Software defect prediction usually is regarded as a classification problem, but classification models will face the class imbalance problem. Although there are many methods to solve the class imbalance problem, there is no method that can fundamentally solve the problem currently. In addition, supervised learning algorithms are always used to train defect prediction models, but obtaining a large amount of high-quality labelled data requires a lot of time and labor cost. In order to solve the class imbalance problem and eliminate the disadvantage of supervised learning, this paper attempts to predict software defects from a new perspective of anomaly detection. We propose an Anomaly Detection Model Based on BiGAN for Software Defect Prediction (ADGAN-SDP). The model proposed in this paper not only does not need to consider the class imbalance problem but also uses a semi-supervised method to train the model. Eight classification-based software defect prediction models are used as the baseline models and compared with ADGAN-SDP model. We evaluate ADGAN-SDP on 19 projects from NASA, AEEEM, and ReLink repositories. The experimental results show that the ADGAN-SDP model, which has a higher recall, outperforms all baseline models. It is suggested that the anomaly detection approach can be applied to the software defect prediction to fundamentally solve the class imbalance problem.

1. Introduction

Software defect prediction can find defects in advance to improve development efficiency and reduce development costs. Therefore, it remains a research highlight [1–3]. At present, most researchers usually use various classifiers to construct software defect prediction models, e.g., naive Bayes [4], decision tree (DT) [5], and support vector machine (SVM) [6], etc. However, these prediction models usually face the class imbalance problem, which leads to the prediction of defective samples as non-defect samples.

Currently, researchers have proposed various methods to solve the class imbalance problem. These methods can be roughly divided into the following three kinds [7, 8]:
However, it takes a lot of cost to obtain a large amount of high-quality labelled data. For this problem, SZZ and its improved algorithms are commonly used in the field of software defect prediction to achieve automatic data labelling [9, 10].

1.1. Motivation. Although researchers have proposed various methods to solve the class imbalance problem, all methods have certain deficiencies to solve the problem. Sampling methods will change the original data distribution which leads to over-fitting or under-fitting problems. Specifically, undersampling methods will lose some important information, while oversampling methods may give rise to over-fitting problems. In cost-sensitive learning methods, it is difficult to accurately determine the cost of the two types of misclassification. As for ensemble learning methods, each weak classification still faces the class imbalance problem. Therefore, to the best of our knowledge, there is no method to fundamentally solve the class imbalance problem.

In order to obtain a large amount of high-quality labelled data, although researchers have tried to improve the original data labelling algorithm: SZZ [9], noise data are still inevitable. Therefore, it is a limitation for a given solution that tries to use supervised learning methods to train software defect prediction models.

According to previous analysis, the current methods still do not completely solve the class imbalance problem which has a great impact on the prediction performance. However, the root cause of the class imbalance problem arises from treating the defect prediction problem as the classification problem and consequently using various classifiers to build software defect prediction models. In addition, a large amount of high-quality labelled data is often required to obtain predictive models with better performance. However, current techniques do not allow for a large amount of high-quality labelled data at the desired cost. Therefore, we need to discard the solution that treats the defect prediction problem as a classification problem. Motivated to study works of Nahar Neela et al. [11] and Afric et al. [12], we decided to treat software defect prediction as an anomaly detection.

1.2. Contribution. In this paper, we propose an Anomaly Detection Model Based on Bidirectional Generative Adversarial networks (BiGAN) [13] for Software Defect Prediction (ADGAN-SDP). It is worth noting that adversarial generative network (GAN) can capture the feature distribution of complex data and has made a lot of breakthroughs in the field of computer vision [14]. From the perspective of anomaly detection, nondefective samples are treated as normal samples, while defective samples are considered as abnormal samples. That is, only normal data samples (nondefective samples) are used to train the prediction model. ADGAN-SDP uses BiGAN to capture the feature distribution of nondefective samples, and determines whether a sample is a nondefective or defective sample according to the loss function. From the perspective of anomaly detection to build software defect prediction model, it not only does not need to consider the class imbalance problem that exists in the classification-based model but also uses a semi-supervised method to train the prediction model.

In empirical research, we compare ADGAN-SDP with eight classification-based software defect prediction (C-SDP) models that are widely used in other software defect prediction researches. Experimental results show that the prediction performance of ADGAN-SDP is superior to eight C-SDP models selected in this paper.

In summary, this paper makes the following contributions:

(1) This paper proposes an anomaly detection model based on bidirectional generative adversarial networks (BiGAN) for software defect prediction. ADGAN-SDP not only does not need to consider the class imbalance problem but also a semi-supervised method is used to train the prediction model.

(2) Through detailed empirical research, we find that the prediction performance of ADGAN-SDP outperforms eight C-SDP models. Thus, it is suggested that the anomaly detection can be applied to the software defect prediction to fundamentally solve the class imbalance problem.

