Local-Based Active Classification of Test Report to Assist Crowdsourced Testing

Junjie Wang[1,3], Song Wang[4], Qiang Cui[1,3], Qing Wang[1,2,3]*, Mingshu Li[1,2,3], Jian Zhai[1,3]
[1]Laboratory for Internet Software Technologies, [2]State Key Laboratory of Computer Science,
Institute of Software Chinese Academy of Sciences, Beijing, China
[3]University of Chinese Academy of Sciences, Beijing, China
[4]Electrical and Computer Engineering, University of Waterloo, Canada
{wangjunjie, cuiqiang, wq, zhaijian}@itechs.iscas.ac.cn, song.wang@uwaterloo.ca

ABSTRACT

In crowdsourced testing, an important task is to identify the test reports that actually reveal fault – `true fault`, from the large number of test reports submitted by crowd workers. Most existing approaches towards this problem utilized supervised machine learning techniques, which often require users to manually label a large amount of training data. Such process is time-consuming and labor-intensive. Thus, reducing the onerous burden of manual labeling while still being able to achieve good performance is crucial. Active learning is one potential technique to address this challenge, which aims at training a good classifier with as few labeled data as possible. Nevertheless, our observation on real industrial data reveals that existing active learning approaches generate poor and unstable performances on crowdsourced testing data. We analyze the deep reason and find that the dataset has significant local biases.

To address the above problems, we propose LOcal-based Active ClassiFication (LOAF) to classify `true fault` from crowdsourced test reports. LOAF recommends a small portion of instances which are most informative within local neighborhood, and asks user their labels, then learns classifiers based on local neighborhood. Our evaluation on 14,609 test reports of 34 commercial projects from one of the Chinese largest crowdsourced testing platforms shows that our proposed LOAF can generate promising results. In addition, its performance is even better than existing supervised learning approaches which built on large amounts of labelled historical data. Moreover, we also implement our approach and evaluate its usefulness using real-world case studies. The feedbacks from testers demonstrate its practical value.

CCS Concepts

•Software and its engineering → Software testing and debugging;

*Corresponding author.

ASE'16, September 3–7, 2016, Singapore, Singapore
© 2016 ACM. 978-1-4503-3845-5/16/09...$15.00
http://dx.doi.org/10.1145/2970276.2970300

Keywords

Crowdsourced Testing, Test Report Classification, Active Learning

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 [1]. A typical test report contains description, screenshots, and an assessment as to whether the worker believed that the software behaved correctly (i.e., `passed`), or behaved incorrectly (i.e., `failed`). In order to attract workers, testing tasks are often financially compensated, especially for these failed reports. Under this context, workers may submit thousands of test reports. However, these test reports often have many false positives, i.e., a test report marked as failed that actually involves correct behavior or behavior that was considered outside of the studied software system. Project managers or testers need to manually verify whether a submitted test report reveals fault – `true fault` (as demonstrated in Table 1).

However, such process is tedious and cumbersome, given the high volume workload. Our observation on one of the Chinese largest crowdsourced testing platforms shows that, approximately 100 projects are delivered per month from this platform, and more than 1,000 test reports are submitted per day on average. Inspecting 1,000 reports usually takes almost half a working week for a tester. Thus, automatically classifying `true fault` from the large amounts of test reports would significantly facilitate this process.

Several existing researches have been proposed to classify issue reports of open source projects using supervised machine learning algorithms [2–5]. Unfortunately, these approaches often require users to manually label a large number of training data, which is both time-consuming and labor-intensive in practice. Therefore, it is crucial to reduce the onerous burden of manual labeling while still being able to achieve good performance.

In this paper, we try to adopt `active learning` to mitigate this challenge. Active learning aims to achieve high accuracy with as few labeled instances as possible. It recommends a small portion of instances which are most informative, and asks user their labels. These labeled reports are then used to train a model to classify the remaining reports. However, our experiments reveal that existing active learning techniques generate poor and unstable performances on these crowdsourced testing data (details are in Section 5.2). We further analyze the deep reason and find that previous active learning techniques assumed data are identically distributed [6]. However, crowdsourced test reports often have significant local bias, i.e., the contained technical terms may be shared by only a subset of reports rather than all the reports (details are in Section 2.3).

To address the local bias and more effectively classify the crowdsourced reports, we proposed `LO`cal-based `A`ctive Clas-si`F`ication approach (LOAF). The idea behind is to query the most informative instance within local neighborhood, then learn classifiers based on local neighborhood (details are in Section 3.2).

We experimentally investigate the effectiveness and advantages of LOAF on 14,609 test reports of 34 commercial projects from one of the Chinese largest crowdsourced testing platforms. Results show that LOAF can achieve 1.00 accuracy with the effort of labeling 10% reports on median. It outperforms both existing active learning techniques and supervised learning techniques which built on large amounts of historical labeled data. In addition, we implement our approach[1], conduct a case study and a survey, to further evaluate the usefulness of LOAF. Feedback shows that 80% testers agree with the usefulness of LOAF and would like to use it in real practice of crowdsourced testing.

These results imply that when historical labeled data are not available, our approach can still facilitate the report classification with high accuracy and little effort. This can reduce the effort for manually inspecting the reports, or collecting large number of high-quality labeled data.

This paper makes the following contributions:

- We propose `LO`cal-based `A`ctive Classi`F`ication (LOAF) to address the two challenges in automating crowdsourced test reports classification, i.e., local bias problem and lacking of labeled data. **To the best of our knowledge, this is the first work to address these two problems of automating test report classification in real industrial crowdsourced testing practice.**
- We evaluate our approach on 14,609 test reports of 34 projects from one of the Chinese largest crowdsourced testing platforms, and results are promising.
- We implement our approach and evaluate its usefulness using real-world case studies.

The rest of this paper are organized as follows. Section 2 describes the basic background of this study. Section 3 discusses the design of our proposed approach. Section 4 shows the setup of our experimental evaluation. Section 5 presents the results of our research questions. Section 6 discloses the threats to the validity of this work. Section 7 surveys related work. Finally, we summarize this paper in Section 8.

2. BACKGROUND

2.1 Crowdsourced Testing

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

[1]The implementation of LOAF are available at http://itechs.iscas.ac.cn/cn/membersHomepage/wangjunjie/wangjunjie/LOAF.zip.

