IEICE TRANS. INF. & SYST., VOL.E95–D, NO.6 JUNE 2012

LETTER

Active Learning for Software Defect Prediction

Guangchun LUO[†], ***Member***, **Ying MA**[†a)], ***and*** **Ke QIN**[†], ***Nonmembers***

SUMMARY An active learning method, called Two-stage Active learning algorithm (TAL), is developed for software defect prediction. Combining the clustering and support vector machine techniques, this method improves the performance of the predictor with less labeling effort. Experiments validate its effectiveness.

key words: *machine learning, defect prediction, active learning, support vector machine*

1. Introduction

Software defect prediction should identify defect-prone modules prior to the release of software. Many researchers apply supervised learning methods on software defect prediction when there exist sufficient labeled data [1]–[3]. At the same time, when labeled data is scarce, some researchers build semi-supervised classifiers to predict software defect modules [4], [5]. In practice, the labeled data are not available in many cases [6], [7], such as when a new project is started in a new domain, the defect prediction technique is applied for the first time, and fault data of the current software version might not be collected. In these cases, supervised learning approaches and semi-supervised learning approaches cannot be used because of the absence of labeled data. Labeled data are fairly expensive to obtain because they require human effort. To the best of our knowledge, few researchers have investigated new technology to support the defect prediction under this circumstance.

The absence of labeled data causes the failure of traditional software defect predictors. Although unsupervised learning has been applied, they also have unstable performance [8], [9]. Different from unsupervised learning, the active learning reduces the number of labeled instances required to achieve a stable performance in the majority of reported results [10]. In order to improve the performance and reduce the labeling efforts, we propose a method called Two-stage Active Learning algorithm (TAL). This method combines the clustering technique with support vector machine (SVM) technique in an active selection way. The main feature of active selection is to select the representative and informative data for labeling.

Our experimental results on real data sets show that TAL results in better performance than cluster method and random selective SVM method. Since obtaining defective modules data is difficult and time-consuming work, this method is helpful for practical application.

Manuscript received November 2, 2011.
Manuscript revised March 2, 2012.

[†]The authors are with University of Electronic Science and Technology of China, Chengdu, China.
a) E-mail: may@uestc.edu.cn

DOI: 10.1587/transinf.E95.D.1680

2. Related Work

There are a few software defect prediction studies which exploit cluster techniques in the predictors. One of the first reports for the defect prediction without prior fault data is [6]. They applied K-means and Neural-Gas algorithm to aggregate the software models into several clusters. Then, experts were required to identify each cluster as defective or non-defective. However, this clustering and expert-based approach is dependent on the capability of the expert who should be specialized in machine learning and software engineering areas.

Seliya et al. [4] proposed a constraint-based semi-supervised clustering scheme. In the first step, the scheme utilizes the labeled program modules for initial seeding of the clusters. Then, the software engineering experts are asked to label them as either defective or non-defective. However, the initial label data in the first step are required and the number of clusters is still a critical issue in this scheme.

Catal et al. [11] proposed a cluster and metrics thresholds based approach, which used metrics thresholds for removing the expert assistance. However, the selection of the number of clusters k is done heuristically, and this process can affect the model's performance drastically. In [12], they applied the X-means clustering method to cluster modules, and then the mean vector of each cluster was checked against the metrics thresholds vector. However, it is still difficult to determine the number of clusters and metrics thresholds.

These methods are all passive leaning methods, which can not use the query data information. We develop an active learning method to predict the defective modules.

3. Active Learning for Software Defect Prediction

3.1 Background

The main difference between active learning and passive learning is that, an active learning system asks the user to label only those instances that would be most informative to classification, while a passive learning system is trained on a large pool of randomly selected labeled data [10]. There are two settings in active learning literature, stream-based selective sampling and pool-based sampling. Pool-based sampling selects the training instances from the pool, which is usually assumed to be static, while stream-based selective sampling decides whether to query or discard the instances in sequence. In this research, we pay more attention to the pool-based sampling method.

There have been many proposed ways of formulating the query strategies in this setting. The *Uncertainty Sampling* is the most commonly used query strategy. This strategy selects the training instances whose classes are least certain to label. The *Query-By-Committee* is a theoretically-motivated query strategy, which has a generalization error of ε with only $O(\log 1/\varepsilon)$ labels. This method maintains a committee of models $\Omega = \{\theta^1, \dots, \theta^C\}$, which are all trained on the current labeled set. This strategy has two steps: randomly select two models; then select the instance to label, when it has different labels. The most informative query is considered to be the instances about which the two models are most disagree. The *Expected Model Change* strategy selects the instances which are likely to most influence the mode. Different form measuring model changes, the *Expected Error Reduction* strategy selects the instance, when the generalization error of the model is likely to be reduced. The *Variance Reduction* strategy reduces this generalization error indirectly by minimizing output variance, $E_T[(\hat{y}-y)^2]$, where $\hat{y}$ is shorthand for the model's predicted output for a given instance x, while y indicates the true label for that instance. The *Density-Weighted* strategy chooses uncertain and representative training instances to label.

