Modeling Software Defects as Anomalies: A Case Study on Promise Repository

Kamrun Nahar Neela*, Syed Ali Asif, Amit Seal Ami, Alim Ul Gias
Institute of Information Technology, University of Dhaka, Dhaka – 1000, Bangladesh.

* Corresponding author. Tel.: +8801675445902; email: bit0431@iit.du.ac.bd
Manuscript submitted June 7, 2017; accepted October 27, 2017.
doi: 10.17706/jsw.12.10.759-772

Abstract: Software defect prediction is a highly studied domain in Software Engineering research due to its importance in software development. In literature, various classification methods with static code attributes have been used to predict defects. However, defected instances are very few compared to non-defected instances and as such lead to imbalanced data. Traditional machine learning techniques give poor results for such data. In this paper an anomaly detection technique for software defect prediction, is proposed which is not affected by imbalanced data. The technique incorporates both univariate and multivariate Gaussian distribution to model non-defected software module. The defected software modules are then predicted based on their deviation from the generated model. To evaluate our approach, we implemented the algorithm and tested it on the NASA datasets from the PROMISE repository. By utilizing this approach, we observed an average balance of 63.36% and 69.06% in univariate model and multivariate model respectively. Without utilizing optimization or filter, this approach yield better result than industry standard of 60%.

Key words: Software defect prediction, anomaly detection, univariate gaussian distribution, multivariate gaussian distribution, promise repository.

1. Introduction

Software is generally divided into a number of modules. Due to time and resource constraints, often faulty modules are given higher priority during testing. If fault-prone modules can somehow be identified and testing can be focused more on those areas, it will greatly decrease cost and time. Researchers found that 80% of the problems reside in only 20% of the modules [1], [2]. Identification of the defect-prone modules is thus extremely beneficial since those few modules are statistically proven to be most bug-prone and testers can therefore test those modules thoroughly. Because of these reasons, software defect prediction has been an important field in software engineering for more than 20 years [3]. Defect prediction techniques help the software teams to ensure quality product incurring minimal cost [4], [5].

Software Defect prediction can be challenging for a number of reasons. One of the major challenges is the Class imbalance problem. It is a problem that we face when the distribution of classes (defected or non-defected) in train data is heavily skewed and one or more classes are over-represented [6]. This leads to negative influence on decision of classifiers. It has been observed that the majority of the defects in software are limited to a small percentage of modules [7], [8]. In any software, the number of defected instances is very few. As a result, classifiers are often unable to detect the faulty modules since the data is imbalanced with majority of the modules being fault-free. This is a well-known problem in machine learning often referred to as learning from imbalanced datasets.

There have been numerous approaches proposed in literature with merits and limitations. Most state-of-the-art techniques use some variant of machine learning. In [8], [9], the authors used Neural Networks in predicting defects. The authors in [9], [10] made comparative studies on the different state-of-the-art approaches to determine which work best. The work in [11] is particularly noteworthy because of the authors' contribution to developing a baseline experiment that most researchers followed since. This allowed a direct comparison of the different approaches proposed in literature [12], [13]. Six issues with existing state-of-the-art techniques were discussed in [14] and while there have been a few papers that try to address the issues raised in the paper, some of the core problems still remain. However, these approaches did not address imbalanced data, thus being less effective for real world problems.

In this paper, we present an Anomaly Detection technique for predicting defects in software. Anomaly detection techniques do not suffer from the Class imbalance problem since it works quite well with skewed distribution. While considering this problem, we consider the hypothesis that defects can be considered as anomalies since they contain anomalous properties. We use both univariate and multivariate Gaussian distribution models as our anomaly detection scheme. Univariate Gaussian distribution is less computationally expensive and for general cases work well. However, there is often correlation between attributes where univariate model has issues to flag anomalies since it does not take correlation into consideration. On the other hand, the multivariate model is more computationally expensive and works well when there is sufficiently large data, but it does consider the correlation between attributes and adjusts accordingly.

We have implemented our proposed framework and compared its performance against state-of-the-art classifiers. We have presented our findings and showed that anomaly detection techniques perform as well as state-of-the-art classifiers without enhancements or optimization. For the univariate model, we get an average balance of 63.36% and for the multivariate model, we get an average balance of 69.06% which is much higher than industry standards of 60% [11]. The results obtained is positive even compared to state-of-the-art techniques considering we used only anomaly detection without any data preprocessing. Thus our hypothesis that defects can be considered as anomalies is justified and moreover, anomaly detection techniques can be used to predict software defects.

The rest of the paper is organized as follows: Section 2 contains summaries of existing literature on the domain. Section 3 discusses the methodology of the proposed research. Section 4 discusses the experimental setup. Section 5 gives the overview of our findings that are most notable. Section 6 reaches some conclusions based on the results and the assumptions we had and discusses scope of future work.