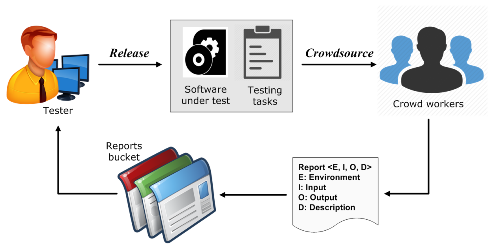


Figure 1: **The procedure of crowdsourced testing [1]**

Table 1: **An example of crowdsourced test report**

Attribute	**Description:** *example*
Environment	Phone type: *Samsung SN9009* Operating system: *Android 4.4.2* ROM information: *KOT49H.N9009* Network environment: *WIFI*
Crowd worker	Id: *123456* Location: *Beijing Haidian District*
Testing task	Id: *01* Name: *Incognito mode*
Input and operation steps	*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".*
Result description	*"Incognito Mode" does not work as expected. The first news, which should be recorded, does not appear in "History".*
Screenshot	返回 历史 清除 书签 收藏 历史 今天 手机新浪网 sina.cn/?wm=4007
Assessment	Passed or failed given by crowd worker: *Failed*

Our experiment is conducted with Baidu crowdsourced testing platform[2]. 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 tasks and are required to submit crowdsourced test reports[3]. Table 1 demonstrates the attributes of a typical crowdsourced report[4]. The platform can automatically record the crowd worker's 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, 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

[2]Baidu (baidu.com) is the largest Chinese search service provider. Its crowdsourcing test platform (test.baidu.com) is also the largest ones in China.

[3]We will simplify "crowdsourced test report" as "crowdsourced report" or "report", potentially avoiding the confusion with "test set" in machine learning techniques.

[4]Reports are written in Chinese in our projects. We translate them into English to facilitate understanding.

false positives, i.e., a test report marked as `failed` that actually involves correct behavior or behavior outside of the studied software system. This is due to the financial compensation mechanism, which usually favors failed reports.

Currently in this platform, testers need to manually inspect these failed test reports to judge whether they actually reveal fault – `true fault`. However, inspecting 1,000 reports manually could take almost half a working week for a tester. Besides, only less than 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.

2.2 Active Learning

Active learning is a subfield of machine learning. The key hypothesis is that, if the learning algorithm is allowed to choose the data from which it learns, it will perform better with less training data [6].

Consider that, for any supervised learning system to perform well, it must often be trained on hundreds (even thousands) of labeled instances. For most of the learning tasks, labeled instances are very difficult, time-consuming, or expensive to obtain. Active learning attempts to overcome the labeling bottleneck by selecting unlabeled instances and asking user their labels (simplified as '*select a query*', or '*query*' in the following paper). In this way, the active learner aims to achieve high accuracy using as few labeled instances as possible, thereby reducing the cost of obtaining labeled data [6].

All active learning techniques involve measuring the informativeness of unlabeled instances. Informativeness represents the ability of an instance in reducing the uncertainty of the classification. For example, the chosen instance is the one which the classifier is least certain how to label [7].

2.3 Local Bias

Local bias refers to the phenomenon that data are heterogeneous within dataset, and their distributions are often different among different parts of the dataset [8]. Crowdsourced reports naturally have local biases, where the contained technical terms may be shared by only a subset of reports rather than all the reports.

The main reason is as follows. Software often has several technical aspects, e.g., display, location, navigation, etc. In crowdsourced testing, each report usually focuses on specific technical aspects of the project, and describes the software behavior with specific technical terms. When classifying a particular report, reports which share more technical terms might be helpful to build classifiers, while reports which share less or none terms might contribute less to the classification.

We present an illustrative example to illustrate the local bias of crowdsourced reports and its influence. We first randomly select an experimental project (P1, a map app) and use K-Means [9] to group the reports into four clusters based on their technical terms (details are in Section 3.1). Each cluster contains a set of reports that share similar technical terms, which should be the testing outcomes for same technical aspect or similar aspects. We then generate the term cloud for each cluster in Figure 2. The size and color demonstrate the frequency of technical terms, with larger and darker ones denoting the terms of higher frequency.

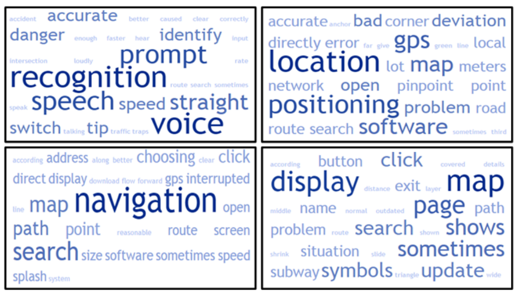


Figure 2: **Illustrative example of local bias**

We can easily observe that the technical terms exert great differences among clusters. The top left cluster might concern with the technical aspect of speech recognition, with terms like *recognition, speech, and voice*. The top right cluster contains such terms as *location, GPS, and deviation*, which might relate to the technical aspect of location. Taken in this sense, models, built on the reports from recognition cluster, may fail to classify the reports in location cluster. Note that, we have tried the number of clusters ranging from 2 to 10, all exposing the local bias problem. Here we only show the results with four clusters for better visualization.

Local bias has been widely investigated in effort estimation and defect prediction studies [10–14]. The most common technique to overcome the local bias is relevancy filtering, e.g., nearest-neighbor similarity [13, 14]. To be more specific, it allows the use of only certain train instances, which are closer to the test instances, for model construction. However, there are scarcely any studies focusing on dealing with the local bias in active learning methods. This paper borrows the idea of relevancy filtering and designs a novel approach to mitigate the local bias in active learning.

3. APPROACH

Figure 3 demonstrates the overview of our approach. We will first illustrate the feature extraction, followed with the details of our `LO`cal-based `A`ctive Classi`F`ication (LOAF).

3.1 Extracting Features

The goal of feature extraction is to obtain features from crowdsourced reports which can be used as input to train machine learning classifiers. We extract these features from the text descriptions of crowdsourced reports.

We first collect different sources of text description together (input and operation steps, result description). Then we conduct **word segmentation**, as the crowdsourced reports in our experiment are written in Chinese. We adopt ICTCLAS[5] for word segmentation, and segment descriptions into words. We then remove **stopwords** (i.e., "am", "on", "the", 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[6] is adopted.

Each of the remaining term corresponds to a feature. For each feature, we take the frequency it occurs in the description as its value. We use the TF (term frequency) instead of

[5]ICTCLAS (http://ictclas.nlpir.org/) is widely used Chinese NLP platform.

[6]LTP (http://www.ltp-cloud.com/) is considered as one of the best cloud-based Chinese NLP platforms.

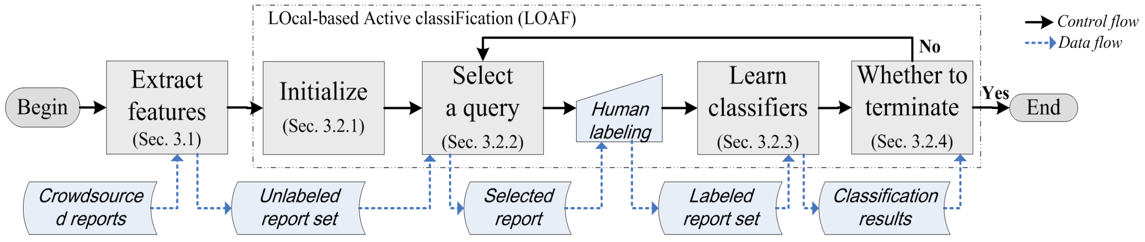


Figure 3: **Overview of our approach**

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. We organize these features into a **feature vector**.

