Journal of Zhejiang University-SCIENCE C: Computers & Electronics
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Multi-instance learning for software quality estimation in object-oriented systems: a case study

Peng HUANG†, Jie ZHU

(*Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*)

†E-mail: superhp@sjtu.edu.cn

Received Feb. 11, 2009; Revision accepted June 18, 2009; Crosschecked Sept. 27, 2009

Abstract: We investigate a problem of object-oriented (OO) software quality estimation from a multi-instance (MI) perspective. In detail, each set of classes that have an inheritance relation, named 'class hierarchy', is regarded as a bag, while each class in the set is regarded as an instance. The learning task in this study is to estimate the label of unseen bags, i.e., the fault-proneness of untested class hierarchies. A fault-prone class hierarchy contains at least one fault-prone (negative) class, while a non-fault-prone (positive) one has no negative class. Based on the modification records (MRs) of the previous project releases and OO software metrics, the fault-proneness of an untested class hierarchy can be predicted. Several selected MI learning algorithms were evaluated on five datasets collected from an industrial software project. Among the MI learning algorithms investigated in the experiments, the kernel method using a dedicated MI-kernel was better than the others in accurately and correctly predicting the fault-proneness of the class hierarchies. In addition, when compared to a supervised support vector machine (SVM) algorithm, the MI-kernel method still had a competitive performance with much less cost.

Key words: Object-oriented (OO) software, Multi-instance (MI) learning, Software quality estimation, Kernel methods
doi:10.1631/jzus.C0910084 **Document code:** A **CLC number:** TN914; TN915; TP311

1 Introduction

In the last decade, the object-oriented (OO) method has become one of the most common paradigms in software design and development. Thus, the problem of OO software quality estimation becomes a focus in OO software engineering. Correctly predicting and detecting the fault-prone module before the software releases are delivered can reduce both manual testing and maintenance cost.

In general, a software quality estimation model is constructed with the following three steps. First, software modules collected from similar or previous releases are labeled as fault-prone (FP) or non-fault-prone (NFP). Then each module is transformed into a corresponding attribute-value vector by a suite of software metrics (Chidamber and Kemerer, 1994). For OO-style software, the specifically designed OO software metrics (Basili *et al*., 1996; Briand *et al*., 2000; Cartwright and Shepperd, 2000) are usually chosen for quality model construction. Finally, these labels and vectors, which store the information about the software, are used together to establish a model.

Software quality estimation models have proven to be effective in many real-world software projects (Khoshgoftaar *et al*., 1997; Cartwright and Shepperd, 2000; Kanmani *et al*., 2007). Traditional OO quality estimation models are mostly built using the supervised learning schema, which views each class in OO software as a basic module, and demands the labels of every module in the training. Reliable labels of the software modules can be obtained only by thorough testing and careful locating, which are time-consuming and costly. Moreover, many practical factors such as multi-site development and project transfer may lessen the credibility of some available labels. Therefore, a semi-supervised learning schema such as multi-instance (MI) learning can greatly

reduce the number of necessary labels, which contributes towards both effort reduction and reliability enhancement. Other important reasons for introducing MI learning to the OO software quality estimation problem include:

1. In practice, fault prediction tools can be used only as an early auxiliary component for large-scale software projects. Manual and automatic testing is still unavoidable to assure the software quality. Thus, a fault prediction model does not need to localize the fault in a very small area.

2. Limited fault data always reduce the size of available training data due to many practical software engineering problems (Seliya and Khoshgoftaar, 2007), while MI learning can utilize plentiful attribute-value vectors and relatively few labels.

3. In OO systems, the complete definition of an inherited method is allowed to be absent in the class that inherits it. A fault or modification in a class may not be separated from other classes in the same class hierarchy, meaning that these classes should be learned as a whole.

4. Even when some inheritance-related software metrics are used, treating class as a basic module (supervised learning schema) can still partly ignore the relational information hidden in the inheritance-related classes as reported in previous research (Huang and Zhu, 2008).

In MI learning, the elemental learning units are labeled bags which are composed of unlabeled instances. Under the given precondition that there is at least one positive instance in a positive bag while no positive instance in the negative bags, the task of MI learning is to predict the label of unseen bags. Dietterich *et al*. (1997) demonstrated that supervised learning algorithms such as the popular decision tree and neural networks do not work well in this scenario compared with their proposed MI learning framework.

