



Classifying crowdsourced mobile test reports with image features: An empirical study[☆]

Yuying Li, Yang Feng^{*}, Rui Hao, Di Liu, Chunrong Fang^{*}, Zhenyu Chen, Baowen Xu

The State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

ARTICLE INFO

Article history:

Received 30 October 2020
Received in revised form 3 October 2021
Accepted 13 October 2021
Available online 21 October 2021

Keywords:

Crowdsourced testing
Test report classification
Image features

ABSTRACT

Crowdsourced testing has become a popular mobile application testing method, and it is capable of simulating real usage scenarios and detecting various bugs with a large workforce. However, inspecting and classifying the overwhelming number of crowdsourced test reports has become a time-consuming yet inevitable task. To alleviate such tasks, in the past decades, software engineering researchers have proposed many automatic test report classification techniques. However, these techniques may become less effective for crowdsourced mobile application testing, where test reports often consist of insufficient text descriptions and rich screenshots and are fundamentally different from those of traditional desktop software. To bridge the gap, we firstly fuse features extracted from text descriptions and screenshots to classify crowdsourced test reports. Then, we empirically investigate the effectiveness of our feature fusion approach under six classification algorithms, namely Naive Bayes (NB), k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF) and Convolutional Neural Network (CNN). The experimental results on six widely used applications show that (1) SVM with fused features can outperform others in classifying crowdsourced test reports, and (2) image features can improve the test report classification performance.

© 2021 Elsevier Inc. All rights reserved.

1. Introduction

Crowdsourced testing has become a popular mobile application test method because it can employ a large workforce at a low cost. Crowdsourced testing can help detect various types of bugs in real usage scenarios, especially for dealing with notorious Android fragmentation issues by providing diverse information for bug fixing, including text descriptions, screenshots and other environment settings, such as brands and models of mobile devices, versions of Android system, locations, network environments, user settings, and operating systems. In general, testers use a crowdsourced testing platform to prepare testing requirements and testing tasks. Then, crowd workers can bid and conduct the tasks, and submit test reports. Finally, testers inspect test reports and assign them to developers for bug fixing.

To support crowdsourced testing, platforms have been developed for preparing tasks and submitting or reporting bugs, which draw the attention of large crowds. Crowdsourced testing has been widely adopted in many commercial testing platforms (such

as Testin,¹ uTest,² Baidu crowd test,³ Alibaba Crowd Test,⁴ and Mooctest⁵) and attracted a large number of crowd workers. Thus, for each task, hundreds of test reports are usually submitted. It is a tedious and time-consuming job to inspect and classify all the reports manually. Each newly reported bug must be triaged to determine whether it describes a significant and new problem or enhancement. If so, it must be assigned to an appropriate developer to fix it. Most triaging tasks, including bug assignment, heavily rely on manual efforts, which is labor-intensive and potentially errors prone (Jeong et al., 2009). In practice, due to the frequent changes in software development teams, it is difficult to identify an appropriate developer who has some experience in fixing similar bugs using a manual classification process (Anvik et al., 2006). Unlike traditional testing, the workers in crowdsourced testing may not be professionals. As the number of reports increases, it is too expensive to employ professionals to inspect and classify test reports. Automated classification methods should be more feasible than expensive human resources. For the Eclipse project, Anvik (2006) reports that an average

[☆] Editor: Gabriele Bavota.

^{*} Corresponding author.

E-mail addresses: fengyang@nju.edu.cn (Y. Feng),
fangchunrong@nju.edu.cn (C. Fang).

¹ <http://www.testin.io>

² <https://www.utest.com>

³ <http://test.baidu.com>

⁴ <https://mqc.aliyun.com/crowdtest>

⁵ <http://www.mooctest.com/>

of 37 bugs per day are submitted to the bug tracking system and 3 person-hours per day are required for the manual triage, and Jeong et al. (2009) report that 44% of bugs have been assigned to the wrong developers in their first assignment. To solve these problems, some machine learning algorithms are employed to conduct automatic bug classification (e.g. Jeong et al., 2009; Anvik et al., 2006; Anvik, 2006; Matter et al., 2009; Xuan et al., 2010).

Traditional test reports often contain text descriptions of the actions of the user and the behavior of the software, and thus most of the past classification researches are based on textual information (e.g. Anvik et al., 2006; Murphy and Cubranic, 2004; Xia et al., 2017; Alenezi et al., 2013; Zhou et al., 2016). Aiming to reduce the human labor costs, some research has proposed text classification approaches to automatic bug classification, including Naive Bayes (Murphy and Cubranic, 2004), k -nearest neighbors and support vector machine (Anvik et al., 2006). In contrast with the test reports of traditional desktop software, testers of mobile applications often prefer to submit screenshots rather than detailed paragraphs of text to describe bugs (Feng et al., 2016; Zhang et al., 2017) (due to factors such as the relative ease to take screenshots and the relative difficulty in long-form text editing on mobile devices). Additionally, screenshots can provide additional information to help reproduce bugs. Due to these factors for mobile application testing (e.g., the relative paucity of text descriptions and a heavy reliance on screenshots), traditional text-description-based test report classification techniques may not be ideal for this new domain of crowdsourced mobile application testing.

In this paper, we fuse textual and image information to classify mobile application test reports. For one test report, we separately extract features from the text description and screenshots; fuse the text features with image features; classify test reports with the fused features based on different classification algorithms. To validate the effectiveness of the proposed approaches, we conduct experiments on six applications with more than 2500 reports from the industry. The experimental results show that image features play a supporting role in the test report classification, with the *accuracy* and *F-measure* increased by about 10%. Secondly, we compare the performance of different automatic test report classification algorithms, and the results show that SVM incorporating fused features achieves the best performance in terms of *accuracy* and *F-measure*. Finally, we investigated the performance effects from different training set sizes and found that SVM outperforms others for a reduced training set. We have uploaded our source code by the URL:⁶

In this study, we make the following contributions.

- To the best of our knowledge, this is the first work to classify crowdsourced test reports based on image features, which can convey complementary information for textual information.
- We fuse text features and image features in each test report and then use different classical algorithms to classify test reports separately.
- We conduct a comprehensive study of over 3000 reports from 6 real industrial mobile applications in crowdsourced testing.

The remainder of this paper is organized as follows. Section 2 describes the background. Section 3 presents the preliminaries of classification techniques and performance measures. Section 4 presents our proposed approach. Section 5 presents the empirical study. Section 6 provides related work. Section 7 presents some discussions, and Section 8 provides the conclusions.

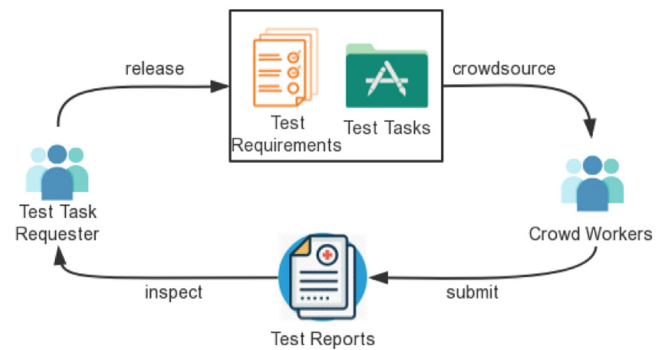


Fig. 1. The procedure of crowdsourced testing.

2. Background

2.1. Crowdsourced testing

The notion of crowdsourcing was proposed by Howe and Jeff in 2006 (Howe, 2006). Crowdsourcing is a problem-solving model, which can accomplish complex tasks by organizing individuals from different places, usually involving both Human and machine computing power (Mao et al., 2016). Crowdsourced testing becomes a widespread method with low cost, high coverage on hardware and software. Fig. 1 shows the procedure of crowdsourced testing. In crowdsourced testing, companies or organizations prepare software under testing requirements and testing tasks. Workers bid and perform testing tasks on their own devices, and edit test reports for the observed abnormal behaviors (Feng et al., 2015; Wang et al., 2016b). To attract more workers, testing tasks are often financially compensated, especially when real bugs are reported. In this context, workers prefer to submit thousands of test reports with short descriptions and rich screenshots. Before delivered to testers, test reports should be classified into different categories, such as incomplete function, page layout, abnormal exit, installation failure, registration failure, update failure, user experience, security, performance, and others. However, inspecting these test reports is time-consuming, tedious, and low-efficient. Hence, we are inspired to automate the classification of crowdsourced test reports efficiently.


2.2. Mobile crowdsourced test reports

Test reports in crowdsourced testing are written in natural language together with some screenshots based on the predefined format. A typical test report is usually composed of different fields, such as device information, crowd worker, test task, description, and screenshots (Feng et al., 2016; Wang et al., 2016a). In our study, we perform crowdsourced testing tasks for six mobile applications on the Moocostest crowdsourced testing platform.

Table 1 shows an example of a crowdsourced test report, which arrays an example of crowdsourced test reports from the real industrial data. Note that, in our study, all test reports are written in Chinese. To facilitate understanding, we translate them into English. The first row is the environment, including the basic configurations of used mobile devices, such as device brand, model, and operating system. The second row introduces the information of the crowd worker submitting the test report. The third row describes the testing task, including ID and name. The last row is the description. It contains detailed descriptions in natural language, and describes bugs and occasionally involves real user experience. Developers can understand the bug scenario via this description and make an initial decision for fixing

⁶ <https://bitbucket.org/YannicLi/2020-classifyingcrowdsourcedtestreports>

Table 1
An example of crowdsourced test report.

Attribute	Description
Environment Device os: Android 6.0.1	Device model: Xiaomi MI 5
Crowd worker	ID: 12567
Testing task Name: SLife usage test report	ID: 137
Description	Share sport data to friends or circle of friends by WeChat or micro-blogs, the system shows "Sharing failed, please try again..."
Screenshot	

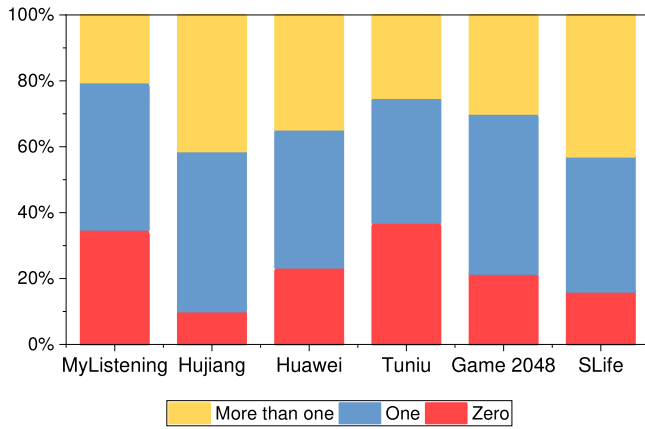


Fig. 2. The number of images in the test report.

the bugs according to such descriptions. The final row, namely the screenshot, provides some necessary images to capture the system symptoms when the bug occurs.

Comparing with traditional test reports, crowd workers would like to describe bugs with direct screenshots and short descriptions rather than complex text descriptions in mobile application testing. It is much more convenient to take screenshots than type longer descriptions on mobile virtual keyboards (Zhang et al., 2017). Meanwhile, such screenshots can be very vivid for developers to understand the bug scenarios and reproduce bugs. Therefore, we are motivated to propose an approach to classify the bug via fusing these features conveying in short text descriptions and rich screenshots in mobile crowdsourced test reports.

In crowdsourced testing platforms, numerous crowd workers submit screenshots to assist bug understanding. Fig. 2 shows the number of screenshots in the test report. 77.1% test reports provide screenshots, and 34.7% provide more than one screenshot. Therefore, test report classification based on fused information (i.e., screenshot information and text description) becomes essential research.