3.2 Local-Based Active Classification (LOAF)

The primary focus of active learning techniques is measuring the informativeness of unlabeled data (details are in Section 2.2), in order to query the most informative instance to help build effective classifier. The design of existing techniques is based on such assumption that data are identically distributed [6]. However, crowdsourced reports usually have significant local bias (details are in Section 2.3), so existing techniques would generate poor and unstable performance on these data (details are in Section 5.2).

To address such problem, we propose the LOcal-based Active ClassiFication (LOAF). The main idea is to query the most informative instance within local neighborhood, then learn classifiers based on local neighborhood. Local neighborhood is defined as one or several nearby (with smallest distance) labeled instances. To cope with the local bias in the investigated dataset, the informativeness in our approach is measured within local neighborhood – denoted as '*local informativeness*'. In details, we first obtain the local neighborhood for each unlabeled instance, and measure its local informativeness. We then select the most local informative unlabeled instance and query its label.

Algorithm 1 and Figure 3 summarize the process of LOAF. LOAF first chooses the initial instance and ask user its label. After that, LOAF would iteratively select one instance, ask user its label, and learn classifiers from the labeled instances. LOAF then leverages the up-to-date labeled information to choose which instance to select next. The process will continue until satisfying one of the termination criteria. We will illustrate the process in more detail in the following subsections.

3.2.1 *Initializing*

This step is to choose the initial test report and ask user its label. Because of the existence of local bias, randomly choosing the initial instance in existing active learning techniques cannot work well [6]. Hence, our approach proposes a new initial selection method to choose the initial test report for labeling. This can mitigate the influence of initial instance and obtain stable good performance.

LOAF first computes the distance between each pair of report, then obtains the nearest distance for each report. Second, LOAF selects the report whose nearest distance is

Algorithm 1 Local-based active classification (LOAF)

```
Input:
    Unlabeled report set $U$; Labeled report set $L$ = null;
    Tag for termination $TM$ = true; Parameter $pr\_local$ and $pr\_stop$;
Output:
    Classification results $C$;
 1: //Initialize
 2: Choose the initial report $UR_i$ according to eq.(1) and query its label;
 3: $L = L \cup UR_i$, $U = U$ - $UR_i$;
 4: while (TM) do
 5:     //Select a query
 6:     Select unlabeled report $UR_i$ from $U$ according to eq.(2) and query its label;
 7:     $L = L \cup UR_i$, $U = U$ - $UR_i$;
 8:     //Learn classifiers
 9:     for each unlabeled report $UR_i$ in $U$ do
10:         Choose $pr\_local$ labeled report $LR_j$ according to eq.(3), learn classifier and obtain the classification result $C_i$ for report $UR_i$.
11:     end for
12:     //Whether to terminate
13:     Judge whether can terminate based on $pr\_stop$, if yes, set TM as false.
14: end while
```

the largest among all reports. This is to ensure the most sparse local region can also build effective classifiers (details are in Section 3.2.2 and 3.2.3).

In detail, the chosen initial instance is computed as follows:

$$\arg\max_{i \in U}\{\min_{j \in U, j \neq i}(distance(UF_i, UF_j))\} \tag{1}$$

where U represents the initial unlabeled dataset, UF_i and UF_j denote the feature vector of i_{th} and j_{th} test report in U, respectively. We apply cosine similarity between two feature vectors to measure their distance. This is because prior study showed that it performs better for high-dimensional text documents than other distance measures (e.g., euclidean distance, manhattan distance) [15].

3.2.2 *Selecting a Query*

This step aims at selecting the most informative test reports and asking user their labels. In each iteration, given a set of labeled reports and unlabeled reports, LOAF will select an unlabeled report based on its local informativeness. In detail, firstly, LOAF obtains the local neighborhood for each unlabeled report, and measures its local informativeness. This is done through computing the distance between the unlabeled report and every labeled report. The labeled report with nearest distance is treated as the local neighborhood for each unlabeled report, and the local informativeness is measured using the nearest distance. Secondly, LOAF selects the unlabeled report whose local informativeness is the largest, which can better serve the local-based classification in Section 3.2.3. This is done by selecting the unlabeled report whose nearest distance is the largest, among all unlabeled reports.

The rationale behind is as follows: to serve as the local-based classification in the next step, it should be guaranteed that each unlabeled instance has nearby reports which can be used to classify itself. Hence, if one instance's nearest neighbor has already been labeled, the local-based classifier would have the higher probability of correctly classifying it. On the other hand, if the nearest labeled instance turns out to be far away, this might imply big uncertainty in model building and classification. From this point of view, the farthest of the nearest labeled instance turns out to be the most informative one for local-based classification.

In detail, the selected instance is computed as follows:

$$\arg\max_{i\in U}\{\min_{j\in L}(distance(UF_i, LF_j)\} \tag{2}$$

where U and L represent the unlabeled dataset and labeled dataset in this iteration, respectively. UF_i and LF_j denote the feature vector of i_{th} unlabeled report and feature vector of j_{th} labeled report, respectively. Similarly, we apply cosine similarity to measure the distance of two feature vectors.

3.2.3 Learning Classifiers

This step aims at learning classifiers to classify unlabeled reports, using labeled reports. Since our investigated dataset has local bias, LOAF uses the local-based classification, i.e., utilizing the reports in one's local neighborhood to conduct the classification. In detail, for each unlabeled report, LOAF uses the *pr_local* most nearest labeled test reports to build classifier.

The *pr_local* labeled reports are chosen as follows:

$$\arg_{pr\_local}\{\min_{j\in L}(distance(UF_i, LF_j)\} \tag{3}$$

where UF_i denotes the feature vector of i_{th} unlabeled report which needs to be classified. LF_j represents the feature vector of j_{th} labeled report, and L represents the labeled dataset in this iteration. Note that, if the size of L is smaller than *pr_local*, then *pr_local* is set as the size of L. Similarly, we apply cosine similarity to calculate the distance of the two feature vectors.

3.2.4 Whether to Terminate

This step is to judge whether the labeling process could be terminated. Our approach considers two different scenarios of termination. The first one is that user can input the maximum effort (e.g., number of labeling instances) they can afford. When the effort is reached, the labeling process will be terminated. The second one is that the approach will decide whether to terminate according to the classification results. If the classification for each unlabeled report remains unchanged in the successive *pr_stop* iterations, LOAF would suggest termination. For both scenarios, the performance in the last iteration is treated as the final classification results.

Note that, as the classification performance in the first scenario is unguaranteed, this paper only evaluates the effectiveness of LOAF under the second scenario.

