Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 8, August 2018, pp. 1695-1704
DOI: 10.23940/ijpe.18.08.p5.16951704

Data Complexity Analysis for Software Defect Detection

Ying Ma[a,b,*], Yichang Li[a], Junwen Lu[a], Peng Sun[c], Yu Sun[d], and Xiatian Zhu[e]

[a]*Xiamen University of Technology*, *Xiamen*, *361024*, *China*
[b]*Engineering Research Center for Software Testing and Evaluation of Fujian Province*, *Xiamen*, *361024*, *China*
[c]*University of Electronic Science and Technology of China*, *Chengdu*, *610054*, *China*
[d]*Xiamen Institute of Software Technology*, *Xiamen*, *361000*, *China*
[e]*Queen Mary*, *University of London*, *London and E1 4NS*, *UK*

Abstract

Most researchers conduct defect detection under the assumption that the training and future test data must be in the same feature space and the same distribution. However, in the practical applications, data sets come from different domains and different distributions. Sometimes, local data in the target projects are limited and data are usually affected by noise. In these cases, the performance of the software defect detection model is uncertain. Firstly, we introduce the data complexity concept into the software engineering from data mining field. Secondly, we investigate the data complexity measurement on public software data sets to find out which complexity metric is appropriate to apply in defect detection. Finally, we analyze the relationship between complexity metrics and model performance to gain valuable insight into the effects of data complexity on defect detection. We are optimistic that our method can provide decision-making support for detection model management and design.

Keywords: defect detection; data mining; data complexity; software metric; software quality assurance

(Submitted on May 11, 2018; Revised on June 20, 2018; Accepted on July 26, 2018)

1. Introduction

Effectively controlling and minimizing the defects in software engineering is particularly urgent. However, eliminating defects not only requires human and financial resources, but also requires large effort in discovering defects in advance. To make matters worse, the accumulation and transmission of defects will lead to deterioration of software reliability and over-expenditure of software development costs. This is an important cause of the software crisis. Therefore, we must find the high-risk components in software engineering as soon as possible to ensure software quality. In 2015 and 2017, the SANS research organization identified the top 25 most common and dangerous software defects. These defects can be divided into three categories: interaction, resource management, and vulnerability protection [1]. In September 2016, there was a major bug in the United Tickets website, which resulted in very low ticket prices, caused a large number of internet users to purchase tickets, and brought serious economic losses to the company. In July 2011, a rear-end event occurred on the train of the EMU in Wenzhou. After investigation, it was found that there were serious defects in the signal equipment of the Wenzhou South Station. In February 2016, Google's Gmail failure prevented the user from accessing the mailbox for hours, due to the hidden software defects of the e-mail security unit.

With the increasing quality demands of software systems, more and more researchers are interested in software defects detection techniques. More than thirty of the largest commercial software companies have contributed to this research, such as Apple, Aspect Security, Breach Securit, CERT, Homeland Security, Microsoft, MITRE, Oracle, and Red Hat. Defect detection aims to prevent the faults that will appear in the later stage of software development and improve the software maintenance plan. Moreover, the defect detection effectively takes into account the safety and economic benefits from the application of the technique in the software development process. Most recently, software defect detection technology is

* Corresponding author.
E-mail address: maying@xmut.edu.cn

indispensable in improving software quality and reliability. With the disclosure of software historical data, more and more researchers have begun to study software defect detection models and have achieved encouraging research results [2-5].

The data complexity measurement is an important research field in data mining, but it is rarely investigated in software defect detection. We try to exploit the data complexity to provide effective information for the detection model, since the performance of the classifier depends on the data attributes, and the data attributes can be obtained through the analysis of the data complexity measurement. Different to prior works, we introduce three group complexity metrics to find the usability of data complexity analysis through investigation the different metric values on different software data sets. Then, we determine the useful metrics for defect prediction by analyzing the relation between the complexity metrics and detecting performance values. This empirical study shows that the performance of defect detectors built on different data sets are related to the complexity metrics. Studying the relationship between data complexity metrics and detection algorithm performance provides powerful support for establishing a stable detection model.

