Domain Adaptation for Test Report Classification in Crowdsourced Testing

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

***Abstract*—In crowdsourced testing, it is beneficial to automatically classify the test reports that actually reveal a fault – *a true fault*, from the large number of test reports submitted by crowd workers. Most of the existing approaches toward this task simply leverage historical data to train a machine learning classifier and classify the new incoming reports. However, our observation on real industrial data reveals that projects under crowdsourced testing come from various domains, and the submitted reports usually contain different technical terms to describe the software behavior for each domain. The different data distribution across domains could significantly degrade the performance of classification models when utilized for cross-domain report classification.**

To build an effective cross-domain classification model, we leverage deep learning to discover the intermediate representation that is shared across domains, through the co-occurrence between domain-specific terms and domain-unaware terms. Specifically, we use the Stacked Denoising Autoencoders to automatically learn the high-level features from raw textual terms, and utilize these features for classification. Our evaluation on 58 commercial projects of 10 domains from one of the Chinese largest crowdsourced testing platforms shows that our approach can generate promising results, compared to three commonly-used and state-of-the-art baselines. Moreover, we also evaluate its usefulness using real-world case studies. The feedback from real-world testers demonstrates its practical value.

***Keywords*-Crowdsourced testing, test report classification, domain adaptation, deep learning**

I. Introduction

Crowdsourced testing is an emerging trend in both the software engineering community and industrial practice [1]–[4]. Crowd workers are required to submit test reports after performing testing tasks in a crowdsourced platform. Generally, they can submit thousands of test reports due to financial incentive and other motivations. However, these test reports often have many false positives, i.e., a test report supposed as *failed* by a worker that actually describes a correct behavior. Project managers or testers need to manually inspect these failed test reports to verify whether they actually reveal a fault — *a true fault*. Besides, only less than 50% of them are finally determined as true faults [1]–[4]. Hence, the process is time-consuming, tedious, and low-efficient. Therefore, it would be beneficial to automatically classify the *true fault* from the large amounts of test reports submitted by crowd workers.

*Corresponding author.

Our observation on real industrial data reveals that the projects under crowdsourced testing come from a large variety of domains, ranging from **travel**, **music**, to **safety** and **photo**. Different technical terms are used in the test reports of different domains to describe the software behavior. For instance, reports in **travel** domain would contain such terms as "location", "navigation", and "place", while reports in **music** domain would contain such terms as "play", "lyrics", and "song" (details are in Figure 2). Consequently, the textual features derived from these two domains are significantly different in their distributions. The different feature distribution across domains would degrade the performance of machine learning classifiers when utilized for cross-domain classification. This is because most machine learning models are designed under the assumption that training set and test set are drawn from the same data distribution [5].

Most of the existing approaches toward reports classification directly leveraged the textual features to build models and did not consider the different data distribution problem across domains [6]–[9]. This would result in the poor performance for cross-domain classification of crowdsourced reports. Several existing studies simply learned a specific classifier within each domain [10], [11]. The drawback is that usually there are not enough labeled training data for each domain, and obtaining labeled data is cost intensive. Our previous work [1] proposed to select similar data instances from training set to build the classifier. Although it can mitigate the distribution difference problem to some extent, it would fail to take effect when only a small number of similar instances can be found, which is quite common in practice.

To overcome the different data distribution problem and more effectively conduct cross-domain report classification, in this work, we propose Domain Adaptation Report claSsification (DARS). It leverages the Stacked Denoising Autoencoders (SDA), which is a powerful representation learning algorithm (also known as deep learning), to learn the high-level features and then utilize these features for classification. In order to abstract the high-level features, SDA discovers the intermediate representation from raw textual terms that is shared across domains. Putting it intuitively, the intermediate representation

is learned through the co-occurrence between the aforementioned domain-specific terms and domain-unaware terms (e.g., terms appeared across domains such as "button", "open", and "wrong").

We experimentally investigate the effectiveness and advantages of DARS on 25,564 test reports of 58 commercial projects in 10 domains from one of the Chinese largest crowdsourced testing platforms. Results show that DARS can achieve 0.77 F1 and 0.84 AUC on median. It significantly outperforms three commonly-used and state-of-the-art baseline report classification approaches. In addition, we conduct a case study and a survey with real-world testers, to further evaluate the usefulness of DARS. The feedback shows that 76% testers agree with the usefulness of DARS and would like to use it in real practice of crowdsourced testing.

This paper makes the following contributions:

- We propose the Domain Adaptation Report claSsification (DARS) approach for cross-domain crowdsourced reports, which can overcome the data distribution difference across domains.
- We evaluate our approach on 25,564 test reports from 58 commercial projects of 10 domains, which collected from one of the Chinese largest crowdsourced testing platforms, and results are promising.
- We evaluate the usefulness of DARS using real-world case studies, and discuss the lessons learned and challenges encountered when adopting our approach in practice.

The rest of this paper is organized as follows. Section II describes the background and motivation of this study. Section III presents the design of our proposed approach. Sections IV and V show the experimental setup and evaluation results respectively. Section VI provides a detailed discussion of the lessons learned and threats to validity. Section VII surveys related work. Finally, we summarize this paper in Section VIII.

II. Background and Motivation

A. Crowdsourced Testing

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

Our experiment is conducted with Baidu crowdsourced testing platform[1]. In general, the testers prepare testing tasks and distribute them on the crowdsourced testing platform. Then, the crowd workers can sign in to conduct the tasks and submit crowdsourced test reports after finishing the tasks. Table I demonstrates the attributes of a typical crowdsourced report. It contains operation steps, result description, screenshots, etc., as well as an assessment as to whether the worker believed that the software behaved correctly (i.e., passed), or behaved incorrectly (i.e., failed). For each test task, the platform also contains the name of the domain to which the related project belongs.

[1]Baidu (baidu.com) is the largest Chinese search service provider. Its crowdsourcing test platform (test.baidu.com) is also one of the largest crowdsourced testing platforms in China.