4. EXPERIMENT SETUP

4.1 Research Questions

We evaluate our approach through three dimensions: effectiveness, advantage, and usefulness. Specifically, our evaluation addresses the following research questions:

Table 2: **Projects under investigation**

	#	**#Fa**	**#TF**	**%TF**		**#**	**#Fa**	**#TF**	**%TF**
P1	321	267	183	68.5	**P2**	874	717	144	20.1
P3	302	168	125	74.4	**P4**	216	144	31	21.5
P5	492	455	280	61.5	**P6**	403	304	25	8.2
P7	688	647	253	39.1	**P8**	1094	727	447	61.4
P9	504	394	157	39.8	**P10**	320	163	61	37.4
P11	637	537	220	40.9	**P12**	436	232	165	71.1
P13	297	223	129	57.8	**P14**	217	157	46	29.3
P15	423	272	250	91.9	**P16**	556	398	251	63.1
P17	815	667	262	39.3	**P18**	307	180	122	67.8
P19	632	545	183	33.5	**P20**	580	323	112	34.7
P21	802	672	177	26.3	**P22**	466	402	102	25.3
P23	1414	1034	500	48.3	**P24**	1502	1152	583	50.1
P25	342	181	76	41.9	**P26**	824	503	131	26.0
P27	524	424	163	38.4	**P28**	391	317	253	35.2
P29	390	334	87	26.0	**P30**	495	417	83	19.9
P31	832	754	465	61.7	**P32**	806	522	199	38.1
P33	358	93	49	52.7	**P34**	452	284	58	20.4
Summary									
#			**#Fa**		**#TF**		**%TF**		
19,712			**14,609**		**6,372**		**43.4**		
Projects for case study									
C1	231	177	87	49.1	**C2**	690	455	182	40.0
C3	1428	1004	392	39.0					

- **RQ1 (Effectiveness)**: How effective is LOAF in classifying crowdsourced reports?

We first investigate the performance of LOAF in classifying crowdsourced report for each experimental project. Then we study the influence of parameter *pr_local* and *pr_stop* on model performance, as well as suggest the optimal *pr_local* and *pr_stop*. Finally, we explore the influence of initial instance on model performance and further demonstrate the effectiveness of our initial selection method.

- **RQ2 (Advantage)**: Can LOAF outperform existing techniques in classifying crowdsourced reports?

To demonstrate the advantages of our approach, we compare the performance of LOAF with both active learning techniques and supervised learning technique (details in Section 4.3).

- **RQ3 (Usefulness)**: Is LOAF useful for software testers?

We conduct a case study and a questionnaire survey in Baidu crowdsourced testing group to further evaluate the usefulness of the proposed LOAF.

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 34 projects. Table 2 provides more details with total number of crowdsourced reports submitted (#), number of `failed` reports (#Fa), number and ratio of reports assessed as `true fault` (#TF, %TF). Due to commercial consideration, we replace detailed project names with serial numbers.

Additionally, we randomly collect three other projects closed in Mar. 10th 2016 to conduct the case study in Section 5.3.

Note that, our classification is conducted on `failed` reports, not the complete set. We exclude the `passed` reports because of the following reason. As we mentioned, `failed` reports can usually involve both correct behaviors and true faults. However, through talking with testers in the company, we find that almost none of the `passed` reports involve true fault.

We use the assessment attribute (Table 1) of each report as the groundtruth label of classification. To verify the validity of these stored labels, we additionally conduct the random sampling and relabeling. In detail, we randomly select 10 projects, and sample 10% of crowdsourced reports from each selected project. A tester from the company is asked to relabel the data, without knowing the stored labels. We then compare the difference between the stored labels and the new labels. The percentage of different labels for each project is all below 4%.

4.3 Experimental Setup and Baselines

To demonstrate the advantages of LOAF, we first compare our approach to three **active learning techniques**, which are commonly-used or the state-of-the-art techniques:

Margin sampling [6]: Randomly choose the initial test report to label. Use all current labeled reports to build classifier, and query a report for which the classifier has the smallest difference in confidence for each classification type (true fault or not). Label it and repeat the process.

Least confidence [6]: Randomly choose the initial test report to label. Use all current labeled reports to build classifier, and query a report that the classifier is least confident about. Label it and repeat the process.

Informative and representative [16]: Randomly choose the initial test report to label. Use all current labeled reports to build classifier. Then query a report which can both reduce the uncertainty of classifier and represent the overall input patterns of unlabeled data. Label it and repeat the process. This is the state-of-the-art technique. We use the package QUIRE[7] for experiments.

To alleviate the influence of initial report on model performance, for each method, we conduct 50 experiments with randomly chosen initial reports.

As there might be some historical data which can be utilized for model building, we also compare our approach with **supervised learning technique**. Inspired by the cross-project prediction in defect prediction [17] and effort estimation [18], our experimental design is as follows: for each project under testing, we choose the most similar project from all available crowdsourced testing projects, then build a classifier based on the reports of that selected project. The similarity between two projects is measured using the method in [19], which is based on the marginal distribution of training set and test set.

Both our approach and these baselines all involve utilizing machine learning classification algorithm to build classifier. We have experimented with Support Vector Machine (SVM) [20], Decision Tree [20], Naive Bayes [21], and Logistic Regression [21]. Among them, SVM can achieve good and stable performance. Hence, we only present the results of SVM in this paper, due to space limit.

4.4 Evaluation Metrics

As the main consideration for active learning is cost-efficient, we utilize **accuracy** and **effort** to evaluate the classification performance.

The F-Measure of classifying true fault after termination, which is the harmonic mean of precision and recall [22], is used to measure the **accuracy**. The reason we do not use precision and recall separately is because most of the F-Measure in our experiments is 1.00, with 100% precision and 100% recall.

We record the percentage of labeled reports among the whole set of reports after termination, to measure the **effort** for the classification.

[7]http://lamda.nju.edu.cn/code_QUIRE.ashx

5. RESULTS

5.1 Answering RQ1 (Effectiveness)

RQ1.1: What is the performance of LOAF in classifying crowdsourced reports?

Table 3 demonstrates the accuracy and effort of LOAF for each experimental project (under the optimal *pr_local* (15) and *pr_stop* (10)), as well as the statistics. Results reveal that in 70% projects (24/34), LOAF can achieve the accuracy of 1.00, denoting 100% precision and 100% recall. In 85% projects (29/34), the accuracy is above 0.98. Furthermore, the minimum accuracy attained by LOAF is 0.95, with precision and recall both above 0.90.

For effort, LOAF merely requires to label 10% of all test reports on median for building effective classifier. Beside, for 88% (30/34) projects, LOAF only needs to label less than 20% of all reports, which can support classifying all the remaining test reports effectively.

The above analysis indicates that LOAF can facilitate the report classification with high accuracy and little effort.

RQ1.2: What is the impact of parameter *pr_local* on model performance and what is the optimal *pr_local* for LOAF?

We use *pr_local* to control the local neighborhood when conducting local-based classification (Section 3.2.3). To answer this question, we vary *pr_local* from 2 to 50 with *pr_stop* as 10, and compare the classification performance. We find that for all experimental projects, there are two patterns of performance trend under changing *pr_local*. Due to space limitation, we only present the performance trend of one random-chosen project for each pattern in Figure 4.