2.3. Problem statement

In this paper, we analyze how to perform crowdsourced test report classification based on textual information and image information. First, we invited three test experts from the industry with more than three years of mobile application testing experience to label the category. The experts classify test reports into 10 categories. The brief descriptions of categories are shown

in Table 2. For each crowdsourced test report, we invited two experts from the industry to label it. If the labels of the two experts are inconsistent, the third expert will review them to determine the category of the test report. If a test report contains multiple bugs and the bugs belong to different categories, the test report is labeled as "Other". Fortunately, the number of test reports with overlapping categories is very small. According to manual statistics, out of the more than 3000 test reports we conducted, we found only 124 overlaps. Testing tasks are often financially compensated, thus, crowd workers tend to submit as many test reports as possible. In this context, workers prefer to submit thousands of test reports with a single bug.

3. Preliminaries

3.1. Classification techniques

In recent years, a variety of classification techniques have been present. We give a brief overview of various main classifiers used in the paper (Pandey et al., 2017; Joulin et al., 2016).

k-Nearest Neighbors (KNN): KNN is a supervised learning predictable classification algorithm (Cunningham and Delany, 2007). KNN classification algorithm predicts the test sample's category according to the k training samples, which are the nearest neighbors to the test sample. k is a user-defined constant, and an unlabeled vector is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for text classification is Hamming distance, which we used in this paper.

KNN is simple and easy to implement. It can be used for non-linear classification. However, for datasets with a large sample size, the computational load is relatively large. When the sample is unbalanced, the prediction deviation is relatively large. We tried a lot of hyperparameters, and when the hyperparameter $k = 5$, we obtain the best classification effect. Thus, we set $k = 5$, which means we calculate the minimum 5 distances and then make the prediction based on the majority vote.

Naive Bayes (NB): Bayesian classifiers are statistical classifiers based on Bayes' theorem, and they can predict class membership probabilities. Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes, which is called class conditional independence (Leung, 2007). NB is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{X} = (x_1, \dots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities for each of K possible outcomes or classes C_k (John and Langley, 1995).

$$p(C_k | x_1, \dots, x_n) \quad (1)$$

However, there is a problem with the above formulation. If the number of features n is large and when a feature can take on a large number of values, then it is infeasible to base such a model on probability tables. Therefore, we reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k | \mathbf{x}) = \frac{p(C_k) p(\mathbf{x} | C_k)}{p(\mathbf{x})} \quad (2)$$

NB has low space requirements and is robust against irrelevant features. In this paper, we use multinomial NB because it is generally appropriate for discrete feature values (McCallum et al., 1998; Kibriya et al., 2004). It is typically used for document classification, with events representing the occurrence of a word in a single document (see the bag of words assumption). The likelihood of observing a histogram \mathbf{x} is given by

$$p(\mathbf{x} | C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}^{x_i} \quad (3)$$

Table 2
Descriptions and examples of each category.

Incomplete function	Some functions are not implemented.	Click “Sort by time”, and the list is still out of order.
Performance	It is a non-functional requirement of software, which describes the performance of the software in terms of responsiveness and stability.	Click the Details button and the interface will be displayed after 5 s.
Page layout	It represents the layout of the page is flawed.	The application can behave differently depending on the size of the user's screen.
Abnormal exit	It is a sudden exit interrupt that occurs when the application is opened.	The application screen flashes past and then returns to the desktop.
Installation failure	It represents the inability to install.	Application cannot be installed.
Registration failure	It represents the inability to Register.	After filling in the registration information, enter the account number, prompt no registration.
Update failure	It represents the inability to update.	Click Update and the system prompts that this version cannot be installed.
Login failure	It represents the inability to log in.	Use QQ authorization login, prompt authorization failure.
User experience	It refers to a feeling that is purely subjective in the process of using the product by users.	The picture is too big, and I need to swipe to see what follows.
Security	It means security risks that may result in unauthorized or internal user access or vandalism.	WeChat authorization failed.
Others	Some test reports that are invalid.	Some reports are just submitted by crowdsourced workers to try to use the crowdsourcing testing platform. We mark them as others.

The naive Bayesian model has a high speed for large numbers of training and queries. Even with very large training sets, there is usually only a relatively small number of features for each project. And it performs well on small-scale data and can handle multi-category tasks, suitable for incremental training. Besides, it is less sensitive to missing data, and the algorithm is relatively simple, often used for text classification.

Its primary disadvantage is the need to calculate the prior probability, and because of the assumption of the independence of the sample attributes, the effect might be not good if the sample attributes are related.

Random Forest (RF): Random forest is a classifier that contains multiple decision trees, and its output category is determined by the mode of the output categories of individual trees (Liaw et al., 2002). Random decision forests correct for decision trees' habit of overfitting to their training set. The random forests algorithm is as follows: Firstly, select a bootstrap sample from the training dataset to build the classifier. Secondly, for each of the bootstrap samples, grow a classification tree, with the following modification: at each node, rather than choose the best split among all predictors, choose the best split from a random sample of the predictors. Thirdly, predict new data by aggregating the predictions of the trees. Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way.

The random forest has the following advantages: It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier (Kleinberg et al., 1996). It gives estimates of what variables are important in the classification. And it generates an internal unbiased estimate of the generalization error as the forest building progresses. Besides, RF can reduce the risk of overfitting by averaging several decision trees. And it reduces the chance of stumbling across a classifier that does not perform well because of the relationship between the train and test data.

Decision Tree (DT): The decision tree classifier by Quinlan (2014) is one of the most well-known machine learning techniques. The decision tree classifier uses a decision tree to observe

an item to conclusions about the item's target value. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Each decision node corresponds to a test X over a single attribute of the input data and has multiple branches, each of which handles an outcome of the test X . Each leaf node represents a class, which is the decision result of a case. A decision tree can be used to classify testing data that has the same features as the training data. Starting from the root node of a decision tree, the test is carried out on the same attribute of the testing case as the root node represents. The decision process uses the branch that tests the attribute value to satisfy the condition. This branch guides the decision process to the child nodes of the root node. Recursively execute the same process until the leaf node is reached. Leaf nodes are associated with classes assigned to test cases.

The decision tree models are useful in statistics, data mining, and machine learning because it is relatively inexpensive to compute with reasonable accuracy. When using the decision tree to test data sets, it runs faster. Besides, the decision tree can be well extended to large databases, and its size is independent of the database size. The disadvantage of the decision tree is that it is difficult to process missing data, and it is prone to over-fitting.

Convolutional Neural Network (CNN): In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNN uses relatively little pre-processing compared to other classification algorithms, while the network learns the filters in a hand-engineered way in traditional algorithms. This independence from prior knowledge and human effort in feature design is a significant advantage. CNN is widely applied in image and video recognition, image classification, medical image analysis, and natural language processing (Collobert and Weston, 2008).

A convolutional neural network consists of one or more convolutional layers and the fully connected layer at the top (corresponding to the classical neural network), as well as the associated weight and pooling layer. This structure enables the convolutional neural network to utilize the two-dimensional structure

of input data. Compared with other deep learning structures, convolutional neural networks can give better results in image recognition. The model can also be trained using the backpropagation algorithm. Compared with other deep and feed-forward neural networks, convolutional neural networks have fewer parameters to consider, making them an attractive deep learning structure.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The “fully-connectedness” of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNN takes a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNN is on the lower extremity.

In this paper, we use the LeNet-5 CNN architecture (LeCun et al., 1998) which is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 average pooling layers, 1 fully connected layer, and 1 output layer. Before a pooling operation, the Sigmoid function is used to include nonlinearity. Euclidean Radial Basis Function units are used to the output layer classify 10 digits. Table 3 shows the detailed settings of the LeNet-5 network structure.

Support Vector Machine (SVM): A linear SVM is a non-probabilistic classifier that, given a training subset with $p - \text{dimensional}$ feature vectors belonging to two classes c_1 and c_2 , attempts to find a maximum margin $(p - 1) - \text{dimensional}$ hyperplane that separates the classes. In other words, the hyperplane has the maximum distance from the nearest training instances on either side. Support vector machines build non-linear classification models from training data for each class. These models are then used for predicting the class of new instances (Anvik, 2007). SVMs transform the original data into higher dimensionality and find a separating hyperplane in the newly mapped data. A hyperplane is found by using the support vectors and margins. If the data consists of two attributes, then it can be drawn in two-dimensional space (Thamaraiselvi and Kaliammal, 2004). Various lines can separate this two-dimensional data, but SVM finds a line with maximum margin (Thamaraiselvi and Kaliammal, 2004). Similarly, for n -dimensional data, a separating $n - 1$ dimensional hyperplane is found.

Classification of test reports according is a single label multi-class problem. Internally SVM divides the test reports into two classes so that one class is C_1 and all other classes are treated as second class (Assuming that test reports have C_2, C_3, C_4 and C_5 categories). Test reports of this class are again divided into two classes, one with test reports of category C_2 and the second class contains test reports having C_3, C_4 and C_5 categories. The same process continues until the test reports of all priority levels are separated. In this paper, we use SVM with a certain kernel. Kernel function can solve the problem of data nonlinearity without considering the mapping process.

The advantage of SVM is that it can solve machine learning problems with small samples, improve generalization performance, solve high-dimensional problems, solve non-linear problems, and avoid structural selection of neural networks and local minima problems. The disadvantage of SVM is that it is sensitive to missing data. There is no general solution for non-linear problems, and the Kernel function must be carefully chosen to deal with them. In this paper the hyperparameters of SVM include $\text{cost} = 100$, $\text{gamma} = 1e - 04$, $\text{kernel} = \text{rbf}$.

3.2. Performance measures

Precision, recall, and macro-averaged F-measure derived from external validity assessment are the most commonly used metrics, which have been widely employed to evaluate the performance of algorithms in document clustering (Panichella et al., 2015; Hammouda and Kamel, 2004) and document classification (Jiang et al., 2011; Li et al., 2008; Zanetti et al., 2013; Tian et al., 2013; Menzies and Marcus, 2015). To evaluate our proposed approaches, we use 2 metrics: accuracy and macro-averaged F-measure.

True Positive (TP): Number of reports correctly labeled to a class.

True Negative (TN): Number of reports correctly rejected from a class.

False Positive (FP): Number of reports incorrectly labeled to a class.

False Negative (FN): Number of reports incorrectly rejected from a class.

Precision: It is the ratio of the number of true positives to the total number of reports labeled by the classifier as belonging to the positive class.

$$Pre = \frac{TP}{TP + FP} \quad (4)$$

Recall: It is the ratio of the number of true positives to the total number of reports that actually belong to the positive class.

$$Rec = \frac{TP}{TP + FN} \quad (5)$$

F-measure or F1-score: It is the harmonic mean of precision and recall.

$$F = 2 * \frac{Pe * Re}{Pe + Re} = \frac{2TP}{2TP + FP + FN} \quad (6)$$

Accuracy: It measures how correctly the classifier labeled the records.

$$Acc = \frac{TP + TN}{TP + FP + FN + TN} \quad (7)$$

4. Proposed approach

4.1. Overview

The process of the classification is shown in Fig. 3. At first, we extract the historical test reports from the Mootest. For each test report, we extract the text description and screenshots. Secondly, we extract the feature of text and screenshots, respectively. For text description, after pre-processing, we extract textual features. For screenshots, we resize them and extract screenshot features using Spatial Pyramid Matching (SPM) (Lazebnik et al., 2006). Next, we implement feature fusion, which can be used as input to train the classifier. Finally, we build a classifier based on six classical classification algorithms, including KNN (k -Nearest Neighbors), NB (Naive Bayes), RF (Random Forest), DT (Decision Tree), CNN (Convolutional Neural Network), SVM (Support Vector Machine).

4.2. Textual feature extraction

The goal of feature extraction is to obtain features from crowd-sourced test reports, which can be potentially used as input to train classifiers. The fusion result of the textual feature vector and image feature vector compose the input of the classifier. For the textual feature, we first extract text descriptions and complete pre-processing. Text description pre-processing is one of the most important steps of textual feature extraction. The

Table 3
Detailed settings of the LeNet-5 network structure.

