ISSN: 2664-9640

An Approach to Estimating the Number of Defects from Lines of Code, with Censored Poisson and Binomial Models

Xiaopeng Xu[1], Xiaochun Zhang[1,*], and Chen Li[2]

[1] School of Management Science and Engineering, Anhui University of Finance and Economics, Bengbu, Anhui, China

[2] Department of Materials and Production, Aalborg University, Aalborg, Denmark

[*]Corresponding Author

Abstract

Software defect prediction plays a vital role in ensuring software quality and optimizing resource allocation in development projects. However, the defect density, measured as defects per thousand lines of code (KLOC), is often not directly observable. Instead, software development teams typically record the presence of defects as a Boolean value, indicating whether a module has one or more reported defects. This paper proposes a novel method to estimate the number of defects in a software project using probability models and censoring techniques, based on the available lines of code (LOC) and defect presence data. We investigated Poisson and Binomial models, implemented with different tools such as JAGS and Metropolis, and obtained consistent results across the models. The experimental results show the proposed approach can provide valuable insights for project managers, enabling them to allocate resources effectively and prioritize quality assurance activities, ultimately leading to improved software reliability and customer satisfaction.

Keywords

Software Defect Estimation; Poisson; Binormal; Promise.

1. Introduction

In the realm of software development, defect prediction is a critical task that directly impacts project success, resource allocation, and product quality. Traditionally, recording detailed defect information for each software module can be a time-consuming and labor-intensive process. As a result, many software development teams opt to simplify data collection by recording only the presence of defects, represented as a Boolean value indicating whether a module has one or more reported defects. While this approach streamlines data collection, it poses challenges for accurate defect prediction due to the inherent diversity in module attributes, such as lines of code (LOC).

Existing defect prediction techniques often rely on classification algorithms based on machine learning, which aim to categorize modules as defective or non-defective. However, these algorithms may not provide quantitative measures like defects per thousand lines of code (KLOC), which are crucial for effective project management and quality assurance. Moreover, the imbalanced nature of defect datasets, where the majority of modules are non-defective, can hinder the performance of classification algorithms.

To address these challenges, we propose a novel method that estimates the number of defects in a software project using probability models and censoring techniques. By treating the number of defects as a latent variable following a probability distribution, we establish a relationship between the observed Boolean value (i.e., the presence of defects) and the

ISSN: 2664-9640

underlying defect count using censoring. This approach enables the estimation of defect density and the prediction of defects in untested code, providing valuable insights for project managers and quality assurance teams.The proposed method offers several advantages over existing defect prediction techniques. First, it allows for the estimation of quantitative measures like defects per KLOC, which are more informative than binary classifications. Second, the method can handle the imbalanced nature of defect datasets by leveraging probability distributions and censoring. Finally, the approach is flexible and can be implemented using various tools and algorithms, ensuring its adaptability to different software development environments.

In this paper, we present the theoretical foundation of the proposed method and demonstrate its application in a real-world software development scenario. We investigate Poisson and Binomial models for defect estimation and compare their performance using different implementation tools. The results highlight the consistency and effectiveness of the proposed approach in predicting defect density and estimating the total number of defects in a project.

The remainder of the paper is organized as follows: Section 2 provides an overview of related work in software defect prediction. Section 3 describes the proposed method, including the probability models and censoring techniques used. Section 4 presents the experimental setup and results, discussing the findings and their implications for software development projects. Finally, Section 5 concludes the paper and outlines future research directions.

2. Related Work

To improve software quality and reliability, many software defect prediction models had been emerged for the past decades [1-5]. However, the appropriate method that ensures the high quality of software production still remains an open question. The existing works can be divided into two parts: machine learning based and deep learning based.