The rest of this paper is organized as follows. Section 2 introduces researching background. Our model: ADGAN-SDP is elaborated in Section 3. We evaluate our model and analyze the results in Section 4. Section 5 discusses relevant experimental results and conclusions. Related works are presented in Section 6. Section 7 provides the threats to validity are discussed in Section 7. We conclude in Section 8.

2. Background

This section mainly introduces the background knowledge of class imbalance problem and anomaly detection.

2.1. Class Imbalance Problem. For software defect data, more than 80% of defects usually exist in only 20% or less of the data. Therefore, the classifiers pay more attention to majority class and ignore minority class, which results in the class imbalance problem [7, 8]. For software defect prediction, when there is a class imbalance problem, the prediction model usually predicts defect samples as defect-free samples, which leads to higher accuracy and lower recall. Therefore, if the class imbalance problem is not well solved, it can greatly affect the performance of defect prediction. At present, the researchers have proposed various methods to solve the class imbalance problem. These methods can be roughly divided into two categories: data-level and algorithm-level methods. Data-level methods include miscellaneous data over-sampling and undersampling methods. The algorithm-level methods solve the class imbalance problem by directly changing the training process to pay more attention to the minority class. Besides, ensemble learning is also a common method used to deal with the class imbalance problem.
2.2. Anomaly Detection. Anomaly detection refers to finding data that are inconsistent with the expected behavior pattern. Anomalies can be classified into three categories, namely, point anomalies, contextual anomalies, and collective anomalies [15]. As an important research field, anomaly detection has been extensively studied and many anomaly detection methods have been proposed [16, 17].

Anomaly detection models can be divided into classification-based, clustering-based, nearest neighbor-based, statistics-based, information theory-based, and spectrum-based. In addition, some deep learning-based anomaly detection technology has also been proposed with the widespread application of deep learning [18]. The classification-based anomaly detection technology uses labelled training samples to train a classification model. Then, the trained classifier is used to classify the test samples into normal or abnormal. The anomaly detection technology based on clustering, which has been extensively studied by many researchers, usually uses various unsupervised or semi-supervised clustering algorithms for anomaly detection. According to the different types of clustering algorithms, the technology can be divided into two categories, namely, anomaly detection based on hierarchical clustering and partitional clustering. Anomaly detection technology based on the nearest neighbor usually needs to define a standard to calculate the similarity or distance between two samples firstly. Then, the correlation between the current sample and other samples is calculated to judge whether the current sample is an abnormal sample. Statistics-based anomaly detection technology uses given data (usually normal samples) and a statistical method to train a statistical model. Then, an unknown sample is tested to determine whether it belongs to the statistical model. If the probability of the sample belonging to the statistical model is lower, the sample is judged to be an abnormal sample; otherwise, the sample is a normal sample. Anomaly detection technology based on information theory assumes that if the data have anomalies, the information content of the data set will be irregular. Kolmogorov complexity, entropy, and relative entropy are usually used to analyze the information content to determine whether there are anomalies in the data. Spectrum-based anomaly detection technology assumes that normal samples and anomalies will present obvious differences when data can be embedded into a lower dimensional subspace [15]. PCA (Principal Component Analysis) is a commonly used technology to embed data to a low-dimensional subspace [19]. Anomaly detection technology based on deep learning has also made a lot of progress in recent years. Autoencoder [20] and GAN models [21] are usually used for anomaly detection. Reinforcement learning [22] can also be used to build anomaly detection models.

3. Our Approach

This part will elaborate the model proposed in this paper. We regard software defect prediction as anomaly detection with similar intuition and motivation as study works of Nahar Neela et al. [11] and Afric et al. [12], but our model is completely different from theirs. This section first briefly introduces the principle of anomaly detection based on BiGAN, then elaborates the ADGAN-SDP model.

3.1. Anomaly Detection Based on GAN. The adversarial generation network (GAN) can obtain complex high-dimensional data distribution, which implies that it can be applied for anomaly detection [23]. In the training phase, since only normal data samples are used to train the GAN model, a trained GAN should be able to obtain normal samples feature distribution. In the testing phase, the loss for normal data samples (nondefective data) is smaller, while the loss for abnormal data samples (defective data) is larger, thus abnormal data samples can be identified [23, 24].