Network layer	Specific settings	Number of training parameters	Output characteristic
Input layer	A black-and-white image of 32×32 pixels	0	$32 \times 32 \times 1$
Conv1 layer	6 convolution kernels of size 5×5 , stride = 1	156	$28 \times 28 \times 6$
Pool1 layer	Pool size is 2×2 , stride = 2	12	$14 \times 14 \times 6$
Conv2 layer	16 convolution kernels of size 5×5 , stride = 1	1516	$10 \times 10 \times 16$
Pool2 layer	Pool size is 2×2 , stride = 2	32	$5 \times 5 \times 16$
FC1 layer	120 neurons	48120	$1 \times 1 \times 120$
FC2 layer	84 neurons	10164	1×84
FC3 layer	10 neurons	840	1×10

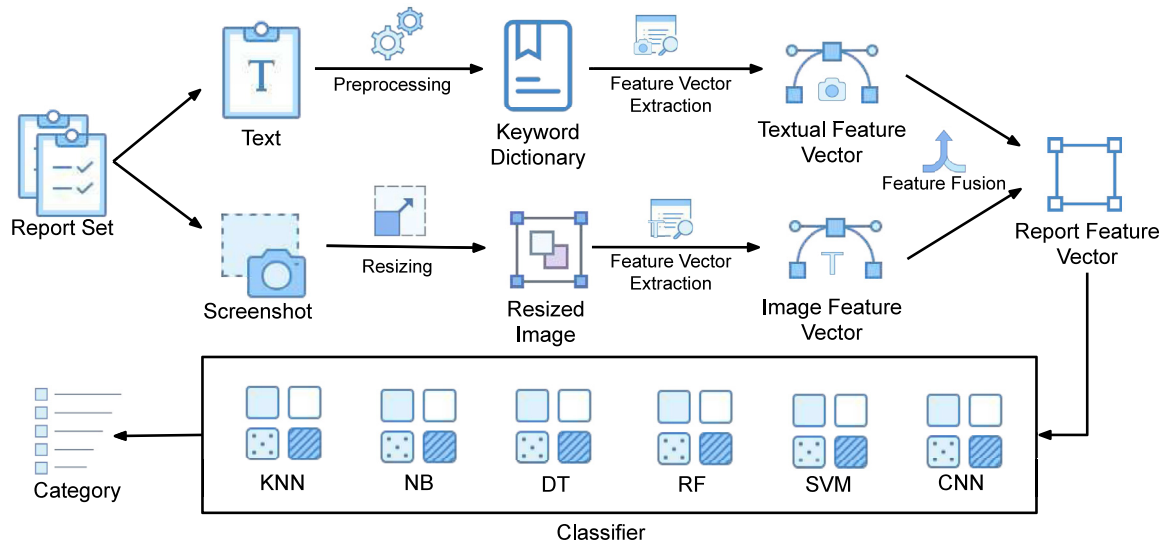


Fig. 3. The workflow of our approach.

description obtained from test reports is in raw form and cannot be directly used for training the classification algorithm. It is first pre-processed to make it useful for training purposes. In contrast to English words with natural spaces, we need a tool for segmentation, as the crowdsourced test reports in our experiment are written in Chinese. Many efficient tools have been developed for word segmentation of Chinese documents, such as ICTCLAS⁷ and IKAnalyzer,⁸ which have been widely adopted by existing studies (Feng et al., 2015; Zhang et al., 2010). In our study, we adopt the Language Technology Platform (LTP) (Che et al., 2010) for word segmentation and segment descriptions into words. It provides high-performance language processing modules, integrating these modules smoothly, using processing results conveniently, and showing processing results directly. LTP is an open-source Chinese NLP tool and widely used in Chinese processing. We then remove stopwords that are considered to be unhelpful for similarity computation and only keep verbs and nouns to reduce noise. Another problem is that workers often use different words to express the same concept. We introduce the synonym replacement technique to mitigate this problem. A synonym library of LTP is adopted.⁹

After pre-processing, we utilize the bag-of-words model to represent textual information. In the bag-of-words model, a text is represented as the multi-set of its words, disregarding grammar

and even word order but keeping multiplicity. The vocabulary obtained after applying the bag-of-words model on data has very large dimensionality. Most of these dimensions are not related to text categorization and thus result in reducing the performance of the classifier. To decrease the dimensionality, we use TF-IDF, a weighting technique popular in information retrieval and textual feature extraction, to capture the relevancy among words, text documents, and particular categories. The process of feature selection is used, which takes the best k terms out of the whole vocabulary that contributes to accuracy and efficiency. The best 100 terms represent the feature of the text description. Finally, we have $T_i = \{t_1, t_2, \dots, t_m\}$, in which, t_i denotes the i th word in the test report r_i , because each text description consists of multiple word.

4.3. Image feature extraction

For screenshots, to extract features for screenshots, we extract screenshot features. Bug screenshots provide not only views of buggy symptoms, but also app-specific visual appearances. We hope to automatically identify application behaviors based on their visual appearance in the screenshots. However, the screenshots often have variable resolution and complex backgrounds. Therefore, modeling the similarity between the screenshots merely based on RGB may not be well suited for our task. To address the challenges, we apply the Spatial Pyramid Matching (SPM) (Lazebnik et al., 2006) to build a global representation of screenshots. SPM is a method for recognizing scene categories based on approximate global geometric correspondence. and its

⁷ <http://ictclas.nlp.ir.org/>

⁸ <http://www.oschina.net/p/ikalyzer>

⁹ http://www.ltp-cloud.com/download#down_cilin

basic approach is subdividing the image iteratively at different levels of resolution and computing the orderless histogram of local features at each sub-region, and then concatenating these spatial histograms with different weights. The final concatenated histogram represents the visual feature of screenshots. We employ s_{ij} to denote the j th screenshot in the test report r_i and $S_i = \{s_{ij} | j = 0 \dots n\}$.

4.4. Feature fusion and classification

Firstly, we regularize the feature vector. In statistics and applications of statistics, normalization can have a range of meanings (Dodge and Commenges, 2006). In the simplest cases, normalization of ratings means adjusting values measured on different scales to a notionally common scale, often prior to averaging. In more complicated cases, normalization may refer to more sophisticated adjustments where the intention is to bring the entire probability distributions of adjusted values into alignment. A different approach to the normalization of probability distributions is quantile normalization, where the quantiles of the different measures are brought into alignment. For machine learning, one of the key problems is to design an algorithm that not only has a small error in a training set but also has good generalization ability in new samples. If a classifier gets trained with so much data, it begins to learn from the noise and inaccurate data entries in the data set and is more prone to overfitting. Because of its powerful representation ability, neural networks are often overfitting, so they need to use many different forms of regularization strategies. However, different classification algorithms often require different feature engineering (Xue et al., 2015; Chandrashekar and Sahin, 2014). To fairly compare the classification effects of different classifiers, we did not manually specify feature selection methods for different classifiers. L2 regularization is a widely used technique to reduce overfitting (Derhab et al., 2020; Ying, 2019; Phaisangittisagul, 2016). For KNN, NB, SVM, and CNN we use L2 regularization to add a regularization term after the cost function:

$$C = C_0 + \frac{\lambda}{2n} \sum_w W^2 \quad (8)$$

C_0 represents the original cost function, and the latter one is the L2 regularization term, which is the sum of the squares of all parameters W divided by the sample size n of the training set. λ is the coefficient of the regular term, weighing the proportion of the regular term and the 20 terms. Unlike the above four methods, Decision Tree does not have a global loss function in the same sense of linear regression and its regularization just not in the form of a penalty to the cost function. Therefore, we do not apply L2 regularization to DT. Similarly, as Random Forest composed of multiple decision trees, L2 regularization is also not applicable. Then we use min-max normalization to normalize them into the range between 0 and 1.

Secondly, to unify the expression of different test reports, we use one vector to represent a test report, and we fuse textual features and image features. If there is more than one screenshot in a test report, to fuse multiple screenshots into one vector, we add the vectors and average them.

$$S'_i = \frac{\sum_{j=0}^n S_{ij}}{n} \quad (9)$$

Then, we represent a software project with submitted crowdsourced test reports with two-dimensional vectors $R(r) = \{r(T_i, S_i) | i = 0 \dots n\}$, in which, S denotes the screenshots (i.e., images) containing the views that may capture symptoms of the bug being reported, and T denotes the text describing the buggy behavior. Each test report is represented as a two-dimensional

vector of length 2, and the first line is T_i , the second line is S'_i . Next, we flatten each vector from the multidimensional vector to the one-dimensional vector. The flattened result is the input of classifiers.

Then, we build classifiers. After we obtain the features for each crowdsourced test report, we divide the test set and the training set. We randomly select 10% of the test report as a test set and the rest as the training set. After we obtain both the training set and test set, we build different machine learning classifiers based on the training set. This step aims at learning classifiers to classify unlabeled reports, using labeled reports. In the training phase, the training set is fed into a classifier to build a classification model. Subsequently, we input the test subset to check how correctly the trained classifier can label the issues. Then we use the test set to evaluate the performance of built classifiers. To ensure the accuracy of the experimental results, we repeat the experiment 30 times for each classifier, each time using different training and test subsets.

5. Experiment

5.1. Research questions

To investigate the effectiveness of screenshots in test report classification and the scenarios of different classification methods, we conduct empirical studies to answer the following three research questions.

RQ1. Effectiveness of Screenshots. How effective are the methods incorporating image features in classifying crowdsourced test reports? RQ1 is designed to inform whether image-understanding methods can assist test report inspection compared with the text-based methods. To further demonstrate the advantages of screenshots, we compare the performance of classification methods based on the fused features (i.e. the text description and screenshots) with those based on the only text descriptions.

RQ2. Effectiveness of classifiers. Which is the best method in classifying crowdsourced test reports? RQ2 is designed to investigate the performance of six widely used methods in classifying crowdsourced reports. To demonstrate the performance, we compare the *accuracy* and *F-measure* of different classification methods.

RQ3. The impact of training set sizes. How does the training set size impact the performance of the methods? RQ3 is designed to investigate the effect of training set size on the performance of different classification algorithms. We investigate the performance of methods against different training set sizes.

RQ4. The efficiency of classifiers. How efficient is the classification of test reports based on fusion features? RQ4 is designed to evaluate the time cost of classifying test reports based on different characteristics. We evaluate the time cost of feature extraction, model building training, and testing for different classifiers.

5.2. Data collection

To produce the dataset for our evaluation, we utilized the results of the 2016 National Software Testing Contest in China,¹⁰ which is the first National Software Testing Contest. The contest simulated crowdsourced testing of several popular mobile applications across multiple domains (including games, education, social media, and so on). We select the six mobile applications, including Mylistening, Hujiang, Huawei store, TuNiu, Game2048,

¹⁰ http://www.moocetest.org/cst2017/cst2016/index_en.html

Table 4
Statistical information of testing applications.

Name	Version	Size	Category	T	R	S	R _s
Mylistening	1.6.2	21.48M	Education	286	473	418	306
Hujiang	2.12.0	14.5M	Education	93	269	436	241
Huawei	2.0.1	61.19M	Health	68	262	327	201
Tuniu	9.0.0	29.65M	Travel	397	531	640	418
Game2048	3.14	2.71M	Games	142	210	219	164
Slife	2.0.3	6.3M	Health	955	1346	2238	1124

Table 5
The distributions of each category.

	Mylistening	Hujiang	Huawei	Tuniu	Game2048	Slife
Incomplete function	73	40	22	75	44	357
Performance	55	25	38	68	28	134
Page layout	27	40	5	335	3	173
Abnormal exit	12	7	27	9	42	98
Installation failure	21	14	1	1	0	42
Registration failure	14	6	10	42	0	126
Update failure	32	9	3	21	3	52
Login failure	12	9	1	49	2	63
User experience	112	51	40	119	70	217
Security	39	22	12	65	3	48
Others	76	13	3	28	16	36

and Slife, to evaluate the performance of the different classifiers. The introduction of each application is as follows:

Mylistening. It is an English listening learning and teaching application. Target users are IELTS, TOEFL students, and teachers.