2. Related Work

Software defect prediction is a widely studied field in Software Engineering and hence there have been numerous works by researchers who have proposed a variety of approaches to predict defects in software. Most commonly used methods are statistical methods and statistical machine learning techniques, which are generally used in combination with some other methods to obtain improved performance. Neural networks have also been used by various researchers alongside traditional statistical methods. Many of the papers discuss limitations of the state-of-the-art techniques and many try to propose ways to remedy the issues. A number of state-of-the-art approaches have been studied. A synopsis of the papers studied, followed by a discussion with pros and cons of the approaches have been given below.

Some of the pioneering works in this domain were done in [15]-[17]. For predicting software defects the authors used multivariate analysis [16], [17] and showed that the accuracy of software quality prediction was not affected by decreasing independent factors. The authors also used zero-inflated Poisson Regression

[15] and displayed that it does better than just Poisson Regression. The authors in [18] applied Poisson Regression model and Binomial Regression models on software defect data that is not normally distributed and concluded that over-dispersion of data can be best dealt with negative binomial distribution. In [19] the authors also used statistical methods to predict defects. For analyzing software data, the authors explore the usefulness of multivariate Regression, Classification and Regression Trees (CART) and Residual Analysis. If the data has minor skewness, the authors showed that multivariate Regression Analysis done much better but, Residual Analysis performed best for data that has high amount of heteroscedasticity.

Artificial Neural Network along with traditional methods are used in [20]. The authors used Case-Based Reasoning (CBR), Stepwise Regression, Artificial Neural Networks (ANN) and Rule Induction. For continuous target function, the performance of stepwise regression were better. However, for discontinuous target functions, the performance of other machine learning techniques were better. The authors suggested CBR as the more favorable approach in terms of overall performance. In [9] the authors used three cost-sensitive neural networks for defect prediction. A downside of the cost sensitive learning method is the misclassification cost issue.

In [9], the authors evaluated different machine learning and statistical predictor models on real-time defect datasets. It was shown that 1R in combination with Instance-based Learning gives consistent predictions. The authors used Consistency based Subset Evaluation technique. They suggested that the size and complexity metrics are not adequate for accurately predicting real-time software defects.

An issue with existing approaches in literature regarding defect prediction prior to 2007 was that there were no baseline experiments and no reliable datasets that researchers could use to directly compare their performances [21], [22]. Moreover, to make accurate predictions of attributes like defects found in complex software projects a rich set of process factors is needed. A causal model which considered both quantitative and qualitative process factors was developed in [14]. For validation, a dataset produced from 31 finished software projects in the consumer electronics industry was presented. The dataset has been of interest to other researchers evaluating models with similar aims. An even more elaborate baseline experiment was discussed by Menzies *et al.* [11]. The authors proposed the use of NASA datasets for evaluating the performance of defect predictors and the use of static code attributes as features for the various learners. The authors also present their findings regarding the debate of which attributes among the many similar relevant attributes are best for predictors. They argued that it is not important which attributes- McCabe [21], Halstead [9] or LOC is used, but it is more important how they are used. They backed their claim by showing there is no significant improvement in result for any one attribute over the others. Moreover, the authors show that such predictors do work well and that mining static code attributes is useful for defect predictors. The authors also show that static code attributes do in fact work well in predicting defects, thus disproving that static code attributes capture very little of source code. The authors implemented their approach using Naïve Bayes and evaluated their results to affirm their hypothesis.

Machine learning and statistical methods all suffer from some problems. Class imbalance problem is one of the examples. The authors in [22] proposed a stratification-based resampling strategy in predicting software defect-proneness that minimizes the effect of the imbalance problem. The sampling methods, as proposed by the authors, can be divided into two groups: undersampling and oversampling. The under-sampling method removes the class examples which occur more frequently. On the other hand, the over-sampling method increases the class examples which occur rarely. These methods get the anticipated class distribution. The authors applied both sampling methods to minimize the effect of the class-imbalance problem. The authors randomly under-sampled the majority class examples while over-sampling the minority class examples with a technique called the SMOTE technique [23]. Results of their experiment showed an improvement of 23% in the average geometric mean classification accuracy.

In paper [14], an aggregate of problems was discussed with existing state-of-the-art defect prediction approaches. The authors identified the problems as follows: i) they found the set of attributes to be correlated with fault, ii) there is an absence of standard measures for performance assessment, iii) there are problems with cross project defect prediction, iv) there are inconsistencies with the economics of software defect prediction, v) the class imbalance problem and vi) the absence of any general framework for the software defect prediction.

In [13] the authors proposed the use of a dictionary learning technique to predict software defect in a way that reduces the effect of misclassification cost issue. The authors used open source software from where they mined metrics and thus learned multiple dictionaries. The authors proposed that one module in software can be represented by a small proportion of other modules because the nature of software is that most modules of a new software are similar to some other module of a known software. The authors also took the misclassification cost issue into account and proposed a cost-sensitive discriminative dictionary learning approach for software defect prediction. The authors implemented their approach and evaluated it for the NASA datasets.