Bidirectional Generative Adversarial networks (BiGAN) [13] is employed in this paper, and the overall architecture of BiGAN is shown in Figure 1. Comparing with the general GAN model, BiGAN not only includes a generator $G$ and a discriminator $D$ but also has an encoder $E$ that maps real input data to a latent encoding representation. By introducing encoder $E$, we not only acquire meaningful semantic representations related to given tasks but also the amount of calculation of the testing phase can be reduced. Specifically, the input of the discriminator of the GAN model usually is real data or generated data and the output is usually the probability value of current sample belonging to real samples. However, the input of discriminator of the BiGAN model is data array, that is $[x, E(x)]$ or $[G(z), z]$, the output is the same as the general GAN. The training objective function of the BiGAN model is similar to general GAN model, so its objective function can also be defined as a minimax function. The objective function is shown in equation (1). There are a variety of methods to train the BiGAN. In this paper, we adopted the strategy of simultaneously training the generator $G$ and the encoder $E$ according to suggestion of Dumoulin et al. [25].

$$\min_{G,E} \max_D V(D) = E[\log(X, E(X))] + E[\log(1 - D(Z, G(Z)))]$$

The anomaly detection based on BiGAN includes two stages. First, BiGAN network model is trained according to the above training strategy using normal data samples (nondefective samples). Then, we need to define a convex function to measure the abnormality of the current sample, that is, if the output value of the function exceeds a predefined threshold, the current sample will be judged as an abnormal sample, otherwise a normal sample. The convex function is shown in the following equation.

$$L(t) = \alpha L_G(t) + (1 - \alpha) L_D(t),$$

where, in equation (2), $t$ represents the current sample to be tested, $L_G$ represents the reconstruction loss of the sample $t$, $L_D$ represents the loss based on the discriminant model $D$ of the sample $t$, and $\alpha$ represents the weight coefficient of two different losses. For detailed information about the anomaly detection model based on BiGAN, please refer to [24].

3.2. Software Defect Prediction Approach Based on BiGAN Anomaly Detection. Algorithm procedure of the ADGAN-SDP model can be divided into three stages: data preprocessing, model construction, and model prediction,
which is shown in Figure 2. Unlike most researches which usually regard software defect prediction as a classification problem, we try to construct the software defect prediction model from the perspective of anomaly detection in order to solve the class imbalance problem. Compared with the classification-based software defect prediction models, our model only needs nondefect samples as training data without considering the class imbalance problem. Our model uses BiGAN to model the nondefect samples, while the classification-based model needs to use classifiers such as random forests, support vector machines, and naive Bayes to model the overall samples. In the prediction stage, our model needs to manually specify a threshold while the classification-based model does not need. In contrast to study works of Nahar Neela et al. [11] and Afric et al. [12], ADGAN-SDP uses BiGAN to build the anomaly detection model so that it can better capture the characteristics of nondefect samples.

In the first stage, we need to perform some necessary data preprocessing (feature selection, data normalization, etc.). In the second stage, the BiGAN model is trained with nondefective samples, so that the distribution characteristics of normal data samples (nondefective samples) can be obtained. In the third stage, we predict whether the current sample contains defects by comparing the loss value of the sample with the predefined threshold [24]. The detail is shown in Algorithm 1.

As shown in Algorithm 1, ADGAN-SDP is an anomaly detection approach based on data reconstruction for software defect prediction. ADGAN-SDP attempts to fit the data distribution of the nondefective samples. Furthermore, because the data distribution of defective samples is different from the nondefective samples, there will be larger reconstruction loss if the current sample comes from defective samples set. ADGAN-SDP software defect prediction framework consists of three stages: data preprocessing stage, model construction stage, and software defect prediction stage. In the data preprocessing stage (lines 1–4), first, we use the Correlation-based Feature Selection (CFS) algorithm [26] to select the best feature subset (line 1). Then, we perform min-max scaling on all project data (line 2). Finally, we divide the dataset into training dataset and test dataset (line 3) and remove defect data samples from the training dataset to obtain the required final training data (line 4).

In the model construction stage (lines 5–6), we use the training data obtained in the previous step to alternate training encoder $E$, generator $G$, and discriminator $D$ of the BiGAN model repeatedly in order to acquire the data distribution of no-defective samples. Specifically, we first train the discriminator $D$ iteratively for $k$ times with the encoder $E$ and generator $G$ remaining unchanged. Then, we train the encoder and generator for only once, while the discriminator $D$ keeps unchanged. Next, we repeat the previous two steps until the specified number of iterations is reached. In the software defect prediction stage (lines 7–16), we make predictions by judging whether the current sample $t$ in the testset is an abnormal sample. Specifically, we calculate the reconstruction loss of the current sample $t$ firstly (line 8). Then, we calculate the corresponding feature matching loss based on the discriminator (line 9). Finally, the final loss of sample $t$ is obtained (line 10). We make predictions by comparing the final loss value with the specified threshold (lines 11–15); if the loss is less than a predetermined threshold, it is a nondefective sample, otherwise it is a defective sample. In the end, the predicted values of each test sample are saved and outputted.
4. Experiment and Result

This section mainly introduces experimental data, evaluation metrics, experiment design, and results and analysis.