Hujiang. It is an English learning platform and provides related educational products and services, such as English speech videos, e-book, and courses.

Huawei. Huawei Sports Health application runs on the mobile phone, providing users with professional sports records, scientific sleep function, and health function.

Tuniu. It is a travel application that provides travel product booking services and provides follow-up services and guarantees.

Game2048. It is a single-player sliding block puzzle game. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048. However, one can continue to play the game after reaching the goal, creating tiles with larger numbers.

Slife. It is a health-like app that automatically records user movements around the clock. It supports automatic recognition and calorie calculations for aerobics such as walking, running, cycling, and helps users automatically generate daily life trajectories. The detailed information of the dataset is shown in Table 4, in which, |T| denotes the number of testers, |R| denotes the number of reports, |S| denotes the number of screenshots and |R_s| denotes the number of reports that contain at least one screenshot. In total, we conducted experiments using more than 3000 test reports from 6 real industrial mobile applications in crowdsourced testing.

We conduct various experiments to determine how well a classifier can label the instances based on different features and how different training set sizes affect the classifier performance. First, as shown in 5.2, we collect 3091 test reports from 6 real industrial mobile applications. We invited three developers from the industry with more than three years of software testing experience to inspect and classify the crowdsourced test reports. After inspection, we eliminate 172 invalid test reports (That is, the test report category is other), such as empty test reports and overlapping categories of reports. For 2919 valid reports, we define the expert-flagged test report category as the ground-truth category. Table 5 shows the distributions of each category.

5.3. Experimental design

After obtaining the above 2919 valid test reports, they are randomly divided into 10 groups. Then, we randomly select one

group as the test set and the remaining data as the training set. Next, we use 6 classifiers (KNN, NB, DT, RF, SVM, and CNN) for experiments. For each classifier, experiments were conducted based on text features, image features, and fusion features. For each set of experiments, we conducted 30 times 10-fold cross-validation to evaluate the performance of the different classifiers. We manually set the hyperparameters that are shown in 3.1. For hyperparameters not mentioned, default values are used. We do not tune them extensively but did perform some limited experiments on a smaller subset of the data to arrive at the given configurations. The specific experimental results will be described in the next section.

5.4. Experimental results

In this section, we conduct various experiments to access the performance of our approaches.

RQ1. Effectiveness of Screenshots.

To demonstrate the effectiveness of image features for test report classification, we compare the performance of the classification methods based on the fused feature (i.e., the textual feature and the image feature) with those only based on the text description with the measures mentioned in Section 3.2. Fig. 4 shows the accuracy of different classification methods. Areas of different colors represent different applications, and different figures show different classification methods. For each application, we use 6 classification algorithms, for each of which we randomly conduct 30 experiments. The red area represents Mylistening, the blue represents Hujiang, the orange represents Huawei, the gray represents Tuniu, the purple represents Game2048, and the brown represents Slife. For each application with the same method, the left box shows the F-measure of the method based on the text description (Acc_t), the middle shows the F-measure of the method based on screenshots (Acc_i), and the right represents the method based on the text description and screenshots (Acc). Fig. 5 shows the F-measure for different classification methods.

The experimental results reveal that the method based on the text description and screenshots perform better than those only based on the text description. For Mylistening, Hujiang, Huawei store, Tuniu, and Slife, the classification accuracy based on text description and screenshots is 9.2% higher than that based only on text features. And the F based on text description and screenshots is 13.3% higher than that based only on text features. It is obvious

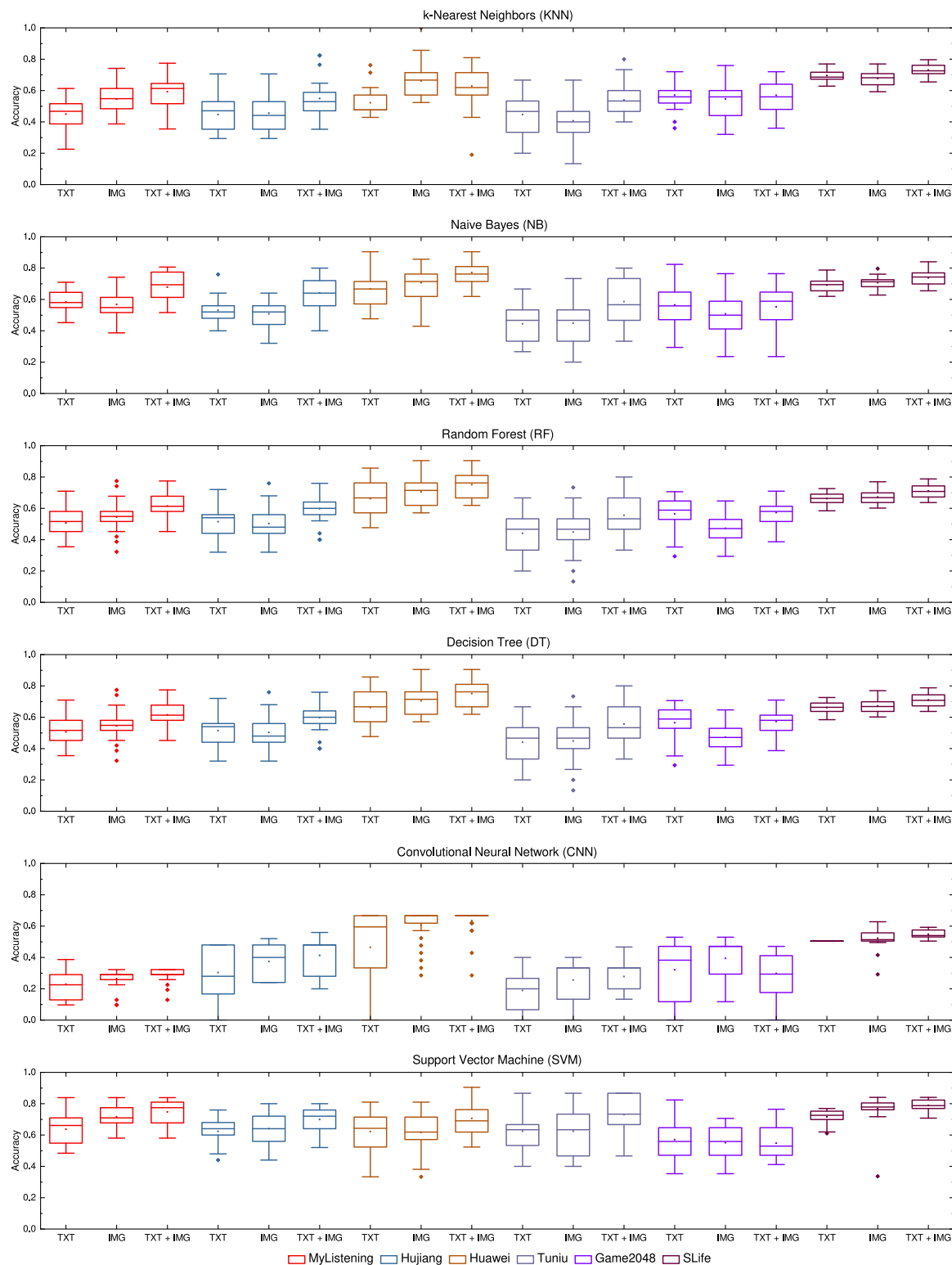


Fig. 4. Average accuracy for Effect of the image in classifying test reports of different applications. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

that image features can improve automated classification test reports. For example, in Mylistening, the Acc of classification results based on fused features is about 10% higher than that only based on text. The F based on the fused feature is about 12% higher than text-based classification. To investigate whether the effectiveness

of methods using text descriptions or fused information is significantly different, we conducted t-tests,¹¹ whose results are shown in Tables 6 and 7. The results indicate that fused features can improve classification efficiency significantly compared with text (where significance occurs with $p < 0.05$) except for Game2048,

¹¹ The data complies with the t-test normality hypothesis.

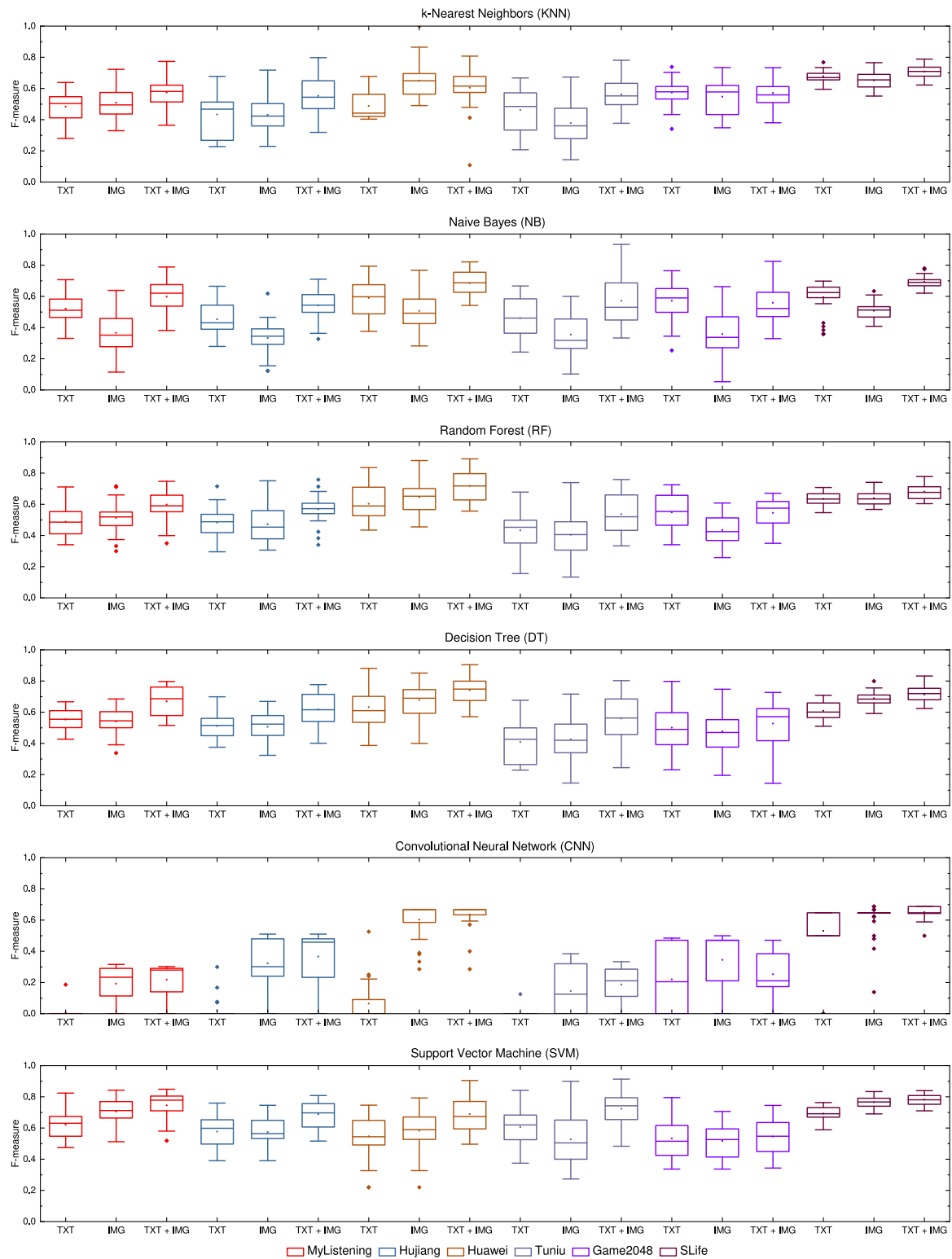


Fig. 5. Weighted average F-measure of effect of the image in classifying test reports of different applications.

which has fewer interfaces than other applications. Unlike other applications, Game2048 not only contains a small number of test reports but also has only one graphic user interface, such that the effect of the image features on test report classification is minimal or even negative.

