IEICE TRANS. INF. & SYST., VOL.E95–D, NO.7 JULY 2012

LETTER

Asymmetric Learning Based on Kernel Partial Least Squares for Software Defect Prediction

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

SUMMARY An asymmetric classifier based on kernel partial least squares is proposed for software defect prediction. This method improves the prediction performance on imbalanced data sets. The experimental results validate its effectiveness.

key words: *defect prediction, class imbalance, kernel partial least squares, machine learning*

1. Introduction

Software defect prediction is an essential part of software quality analysis and has been extensively studied in the domain of software-reliability engineering [1]–[5]. However, As pointed out by Menzies et al. [2] and Khoshgoftaar et al. [4], the class imbalance problem encountered in real-world data sets often degrades the performance of defect predictors. The software defect data set is class imbalanced when the majority of defects in a software system are located in a small percentage of the program modules. Existing approaches to solving the class imbalance problem mainly include data-level and algorithm-level methods, which are compared in [4]. Their results show that the algorithm-level method AdaBoost almost always outperforms even the best data-level methods in software defect prediction. Most recently, Qu et al. [6] proposed an asymmetric classifier APLSC, which is based on linear partial least squares, to tackle the class imbalance problem.

In this paper, we develop a kernel based asymmetric learning method, called Asymmetric Kernel Partial Least Squares Classification (AKPLSC), which is able to nonlinearly extract the favorable features and retrieve the loss caused by class imbalance problem.

2. Asymmetric Kernel Partial Least Squares Classifier for Software Defect Prediction

Linear Partial Least Squares (PLS) [7] is an effective linear transformation, which performs the regression on the subset of extracted latent variables. Kernel PLS [8] first performs nonlinear mapping $\Phi : \{x_i\}_{i=1}^n \in \mathbb{R}^N \to \Phi(x) \in \mathbb{F}$ to project an input vector to a higher dimensional feature space. Then linear PLS is used in this feature space.

In software defect prediction, $L = \{(x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell)\} \subset X \times Y$ denotes the labeled example set with size ℓ and $U = \{x_{\ell+1}, x_{\ell+2}, \ldots, x_{\ell+u}\} \subset X$ denotes the unlabeled example set with size u. For labeled examples, $Y = \{+1, -1\}$, the defective modules are labeled '+1', the non-defective modules are labeled '−1'. Software defect data sets are highly imbalanced, i.e. the examples of the minority class (defective modules) are heavily under-represented in comparison to the examples of majority class (non-defective modules).

Given the center M, the radius of the class region r, and the parameter of overlapping η, the relationship of the two classes can be expressed as $M_{+1} - M_{-1} = \eta(r_{+1} - r_{-1})$. The parameter η indicates the level of overlapping between the region of the two classes (The smaller value of η, the higher overlapping). APLSC suffers from the high overlapping, especially when the data sets are nonlinear separable [6].

In order to overcome this overlapping problem, a kernel method is exploited here. Kernel PLS [8] corresponds to solving the eigenvalue equation as follows:

$$\Phi\Phi^T\Psi\Psi^T\tau = \lambda\tau \tag{1}$$

where Φ and Ψ denote the matrix of mapped X-space data $\Phi(x)$ and the matrix of mapped Y-space data $\Psi(y)$ in the feature space $\mathbb{F}$, respectively. The nonlinear feature selection methods can reduce the overlapping level of the two classes, but the class imbalance problem makes them fail to distinguish the minority class [6]. In order to retrieve the loss caused by class imbalance problem, we want to get the bias $\hat{b}$ of the kernel PLS Classification, KPLSC [8].

APLSC can be expressed as $\hat{Y} = sign\Big(\sum_{i=1}^k m_i t_i - b\Big)$, which is derived from the regression model of the linear PLS, $\hat{y} = \sum_{i=1}^k m_i t_i$, where k is the number of the latent variables, t_i is the ith score vector of testing data, m_i indicates the direction of ith score, and the bias b is equal to $m_1(M_{+1} - r_{+1}\eta)$. Different from APLSC, the kernel PLS regression is $\hat{y} = \sum_{i=1}^\ell \alpha_i \kappa(x_i, x)$, where ℓ is the size of labeled example set, $\kappa(x_i, x)$ is a kernel function, and α_i is dual regression coefficient. Then AKPLSC can be expressed as:

$$\hat{Y} = sign\Big(\sum_{i=1}^{\ell} \alpha_i \kappa(x_i, x) - \hat{b}\Big) \tag{2}$$

where α_i is dual regression coefficient, which can be obtained from kernel PLS, as shown in Algorithm 1. $\hat{b}$ is the bias of the classifier.

Since kernel PLS put most of the information on the

Manuscript received December 13, 2011.
Manuscript revised March 5, 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.2006

Table 1 Metrics used in the experiment.