This paper is divided into five parts. Section 1 provides an introduction into the research background and main research contents of this paper. Section 2 briefly reviews the related work of software defect detection and data complexity measurement. Section 3 presents data complexity metrics and performance metrics used in our investigation and the analysis procedure for defect detection. Section 4 describes the software defect data sets and shows the experimental results with analysis of the complexity metrics and detection performances. Section 5 summarizes the main results of this paper and discusses further future research.

2. Related Work

2.1. Software Defect Detection

The general process of software defect detection is shown in Figure 1. The specific process of an example for building detection models is as follows:

- Extract data set: The features of software modules are generally selected, such as complexity, structural dependencies, keywords, and so on. Then, combine labels with example features to form training sets. Samples correspond to the software modules with defective or non-defective labels.
- Create a detection model: Use machine learning algorithms, such as support vector machines (SVMs) or Bayesian networks, to build detection models on the training set. The detection model can be tested for new samples to detect defects.
- Detection and Evaluation: The assessment of the detection model requires testing samples by comparing the output of the detection model to the real labels of the test data. The typical representatives of the evaluation separate the training set and test set with 10-fold cross-validation.

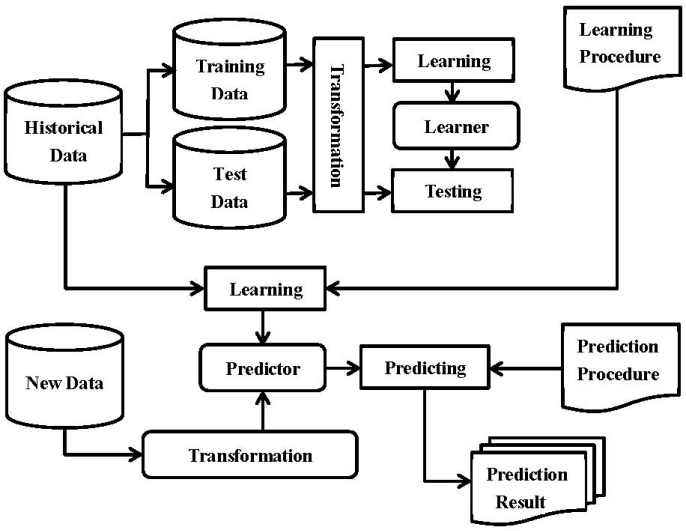


Figure 1. Software defect detection procedure

Various studies have shown that it is effective to describe the software defect detection model through the effective information provided by the software static code attributes [2-7]. However, the complexity of the data reflecting the geometric distribution of the data set has a major effect on the class separability of the data sets [8-10]. The metric can be used to detect in advance whether the defect detection model will have reliable performance. To build a detection model, it is necessary to collect historical data from the software system as a learning set. There is a gap between the collected learning set and the adaptability of the detection model, since it is necessary to use another data set to test its detection performance. Therefore, the performance of the model should be estimated by comparing the real defects with the detected defects in the test set.

2.2. Data Complexity Measurement

In machine learning, the performance of learning algorithms depends on parameters and training data. However, without fully understanding the significance of learning data, most of the algorithms are focused on adjusting the parameters of the model to improve accuracy. Therefore, they only provide information for the collection of information in the learning algorithm. It is impossible to obtain information on error classified examples without mentioning the reason for misclassification. Although some researchers have come up with some speculations, they have not specifically investigated them. Based on this situation, some researchers have proposed that the complexity of the data can be analyzed to determine the reason for the incorrect classification of some certain samples.