4.1. Experimental Data. The experimental data used in this paper comes from NASA, AEEEM, and Relink, three public software defect databases. In the software defect prediction field, NASA is a widely used public dataset. The NASA dataset comes from the projects of the national aeronautics and space administration of the United States [27]. The AEEEM dataset was published by D’Ambros et al. [28]. It contains about five Java projects. Relink defect dataset was provided by Wu et al. [29]. It contains three Java projects. This paper uses nine, five, and three projects from the NASA, AEEEM, and Relink, respectively. The detailed information of experimental data is shown in Table 1. The table shows the dataset name (column 1), the projects name (column 2), granularity level (column 3), the number of features (column 4), the number of instances (column 5), the number of defective samples (column 6), and the defect rate (column 7).

4.2. Evaluation Metrics. Because the software defect prediction dataset has the nature of class imbalance, we use AUC (Area under the ROC Curve), F1 value, and G-means, three evaluation indicators to evaluate the prediction performance of the models [30, 31].

AUC refers to the area under ROC (receiver operating characteristic curve) curve. Meanwhile, the ROC curve is generated by TPR (true-positive rate) and FPR (false-positive rate) considering different decision thresholds.

F1 value is an evaluation index that considers both precision and recall. The calculation method is shown in the following equation.

\[
F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

(3)

G-mean is another commonly used evaluation indicator that is similar to the F1 value, which considers both specificity and recall. The calculation method is presented in the following equation.

\[
G - \text{mean} = \sqrt{\text{Recall} \times (1 - \text{Specificity})}
\]

(4)

4.3. Experiment Setup. ADGAN-SDP is mainly proposed to solve the class imbalance problem. Therefore, we choose cost-sensitive learning, oversampling, undersampling, and ensemble learning, four types of methods, to solve the class imbalance problem as comparison methods. Cost-sensitive learning methods try to solve the class imbalance problem by giving different misclassification costs and minimizing the total expected costs. Cost-sensitive learning methods are implemented by parameter setting provided in the scikit-learn library, wherein the parameter “class_weight” is set to “balanced.” We use SMOTE (Synthetic Minority Over-sampling Technique) [32] and RandomUnderSampler as oversampling and undersampling methods, respectively. They are implemented by invoking imblearn tool with default settings. RF (Random Forest) [33] and ADA (AdaBoost) [34], which are implemented by calling scikit-learn toolkit with default settings, are used as ensemble learning method. We use SVM (Support Vector Machine) and RF (Random Forest), two classifiers with better performance, to

---

**Algorithm 1: Software defect prediction framework based on BiGAN anomaly detection.**

**Input:** original data set $S_{data}$

**Output:** the predicted result of the testset

/* Data preprocessing stage */

1. $S_{data}$ $\rightarrow$ CFS $S_{data}$
2. $S_{data}$ $\rightarrow$ MinMaxScaling $S_{new}$
3. $S_{new}$ $\rightarrow$ $S_{train}$ (80%) $\bigcup$ $S_{test}$ (20%)
4. $S_{final\_train}$ $\leftarrow$ $S_{train}$ $-$ $S_{defect\_train}$

/* Model construction stage */

5. Initialize encoder $E$, generator $G$, and discriminator $D$ in the BiGAN model
6. Train BiGAN model using $S_{final\_train}$

/* Software defect prediction stage */

/* for $t$ in $S_{test}$ do */

7. $L_q(t) = t - G(E(t))$
8. $L_D(t) = F_D(t, E(t)) - F_D(G(E(t)), E(t))$
9. $L(t) = a L_q(t) + (1 - a) L_D(t)$
10. if $L(t) < \text{P\_threshold}$ then
11. $\text{pre\_result}[t] = 0$
12. else
13. $\text{pre\_result}[t] = 1$
14. end if
15. end for
16. return $\text{pre\_result}$
evaluate the effect of four types of methods on the 19 projects. Two classifiers are implemented by invoking the scikit-learn toolkit with default settings. Therefore, there are RFB (RF + Balance), SVMB (SVM + Balance), RFS (RF + SMOTE), SVMS (SVM + SMOTE), RFU (RF + Undersampling), SVMU (SVM + Undersampling), RF, and ADA (AdaBoost), eight comparison models.

All software defect prediction models studied in this paper are implemented in python language. ADGAN-SDP model uses the tensorflow deep learning framework. The CFS feature selection algorithm [26] is implemented by invoking the weka tool with default settings. The ADGAN-SDP model consists of three parts, namely, Encoder, Discriminator, and Generator. The three parts are all composed of four layers, using the relu activation function. AdamOptimizer is used as optimizer. Due to the difference of each project, the threshold value of each project and the weight value of two kinds of losses are determined manually according to the experimental results.