TABLE I: An example of crowdsourced test report

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

In order to attract more workers, testing tasks are often financially compensated, especially for these failed reports. Under this context, workers can submit thousands of test reports. Usually, this platform delivers approximately 100 projects per month, and receives more than 1,000 test reports per day on average. However, some of the test reports are false positives, i.e., a test report marked as *failed* that actually involves correct behavior or behavior outside of the studied software system.

Currently in this platform, testers need to manually inspect these failed test reports to judge whether they actually reveal a fault – *a true fault*. However, inspecting 1,000 reports manually could take almost half a week for a tester. Besides, only less than 50% of them are finally determined as true faults. Obviously, such process is time-consuming and low-efficient.

B. Stacked Denoising Autoencoders (SDA)

The Stacked Denoising Autoencoder (SDA) is an artificial neural network for unsupervised learning of effective representation [12], [13]. It consists of multiple layers of denosing autoencoders.

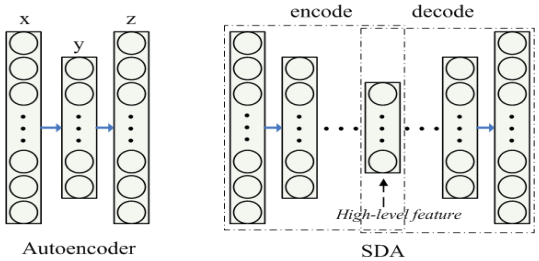

Fig. 1: Autoencoder and Stacked Denoising Autoencoders

An **autoencoder** [12] (shown in Figure 1) takes an input vector $x \in [0,1]^d$, and codes it to a hidden representation $y \in [0,1]^{d'}$ through a deterministic mapping $y = f_\theta(x) = s(Wx+b)$, parameterized by $\theta = W, b$. W is a $d' \times d$ weight matrix and b is a bias vector. The resulting latent representation y is then decoded to a "reconstructed" vector z in input space $z = g_{\theta'}(y) = s(W'y+b')$ with $\theta' = W', b'$. The weight matrix W' of the reverse mapping is constrained by $W' = W^T$. The parameters of this model are optimized to minimize the *average reconstruction error*:

$$argmin\frac{1}{n}\sum_{i=1}^{n} L(x(i), g_{\theta'}(f_{\theta(x(i))})) \quad (1)$$

where L is a loss function of *reconstruction cross-entry*. The autoencoder is trained by stochastic gradient descent, in which *number of training iterations* is an input parameter to balance the time cost, and error rate between the reconstructed vector and the input vector.

The **Denoising Autoencoder (DA)** [13] incorporates a slight modification to the autoencoder, i.e., corrupts the inputs before coding them into the hidden representation. It is trained to reconstruct (or denoise) the original input x from its corrupted version $\tilde{x}$ by minimizing $L(x, g(h(\tilde{x})))$.

A typical choice of corruption is binary masking noise. It sets a fraction of the features of each input to zero, in which the *level of noise* is an input parameter. This is natural for the textual features of crowdsourced test reports, where person-specific term preferences can influence the existence or absence of words.

The **Stacked Denoising Autoencoder (SDA)** [13] (shown in Figure 1) stacks a series of DAs together to build a deep architecture, by feeding the hidden representation of the t^{th} DA as input into the $(t+1)^{th}$ DA. The training is performed greedily, layer by layer. The latent representation in the last DA is treated as the high-level features. The *number of hidden layers* and *the number of nodes in each layer* are input parameters which can be set based on users' demand.

C. Motivation

Most of the existing test report classification approaches assume that the training set and test set are drawn from the same data distribution. This means that the performance would decline rapidly when using these approaches to classify the new coming test reports that have different distributions with the training set.

However, our observation on real industrial data reveals that crowdsourced reports across domains are under different distributions. The main reason is as follows. The projects under crowdsourced testing come from a large variety of domains, ranging from travel, music, to safety and photo. Moreover, different domains focus on different functional and technical aspects. Test reports from different domains usually use specific technical terms to describe the software behavior.

We present the term clouds of the textual descriptions of test reports for four randomly selected domains, i.e., *travel, music, safety*, and *photo,* to illustrate the distribution difference and its influence in Figure 2. We can easily observe that the technical terms exert significant differences among different domains. For instance, reports in the **travel** domain (the upper left subfigure of Figure 2) contain such terms as "location", "navigation", and "place", while reports in the **music** domain (the upper right subfigure of Figure 2) contain such terms as "play", "lyrics", and "song". Consequently, textual features derived from these two domains are significantly different in their distributions. Taken in this sense, models with textual features, built on the reports from **travel** domain, may fail to effectively classify the reports from **music** domain. To mitigate this problem, new features and approaches are required.

In Figure 2, we also notice that projects across domains share a certain amount of common terms. For example, many projects contain such behavioral terms like "display", "setup",

Fig. 2: Illustrative examples for the different data distribution of crowdsourced reports

and "download", or describable terms like "none", "wrong", and "missing". These shared terms could help bridge the gap across domains. More specifically, in **travel** domain, there are descriptions like "display location of building", while in **music** domain, descriptions like "display lyrics of song" are quite common. With the help of representation learning algorithm, such terms as "lyrics", "song" and term "location" would establish a relationship through the shared term "display". Based on the co-occurrence, the representation learning algorithm would map the raw terms to high-level features. The more similar context two terms share, the more similar their high-level features would be. The high-level features can then be used as input to build prediction models and predict the crowdsourced reports.

III. Approach

Figure 3 illustrates the overview of Domain Adaptation Report claSsification (DARS) approach. Generally speaking, DARS first trains the SDA to effectively encode the input features. Then the training set and test set are fed into the trained SDA to generate high-level features. The classification is then conducted based on the high-level features.

DARS consists of four major steps: 1) extract textual features, 2) train the SDA, and 3) leverage the SDA to generate high-level features based on textual features, and 4) build classifiers and conduct classification using the learned high-level features.