The pioneering work on data complexity includes the complication of some binary classification problems and the collection of three types of metrics: class overlap metrics, separability metrics, geometry, topology, and manifold density metrics [11]. However, there are also extensions based on this in other literatures. For example, in the work of Singh [12], data complexity metric evaluation and two new data complexity metrics are proposed. Many researchers use these metrics to solve different data mining problems. For example, Baumgartner and Somorjai [13] have defined special metrics for regular linear classifiers. Other researchers try to use these metrics to explain the performance of the learning algorithm and optimize the decision tree created under the binary data set or to analyze the model obtained when fuzzy UCS and application data flow are analyzed [14]. Measurements of data complexity have also been used in other related fields, such as gene expression in bioinformatics. The current research on the complexity of data is mainly comprised of two aspects. The first aspect is to create appropriate questions for a given classification algorithm, using only data features, and then decide on areas for its ability to adapt. The recently published articles of this research, such as Luengo and Herrera's article, for the first time proposed the determination of individual classifier adaptability [14]. Similarly, Saez et al. studied the complexity effects of nearest neighbor classifier data [15]. Luengo and Herrera analyzed the relationship between regions of the same classifier's ability to adapt, indicating that the classifier can use the same source of data complexity [16]. The data complexity metrics in this paper are mainly derived from research [15], including class overlap metrics, class separability metrics, geometry, topology, and manifold density metrics.

3. Data Complexity Measure for Software Data

The software module static metric data [2, 6] is commonly used, such as Line of Code (LOC), class weight (WMC), inheritance tree length (DIT), and class response level (RFC), as shown in subsection 5.1. At present, there are very few works that emphasize the complexity of software data itself. We aim to investigate the relation between complexity of software data sets and performance of defect detection. The detail of the analysis procedure is shown in Figure 2. First, we introduce the complexity metrics.

3.1. Class Overlap Metrics

Class overlap metrics are mainly used to evaluate the effectiveness of a single feature space in a separate class or the composite effect of multiple feature spaces.

F_1: The maximum Fisher discriminant rate refers to the discrimination rate that can better distinguish the two class attributes. The calculation formula is as follows:

$$c(u) = \max_{1\cdots d} \frac{(\mu_{i,N} - \mu_{i,P})^2}{\sigma_{i,N}^2 + \sigma_{i,P}^2} \tag{1}$$

Where d is the number of attributes, and $\mu_{i,N}$, $\mu_{i,P}$, $\sigma^2_{i,N}$, $\sigma^2_{i,P}$ are the means and standard deviation of attribute i in defect-free modules and defective modules, respectively.

F_2 : The volume of the overlapping area, i.e., the number of overlapping features of the two classes. The maximum value and minimum value of the feature f_i of the data set of class C_j are max(f_i;C_j) and min(f_i;C_j), respectively. The metric formula for the software defect datasets is:

$$F_2 = \prod_{1\cdots d} \frac{\min\max x_i - \max\min x_i}{\max(f_i, C_N \bigcup C_P) - \min(f_i, C_N \bigcup C_P)} \tag{2}$$

F_3 : Maximum feature efficiency, the largest percentage point that can be distinguished after clearing points that have been determined to fall outside the feature overlap area.

3.2. Separability Metrics

The class separability measure proposes class-independent indirect properties. Suppose a class is composed of a single or multiple manifolds, that is, the probability distribution of a given class is supported. According to the interdependence of distribution, location, and manifold, the two classes can be appropriately separated, but the separation is not described in the way of design.

L_1 : Minimize the sum of error distances by linear programming. L_1 is the value of an objective function that formulates a linear classifier that minimizes the sum of the error distances by formulating a linear programming approach. This method minimizes the sum of distances from misclassified points to classified hyper ellipsoids. The metric is normalized by the number of sample points of the classification problem and the length of the diagonal of the hyper-rectangular region that closes all training samples in the feature space.

L_2 : The linear classifier error rate through linear programming, i.e., the linear classifier error rate measured as L_1 with the training set.

N_1 : The percentage of another class is connected by means of the minimum spanning tree, i.e., N_1 is calculated from the minimum spanning tree that connects all the nearest neighbors. Then, the number of points connected to other classes is calculated through the edges of the spanning tree, and the calculated numbers are all considered as points at the edge of the class. N_1 represents the percentage of calculated points in the dataset.

N_2 : The average (internal) class nearest-neighbor distance ratio is calculated as:

$$N_2 = \frac{\sum_{i=0}^{m} \operatorname{int} ra(x_i)}{\sum_{i=0}^{m} \operatorname{int} er(x_i)} \tag{3}$$

Where m is the number of samples in the data set, $intra(x_i)$ the nearest neighbor distance in the class, and $inter(x_i)$ the nearest neighbor distance in other classes. N_2 compares the distance distribution of inner classes to other nearest neighbors. If the value is small, it means that the examples of the same class in the feature space are close to each other, and conversely if the value is larger, the examples in the same class are relatively scattered.