There have been various papers that only improve the performance by optimizing existing techniques. In [24], the authors proposed an approach that improves the performance of an existing state-of-the-art technique. The authors identify the optimal set of software features by using different feature selection techniques for defect prediction. Next, the authors identify a predictor that gives maximum accuracy. Although the author shows promising results, the authors did not consider the class imbalance problem. In [25], the authors suggested a framework. The framework supports neutral and inclusive comparison between competing prediction systems. The framework includes *(i) scheme assessment and (ii) defect prediction components.* The scheme evaluation analyses the performance of different learning techniques. The defect prediction component uses the evaluated technique and constructs a new model. The authors demonstrated the performance of their framework on publicly available software defect datasets as well as simulated sets and presented that it performs better than most state-of-the-art techniques.

Various other papers discuss which attributes are best for defect prediction. In [26], the authors addressed the problem of selection of attributes. The authors proposed a Log Filtering solution to select an attribute set. The authors implemented their proposed approach and showed that their technique can effectively improve existing techniques. Since the paper discussed only on feature selection, it did not discuss which learner or classifier will give best results. Often, a subset that gives good results for one learner may not work as well for another. The authors' approach overlooked this in their approach. In [12], Qiao *et al*. proposed a feature weighted approach based on a distance measure for software defect prediction. The authors designed a ranking algorithm that maintains a rank list of the features. The ranking algorithm updates the feature weights according to sample similarity of different classes. Subsets are then picked from the list and KNN is used to evaluate them. The authors implemented their proposed approach on the NASA dataset and the selected features give good results. However, the authors tackle one problem of selecting features. Often times the same feature subset can perform very well for one learner but not as well for another. No such learners were proposed in this study.

As mentioned in [8], [27], statistical machine learning algorithms for defect prediction have issues due to the class imbalance problem since the number of defected modules is far less than the number of non-defected ones. Classifiers are then unable to detect faulty modules since they work well when there are almost equal number of instances for all the classes in the training data. The classifier can have very high accuracy but give misleading results due to the bias in the training data. In our paper, we look to address the class imbalance problem. Most state-of-the-art techniques either do not account for this issue or they look to address the issue by using a combination of methods. In our approach, we explore the use of anomaly detection techniques to counter the class imbalance problem.

3. Defect Prediction Based on Anomaly Detection

In this section, an approach based on anomaly detection is proposed for software defect prediction. By anomaly detection, we refer to the identification of events that result in unexpected behavior. Anomaly detection algorithms typically work by finding a general pattern from the data attributes. Cases that deviates from the general pattern i.e. quite different from the mean or norm are considered as exceptional cases or anomalies. In case of a software, some modules exhibit unusual and unexpected behavior, which are anomalies of the system. Those modules are considered as defected whereas other modules are non-defected. The primary objective of the proposed anomaly detection approach is to find out the general pattern of the non-defected modules. Based on their deviation from the pattern, some modules can be identified as either defected or non-defected.

3.1. Software Defects as Anomalies

In software, generally, a very small portion of the code-base is defected with the rest being non-defected. The small portion of the codebase that contains defects can be considered anomalous. In figure 1, we plotted two code attributes – Lines of Code (LOC) and Operator Count of every modules of a synthetically developed defected software. It is observed that the some modules are clustered in the middle and can be considered as non-anomalous while the other modules that reside outside of that area are anomalous. In the context of defect prediction, the cluster contains the non-defected modules while the outliers are the defected modules.

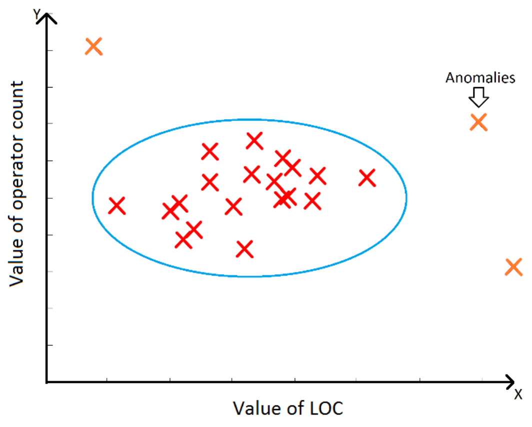

Fig. 1. Representing software defects as anomalies.

Software defect prediction is generally done using either static code attributes or runtime attributes. Runtime attributes are more difficult to work with and do not always reproduce the same results. Static code attributes are more commonly used for defect prediction because they are easy to use, can be mined directly from the code with minimal effort and they have been proven to give good results in defect prediction [11]. Like most other state-of-the-art techniques, static code attributes have been used for anomaly detection in this paper. If a particular software module is defected, it is expected that static code attributes based on LOC (Lines of Code), McCabe [9], Halstead [11], [22] metric will exhibit anomalous behavior and patterns for that software module.