These strategies are all proposed with some limited labeled data. However, the prior labeled data are not available in many cases [6], [7], these strategies cannot be applied in software defect prediction directly. Combining the clustering and SVM techniques, we develop a Two-stage Active Learning (TAL) method to deal with this problem.

3.2 Two-Stage Active Learning

In traditional defect predictor, $L = \{(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l)\} \subset X \times Y$ denotes the labeled data set with size l and $U = \{x_{l+1}, x_{l+2}, \dots, x_{l+u}\} \subset X$ denotes the unlabeled data set with size u. In our active learning settings, no labeled data is available at the very beginning, i.e., $l = 0$.

We can not use existing active learning methods directly, since we can not build classifier without any labeled data. In order to get the initial labeled data, which called representative data, we exploit the clustering technique. Then, we combine active learning techniques, SVM-based active learning, to get the informative data [10]. Note that the number of the representative data and informative data is very small, so the proposed method can greatly reduce the labeling efforts.

3.2.1 Representative Set Collection Stage

The centers of the clusters are the representative data in K-means [13]. Representative set denoted as $\Psi_R(D)$ consist of these representative data of the data set D. In order to find $\Psi_R(D)$, we exploit the clustering method. Let C_j denote the jth cluster, then $\bigcup_{j=1}^{k} C_j = D$, and $C_i \cap C_j = \emptyset$ $(i, j = 1, 2 \dots k, i \neq j)$, where k is the number of clusters. Let R_j denote the center of the cluster C_j, then $\Psi_R(D) = \bigcup_{j=1}^{k} R_j$. In K-means algorithm, the quality of the clustering is determined by the following error function [13]:

$$E = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - R_j\|^2 \tag{1}$$

where C_j is the jth cluster, $x_i \in U$, $i = 1, 2 \dots u$. By applying the K-means clustering technique, we can find the representative data. However, we found that the two classes may not be contained in the $\Psi_R(D)$, since the defective modules are rare in the data sets. We develop another clustering technique, called Cascade-cluster, based on K-means cluster technique. The pseudo-code of Cascade-cluster is shown as Algorithm 1.

Cascade-cluster iteratively calls the K-means cluster to find the centers of the clusters, until the centers with different labels are found. Since there are only two classes in our defect prediction, the centers of clusters are initialized by two random selected examples. Theoretically, $\lfloor \log_2 n \rfloor + 1$ (or $\lceil \log_2(n+1) \rceil$) calling times are needed in the worst case, where n is the number of instances.

Algorithm 1 Cascade-cluster algorithm.

```
Require:
        $D_u$: A data set of $u$ unlabeled instances
        $K$: K classes problem
Initialize:
        Random initial $R_1, R_2, \dots, R_K$ % K centers
 1: repeat
 2:    for each input instance $x \in D_u$ do
 3:       Calculate $j = \arg\min |x - R_i|, i = 1, 2, \dots, K$
 4:       Assign $x$ to the cluster $C_j$
 5:    end for
 6:    for each center $R_i$ do
 7:       Update $R_i = \sum_{x \in C_i} x / |C_i|$
 8:    end for
 9:    Compute Error according to (1)
10: until E not change or each cluster not change
11: Label the $y_i$ for each $R_i$
12: while $|\bigcup_{i=1}^{K} y_i| \neq K$ do
13:    for each cluster $C_i$ do
14:       Initial $D_u \leftarrow C_i$, repeat Cascade-cluster algorithm,
15:    end for
16: end while
```

3.2.2 Informative Set Collection Stage

We can build a classifier using the representative set, which are obtained from the first stage. Then, we focus on instances closest to the decision boundary of the classifier. These instances are easy misclassified, but contain very important information for building classifier [10]. Here, the set of all these instances is called informative set. The informative set of the data set D can be denoted as $\Psi_I(D)$.

In order to select the informative set $\Psi_I(D)$, we exploit the Support Vector Machine (SVM) technique. This technique attempts to find a maximum margin hyperplane, which is induced from available examples, called support vectors. This hyperplane maximizes the distance to the closest points from each class. The classification boundary will generalize best to future data, through this maximum margin way. In our settings, the informative point is closest to the decision boundary w vector [14]. The distance can be simply calculated by $|w \cdot x|$. If the data are nonlinearly separable, we can introduce the Mercer kernel. Then this value is $|w \cdot \Phi(x)|$. We repetitively select these points to form the $\Psi_I(D)$ set.