It is obvious that screenshots can more intuitively reproduce the usage scenarios where the bug occurs and are not subject to

subjective factors of crowd workers. Although text can describe bug information flexibly, for mobile application testing, using screenshots is more convenient than typing text on a phone keyboard. As we can expect, screenshots will play a more important role in crowdsourced mobile test report classification. In summary, in addition to single-interface applications, screenshots

Table 6Statistical significance results (p) of Acc.

	KNN	NB	RF	DT	CNN	SVM
Mylistening	0.000	0.000	0.000	0.000	0.001	0.000
Hujiang	0.000	0.002	0.001	0.000	0.002	0.000
Huawei	0.000	0.000	0.000	0.000	0.001	0.000
Tuniu	0.000	0.002	0.001	0.000	0.001	0.000
Game 2048	0.476	0.202	0.357	0.377	0.252	0.219
Slife	0.000	0.000	0.000	0.000	0.000	0.000

Table 7Statistical significance results (p) of F .

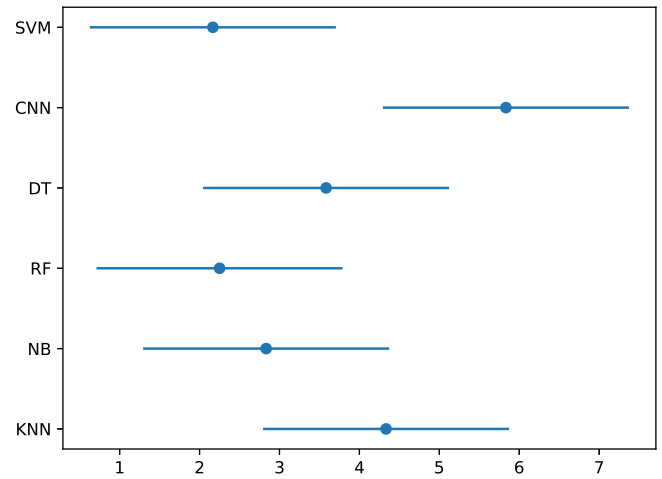
	KNN	NB	RF	DT	CNN	SVM
Mylistening	0.000	0.000	0.000	0.000	0.000	0.000
Hujiang	0.000	0.001	0.001	0.000	0.000	0.001
Huawei	0.000	0.000	0.000	0.000	0.000	0.000
Tuniu	0.000	0.001	0.001	0.000	0.000	0.000
Game 2048	0.363	0.305	0.455	0.266	0.211	0.354
Slife	0.000	0.000	0.000	0.000	0.000	0.000

can help improve the effectiveness of crowdsourced mobile test reports.

RQ2. Effectiveness of classifiers.

Among the classifiers we used, we aim to find out which one works best on the given datasets. Table 8 shows the average of repeated experiments and illustrates the performance of different test report classification algorithms. For each application, we show the Acc and F of the classification based on the fused feature (i.e., the text feature and the image feature). It can be observed that the Support Vector Machine (SVM) gives the highest average accuracy and F -measure for most datasets we considered. In Mylistening, Hujiang, Tuniu, and Slife, it can achieve the highest performance. It can be concluded that classification using SVM performs better than others in these applications, and it can outperform the second-best one by 10%–20% better on average. For Huawei store, though SVM achieves 0.706 of Acc, which is 0.065 lower than DT, and F is 0.053 lower than DT. When using Naive Bayes (NB) to classify, ACC is the highest, while F is the highest when using k -Nearest Neighbors (KNN). As shown in Table 4, there are fewer test reports of the Huawei store and Game 2048. Therefore, based on the results of experiments with Huawei store, we can infer that DT is more suitable for the small-scale dataset. In Game2048, it has only one interface that means screenshots could not effectively support test report classification. The negative effect of image features affects the accuracy of report classification and also makes the performance of different classification algorithms unstable.

In general, different classification algorithms have different characteristics and are suitable for different usage scenarios. In our experiment, the classification using k -Nearest Neighbors (KNN) is less affected by the scale and quality of the dataset. It can be observed that the ACC and F of classification using this algorithm are not much different for different applications. Although the number of test reports in Slife is far more than that of other applications, the accuracy and F -measure do not improve significantly. The main advantage of KNN is simple to understand and equally easy to implement. However, as the dataset grows, the efficiency or speed of KNN may decline. For Naive Bayes (NB), as shown, the Acc and F of classification based on text description tends to be high. It is less sensitive to missing data, and the algorithm is simpler and is often used for text classification. Naive Bayes classifiers mostly used in text classification (due to better results in independence rule) have a higher success rate as compared to other algorithms. For Decision Trees (DT), as mentioned above, it is more suitable if the data set is small. And it is easy to use and explain with simple math, no complex

**Fig. 6.** Friedman test results of Accuracy.

formulas. It also presents visually all of the decision alternatives for quick comparisons in a format that is easy to understand with only brief explanations. To some extent, Random Forests (RF) are a combination of DT, and it also performs better when the data set is small. For CNN, the experimental results of Slife are obviously better than other applications, because Slife has more test reports. CNN can perform better if we can provide sufficient data and processing power. However, the computing cost can be huge and the speed of recognition in an embedded scenario is terrible. SVM is the most suitable classification algorithm for the current experimental data.

In addition, for all classification algorithms, ACC of Slife and Huawei store is generally higher than other applications. Slife has far more test reports than other applications, so the classification prediction accuracy and F -measure are higher. In Huawei store, during the experiment, we found that screenshots in different bugs are quite different, and fewer different bugs are appearing on the same page, so the classification accuracy and F -measure are higher. Therefore, the scale and quality of the training set have a significant impact on the classification results. Especially, when using Convolutional Neural Network (CNN) for classification, this effect is more obvious, as shown in Table 8.

To detect the differences between classifiers, we use the Friedman test and the Nemenyi test. Firstly, we use the non-parametric Friedman test to evaluate the rejection of the hypothesis that all the classifiers perform equally well for a given level. We rank the classifiers for each application, the better performing classifier getting the higher rank. Tables 9 and 10 shows the average ranks of the 3 experimental applications. We can observe that SVM obtains the best average performances for all data sets.

Then, the Friedman test compares the average ranks of the classifiers and calculates the Friedman statistic as follows.

$$\tau_{\chi^2} = \frac{12N}{k(k+1)} \left(\sum_{i=1}^k r_i^2 - \frac{k(k+1)^2}{4} \right) \quad (10)$$

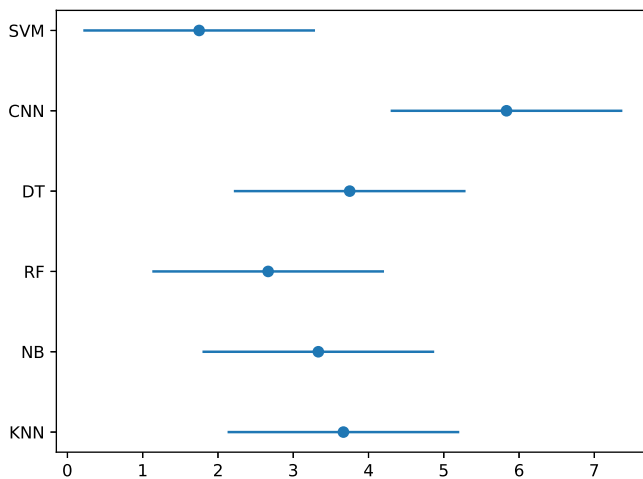
where k represents the number of classifiers, N represents the number of data sets, R_i is the average rank of the i th classifier. In this paper, $k = 6$ and $N = 6$. The Friedman test results is shown in Figs. 6 and 7. The Friedman statistic is calculated as above, and distributed according to τ_{χ^2} with $C - 1$ degrees of freedom, when k and N are big enough. So, the null hypothesis are rejected.

Next, we can proceed with a post-hoc test.

We use the Nemenyi test is used when all classifiers are compared to each other. The performance of two classifiers is

Table 8
Performance comparison between different methods.

Algorithm	Accuracy						
		KNN	NB	RF	DT	CNN	SVM
Mylistinging	TXT	0.449	0.545	0.506	0.584	0.229	0.637
	IMG	0.545	0.442	0.545	0.568	0.265	0.716
	TXT + IMG	0.592	0.624	0.615	0.678	0.288	0.748
Hujiang	TXT	0.447	0.494	0.515	0.531	0.304	0.623
	IMG	0.455	0.428	0.503	0.508	0.375	0.640
	TXT + IMG	0.549	0.575	0.597	0.641	0.413	0.699
Huawei	TXT	0.522	0.641	0.662	0.667	0.465	0.622
	IMG	0.662	0.584	0.705	0.706	0.605	0.617
	TXT + IMG	0.629	0.724	0.751	0.771	0.635	0.706
Tuniu	TXT	0.447	0.504	0.440	0.444	0.189	0.624
	IMG	0.407	0.427	0.449	0.449	0.256	0.624
	TXT + IMG	0.540	0.593	0.556	0.586	0.278	0.731
Game2048	TXT	0.571	0.618	0.565	0.565	0.321	0.571
	IMG	0.545	0.484	0.508	0.508	0.394	0.551
	TXT + IMG	0.569	0.584	0.553	0.553	0.298	0.547
Slife	TXT	0.696	0.621	0.661	0.691	0.504	0.716
	IMG	0.678	0.609	0.672	0.707	0.523	0.768
	TXT + IMG	0.730	0.722	0.711	0.737	0.548	0.789
Algorithm	F-measure						
		KNN	NB	RF	DT	CNN	SVM
Mylistinging	TXT	0.484	0.520	0.488	0.553	0.006	0.622
	IMG	0.508	0.366	0.514	0.540	0.192	0.706
	TXT + IMG	0.574	0.599	0.598	0.669	0.218	0.745
Hujiang	TXT	0.433	0.453	0.483	0.511	0.021	0.577
	IMG	0.431	0.332	0.472	0.506	0.323	0.572
	TXT + IMG	0.552	0.542	0.568	0.617	0.367	0.688
Huawei	TXT	0.487	0.591	0.603	0.632	0.064	0.546
	IMG	0.650	0.507	0.646	0.678	0.602	0.582
	TXT + IMG	0.605	0.685	0.717	0.742	0.634	0.689
Tuniu	TXT	0.462	0.460	0.433	0.408	0.004	0.607
	IMG	0.379	0.355	0.404	0.426	0.146	0.527
	TXT + IMG	0.562	0.572	0.536	0.559	0.187	0.724
Game2048	TXT	0.573	0.572	0.548	0.500	0.219	0.533
	IMG	0.547	0.358	0.435	0.477	0.345	0.517
	TXT + IMG	0.571	0.559	0.545	0.527	0.253	0.545
Slife	TXT	0.678	0.595	0.635	0.609	0.531	0.692
	IMG	0.642	0.508	0.639	0.689	0.613	0.766
	TXT + IMG	0.709	0.693	0.681	0.713	0.650	0.782

**Fig. 7.** Friedman test results of *F-measure*.

significantly different, if the corresponding average ranks differ by at least the critical difference:

$$CD = q_{\alpha} \sqrt{\frac{k(k+1)}{6N}} \quad (11)$$

Table 9

The average ranks of the *Accuracy* for the Friedman/Nemenyi test.

Classifier	Average rank
<i>k</i> -Nearest Neighbors (KNN)	4.33
Naive Bayes (NB)	2.83
Random Forest (RF)	2.25
Decision Tree (DT)	3.58
Convolutional Neural Network (CNN)	5.83
Support Vector Machine (SVM)	2.16

Nemenyi test calculates all pairwise comparisons between different classifiers and checks which classifier's performance differences exceed the critical difference. The results of the pairwise comparisons are shown in Tables 11 and 12, which critical difference (CD) value is 3.08.