Peng et al. combine various measures to evaluate the quality of classifiers for software defect prediction. The experiment used 13 classifiers on 11 software defect datasets, and the results indicate that SVM, C4.5 and KNN were the top algorithms [6]. Pendharkar et al. selected the appropriate inputs and learned a classification. The proposed hybrid exhaustive searchs and probabilistic neural networks worked well for the small scale datasets [7]. Tumar et al. develop an adaptive synthetic sampling approach based on binary moth flame optimization. For the PROMISE dataset, the proposed technique increases the prediction performance for various classifiers. The KNN algorithm wins the best runtime and the linear discriminant analysis gets the highest auc value [8]. Aquil et al. put forward a stacking classifier technique to build an efficient software defect prediction model. In the evaluation stage, 13 datasets were used to test different prediction methods, and had the best results of all the methods compared [9]. Turabieh et al. developed a classifier to address the software defect prediction problem. The work used 19 software defect datasets for evaluating the proposed method, and achieved a high accuracy performance [10].

Most early software defect prediction researches based on machine learning were involved in buggy and non-buggy, i.e. the binary classification problem. However, the information is useful but not enough. In addition, Features selection also a challenge for machine learning. Liang et al. introduced a Seml framework, which combines word embedding and LSTM model for software defect prediction. The results of the experiments show Seml outperforms three compared approaches on most of the datasets [11]. Phan et al. proposed a multi-view CNN based on labeled directed graphs. In order to adapt to dynamic structures for local regions of graphs, convolutional filters were designed for the model. The results show that the proposed model outperforms compared deep neural networks for malware analysis and software defect prediction task[12]. Song et al. identified the critical software defects based on neural networks algorithms. The work is an improved Elman neural network for software defect prediction. The

ISSN: 2664-9640

authors evaluated the proposed model on 7 PROMISE repositories projects. The results suggest that the improved Elman neural network model is prominent [13]. In [14], the authors present a combined approach for software defect prediction. The proposed neural network model with PCA feature reduction scheme improved the prediction performance. The experiments were implemented on NASA dataset. The results show that the proposed approach can provide better performance for software defect prediction. In [15], authors introduced a novel deep learning framework to predict the number of defects. In which, a module consisted of a set of software metrics, and predicted the amount of defects. The results illustrated that the proposed approach can improve software defect prediction performance.

The aforementioned literatures highlight two significant challenges: (1) estimating the number of defects if only the presence of defects is known; (2) evaluating the quality of parameter estimations. To overcome the above issues, we propose an estimation approach for software defects.

3. Methods

Without loss of generality, any datum of Promise dataset can be used as benchmark of this research, and CM1 is adopted here. It provides much information, such as "lines of code" (LOC), "complexity", or "module has/has not one or more reported defects" (i.e. the presence of defects). For the sake of simplicity, we investigate only LOC and the presence of defects, with which to estimate the number of defects.

First, the distribution of LOC is studied. Second, several models of defect density are proposed. With this information, the number of defects can be estimated.

3.1. Lines of Code

Lines of code are elementary information of any module. As shown in Figure 1, in CM1, there are 498 modules, from which only three have more than 350 lines, while most modules have fewer than 200 lines. Our first question will be: what is the distribution of LOC?

We have two options:

- Poisson Distribution
- Gamma Distribution

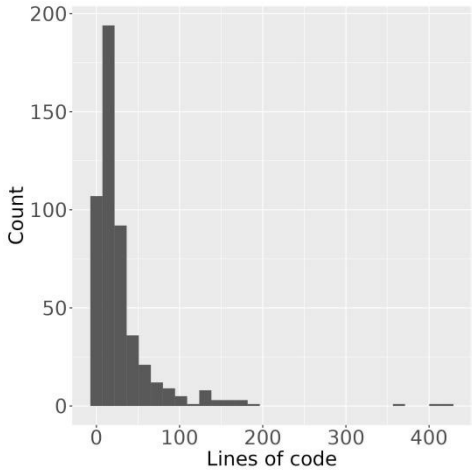

Figure1. Histogram of LOC

Poisson distribution (with parameter λ) seems appropriate for LOC, which is an integer.