So far we have described how we get the two subsets $\Psi_R(D)$ and $\Psi_I(D)$ of the date set. Through this active learning process, we also build a SVM classifier, as shown in Algorithm 2. Existing active learning algorithms select the

Algorithm 2 Two-stage Active Learning (TAL).

```
Require:
        $D$: A data set of n instances
Initialize:
        $D_l = \emptyset$; $n_l = 0$ % no labeled data is available
        $D_u = D$; $n_u = n$ % the pool of unlabeled data
        $\Psi_R(D_u) = \emptyset$; $\Psi_I(D_u) = \emptyset$;
 1: Get the $\Psi_R(D_u)$, using Cascade-cluster algorithm on $D_u$
 2: Update $D_l = D_l \cup \Psi_R(D_u)$; $D_u = D_u/\Psi_R(D_u)$
 3: while the number of queries or the required accuracy is reached do
 4:     Biuld SVM classifier on $D_l$
 5:     Get the $\Psi_I(D_u)$
 6:     $D_l = D_l \cup \Psi_I(D_u)$; $D_u = D_u/\Psi_I(D_u)$
 7: end while
```

unlabeled data to label, until all the classes are found. Since positive instances are relatively rare in the software defect data set, many of the randomly chosen instances will be negative. These algorithms will cost a lot of label efforts. The Cascade-cluster can guarantee to find the positive instance. The simple SVM based active learning can reduce the label effort. By combing these two techniques, the proposed TAL algorithm can improve the performance of the predictor. It will be shown in the next experimental section.

4. Experiments

4.1 Data Set

All our source data come from the NASA data sets, which are collected at the function level [15]. Hence, a module is a function or method in our experiment.

pc1: This data set is from an Earth-orbiting satellite software that is no longer operational. It consists of 40 KLOC of C code. The software measurement data set contains 1107 modules, of which 76 (6.97%) have one or more defects and 1031 (93.03%) have no defect.

cm1: This data set is from a science instrument written in C code, with approximately 20 KLOC. It contains 498 modules, of which 49 (9.84%) have one or more defects and 449 (90.16%) have no defect.

They contain 21 attributes, five Loc attributes, three McCabe attributes, four Basic Halstead attributes, eight Derived Halstead attributes, and one BranchCount attribute.

4.2 Experimental Settings

In order to investigate the performances of our algorithms, we compare it with K-means Clustering, and Random selective SVM (Gaussian RBF kernel $\gamma = 10$, $c = 1000$). For each data set, we perform 5-fold cross validation, as per following details.

K-means Clustering: Let N be the number of labeled examples. Firstly, cluster each data set into N clusters, and then label each cluster as the class of the center. Since this method is not a classification algorithm, we randomly select N examples as initial data for five times.

Random selective SVM: Each data set is split into five folds, four folds are used as a pool, and the left one fold is used for testing. N examples are randomly selected from the pool for labeling. Then SVM is built on the N labeled examples, and tested on the testing fold.

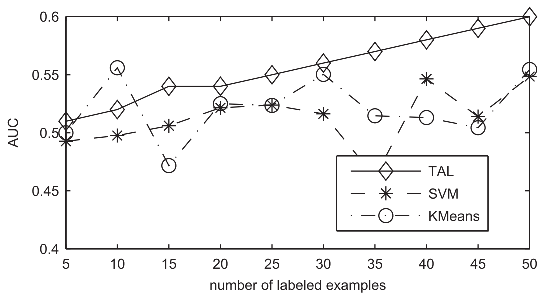
TAL: Each data set is also split into five folds, four folds are used as a pool, and the left one fold is used for testing. Algorithm 1 is used to select the representative data from the pool to label. Then, SVM is built on these labeled data. The informative data are selected from the rest data in the pool for labeling until the number of labeled examples N is reached, as shown in steps 3–7 in Algorithm 2. Note that the number N is the total number of the two stages. Finally, the obtained SVM is tested on the testing fold.

This operation allows for a variation of training and testing sets during the experiments. The number of labeled examples N is increased from 5 to 50 in our experiments.

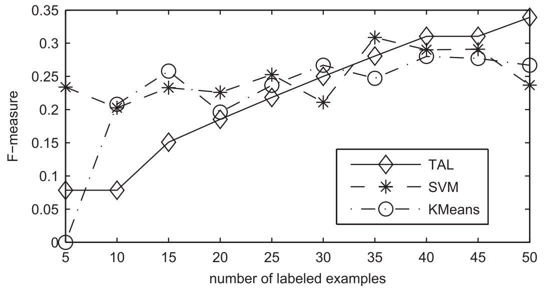
4.3 Experimental Result

We use AUC (area under the receiver operator characteristic curve) and $F\text{-}measure$ (the weighted harmonic mean of precision and recall) performance metrics which are commonly used in the field of software defect prediction.