A. Extracting Textual Features

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

We first collect different sources of text descriptions together (input and operation steps, result description). Then we conduct **word segmentation**, as the crowdsourced reports in our experiment are written in Chinese. We adopt ICTCLAS[2] for word segmentation, and segment descriptions into words. We then remove **stopwords** (i.e., "on", "the", etc.) to reduce noise. Because workers often use different words to express the same concept, we introduce the **synonym replacement**

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

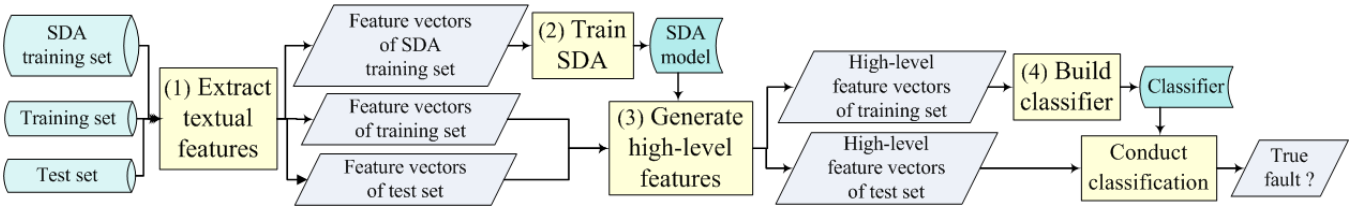

Fig. 3: Overview of DARS

technique to mitigate this problem. Synonym library of LTP[3] is adopted.

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

B. Training SDA

To generate the high-level features for classifying crowdsourced reports, we need to first train the SDA using the SDA training set. Existing researches showed that the larger size of SDA training set, the better performance can be achieved [14]–[16]. Moreover, the representation learning technique does not involve the class information (i.e., whether the report is a true fault or not). Hence, common practice would utilize all available data instances in the training process [14]–[16]. To train a SDA is to determine the weights w and the biases b, so that the trained SDA can effectively encoder the input features.

As mentioned in Section II-B, to train an effective SDA for generating the high-level features, we need to tune four parameters, which are: 1) *the number of hidden layers*, 2) *the number of nodes in each hidden layer*, 3) *the level of noise*, and 4) *the number of training iterations*. Existing work that leveraged the SDA to generate features for natural language processing and image recognition [13] [14] reported that the performance of the SDA-generated features is sensitive to these parameters. We show how we tune these parameters in Section IV-E.

The nodes in the input and output layers are equal to the size of feature vector. For other layers, we set the number of nodes to be the same to simplify our model. Besides, the SDA requires the input vectors be the same length. To use the SDA to generate high-level features, we first collect all the features appeared in the SDA training set and organize them into a joint feature vector. Then we transform the original feature vectors to the the joint feature vectors. In detail, for a feature contained in the original feature vector, we use its original value as the value in the joint feature vector. Otherwise, we set its value to zero. Adding zeros does not affect the results, because it simply means that the specific term does not appear in the report.

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

Note that, the SDA requires the values of input data ranging from 0 to 1. As the input vector represents the term frequency of each feature, its value can be larger than 1. To satisfy this requirement, we normalize the values in the feature vectors of the SDA training set using min-max normalization [17].

C. Generating High-level Features

After we have trained a SDA, both the weights w and the biases b (details are in Section II-B) are fixed. For the feature vectors of training set and test set, we first map them into the joint feature vector as described in Section III-B. Then we use min-max normalization to normalize them into the range between 0 and 1.

The normalized joint feature vectors of the training set and test set are fed into the SDA respectively. The representations in last hidden layer of SDA are then treated as the high-level features.

D. Building a Classifier

After we obtain the generated high-level features for each crowdsourced report in both the training set and test set, we build a machine learning classifier based on the training set. Then we use the test set to evaluate the performance of the built classifier. The details of how we choose the training set and the test set are shown in Section IV-C.

To better assist the manual inspection (details are in Section V-C), the classifier will provide the probability for the reports being true faults. The bigger the probability value, the more likely the report contains a true fault. To compute the F1 (details are in Section IV-D), we use 0.5 as cutoff value, denoting that reports with a probability value larger than 0.5 are treated as true faults, and vice versa.

IV. Experiment Setup

A. Research Questions

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

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

We investigate the performance of DARS in classifying crowdsourced reports under different experimental settings (details are in Section IV-C).

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

To demonstrate the advantages of DARS, we compare its performance with three baseline methods (details are in Section IV-C).

TABLE II: Projects under investigation

Domain	# Project	# Report	# Failed report	# True fault	%True fault
Efficiency	9	4237	2198	884	40.2%
Entertainment	8	6423	4913	2700	54.9%
Music	4	2160	1497	450	30.0%
News	6	2740	2517	1112	44.1%
Photo	4	2207	1498	564	37.6%
Read	7	2042	3136	875	27.9%
Safety	3	2216	1916	762	39.7%
Shopping	3	2280	1836	842	45.8%
Tool	10	5320	4409	1628	36.9%
Travel	4	1883	1644	529	32.1%
Summary	58	31,508	25,564	10,346	40.4%
C1 (Shopping)	1	231	177	87	49.1%
C2 (Tool)	1	690	455	182	40.0%
C3 (Efficiency)	1	1428	1004	392	39.0%

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

We conduct a case study and a survey in Baidu crowdsourced testing group to further evaluate the usefulness of our approach.

B. Data Collection

Our experiment is based on crowdsourced reports from the repositories of Baidu crowdsourced testing platform. We collect all crowdsourced testing projects closed between Oct. 1st 2015 and Oct. 31st 2015. We group these projects into domains based on the domain name recorded in the platform (details are in Section II-A). Table II provides details of these domains with the number of projects, the number of submitted reports, the number of reported *failed* reports, and the number and the ratio of *true faults* in failed reports.