The main contribution of this study is to investigate the effect of MI learning in estimating OO software quality. Several popular MI learning algorithms were selected and experimentally evaluated on five datasets collected from an industrial optical communication software project. A supervised support vector machine (SVM) algorithm was also studied for comparison. The experimental results revealed that the kernel method combining a special MI kernel and SVM has a superior performance in terms of prediction accuracy and Type I and Type II error ratios among the studied MI learning methods. Although it was slightly inferior to the two-step supervised support vector classifier (SVC), the MI-kernel method still had a competitive outcome with much less effort in obtaining labels.

2 Related works

2.1 Supervised learning method

Most attention of related works is given to supervised learning methods for software quality estimation. Briand *et al*. (2000) took logistic regression to explore the relationships between four groups of OO metrics and the fault-detected system classes during testing. Khoshgoftaar *et al*. (1997) applied back propagation neural networks to predict the fault-prone software modules in a large telecommunication project. Kanmani *et al*. (2007) investigated two fault prediction models based on OO metric and neural networks using a dataset of software modules designed by students. Among the two neural networks, the probabilistic neural network outperformed the back propagation one in accurately predicting the fault-proneness of the OO modules. Reformat *et al*. (2003) exploited several techniques of computational intelligence to support the quality assessment of individual software objects. These techniques covered granular computing, neural networks, self-organizing maps and evolutionary-based developed decision. Recently, Elish and Elish (2008) evaluated the capability of SVMs in predicting defect-prone software modules and compared their prediction performance to eight statistical and machine learning models in the context of four National Aeronautics and Space Administration (NASA) datasets.

Several other supervised learning approaches including regression trees (Khoshgoftaar *et al*., 2002), Bayesian belief networks (Fenton *et al*., 2002), decision trees (Huang *et al*., 2006), genetic programming (Evett *et al*., 1998), and random forests (Guo *et al*., 2004) have also been reported for software quality estimation.

2.2 Semi-supervised learning method

Semi-supervised learning for software quality estimation was first introduced by Seliya and

Khoshgoftaar (2007). They viewed the class labels as missing values and used an expectation-maximization (EM) algorithm to estimate the labels, which represented the class fault-proneness. According to their study, the semi-supervised learning using an EM algorithm provided comparable results to the compared supervised classifiers. Catal *et al*. (2008) studied a software quality prediction model using a simple semi-supervised learning algorithm called YATSI (Driessens *et al*., 2006). This semi-supervised approach had a similar performance to the leading supervised classifier using random forest (RF) on some public software datasets.

In our prior work (Huang and Zhu, 2008), a kernel method with a layered kernel has been applied to quality prediction of the class hierarchy in OO systems. Compared to the statistical kernel, the layered kernel showed a superior performance for both the artificial and real-life datasets.

2.3 MI learning applications

MI learning was first introduced for drug activity prediction (Dietterich *et al*., 1997). As a new learning framework, MI learning has been given much attention and applied to many machine learning problems, such as scene classification (Zhou and Zhang, 2006), web mining (Zhou *et al*., 2005), data mining applications (Ruffo, 2000), text categorization (Andrews *et al*., 2003), image categorization (Tang *et al*., 1999), and object class reorganization (Chen *et al*., 2006). To our knowledge, MI learning for OO software quality estimation has not been reported.

3 Multi-instance learning for object-oriented software quality estimation

The MI learning problem in this paper is limited to the definition below, which is consistent with its standard form (Dietterich *et al*., 1997):

Definition (MI problem) An MI concept is a function c_{MI} on $2^{\chi}\to\Omega$. Here $\Omega=\{+1, -1\}$ and χ denotes the instance space. It is defined as

$$c_{\mathrm{MI}}(X)=-1 \text{ iff } \exists x\in X: c_1(x)=-1, \qquad (1)$$

where c_{MI} is a specific concept from a concept space, and $X\subset\chi$ is a set of instances. Note that compared to the usual formulation of the MI setting, positive and negative labels have changed their roles in Eq. (1).