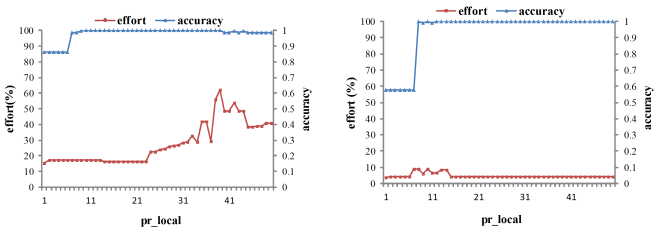
We can observe that *pr_local* indeed could influence the model performance, which indicates the need for finding the optimal *pr_local*. In the first pattern, the performance would remain best from *pr_local* is 15, thus the optimal *pr_local* ranges from 15 to 50. In the second pattern, only when *pr_local* is between 13 and 23, the accuracy is highest while the effort is lowest, thus the optimal *pr_local* is in a narrower range (from 13 to 23). The reason why performance keeps unchanged in the first pattern is that the actual labeled instances are fewer than *pr_local*, so that the actual *pr_local* used in such scenario is the number of labeled instances, not the demonstrated *pr_local*. Generally speaking, too small and too large *pr_local* can both result in low accuracy and high effort. This may be because small *pr_local* will result in the overfitting of classifiers, while large *pr_local* can easily bring noise to the model.

To determine the optimal *pr_local* in our approach, we first obtain the value of optimal *pr_local* for each experimental project. Then we count the occurrence of each value and consider the value with most frequent occurrence as optimal *pr_local*, which is 15, 16, and 17 in our context. We simply use 15 in the following experiments for saving computing cost.

Table 3: **Comparison of classification performance with different active learning techniques (RQ1.1 & RQ2.1)**

	LOAF		**Margin Samp.**		**Least Conf.**		**Infor. Repre.**	
	acc.	eff.	acc.	eff.	acc.	eff.	acc.	eff.
P1	1.00	13	0.74,0.94	28,55	0.74,0.97	27,42	0.75,0.97	34,45
P2	1.00	15	0.62,0.97	26,42	0.84,0.95	16,32	0.80,0.96	16,42
P3	1.00	52	0.85,0.95	29,56	0.87,0.95	25,58	0.87,0.95	28,40
P4	1.00	8	0.18,0.98	43,73	0.22,0.98	20,44	0.20,0.93	20,56
P5	1.00	13	0.30,0.99	38,88	0.43,0.95	32,44	0.40,0.94	17,54
P6	0.97	14	0.07,0.96	15,24	0.12,0.95	20,34	0.10,0.95	26,40
P7	1.00	6	0.32,0.96	28,78	0.42,0.99	32,54	0.31,0.99	33,65
P8	1.00	2	0.58,0.96	24,51	0.82,0.98	11,24	0.76,0.97	13,24
P9	1.00	3	0.61,0.96	34,54	0.84,0.98	38,49	0.80,0.95	37,49
P10	1.00	11	0.47,0.95	49,89	0.41,0.99	48,53	0.40,0.98	47,66
P11	1.00	2	0.92,0.97	41,81	0.92,0.98	32,53	0.86,0.99	56,80
P12	1.00	10	0.90,0.99	44,84	0.88,0.98	26,34	0.88,0.97	32,38
P13	1.00	14	0.92,0.99	45,95	0.92,0.98	27,45	0.92,0.98	32,80
P14	0.98	25	0.78,0.97	48,68	0.78,0.98	48,72	0.76,0.98	51,71
P15	0.96	22	0.84,0.95	26,38	0.86,0.99	22,35	0.84,0.95	25,36
P16	0.98	14	0.52,0.95	36,86	0.60,0.98	36,78	0.52,0.96	19,64
P17	1.00	4	0.38,0.98	38,68	0.36,0.99	35,52	0.31,0.96	17,23
P18	0.99	25	0.90,0.98	28,98	0.90,0.99	27,54	0.92,0.99	27,34
P19	1.00	2	0.47,0.97	34,84	0.64,0.97	26,74	0.40,0.96	26,43
P20	1.00	3	0.80,0.99	34,84	0.76,0.97	22,46	0.80,0.99	32,68
P21	0.99	12	0.04,0.98	13,44	0.43,0.97	17,37	0.26,0.97	13,32
P22	1.00	7	0.17,0.97	12,32	0.68,0.97	26,38	0.34,0.96	21,38
P23	1.00	1	0.86,0.98	56,86	0.90,0.94	43,63	0.88,0.97	47,62
P24	1.00	1	0.72,0.92	37,57	0.76,0.95	35,45	0.78,0.96	35,43
P25	1.00	11	0.90,0.93	36,90	0.91,1.00	36,52	0.90,0.98	47,62
P26	1.00	3	0.44,0.94	28,56	0.64,0.97	30,53	0.50,0.98	34,50
P27	1.00	3	0.30,0.97	31,81	0.37,0.99	21,46	0.22,0.99	24,59
P28	1.00	4	0.90,0.98	17,47	0.92,0.98	24,32	0.90,0.98	24,42
P29	0.95	20	0.59,0.98	28,80	0.56,0.98	23,47	0.44,0.98	24,42
P30	0.99	14	0.90,0.96	20,60	0.89,0.96	23,50	0.88,0.98	35,54
P31	1.00	2	0.82,0.99	30,40	0.80,0.98	25,42	0.82,0.98	24,38
P32	0.95	9	0.90,0.97	45,78	0.94,0.97	41,56	0.90,0.95	44,62
P33	1.00	15	0.96,1.00	62,98	0.96,0.98	37,64	0.96,0.99	46,64
P34	0.95	15	0.40,0.96	33,48	0.64,0.98	22,35	0.52,0.97	32,42
min	0.95	1	0.04,0.92	13,24	0.12,0.94	11,24	0.10,0.93	13,23
max	1.00	52	0.96,0.99	62,98	0.96,1.00	48,78	0.96,0.99	56,80
med.	1.00	10	0.67,0.97	33,70	0.77,0.98	26,46	0.77,0.97	30,47
avg.	0.99	11	0.61,0.96	33,67	0.69,0.97	28,48	0.64,0.96	30,50

Note: The two numbers in one cell represent minimum and maximum value of 50 random experiments.



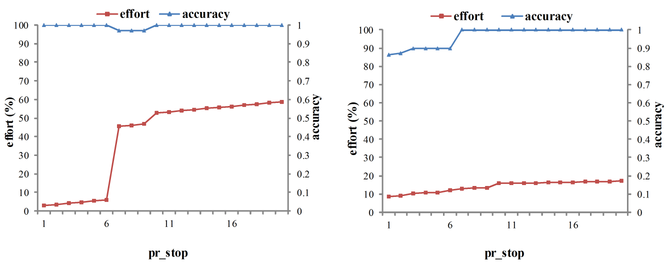
(a) Project 3 (b) Project 10

Figure 4: **Influence of pr_local on performance (RQ1.2)**

RQ1.3: What is the impact of parameter *pr_stop* on model performance and what is the optimal *pr_stop* for LOAF?