In our experiment, we randomly choose 80% of each project as the training data and the rest as the test data. The number of training on each project is set to 100. In order to ensure that the experimental results are not affected by randomness, we repeat the experiment 10 times on each project and take the average value of 10 times as the final result. AUC and F1 value, two evaluation indicators, use tools provided by the scikit-learn library. Since many experiments in this paper are implemented by third-party tools, the reproducibility of the experimental results can be greatly improved.

4.4. Experiments and Analysis. The experimental results are provided in Figure 3. It is box plots of prediction performance using AUC, G-mean, and F1, three evaluation indicators. The experimental detailed results are presented in Tables 2–4, the best predicted value of each project is in bold. What stands out from Figure 3 is that the prediction performance of ADGAN-SDP is significantly better than other models. As shown in Figure 3(a), the first, the second, and the third quartile of ADGAN-SDP all higher other comparison models. The results of Figure 3(b) are similar with Figure 3(a). However, results of Figure 3(c) are somewhat different from other the two figures. Figure 3(c) shows that only the second and the third quartile of ADGAN-SDP rank higher than other comparison models. Further observation shows that for class imbalance problems dealing, ensemble learning method has the worst effect, while the other three types of methods have their own advantages in different classifiers.

Tables 2–4 are the detailed experiment results. From Table 2 we can see that ADGAN-SDP achieves the best AUC value on 12 out of 19 projects; this performance is better than the other comparison models. Closer inspection of the table shows that the mean of ADGAN-SDP in 19 projects is higher than the other eight comparison models. ADGAN-SDP outperforms RFB, SVMB, RFS, SVMS, RFU, SVMU, RF, and ADA by 19.86%, 5.86%, 10.9%, 6.98%, 6.03%, 9.56%, 20.08%, and 22.94%, respectively, in terms of average AUC value. It can be seen from the data in Table 4 that ADGAN-SDP achieves the best F1 value on 9 out of 19 projects; this performance is better than the other comparison models. However, Table 4 presents that the mean of ADGAN-SDP in 19 projects is also higher than other comparison models except for the RFS model. ADGAN-SDP outperforms RFB, SVMB, SVMS, RFU, SVMU, RF, and ADA by 21.59%, 1.9%, 3.67%,

Table 1: Information table of experiment data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Project</th>
<th>Level</th>
<th>#feature</th>
<th># instances</th>
<th># defects</th>
<th>%defect (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASA</td>
<td>CM1</td>
<td>Method</td>
<td>37</td>
<td>327</td>
<td>42</td>
<td>12.8</td>
</tr>
<tr>
<td></td>
<td>JM1</td>
<td>Method</td>
<td>21</td>
<td>7782</td>
<td>1672</td>
<td>21.5</td>
</tr>
<tr>
<td></td>
<td>KC3</td>
<td>Method</td>
<td>39</td>
<td>194</td>
<td>36</td>
<td>18.6</td>
</tr>
<tr>
<td></td>
<td>MC1</td>
<td>Method</td>
<td>38</td>
<td>1988</td>
<td>46</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>MC2</td>
<td>Method</td>
<td>39</td>
<td>125</td>
<td>44</td>
<td>35.2</td>
</tr>
<tr>
<td></td>
<td>MW1</td>
<td>Method</td>
<td>37</td>
<td>253</td>
<td>27</td>
<td>10.7</td>
</tr>
<tr>
<td></td>
<td>PC1</td>
<td>Method</td>
<td>37</td>
<td>705</td>
<td>61</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td>PC2</td>
<td>Method</td>
<td>36</td>
<td>745</td>
<td>16</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>PC3</td>
<td>Method</td>
<td>37</td>
<td>1077</td>
<td>134</td>
<td>12.4</td>
</tr>
<tr>
<td></td>
<td>PC4</td>
<td>Method</td>
<td>37</td>
<td>1287</td>
<td>177</td>
<td>13.8</td>
</tr>
<tr>
<td></td>
<td>PC5</td>
<td>Method</td>
<td>38</td>
<td>1711</td>
<td>471</td>
<td>27.5</td>
</tr>
<tr>
<td>AEEEM</td>
<td>EQ</td>
<td>Class</td>
<td>61</td>
<td>324</td>
<td>129</td>
<td>39.8</td>
</tr>
<tr>
<td></td>
<td>LC</td>
<td>Class</td>
<td>61</td>
<td>691</td>
<td>64</td>
<td>9.3</td>
</tr>
<tr>
<td></td>
<td>ML</td>
<td>Class</td>
<td>61</td>
<td>1862</td>
<td>245</td>
<td>13.2</td>
</tr>
<tr>
<td></td>
<td>JDT</td>
<td>Class</td>
<td>61</td>
<td>997</td>
<td>206</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>PDE</td>
<td>Class</td>
<td>61</td>
<td>1497</td>
<td>209</td>
<td>14</td>
</tr>
<tr>
<td>Relink</td>
<td>Apache</td>
<td>File</td>
<td>26</td>
<td>194</td>
<td>98</td>
<td>50.5</td>
</tr>
<tr>
<td></td>
<td>Safe</td>
<td>File</td>
<td>26</td>
<td>56</td>
<td>22</td>
<td>39.3</td>
</tr>
<tr>
<td></td>
<td>ZXing</td>
<td>File</td>
<td>26</td>
<td>399</td>
<td>118</td>
<td>29.6</td>
</tr>
</tbody>
</table>
2.79%, 7.88%, 23.78%, and 36.4%, respectively, in terms of average $F_1$ value. In addition, we also apply win/draw/loss statistics for ADGAN-SDP and other comparison models. The statistical results of the “W/D/L” row in Tables 2–4 provide that the prediction performance of ADGAN-SDP outperforms other comparison models.