As shown in Fig. 1, the basic learning unit in our study is not a single class, but a bag of classes. The MI bagging schema in this study is clustering by inheritance relation. Preliminary experiments (Huang and Zhu, 2008) have showed that knowledge concerning the inheritance relationship can be better reserved in the class hierarchies if they are trained as a whole rather than as a group of separate instances. In our MI learning model for OO software, each set of classes that have inheritance relation, named 'class hierarchy' (Fig. 2), is regarded as a bag, while each class is regarded as an instance. Each class instance is represented by a vector $\boldsymbol{C}=[c_1, c_2, \ldots, c_n]$, where c_i (i=1, 2, …, n) denotes the measurement on this class using the ith member in the suite of software metrics for this project. The suite of OO software metrics selected in our experiments is described in Section 5.2. Now the MI prediction task of our software quality estimation is to label the untested class hierarchy as FP (negative) or NFP (positive). A positive bag means that no fault-prone class exists in this class hierarchy, while a negative bag is a class hierarchy containing at least one fault-prone class.

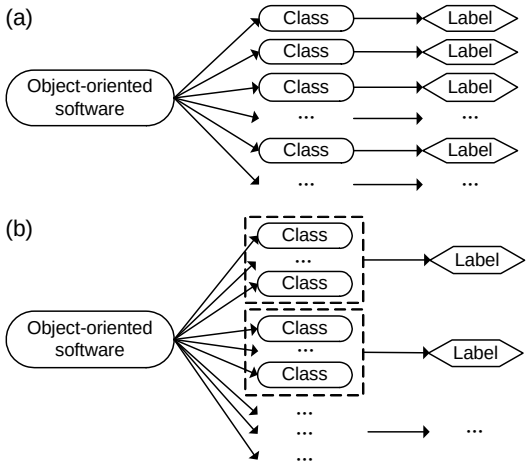

Fig. 1 The difference between (a) supervised learning and (b) MI learning in software quality estimation

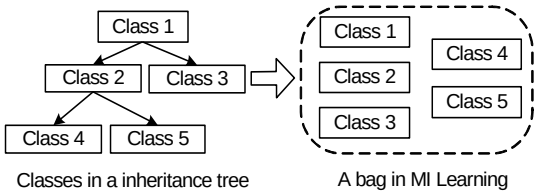

Fig. 2 An example of an MI bag for object-oriented software

It is worth mentioning that encapsulating each class hierarchy into a bag is an effective but not the only available schema for the MI learning in OO software quality modeling. Other bagging rules such as clustering by software function or by designer can also be considered.

4 Selected MI learning algorithms for evaluation

Many learning algorithms have been developed for MI problems. All these algorithms can be divided into two major categories: specifically designed approaches and approaches adapted from existing supervised learning methods. The axis-parallel rectangles (APR) algorithms and its variants (Dietterich *et al*., 1997), the diverse density (DD) approach (Maron and Lozano-Pérez, 1998) and its upgraded version EM-DD (Zhang and Goldman, 2001), and a boosting method devised to find the optimal ball in MI space (Auer and Ortner, 2004) belong to the first category. The lazy learning algorithms Bayesian-kNN and Citation-kNN (Wang and Zucker, 2000), the decision tree algorithm MI-ID3 (Chevaleyre and Zucker, 2001), the back propagation (BP) neural network MI-BPN (Zhang and Zhou, 2004), and the MI versions of SVM (Andrews *et al*., 2003) are in the second category. Recently, some MI-kernels were also introduced (Gartner *et al*., 2002) to solve the MI problems.

Several representative algorithms (Table 1) were selected to investigate the MI learning in OO software quality estimation. The MI-kernel method was implemented using the SimpleSVM (SSVM) toolkit (Vishwanathan *et al*., 2003), and other algorithms were based on the open source WEKA (Waikato Environment for Knowledge Analysis, http://www.cs.waikato.ac.nz/~ml/weka/) learning toolkit. The experimental results are detailed in Section 6.

Table 1 Selected MI learning methods for comparison

Symbol	Method description
MIOptimal-Ball	Optimal ball algorithm for MI (Auer and Ortner, 2004)
Citation-kNN	Citation *k*-nearest neighbors algorithm (Wang and Zucker, 2000)
MI-SVM	Support vector machine for MI (Andrews *et al*., 2003)
MI-DD	Diverse density approach for MI (Maron and Lozano-Pérez, 1998)
MI-EMDD	Diverse density approach with expectation maximization for MI (Zhang and Goldman, 2001)
MI-ID3	MI version of the decision tree algorithms ID3 (Chevaleyre and Zucker, 2001)
MI-BPN	Back propagation neural networks for MI (Zhang and Zhou, 2004)
MI-kernel	An MI kernel method