$$Xi \sim Poisson(\lambda) \quad (1)$$

ISSN: 2664-9640

For a full Bayesian model, a distribution for λ is also defined, such as uniform over the positive real numbers:

$$\lambda \sim U(0,\infty) \tag{2}$$

Being an integer, LOC can instead be treated as positive real number following Gamma distribution.

$$Xi \sim Gamma(\alpha, \beta) \tag{3}$$

α and β follow uniform distribution:

$$\alpha, \beta \sim U(0,\infty) \tag{4}$$

3.2. Defect Density

Imbalance is a noticeable property of the data because most modules (90%) of CM1 are labeled with no defect. In other words, 90% of correctness can be assured by assuming no defect in the binary classification. Therefore, classification accuracy may be not a feasible measure for imbalanced datasets.

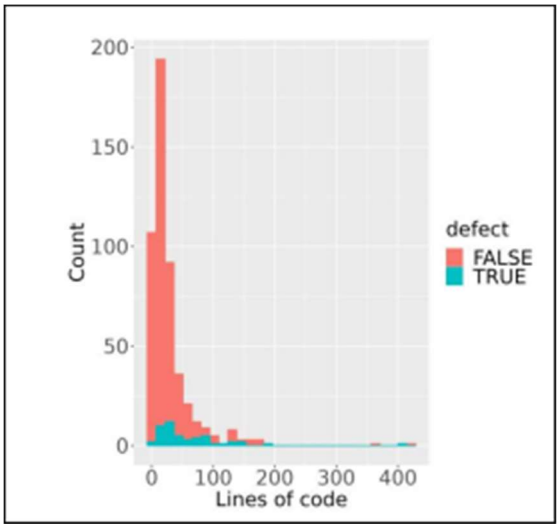

Figure 2. Histogram of LOC for defective and defectless modules

Knowing defective or not, as well as lines of code, we are actually interested in the number of defects. There is a relationship between the former and latter: defective implies that the number of defects is an unknown integer that is greater than one, and no defect means the number of defects is zero. The relationship is called censoring in the fields of reliability and survival analysis. With censoring, the number of defects can be inferred from the presence of defects.

As the number of defects is unknown, evaluating the fit of the model may be inappropriate. Our scheme is to assume Poisson or Binomial distribution for the number of defects, and to compare the two results.

- Poisson without intercept

The defect per line of code (defect density) is modeled with θ, which is shared by all modules. If the line of code of ith module is Xi, then the number of defects of ith module follows Poisson

ISSN: 2664-9640

distribution with parameter θXi. Also, if the lines of code Xi is zero, the number of defects will also be zero, i.e. having not intercepted.

$$Yi \sim Poisson(\theta Xi) \quad (5)$$

The number of defects Yi is an unobservable latent variable, and after censoring, Zi is observed. If Yi is zero, Zi will also be zero, otherwise, one.

- Poisson with intercept

Other factors, in addition to the lines of code, such as interface complexity, may also affect the number of defects. These factors can be modeled with an intercept c.

$$Yi \sim Poisson(\theta Xi + c) \quad (6)$$

If the lines of code is zero, the number of defects can be nonzero. Intercept can measure the contribution of other factors to the number of defects.

- Binomial Distribution

Binomial distribution can also model the number of defects. If there is Xi lines of code in the ith module, and the defective rate per line is θ, then the number of defects follows Binomial distribution.

$$Yi \sim Binomial(\theta, Xi) \quad (7)$$

The Binomial distribution can be approximated by Poisson distribution, if Xi is large and θ is small. In CM1 data, the median of Xi, which is only 17, may be not large enough. Therefore, we will compare the results of the two models. Freeing from the above approximation condition, binomial distribution has the limitation of omitting intercept.

4. Experiment

Two MCMC algorithms are used to estimate the parameters of probability models: 1) Gibbs Sampling and 2) Metropolis.