Note that, our classification is conducted on *failed* reports, not the complete set. We exclude the *passed* reports because of the following reason. As we mentioned, failed reports can usually involve both correct behaviors and true faults. However, through talking with testers in the company, we find that almost none of the passed reports involve true faults. This maybe because that the compensation favors the faults, so the crowd workers are very unlikely to miss the faults.

Additionally, we randomly collect three other projects (belonged to three domains) closed in March 10th 2016 to conduct the case study in Section V-C.

The assessment attribute (see Table I) of each report can be treated as the groundtruth label of classification. For the reports which have the assessment attribute (only 2106 reports), we simply use the stored value as the label of classification. For other reports, two testers in the company were asked to assign the assessment label for each report. After their separate labeling, we analyzed the differences amongst their labeling, and conducted follow-up interviews until common consensus was reached. One may argue that the testers can also produce false positives during the labeling process, which would impact the results of our experiments. The well-controlled labeling process, with two testers and follow-up interviews, can help alleviate this problem. More importantly, the testers in the company are more experienced than crowd workers, and they did this without financial compensation on faults. Therefore, we believe the groundtruth labels are relatively trustworthy.

C. Experimental Setup and Baselines

As we mentioned in Section III-B, training a SDA does not involve the class information. Following previous studies [14], [15], we treat all the experimental crowdsouced reports as *SDA training set*, denoting that we use these data to extract the *feature vectors of SDA training set* and train the SDA model (details are in Figure 3). For the 10 domains under investigation, we use all crowdsourced reports of one domain as the *test set* and extract the *feature vectors of test set*. We randomly select K number of crowdsourced reports from the other nine domains to act as the *training set* and extract the *feature vectors of training set*. K is set as 100, 200, 500, 1,000, 1,500, 2,000, 3,000, and 5,000 respectively to investigate the influence of training set size on the model performance. The experiment for each K is repeated 50 times to ensure the stability of the results.

To further explore the performance of our proposed approach, we compare DARS with three typical baseline approaches.

Domain-unaware classification (DUC): It is the most straightforward prediction approach and does not consider the distribution difference among different domains. It builds the machine learning classifiers using all the crowdsourced reports in the training set and conducts classification on the test set.

Transfer component analysis (TCA+) [5]: It is the state-of-the-art technique for domain adaptation. TCA aims to find a latent feature space for both the training set and test set by minimizing the distance between the data distributions while preserving the original data properties. TCA+ extends TCA with automatically normalization, and can yield better performance than TCA.

Cluster-based classification approach (CURES) [1]: It is the state-of-the-art technique to classify crowdsourced reports which can also mitigate the difference of data distribution in crowdsourced reports. It first clusters similar reports of training set together, and builds classifiers based on the reports of each cluster. Then it selects the most similar clusters with the test set, and conducts classification.

Both DARS and these baselines involve utilizing different machine learning classification algorithms to build classifiers. In this work, we experiment with Linear Regression (LR) [18], which is widely reported as effective in many different classification tasks in software engineering [1], [5], [19].

D. Evaluation Metric

To evaluate our proposed approach, we use two metrics: ***F1*** and ***AUC***.

F1 is a widely adopted metric to evaluate issue report classification techniques [6]–[8]. It is the harmonic mean of precision and recall of classifying true fault [17]. AUC is the most popular and widely used metric for evaluating classification performance on imbalanced data [17] . As the dataset of crowdsourced reports is usually imbalanced (i.e., less true faults), we specifically use this metric. It is the area under ROC curve[4], which measures the overall discrimination ability of a

[4]The ROC curve is created by plotting the true positive rate.

classifier. The AUC for a perfect model would be 1, and for a model predicting all instances as true or false would be 0.

E. Parameter Settings for Training a SDA Model

Many SDA applications [13]–[15] report that an effective SDA needs well-tuned parameters, i.e., 1) the number of hidden layers, 2) the number of nodes in each hidden layer, 3) the level of noise, and 4) the number of training iterations. In this study, since we leverage the SDA to generate the high-level features, we need to consider the impact of the four parameters. We tune these parameters by conducting experiments with different values of the parameters on our experimental data.

We use all the crowdsourced reports to train the SDA with respect to the specific values of the four parameters. Then, we use the trained SDA to generate high-level features for all the crowdsourced reports. After that, we use the original labelled reports (Section IV-B) to build a classifier and apply it to the unlabeled reports. Lastly, we evaluate the specific values of the parameters by the AUC score.

1) Setting the number of hidden layers and the number of nodes in each layer: Since the number of hidden layers and the number of nodes in each hidden layer interact with each other, we tune these two parameters together. For the number of hidden layers, we experiment with 8 discrete values including 1, 2, 3, 4, 5, 10, 20, and 50. For the number of nodes in each hidden layer, we experiment with 8 discrete values including 20, 50, 100, 200, 300, 500, 800, and 1,000. When we evaluate these two parameters, we set the level of noise to 0.1 and number of training iterations to 200, and keep it constant.

Figure 4 illustrates the AUC for tuning the number of hidden layers and the number of nodes in each hidden layer together. When the number of nodes in each layer is fixed, with the increase number of hidden layers, the AUC are convex curves. Most curves peak at the point where the number of hidden layers is 3. If the number of hidden layers remains unchanged, the best AUC happens when the number of nodes in each layer is 200 (the top line). As a result, we choose the number of hidden layers as 3 and the number of nodes in each hidden layer as 200. Thus, the number of high-level features is 200.

2) Setting the level of noise: The level of noise is also an important parameter for the SDA. We experiment with 7 discrete values ranging from 0 to 0.5. Noise level of 0 denotes the input features remain unchanged, while level of 0.5 denotes that 50% of original features would be randomly set to zero. Figure 4 shows the AUC for tuning this parameter. We can see that training the model without noise cannot reach the best performance. Instead, the best performance is achieved when the level of noise is 0.1, denoting that 10% of original features would be randomly set as zero. Therefore, we choose the level of noise as 0.1.