3.2. Anomaly Detection Using Gaussian Distribution

Anomaly detection based on Gaussian distribution is typically used to identify exceptional cases. In our case, we used the Gaussian distribution to detect defected software module. To be more precise, we utilized two different models for detecting software defects. These models are based on univariate and multivariate Gaussian distribution respectively.

3.2.1. Univariate gaussian distribution based model

In terms of statistics and probability theory, a univariate distribution is a probability distribution of a single random variable. In this case a Gaussian or normal distribution is used, which allows to identify outliers easily since most of the values of a normal distribution are clustered around the mean. Using an attribute (like LOC) of the non-defected modules of a software, a Gaussian distribution based model can be created using the probability density function of equation 1.

$$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{1}$$

where,

$$\mu = \frac{1}{m}\sum_{k=1}^{m} x_i \tag{2}$$

is the mean or expected value of the distribution and

$$\sigma^2 = \frac{1}{m}\sum_{k=1}^{m}(x_i-\mu)^2 \tag{3}$$

is the variance of the distribution.

The mean and variance should be calculated from all the values of a particular attribute of the non-defected modules. This will actually establish a pattern for the non-defected modules. Thus the defected modules can be identified based on the deviation from the pattern. To be more precise, the model will determine anomalies depending on a threshold value (e). The further the value of the attribute of a new module is from the mean value, the more likely it is to be an anomaly. That is the approach will calculate $p(x)$, where x is the value of a particular attribute of a module, and if $p(x) < e$ then that module will be flagged as defected. This threshold can be derived through trial and error basis over the cross validation dataset.

However, in most of the cases a single attribute will not yield a satisfactory result. In that case, multiple attributes should be used. In order to find the probability density of multiple features, the product of each probability density of the selected attributes can be used according to equation (4). The product of the density is used because the features are considered independent.

$$p(x) = p(x_1;\mu_1,{\sigma_1}^2) * p(x_2;\mu_2,{\sigma_2}^2) \dots\dots\dots p(x_n;\mu_n,{\sigma_n}^2) \tag{4}$$

In equation (4) x represents a vector of attributes ($x_1, x_2, x_3, \dots x_n$). As stated earlier, each individual mean and variance for all those attributes should be calculated from the non-defected modules only. The model will be based on those pre-calculated means and variances.

Fig. 2 represents an example of univariate Gaussian distribution with only one p(x). The normal distribution has a mean of 0 and a standard deviation of 1. This is an ideal case where all non-anomalous instances occur very close to the mean and anomalous instances occur two standard deviations away from the mean. An appropriate threshold can identify those instances as anomalies.

3.2.2. Multivariate Gaussian distribution

Multivariate Gaussian distribution can take into account the correlation between the features. This is the strength of multivariate Gaussian distribution in anomaly detection because features are often correlated and factoring in the correlation is necessary for getting good results. In figure 3, a multivariate Gaussian distribution for two attributes is shown.

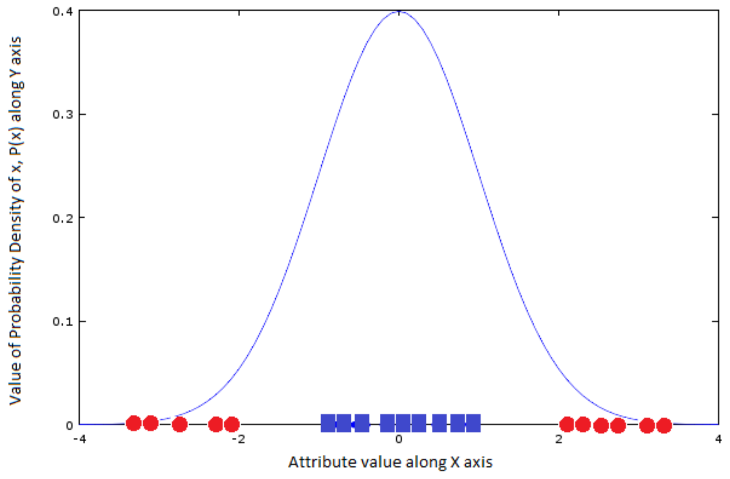

Fig. 2. Anomaly detection using univariate Gaussian distribution.

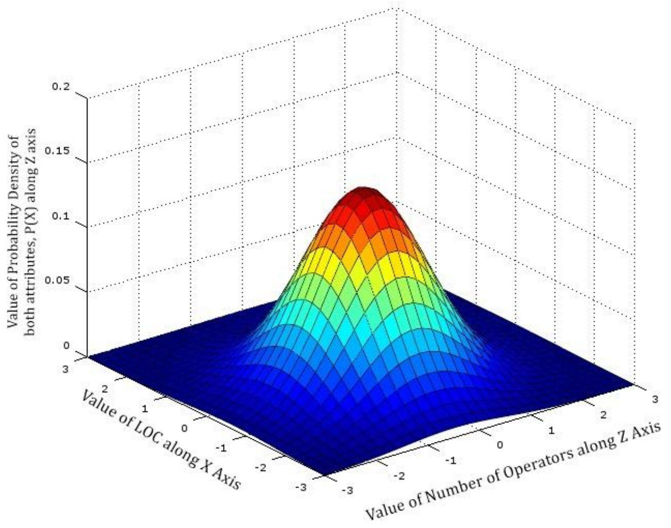

Fig. 3. Multivariate Gaussian distribution for two attributes of a software module.