In general, compared with CNN, the performance of SVM is significantly different, while the performance of other classifiers is not significant.

RQ 3. The impact of different training set sizes.

Tables 13–18 presents the accuracy and F-measure for different classification algorithms with different amounts of training data (ratio 0.1–ratio 0.9). Note that in our longitudinal data setup, we divide our data into 10 non-overlapping frames. Thus one frame corresponds to 10 percent (1/10) of the total number of test reports. At a ratio 0.1, the amount of training data is 10

Table 10The average ranks of the *F-measure* for the Friedman/Nemenyi test.

Classifier	Average rank
<i>k</i> -Nearest Neighbors (KNN)	3.66
Naive Bayes (NB)	3.33
Random Forest (RF)	2.67
Decision Tree (DT)	3.75
Convolutional Neural Network (CNN)	5.83
Support Vector Machine (SVM)	1.75

Table 11Pairwise comparisons (*Accuracy*) of Nemenyi test.

	KNN	NB	RF	DT	CNN	SVM
KNN	0	1.5	2.08	0.75	1.5	2.17
NB		0	0.28	0.75	3	0.67
RF			0	1.33	3.58	0.09
DT				0	2.25	1.42
CNN						3.67
SVM						0

Table 12Pairwise Comparisons (*F-measure*) of Nemenyi Test.

	KNN	NB	RF	DT	CNN	SVM
KNN	0	0.33	0.99	0.09	0.83	1.91
NB		0	0.66	0.42	2.5	1.58
RF			0	1.08	3.16	0.91
DT				0	2.10	2.00
CNN						4.08
SVM						0

percent of the total number of test reports, and a ratio 0.9, the amount of training data is 90 percent of the total number of test reports.

We take *k*-Nearest Neighbors (KNN) as an example to introduce the experimental results. The results of other methods are similar, and the experimental data are attached. Tables 13–18 is clearly shown that *F* increases as training to testing ratio increases. The highest *F* is obtained when the ratio of the training set is 90%, and the testing set is 10%. For example, in Mylistening, its *F* achieves the maximum of 0.592 when we divided the training set and the test set according to the ratio of 90% of the training set and it has a minimum of 0.314 when according to the ratio of 10% of the training set. Except for Huawei, the *Acc* and *F* of the classification results increases with the increase of the training set. In Huawei, the *F* is from 0.483–0.659 and the *Acc* is best when the training set ratio is 80%. We consider that 80 percent of the data set can predict test report classification results effectively. For Slife, the *Acc* and *F* of classification results change gently with the increase of the training set, because it contains enough test reports to complete classification. The results show that experiments using NB, DT, RF, CNN, and SVM were similar. The experiment reveals that with the increase of training set size, both the *Acc* and *F* would first improve and then remain almost unchanged. In Mylistening, Hujiang, Huawei, Tuniu and Game2048, We focus on that, when the training set ratio is 70%, increasing the training set has little effect on the accuracy of classification results. As is shown in Table 4, the five apps contained 2881 test reports. We consider that 70% of test reports can be effectively classified as training sets.

With so small numbers of test reports, CNN might not work well. However, our experimental projects are the results of the national software-testing contest and it simulates a real crowdsourcing test process and reflects the number of test reports in a real-world crowdsourcing scenario. Also, for CNN, classifying test reports based on fusion features is time-consuming (as shown in Section 5.4), so we do not recommend using deep learning

algorithms for crowdsourcing test report classification. For KNN, NB, DT, RF and SVM, merely 350 labeled data test reports are needed for achieving relatively satisfactory performance. In conclusion, SVM generally performs better than other algorithms, and can better solve the classification problem in small sample cases. In summary, machine learning algorithms such as KNN, NB, DT, RF and SVM can classify crowdsourced test reports almost in real-time.

RQ 4. The efficiency of classifiers.

The purpose of the classification of test reports is to further automate the assignment of test reports. If the classification time cost of an automated test report is too long, it may not be suitable for real industry applications. To evaluate the time cost of feature extraction, we verify the time cost (s) of extracting different features for test report classification, as shown in Table 19.

First, we extract the textual feature from each test report, perform the procedure shown in Section 4.2, and record the required time, as is shown in the second row of Table 5. Next, we extract the image feature from each test report, perform the procedure shown in Section 4.3, and record the required time, as is shown in the third row of Table 5. Third, we fuse the results of the above two steps and record the time. Finally, we add up the times for the three steps above, as is shown in the fourth row of Table 5.

Compared with text, it takes more time to extract image features because the features of images are more complex. The time cost of test report classification based on fusion feature is slightly larger than the sum of text and image. The integration process also takes a small amount of time, which is acceptable. The time cost of feature extraction can meet the requirements of automated crowdsourced test report classification. In the industry, the classifier could automatically classify test reports in real-time or automatically assign bugs to developers.

Table 20 shows the average time cost(s) of model building, training, and classification. As shown in the second paragraph of Section 5.3, we repeated the experiment 30 times for each application and recorded the time points for the above steps. For KNN, NB, RF, DT, CNN, and SVM, the time cost is very small and can classify test reports in real-time. Compared with the above machine learning algorithms, CNN is more time-consuming to classify crowdsourced test reports. With the increase in the number of test reports, the time cost of using CNN to classify test reports increases significantly. In conclusion, SVM generally performs better than other algorithms, and can better solve the classification problem in small sample cases.

6. Related work

The main related works are crowdsourced testing, test report classification, and application of image understanding on testing.

6.1. Crowdsourced testing

Crowdsourcing refers to the process of an organization crowdsourcing their work to undefined and generally large networks of people in an open call form (Howe, 2006; Mao et al., 2016). Crowdsourced testing has been widely applied to solve resolve software engineering problems. For example, Dolstra et al. (2013) described a method for crowdsourcing of GUI tests based on instantiating the system under test in virtual machines that are served to a geographically dispersed pool of workers. Ning Chen and Sunghun Kim presented PAT (Chen and Kim, 2012), which decomposes object mutation and complex constraint solving problems into small puzzles for humans to solve. Gomide et al. (2014) proposed a usability testing process based on crowdsourcing,

Table 13
Training to testing ratio of k -Nearest Neighbors (KNN).

Application	Accuracy								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.317	0.366	0.408	0.476	0.519	0.542	0.563	0.590	0.592
Hujiang	0.320	0.389	0.444	0.484	0.492	0.506	0.521	0.533	0.549
Huawei	0.383	0.459	0.520	0.594	0.628	0.633	0.636	0.654	0.629
Tuniu	0.231	0.369	0.363	0.394	0.446	0.488	0.528	0.531	0.540
Game2048	0.354	0.402	0.446	0.508	0.538	0.550	0.537	0.553	0.569
Slife	0.541	0.607	0.665	0.696	0.704	0.706	0.706	0.710	0.730
Application	F-measure								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.304	0.349	0.394	0.456	0.498	0.519	0.576	0.573	0.574
Hujiang	0.315	0.363	0.413	0.479	0.469	0.490	0.514	0.540	0.552
Huawei	0.416	0.460	0.515	0.584	0.580	0.592	0.929	0.592	0.605
Tuniu	0.200	0.312	0.340	0.356	0.415	0.455	0.512	0.545	0.562
Game2048	0.389	0.404	0.475	0.490	0.510	0.520	0.546	0.548	0.571
Slife	0.526	0.578	0.670	0.672	0.681	0.691	0.701	0.691	0.709

Table 14
Training to testing ratio of Naive Bayes (NB).

Application	Accuracy								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.420	0.449	0.474	0.504	0.532	0.566	0.591	0.613	0.624
Hujiang	0.324	0.372	0.424	0.487	0.487	0.511	0.565	0.569	0.575
Huawei	0.310	0.370	0.503	0.532	0.609	0.658	0.661	0.695	0.724
Tuniu	0.255	0.402	0.441	0.477	0.501	0.490	0.556	0.582	0.593
Game2048	0.296	0.349	0.366	0.437	0.568	0.584	0.577	0.570	0.584
Slife	0.506	0.573	0.629	0.649	0.637	0.643	0.632	0.635	0.722
Application	F-measure								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.378	0.427	0.455	0.496	0.541	0.560	0.561	0.581	0.599
Hujiang	0.305	0.350	0.396	0.458	0.461	0.470	0.521	0.540	0.542
Huawei	0.321	0.397	0.452	0.529	0.597	0.626	0.620	0.657	0.685
Tuniu	0.291	0.380	0.393	0.436	0.461	0.501	0.522	0.545	0.572
Game2048	0.254	0.293	0.323	0.387	0.487	0.559	0.538	0.546	0.559
Slife	0.524	0.544	0.588	0.617	0.610	0.651	0.673	0.681	0.693

Table 15
Training to testing ratio of Random Forest (RF).

Application	Accuracy								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.301	0.366	0.400	0.492	0.505	0.546	0.570	0.598	0.615
Hujiang	0.363	0.408	0.439	0.477	0.498	0.496	0.500	0.501	0.597
Huawei	0.486	0.506	0.583	0.640	0.662	0.674	0.692	0.726	0.751
Tuniu	0.204	0.344	0.360	0.388	0.448	0.473	0.501	0.636	0.556
Game2048	0.334	0.398	0.445	0.480	0.489	0.476	0.511	0.565	0.573
Slife	0.542	0.586	0.630	0.638	0.650	0.655	0.673	0.696	0.711
Application	F-measure								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.324	0.383	0.425	0.461	0.472	0.492	0.527	0.540	0.598
Hujiang	0.320	0.372	0.405	0.449	0.460	0.464	0.470	0.518	0.568
Huawei	0.459	0.482	0.606	0.629	0.624	0.653	0.670	0.680	0.717
Tuniu	0.215	0.313	0.333	0.379	0.420	0.462	0.494	0.529	0.536
Game2048	0.323	0.387	0.424	0.471	0.461	0.472	0.486	0.537	0.545
Slife	0.526	0.557	0.595	0.616	0.643	0.668	0.658	0.663	0.681

which can reduce time and costs related to traditional usability tests.

Many other researchers focus on solving the newly encountered problem in crowdsourced testing, such as distinguishing duplicate reports (Wang et al., 2009), prioritizing test reports (Feng et al., 2015, 2016), classifying test reports (Feng et al., 2016), etc. Feng et al. (2015) and Feng et al. (2016) presented prioritization techniques to assist inspections of crowdsourced test reports. They designed strategies to dynamically select the riskiest and diversified test reports for inspection in each iteration. Wang et al. attempt to identify the false positives from raw test reports by adopting both a cluster-based

classification approach (Wang et al., 2016a) and active learning (Wang et al., 2016b). Similarly, our study aims to resolve crowdsourced testing problems. In this paper, we compare different classical algorithms to improve the efficiency of the test report classification.

6.2. Test report classification

The most relevant work of test report classification is detecting duplicate test reports. In a bug repository, some bug reports are marked as duplicates since such test reports are just similar to some other handled ones. Runeson et al. (2007) and Wang

Table 16
Training to testing ratio of Decision Tree (DT).