MHMC is used for Binomial model, while JAGS for other models. The ability of coping with censored data is a benefit of MCMC. All experiments adopt the same configuration:

- 3 chains
- 5000 iterations per chain
- 5000 burn-in

4.1. Lines of Code

As shown in Table 1, the λ parameter of Poisson distribution, which is the average lines of code, is about 30. The 3rd step of Bayesian analysis is to evaluating the fit of the model. As shown in Figure 3 of the quantile-quantile plots, the disparate between theoretical and empirical distributions is obvious. It suggests that the LOC does not follow Poisson distribution. For Poisson, the mean and variance should be equal, but in this case, the mean of LOC is about 30 and the variance is much larger than 30.

ISSN: 2664-9640

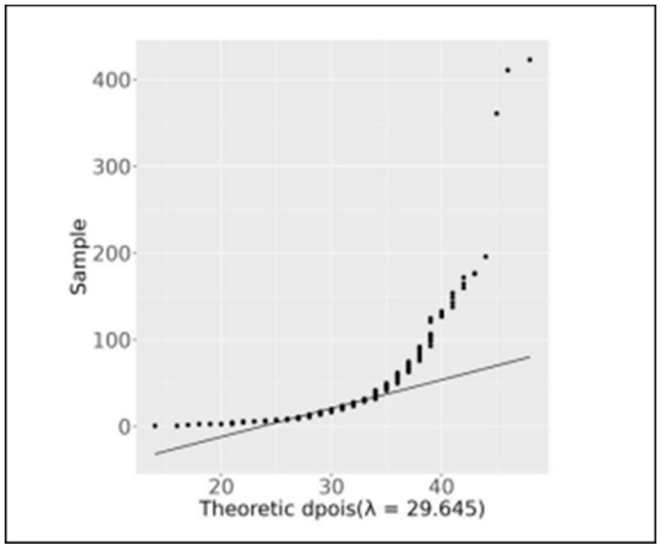

Figure 3. Quartile-quartile plots for Poisson model

Table 1. Parameter estimates for Gamma distribution

Parameter	Mean	SD	Time-series SE
λ	29.646482	0.243528	0.002498

Experimented with similar configuration for Gamma distribution, the shape parameter is about 1.080, and rate 0.036. The quantile-quantile plots (Figure 4) show that theoretical and empirical distributions are more comparable than their Poisson counterpart.

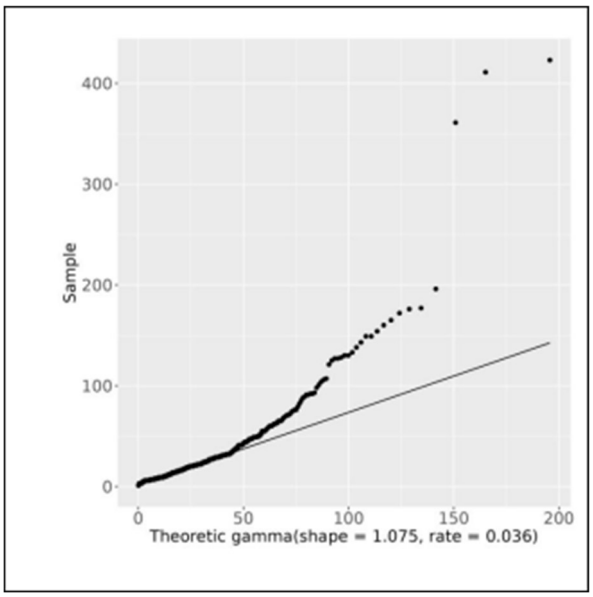

Figure 4. Quartile-quartile plots for Gamma model

4.2. The Number of Defects

Since LOC is observed, the estimated parameters of Gamma distribution, α and β, are similar to Table 1. In Table 2, the average defects per KLOC (thousands of lines of code), which is θ, is about 3 for CM1.

Above experiments are modeled with BUGS language, implemented with JAGS. The same models can also be transformed to potential energy and implemented using Metropolis algorithm. The advantage is flexibility.