N_3 : The classifier error rate. Use the leave-one-out verification method to estimate the error rate of the nearest neighbor classifier. N_3 shows the closeness of the example between different classes. If the metric is small, it means that there is a large difference in the distance in the edge of the class.

3.3. Geometry, Topology and Manifold Density Metrics

To a certain extent, geometric metrics, topological metrics, and popular density metrics are used to estimate whether two classes are separable by detecting the existence and distribution of class boundaries. The role of a single feature space is to combine and organize the same marks, usually distance metrics rather than separate evaluations.

L_3 : Non-linear metrics in linear classifiers trained by linear programming: Hoekstra and Duin propose a classifier nonlinear metric with respect to a given data set. A test set is constructed by setting linear interpolation (random coefficients) between these points, given a training set and a random extraction point in the same class. Then, the BER of the classifier (trained from a given training set) is measured on this test set.

N_4 : The non-linear metric of the 1-NN classifier, that is, the error of the nearest neighbor classifier. This metric is used to adjust the nearest neighbor boundary with an interval distribution or overlapping problems between convex shell classes.

T_1 : The ratio of the number of hyperspheres. E-fields are given by the total number of points: E-field pre-topology describes the local aggregation of bit-error rates in the point set. The E-field through the hypersphere method can cover the example space. A hyperspherical list covering two classes consists of a comprehensive description of the class. The number and size of hyperspheres represent the number of points that tend to cluster in the hypersphere or the distribution in the shell structure. The remaining hyperspheres are smaller on the problem of being closer to other classes than their own. T_1 is a standard number of hyperspheres retained by the total number of points.

Algorithm 1 Data complexity analysis for defect detection procedure

Require:

The sets of complex metrics, $C = \{F_1, F_2, F_3, N_1, N_2, N_3, N_4, L_1, L_2, L_3, T_1\}$;

The software defect data sets,$D = \{CM1, JM1, KC1, KC2, KC3, MW1, PC1, PC2, PC3, PC4, PC5\}$;

The sets of machine learning algorithms, $A = \{KNN, C4.5, Naive\ Bayes\}$;

The sets of performance metrics, $P = \{TP\ Rate, FP, Rate, Precision, Recall,$ F-Measure, $ROC\ Area\}$;

Ensure:

for each data set D_i from D **do**
  for each algorithm a_j from A **do**
    Construct detectors m_{ij};
    for each performance metric p_k from P **do**
      Using the p_k to evaluate the m_{ij}, obtain the value v_{ijk};
    end for
    Compose the value vector v_{ij};
    for each complex metric c_s from C **do**
      Analyse the correlation for the v_{ij};
    end for
  end for
end for

Figure 2. Software defect detection procedure

3.4. Performance Metrics

To estimate the performance of the detection model, we use the detection classifier confusion matrix in WEKA [17], as shown in Table 1. The specific meanings of a, b, c, d in the table are explained as follows:

a: The actual defective module is detected as the number of defective modules.
b: The actual defective module was detected as the number of modules without defects.
c: The actual non-defective module is detected as the number of defective modules.
d: The actual non-defective module is detected as the number of defective modules.

Table 1. Detection model confusion matrix

Software module	Detection defect	Detection without defect
Real defective module	a	b
Real non-defective module	c	d

The six performance metrics [18] used in this paper are commonly used in defect detection. They are TP Rate, Precision, Recall, F-Measure, and ROC Area. TP Rate: Rate that is correctly classified as defective and has the same value as recall.

$$\text{TP Rate} = \frac{a}{a+b} \tag{4}$$

FP Rate, the ratio of misclassifications, that is, the actual module is defective, but it is detected as the probability of no defects.

$$\text{FP Rate} = \frac{b}{a+b} \tag{5}$$

Precision, the precision rate, defines the probability of an actual defective module for the number of modules detected as defective.