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

Table 2 Comparative performance evaluation for the six methods.

data	RUS	AdaBoost	PLSC	APLSC	KPLSC	AKPLSC
cm1	0.676	0.725	0.534	0.542	0.756	0.761
pc1	0.769	0.846	0.505	0.508	0.794	0.800
kc3	0.696	0.690	0.734	0.743	0.791	0.801

Algorithm 1 AKPLSC

Require:
Labeled and unlabeled data sets, L and U; number of components, k
Ensure:
Asymmetric Kernel Partial Least Squares Classifier, H;
$K_{ij} = \kappa(x_i, x_j), i, j = 1, \ldots, \ell, x_i, x_j \in L$;
$K_1 = K, \hat{Y} = Y$ % K is the kernel matrix, Y is the label vector.
for $j = 1, \ldots, k$ **do**
  $\beta_j = \beta_j / \|\beta_j\|$ % β_j is projection directions
  repeat
    $\beta_j = \hat{Y}\hat{Y}' K_j \beta_j$
    $\beta_j = \beta_j / \|\beta_j\|$
  until convergence
  $\tau_j = K_j \beta_j$ % τ_j is the score
  $c_j = \hat{Y}' \tau_j / \|\tau_j\|^2$ % c_j is the direction of the score
  $\hat{Y} = \hat{Y} - \tau_j c_j'$ % $\hat{Y}$ is the deflation of Y
  $K_{j+1} = (I - \tau_j \tau_j' / \|\tau_j\|^2) K_j (I - \tau_j \tau_j' / \|\tau_j\|^2)$
end for
$B = [\beta_1, \ldots, \beta_k], T = [\tau_1, \ldots, \tau_k]$
$\alpha = B(T' K B)^{-1} T' Y$; % α is the vector of dual regression coefficients
Calculate $\hat{b}$ according to Eq. (3);
$H(x) = sign\left(\sum_{i=1}^{\ell} \alpha_i \kappa(x_i, x) - \hat{b} \right), x \in U$;
return H;

first dimension, the bias in the AKPLSC can be computed similarly as [6]:

$$\hat{b} = c_1 * (M_{+1} - r_{+1}\eta) = c_1 * \frac{M_{+1} r_{-1} + M_{-1} r_{+1}}{r_{-1} + r_{+1}} \qquad (3)$$

where c_1 indicates the direction of the first score τ_1, the centers (M_{+1}, M_{-1}) and radiuses (r_{+1}, r_{-1}) are computed based on τ_1, which can be obtained from Eq. (1). After centering[†] the data, AKPLSC can be described as Algorithm 1.

3. Experimental Result

The experimental data sets come from NASA projects [9], which are developed in different languages, at different sites by different teams, as shown in Table 3. cm1 is drawn from a NASA spacecraft instrument project, pc1 is from a flight software for earth orbiting satellite, and kc3 is from a software for the collection, processing and delivery of satellite meta data. All of metrics are given by [9], as shown in Table 1. cm1 and pc1 contain 21 metrics, 5 different lines of code metrics, 3 McCabe metrics, 4 base Halstead metrics, 8 derived Halstead metrics, 1 branch-count metric. kc3 also contains another 18 metrics.

Table 3 Data sets.

data	language	#modules	#attr.	size(loc)	%defective
cm1	C++	498	21	14,763	9.83
pc1	C	1,109	21	25,924	6.94
kc3	JAVA	458	39	7,749	9.39

In order to investigate the performance of AKPLSC (Gaussian kernel $\kappa(x, y) = \exp(-\|x - y\|^2)$ is used here), we compare it with random undersampling (RUS) [4], AdaBoost [4], Partial Least Squares Classifier (PLSC) [7], APLSC [6], and KPLSC [8]. For each data set, we perform a 10×5-fold cross validation. We use the area under a receiver operating characteristic curve (AUC) performance metric, which is commonly used in software defect prediction research area, to evaluate the performance of a classifier. The results for the six methods are shown in Table 2. We can see that AKPLSC outperforms other methods on all the data sets, except for pc1, compared with AdaBoost.

4. Conclusion

A new kernel-based asymmetric learning algorithm, AKPLSC, is proposed for software defect prediction. This method can nonlinearly extract the feature information and retrieve the loss caused by class imbalance in software defect data sets. Experiments validate its effectiveness.

Acknowledgments

We thank R. Rosipal for providing us with the KPLSC code,

[†]Centering data moves the origin to the center of mass [8]: $K = K - \frac{1}{\ell} JJ'K - \frac{1}{\ell} KJJ' + \frac{1}{\ell^2}(J'KJ)JJ'$, where J is the all 1s vector.

and LeVis Group for APLSC code. 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] T.M. Khoshgoftaar, E.B. Allen, and J. Deng, "Using regression trees to classify fault-prone software modules," IEEE Trans. Reliab., vol.51, no.4, pp.455–462, 2002.

[2] 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.

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

[4] C. Seiffert, T.M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," IEEE Trans. Syst. Man Cybern. A, Syst. Humans, vol.39, no.6, pp.1283–1294, 2009.

[5] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," Proc. 15th International Symposium on Software Reliability Engineering (ISSRE 2004), pp.417–428, 2004.

[6] H.N. Qu, G.Z. Li, and W.S. Xu, "An asymmetric classifier based on partial least squares," Pattern Recognit., vol.43, no.10, pp.3448–3457, 2010.

[7] M. Barker and W.S. Rayens, "Partial least squares for discrimination," J. Chemometrics, vol.17, no.3, pp.166–173, 2003.

[8] R. Rosipal, L.J. Trejo, and B. Matthews, "Kernel PLS-SVC for linear and nonlinear classification," Proc. 20th International Conference on Machine Learning (ICML 2003), pp.640–647, 2003.

[9] http://promisedata.org/repository