The results for all the three methods are shown in Figs. 1, 2. We can see that TAL is superior to other algorithms in the aspect of AUC. The $F\text{-}measure$ rate of TAL increases with increasing labeled data. When more than 35 instances are labeled, the TAL outperforms other algorithms in the aspect $F\text{-}measure$. Both of the figures show that TAL learning algorithm can improve the performance, by providing selected queries. That is to say, the quality of the classifier not only depends on the absolute number of labeled instances, but also depends on the representation and information of the labeled instances.

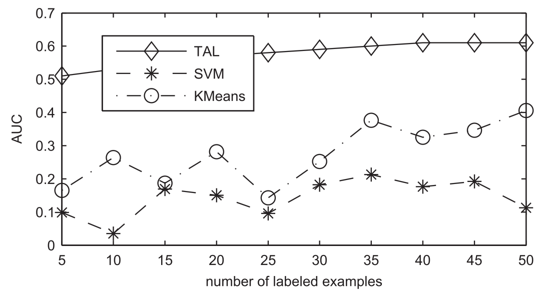


(a) AUC performance

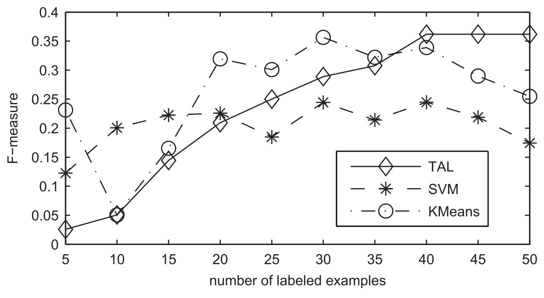


(b) *F-measure* performance

Fig. 1 Performances compared on cm1.



(a) AUC performance



(b) *F-measure* performance

Fig. 2 Performances compared on pc1.

5. Conclusion

In this paper, we introduce the active learning method into software defect prediction. The proposed algorithm TAL exploits the clustering technique and SVM technique to select the representative and informative data for labeling. Experimental results on real data sets validate its effectiveness.

Acknowledgments

This work was supported in part by the fundamental research funds for new century excellent talents in university (NO. NCET-10-0298) and foundation of science and technology department of Sichuan province (NO. 2011GZ0192). We appreciate the encouraging suggestions from anonymous reviewers.

References

[1] C. Catal, "Review: Software fault prediction: A literature review and current trends," Expert Systems with Applications: An International Journal, vol.38, no.4, pp.4626–4636, 2011.

[2] O. Mizuno and T. Kikuno, "Prediction of fault-prone software modules using a generic text discriminator," IEICE Trans. Inf. & Syst., vol.E91-D, no.4, pp.888–896, April 2008.

[3] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," Inf. Softw. Technol., vol.54, no.3, pp.248–256, 2012.

[4] N. Seliya and T.M. Khoshgoftaar, "Software quality analysis of unlabeled program modules with semi-supervised clustering," IEEE Trans. Syst. Man Cybern., A, Syst. Humans, vol.37, no.2, pp.201–211, 2007.

[5] C. Catal and B. Diri, "Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction," Expert Systems, vol.26, no.5, pp.458–471, 2009.

[6] S. Zhong, T.M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," IEEE Intelligent Systems, vol.19, no.2, pp.20–27, 2004.

[7] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Trans. Softw. Eng., vol.33, no.1, pp.2–13, 2007.

[8] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," Proc. 4th International Workshop on Predictor Models in Software Engineering (PROMISE 2008), pp.47–54, 2008.

[9] T.M. Khoshgoftaar and N. Seliya, "Software quality classification modeling using the SPRINT decision tree algorithm," Proc. 4th IEEE International Conference on Tools with Artificial Intelligence, pp.365–374, 2002.

[10] B. Settles, "Active learning literature survey," Computer Sciences Technical Report 1648, University of Wisconsin-Madison, http://active-learning.net/

[11] C. Catal, U. Sevim, and B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules," Proc. 6th International Conference on Information Technology: New Generations, pp.199–204, 2009.

[12] C. Catal, U. Sevim, and B. Diri, "Metrics-driven software quality prediction without prior fault data," Electronic Engineering and Computing Technology, Electronic Engineering and Computing Technology Series: Lecture Notes in Electrical Engineering, vol.60, pp.189–199, 2010.

[13] A.K. Jain, "Data clustering: 50 years beyond K-means," Pattern Recognit. Lett., vol.31, no.8, pp.651–666, 2010.

[14] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," J. Machine Learning Research, vol.2, pp.45–66, 2001.

[15] G. Boetticher, T. Menzies, and T. Ostrand, The PROMISE Repository of Empirical Software Engineering Data, 2007, http://promisedata.org/repository