$$\text{Precision} = \frac{a}{a+c} \tag{6}$$

Recall, the feedback rate that equals the TP Rate, is an integrity measure that defines the ratio of actual defective modules compared to the total defective module. F-Measure, an evaluation method combining the feedback rate with the precision rate, is defined as the harmonic mean of the precision rate and the feedback rate. The formula is as follows:

$$\text{F-Measure} = \frac{(\alpha+1)\times \text{recall} \times \text{precision}}{\text{recall} + \alpha \times \text{precision}} \tag{7}$$

This value is generally greater than 0.5. The closer to 1, the better the performance of the corresponding detection model. This article uses a 10-fold cross-validation method. That is, the average data set is divided into ten parts, of which nine are for training and one is for testing. The average value of each result is used as the accuracy estimate of the algorithm. The advantage of this method is that all samples are used as the test set training set, and each sample will be verified once.

4. Experiment

4.1. Data Set

All of the data sets in this paper come from NASA's data set, where eleven subsystems were selected as experimental data sets, and the data sets can be obtained from public data PROMISE [19], as shown in Table 2. All the complexity features extracted from the original software provide data support for inspection, as shown in Table 3.

Table 2. Software datasets

Project	Examples	%Defective	Description
cm1	498	9.83	Spacecraft instrument
jm1	10878	18.50	A zero gravity experiment
kc1	1212	26.00	Storage management
kc2	522	20.49	Storage management
kc3	458	9.38	Storage management
mc2	161	32.30	Video guidance system
pc1	1109	6.94	Flight software
pc2	745	2.85	Flight software
pc3	1077	12.4	Flight software
pc4	1458	12.2	Flight software
pc5	17186	3.1	Flight software

Table 3. All features in NASA datasets

Type	#	Feature
Loc	5	Halsteads count of blank lines; McCabes line count of code; Halsteads line count; Halsteads count of lines of comments; line count of code and comment
McCabe	3	cyclomatic complexity; essential complexity; design complexity
Halstead	12	unique operators; unique operands; total operators; total operands; total operators and operands; volume; program length; difficulty; intelligence; effort; volume on minimal implementation; time estimator
BranchCount	1	branch count
Others	18	global data complexity; cyclomatic density; decision count; decision density; global data density; essential density; design density; loc executable; parameter count; percent comments; normalized cyclomatic complexity; modified condition count multiple condition count; node count; maintenance severity; condition count; global data complexity; call pairs; edge count

4.2. Experimental Results

This subsection verifies whether the complexity measures F_1, F_2, F_3, N_1, N_2, N_3, N_4, L_1, L_2, L_3, and T_1 are effective in describing the performance of defect detector on different datasets. Experimental results show that among the eleven complexity metrics, compared to the performance of data mining algorithms, L_2 is the most effective metric in describing the difficulty of detection, as shown in Table 4. Using this data complexity, you can better understand the relationship between data and detectors. This experiment is divided into the following steps:

Obtain the data complexity measurement of data mining algorithms and use the 10-fold cross validation method to estimate the accuracy of the detector. The selected classification method is a part of the classical algorithm, including K-NN (K=1) algorithm, C4.5 algorithm, Naive Bayes algorithm, and the obtained performance evaluation index value of the algorithm is analyzed to select an appropriate index.

Then, calculate the values of the measures F_1, F_2, F_3, N_1, N_2, N_3, N_4, L_1, L_2, L_3, and T_1 according to the calculation formula of the data complexity measure, analyze the data complexity measure, and filter the effective data complexity measure. According to the data metrics obtained in the previous two steps, draw trend graphs and analyze the relationship between them. To further verify the accuracy of the experiment, calculate the correlation coefficient between metrics based on statistical knowledge and draw relevant conclusions.