To check if the differences of the prediction performance of ADGAN-SDP and eight comparison models are statistically significant, we use the Wilcoxon signed-rank test at 95% significance level on three evaluation indicators. To make statistics more accurate, we also use Bonferroni to correct statistics results [35]. Besides, we also apply Cohen’s delta (a nonparametric effect size measure) [36] to measure the effect size with four-level effect size, which is shown in Table 5. Statistical test results are shown in “p value” and “Cliff’s d” row of Tables 2–4. Table 2 presents that ADGAN-SDP shows significant improvements ($p$ values < 0.05) only on the ADA model with medium size effect ($\delta = 0.4294$).

Finally, we conduct the nonparametric Friedman test with post-hoc Nemenyi test [37] ($\alpha = 0.05$) comparing ADGAN-SDP with eight comparison models over the 19 projects on three evaluation metrics. The analysis results are presented in Figure 4.

In Figure 4, the results of the post-hoc test provide that ADGAN-SDP always belongs to the top rank group in terms of AUC and G-mean two indicators. However, SVMB belongs to the top rank group in terms of $F_1$. These observations indicate that ADGAN-SDP performs significantly better than the eight comparison models in terms of AUC and G-mean two indicators.

However, we notice that prediction performance of $F_1$ metric is different from AUC and G-mean two metrics. We guess that it may be caused by different recall and precision.

Figure 3: The boxplots of ADGAN-SDP model and other comparison models under (a) AUC, (b) G-mean, and (c) $F_1$ evaluation metrics.
of models. In order to analyze the reason, average precision and recall of different models on 19 projects are shown in Figure 5. As shown in Figure 5, this is an expected result that precision is inversely proportional to recall. From the figure we can see that ADGAN-SDP has higher recall rate, but lower precision. For F1 value, the prediction performance of ADGAN-SDP is not ideal due to considering both precision and recall.
Table 4: Prediction result of ADGAN-SDP and other comparison models on 19 projects under F1 metric.

