Cagatay Catal*

A Comparison of Semi-Supervised Classification Approaches for Software Defect Prediction

Abstract: Predicting the defect-prone modules when the previous defect labels of modules are limited is a challenging problem encountered in the software industry. Supervised classification approaches cannot build high-performance prediction models with few defect data, leading to the need for new methods, techniques, and tools. One solution is to combine labeled data points with unlabeled data points during learning phase. Semi-supervised classification methods use not only labeled data points but also unlabeled ones to improve the generalization capability. In this study, we evaluated four semi-supervised classification methods for semi-supervised defect prediction. Low-density separation (LDS), support vector machine (SVM), expectation-maximization (EM-SEMI), and class mass normalization (CMN) methods have been investigated on NASA data sets, which are CM1, KC1, KC2, and PC1. Experimental results showed that SVM and LDS algorithms outperform CMN and EM-SEMI algorithms. In addition, LDS algorithm performs much better than SVM when the data set is large. In this study, the LDS-based prediction approach is suggested for software defect prediction when there are limited fault data.

Keywords: Defect prediction, expectation-maximization, low-density separation, quality estimation, semi-supervised classification, support vector machines.

***Corresponding author: Cagatay Catal,** Department of Computer Engineering, Istanbul Kultur University, Istanbul 34156, Turkey, e-mail: c.catal@iku.edu.tr

1 Introduction

The activities of modern society highly depend on software-intensive systems, and therefore, the quality of software systems should be measured and improved. There are many definitions of software quality, but unfortunately, the general perception is that software quality cannot be measured or should not be measured if it works. However, this assumption is not true, and there are many software quality assessment models such as SQALE that investigate and evaluate several internal quality characteristics of software. Software quality can be defined differently based on different views of quality [19]:

- Transcendental view: quality can be described with abstract terms instead of measurable characteristics and users can recognize quality if it exists in a software product.
- Value-based view: quality is defined with respect to the value it provides and customers decide to pay for the software if the perceived value is desirable.
- User view: quality is the satisfaction level of the user in terms of his/her needs.
- Manufacturing view: process standards improve the product quality and these standards must be applied.
- Product view: This view of quality focuses on internal quality characteristics.

In this study, we use the product view definition of software quality, and internal quality characteristics are represented with metrics values. Objective assessment of software quality is performed based on measurement data collected during quality engineering processes. Quality assessment models are quantitative analytical models and more reliable compared with qualitative models based on personal judgment.

They can be classified into two categories: generalized and product-specific models [19]:

1. Generalized models: Project or product-specific data are not used in generalized models and industrial averages help to estimate the product quality roughly. They are grouped into three subcategories:
   - Overall models: a rough estimate of quality is provided.

   - Segmented models: different quality estimates are applied for different product segments.
   - Dynamic models: dynamic model graphs depict the quality trend over time for all products.
2. Product-specific models: Product-specific data are used in these models in contrast to the generalized models.
   - Semi-customized models: These models apply historical data from previous releases of the same product instead of a profile for all products.
   - Observation-based models: These models only use data from the current project, and quality is estimated based on observations from the current project.
   - Measurement-driven predictive models: These models assume that there is a predictive relation between software measurements and quality.

Software defect prediction models are in the measurement-driven predictive models group, and the objective of our research is to evaluate several semi-supervised classification methods for software defect prediction models when there are limited defect data. Defect prediction models use software metrics and previous defect data. In these models, metrics are regarded as independent variables and defect label is regarded as dependent variable. The aim is to predict the defect labels (defect-prone or non-defect-prone) of the modules for the next release of software. Machine-learning algorithms or statistical techniques are trained with previous software metrics and defect data. Then, these algorithms are used to predict the labels of new modules. The benefits of this quality assurance activity are impressive. Using such models, one can identify the refactoring candidate modules, improve the software testing process, select the best design from design alternatives with class level metrics, and reach a dependable software system.

Software defect prediction models have been studied since 1990s and up to now, and defect-prone modules can be identified prior to system tests using these models. According to recent studies, the probability of detection (PD) (71%) of defect prediction models may be higher than PD of software reviews (60%) if a robust model is built [15]. A panel at the IEEE Metrics 2002 conference did not support the claim [6] that inspections can find 95% of defects before testing, and this detection ratio was around 60% [15]. One member of the review team may inspect 8–20 lines of code per minute. This process is performed by all the members of a team consisting of 4–6 members [15]. Therefore, software defect prediction approaches are much more cost-effective in detecting software defects compared with software reviews. Readers who want to get more information about software defect prediction may read our literature review [2] and systematic review papers [4].