It is a 3-D representation of the bell-curve. multivariate Gaussian distribution can be used for any number of attributes so long as the number of instances is higher than the number of attributes selected for the distribution. The probability density of the normal distribution for multivariate Gaussian distribution is:

$$p(x;\mu,\Sigma) = \frac{1}{\sqrt[n]{2\pi}\sqrt{\Sigma}} exp\left(-\frac{(x-\mu)^T}{2\Sigma(x-\mu)}\right) \tag{5}$$

Here x is a vector of attributes and there are two parameters mean (μ) and the covariance matrix (Σ). The mean and covariance matrix should be calculated from the attributes of the non-defected modules. The mean can be calculated for each attribute using equation (2). The covariance matrix, for m number of instances, can be calculated using equation (6).

$$\Sigma = \frac{1}{m}\sum_{k=1}^{m}(x_i - \mu)\,(x_i - \mu)^T \tag{6}$$

In the approach presented, unlike univariate Gaussian distribution, in multivariate, the $p(x)$ for a module is calculated directly. If the $p(x)$ is less than a calculated threshold value e, it is flagged as an anomaly.

Because multivariate Gaussian distribution takes into account the covariance matrix rather than simple variance, it can capture correlation, as stated before. This can result to a better performance compared to the univariate model because the attributes could be correlated.

4. Experimental Setup

We implemented our framework and evaluated it using publicly available datasets. We used static code attributes and followed the attribute selection scheme from a state-of-the-art technique. To determine threshold, we used cross-validation set and used the threshold that gives best performance. These are elaborated in the following subsections.

4.1. Data Preparation

For the experiments, datasets come from several NASA projects and are available at the PROMISE Repository (*http://openscience.us/repo/defect/*) of Software Engineering Databases. Those projects have been developed in C, C++ and Java programming languages. The data is collected from various projects including spacecraft instrument, storage management, flight software, scientific data processing and real time projects. The collection of those publicly available datasets is useful for creating predictive software models. There were 10 datasets but we used only 7 datasets as other 2 datasets are in different formats and one dataset was unavailable from the repository. The datasets along with their various properties are shown in Table 1.

For anomaly detection algorithms, it is important that non-defected classes are used for train dataset. This is the reason there are no defected classes in the train data. To prepare data, the defected classes are separated from the non-defected classes. Next, the data is divided into three classes by selecting from the non-defected class randomly and then from the defected class randomly. The algorithm was run multiple times to ensure that output is not biased. The three classes are:

- Train data contains 60% non-defected randomly picked modules.
- Cross-validation data file has 50% of rest of the non-defected modules picked randomly and 50% defected modules picked randomly.
- Test data contains 50% of rest of the non-defected modules picked randomly and 50% defected modules picked randomly.

Table 1. Datasets Used for the Experiment

System Type	Language	Dataset	Total LOC	#Modules	% Defected
Spacecraft Instrument	C	cm1	17K	506	9
Storage Management for ground data	Java	kc3	8K	456	9
Database	C	mw1	8K	404	7
Flight software for orbiting satellite	C	pc1	26K	1108	6
“	“	pc2	25K	5590	0.4
“	“	pc3	36K	1564	10
“	“	pc4	30K	1458	12

4.2. Attribute Selection

Selecting good attributes is an important step. Any machine learning technique requires proper identification of attributes that can be used to train the learning framework. Initially, we used all attributes for predicting defects but we found that the results can be further improved. In paper [11], the authors had addressed this issue. They found that using too many attributes that are not relevant or are dependent may lead to reduced performance. Dependent attributes are an indication that the attribute may be redundant i.e. one attribute may be highly correlated with another and the use of one attribute in the predictor already accounts for the other dependent attribute. The authors in [11] showed that selecting a subset of the attributes give best results in predicting defects. They used *Exhaustive subsetting* and *Iterative subsetting* to select attributes. We followed their approach for selecting attributes. For our datasets, based on the work in [11], the selected attributes are listed in Table 2.