3) Setting the number of iterations: The number of iterations is another important parameter for building an effective SDA. During the training process, the SDA adjusts weights to narrow down the error rate between reconstructed input data and original input data in each iteration. In general, the bigger the number of iterations, the lower the error rate. However, there is a trade-off between the number of iterations and the time cost. To balance the number of iterations and the time cost, we conduct experiments with 9 discrete values, ranging from 1 to 5000. We use error rate to evaluate this parameter. Figure 4 demonstrates that, as the number of iterations increase, the error rate decreases slowly with the corresponding time cost increases exponentially. In this study, we set the number of iterations to 200, with the error rate is about 0.09 and the time cost is about 23 minutes. Note that, we only need to conduct the SDA training process once, then we can utilize the trained SDA to generate the high-level features for different training set and test set. This is why we suppose the time for training SDA (23 minutes) is acceptable.

V. Results and Analysis

A. Answering RQ1 (Effectiveness)

Figure 5 demonstrates the F1 and AUC of DARS for the 50 experiments of all experimental domains, under different training set size. We can see that with the increase of training set size, both the F1 and AUC would first improve and then remain almost unchanged.

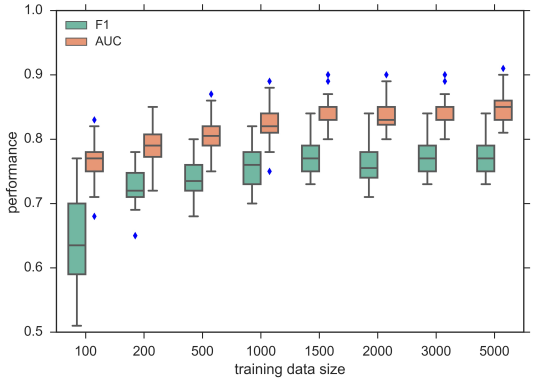

Fig. 5: The effectiveness of DARS (RQ1)

We then conduct Mann-Whitney Test for the performance of classification under each adjacent training set sizes. Results reveal that when the training set size is smaller than 1,500, the p-value of both F1 and AUC for each adjacent train sets are less than 0.05. When the train set size is larger than 1,500, the p-value of both F1 and AUC for each adjacent training sets are more than 0.05. This illustrates that a training set with less than 1,500 instances would decrease the performance, and a training set with more than 1,500 instances cannot increase the performance significantly. Thus, 1,500 is the relative optimal train set size.

We focus on the performance with the training set size as 1,500. The F1 ranges from 0.72 to 0.82, with the median F1 as 0.77. The AUC ranges from 0.80 to 0.90, with the median AUC as 0.84.

This implies that DARS merely needs 1,500 labeled data instances for achieving relatively satisfactory performance. It benefits from the learned high-level features, which can be learned based on the unlabeled data. There are usually large amounts of unlabeled data and they are relatively easy to collect. Our approach provides a way to effectively utilize these unlabeled data.

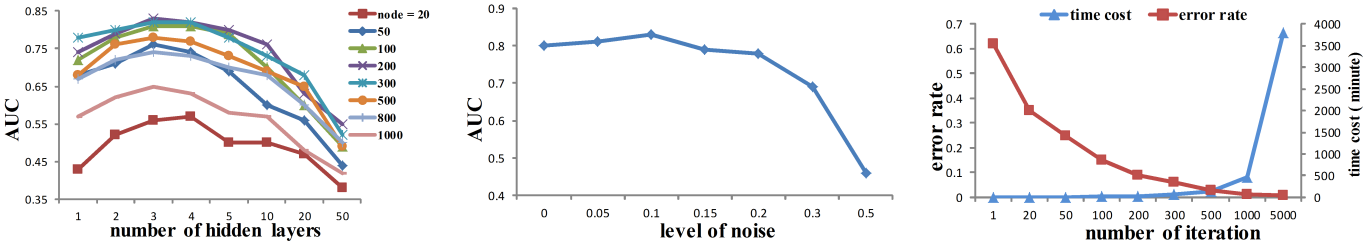

Fig. 4: Performance with different parameters

TABLE III: Performance for each test domain (RQ1)

Domain	F1			AUC		
	min	max	median	min	max	median
Efficiency	0.741	0.761	0.752	0.825	0.835	0.831
Entertainment	0.751	0.820	0.785	0.824	0.859	0.840
Music	0.720	0.747	0.735	0.798	0.813	0.809
News	0.814	0.846	0.832	0.877	0.907	0.896
Photo	0.727	0.763	0.740	0.806	0.823	0.814
Read	0.761	0.795	0.780	0.847	0.859	0.854
Safety	0.709	0.730	0.713	0.783	0.799	0.794
Shopping	0.762	0.816	0.802	0.840	0.862	0.853
Tool	0.726	0.758	0.739	0.802	0.815	0.810
Travel	0.700	0.811	0.758	0.781	0.845	0.819

Table III demonstrates the classification performance for each domain acting as test set, with training set size as 1,500. We can easily observe that the performance for different domains might vary to some extent. The median F1 ranges from 0.71 to 0.83 for different domains, while the AUC varies from 0.79 to 0.89.

The worst performance occurs in such domains as safety and music. We further analyze the underlying reason. There are large amounts of new terms appearing in the crowdsourced reports of these two domains, e.g., terms for song name. These terms would bring noise when establishing the intermediate representation.

B. Answering RQ2 (Advantage)

Table IV illustrates the performance of DARS and three baselines. Due to space limit, we only present 4 of the 8 training set sizes (shown in Figure 5) with intervals. As we mentioned in Section IV-C, we randomly choose the crowdsourced reports to act as the training set and repeat 50 times for each training set size. We present the minimum, maximum, and median performance of the random experiments for these methods. The value with dark background denotes the best F1 or AUC for each training set size.

At first glance, we can find that DARS can achieve the highest median F1 and AUC with the smallest variances, for every training set size. We also conducted Mann-Whitney Test for both F1 and AUC between DARS and each baseline. The p-value for all the tests are less than 0.05. This further illustrates the effectiveness and advantages of our approach.