In our work, we use *pr_stop* to decide whether to terminate (Section 3.2.4). To answer this question, we use the optimal *pr_local* obtained earlier, which is 15, and vary *pr_stop* from 2 to 20 for experiments. Through examining the performance trend for all experimental projects, we find that there are also two patterns of performance trend under changing *pr_stop*. We only present the performance trend of one random-chosen project for each pattern in Figure 5, because of space limitation.

Similarly, we can observe that *pr_stop* indeed could influence the model performance, thus suggesting the optimal *pr_stop* would be helpful. It is easily understood that with



(a) Project 3 (b) Project 10

Figure 5: **Influence of pr_stop on performance (RQ1.3)**

the increase of *pr_stop*, the effort would increase correspondingly. From the first pattern, we can observe that accuracy might occasionally decrease with the increase of *pr_stop*, but can recover when *pr_stop* is 10. The second pattern shows that the accuracy can reach the highest and remain unchanged, from *pr_stop* is 7.

To determine the optimal *pr_stop*, we first obtain the value of *pr_stop* when the accuracy reach the highest and keep stable for each project (10 and 7 for the two demonstrated projects). We then consider the maximum among these values as optimal *pr_stop*, which is to ensure all the projects can reach the highest accuracy. In our experimental context, the optimal *pr_stop* is 10.

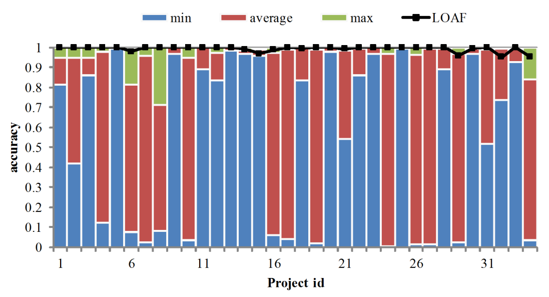
RQ1.4: What is the impact of initial instance on model performance and how effective of our initial selection method?

To answer this question, we experiment with random-chosen report acting as the initial instance, and repeat N times (N is set as half of project size). Figure 6 demonstrates the min, average and max value of model performance for random selection of initial report.

Nearly 40% (13/34) projects would undergo quite low accuracy (min value is less than 0.2) when randomly choosing initial report. For effort, nearly 60% (20/34) projects would involve quite high effort (max value is more than 30%) when choosing some random report for initial labeling. This reveals that random selection of initial instance can usually fall into low performance. The results further indicate the great necessity to design a method for the selection of initial instance to ensure a stable performance.

We then compare the performance of our initial selection method (details in Section 3.2.1) with the statistics of random selection. For accuracy, in 70% (24/34) projects, our method of initial selection can achieve the maximal value which the random selection can ever achieve. For other projects, the difference between our method and the maximal value of random selection is almost negligible, ranging from 0.008 to 0.04. These findings reveal that our initial selection method can achieve high and stable accuracy even compared with the best accuracy which random selection can ever reach.

For effort, in 97% (33/34) projects, our method of initial selection requires less effort than the maximal effort which random selection would require. In 55% (19/34) projects, our method requires less effort than the average effort required by random selection. In 86% (13/15) of remaining projects, the difference between the effort of our method



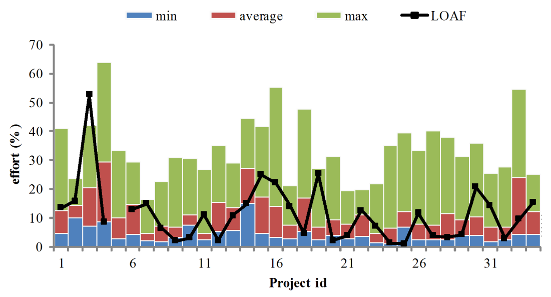


Figure 6: **Influence of initial instance on model performance (RQ1.4)**

and the average effort of random selection is smaller than 10%. These illustrations further reveal that our initial selection method is superior to random selection, considering its stable results, high accuracy, and little effort.

5.2 Answering RQ2 (Advantage)

RQ2.1: How does LOAF compare to existing *active learning techniques* in classifying crowdsourced reports?

Table 3 illustrates the performance of LOAF and three active learning baselines. As we mentioned in Section 4.3, we conduct 50 experiments for each active learning method because their random initialization can effect the final performance. We present the minimum and maximum performance of the random experiments for these methods. LOAF has well-designed initial selection step (Section 3.2.1), so its performance is unique. The value with dark background denotes the best accuracy or effort for each project.

At first glance, we can find that all the three baseline methods can occasionally fall into quite low accuracy and more effort, for most of the projects. On the contrary, as we have well-designed method to choose the initial report, LOAF can achieve high and stable performance. We also noticed that even the state-of-the-art method ('Infor. Repre.' method in Table 3) can not perform well for crowdsourced testing reports. This may stem from the fact that it does not consider the local nature of the dataset.

We then compare the performance of LOAF with the best performance which the baselines can ever achieve (highest accuracy or least effort). We can observe that all statistics (min, max, median and average) of accuracy for LOAF is higher than the best performance of baselines, denoting our approach can achieve a more accurate classification performance (the median value is 1.00). In addition, these statistics for effort of our approach are all less than the baselines (10% vs. 26% for median value).

We then shift our focus to the performance of each project. For more than 85% of all projects (29/34), our approach can attain both the highest accuracy and least effort, even compared with the best performance of baselines. Besides, for one of the remaining projects, our approach can achieve much higher accuracy, with 27% more effort at worst (P3), than the best baseline approach. For other four projects, our approach has slight decline in accuracy but with least effort. We also noticed that the decline is quite small with the maximum being 0.03 (0.95 vs. 0.98), and all the accuracy of our approach is above 0.95.

When it comes to the time and space cost of the mentioned techniques, it takes less than 1.0 seconds and less then 4.0MB memory for each step. Due to space limit, we would not present the details.

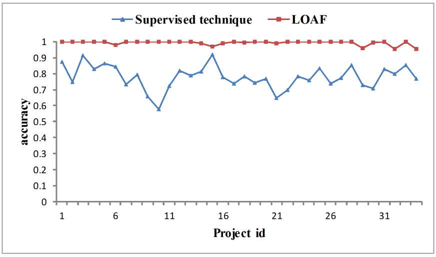


Figure 7: **Comparison of classification performance with supervised learning technique (RQ2.2)**

RQ2.2: How does LOAF compare to existing *supervised learning technique* in classifying crowdsourced reports?

Figure 7 illustrates the accuracy of LOAF and supervised prediction technique. We do not present the effort because supervised learning relies on historical data for classification, thus the effort is labeling all the training data.