Table 2. Selected Attributes for Every Dataset

Datasets	Selected attributes
pc1	call_pairs, num_unique_operators, number_of_lines
mw1	error_est, node_count, num_unique_operators
kc3	loc_executable, level, prog_time
cm1	loc_comments, num_unique_operators, num_unique_operands
pc2	loc_comments, percent_comments
pc3	loc_blanks, content, number_of_lines
pc4	loc_blanks, loc_code_and_command, percent_comments

5. Result Analysis

To measure the performance of our proposed approach, we determined the accuracy, probability of detection (pd), probability of false alarm (pf) and balance. Accuracy is not a good measure for predicting defects. Most of the methods are likely to be non-defected and as such, an algorithm that classifies all methods as non-defected will still yield moderately high accuracy which is misleading [11]. Rather, balance, pd and pf are more standard measures for measuring the performance of a defect prediction scheme. The equation for finding balance is as follows:

$$\text{Balance} = 1 - \frac{\sqrt{(0-pf)^2 + (1-pd)^2}}{\sqrt{2}} \qquad (7)$$

Table 3. Results for Univariate Gaussian Distribution Using Selected Attributes

Dataset	Accuracy	*pd*	*pf*	Balance
mw1	78.33%	68.29%	17.39%	74.42%
cm1	74.39%	61.90%	21.31%	69.13%
pc1	75.63%	35.48%	14.73%	53.21%
pc2	57.79%	87.50%	43.84%	67.77%
pc3	64.84%	50.30%	26.46%	60.19%
pc4	56.52%	50.56%	41.41%	54.40%
kc3	78.00%	50.00%	6.25%	64.37%

For univariate Gaussian distribution, we found the accuracy, pd, pf and balance for selected few attributes that we found from our attribute selection step. They are shown in Table 3 for the various datasets. Here, average balance is 63.36% which is a good performance compared to existing state-of-the-art machine learning techniques. We observed the highest balance for the mw1 dataset. The dataset kc3 shows lowest pf and pc2 shows highest pd. To show that the selected attributes give optimal result, the most occurred attributes in the datasets along with the attributes selected previously and we calculate balance, pd, pf and accuracy for these attributes. They are presented in Table 4. The average balance is 61.96%. It is seen that the average balance has gone down so we can say we were right when we select the attributes following the method in [11].

Table 4. Results for Univariate Gaussian Distribution Using Most Occurred Attributes Along with Previously Selected Attributes

Dataset	Accuracy	*Pd*	*pf*	Balance
mw1	78.33%	64.29%	17.39%	71.91%
cm1	74.39%	61.90%	21.31%	69.13%
pc1	75.63%	35.48%	14.73%	53.21%
pc2	81.17%	50.00%	17.12%	62.63%
pc3	68.75%	64.18%	29.63%	67.13%
pc4	66.38%	39.33%	24.22%	53.81%
kc3	56.00%	55.56%	43.75%	55.90%

Previously, results were obtained for univariate Gaussian distribution. However, there exists correlation between attributes as shown in Table 7. univariate Gaussian distribution cannot account for the correlation. This is why we use multivariate Gaussian distribution with our selected attributes for the datasets. We determine the accuracy, pd, pf and balance. They are presented in Table 5. We can see a significant increase in balance for the multivariate Gaussian distribution. Here average balance is 69.06% which is much higher than that found in univariate method. This serves to strengthen the hypothesis that the high correlation between various attributes is a significant factor in predicting defects using anomaly detection algorithms. As a result, multivariate Gaussian distribution generally performs better than univariate Gaussian distribution method. Therefore we use multivariate Gaussian distribution to detect anomalies using the most occurred attributes in the datasets along with the selected attribute set. We find that the average balance has gone down. The results are shown in Table 6. Here average balance is 65.99%, which is lower than the average balance obtained when only selected attributes are used.

Table 5. Results for Multivariate Gaussian Distribution Using Selected Attributes

Dataset	Accuracy	*pd*	*pf*	Balance
mw1	73.33%	71.43%	26.09%	72.64%
cm1	65.85%	71.43%	36.07%	67.47%
pc1	60.00%	58.06%	39.53%	59.25%
pc2	82.47%	74.76%	16.44%	78.70%
pc3	78.13%	69.66%	18.52%	74.86%
pc4	69.28%	62.92%	28.52%	66.92%
kc3	74.00%	50.00%	12.50%	63.56%

It is seen that the balance is higher for multivariate Gaussian distribution compared to univariate Gaussian distribution. This is because the attributes are highly correlated as seen in Table 7. The correlation between certain attributes are presented in Table 8. The results strengthen our argument. However, we see that for *mw1, cm1* and *kc3* the balance for multivariate Gaussian distribution is lower than that of univariate Gaussian distribution. This is because there is less correlation between attributes in those datasets.

