowdsourced reports stored in repository to construct the ground truth. However, this is addressed to some extent due to the fact that testers in the company have no knowledge that this study will be performed for them to artificially modify their labeling. Besides, we have verified its validity through random sampling and relabeling.

## 7. RELATED WORK

### 7.1 Crowdsourced Testing

Crowdsourcing is the activity of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call [20]. Chen and Kim [4] applied crowdsourced testing to test case generation. They investigated object mutation and constraint solving issues underlying existing test generation tools, and presented a puzzle-based automatic testing environment. Musson et al. [17] proposed an approach, in which the crowd was used to measure real-world performance of software products. The work was presented with a case study of the Lync communication tool at Microsoft. Gomide et al. [9] proposed an approach that employed a deterministic automata to help usability testing. The idea is to capture crowd worker's biofeedback from mouse movements and a skin sensor, for revealing their hesitation behaviors. Adams et al. [8] proposed MoTIF to detect and reproduce context-related crashes in mobile apps after their deployment in the wild. All the studies above use crowdsourced testing to solve some problems in traditional

software testing activities. However, our approach is to solve the new encountered problem in crowdsourced testing.

Feng et al. [5] proposed test report prioritization methods for use in crowdsourced testing. They designed strategies to dynamically select the most risky and diversified test report for inspection in each iteration. There are two aspects distinguishing our work from theirs. On one hand, our approach can work unsupervised, without manual intervention, which can further reduce human effort. On the other hand, their method was evaluated only on 3 projects with students acting as crowd workers, while our evaluation is conducted in 35 projects from Chinese largest crowdsourced testing platform.

### 7.2 Issue Report Classification

Issue reports are valuable resources during software maintenance activities. Automated support for issue report classification can facilitate understanding, resource allocation and planning. Menzies and Marcus [16] proposed an automated severity assessment method by text mining and machine learning techniques. Tian et al. [22] propose DRONE, a multi factor analysis technique to classify the priority of bug reports. Wang et al. [25] proposed a technique combining natural language and execution information to detect duplicate failure reports. Zanetti et al. [26] proposed a method to classify valid bug reports based on nine measures quantifying the social embeddedness of bug reporters in the collaboration network. Zhou et al. [27] propose a hybrid approach by combining both text mining and data mining techniques of bug report data to automate the classification process. Wang et al. [24] proposed FixerCache, an unsupervised approach for bug triage by caching developers based on their activeness in components of products. Mao et al. [13] proposed content-based developer recommendation techniques for crowdsourcing tasks. Borg et al. [2] examined the impact of networks on issue report classification.

This work is to classify test report in crowdsourced testing, which is different from the aforementioned studies in two ways. Firstly, crowdsourced reports are more noise than issue reports, because they are submitted by non-specialized crowd workers and often under financial incentives. In this sense, classifying them is more valuable, yet possesses more challenges. Secondly, previous studies evaluated their methods on open source dataset and did not explore the local nature of dataset. However, this can not work well in our industrial context. This is why we propose this cluster-based approach.

There were researches to classify app reviews as bug reports, feature requests, user experiences, and ratings [12], or as feature request, problem discovery, information seeking and information giving [10][18], which can help to deal with the large amount of reviews. App reviews are often considered as issue reports by users, who behave unprofessionally as crowd workers in our context. But these related methods could not deal with the dataset with local bias, hence can not work in our context. Moreover, the performances of our cluster-based classification surpass theirs (F-measure is 0.72 on average in [18]) a lot.

## 8. CONCLUSIONS AND FUTURE WORK

This paper proposes a cluster-based classification approach to classify *true fault* from the large amounts of crowdsourced reports. This can potentially help to reduce the effort re-

quired for manually inspecting the reports and facilitate project management in crowdsourced testing. Our approach is designed to overcome the local bias of crowdsourced reports, which is a common phenomenon and has been investigated in other software engineering activities. Experiments on 15,095 test reports from Chinese largest crowdsourced testing platform are conducted and results prove its effectiveness.

It should be pointed out, however, that the presented material is just the starting point of the work in progress. We are closely collaborating with Baidu crowdsourced platform and planning to deploy the approach online. Returned results will further validate the effectiveness, as well as guide us in improving our approach. Future work will also include exploring transfer learning and other techniques to further improve the model performance and stability.

## 9. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under grant No.61432001, No.91318301, and No.91218302. We would like to thank the testers in Baidu for their great efforts in supporting this work.

## 10. REFERENCES

- [1] A. Berson, S. Smith, and K. Thearling. An overview of data mining techniques. *Building Data Mining Application for CRM*, 2004.
- [2] M. Borg, D. Pfahl, and P. Runeson. Analyzing networks of issue reports. In *CSMR 2013*, pages 79–88.
- [3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [4] N. Chen and S. Kim. Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles. In *ASE 2012*, pages 140–149.
- [5] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu. Test report prioritization to assist crowdsourced testing. In *ESEC/FSE 2015*, pages 225–236.
- [6] H. Finch. Comparison of distance measures in cluster analysis with dichotomous data. *Journal of Data Science*, 3:85–100, 2005.
- [7] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *ICSE 2015*, pages 789–800.
- [8] M. Gómez, R. Rouvoy, B. Adams, and L. Seinturier. Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring. In *Proceedings of the 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*.
- [9] V. H. M. Gomide, P. A. Valle, J. O. Ferreira, J. R. G. Barbosa, A. F. da Rocha, and T. M. G. d. A. Barbosa. Affective crowdsourcing applied to usability testing. *International Journal of Computer Science and Information Technologies*, 5(1):575–579, 2014.
- [10] E. Guzman, M. El-Halaby, and B. Bruegge. Ensemble methods for app review classification: An approach for software evolution. In *ASE 2015*, pages 771–776.
- [11] S. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.
- [12] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *RE 2015*, pages 116–125.
- [13] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman. Developer recommendation for crowdsourced software development tasks. In *SOSE 2015*, pages 347–356.
- [14] T. Menzies, A. Butcher, A. Marcus, D. Cok, F. Shull, B. Turhan, and T. Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on software engineering*, 39(6):822–834, 2013.
- [15] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok. Local vs. global models for effort estimation and defect prediction. In *ASE 2011*, pages 343–351.
- [16] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In *ICSM 2012*, pages 346–353.
- [17] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly. Leveraging the crowd: How 48,000 users helped improve lync performance. *IEEE Software*, 30(4):38–45, 2013.
- [18] S. Panichella, A. D. 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 *ICSM 2015*, pages 281–290.
- [19] P. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53–65, 1987.
- [20] K.-J. Stol and B. Fitzgerald. Two’s company, three’s a crowd: A case study of crowdsourcing software development. In *ICSE 2014*, pages 187–198.
- [21] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2588, 2010.
- [22] Y. Tian, D. Lo, and C. Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *ICSM 2013*, pages 200–209.
- [23] B. Turhan. On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering*, 17(1–2):62–74, 2012.
- [24] S. Wang, W. Zhang, and Q. Wang. Fixercache: Unsupervised caching active developers for diverse bug triage. In *ESEM 2014*, pages 25:1–25:10.
- [25] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE 2008*, pages 461–470.
- [26] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer. Categorizing bugs with social networks: A case study on four open source software communities. In *ICSE 2013*, pages 1032–1041.
- [27] Y. Zhou, Y. Tong, R. Gu, and H. Gall. Combining text mining and data mining for bug report classification. In *ICSM 2014*, pages 311–320.