5 An empirical study

5.1 Description of the software system

The software project used to investigate the MI learning based OO software quality estimation was taken from the optical communication domain. Written in C++, this software project implements mainly commuting and transporting data packages varying from 2 kb to 10 Gb in the synchronous digital hierarchy (SDH) networks. It was developed by the software development team in the optical networks department of Alcatel-Lucent, Shanghai. Another independent system verification and validation team (SVT) in the same department undertook the testing. After the codes passed the functional test performed by software team (SW) itself, they were verified and validated by the SVT in the practical communication environment. Hundreds of test-cases, including both manual and automatic tests, were executed to ensure product quality. If the test results did not meet requirements, corresponding modification reports (MRs) were proposed. Each MR was assigned a severity level from 1 to 5 with increasing severity.

The selected software release comprised about 170 000 lines of code and a total of 425 classes. After clustering by inheritance relation, the whole dataset had 175 bags including 33 multi-class bags. The biggest bag contained 32 instances. The mean inheritance height of all class hierarchies (bags) and multi- class class hierarchies (bags) was 1.57 and 2.33, respectively. Because multiple inheritance mechanism tends to trigger code collision and misunderstanding, it was seldom used in this release.

5.2 Dataset construction

The whole dataset was constructed with the labels of the class hierarchies and a suite of appropriate software metrics. In our work, the fault-proneness label of the class hierarchy (bag) was determined by

the number of MRs and their severity. In general, if a bag's weighted sum of the related MRs is more than 2.5, it is viewed as fault-prone (negative).

Software metrics are usually used to capture the characters and the quality trends of software modules. For OO software projects, many specifically designed OO metrics have been proposed and evaluated. Briand *et al*. (2000) investigated 64 existing software metrics in OO systems. All these metrics can be divided into four groups, i.e., size, coupling, cohesion and inheritance measure. In this study, 21 commonly used software metrics (Table 2) in each of the four groups were taken to measure each class, which cover the core sets in Kanmani *et al*. (2007) and Chidamber and Kemerer (1994). In general, fewer than 10 well-selected metrics are quite powerful in application (Berard, 1998). According to Briand *et al*. (2000) and Kanmani *et al*. (2007), many existing OO metrics are based on similar ideas and hypotheses, and therefore they are almost always interrelated and somewhat redundant. Thus, a large number (e.g., 20 to 30, or more) of different metrics can store sufficient information for the vectors for classes.

Table 2 Selected OO software metrics

Metric	Description
CBO	Coupling between object classes
CSAO	Class size (attributes & operations)
CSA	Class size (attributes)
CSI	Class specialization index
CSO	Class size (operations)
DIT	Depth of inheritance tree
LOC	Lines of code
LOCM	Lack of cohesion methods
NAAC	Number of attributes added
NAIC	Number of attributes inherited
NAOC	Number of operations added
NOIC	Number of operations inherited
NPavgC	Average number of method parameters
NSUB	Number of sub classes
OSavg	Average operation size
PA	Private attribute usage
PPPC	Percentage public/protected members
RFC	Response for class
SLOC	Source lines of code
TLOC	Total lines of code
WMC	Weighted methods in class

Since the software release delivered to the system verification and validation team has already been debugged, the distribution of positive and negative instances in the whole dataset tends to be unbalanced. According to our labeling schema, only 27 instances were labeled as faulty (negative) compared with 148 labeled as non-faulty (positive). To investigate the effect of unbalanced data, five datasets were taken from the software project. D_1 is the whole dataset, D_2 (100 bags in total) and D_3 (50 bags in total) contain all 33 multi-class instances. D_4 (100 bags in total) and D_5 (50 bags in total) contain all 27 negative instances. The remaining instances in D_2–D_5 were randomly selected. The details of the datasets are tabulated in Table 3.

Table 3 Experimental datasets

Dataset	Number of datasets			
	Positive	Negative	Single-class	Multi-class
D_1	148	27	142	33
D_2	82	18	67	33
D_3	36	14	17	33
D_4	73	27	70	30
D_5	23	27	22	28

5.3 Parameters for investigated algorithms

The parameters for each of the selected MI learning algorithms are listed below. If a parameter varied in a group of values in the experiment, only the best result was recorded.