We can easily observe for all the projects, the accuracy of LOAF is higher than the accuracy obtained by supervised learning. The median accuracy of supervised learning for all projects is 0.77, which is much smaller than that of LOAF (1.00). This indicates that LOAF can perform even better than supervised classification, which built on the large amount of historical labelled data. The reason might be originated from the fact that existing techniques cannot well deal with the local bias of crowdsourced testing data.

5.3 Answering RQ3 (Usefulness)

RQ3: Is LOAF useful for software testers?

To further assess the usefulness of LOAF, we use the implementation of our proposed LOAF to conduct a case study and a questionnaire survey in Baidu. We randomly select three projects for our case study (details are in Table 2). We collect these test reports as soon as it was closed, without any labeling information. Six testers from the crowdsourced testing group are involved. We divide them into two groups according to their experience, with details summarized in Table 4.

The goal of this case study is to evaluate the usefulness of LOAF in classifying the `true fault` from the crowdsourced test reports. Firstly, each practitioner in Group A is asked

Table 4: **Participant of case study (RQ3)**

Group A		Group B
A1	**3-5 years experience in testing**	B1
A2	**1-3 years experience in testing**	B2
A3	**<1 years experience in testing**	B3

to do the classification using LOAF. As a comparison, practitioners in Group B are asked to do it manually. To build the ground truth, we gather all the classification outcomes from the practitioners. Follow-up interviews are conducted to discuss the differences among them. Common consensus is reached on all the difference and a final edition of classification is used as the ground truth, details in Table 2.

Besides the accuracy and effort evaluation metrics in prior experiments, we also record the time taken for the classification. Table 5 presents the detailed results.

Table 5: **Results of case study (RQ3)**

	Results from Group A			**Results from Group B**		
Project	**C1**	**C2**	**C3**	**C1**	**C2**	**C3**
Accuracy	0.98,0.99	0.99,1.00	0.98,0.99	0.97,1.00	0.95,1.00	0.86,0.95
Effort(%)	15,15	4,4	15,15	100,100	100,100	100,100
Time(min)	15,18	12,18	64,87	72,90	200,240	370,410

Note: The two numbers in one cell represent minimum and maximum value from the three practitioners.

The classification performance of LOAF (Group A) is much better than the performance by manual (Group B), with higher accuracy, lower effort and less time. In particular, with the increase of project size, the consumed effort and time of manual classification can dramatically increase, while its accuracy would also drop obviously. Although testers can assign wrong labels to the queried reports when using LOAF, the final accuracy is less affected by these mistakes. This further denotes the effectiveness and usefulness of LOAF in real practice.

In addition, we design a questionnaire and conduct a survey to consult testers about the usefulness of LOAF. The questionnaire first demonstrates a short description about LOAF, a visualized classification process and a summarized evaluation result on 34 projects. Then it asks three questions shown in Table 6. We provide five options for the first two questions, and allow respondents freely express their opinion for the third question.

We send invitation emails to the testers who are involved in the report classification in Baidu crowdsourced testing group. We totally received 24 responses out of 37 requests.

As indicated in Table 6, of all 24 respondents, 19 of them (80%) agree that LOAF is useful for report classification and they would like to use it. This means testers agree the usefulness of LOAF in general. Only 2 (8%) hold conservation options and 3 (12%) disagreed. When it comes to the reason of disagreement, they mainly worry about its flexibility and performance on new project, as well as the recall of faults. This paves the direction for further research. In addition, the project manager shows great interest in LOAF, and is arranging to deploy LOAF on their platform to assist the classification process.

6. THREATS TO VALIDITY

The external threats concern the generality of this study. First, our experiment data consist of 34 projects collected from one of the Chinese largest crowdsourced testing platforms. 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 of projects 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 languages. 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 only utilize textual features to build classifiers, without including other features. Besides, we only use cosine similarity for measuring the distance between crowdsourced reports, without trying other measures. The experiment outcomes have proved the effectiveness of our method. Anyhow, we will try more features (e.g., the attributes of report) and other metrics for distance, to further investigate their influence on model performance.

Construct validity of this study mainly questions the data processing method. We rely on the assessment 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

Crowdsoucing 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 [23]. Chen and Kim [24] 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. [25] 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. [26] proposed an approach that employed a deterministic automata to help usability testing. Adams et al. [27] 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. [1] 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. Their method was evaluated only on 3 projects with students acting as crowd workers, while our evaluation is conducted on 34 projects with case study in one of the Chinese largest crowdsourced testing platform. Besides, our approach requires much less effort than theirs. Our previous work [28] proposed a cluster-based classification approach to effectively classify crowdsourced reports when facing with plenty of training data. However, training data is often not available. This work can reduce the effort for collecting training data while still being able to achieve good performance.

Table 6: **Results of survey (RQ3)**

Questions	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Total
Q1. Do you think LOAF is useful to classify "true fault" from crowdsourced test report?	0	1	0	3	20	24
Q2. Would you like to use LOAF to help with the report classification task?	0	3	2	8	11	24
If Disagree for either of the question, please give the reason.	Worry about its accuracy on other projects; Flexibility on new projects; Might miss crucial fault;					3

7.2 Automatic Classification in Software Engineering

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 [2] proposed an automated severity assessment method by text mining and machine learning techniques. Tian et al. [3] proposed DRONE, a multi factor analysis technique to classify the priority of bug reports. Wang et al. [4] proposed a technique combining natural language and execution information to detect duplicate failure reports. Zanetti et al. [29] 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. [5] proposed a hybrid approach by combining both text mining and data mining techniques of bug report data to automate the classification process. Wang et el. [30] proposed FixerCache, an unsupervised approach for bug triage by caching developers based on their activeness in components of products. Mao et al. [31] proposed content-based developer recommendation techniques for crowdsourcing tasks. Our 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 utilized supervised learning techniques to conduct the classification, which often requires users to manually label plenty of training data. Our proposed approach reduces the burden of manual labeling while still being able to achieve good performance.

There were researches to classify app reviews as bug reports, feature requests, etc. [32–34], which can help deal with the large amounts 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 need large number of labeled data to learn the supervised model, which is time-consuming. Moreover, the performances of our approach surpass theirs (F-measure is 0.72 to 0.95 in [34]) a lot.

Some other studies focus on differentiating between malicious behaviors and benign behaviors of app, based on data flow or context information [35, 36] . This motivates us to utilize new sources of information to conduct the report classification in future work.

7.3 Active Learning in Software Engineering

There is a wealth of active learning studies in machine learning literature, such as [16, 37]. Several studies have utilized active learning techniques for software engineering tasks. Bowring et al. [38] proposed an automatic approach for classifying program behavior by leveraging Markov model and active learning. Lucia et al. [39] proposed an approach to actively incorporate user feedback in ranking clone anomaly reports. Wang et al. [40] proposed a technique that can refine the results from a code search engine by actively incorporating incremental user feedback. Thung et al. [41] proposed an active semi-supervised defect prediction approach to classify defects into ODC defect type. Ma et al. [42] proposed an active approach for detecting malicious apps. These aforementioned studies merely utilized existing active learning techniques to solve the software engineering tasks, without considering the speciality of these tasks.