Most of the studies in literature assume that there is enough defect data to build the prediction models. However, there are cases when previous defect data are not available, such as when a company deals with a new project type or when defect labels in previous releases may have not been collected. In these cases, we need new models to predict software defects. This research problem is called software defect prediction of unlabeled program modules.

Supervised classification algorithms in machine learning can be used to build the prediction model with previous software metrics and previous defect labels. However, sometimes, we cannot have enough defect data to build accurate models. For example, some project partners may not collect defect data for some project components, or the execution cost of metrics collection tools on the whole system may be extremely expensive. In addition, during decentralized software development, some companies may not collect defect data for their components. In these situations, we need powerful classifiers that can build accurate classification models with very limited defect data or powerful semi-supervised classification algorithms that can benefit from unlabeled data together with labeled one. This research problem is called software defect prediction with limited defect data. This study is an attempt to identify a semi-supervised classification algorithm for limited defect data problem.

We applied low-density separation (LDS), support vector machine (SVM), expectation-maximization (EM-SEMI), and class mass normalization (CMN) methods to investigate their performance on NASA data sets, which are CM1, KC1, KC2, and PC1. Although previous research on semi-supervised defect prediction reported that EM-SEMI-based semi-supervised approach is very effective [16], we showed that LDS and SVM algorithms provide better performance than EM-based prediction approach. Seliya and Khoshgoftaar [16]

stated that their future work is to compare the EM-based approach with other semi-supervised learning approaches such as SVM or co-training. Therefore, the scope of our study is similar to their article's scope, but we extend their study with new algorithms and evaluate several methods to find the best method. The details of the algorithms used in this study are explained in Section 3. In addition, recent studies on the use of semi-supervised learning methods for defect prediction are shown in detail in Section 2.

This article is organized as follows: Section 2 presents the related work in the software defect prediction area. Section 3 explains the semi-supervised classification methods, whereas Section 4 shows the experimental results. Finally, Section 5 presents the conclusions and the future works.

2 Related Work

Seliya and Khoshgoftaar [16] applied EM algorithm for software defect prediction with limited defect data, and they reported that the EM-based approach provides remarkable results compared with classification algorithms. In the research project started in April 2008, we showed that naïve Bayes algorithm is the best choice to build a supervised defect prediction model for small data sets, and yet another two-stage idea (YATSI) algorithm may improve the performance of naïve Bayes for large data sets [3]. Researchers are very optimistic to apply unlabeled data together with labeled one when there is not enough labeled data for classification. However, we showed some evidence that this optimistic idea is not true for all the classifiers on all the data sets [3]. Naïve Bayes performance degraded when unlabeled data were used for YATSI algorithm on KC2, CM1, and JM1 data sets [3]. Recently, Li et al. [11] proposed two semi-supervised machine-learning methods (CoForest and ACoForest) for defect prediction. They showed that CoForest and ACoForest can achieve better performance than traditional machine-learning techniques such as naïve Bayes. Lu et al. [13] developed an iterative semi-supervised approach for software fault prediction and changed the size of labeled modules from 2% to 50%. They showed that this approach is better than the corresponding supervised learning approach with random forest as the base learner. This approach improves the performance only if the number of labeled modules exceeds 5%. Lu et al. [14] investigated a variant of self-training called fitting-the-confident-fits (FTcF) as a semi-supervised learning approach for fault prediction. In addition to this algorithm, they used a preprocessing step called multidimensional scaling (MDS). They showed that this model, based on FTcF and MDS, is better than best supervised learning algorithms such as random forest. Jiang et al. [7] proposed a semi-supervised learning approach called ROCUS for software defect prediction. They showed that this approach is better than a semi-supervised learning approach that ignores the class-imbalance nature of the task. In these articles, researchers did not investigate the methods we used in this article. The drawback of these approaches that use semi-supervised learning methods is that they need at least some labeled data points [10].

3 Methods

In this section, semi-supervised classification methods that have been investigated are explained.

3.1 Low-Density Separation

Chapelle and Zien [5] developed the LDS algorithm in 2005. They showed that LDS provides better performance than SVM, manifold, transductive support vector machine (TSVM), graph, and gradient TSVM. LDS first calculates the graph-based distances and then applies gradient TSVM algorithm. LDS algorithm combines two algorithms [5]: graph (training an SVM on a graph-distance-derived kernel) and $\nabla TSVM$ (training a TSVM by gradient descent).

