Research on Defect Priority Classification of Crowdsourcing Testing for Mobile Applications

Ying Liu₁,*, Chunyan Ma₁, Zhanwei Dong₁, Tao Zhang₁, Jing Cheng² and Jie Zhang₁

₁School of software, Northwestern Polytechnical University, Xi’an, China
²School of computer and engineering, XI’AN Technology University, Xi’an, China

*Email address: 894749065@qq.com

Abstract. Crowdsourcing testing technology has developed in recent years with the development of software testing, which can speed up releasing cycle and improve the quality of testing. It is of great practical value to study the priority classification and cause analysis of defect reports by using the potential information of crowdsourcing test defect reports. This paper combines the research of mobile application crowdsourcing test defect report with machine learning data analysis technology, studies the priority classification of mobile application crowdsourcing test defect report, and then carries out defect cause analysis on the basis of defect priority classification. Defect classification is an intuitive reflection of defect research. This paper takes defect priority classification as the breakthrough point of defect report research, uses σ−AdaBoostSVM classification algorithm to classify defect reports, and then carries out cause analysis after defect report classification, which is conducive to the faster location and repair of defects. The experimental verification results demonstrate the effectiveness of the proposed method.

1. Introduction

Mobile application crowdsourcing testing technology has developed rapidly in recent years, and crowdsourcing testing is a new field in the development of software testing [1-2]. It takes advantage of crowdsourcing and cloud platform to improve the effectiveness and efficiency of software testing. Mobile application software is tested on various practical platforms to make it more reliable, cost-effective and faster. However, due to the characteristics of crowdsourcing, it is more difficult to analyze defect reports in crowdsourcing for mobile applications. Various uncertainties and irregularities make it difficult to classify defect report data and analyze the causes.

Because of the wide application of crowdsourcing, it has received extensive attention in the academic circle in recent years [3]. Many experts and scholars have made a summary of the relevant research work on crowdsourcing in top conferences and periodicals. Catherine studied the method, data selection and data credibility of crowdsourcing [4]. Ke Mao studied the applications of the crowdsourcing in the life cycle of software engineering [5]. Doan categorized the crowdsourced systems according to the type of problem and the way of cooperation [6]. Zhao Y summarized the research status of crowdsourced technology and predict the future research area [7]. Additionally, there is a small amount of content related to mobile application testing. Jerry Gao [8] and Kirubakaran [9] gave the basic concepts of mobile application testing, and analyzed the needs, challenges and main problems of mobile application testing.
In the process of crowdsourcing testing of mobile applications, there will be a large number of defect reports, and the research on these defect reports is particularly important. For this reason, this paper proposes a relatively perfect pre-processing scheme for mobile application crowdsourcing test defect report. Through the analysis of key attributes in defect report, defect keywords are extracted, and the weight of defect keywords is calculated and the final defect normalization is processed. After this series of defect data pre-processing, the formal classification of defects is paved. This paper mainly studies the preprocessing technology of defect data in crowdsourcing test for mobile applications, and extracts feature sets of defect keywords by analyzing and processing defect data. In this chapter, defect reports are classified according to the feature set extracted before. The defect priority classification algorithm based on $\sigma$-AdaBoostSVM is used to classify defect reports. Finally, the expected results of defect classification are evaluated.

2. Priority Classification of Defects in Crowdsourcing Testing for Mobile Applications

2.1. Adaboost classifier
AdaBoost algorithm is a process of iteratively updating the weights of feature words, in which the weights of each feature word represent the probability that the feature words are classified. In each iteration process, the weights assigned by the classified feature words will gradually increase. In the next iteration, AdaBoost algorithm will pay more attention to the feature words whose weights increased in the previous iteration. If a feature word is iterated many times, the weight of the feature words will become larger and larger. Such feature words are called highly sensitive feature words. The new data set with modified weights is transmitted to the next round of classifiers for training. Finally, the classifiers obtained from each training are fused to get the final classifier decision maker. It can be seen that AdaBoost classifier can remove unnecessary training feature words and focus on key training feature words. The flow chart of AdaBoost classification algorithm is as follows:

(1) Initialize all training defect reports with a weight of $1/N$, where $N$ is the number of defect reports;

(2) from $m=1, \ldots , M$:

a. Train weak classifier $y_m$, to minimum weight error function.

$$E_m = \sum_{n=1}^{N} \alpha_n^{(m)} I(y_m(X_n)) \neq t_n$$  (1)

b. Then calculate the weight of the classifier.