Application	Accuracy								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.308	0.400	0.415	0.422	0.436	0.498	0.521	0.641	0.641
Hujiang	0.309	0.329	0.414	0.443	0.445	0.467	0.531	0.575	0.635
Huawei	0.346	0.392	0.467	0.533	0.622	0.632	0.698	0.742	0.771
Tuniu	0.212	0.296	0.323	0.358	0.406	0.467	0.540	0.556	0.586
Game2048	0.369	0.395	0.455	0.464	0.454	0.487	0.500	0.506	0.553
Slife	0.482	0.573	0.577	0.595	0.600	0.626	0.673	0.716	0.737
Application	F-measure								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.289	0.365	0.386	0.406	0.418	0.461	0.495	0.567	0.669
Hujiang	0.309	0.330	0.385	0.424	0.427	0.462	0.536	0.582	0.617
Huawei	0.311	0.370	0.449	0.504	0.608	0.614	0.687	0.724	0.742
Tuniu	0.206	0.256	0.302	0.312	0.389	0.457	0.521	0.543	0.559
Game2048	0.339	0.364	0.423	0.435	0.423	0.454	0.461	0.509	0.527
Slife	0.484	0.539	0.556	0.576	0.577	0.596	0.655	0.689	0.713

Table 17
Training to testing ratio of Convolutional Neural Network (CNN).

Application	Accuracy								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.108	0.140	0.157	0.166	0.187	0.200	0.256	0.234	0.288
Hujiang	0.201	0.129	0.195	0.256	0.274	0.310	0.376	0.378	0.413
Huawei	0.133	0.165	0.349	0.404	0.444	0.576	0.589	0.623	0.635
Tuniu	0.098	0.135	0.116	0.102	0.094	0.142	0.177	0.238	0.278
Game2048	0.137	0.145	0.153	0.176	0.184	0.227	0.235	0.265	0.298
Slife	0.106	0.215	0.167	0.326	0.447	0.425	0.483	0.537	0.546
Application	F-measure								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.050	0.077	0.125	0.134	0.166	0.181	0.201	0.226	0.218
Hujiang	0.170	0.133	0.165	0.224	0.255	0.300	0.306	0.321	0.367
Huawei	0.113	0.133	0.310	0.378	0.435	0.531	0.565	0.627	0.634
Tuniu	0.056	0.129	0.081	0.078	0.090	0.103	0.117	0.144	0.187
Game2048	0.113	0.115	0.124	0.151	0.157	0.200	0.229	0.233	0.253
Slife	0.087	0.203	0.158	0.358	0.463	0.400	0.562	0.633	0.650

Table 18
Training to testing ratio of Support Vector Machine (SVM).

Application	Accuracy								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.523	0.583	0.645	0.690	0.703	0.711	0.718	0.730	0.748
Hujiang	0.365	0.402	0.532	0.583	0.627	0.638	0.673	0.683	0.699
Huawei	0.401	0.484	0.521	0.605	0.642	0.657	0.663	0.689	0.706
Tuniu	0.336	0.390	0.445	0.539	0.597	0.695	0.706	0.724	0.731
Game2048	0.306	0.407	0.490	0.506	0.523	0.547	0.569	0.542	0.547
Slife	0.457	0.523	0.614	0.686	0.675	0.741	0.775	0.794	0.789
Application	F-measure								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mylistening	0.538	0.595	0.634	0.669	0.694	0.704	0.734	0.726	0.745
Hujiang	0.380	0.399	0.512	0.562	0.600	0.619	0.649	0.661	0.688
Huawei	0.374	0.463	0.523	0.627	0.635	0.644	0.653	0.681	0.689
Tuniu	0.315	0.374	0.433	0.547	0.591	0.684	0.698	0.703	0.724
Game2048	0.312	0.385	0.486	0.500	0.503	0.521	0.543	0.522	0.545
Slife	0.454	0.491	0.583	0.665	0.691	0.723	0.735	0.765	0.782

et al. (2009) removed duplicate test reports based on supervised natural language processing approaches. Wang et al. (2016a) proposed a technique combining natural language and execution information to detect duplicate failure reports. However, they only removed duplicate reports and could not classify them into different categories, which means they could not provide more support for test report inspection.

Many studies also focus on test report classification. Zanetti et al. (2013) developed an automated classification scheme that can be integrated into bug tracking platforms. Tian et al. (2013) propose DRONE, a multi-factor analysis technique to classify the priority of test reports. Zhou et al. (2016) proposed a hybrid

approach by combining both text mining and data mining techniques of test report data to automate the classification process.

Our work is to classify test reports in crowdsourced testing, which is different from the aforementioned studies. Crowdsourced reports are noisier than bug reports because they are submitted by non-specialized crowd workers. Besides, crowd workers tend to submit as many test reports as possible under financial incentives. In this sense, classifying them is more valuable and challenging.

There were many studies to classify app reviews as bug reports, feature requests, user experiences, and rating (Maalej and

Table 19

The time cost (s) of feature extraction.

	Mylistening	Hujiang	Huawei	Tuniu	Game2048	Slife
TXT	25.614	12.726	14.726	9.816	12.019	40.817
IMG	146.313	169.723	117.132	286.213	108.113	862.103
TXT + IMG	172.362	185.193	134.139	301.019	121.293	100.123

Nabil, 2015), or as a feature request, problem discovery, information seeking, and information giving (Guzman et al., 2015; Panichella et al., 2015), which can help to deal with a large number of reviews. App reviews are often considered as test reports by users, who behave unprofessionally as crowd workers in our context. But crowd workers prefer to submit screenshots to describe a bug, and the screenshots can help to classify test reports. These related methods could not deal with the test report with screenshots hence cannot work in our context.

6.3. Application of image understanding on testing

Some studies focus on the application of Image Understanding on Testing. In Michail (2002), Michail et al. proposed a static approach, GUISearch, to guide search and browsing of its source code by using the GUI of an application. They further proposed a dynamic approach to obtain an explicit mapping from high-level actions to low-level implementation by identifying execution triggered by user actions and visually describing actions from a fragment of the application displayed (Chan et al., 2003). Kurlander and Feiner (1988) introduced the notion of an editable graphical history that can allow the user to review and modify the actions performed with a graphical interface. Similarly, Michail and Xie (2005) used before/after screenshots to visually describe application state at a very high level of abstraction to help users avoid bugs in GUI applications. However, images in these works are provided to developers or users directly without machine understanding.

Image-understanding techniques have been used in cross-browser issues for web applications. Cai et al. propose the VIPS algorithm (Cai et al., 2003), which segments a web page screenshot into visual blocks to infer the hierarchy from the visual layout, rather than from the DOM. Choudhary et al. (2010) proposed a tool called WEBDIFF to automatically identify cross-browser issues in web applications. Given a page to be analyzed, the comparison is performed by combining a structural analysis of the information in the page's DOM and a visual analysis of the pages'

appearance, obtained through screen captures. In recent years, some studies use image-understanding techniques for Android applications. In 2016, Feng et al. (2016) proposed an approach to test-report prioritization that utilizes a hybrid analysis technique, which is both text-based and image-based. This approach is a fully automatic diversity-based prioritization technique to assist the inspection of crowdsourced mobile application test reports. To further classify test reports into different categories, we compare different methods for classifying test reports using fused information (i.e., text and screenshots).

7. Discussion

In this paper, all crowdsourced test 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 words as tokens for representation learning. For common NLP libraries, such as NLTK (Loper and Bird, 2002), preprocessing techniques for different languages are provided.

We collected test reports of MyListening, Hujiang, Huawei store, Tuniu, Game 2048, and SLife from the industrial crowd-sourced testing platform. These applications are in different app categories from a famous Chinese company and have a large crowd of users, and they are indeed representative.

The experimental results may be affected by the quality of the test reports. Some noisy data may reduce the reliability of the results. In order to avoid this problem, we collect high-quality test reports by removing meaningless test reports and comments. We believe that this can help to decrease the adverse effects of the experimental results.

The applications we used to complete the experiment have different scales. There are 1346 test reports of Slife, 473 test reports of MyListening, 269 reports of Hujiang, 262 reports of Huawei, 531 reports of Tuniu, 210 reports of Game 2048. The accuracy and F-measure show the impact of data size. It can be seen that: 1. For almost all applications, accuracy and F-measure of test report classifier increase slightly in training data size, in addition to Game2048. 2. Sample size 350 may be enough for training test report classifier, and a larger training sample size may not be needed.

The construct validity of this study mainly questions the data processing method. We rely on the assessment attribute of crowdsourced reports stored in the repository to construct the

Table 20

The time cost (s) of model building, training, and classification.

		KNN	NB	RF	DT	CNN	SVM
Mylistening	Model building	0.002	0.003	0.002	0.004	50.254	0.002
	Training	0.002	0.003	0.003	0.004	63.863	0.003
	Classification	0.003	0.003	0.004	0.005	49.856	0.002
Hujiang	Model building	0.001	0.002	0.002	0.003	23.021	0.001
	Training	0.002	0.003	0.003	0.003	22.132	0.002
	Classification	0.002	0.004	0.003	0.003	14.791	0.001
Huawei	Model building	0.001	0.002	0.002	0.003	17.687	0.001
	Training	0.002	0.003	0.003	0.003	12.079	0.002
	Classification	0.002	0.004	0.003	0.003	12.917	0.001
Tuniu	Model building	0.001	0.002	0.002	0.003	13.687	0.001
	Training	0.002	0.001	0.003	0.002	16.658	0.002
	Classification	0.001	0.002	0.003	0.003	14.723	0.001
Game2048	Model building	0.001	0.002	0.001	0.002	15.923	0.001
	Training	0.002	0.001	0.003	0.004	14.239	0.002
	Classification	0.002	0.002	0.002	0.002	14.412	0.001
Slife	Model building	0.003	0.004	0.003	0.003	207.571	0.001
	Training	0.003	0.003	0.003	0.003	189.318	0.002
	Classification	0.003	0.004	0.004	0.003	79.231	0.001

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.

Automated parameter optimization may also affect test report classification results. Recent studies have focused on the effect of parameter setting on defect prediction. Different parameter settings can affect the result of defect prediction. Koru and Liu (2005), Mende (2010) and Mende and Koschke (2009) point out that selecting different parameter settings can impact the performance of defect models. The default parameters in the research toolkit are also not necessarily optimal (Jiang et al., 2008; Tosun and Bener, 2009). Tantithamthavorn et al. (2018) studied the impact of automated parameter optimization on defect prediction models. They conclude that automated parameter optimization can have a large impact on the performance stability, performance improvement, model interpretation, and ranking of defect prediction models. However, some classification techniques are not impacted by such optimization, such as random forest and support vector machines. In future work, we will consider the impact of parameter settings on the crowdsourced test report classification performance and try to use more parameter optimization tools or methods.

8. Conclusion

In this paper, we use the fusion of text information and screenshots to classify mobile application test reports from crowdsourced testing. To validate the effectiveness of the proposed methods, we conducted experiments on six industrial applications with more than 2500 reports from real-world crowdsourced testing. The experimental results show that image features play a supporting role in test report classification with SVM, which can increase the classification accuracy by about 10%. Besides, classifying test reports using SVM can outperform comparative methods. Furthermore, the larger the training set, the better performance the classification can achieve.

Further experiments are needed to gain a more specific understanding of image information. As we measure the similarity between images using SPM for test report classification, future work may rely on advanced image upstanding techniques, such as that can extract widgets from screenshots, to gain semantics GUI similarity. Another issue is to utilize a higher dimensional feature space as learned by Neural Network architectures to jointly embed image and text data.

CRedit authorship contribution statement

Yuying Li: Methodology, Software, Writing – original draft, Validation. **Yang Feng:** Conceptualization, Writing – review & editing. **Rui Hao:** Algorithm implementation. **Di Liu:** Experiment assistant. **Chunrong Fang:** Project administration. **Zhenyu Chen:** Project administration. **Baowen Xu:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by the National key research and development program of China (2018YFB1403400) and the National natural science foundation of China (61802171).