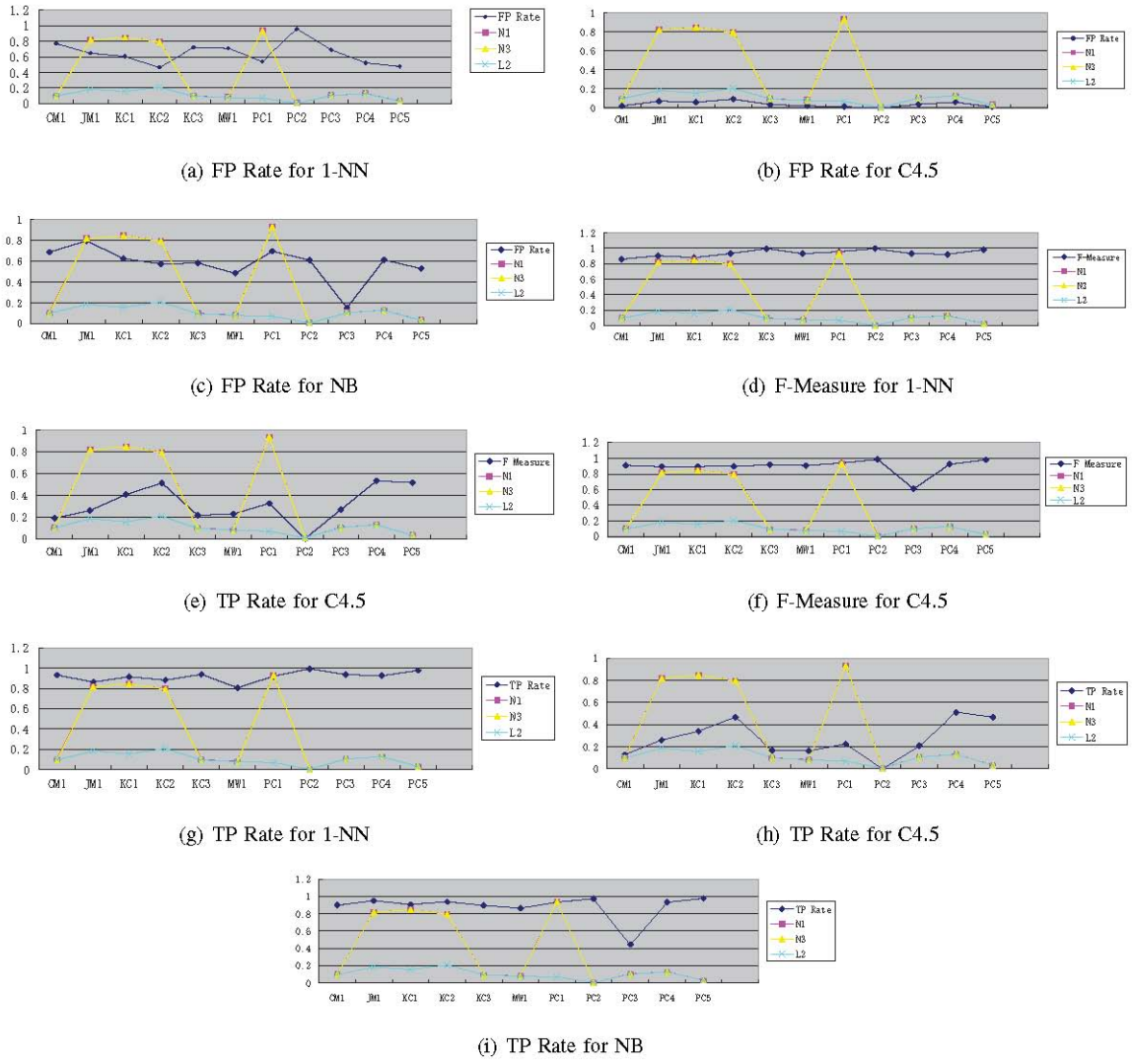
Since most of the metrics we used are defined as classification problems between two classes (defective and non-defective), the data sets need to be binarized. The values of data complexity metrics for the eleven metrics, F_1, F_2, F_3, N_2, N_4, L_1, L_3, and T_1 are the same for all the data sets. This means that they should be discarded because they are not representative. However, in the 11 data sets, the data complexity metrics N_1, N_3, and L_2 have more significant changes in the value range, which facilitates the analysis of the results. In order to discuss the relationship between the algorithm performance evaluation index, and data complexity measure, for each performance evaluation metrics of TP Rate, FP Rate, and F-Measure, the curve of complexity metrics N_1, N_3, and L_2 has been described corresponding to different algorithms. This is shown in Figure 3.