$$\alpha_m = \ln\left\{\frac{1-E_m}{E_m}\right\}$$  (2)

c. Update the weight.

$$w_{m+1,j} = \frac{w_{m,j} \exp(-\alpha t_i y_m(x_j))}{Z_m}$$  (3)

where, $Z_m = \sum_{i=1}^{N} w_{m,j} \exp(-\alpha t_i y_m(x_j))$ is the normalization factor, which makes the sum of all $w$ is 1.

d. Obtain the final strong classifier.

$$Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$$  (4)

2.2. Defect Priority Classification Algorithm
The defect priority classification algorithm based on $\sigma$-AdaBoost SVM needs to consider the multi-classification problem and large sample problem when classifying the defect reports of crowdsourcing test for pre-processed mobile applications. Therefore, the traditional AdaBoost classification method can't fully meet the classification needs of defect reports. Secondly, the difference of AdaBoost
classification algorithm and classification accuracy are difficult to balance, so the classification effect of this method is not very ideal. In summary, this paper proposes an improved $\sigma - \text{AdaBoostSVM}$ defect priority classification algorithm to classify the defect reports processed in crowdsourcing testing.

In essence, the defect priority classification algorithm based on $\sigma - \text{AdaBoostSVM}$ trains SVM as AdaBoost weak classifier, and adjusts the kernel function parameters of SVM constantly during the training process, so that the weak classifier achieves a balance between classification accuracy and difference. This method mainly takes advantage of the advantages of AdaBoost algorithm, that is, adjusting the weight of defect training set once every training round and adjusting the parameters of the weak classifier’s kernel function, so that the classification accuracy and difference of the weak classifier can reach a certain balance, and ultimately improve the generalization performance of the final classifier. In this way, the next classifier trained will focus on the samples of these anomaly classifications.

Firstly, the noise around the positive class (defect report with defect keywords) is removed by using sampling rules, and reliable points are retained. At the same time, the reliable negative class (defect report without defect keywords) samples are extracted to be equivalent to the positive class samples. The central vector of the positive class is calculated, and the Euclidean distance from each sample of the negative class to the positive class set is calculated, and the negative class sample with the largest distance is selected. The steps of the improved $\sigma - \text{AdaBoostSVM}$ defect priority classification algorithm are as follows:

**Step one:** Input a set of training samples for defect reporting $\{(x_i, y_i), \ldots, (x_n, y_n)\}$, initial kernel parameters $\sigma_{\text{ini}}$, minimum kernel parameters $\sigma_{\text{min}}$, length of steps $\sigma_{\text{step}}$, punishment factor $M=8$.

**Step two:** Data Processing: Divide the training sample of defect report into several local training data sets of defect report $\{\text{train}1, \ldots, \text{train }N\}$.

**Step three:** Initialization of training sample weights for defect reporting

$$\omega = \{\omega_i = \frac{1}{C_n} | j = 1, 2, \ldots, N\} \quad (5)$$

Where $C_n$ represents the number of defect report samples for each category, $\omega_i$ represents the weight of this class of samples.

**Step four:** Do while ($\sigma > \sigma_{\text{min}}$)

1. A weak classifier $h_t$ is trained with local defect report training data set.
2. Calculate the error rate of $h_t$:

$$E_t = \sum_{i=1}^{N} \omega_i^t, y_i \neq h_t(x_i) \quad (6)$$

Where $\epsilon_t$ is the error rate of $h_t$, $\omega_i^t$ is the error rate of defect keywords for each defect weak classifier from i to t.

3. Calculate the difference of $h_t$:

$$D_t = \sum_{i=1}^{N} d_i(x_i) \quad (7)$$

Where $d_i(x_i)$ is the difference of Defect Keyword I in Defect Reporting Training Set t, $D_t$ represents the sum of the differences of all defect keywords from 1 to N.

4. If $\epsilon_t > 0.5$, $\sigma$ reduce one step $\sigma_{\text{step}}$, and return to (1).

5. Set the weight of weak classifier $h_t$.

$$\alpha_t = \ln \left( \frac{1 - E_t}{E_t} \right) \quad (8)$$

Where $\frac{1 - E_t}{E_t}$ is the iterative calculation of error rate of defective keywords.

6. Update the weight of training samples.
\[ o^{i+1}_t = \frac{o^i_t \exp(-\alpha_i y_i) h_t(x_i)}{C_t} \]  

(9)

Where \( C_t \) is a normalized constant, \( C_t = \sum_{i=1}^{N_t} o^{i+1}_t = 1 \). \( h_t(x_i) \) is the weak classifier of defects obtained by defect keyword I in defect report training set t.