Among the three baselines, the performance of DUC is the worst, denoting that classifiers without considering the data distribution difference across domains would result in quite bad performance.

The performance of TCA+ is worse than DARS in both F1 and AUC for all the training set sizes. It is reasonable

TABLE IV: Comparison of performance with baselines (RQ2)

		F1				AUC			
		100	500	1500	3000	100	500	1500	3000
Min	**DARS**	0.507	0.682	0.731	0.728	0.684	0.747	0.800	0.809
	DUC	0.122	0.166	0.222	0.248	0.298	0.318	0.402	0.382
	TCA+	0.272	0.305	0.345	0.502	0.462	0.502	0.498	0.442
	CURES	0.305	0.355	0.435	0.502	0.474	0.512	0.603	0.635
Max	**DARS**	0.765	0.801	0.837	0.837	0.825	0.869	0.901	0.907
	DUC	0.625	0.667	0.717	0.783	0.783	0.793	0.793	0.808
	TCA+	0.770	0.724	0.754	0.781	0.797	0.811	0.802	0.805
	CURES	0.661	0.711	0.803	0.801	0.688	0.721	0.799	0.831
Median	**DARS**	0.636	0.738	0.773	0.771	0.767	0.802	0.831	0.834
	DUC	0.356	0.398	0.441	0.456	0.485	0.501	0.572	0.582
	TCA+	0.526	0.559	0.602	0.627	0.647	0.670	0.682	0.698
	CURES	0.504	0.543	0.638	0.646	0.626	0.663	0.704	0.736
Var.	**DARS**	0.075	0.029	0.031	0.031	0.031	0.028	0.026	0.026
	DUC	0.113	0.121	0.109	0.086	0.100	0.112	0.071	0.074
	TCA+	0.114	0.115	0.114	0.060	0.097	0.085	0.079	0.082
	CURES	0.086	0.087	0.070	0.094	0.058	0.056	0.051	0.062

because existing domain adaption methods (i.e., TCA+) generate the new features based on linear projections of raw features. However, deep learning techniques (e.g., SDA) could learn the high-level features from the non-linear mapping of the raw features, thus can encode complex data variations. Furthermore, SDA has been proven to be more effective than existing domain adaptation methods in other tasks [14], [15].

Our approach also outperforms CURES, which is the state-of-the-art technique to classify the crowdsourced reports. This is because CURES relies on the historical similar reports to construct the classifier. When the training set is not large enough, CURES's performance would be degraded.

C. Answering RQ3 (Usefulness)

To further assess the usefulness of DARS, we conduct a case study and a survey in Baidu. We randomly select three projects for our case study (details are in Table II). Six testers from the crowdsourced testing group are involved. We divide them into two groups according to their experience, with details summarized in Table V.

TABLE V: Participant of case study (RQ3)

Group A		Group B
A1	**2-5 years' experience in testing**	B1
A2	**1-2 years' experience in testing**	B2
A3	**0-1 years' experience in testing**	B3

The goal of this case study is to evaluate the usefulness of DARS in classifying *true faults* from the crowdsourced test reports. Firstly, we utilize the trained SDA to generate the high-level features for all the reports of the three experimental projects. We then use the originally labeled reports (Section IV-B) to build a classifier. Through conducting classification for the three projects, we obtain the probability of each report being a true fault.

For both groups, we ask the practitioners to label each report with "yes" or "no", denoting whether the report involves a true fault. For practitioners in Group B, we only present them the crowdsourced test reports under classification. For practitioners in Group A, besides the crowdsourced reports, we provide them with the predicted probability for each report being a true fault. More than that, the reports are displayed by the probability values in descending order.

To build the ground truth, we gather all the classification outcomes from the practitioner. Follow-up interviews are conducted to discuss the differences among them. Common consensus is reached on all the difference and a final edition of classification is used as the ground truth (details are in Table II).

TABLE VI: Results of case study (RQ3)

	Results from Group A			Results from Group B		
Project	**C1**	**C2**	**C3**	**C1**	**C2**	**C3**
Precision	0.90,0.94	0.80,0.86	0.76,0.88	0.90,0.94	0.86,0.90	0.80,0.90
Recall	0.96,1.00	0.94,0.98	0.91,0.98	0.96,0.99	0.93,0.99	0.90,0.96
F1	0.93,0.97	0.86,0.91	0.82,0.92	0.93,0.97	0.89,0.94	0.84,0.92
Time(min)	40,52	80,112	128,164	72,86	186,232	360,394

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

We mentioned that we only require the practitioners to assign the label of "yes" or "'no", because the probability of being true faults is complex to measure by manual. Hence, we do not present the AUC which requires the probability to compute. Instead, besides F1, we present the precision, recall, and the time taken for the classification in Table VI.

The classification assisted with the probability provided by DARS (Group A) can find as many true faults as the classification without assistance (Group B), with far less time. In particular, with the increase of project size, the consumed time of manual classification without assistance can dramatically increase, while its accuracy (F1) does not show the fundamental difference.

In addition, we design a questionnaire and conduct a survey to ask testers about the usefulness of DARS. The questionnaire first demonstrates a short description about DARS, what DARS can provide for the classification and the summarized evaluation results on 58 projects. Then it asks three questions shown in Table VII. We provide five options for the first two questions, and allow respondents freely express their opinion for the third question.

We send invitation emails to the testers who are involved in the report classification in Baidu crowdsourced testing group. We totally receive 21 responses out of 47 requests.

As indicated in Table VII, of all 21 respondents, 16 of them (76%) agree that DARS is useful for report classification and they would like to use it. This means testers agree the usefulness of DARS in general. Only 2 hold conservation options and 3 disagree. When it comes to the reason for disagreement, they mainly worry about its flexibility on new projects, the unsatisfactory recall, as well as the effort still needed to take. In addition, the project manager shows great interest in DARS, and is arranging to deploy it on their platform to assist the classification process.