1. Citation-kNN: numbers of citation (C_n) and reference (R) both varied from 1 to 10.

2. MI-SVM: the regularization parameter (C) was set at 1; the kernel function was RBF (radial basis function), and its bandwidth (γ) was set at 0.5.

3. MI-BPN: a three-layered multi-layer perceptron (MLP) was used as the network architecture. The number of hidden nodes is set to 4, the learning rate is set to 0.3, the momentum is set to 0.2, and the training time is set to 500 epochs.

4. MI-DD, MI-EMDD, MI-ID3 and MIOptimalBall: default setting of WEKA.

5. MI-kernel: the investigated kernel function is in the form of Eq. (2), and the parameter p varied from 1 to 10.

$$K_{\mathrm{MI}}(X,Y)=\sum_{x\in X,y\in Y}k_{\mathrm{I}}^{p}(x,y). \tag{2}$$

Here $k_{\mathrm{I}}(x, y)$ is a kernel on instance space χ, and $X, Y\subseteq\chi$.

5.4 Estimation performance measures

To evaluate the performance of each prediction model, the prediction accuracy was measured. Moreover, Type I and Type II error ratios were also measured because of the different positive and negative ratios of the bags in the experimental datasets. The definitions of the three measures are given in Eqs. (3)–(5):

$$\text{Accuracy} = (P_T + N_T) / (P + N), \quad (3)$$

$$\text{Type I error} = P_F / P, \quad (4)$$

$$\text{Type II error} = N_F / N. \quad (5)$$

Here P and N denote the total number of positive and negative bags respectively: $P=P_T+P_F$, $N=N_T+N_F$. They are derived from a confusion matrix as shown in Table 4.

Table 4 A confusion matrix

	Matrix element	
	Non-fault-prone (positive)**	Fault-prone (negative)**
Non-fault-prone (positive)*	True positive, P_T	False positive, P_F
Fault-prone (negative)*	False negative, N_F	True negative, N_T

* Actual; ** Predicted

6 Results and analysis

6.1 Comparison among MI methods

On each dataset, 5-fold cross validation (Kohavi, 1995) was executed to evaluate the performance of all the selected MI learning algorithms above mentioned. Fig. 3 shows the prediction accuracy of all the MI learning methods on the datasets D_1–D_5.

The experiments indicated that the MI-kernel method always performed best in the prediction accuracy on all the five datasets (Fig. 3). Although compared to other MI learning algorithms, the MI-kernel method had a slightly better performance on D_2 and D_5, it significantly gave a best overall prediction (average 92.03% accuracy) on D_1–D_5 (Table 5). Moreover, the MI-kernel method also showed a competitive performance in terms of Type I (lowest) and Type II (second lowest) error ratios on D_1–D_5.

A close inspection of the detailed experiment results in Table 6 indicates that as the proportion of multi-class bags increased on D_1–D_3, the MI-kernel's performance improved and attained the best result (94.00% accuracy, with 5.56% Type I and 7.14% Type II errors) on D_3 while other MI methods showed no obvious improvement or even degraded.

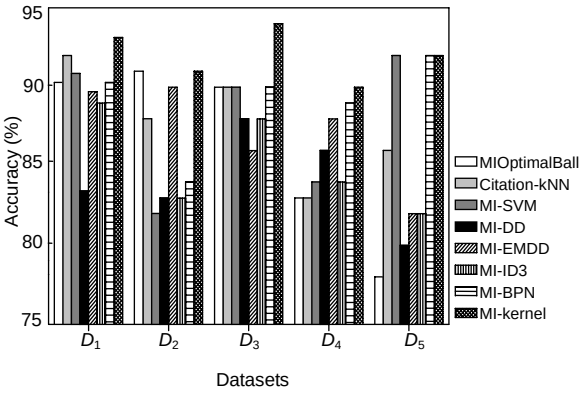

Fig. 3 Comparison of the prediction accuracy on D_1–D_5 (MI methods)

Table 5 Average accuracy and Type I and Type II error ratios over all datasets (MI methods)

Method	Accuracy (%)	Type I error (%)	Type II error (%)
MIOptimalBall	86.46	17.11	8.15
Citation-kNN	87.80	11.25	20.95
MI-SVM	87.77	11.49	21.06
MI-DD	84.09	11.54	36.19
MI-EMDD	87.14	14.49	13.97
MI-ID3	85.34	11.14	31.75
MI-BPN	89.06	10.81	16.24
MI-kernel	92.03	6.98	11.43

As the ratios of the negative and positive bags in D_1, D_4, and D_5 tended to be balanced, some MI methods (MIOptimalBall, Citation-kNN, MI-EMDD, MI-ID3) had a decrease in prediction accuracy while the MI-kernel had stable discrimination accuracy. At the same time, the Type I and Type II error ratios of the MI-kernel were still at a relatively low level.