<table>
<thead>
<tr>
<th>Project</th>
<th>RFB</th>
<th>SVMB</th>
<th>RFS</th>
<th>SVMS</th>
<th>RFU</th>
<th>SVMU</th>
<th>RF</th>
<th>ADA</th>
<th>ADGAN-SDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>0.0692</td>
<td>0.2874</td>
<td>0.1924</td>
<td>0.2740</td>
<td>0.3131</td>
<td>0.0541</td>
<td>0.1215</td>
<td><strong>0.4516</strong></td>
<td></td>
</tr>
<tr>
<td>JM1</td>
<td>0.2574</td>
<td>0.3516</td>
<td>0.3784</td>
<td>0.3477</td>
<td>0.4010</td>
<td>0.3060</td>
<td>0.2739</td>
<td>0.1718</td>
<td><strong>0.4083</strong></td>
</tr>
<tr>
<td>KC3</td>
<td>0.2700</td>
<td>0.3745</td>
<td>0.3285</td>
<td>0.3115</td>
<td>0.3081</td>
<td>0.3257</td>
<td>0.1579</td>
<td>0.1424</td>
<td><strong>0.4000</strong></td>
</tr>
<tr>
<td>MC1</td>
<td>0.3623</td>
<td>0.1155</td>
<td><strong>0.3686</strong></td>
<td>0.0911</td>
<td>0.1225</td>
<td>0.1060</td>
<td>0.4523</td>
<td>0.1023</td>
<td>0.1750</td>
</tr>
<tr>
<td>MC2</td>
<td>0.4716</td>
<td>0.4698</td>
<td>0.4975</td>
<td>0.3997</td>
<td>0.5080</td>
<td>0.4153</td>
<td>0.3668</td>
<td>0.4397</td>
<td>0.5000</td>
</tr>
<tr>
<td>MW1</td>
<td>0.2558</td>
<td>0.3896</td>
<td>0.3701</td>
<td>0.4149</td>
<td>0.2912</td>
<td>0.2757</td>
<td>0.2287</td>
<td>0.1928</td>
<td><strong>0.5263</strong></td>
</tr>
<tr>
<td>PC1</td>
<td>0.1816</td>
<td>0.2916</td>
<td>0.3982</td>
<td>0.3027</td>
<td>0.3129</td>
<td>0.2542</td>
<td>0.2148</td>
<td>0.2422</td>
<td>0.2381</td>
</tr>
<tr>
<td>PC2</td>
<td>0.0000</td>
<td>0.0986</td>
<td>0.0467</td>
<td>0.0886</td>
<td>0.0996</td>
<td>0.1021</td>
<td>0.0000</td>
<td>0.0400</td>
<td><strong>0.2500</strong></td>
</tr>
<tr>
<td>PC3</td>
<td>0.1941</td>
<td>0.4025</td>
<td>0.3795</td>
<td>0.4129</td>
<td>0.3903</td>
<td>0.3798</td>
<td>0.2214</td>
<td>0.2439</td>
<td><strong>0.4138</strong></td>
</tr>
<tr>
<td>PC4</td>
<td>0.3139</td>
<td>0.4111</td>
<td><strong>0.4606</strong></td>
<td>0.4209</td>
<td>0.4235</td>
<td>0.3773</td>
<td>0.3443</td>
<td>0.2908</td>
<td>0.3418</td>
</tr>
<tr>
<td>PC5</td>
<td>0.4461</td>
<td>0.4655</td>
<td>0.5141</td>
<td>0.4718</td>
<td>0.5279</td>
<td>0.4573</td>
<td>0.4511</td>
<td>0.3545</td>
<td><strong>0.4678</strong></td>
</tr>
</tbody>
</table>

The bold values of each row represent best predicted values of each project except last three rows.

Table 5: Cliff's delta and the effectiveness level [36].

| Cliff's Delta (|δ|) | Effectiveness level |
|----------------|---------------------|
| 0.000 ≤ |δ| < 0.147 | Negligible |
| 0.147 ≤ |δ| < 0.330 | Small |
| 0.330 ≤ |δ| < 0.474 | Medium |
| 0.474 ≤ |δ| < 1.000 | Large |

Figure 4: The ranks on (a) AUC, (b) G-mean, and (c) F1 for ADGAN-SDP model and other comparison models with post-hoc Nemenyi test. Models connected by gray lines are not significantly different.
In summary, these results show that the prediction performance of ADGAN-SDP model is better than classification-based software defect prediction models. The evidences from those studies suggest that the anomaly detection models can be applied to the software defect prediction to fundamentally solve the problem of class imbalance.

5. Discussion

5.1. Discussion on Experimental Results. Because of the class imbalance in itself, it is a limitation for a given solution that tries to tackle the problem of defect prediction as a binary classification problem. We use anomaly detection methods to build software defect prediction models that can fundamentally solve the class imbalance problem. We verify the effectiveness of ADGAN-SDP by using eight prediction models with better performance as baseline models, while there are other better and improved classification models. By means of observing Table 1 and experimental results, we find that ADGAN-SDP can achieve better prediction performance compared with other baseline models when the defect rate is higher. These findings suggest that ADGAN-SDP can find more defective instances so as to result in higher recall, which further supports the experimental results of Figure 5. Taken together, ADGAN-SDP has higher recall and lower precision, while in other baseline models it is contrary. Furthermore, lower recall means more false negative (defective instance marked nondefective instance), while lower precision means more false positive (nondefective instance marked defective instance). In practice, the cost for incorrectly marking a nondefective instance as a defective instance is far lower than the cost for marking a defective sample as a nondefective instance. For training prediction models, the model proposed in this paper only needs nondefective samples, and nondefective samples are easier to obtain than defective samples. Therefore, it is more favorable to use anomaly detection methods to build defect prediction models. Theoretically, ADGAN-SDP has natural advantages compared with classification-based models. The software defect prediction model based on anomaly detection proposed in this paper is essentially different from the past classification-based models; the former only tries to capture the feature distribution of nondefective samples, while the latter needs to know the overall feature distribution including nondefective and defective samples. In summary, these discussions are very encouraging. We recommend using anomaly detection models instead of classification models to build software defect prediction models.