Kocaguneli et al. [43] proposed QUICK, which is an active learning method that assists in finding the essential content of software effort estimation data, so as to simplify the complex estimation methods. Nam et al. [44] proposed novel approaches to conduct defect prediction on unlabeled datasets in an automated manner. These two researches have something in common with our work – they are new designed active learning methods for specialized task. But their methods can not be directly used for the classification of crowdsourced reports and could not handle the local bias in dataset.

8. CONCLUSION

This paper proposes `LO`cal-based `A`ctive Classi`F`ication (LOAF) to address the two challenges in automating crowdsourced test reports classification, i.e., local bias problem and lacking of historical labeled data. We evaluate LOAF from the standpoints of effectiveness, advantage, and usefulness in one of the Chinese largest crowdsourced testing platforms, and results are promising.

Active learning techniques are promising when facing such situation that data are abundant but labels are scarce or expensive to obtain. This is becoming quite common in software engineering practice for the last decade. Hence, our approach can well motivate the various classification problems which have required plenty of labeled data, e.g., report classification [4, 5], app review classification [32, 33].

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 other features and 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, No.91218302, and No.61303163. We would like to thank the testers in Baidu for their great efforts in supporting this work.

10. REFERENCES

[1] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu, "Test report prioritization to assist crowdsourced

testing," in *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (FSE 2015)*, 2015, pp. 225–236.

[2] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of IEEE International Conference onSoftware Maintenance (ICSM 2008)*, 2008, pp. 346–355.

[3] Y. Tian, D. Lo, and C. Sun, "Drone: Predicting priority of reported bugs by multi-factor analysis," in *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM 2013)*, 2013, pp. 200–209.

[4] 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 *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*, 2008, pp. 461–470.

[5] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," in *Proceedings of the 30th IEEE International Conference on Software Maintenance (ICSM 2014)*, 2014, pp. 311–320.

[6] B. Settles, "Active learning literature survey," University of Wisconsin-Madison, Tech. Rep., 2010.

[7] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1994)*, 1994.

[8] A. Storkey, *Dataset shift in machine learning.* The MIT Press, Cambridge, MA, 2009.

[9] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[10] Y. Yang, Z. He, K. Mao, Q. Li, V. Nguyen, B. Boehm, and R. Valerdi, "Analyzing and handling local bias for calibrating parametric cost estimation models," *Information and Software Technology*, vol. 55, no. 8, pp. 1496 – 1511, 2013.

[11] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (FSE 2009)*, 2009, pp. 91–100.

[12] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 2011, pp. 343–351.

[13] 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*, vol. 39, no. 6, pp. 822–834, 2013.

[14] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013)*, Oct 2013, pp. 45–54.

[15] H. Finch, "Comparison of distance measures in cluster analysis with dichotomous data," *Journal of Data Science*, vol. 3, pp. 85–100, 2005.

[16] S.-J. Huang, R. Jin, and Z.-H. Zhou, "Active learning by querying informative and representative examples," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

[17] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, Oct. 2009.

[18] L. L. Minku and X. Yao, "How to make best use of cross-company data in software effort estimation?" in *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, 2014, pp. 446–456.

[19] S. Hido, T. Idé, H. Kashima, H. Kubo, and H. Matsuzawa, "Unsupervised change analysis using supervised learning," in *Proceedings of 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008)*, 2008, pp. 148–159.

[20] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[21] A. Berson, S. Smith, and K. Thearling, "An overview of data mining techniques," *Building Data Mining Application for CRM*, 2004.

[22] C. D. Manning, P. Raghavan, and H. SchÍźtze, *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[23] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: A case study of crowdsourcing software development," in *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, 2014, pp. 187–198.

[24] N. Chen and S. Kim, "Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, 2012, pp. 140–149.

[25] 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*, vol. 30, no. 4, pp. 38–45, 2013.

[26] 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*, vol. 5, no. 1, pp. 575–579, 2014.

[27] M. Gómez, R. Rouvoy, B. Adams, and L. Seinturier, "Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring," in *Proceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2016, pp. 88–99.

[28] J. Wang, Q. Cui, Q. Wang, and S. Wang, "Towards effectively test report classification to assist crowdsourced testing," in *Proceedings of ACM/IEEE*

International Symposium on Empirical Software Engineering and Measurement (ESEM 2016), 2016.

[29] 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 *Proceedings of the International Conference on Software Engineering (ICSE 2013)*, 2013, pp. 1032–1041.

[30] S. Wang, W. Zhang, and Q. Wang, "FixerCache: Unsupervised caching active developers for diverse bug triage," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2014)*, 2014, pp. 25:1–25:10.

[31] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Proceedings of IEEE Symposium on Service-Oriented System Engineering (SOSE 2015)*, 2015, pp. 347–356.

[32] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *Proceedings of the 23rd IEEE International Requirements Engineering Conference (RE 2015)*, 2015, pp. 116–125.

[33] E. Guzman, M. El-Halaby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, 2015.

[34] 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 *Proceedings of the 31st IEEE International Conference on Software Maintenance (ICSM 2015)*, 2015, pp. 281–290.

[35] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, vol. 1, May 2015, pp. 426–436.

[36] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, vol. 1, May 2015, pp. 303–313.

[37] S. Chakraborty, V. Balasubramanian, A. R. Sankar, S. Panchanathan, and J. Ye, "Batchrank: A novel batch mode active learning framework for hierarchical classification," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2015)*, 2015, pp. 99–108.

[38] J. F. Bowring, J. M. Rehg, and M. J. Harrold, "Active learning for automatic classification of software behavior," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*, 2004, pp. 195–205.

[39] Lucia, D. Lo, L. Jiang, and A. Budi, "Active refinement of clone anomaly reports," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, June 2012, pp. 397–407.

[40] S. Wang, D. Lo, and L. Jiang, "Active code search: Incorporating user feedback to improve code search relevance," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE 2014)*, 2014, pp. 677–682.

[41] F. Thung, X.-B. D. Le, and D. Lo, "Active semi-supervised defect categorization," in *Proceedings of the 23rd International Conference on Program Comprehension (ICPC 2015)*, 2015, pp. 60–70.

[42] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," in *Proceedings of the 39th Annual Computer Software and Applications Conference (COMPSAC 2015)*, vol. 2, July 2015, pp. 179–184.

[43] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, and R. Madachy, "Active learning and effort estimation: Finding the essential content of software effort estimation data," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1040–1053, Aug 2013.

[44] J. Nam and S. Kim, "CLAMI: Defect prediction on unlabeled datasets," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, Nov 2015, pp. 452–463.