In general, the specific MI-kernel method has a solid performance concerning accuracy and Type I and Type II error ratios over all the datasets. This means that the MI-kernel method was effective and robust for the datasets with unbalanced distributions of bag labels or bag size in this study. Thus, this characteristic makes the MI-kernel method a potentially suitable approach in predicting the quality of large-scale and highly complex OO software projects.

Table 6 Experimental results of different MI methods over all datasets

Method	Accuracy (%)				
	D_1	D_2	D_3	D_4	D_5
MIOptimalBall	90.29	91.00	90.00	83.00	78.00
Citation-kNN	92.00	88.00	90.00	83.00	86.00
MI-SVM	90.86	82.00	90.00	84.00	92.00
MI-DD	83.43	83.00	88.00	86.00	80.00
MI-EMDD	89.71	90.00	86.00	88.00	82.00
MI-ID3	89.71	83.00	88.00	84.00	82.00
MI-BPN	90.29	84.00	90.00	89.00	92.00
MI-kernel	93.14	91.00	94.00	90.00	92.00
Method	Type I error (%)				
	D_1	D_2	D_3	D_4	D_5
MIOptimalBall	9.46	10.98	13.89	16.44	34.78
Citation-kNN	5.41	10.98	5.56	8.22	26.09
MI-SVM	9.46	8.54	11.11	10.96	17.39
MI-DD	8.11	8.54	11.11	8.22	21.74
MI-EMDD	8.78	9.76	13.89	9.59	30.74
MI-ID3	4.73	8.54	11.11	9.59	21.74
MI-BPN	6.76	12.20	11.11	10.96	13.04
MI-kernel	5.41	9.76	5.56	5.48	8.70
Method	Type II error (%)				
	D_1	D_2	D_3	D_4	D_5
MIOptimalBall	11.11	0.00	0.00	18.52	11.11
Citation-kNN	22.22	16.67	21.43	40.74	3.70
MI-SVM	7.41	61.11	7.14	29.63	0.00
MI-DD	62.96	55.56	14.29	29.63	18.52
MI-EMDD	18.52	11.11	14.29	18.52	7.41
MI-ID3	40.74	55.56	14.29	33.33	14.81
MI-BPN	25.93	33.33	7.14	11.11	3.70
MI-kernel	14.81	5.56	7.14	22.22	7.41

6.2 Comparison with a supervised SVM classifier

A supervised SVM learning algorithm with an RBF kernel was also evaluated on D_1–D_5 for comparison. Since supervised learning algorithms cannot be applied directly to MI problems, different training and testing strategies were used. In the training phase, the supervised SVM was trained using labeled instances in the training bags. In the testing phase, the SVM classifier was utilized to predict the labels of every instance in the testing bags. Then the bag-labels were calculated using the predicted instance-labels according to Eq. (1). The parameters for this supervised SVM were the same as those for MI-SVM and all instance-labels were also given.

Table 7 reveals that the MI-kernel method was a little less effective in the prediction accuracy than the supervised SVM. However, this supervised SVM requires the labels of every instance in the training set and indirectly predicts the bag-labels. Considering the cost of obtaining the labels, the MI learning algorithms such as the MI-kernel method are at least competitive if no better choices.