Table 6. Results for Multivariate Gaussian Distribution Using Most Occurred Attributes along with Previously Selected Attributes

Dataset	Accuracy	*pd*	*pf*	Balance
mw1	66.67%	71.43%	34.78%	68.17%
cm1	65.85%	71.43%	36.07%	67.47%
pc1	60.00%	58.06%	39.53%	59.25%
pc2	66.88%	75.00%	33.56%	70.41%
pc3	72.66%	71.64%	26.98%	72.32%
pc4	70.14%	62.92%	27.34%	67.42%
kc3	64.00%	44.40%	25.00%	56.92%

Table 7. Correlation between Attributes

Dataset	Attributes	Correlation
	loc_comments, num_unique_operators	50.88%
cm1	loc_comments, num_unique_operands	69.03%
	num_unique_operators, num_unique_operands	59.08%
	error_est, node_count	66.66%
mw1	error_est, num_unique_operators	63.56%
	node_count, num_unique_operators	55.09%
	loc_executable, level	-58.87%
kc3	loc_executable, prog_time	91.04%
	level, prog_time	-46.87%
	loc_code_and_command, num_unique_operators	78.72%
pc1	loc_code_and_command, number_of_lines	79.46%
	num_unique_operators, number_of_lines	83.56%
pc2	loc_comments, percent_comments	84.71%
	loc_blanks, loc_code_and_command	80.26%
pc4	loc_blanks, percent_comments	71.53%
	loc_code_and_command, percent_comments	68.05%
	loc_blanks, content	78.43%
pc3	loc_blanks, number_of_lines	63.67%
	content, number_of_lines	79.81%

Table 8 shows a comparative study of the results of our technique with that of other state-of-the-art techniques. We see that our approach generally performs better than other approaches where classifiers are used without any optimization or additional steps. And even for techniques that have higher performance, the difference is not very high. Out target was to determine whether anomaly detection algorithm would work for defect prediction and if it does, can it perform better than classifiers. We see that it does. Thus, we can say that defects can be considered as anomalies and anomaly detection algorithms can predict defects.

Table 8. Comparison of Balance for Different Datasets Using Different Approaches

		Dataset						
		pc1	pc2	pc3	pc4	kc3	mw1	cm1
Approaches	Multivariate Gaussian Distribution	59.25%	78.70%	74.86%	66.92%	63.56%	72.64%	67.47%
	Univariate Gaussian Distribution	53.21%	67.77%	60.19%	54.40%	64.37%	74.42%	69.13%
	Naive Bayes (selection of attribute with log filtering) [17]	70.40%	74.68%	72.32%	82.72%	75.29%	65.77%	68.00%
	CDDL (cost-sensitive discriminative dictionary learning) [13]	77.23%	---	74.38%	78.73%	68.40%	76.91%	68.02%
	Naïve Bayes [11]	61.32%	77.86%	71.50%	79.45%	70.46%	64.44%	71.98%
	Naïve Bayes + LOG+ FS [17]	67.50%	76.90%	74.80%	83.00%	71.30%	61.60%	67.80%

6. Conclusion and Future Work

Software defect prediction is continuously being explored as a software engineering research topic due to its importance. Various classifiers and machine learning techniques have been applied in recent literature that predicts defects in software using static code attributes. However, majority of these approaches are affected by imbalanced data. This type of data is regularly observed in real world software data and must be considered while predicting defects. Therefore, our objective was to propose an approach that should be unaffected by imbalanced data while predicting defects. It is established that Gaussian methods are appropriate for handling such data. We hypothesized that defected software modules will contain anomalous properties. Considering this approach, we experimented with NASA datasets collected from PROMISE repository and have shown that our approach produces better result compared to traditional classifiers without any optimization or filter. Therefore, defected software modules are representable as anomalies and can be used in this way for defect prediction. Furthermore, if we preprocess data with the goal of anomaly detection our approach may perform even better. Finally, whether or not anomaly detection can be considered while selecting attributes of software data to further enhance the performance can be explored in future.

Acknowledgment

The research conducted on this paper is funded by ICT Fellowship program from ICT Division, Government of People's Republic of Bangladesh. The award no. is 56.00.0000.028.33.079.17-223.

References

[1] Weyuker, E. J., Ostrand, T. J., & Bell, R. M. (2006, September). Adapting a fault prediction model to allow widespread usage. *Proceedings of the Second International Promise Workshop* (p. 1).

[2] Weyuker, E. J., Ostrand, T. J., & Bell, R. M. (2008). Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, *13(5)*, 539-559.

[3] Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*, *19(2)*, 77-131.

[4] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38(6)*, 1276-1304.

[5] Jiang, Y., Cuki, B., Menzies, T., & Bartlow, N. (2008, May). Comparing design and code metrics for software quality prediction. *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering* (pp. 11-18).

[6] Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data Analysis*. *6(5)*, 429-449.

[7] Andersson, C. (2007). A replicated empirical study of a selection method for software reliability growth models. *Empirical Software Engineering*, *12(2)*, 161.

[8] Fenton, N. E., & Ohlsson, N. (2000). Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, *26(8)*, 797-814.

[9] Zheng, J. (2010). Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, *37(6)*, 4537-4543.

[10] Challagulla, V. U., Bastani, F. B., Yen, I. L., & Paul, R. A. (2005, February). Empirical assessment of machine learning based software defect prediction techniques. *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005* (pp.