ISSN: 2664-9640

Table 2. Poisson model without intercept

Parameter	Mean	SD	Time-series SE
α	1.079870	0.060870	1.337e-03
β	0.036508	0.002606	5.792e-05
θ	0.003394	0.000484	3.937e-06

Assuming the number of defects follows Binomial distribution with parameters θ and LOC, from Table 3, we find that the estimated α and β are similar to those in Table 1 and Table 2. The fact indicates that the results of different algorithms are consistent. The estimated θ is matched with Table 2, therefore, two different models can draw comparable results, that is, 3 defects per KLOC.

Table 3. Binomial model

Parameter	Mean	SD
α	1.07555	0.0609205
β	0.0362623	0.00256943
θ	0.00371736	0.000541352

In addition to LOC, other factors, such as interface complexity, may also contribute to the number of defects. The influence is modeled with the intercept, which signifies average number of defects for modules with zero line. As shown in Table 4, with intercept, the number of defects per KLOC drops to 2. As the estimated c is about 10 times to θ, the defects introduced by other factors are close to the effect of 10 LOC. As the median of LOC is only 17, other factors have notable influence on the number of defects.

Table 4. Poisson model with intercept

Parameter	Mean	SD	Time-series SE
α	1.079792	0.0598676	1.285e-03
β	0.036482	0.0025294	5.411e-05
c	0.023280	0.0137021	2.070e-04
θ	0.002676	0.0005887	7.591e-06

5. Discussion

This research investigates three models for the number of defects:

- Poisson without intercept
- Poisson with intercept
- Binomial

The tools are based on two algorithms:

- Gibbs
- Metropolis

For the distribution of LOC, Gamma fits better than Poisson.The fact that two algorithms arrive similar estimates for parameters of Gamma distribution α and β testifies the implementation of Metropolis algorithm. In all three models, the estimated parameters of Gamma distribution are similar. Because the LOC is actually observed, the modelling of number of defects and LOC is uncorrelated.

Both Binomial and Poisson models obtain 3 defects per KLOC, which may reflect the true rate. Only CM1 of Promise dataset is used; other datasets may have different results.The intercept is

ISSN: 2664-9640

10 times to the number of defects per LOC. Therefore, other factors than LOC can influence the number of defects. The specific contributing factors in Promise dataset are to be identified.

The experimental results demonstrate the effectiveness of the proposed method in estimating the number of defects using LOC and defect presence data. By investigating Poisson and Binomial models with different implementation tools, such as JAGS and Metropolis, we observed consistent results across the models. Both the Poisson and Binomial models yielded similar estimates for defect density, with the CM1 dataset showing approximately 3 defects per KLOC.

The inclusion of an intercept term in the Poisson model provided valuable insights into the influence of factors beyond LOC on the number of defects. The estimated intercept coefficient suggested that factors like interface complexity can have a significant impact on defect occurrence, with their contribution being comparable to the effect of 10 LOC. This finding underscores the importance of considering additional factors in defect prediction models to improve their accuracy and reliability.

While the proposed method has been applied to the CM1 dataset from the Promise repository, it is essential to assess its generalizability by evaluating its performance on other datasets. Future research should focus on validating the method across diverse software projects and domains to ensure its robustness and applicability in various contexts. Moreover, identifying and incorporating specific contributing factors from the Promise dataset, such as complexity measures or developer experience, could further enhance the predictive power of the models.

The proposed method offers several practical implications for software development projects. By providing quantitative estimates of defect density and the total number of defects, project managers can make informed decisions regarding resource allocation and quality assurance strategies. The ability to predict defects in untested code allows for proactive measures to be taken, such as targeted testing or code reviews, ultimately reducing the risk of defect spillover into later stages of the development lifecycle.

Furthermore, the proposed method can be integrated into existing software development processes and tools, enabling seamless defect prediction and monitoring. The consistency of results across different implementation tools demonstrates the flexibility of the approach, allowing development teams to choose the most suitable tools based on their specific requirements and expertise. However, it is important to acknowledge the limitations of the proposed method. The accuracy of defect predictions relies on the quality and representativeness of the input data, including LOC and defect presence information. Ensuring the reliability and consistency of data collection processes is crucial for obtaining meaningful results. Additionally, the method assumes that the relationship between LOC and the number of defects follows specific probability distributions, which may not hold true in all software projects.