**Step five:** Integration. Sort all weak classifiers according to their accuracy, select the first N models, and use the voting method to get the strong classifier model.

### 2.3. Evaluate criteria for classification results

In this paper, \( \sigma - AdaBoostSVM \) defect priority classification algorithm is used to classify mobile application crowdsourcing test defect reports. After classification, how the results need to be evaluated? In this chapter, machine learning evaluation method is used to evaluate the results. The performance of defect classifier is evaluated by Precision, Recall, Average Accuracy and Harmonized Average. The accuracy, recall, Average Accuracy and Harmonized Average measure the accuracy and completeness of the algorithm.

Firstly, defect priority is classified according to the \( \sigma - AdaBoostSVM \) defect priority classification algorithm, and defect reports are classified into positive or negative categories. Positive category refers to defect reports that contain defect keywords, while negative category refers to defect reports that do not contain defect keywords. But in the actual classification, there will be four sample situations:

1. **True Positive (TP):** Positive samples predicted by the model as positive classes. That is, defect keywords appear in defect reports containing defect keywords.
2. **False Positive (FP):** Negative samples predicted by the model as positive. That is, defect keywords do not appear in defect reports containing defect keywords.
3. **False Negative (FN):** Positive samples predicted by the model as negative classes. That is, defect keywords appear in defect reports that do not contain defect keywords.
4. **True Negative (TN):** Negative samples predicted by the model as negative classes. That is, defect keywords do not appear in defect reports that do not contain defect keywords.

The contingency table is shown in Table 1, where 1 represents the positive class and 0 represents the negative class.

| Prediction | 1 | 0 | Total |
|------------|---|---|-------|
| Fact       |   |   |       |
| 1          | TP| FN| TP+FN |
| 0          | FP| TN| FP+TN |
| Tota       | TP+FP| FN+TN| TP+FN+FP+TN |

Average accuracy measures the accuracy and completeness of the results generated by the algorithm.

\[ AVGP = \frac{\text{Num}(\text{instances of correct markup})}{\text{Num}(\text{training cases})} \]  

(10)

### 3. Experimental verification

#### 3.1. Experimental process

This experiment uses k-fold cross validation algorithm to verify the accuracy of classification algorithm. The data set is divided into 10 parts on average. Nine of them are used as training data in turn, and the other one is used as test data to calculate the accuracy or recall rate of each round. Repeat 10 times, then calculate the average of 10 times as the final evaluation results.
We randomly selected 82% data from 2428 mobile application crowdsourcing test defect reports obtained from a mobile crowdsourcing test platform according to the proportion of defect grade P1-P5 as training sample set, and the remaining 18% defect reports as test set, as shown in table 2.

Table 2. Statistical information of training samples and experimental data for defect reporting.

| Defect description | Defect level | Number of Defect | Number of Data set | Number of experimental data |
|--------------------|--------------|------------------|--------------------|---------------------------|
| Deadly             | P5           | 102              | 84                 | 18                        |
| Serious            | P4           | 215              | 177                | 38                        |
| flaw               | P3           | 1687             | 1390               | 297                       |
| Suggest            | P2           | 361              | 297                | 64                        |
|                     | P1           | 63               | 52                 | 11                        |

3.2. Experimental result analyses

In this paper, the average accuracy is used to evaluate the performance of the classifier. In order to evaluate the impact of the size of training data set on classifier, 20, 40, 60, ..., 200 Bug reports were used as training samples for the nine data used in the experiment. Table 3 and Figure 1 show the average accuracy under different training sample sizes.

Table 3. Average Accuracy Corresponding to Different Sample Numbers.