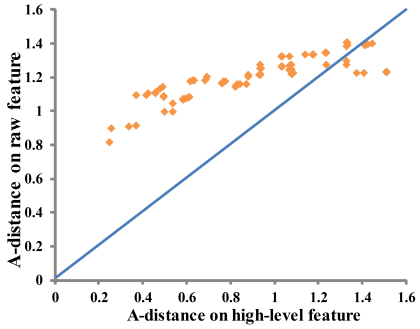

Fig. 6: A-distance for every domain pair

VI. Discussion

A. Why Does it Work?

We have mentioned that the major challenge for crowdsourced reports classification is the different data distribution of reports across domains. Simply utilizing the reports in other domains to build the classifier can easily result in low accuracy.

We suppose the representation learning algorithm could make the feature distributions across domains more similar, thus reduce the noise introduced by the domain-specific information. To intuitively demonstrate it, we examine the A-distance [20] between the feature distributions of each pair of domains. A-distance is a measure of similarity between the probability distribution of two datasets. We hypothesize that it should be more difficult to discriminate between different domains after the representation learning, which implies the effectiveness of domain adaptation. This can be illustrated by more similar feature distributions. Common practice for obtaining A-distance would compute the generalization error ϵ of a classifier trained to discriminate between two domains [20]. Then A-distance is measured as $2(1-2\epsilon)$.

Figure 6 reports the A-distance for raw features and high-level features for every domain pair. As expected, A-distance is decreased for high-level features, which implies that the feature distributions between domains become more similar after the representation learning. This is why the classification based on high-level features can achieve higher performance.

B. Lessons Learned

Based on the aforementioned studies and an un-structured interview, which is conducted with the test manager of Baidu and involved practitioners of the case study, the following lessons can be learned.

1) Testers care false negative more than false positive: In terms of the classification performance, we found that the testers care more about false negatives (i.e., reports with true faults that are misclassified as reports without faults) than false positives (i.e., reports without faults that are misclassified as reports with faults). In other words, a higher recall is expected, even at the expense of lower precision. This is rational because missing a true fault might introduce serious quality issues in the future, and require extra budget to debug and fix it.

Moreover, for the reports with quite high or low predicted probability as being true faults, the practitioners tend to classify them consistent with the provided probability. Specifically,

TABLE VII: Results of survey (RQ3)

Questions	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Total
Q1. Do you think DARS is useful to help classify "true fault" from crowdsourced test report?	0	2	1	5	13	21
Q2. Would you like to use the probability provided by DARS to help with the classification task?	0	3	2	3	13	21
If Disagree for either of the question, please give the reason.	Still need much effort; Worry the accuracy on other projects; The recall is unsatisfying;					3

by observing the manually labelling process of testers, we found that in most cases the testers will label a test report as a true fault without any consideration, if its probability is large enough (e.g., larger than 0.8). That is to say, for the reports which are actually not faults but have high probability, the practitioners would just label them as true faults. Similarly, for the faults which are assigned with quite low probability, there would be hardly any chance for them to be classified correctly. This is the root cause of low performance in Table VI. The testers also mentioned that, it would be preferred if an approach can help remove a portion of reports which are definitely not faults. To address this challenge, we would like to explore other approaches to help filter this kind of reports.

2) Providing the probability for a report is more actionable and scalable: The testers also mentioned that the prediction outcomes should be actionable to make DARS more practical. They supposed that tagging each report with its probability is a good choice, as this kind of information suggested them the uncertainty of the prediction. If an automated approach labels each report as binary category, i.e., whether it is a true fault, it would be hard to decide which reports need more human inspection.

Furthermore, they thought that the results in the form of probability values are more scalable than results with binary category. Users can set up the cutoff point, e.g., reports with probability larger than 0.7 are treated as true faults, according to their demands when utilizing these probability values for classification. In this way, based on the same set of prediction results, the testers can customize different kinds of manual classification designs considering the planned time and effort.

3) Testers need some evidences to prove the effectiveness of DARS: We have attached the summarized evaluation results of DARS on the questionnaire. The testers mentioned that these results played an important role in convincing them that DARS is useful and they would like to use it. They supposed it would be better to provide some video materials to demonstrate how the probability can be utilized in the manual classification process.

4) DARS might also be useful to crowd workers : The tester manager also suggested that it would be useful to show the prediction results to the crowd workers before they submit reports. If their reports suffered from low predicted probability, the crowd workers might choose to conduct the testing again or not to submit the report. Both practices can help improve the quality of test reports and the efficiency of crowdsourced testing.

C. Threats to Validity

The external threats concern the generality of this study. Firstly, our dataset consists of 58 projects covering 10 domains, collecting from one of the Chinese largest crowdsourced testing platforms. The various domains of projects and the size of data relatively reduce this threats. Secondly, all crowdsourced reports investigated in this study are written in Chinese, and we cannot assure that similar results can be observed on crowdsourced projects in other languages. But this is alleviated as we did not conduct semantic comprehension, but rather simply tokenize sentence and use words as tokens for representation learning.

Regarding internal threats, rather than experimentally investigate the influence of the SDA training set on the classification performance, we simply treat all the available data instances as the SDA training set. This is supported by the findings in several previous studies [14]–[16]. Moreover, the experiment outcomes have proven its effectiveness. Anyhow, we will conduct well-designed experiments to further examine the influence.

Construct validity of this study mainly questions the data processing method. Since only a small portion of reports have the assessment attribute, we rely on human labeling to help construct the ground truth. However, this is addressed to some extent due to the fact that two testers were involved in the labeling process and we have conducted follow-up interviews to resolve the difference amongst the labeling process.

VII. Related Work

a) Crowdsourced Testing: Crowdsourced testing has been applied to generate test cases [21], measure real-world performance of software products [22], help usability testing [23], as well as detect and reproduce context-related bugs [24]. All the studies above use crowdsourced testing to solve the problems in traditional software testing activities. However, our approach is to solve the new encountered problem in crowdsourced testing.