SVM was first introduced in 1992 [1], and now it is widely used for machine-learning research. SVM is a supervised learning method that is based on the statistical learning theory and the Vapnik-Chervonenkis dimension. Although standard SVM learns to separate the hyperplane on labeled data points, TSVM learns to separate the hyperplane on both labeled and unlabeled data points. Although TSVM appears to be a very powerful semi-supervised classification method, its objective function is nonconvex, and therefore, it is difficult to minimize this function, and sometimes, bad results may results from optimization heuristics such as SVMlight [5]. Because LDS includes two algorithms that are based on SVM and TSVM, LDS can be thought of as an extended version of SVM algorithms for semi-supervised classification. We used the MATLAB (MathWorks, Natick, MA, USA) implementation of LDS algorithm, which can be accessed from www.kyb.tuebingen.mpg.de/bs/people/chapelle/lds. The motivation and theory behind this algorithm are discussed in a paper published by Chapelle and Zien [5]. They have shown that the combination of graph and $\nabla TSVM$ yields a superior semi-supervised classification algorithm.

3.2 Support Vector Machine

We used SVMlight [8] software for the experiments. SVMlight is an implementation of Vapnik's SVM [20] in C programming language for pattern recognition and regression. There are several optimization algorithms [9] in the SVMlight software.

3.3 Expectation-Maximization

EM, which is an iterative algorithm, is used for maximum likelihood estimation with incomplete data. EM replaces missing data with estimated values, estimates model parameters, and then re-estimates the missing data [17]. Therefore, it is an iterative approach. In this study, the class label is used as the missing value. The theory and foundations of EM are discussed by Little and Rubin [12]. EM algorithm has been evaluated for semi-supervised classification in literature [17], and we included it to compare the performance of LDS with the performance of EM. We have noted that LDS algorithm outperforms EM algorithm.

3.4 Class Mass Normalization

Zhu et al. [21] developed an approach that is based on a Gaussian random field model defined on a weighted graph. On the weighted graph, vertices are labeled and unlabeled data points. The weights are given in terms of a similarity function between instances. Labeled data points are shown as boundary points, and unlabeled points appear as interior points. The CMN method was adopted to adjust the class distributions to match the prior knowledge [21]. Zhu et al. [21] chose Gaussian fields over a continuous state space instead of random fields over the discrete label set. They showed that their approach was promising on several data sets, and they used only the mean of the Gaussian random field, which is characterized in terms of harmonic functions and spectral graph theory [21]. In this article, we used this approach, which uses the CMN method, for our benchmarking analysis.

4 Empirical Case Studies

This section presents the data sets and performance evaluation metrics that were used in this study and the results of the experimental studies.

4.1 System Description

NASA projects located in the PROMISE repository were used for our experiments. We used four public data sets, which are KC1, PC1, KC2, and CM1. The KC1 data set, which uses the C++ programming language, belongs to a storage management project for receiving/processing ground data. It has 2109 software modules, 15% of which are defective. PC1 belongs to a flight software project for an Earth-orbiting satellite. It has 1109 modules, 7% of which are defective, and uses the C implementation language. The KC2 data set belongs to a data-processing project, and it has 523 modules. The C++ language was used for the KC2 implementation, and 21% of its modules have defects. CM1 belongs to a NASA spacecraft instrument project and was developed in the C programming language. CM1 has 498 modules, of which 10% are defective.

4.2 Performance Evaluation Parameter

The performance of the semi-supervised classification algorithms was compared using the area under receiver operating characteristic (ROC) curve (AUC) parameter. ROC curves and AUC are widely used in machine learning for performance evaluation. ROC curve was first used in signal detection theory to evaluate how well a receiver distinguishes a signal from noise, and it is still used in medical diagnostic tests. The ROC curve plots probability of false alarm (PF) on the x-axis and the PD on the y-axis. A ROC curve is shown in Figure 1. ROC curves pass through points (0, 0) and (1, 1) by definition. Because the accuracy and precision parameters are deprecated for unbalanced data sets, we used AUC values for benchmarking.