263-270). IEEE.

[11] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, *33(1)*.

[12] Yu, Q., Jiang, S. J., Wang, R. C., & Wang, H. Y. A feature selection approach based on a similarity measure for software defect prediction.

[13] Jing, X. Y., Ying, S., Zhang, Z. W., Wu, S. S., & Liu, J. (2014, May). Dictionary learning based software defect prediction. *Proceedings of the 36th International Conference on Software Engineering* (pp. 414-423).

[14] Arora, I., Tetarwal, V., & Saha, A. (2015). Open issues in software defect prediction. *Procedia Computer Science*, *46*, 906-912.

[15] Khoshgoftaar, T. M., Gao, K., & Szabo, R. M. (2001, November). An application of zero-inflated poisson regression for software fault prediction. *Proceedings of the 12th International Symposium on Software Reliability Engineering, 2001. ISSRE 2001* (pp. 66-73).

[16] Munson, J. C., & Khoshgoftaar, T. M. (1990). Regression modelling of software quality: Empirical investigation. *Information and Software Technology*, *32(2)*, 106-114.

[17] Khoshgoftaar, T. M., & Munson, J. C. (1990). Predicting software development errors using software complexity metrics. *IEEE Journal on Selected Areas in Communications*, *8(2)*, 253-261.

[18] Succi, G., Stefanovic, M., & Pedrycz, W. (2001). Advanced statistical models for software data. Department of Electrical and Computer Engineering, University of Alberta, Canada.

[19] Pickard, L., Kitchenham, B., & Linkman, S. (1999). An investigation of analysis techniques for software datasets. *Proceedings of the Sixth International Software Metrics Symposium*.

[20] Shepperd, M., & Kadoda, G. (2001). Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, *27(11)*, 1014-1022.

[21] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, (*4*), 308-320.

[22] Halstead, M. H. (1977). *Elements of Software Science* (Vol. 7, p. 127). New York: Elsevier.

[23] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321-357.

[24] Jacob, S., & Raju, G. (2017). Software defect prediction in large space systems through hybrid feature selection and classification. *International Arab Journal of Information Technology (IAJIT)*, *14*(2).

[25] Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering*, *37*(3), 356-370.

[26] Sharmin, S., Arefin, M. R., Abdullah-Al Wadud, M., Nower, N., & Shoyaib, M. (2015, December). SAL: An effective method for software defect prediction. *Proceedings of the 18th International Conference on Computer and Information Technology (ICCIT), 2015* (pp. 184-189).

[27] Fenton, N., Neil, M., Marsh, W., Hearty, P., Radlinski, L., & Krause, P. (2007, May). Project data incorporating qualitative factors for improved software defect prediction. *Proceedings of the Third International Workshop on Predictor Models in Software Engineering* (p. 2).

Kamrun Nahar Neela is a recently graduated student at the Institute of Information Technology, University of Dhaka in Bangladesh, where she completed her B.Sc. in Software Engineering in 2015 and then later completed her M.Sc. in software engineering in 2017. Her area of research during her time at IIT was in software quality, and in particular, defect prediction. She completed his internship from BRAC IT Services in Dhaka from January 2015-June 2015 where she worked on developing BRAC Bank's mobile applications.

Syed Ali Asif completed both his B.Sc. and M.Sc. in software engineering from the University of Dhaka, Institute of Information Technology in Bangladesh. His area of research during that time was software engineering, particularly in areas such as requirements prioritization as well as cloud computing. He completed his internship from Leeds Software Ltd in Dhaka in 2015. As of July, 2018, he is a graduate student at University of Delaware pursuing his PhD in Computer and Information Science. His current research interest lies in artificial intelligence and machine learning

Amit S. Ami received his B.Sc. in information technology (major in software engineering) from the Institute of Information Technology, University of Dhaka, Bangladesh in the year 2012. He received his M.Sc. in software engineering from the same university in year 2014. He worked for several years in industry as a software engineer. He was also a Microsoft student partner and a ACM student member. Additionally, he worked as the organizer with UX Saturday, Google Developer Group, Bangladesh and Mozilla Bangladesh. He joined as a lecturer at the Institute of Information Technology, University of Dhaka in the year 2014 and is currently working there. His research interests include experimental software engineering, software development life cycle, mobile application engineering and testing, and mining software repositories.

Alim Ul Gias completed M.Sc. in software engineering at Institute of Information Technology with thesis on Adaptive Software Testing. He received his bachelor's degree in information technology (major in software engineering) from the same institute. Alim completed his internship at Grameenphone Ltd., Bangladesh from July-December 2011. Currently he has been working as a lecturer at IIT DU since 16 Sept 2014. The broad domain of his research is automated software engineering. To be more specific, he likes to address the challenges which are involved in automating any steps (requirements engineering, software design, software testing, etc.) of the software development life cycle (SDLC).