Some other studies focus on solving the new encountered problem in crowdsourced testing. Feng et al. [2], [4] proposed test report prioritization methods for use in crowdsourced testing. They designed strategies to dynamically select the most risky and diversified test report for inspection in each iteration. Our previous work [1], [3] proposed to mitigate the distribution difference problem in crowdsourced report classification, through selecting similar data instances from training set to build the classifier. However, its performance would decline rapidly when there are only a small number of similar instances, which is quite common in practice.

b) Issue Reports Classification: Previous researches have proposed to automatically classify issue reports into different priority levels [6], distinguish duplicate reports [7], conduct bug triage [9], classify valid bug reports [8], etc.

Our work focuses on classifying test reports in crowdsourced testing, which is different from the aforementioned studies in two ways. Firstly, crowdsourced reports are more noise than issue reports, as they are submitted by non-specialized crowd workers under financial incentives. In this sense, classifying them is more valuable, yet possesses more challenges. Secondly, crowdsourced test reports across domains demonstrate significant difference in data distribution. Previous approaches cannot achieve satisfactory performance for the cross-domain crowdsourced report classification.

c) Deep Learning in Software Engineering: Recently, deep learning algorithms have been adopted to improve research tasks in software engineering. Yang et al. [25] proposed an approach that leveraged deep learning to generate features from existing features and then used these new features to predict whether a commit is buggy or not. Wang et al. [26] then proposed to apply deep belief network to learn semantic features from source code to improve both within-project and cross-project defect prediction performance. Their results show that the automatically learned semantic features can improve the performance of defect prediction. Lam et al. [27] combined deep learning algorithms and information retrieval techniques to improve fault localization. Raychev et al. [28] and White et al. [29], [30] leveraged deep learning to model program languages for code suggestion and code clone detection. Gu et al. [31] proposed a new approach to learn API usage by using deep learning.

Different with the above researches, in our work, we explore the application of deep learning techniques to overcome the distribution difference problem in crowdsourced report classification.

VIII. Conclusion

This paper proposes Domain Adaptation Report claSsification (DARS) approach to overcome the data distribution difference across domains in crowdsourced reports classification. We evaluate DARS from the standpoints of effectiveness, advantage, and usefulness in one of the Chinese largest crowdsourced testing platforms, and results are promising.

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

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under grant No.61602450, No.61432001, No.91318301, and No.91218302. We would like to thank the testers in Baidu for their great efforts in supporting this work.

References

[1] J. Wang, Q. Cui, Q. Wang, and S. Wang, "Towards effectively test report classification to assist crowdsourced testing," in *ESEM '16*, 2016, pp. 6:1–6:10.

[2] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu, "Test report prioritization to assist crowdsourced testing," in *FSE '15*, pp. 225–236.

[3] J. Wang, S. Wang, Q. Cui, and Q. Wang, "Local-based active classification of test report to assist crowdsourced testing," in *ASE '16*, 2016, pp. 190–201.

[4] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective test report prioritization using image understanding," in *ASE '16*, 2016, pp. 202–213.

[5] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *ICSE '13*, 2013, pp. 382–391.

[6] Y. Tian, D. Lo, and C. Sun, "Drone: Predicting priority of reported bugs by multi-factor analysis," in *ICSM '13*, pp. 200–209.

[7] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *ICSE '08*, pp. 461–470.

[8] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: A case study on four open source software communities," in *ICSE '13*, pp. 1032–1041.

[9] S. Wang, W. Zhang, and Q. Wang, "FixerCache: Unsupervised caching active developers for diverse bug triage," in *ESEM '14*, pp. 25:1–25:10.

[10] E. Guzman, M. El-Halaby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution," in *ASE '15*, 2015, pp. 771–776.

[11] S. Panichella, A. D. Sorbo, E. Guzman, C. A.Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? Classifying user reviews for software maintenance and evolution," in *ICSM '15*, pp. 281–290.

[12] Y. Bengio, "Learning deep architectures for AI," *Journal Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[13] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *ICML '08*, 2008, pp. 1096–1103.

[14] M. Chen, K. Q. Weinberger, Z. Xu, and F. Sha, "Marginalizing stacked linear denoising autoencoders," *Journal of Machine Learning Research*, pp. 3849–3875, 2015.

[15] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *KDD '15*, pp. 1235–1244.

[16] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, Dec. 2010.

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

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

[19] J. Nam and S. Kim, "CLAMI: Defect prediction on unlabeled datasets," in *ASE '15*, pp. 452–463.

[20] S. Ben-david, J. Blitzer, K. Crammer, and O. Pereira, "Analysis of representations for domain adaptation," in *NIPS '07*, pp. 137–144.

[21] N. Chen and S. Kim, "Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles," in *ASE '12*, pp. 140–149.

[22] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly, "Leveraging the crowd: How 48,000 users helped improve lync performance," *IEEE Software*, vol. 30, no. 4, pp. 38–45, 2013.

[23] V. H. M. Gomide, P. A. Valle, J. O. Ferreira, J. R. G. Barbosa, A. F. da Rocha, and T. M. G. d. A. Barbosa, "Affective crowdsourcing applied to usability testing," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 575–579, 2014.

[24] M. Gómez, R. Rouvoy, B. Adams, and L. Seinturier, "Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring," in *MOBILESoft '16*, pp. 88–99.

[25] X. Yang, D. Lo, X. xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *QRS '15*, pp. 17–26.

[26] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *ICSE '16*, pp. 297–308.

[27] A. Lam, A. Nguyen, H. Nguyen, and T. Nguyen, "Combining deep learning with information retrieval to localize buggy files for bug reports," in *ASE '15*, pp. 476–481.

[28] V. Raychev, M. Vechev, and E. Yahav, "Code completion with statistical language models," in *PLDI '14*, pp. 419–428.

[29] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *MSR '15*, pp. 334–345.

[30] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *ASE '16*, pp. 87–98.

[31] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *FSE '16*, pp. 631–642.