5.2. Time Efficiency of ADGAN-SDP. Table 6 provides model training and test time across 19 projects. From Table 6, we notice that the training and test time of ADGAN-SDP model is reasonable. Specifically, on average, it takes about 11.265 seconds to train a model, and 0.078 seconds to predict the label of instances in the test set using the model. It is worth noting that the model does not need to be updated once the training is completed. ADGAN-SDP model training time is much longer than that of other models across all projects, which implies that ADGAN-SDP model is more complex than other methods. ADGAN-SDP model test time is slightly longer than other models, but we believe it is still acceptable. In summary, ADGAN-SDP model can be applied to real software defect prediction in terms of time efficiency.

6. Related Work

6.1. Class Imbalance Problem. The class imbalance problem is one of the main problems affecting the performance of defect prediction. Researchers have proposed several methods to solve the problem. Bennin et al. [38] proposed a data oversampling method called MAHAKIL; this method was based on the genetic theory of chromosomes to synthesize new minority samples. MAHAKIL could increase the diversity of data and reduce data over-fitting risk. Arar and
Ayan [39] used cost-sensitive artificial neural networks to predict software defects. Meanwhile, they used a bee colony algorithm to train prediction models. For solving the class imbalance problem, Zhang et al. [40] proposed a model called the cost-sensitive residual convolutional neural network (CS-ResNet) for PCB cosmetic defect detection. This model added a cost-sensitive adjustment layer that assigned larger weights to minority real defects based on the class-imbalance degree in the standard ResNet [41]. In addition, ensemble learning can also be used to deal with class imbalance problems [42, 43]. Unlike most researchers who regard defect prediction as a classification, we study defect prediction from the perspective of anomaly detection to completely solve the problem of class imbalance. Besides, instead of the supervised learning method commonly used in the classification model to train the prediction model, a semi-supervised method is employed in this paper.

### 6.2. Software Defect Prediction Based on Anomaly Detection

At present, the research on treating defect prediction as anomaly detection is relatively less. In order to solve the class imbalance problem, Nahar Neela et al. [11] first tried to apply anomaly detection to software defect prediction. They first used Exhaustive subsetting and Iterative subsetting to select the best attributes set. Then, they modeled nondefected software module by incorporating both univariate and multivariate Gaussian distribution. Finally, unseen instances were determined as nondefective or defective based on their deviation from the generated model. Afric et al. [12] proposed the Reconstruction Error Probability Distribution (REPD) model for source code defect prediction. They employed autoencoder neural network architecture to establish an anomaly detection model, and classified the examples as nondefective or defective according to different reconstruction error. The experimental results showed that the REPD model achieved significantly better predicted accuracy.
results. Those study works that treat defect prediction as anomaly detection are quite promising for solving the class imbalance problem. As far as we know, we are the first to propose a model like ADGAN-SDP, which utilized BiGAN to construct defect prediction model based on anomaly detection.

7. Threat to Validity

Internal validity threats. We have carefully checked the codes and data. However, there may still errors that are ignored. Moreover, eight baseline models are implemented by calling the Imbalance and Scikit-learn toolkits with default settings. In the field of machine learning, different parameter settings lead to different experimental results. Therefore, other tools or different parameter settings may deviate from the experimental results in this paper.

External validity threats. The experimental data in this paper use 19 projects from NASA, AEEEM, and ReLink. However, in order to further prove the generality of ADGAN-SDP, it is necessary to experiment on more projects.

Construct validity threats. We use AUC, G-mean, and F1, three widely used evaluation indicators. In addition, precision and recall, two evaluation indicators, are also used. Moreover, the Wilcoxon signed-rank test and post-Nemenyi test are applied to investigate the statistical difference between ADGAN-SDP and baseline models.

8. Conclusion

For software defect prediction, class imbalance problem makes prediction model more biased towards majority classes so as to reduce prediction performance. In addition, due to the lack of a large amount of high-quality labelled data, a robust high-quality prediction model usually cannot be obtained. In response to the above challenges, this paper attempts to make a binary classification problem convert to an anomaly detection problem. The ADGAN-SDP model can not only fundamentally avoid the class imbalance problem but also alleviate the problem of lack of a large amount of high-quality labelled data to a certain extent because the semi-supervised training method is used. The experimental results show that ADGAN-SDP model outperforms all baseline models. In addition, the ADGAN-SDP model has more recall compared with other baseline models. These findings suggest that the anomaly detection model has more advantages than classification-based models for software defect prediction. In summary, the anomaly detection method can be applied to the software defect prediction to fundamentally solve the problem of class imbalance.

Data Availability

NASA, AEEEM, and Relink public datasets were used to support the findings of this study. The NASA dataset is available at https://www.researchgate.net/search/publication?q=NASA%2BMMDP. AEEEM dataset can be derived from https://bug.inf.usi.ch/. ReLink dataset is available at http://www.cse.ust.hk/~scc/ReLink.htm.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China under grant no. 61673384; and Natural Science Foundation of Jiangsu Province under grant no. BK20181353.

References

Scientific Programming