Table 4. The complexity value on each data set

Data set	CM1	JM1	KC1	KC2	KC3	MW1	PC1	PC2	PC3	PC4	PC5
F_1	0	0	0	0	0	0	0	0	0	0	0
F_2	1	1	1	1	1	1	1	1	1	1	1
F_3	0	0	0	0	0	0	0	0	0	0	0
N_1	0.099	0.817	0.845	0.795	0.096	0.079	0.932	0.004	0.103	0.128	0.03
N_2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
N_3	0.097	0.817	0.845	0.793	0.094	0.077	0.931	0.004	0.102	0.127	0.03
N_4	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
L_1	1	1	1	1	1	1	1	1	1	1	1
L_2	0.097	0.183	0.155	0.207	0.094	0.077	0.069	0.004	0.102	0.127	0.03
L_3	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
T_1	1	1	1	1	1	1	1	1	1	1	1

The curve trend shows the characteristics of the three data complexity metrics and the performance evaluation metrics of the algorithm changing with the data set. The relationship between them can be intuitively and clearly analyzed according to the curve trend. A comprehensive analysis of the nine maps shows that relative to the data complexity metrics N_1, N_3, L_2, and the algorithm performance evaluation metrics are changed together. Therefore, the three data complexity metrics

can be used to demonstrate the separability of data sets, thereby improving the process of building a software defect detection model. Although the trend of the metric L_2 and performance are not similar in these nine charts, L_2 is still a good metric of data complexity in the software defect detection field.



(a) FP Rate for 1-NN (b) FP Rate for C4.5

(c) FP Rate for NB (d) F-Measure for 1-NN

(e) TP Rate for C4.5 (f) F-Measure for C4.5

(g) TP Rate for 1-NN (h) TP Rate for C4.5

(i) TP Rate for NB

Figure 3. The curve of detection performance and complexity metrics

4.3. Experimental Analysis

To further study the relevance of complexity metrics and performances of the models, we calculate the correlation coefficient according to Equation (8). The specific values are shown in Table 5.

$$r=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{8}$$

It can be seen that the absolute value of the correlation coefficient between the corresponding performance evaluation index F-Measure and the data complexity measure L_2 in the 1-NN algorithm reaches 0.8. They have a high degree of correlation. For the evaluation indicator F-Measure, all three algorithms show that the absolute value of the correlation coefficient between this indicator and the data complexity measure L_2 is greater than the values of N_1 and N_3. According to the definition of the algorithm performance evaluation index, F-Measure is the average of the precision rate and the

feedback rate, and it is more representative. Comprehensive analysis shows that the absolute value of the correlation coefficient between each algorithm performance evaluation index and data complexity measurement index is greater than 0. Therefore, it can be concluded that there is a correlation between the performance evaluation index of the algorithm and the complexity of the data, especially with the L_2 metric using the 1-NN method.

Table 5. The absolute value of the correlation coefficient between the model performance and data complexity metrics

Algorithm	Complexity metric	F-Measure	FP Rate	TP Rate
1-NN	N_1	0.458	0.205	0.283
	N_3	0.458	0.206	0.282
	L_2	0.687	0.808	0.711
C4.5	N_1	0.536	0.310	0.275
	N_3	0.535	0.310	0.275
	L_2	0.535	0.310	0.275
NB	N_1	0.445	0.058	0.221
	N_3	0.446	0.058	0.221
	L_2	0.228	0.323	0.039

5. Conclusions and Future Work

This study introduces the data complexity analysis method into the software engineering field for investigating the relationship between data complexity and performance of the software defect detection model. The experimental results show that the relationship between data distribution and the performance of defect detection model is significant. Data complexity metrics can provide effective information for the software data set and provide strong support for the establishment of stable detection models. The most significant data complexity metric is L_2.