| Sample size | 20 | 40 | 60 | 80 | 100 | 120 | 140 | 160 | 180 | 200 |
|-------------|----|----|----|----|-----|-----|-----|-----|-----|-----|
| Train set1  | 0.543 | 0.592 | 0.604 | 0.683 | 0.713 | 0.769 | 0.722 | 0.872 | 0.796 | 0.783 |
| Train set2  | 0.564 | 0.634 | 0.638 | 0.716 | 0.655 | 0.769 | 0.798 | 0.825 | 0.769 | 0.792 |
| Train set3  | 0.605 | 0.665 | 0.737 | 0.759 | 0.779 | 0.781 | 0.806 | 0.788 | 0.775 | 0.796 |
| Train set4  | 0.483 | 0.512 | 0.687 | 0.803 | 0.646 | 0.636 | 0.743 | 0.799 | 0.812 | 0.806 |
| Train set5  | 0.557 | 0.578 | 0.635 | 0.714 | 0.747 | 0.741 | 0.791 | 0.843 | 0.792 | 0.808 |
| Train set6  | 0.619 | 0.685 | 0.723 | 0.755 | 0.785 | 0.768 | 0.835 | 0.819 | 0.787 | 0.799 |
| Train set7  | 0.587 | 0.601 | 0.643 | 0.701 | 0.762 | 0.737 | 0.805 | 0.847 | 0.801 | 0.797 |
| Train set8  | 0.499 | 0.534 | 0.548 | 0.587 | 0.647 | 0.719 | 0.799 | 0.822 | 0.768 | 0.784 |
| Train set9  | 0.536 | 0.573 | 0.702 | 0.755 | 0.747 | 0.721 | 0.802 | 0.799 | 0.803 | 0.774 |

Figure 1. Average Accuracy of Different Training Samples

From the experimental results, we can draw the following conclusions:
1) For all defective keyword categories, the average accuracy of Bug priority increases with the increase of the number of training samples, especially in the range of 20 to 160 samples.

2) Among all kinds of defective keywords, the average accuracy is the lowest when the number of training samples is 20. When the number of training samples reaches 160, the average accuracy reaches the required height, and when the number of training samples is 160, the average accuracy is the highest.

3) Among all kinds of defective keywords, when the number of training samples is 20, the average accuracy is the lowest, indicating that the training samples are 20, which cannot meet the training requirements. When the number of training samples is 160, the average accuracy is the highest, which has met the requirements of training defect priority classifier. It can be seen from the figure that when the number of training samples increases again, the average accuracy rate has stabilized, indicating that there is no need for a larger number of training samples.

4. Conclusions
Based on the analysis of AdaBoost classification algorithm, this paper obtains a further optimized classification algorithm $\sigma$ – AdaBoostSVM for mobile application crowdsourcing test defect report. This algorithm mainly aims at the problem of large amount of data and multi-classification, and plays a key role in the classification of mobile application crowdsourcing test defect report. Then, AdaBoostSVM defect classification algorithm is used to classify the defect reports of crowdsourcing testing. The results of classification are further normalized. The classification results are processed by using the data normalization methods of linear function normalization and 0-means normalization. Finally, the average accuracy and harmonic classification results are used to analyze and evaluate. The use of sections to divide the text of the paper is optional and left as a decision for the author. Where the author wishes to divide the paper into sections the formatting shown in table 2 should be used.

Acknowledgments
This work was supported by Graduate Creative Innovation Seed Fund of Northwestern Polytechnical University.

References
[1] Muccini H, Di Francesco A and Esposito P 2012 Software testing of mobile applications: Challenges and future research directions (Proceedings of the 7th International Workshop on Automation of Software Test) pp 29-35
[2] Latoza T and Hoek A 2016 Crowdsourcing in Software Engineering: Models, Motivations, and Challenges (IEEE Software vol 33) chapter 1 pp 74-80
[3] Yuxiang Zhao and Qinghua Zhu 2014 Evaluation on crowdsourcing research: Current status and future direction (Information Systems Frontiers vol 16) chapter 3 pp 417-434
[4] Catherine C, Marshall Microsoft Research, Silicon Valley, Mountain View and CA Frank M 2013 Shipman.Experiences surveying the crowd: reflections on methods, participation, and reliability (Proceeding WebSci '13 Proceedings of the 5th Annual ACM Web Science Conference) pp 234-243
[5] Ke Mao, Licia Capra, Mark Harman and Yue Jia 2016 A survey of the use of crowdsourcing in software engineering (The journal of systems and software vol 126) pp 57-84
[6] Doan A, Ramakrishnan R and Halevy A Y 2011 Crowdsourcing systems on the word-wide web (Communications of the ACM vol 54) chapter 4 pp 86-96
[7] Zhao Y and Zhu Q 2014 Evaluation on crowdsourcing research: Current status and future direction (Information Systems Frontiers vol 16) chapter 3 pp 417-434
[8] Jerry Gao, X. Bai, W. T. Tsai, and T 2014 Uehara Mobile application testing: a tutorial (IEEE Computer vol 47) chapter 2 pp 26-35
[9] Kirubakaran, B and Karthikeyani, V 2013 Mobile application testing — Challenges and solution approach through automation (Proceedings of the 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME)) pp 79-84