Equations (1) and (2) were used to calculate PD and PF values, respectively. Each pair (PD, PF), represents the classification performance of the threshold value that produces this pair. The pair that produces the best classification performance is the best threshold value to use in practice [18].

$$\mathrm{PD}=\frac{TP}{TP+FN}, \tag{1}$$

$$\mathrm{PF}=\frac{FP}{FP+TN}. \tag{2}$$

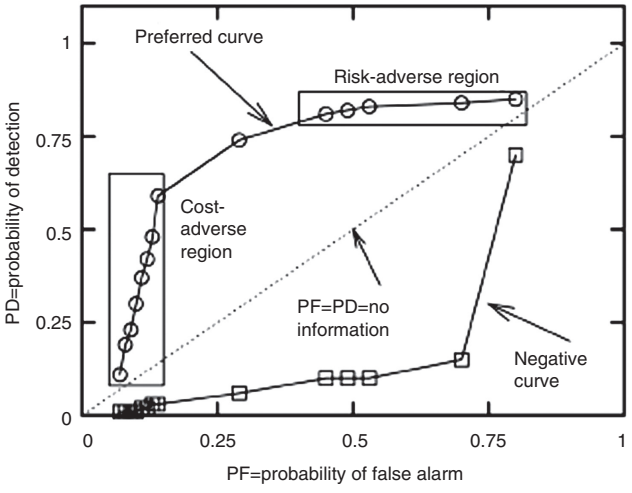

Figure 1. ROC Curve [15].

4.3 Results

We simulated small labeled-large unlabeled data problem with 5%, 10%, and 20% rates and evaluated the performance of each semi-supervised classification algorithm under these circumstances. This means that we trained on only 5%, 10%, and 20% of the data set and tested on the rest for LDS, SVM, CMN, and EM-SEMI algorithms.

Tables 1–4 show the performance results of each algorithm on different data sets with respect to these rates, and we compared classifiers using AUC values. According to experimental results in Tables 1 and 2, the best classifier for CM1 and KC2 is SVM because SVM has the highest AUC values for these data sets. According to experimental results in Tables 3 and 4, the best classifier for KC1 and PC1 is LDS because LDS has the highest AUC values for these data sets.

For all the data sets, SVM and LDS, which are based on SVM, provide better performance than EM-SEMI algorithm. When we looked at the size of data sets, we have noted that the size of data set is large when LDS provides better performance than SVM. Based on these experimental results, we can express that applying LDS algorithm is an ideal solution for large data sets when there are not enough labeled data.

Table 1. Performance Results on CM1.

Algorithm	5%	10%	20%
LDS	0.66	0.70	0.73
SVM	0.76	0.76	0.73
CMN	0.65	0.67	0.74
EM-SEMI	0.47	0.47	0.53

Table 2. Performance Results on KC2.

Algorithm	5%	10%	20%
LDS	0.80	0.81	0.82
SVM	0.84	0.85	0.85
CMN	0.69	0.69	0.69
EM-SEMI	0.47	0.56	0.55

Table 3. Performance Results on KC1.

Algorithm	5%	10%	20%
LDS	0.77	0.78	0.79
SVM	0.76	0.75	0.70
CMN	0.73	0.76	0.76
EM-SEMI	0.49	0.48	0.50

Table 4. Performance Results on PC1.

Algorithm	5%	10%	20%
LDS	0.73	0.75	0.75
SVM	0.70	0.73	0.70
CMN	0.63	0.61	0.80
EM-SEMI	0.60	0.58	0.55

We compared the results of this study with the performance results we achieved in our previous research [3], and we showed that SVM or LDS provides much better performance than the algorithms we used in our previous research [3]. This research also indicates that although EM algorithm has been suggested for semi-supervised defect prediction in literature [17], it is very hard to get acceptable results with this algorithm.

All the classifiers reached their highest AUC values for the KC2 data set. This empirical observation may indicate that the KC2 data set has very few noisy modules or does not have any noisy module. This observation has been observed in our previous research as well [3]. Although CM1 and KC2 data sets have roughly the same number of modules, the classifiers' performances are higher for KC2 than CM1. Furthermore, lines of code for KC2 are approximately two times longer than CM1 even though they have roughly the same number of modules. According to these observations, we can conclude that SVM performance variations for KC2 and CM1 data sets may be due to the different usage of lines of code per method.