6. Conclusion

In this paper, we proposed a novel method for estimating the number of software defects using probability models and censoring techniques, based on LOC and defect presence data. The method addresses the challenges associated with traditional defect prediction approaches, which often rely on binary classification and may not provide quantitative measures of defect density.

The experiment results show its effectiveness in predicting defect density and estimating the total number of defects in a project. The consistency of results across Poisson and Binomial models, implemented with different tools, highlights the robustness and reliability of the approach. Moreover, the proposed method contributes to the growing body of knowledge in software defect prediction research. The integration of probability models and censoring

ISSN: 2664-9640

techniques offers a fresh perspective on defect estimation, opening up new avenues for further exploration and refinement.

As the demand for high-quality software continues to grow, the adoption of such data-driven defect prediction techniques will become increasingly crucial for the success of software development projects. Future research should focus on validating the method across diverse datasets, incorporating additional contributing factors, and exploring alternative probability models to capture the complexity of software defects.

Acknowledgments

The authors would like to acknowledge partial support by the project of the Natural Science Foundation of the Education Department of Anhui Province (Grant No. KJ2020A0012). This study is also partially supported by EU-funded AI REDGIO 5.0 project (Grant Agreement ID: 101092069).

References

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Transactions on Software Engineering, vol. 38(2012)No.6, p.1276–1304.

[2] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G.Q. Zhang, Transfer learning using computational intelligence, Knowledge-Based Systems, vol. 80(2015)No. C, p.14–23.

[3] T. J. Wang, Z.W. Zhang, X. Y. Jing, and L.Q. Zhang, Multiple kernel ensemble learning for software defect prediction, Autom Softw Eng, vol.23(2016), p.569–590.

[4] X. Chen, D. Zhang, Y.Q. Zhao, Z.Q. Cui, C. Ni, Software defect number prediction: unsupervised vs supervised methods, Information and Software Technology, vol. 106(2019), p.161–181.

[5] Z. Xu, J. Liu, etc., Software defect prediction based on kernel PCA and weighted extreme learning machine, Information and Software Technology, vol.106(2019), p.182–200.

[6] Y. Peng, Gang. K, G.X. Wang, and H.G. Wang, Empirical evaluation of classifiers for software risk management, International Journal of Information Technology & Decision Making, (3009),p.749–767.

[7] Pendharkar PC, Exhaustive and heuristic search approaches for learning a software defect prediction model, Eng Appl Artif Intell, vol. 23(2010)no. 1, p.34–40.

[8] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction, IEEE Access, vol.8(2020), p.8041–8055.

[9] M.A.I. Aquil, Predicting software defects using machine learning techniques, International Journal of Advanced Trends in Computer Science and Engineering, vol.9(2020)no.4, p. 6609–6616.

[10] Turabieh H, Mafarja M, and Li X, Iterated feature selection algorithms with layered recurrent neural network for software fault prediction, Expert Syst Appl, vol. 122(2019), p.27–42.

[11] H. Liang, Y. Yu, L. Jiang, and Z. Xie, Seml: A semantic LSTM model for software defect prediction, IEEE Access, vol. 7(2019), p.83812–83824.

[12] A.V. Phan, M.L. Nguyen, Y.L. H. Nguyen, and L.T. Bui, DGCNN: A convolutional neural network over large-scale labeled graphs, Neural Networks, vol. 108(2018), p.533–543.

[13] K. Song, S.K. Lv, D. Hu, and P. He, Software defect prediction based on elman neural network and cuckoo search algorithm, Mathematical Problems in Engineering, (2021), p.1-14.

[14] R. Jayanthi, L. Florence, Software defect prediction techniques using metrics based on neural network classifier, Cluster Comput, vol. 22(2019) no.1, p. 77–88.

[15] L. Qiao, X.S. Li, Q. Umer, and P. Guo, Deep learning based software defect prediction, Neurocomputing, vol. 385(2020), p.100–110.