References

- Alenezi, M., Magel, K., Banitaan, S., 2013. Efficient bug triaging using text mining. *J. Softw.* 8, 2185–2190.
- Anvik, J., 2006. Automating bug report assignment. In: *International Conference on Software Engineering*.
- Anvik, J.K., 2007. *Assisting Bug Report Triage through Recommendation*. University of British Columbia.
- Anvik, J., Hiew, L., Murphy, G.C., 2006. Who should fix this bug? In: *Proceedings of the 28th International Conference on Software Engineering*. ACM, pp. 361–370.
- Cai, D., Yu, S., Wen, J.-R., Ma, W.-Y., 2003. Vips: A vision-based page segmentation algorithm. Microsoft Technical Report, MSR-TR-2003-79.
- Chan, K., Liang, Z.C.L., Michail, A., 2003. Design recovery of interactive graphical applications. In: *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, pp. 114–124.
- Chandrashekar, G., Sahin, F., 2014. A survey on feature selection methods. *Comput. Electr. Eng.* 40, 16–28.
- Che, W., Li, Z., Liu, T., 2010. LTP: A Chinese language technology platform. In: *Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations*. Association for Computational Linguistics, pp. 13–16.
- Chen, N., Kim, S., 2012. Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles. In: *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, pp. 140–149.
- Choudhary, S.R., Versee, H., Orso, A., 2010. Webdiff: Automated identification of cross-browser issues in web applications. In: *2010 IEEE International Conference on Software Maintenance*. IEEE, pp. 1–10.
- Collobert, R., Weston, J., 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In: *Proceedings of the 25th International Conference on Machine Learning*. ACM, pp. 160–167.
- Cunningham, P., Delany, S.J., 2007. K-nearest neighbour classifiers. *Mult. Classif. Syst.* 34, 1–17.
- Derhab, A., Aldweesh, A., Emam, A.Z., Khan, F.A., 2020. Intrusion detection system for Internet of Things based on temporal convolution neural network and efficient feature engineering. *Wirel. Commun. Mob. Comput.* 2020.
- Dodge, Y., Commenges, D., 2006. *The Oxford Dictionary of Statistical Terms*. Oxford University Press on Demand.
- Dolstra, E., Vliegendorst, R., Pouwelse, J., 2013. Crowdsourcing GUI tests. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, pp. 332–341.
- Feng, Y., Chen, Z., Jones, J.A., Fang, C., Xu, B., 2015. Test report prioritization to assist crowdsourced testing. In: *ESEC/SIGSOFT FSE*. pp. 225–236.
- Feng, Y., Jones, J.A., Chen, Z., Fang, C., 2016. Multi-objective test report prioritization using image understanding. In: *IEEE/ACM International Conference on Automated Software Engineering*. pp. 202–213.
- Gomide, V.H., Valle, P.A., Ferreira, J.O., Barbosa, J.R., Da Rocha, A.F., Barbosa, T., 2014. Affective crowdsourcing applied to usability testing. *Int. J. Comput. Sci. Inf. Technol.* 5, 575–579.
- Guzman, E., El-Haliby, M., Bruegge, B., 2015. Ensemble methods for app review classification: An approach for software evolution (N). In: *Automated Software Engineering, ASE, 2015 30th IEEE/ACM International Conference on*. IEEE, pp. 771–776.
- Hammouda, K.M., Kamel, M.S., 2004. Efficient phrase-based document indexing for web document clustering. *IEEE Trans. Knowl. Data Eng.* 1279–1296.
- Howe, J., 2006. The rise of crowdsourcing. *Wired Mag.* 14, 1–4.
- Jeong, G., Kim, S., Zimmermann, T., 2009. Improving bug triage with bug tossing graphs. In: *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM, pp. 111–120.
- Jiang, Y., Cukic, B., Ma, Y., 2008. Techniques for evaluating fault prediction models. *Empir. Softw. Eng.* 13, 561–595.
- Jiang, J.-Y., Liou, R.-J., Lee, S.-J., 2011. A fuzzy self-constructing feature clustering algorithm for text classification. *IEEE Trans. Knowl. Data Eng.* 23, 335–349.
- John, G.H., Langley, P., 1995. Estimating continuous distributions in Bayesian classifiers. In: *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., pp. 338–345.
- Joulin, A., Grave, E., Bojanowski, P., Mikolov, T., 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.
- Kibria, A.M., Frank, E., Pfahringer, B., Holmes, G., 2004. Multinomial naive bayes for text categorization revisited. In: *Australasian Joint Conference on Artificial Intelligence*. Springer, pp. 488–499.
- Kleinberg, E., et al., 1996. An overtraining-resistant stochastic modeling method for pattern recognition. *Ann. Statist.* 24, 2319–2349.
- Koru, A.G., Liu, H., 2005. An investigation of the effect of module size on defect prediction using static measures. In: *Proceedings of the 2005 Workshop on Predictor Models in Software Engineering*. pp. 1–5.
- Kurlander, D., Feiner, S., 1988. Editable graphical histories. In: *1988 IEEE Workshop on Visual Languages*. IEEE, pp. 127–134.

- Lazebnik, S., Schmid, C., Ponce, J., 2006. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR'06, vol. 2, pp. 2169–2178. <https://doi.org/10.1109/CVPR.2006.68>.
- LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324.
- Leung, K.M., 2007. Naive Bayesian classifier. Polytechnic University Department of Computer Science/Finance and Risk Engineering.
- Li, Y., Luo, C., Chung, S.M., 2008. Text clustering with feature selection by using statistical data. *IEEE Trans. Knowl. Data Eng.* 20, 641–652.
- Liaw, A., Wiener, M., et al., 2002. Classification and regression by randomforest. *R News* 2, 18–22.
- Loper, E., Bird, S., 2002. NLTK: The natural language toolkit. *arXiv preprint cs/0205028*.
- Maalej, W., Nabil, H., 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In: 2015 IEEE 23rd International Requirements Engineering Conference, RE. IEEE, pp. 116–125.
- Mao, K., Capra, L., Harman, M., Jia, Y., 2016. A survey of the use of crowdsourcing in software engineering. *J. Syst. Softw.* 126.
- Matter, D., Kuhn, A., Nierstras, O., 2009. Assigning bug reports using a vocabulary-based expertise model of developers. In: IEEE International Working Conference on Mining Software Repositories.
- McCallum, A., Nigam, K., et al., 1998. A comparison of event models for naive bayes text classification. In: AAAI-98 Workshop on Learning for Text Categorization. Citeseer, pp. 41–48.
- Mende, T., 2010. Replication of defect prediction studies: Problems, pitfalls and recommendations. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp. 1–10.
- Mende, T., Koschke, R., 2009. Revisiting the evaluation of defect prediction models. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering, pp. 1–10.
- Menzies, T., Marcus, A., 2015. Automated severity assessment of software defect reports. In: IEEE International Conference on Software Maintenance, pp. 346–355.
- Michail, A., 2002. Browsing and searching source code of applications written using a GUI framework. In: Proceedings of the 24th International Conference on Software Engineering. ACM, pp. 327–337.
- Michail, A., Xie, T., 2005. Helping users avoid bugs in GUI applications. In: Proceedings of the 27th International Conference on Software Engineering. ACM, pp. 107–116.
- Murphy, G., Cubranic, D., 2004. Automatic bug triage using text categorization. In: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. Citeseer.
- Pandey, N., Sanyal, D.K., Hudait, A., Sen, A., 2017. Automated classification of software issue reports using machine learning techniques: An empirical study. *Innov. Syst. Softw. Eng.* 13, 279–297.
- Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C.A., Canfora, G., Gall, H.C., 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In: Software Maintenance and Evolution, ICSME, 2015 IEEE International Conference on. IEEE, pp. 281–290.
- Phaisangittisagul, E., 2016. An analysis of the regularization between l2 and dropout in single hidden layer neural network. In: 2016 7th International Conference on Intelligent Systems, Modelling and Simulation, ISMS. IEEE, pp. 174–179.
- Quinlan, J.R., 2014. C4. 5: Programs for Machine Learning. Elsevier.
- Runeson, P., Alexandersson, M., Nyholm, O., 2007. Detection of duplicate defect reports using natural language processing. In: International Conference on Software Engineering, pp. 499–510.
- Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2018. The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Softw. Eng.* 45, 683–711.
- Thamaraiselvi, G., Kaliyammal, A., 2004. Data Mining: Concepts and Techniques. Ap Professional, pp. 1–18.
- Tian, Y., Lo, D., Sun, C., 2013. DRONE: Predicting priority of reported bugs by multi-factor analysis. In: IEEE International Conference on Software Maintenance, pp. 200–209.
- Tosun, A., Bener, A., 2009. Reducing false alarms in software defect prediction by decision threshold optimization. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 477–480.
- Wang, J., Cui, Q., Wang, Q., Wang, S., 2016a. Towards effectively test report classification to assist crowdsourced testing. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, p. 6.
- Wang, J., Wang, S., Cui, Q., Wang, Q., 2016b. Local-based active classification of test report to assist crowdsourced testing. In: Automated Software Engineering, ASE, 2016 31st IEEE/ACM International Conference on. IEEE, pp. 190–201.
- Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J., 2009. An approach to detecting duplicate bug reports using natural language and execution information. In: ACM/IEEE International Conference on Software Engineering, pp. 461–470.
- Xia, X., Lo, D., Ding, Y., Al-Kofahi, J.M., Nguyen, T.N., Wang, X., 2017. Improving automated bug triaging with specialized topic model. *IEEE Trans. Softw. Eng.* 43, 272–297.
- Xuan, J., Jiang, H., Ren, Z., Yan, J., Luo, Z., 2010. Automatic bug triage using semi-supervised text classification. In: International Conference on Software Engineering and Knowledge Engineering. IEEE.
- Xue, B., Zhang, M., Browne, W.N., Yao, X., 2015. A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evol. Comput.* 20, 606–626.
- Ying, X., 2019. An overview of overfitting and its solutions. *J. Phys. Conf. Ser.* 1168, 022022.
- Zanetti, M.S., Scholtes, I., Tessone, C.J., Schweitzer, F., 2013. Categorizing bugs with social networks: A case study on four open source software communities. In: International Conference on Software Engineering, pp. 1032–1041.
- Zhang, T., Chen, J., Luo, X., Li, T., 2017. Bug reports for desktop software and mobile apps in GitHub: What is the difference? *IEEE Softw.* PP, 1.
- Zhang, R., Zeng, Q., Feng, S., 2010. Data query using short domain question in natural language. In: Web Society, SWS, 2010 IEEE 2nd Symposium on. IEEE, pp. 351–354.
- Zhou, Y., Tong, Y., Gu, R., Gall, H., 2016. Combining text mining and data mining for bug report classification. *J. Softw. Evol. Process* 28, 150–176.

Yuying Li is a Ph.D candidate in the Software Institute, Nanjing University, under the supervision of Prof. Zhenyu Chen and Prof. Baowen Xu. Her research interest is crowdsourced software engineering.

Yang Feng is an assistant researcher at the department of Computer Science and Technology, Nanjing University. His research focuses on the areas of the program comprehension, testing, debugging, program analysis, and crowdsourced software engineering.

Rui Hao is a Ph.D candidate in the Software Institute, Nanjing University, under the supervision of Prof. Zhenyu Chen. Her research interest is crowdsourced software engineering and Android testing.

Di Liu is a Ph.D candidate at the department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Baowen Xu. His research interest is software testing and crowdsourced software engineering.

Chunrong Fang is an assistant researcher in the Software Institute, Nanjing University. His current research interests include software engineering, artificial intelligence, etc.

Zhenyu Chen received the bachelor and Ph.D. degrees in mathematics from Nanjing University. He is currently a professor in Software Institute, Nanjing University. His research interests focus on software analysis and testing. He has more than 80 publications at major venues including TOSEM, TSE, ICSE, FSE, ISSTA, ASE, ICST, etc. He has served as a PC co-chair of QRS 2016, QSIC 2013, AST 2013, IWPDP 2012, and a PC member of many international conferences. He has more than 30 patents and some have been used in Baidu, Alibaba, Huawei, etc. He is the founder of moctest.net.

Baowen Xu was born in 1961. He is a professor at the department of Computer Science and Technology, Nanjing University. His research areas are programming languages, software engineering, concurrent software and web software.