Table 7 Comparison of experimental results of the MI-kernel method and a supervised SVM

Dataset	Method	Accuracy (%)	Type I error (%)	Type II error (%)
D_1	MI-kernel	93.14	5.41	14.81
	SVM (supervised)	95.43	2.03	18.52
D_2	MI-kernel	91.00	9.76	5.56
	SVM (supervised)	93.00	3.66	22.22
D_3	MI-kernel	94.00	5.56	7.14
	SVM (supervised)	94.00	0.00	21.43
D_4	MI-kernel	90.00	5.48	22.22
	SVM (supervised)	91.00	6.85	14.81
D_5	MI-kernel	92.00	8.70	7.41
	SVM (supervised)	92.00	4.35	11.11

7 Conclusion and future work

This paper proposed a novel MI learning approach for software quality estimation in OO systems. With an MI perspective, this approach attempted to apply an MI-kernel method to the fault-proneness prediction of OO software modules. In detail, the OO software quality estimation was considered as an MI problem by regarding each ‘class hierarchy’, which is a set of inheritance-related classes, as a bag and each class as an instance. The MI learning schema is suitable for non-thoroughly tested or in-progress OO software projects for which it is hard and unnecessary to ascertain the fault-proneness of every relatively small module.

Several representative MI learning algorithms were evaluated on five datasets from an optical communication software project. The experimental results showed that the MI-kernel method performs better when compared to other MI learning approaches in terms of overall predication accuracy and type I and type II error ratios, and was consistently efficient for the datasets with unbalanced size or label distribution. When compared to a supervised SVM learning algorithm in the same experiments, the MI-kernel method still had close and encouraging results with far fewer labels.

In the presented work, the OO software quality estimation was limited to a standard MI form, which is relatively naive. That is, the non-fault-proneness class hierarchy must have no fault-prone class instance. In fact, this decision criterion may vary for different software releases; e.g., newly started projects could have a higher threshold than the maintained release. Future work may extend the standard MI model to a more generalized one (Weidmann *et al*., 2003). Moreover, other kernels for multi-instance or structured data may also be studied for the MI based quality estimation in OO software systems.

References

Andrews, S., Tsochantaridis, I., Hofmann, T., 2003. Support Vector Machines for Multiple-Instance Learning. Proc. 15th Advances in Neural Information Processing Systems, p.561-568.

Auer, P., Ortner, R., 2004. A Boosting Approach to Multiple Instance Learning. Proc. 15th European Conf. on Machine Learning, p.63-74. [doi:10.1007/b100702]

Basili, V., Briand, L., Melo, W., 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Software Eng.*, **22**(10):751-761. [doi:10.1109/32.544352]

Berard, E.V., 1998. Metrics for Object-Oriented Software Engineering. Available at http://www.ipipan.gda.pl/~marek/objects/TOA/moose.html [Accessed on Dec. 10, 2009].

Briand, L., Wust, J., Daly, J., Victor, P.D., 2000. Exploring the relationships between design measures and software quality in object-oriented systems. *J. Syst. Software*, **51**(3):245-273. [doi:10.1016/S0164-1212(99)00102-8]

Cartwright, M., Shepperd, M., 2000. An empirical investigation of an object-oriented software system. *IEEE Trans. Software Eng.*, **26**(8):786-796. [doi:10.1109/32.879814]

Catal, C., Diri, B., 2008. A fault prediction model with limited fault data to improve test process. *LNCS*, **5089**:244-257. [doi:10.1007/978-3-540-69566-0_21]

Chen, Y., Bi, J., Wang, J.Z., 2006. MILES: multiple-instance learning via embedded instance selection. *IEEE Trans. Pattern Anal. Mach. Intell.*, **28**(12):1931-1947. [doi:10.1109/TPAMI.2006.248]

Chevaleyre, Y., Zucker, J.D., 2001. Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. *LNCS*, **2056**:204-214. [doi:10.1007/3-540-45153-6]

Chidamber, S., Kemerer, C., 1994. A metrics suite for object-oriented design. *IEEE Trans. Software Eng.*, **20**(6):476-493. [doi:10.1109/32.295895]

Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T., 1997. Solving the multiple instance problem with axis-parallel rectangles. *Artif. Intell.*, **89**(1-2):31-71. [doi:10.1016/S0004-3702(96)00034-3]

Driessens, K., Reutemann, P., Pfahringer, B., Leschi, C., 2006. Using weighted nearest neighbor to benefit from unlabeled data. *LNCS*, **3918**:60-69. [doi:10.1007/11731139]

Elish, K.O., Elish, M.O., 2008. Predicting defect-prone software modules using support vector machines. *J. Syst. Software*, **81**(5):649-660. [doi:10.1016/j.jss.2007.07.040]

Evett, M., Khoshgoftar, T., Chien, P.D., Allen, E., 1998. GP-Based Software Quality Prediction. Proc. 3rd Annual Genetic Programming Conf., p.60-65.