Due to the inherent property of the software data sets, only N_1, N_3, and L_2 have usable values in the data complexity metrics on the data set studied. In the future, we will use more types of data to verify our findings. In addition, we may investigate other data complexity metrics to establish a more stable defect detection model.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61502404, 61672442), Natural Science Foundation of Fujian Province of China (Grant No. 2016J01326), Distinguished Young Scholars Foundation of Fujian Educational Committee (Grant No. DYS201707), International S&T Cooperation Program (Grant No. E201402000), and Open Fund of Engineering Research Center for Software Testing and Evaluation of Fujian Province. We thank the anonymous reviewers for their greatly helpful comments.

References

1. J. D. Strate and P. A. Laplante, "A Literature Review of Research in Software Defect Reporting," *IEEE Transactions on Reliability*, Vol. 62, No. 2, pp. 444-454, June 2013
2. T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on Software Engineering*, Vol. 33, No. 1, pp. 2-13, January 2007
3. M. X. Liu, L. S. Miao, and D. Q. Zhang, "Two-Stage Cost-Sensitive Learning for Software Defect Prediction," *IEEE Transaction Reliability*, Vol. 63, No. 2, pp. 676- 686, June 2014
4. Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer Learning for Cross-Company Software Defect Prediction," *Information and Software Technology*, Vol. 54, No. 3, pp. 248-256, March 2012
5. J. Ren, K. Qin, Y. Ma, and G. Luo, "On Software Defect Prediction using Machine Learning," *Journal of Applied Mathematics*, No. 3, pp. 201-211, 2014
6. P. A. Laplante and J. F. Defranco, "Software Engineering of Safety-Critical Systems: Themes From Practitioners," *IEEE Transactions on Reliability*, Vol. 99, pp. 1-12, September 2017
7. X. Yang, K. Tang, and X. Yao, "A Learning-to-Rank Approach to Software Defect Prediction," *IEEE Transactions on Reliability*, Vol. 64, No.1, pp. 234-246, March 2015
8. T. K. Ho and M. Basu, "Complexity Measures of Supervised Classification Problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 3, pp. 289-300, March 2002
9. Y. Ma, K. Qin, and S. Zhu, "Discrimination Analysis for Predicting Defect-Prone Software Modules," *Journal of Applied Mathematics*, No. 1, 2014
10. S. Wang and J. Wei, "Feature Selection based on Measurement of Ability to Classify Subproblems," *Neurocomputing*, Vol. 224, pp. 155-165, February 2017
11. M. Basu and T. K. Ho, "Data Complexity in Pattern Recognition," *Springer Science & Business Media*, 2006

12. S. Singh, "Multiresolution Estimates of Classification Complexity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 25, No. 12, pp. 1534-1539, December 2003
13. R. Baumgartner and R. L. Somorjai, "Data Complexity Assessment in Under-Sampled Classification of High-Dimensional Biomedical Data," *Pattern Recognition Letters*, Vol. 27, No. 12, pp. 1383-1389, September 2006
14. L. Morn-Fernndez, V. Boln-Canedo, and A. Alonso-Betanzos, "Can Classification Performance be Predicted by Complexity Measures? A Study using Microarray Data," *Knowledge & Information Systems*, pp. 1-24, October 2016
15. L. Morn-Fernndez, V. Boln-Canedo, and A. Alonso-Betanzos, "Centralized vs. Distributed Feature Selection Methods based on Data Complexity Measures," *Knowledge-based Systems*, Vol. 117, pp. 27-45, February 2017
16. J. Luengo and A. Herrera, "An Automatic Extraction Method of the Domains of Competence for Learning Classifiers using Data Complexity Measures," *Knowledge and Information Systems*, Vol. 42, No. 1, pp. 147-180, October 2015
17. I. H. Witten and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques," *Morgan Kaufmann*, 2005
18. M. Sokolova and G. Lapalme, "A Systematic Analysis of Performance Measures for Classification Tasks," *Information Processing & Management*, Vol. 45, No. 4, pp. 427-437, May 2009
19. G. Boetticher, T. Menzies, and T. Ostrand, "The PROMISE Repository of Empirical Software Engineering Data," *West Virginia University Department of Computer Science*, 2007