5 Conclusions and Future Works

Software defect prediction using only few defect data is a challenging research area, where only very few studies were made. The previous works were based on EM-based approaches, and in this study, we showed that the LDS-based defect prediction approach works much better than EM-based defect prediction [3]. The main contribution of our article is two-fold. First, this study compares several semi-supervised algorithms for software defect prediction with limited data and suggests a classifier for further research. Second, we empirically show that LDS is usually the best semi-supervised classification algorithm for large data sets when there are limited defect data. In addition, SVM algorithm worked well on smaller data sets. However, the difference between the performance of SVM and LDS for small data sets is small, and quality managers may directly use LDS algorithm for semi-supervised defect prediction without considering the size of the data set. Because LDS provided high performance for our experiments, there is a potential research opportunity to investigate the LDS for much better models for semi-supervised software fault prediction. We did not encounter any article in literature that used LDS for this problem. In the future, we are planning to compare the performance of LDS with the recent approaches explained in Section 2 and correspond with their respective authors. We are also planning to use genetic algorithms to optimize the parameters of LDS algorithm and propose a new algorithm that can achieve higher performance. In addition, case studies of other systems such as open-source projects, e.g., eclipse, may provide further validation of our study.

Received May 8, 2013; previously published online August 31, 2013.

Bibliography

[1] B. E. Boser, I. Guyon and V. Vapnik, A training algorithm for optimal margin classifiers, *COLT* **1992** (1992), 144–152.
[2] C. Catal, Software fault prediction: a literature review and current trends, *Expert Syst. Appl.* **38** (2011), 4626–4636.
[3] C. Catal and B. Diri, Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction, *Expert Syst. J. Knowledge Eng.* **26** (2009), 458–471.
[4] C. Catal and B. Diri, A systematic review of software fault prediction studies, *Expert Syst. Appl.* **36** (2009), 7346–7354.
[5] O. Chapelle and A. Zien, Semi-supervised classification by low density separation, in: *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pp. 57–64, Society for Artificial Intelligence and Statistics, 2005.
[6] M. Fagan, Advances in software inspections, *IEEE Trans. Software Eng.* **12** (1986), 744–751.
[7] Y. Jiang, M. Li and Z. Zhou, Software defect detection with ROCUS, *J. Comput. Sci. Technol.* **26** (2011), 328–342.
[8] T. Joachims, *Making large-scale SVM learning practical, advances in kernel methods – support vector learning*, MIT Press, Cambridge, MA, 1999.
[9] T. Joachims, *Learning to classify text using support vector machines*, dissertation, 2002.
[10] E. Kocaguneli, B. Cukic and H. Lu, Predicting more from less: synergies of learning, in: *2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2013)*, pp. 42–48, West Virgina University, San Francisco, CA, 2013.

[11] M. Li, H. Zhang, R. Wu and Z. Zhou, Sample-based software defect prediction with active and semi-supervised learning, *Automated Software Eng*. **19** (2012), 201–230.
[12] R. J. A. Little and D. B. Rubin, *Statistical analysis with missing data*, Wiley, New York, 1987.
[13] H. Lu, B. Cukic and M. Culp, An iterative semi-supervised approach to software fault prediction, in: *Proceedings of the 7th International Conference on Predictive Models in Software Engineering (PROMISE '11)*, 10 pp., ACM, New York, NY, Article 15, 2011.
[14] H. Lu, B. Cukic and M. Culp, Software defect prediction using semi-supervised learning with dimension reduction, in: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, pp. 314–317, ACM, New York, NY, 2012.
[15] T. Menzies, J. Greenwald and A. Frank, Data mining static code attributes to learn defect predictors, *IEEE Trans. Software Eng.* **33** (2007), 2–13.
[16] N. Seliya and T. M. Khoshgoftaar, Software quality estimation with limited fault data: a semi-supervised learning perspective, *Software Qual. Contr.* **15** (2007), 327–344.
[17] N. Seliya, T. M. Khoshgoftaar and S. Zhong, Semi-supervised learning for software quality estimation, in: *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '04)*, pp. 183–190, IEEE Computer Society, Washington, DC, 2004.
[18] R. Shatnawi, W. Li, J. Swain and T. Newman, Finding software metrics threshold values using ROC curves, *J. Software Maintenance Evol. Res. Pract.* **22** (2010), 1–16.
[19] J. Tian, *Software quality engineering: testing, quality assurance, and quantifiable improvement*, John Wiley & Sons, Hoboken, New Jersey, USA, 2005.
[20] V. Vapnik, *The nature of statistical learning theory*, Springer, New York, NY, USA, 1995.
[21] X. Zhu, Z. Ghahramani and J. D. Lafferty, Semi-supervised learning using Gaussian fields and harmonic functions, *ICML* **2003** (2003), 912–919.