Fenton, N., Krause, P., Neil, M., 2002. Software measurement: uncertainty and causal modeling. *Software*, **19**(4):116-122. [doi:10.1109/MS.2002.1020298]

Gartner, T., Flach, P.A., Kowalczyk, A., Smola, A.J., 2002. Multi-Instance Kernels. Proc. 19th Int. Conf. on Machine Learning, p.179-186.

Guo, L., Ma, Y., Cukic, B., Singh, H., 2004. Robust Prediction of Fault-Proneness by Random Forests. Proc. 15th Int. Symp. on Software Reliability Engineering, p.417-428. [doi:10.1109/ISSRE.2004.35]

Huang, P., Zhu, J., 2008. Predicting the fault-proneness of class hierarchy in object-oriented software using a layered kernel. *J. Zhejiang Univ. Sci. A*, **9**(10):1390-1397. [doi:10.1631/jzus.A0720073]

Huang, S.J., Lin, C.Y., Chiu, N.H., 2006. Fuzzy decision tree approach for embedding risk assessment information into software cost estimation model. *J. Inf. Sci. Eng.*, **22**(2):297-313.

Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., 2007. Object-oriented software fault prediction using neural networks. *Inf. Software Technol.*, **49**(5):483-492. [doi:10.1016/j.infsof.2006.07.005]

Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P., Aud, S.J., 1997. Application of neural networks to software quality modeling of a very large telecommunications systems. *IEEE Trans. Neur. Networks*, **8**(4):902-909. [doi:10.1109/72.595888]

Khoshgoftaar, T.M., Allen, E.B., Deng, J., 2002. Using regression trees to classify fault-prone software modules. *IEEE Trans. Rel.*, **51**(4):455-462. [doi:10.1109/TR.2002.804488]

Kohavi, R., 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. Proc. 4th Int. Joint Conf. on Artificial Intelligence, p.1137-1143.

Maron, O., Lozano-Pérez, T., 1998. A Framework for Multiple Instance Learning. Proc. 10th Advances in Neural Information Processing Systems, p.570-576.

Reformat, M., Pedrycz, W., Pizzi, N.J., 2003. Software quality analysis with the use of computational intelligence. *Inf. Software Technol.*, **45**(7):405-417. [doi:10.1016/S0950-5849(03)00012-0]

Ruffo, G., 2000. Learning Single and Multiple Instance Decision Trees for Computer Security Applications. PhD Thesis, Department of Computer Science, University of Turin, Torino, Italy, p.425-432.

Seliya, N., Khoshgoftaar, T.M., 2007. Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Qual. J.*, **15**(3):327-344. [doi:10.1007/s11219-007-9013-8]

Tang, M.H., Kao, M.H., Chen, M.H., 1999. An Empirical Study on Object Oriented Metrics. Proc. 6th Int. Conf. on Software Metrics Symp., p.242-249.

Vishwanathan, S.V.N., Smola, A.J., Murty, M.N., 2003. Simple SVM. Proc. 20th Int. Conf. on Machine Learning, p.760-767.

Wang, J., Zucker, J.D., 2000. Solving Multiple-Instance Problem: A Lazy Learning Approach. Proc. 17th Int. Conf. on Machine Learning, p.1119-1125.

Weidmann, N., Frank, E., Pfahringer, B., 2003. A Two-level Learning Method for Generalized Multi-Instance Problem. Proc. European Conf. on Machine Learning, p.468-479. [doi:10.1007/b13633]

Zhang, M.L., Zhou, Z.H., 2004. Improve multi-instance neural networks through feature selection. *Neur. Process. Lett.*, **19**(1):1-10. [doi:10.1023/B:NEPL.0000016836.03614.9f]

Zhang, Q., Goldman, S.A., 2001. EM-DD: An Improved Multiple-Instance Learning Technique. Proc. 14th Advances in Neural Information Processing Systems, p.1073-1080.

Zhou, Z.H., Zhang, M.L., 2006. Multi-Instance Multi-Label Learning with Application to Scene Classification. Proc. Advances in Neural Information Processing Systems, p.1609-1616.

Zhou, Z.H., Jiang, K., Li, M., 2005. Multi-instance learning based Web mining. *Appl. Intell.*, **22**(2):135-147. [doi:10.1007/s10489-005-5602-z]