Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A novel kernel to predict software defectiveness

Ahmet Okutan[a,*], Olcay Taner Yildiz[b]

[a] Mobipath Erenet Ltd, Maltepe, Istanbul, Turkey
[b] Department of Computer Engineering, Isik University, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 18 June 2015
Revised 3 March 2016
Accepted 3 June 2016
Available online 11 June 2016

Keywords:
Defect prediction
SVM
Kernel methods

ABSTRACT

Although the software defect prediction problem has been researched for a long time, the results achieved are not so bright. In this paper, we propose to use novel kernels for defect prediction that are based on the plagiarized source code, software clones and textual similarity. We generate precomputed kernel matrices and compare their performance on different data sets to model the relationship between source code similarity and defectiveness. Each value in a kernel matrix shows how much parallelism exists between the corresponding files of a software system chosen. Our experiments on 10 real world datasets indicate that support vector machines (SVM) with a precomputed kernel matrix performs better than the SVM with the usual linear kernel in terms of *F*-measure. Similarly, when used with a precomputed kernel, the *k*-nearest neighbor classifier (KNN) achieves comparable performance with respect to KNN classifier. The results from this preliminary study indicate that source code similarity can be used to predict defect proneness.

1. Introduction

Many software engineering projects run out of budget and schedule. This is one of the biggest problems that the software development industry has met so far and many attempts have been made to increase the success rate of the software projects. One possible solution is defect prediction, that is, to predict a software defect or failure before it is observed and take necessary mitigating actions.

Not all defects have the same priority considering their effects in the maintenance phase. For example, a severe bug in an accounting software can be very critical and may have a high priority, whereas a problem in the tool tip text of some screen control may not be so important. A software module with one non-critical bug is much more preferable to a module with many critical bugs. The number of bugs together with their severity are two important factors to decide on the extent of defect proneness.

Deciding on the defectiveness of a software is very critical and important to plan the testing and maintenance phases of a software project. First, in testing period, it is possible to focus more on the defect prone modules or modules where there are more errors comparatively. Second, since more defects are fixed during the test period, the maintenance cost of the project decreases and this causes a decrease in the total cost of the project also.

There are numerous studies in the literature about defect prediction using classification and regression techniques. In the classification case, the software modules (classes or files) are marked as defective or not, that is, a binary classification problem is solved. The focus is on defect proneness rather than its extent. However in the regression case, the number of faults in each module is estimated and the emphasis is on the number of faults rather than defect proneness. There are studies where either a specific statistical method is researched or the performance of several statistical methods are compared. For example, (Pickard et al., 1999) compare the efficiency of residual analysis, multivariate regression and classification and regression trees (CART) on the software datasets which were created by simulation. Giancarlo Succi and Stefanovic (2001) compare Poisson regression model with binomial regression model to deal with software defect data that is not distributed normally. They observe that the zero-inflated negative binomial regression model shows the best ability to describe the high variability in the dependent variable. Schneidewind (2001) shows that logistic regression method is not very successful alone. But when used together with Boolean discriminant functions (BDF) it gives more accurate results.

In recent years, the amount of research done on defect prediction using machine learning algorithms has increased slightly compared to traditional methods (Catal and Diri, 2009). Many algorithms have been studied and as a consequence, some of these algorithms have been marked to be better than others on the selected data sets. We believe that, most of the time it is difficult to generalize the defect prediction results. According to Myrtveit

* Corresponding author.
E-mail addresses: ahmet.okutan@gmail.com (A. Okutan), olcaytaner@isikun.edu.tr (O. Taner Yildiz).

http://dx.doi.org/10.1016/j.jss.2016.06.006
0164-1212/© 2016 Elsevier Inc. All rights reserved.

et al. more reliable research procedures are needed to have confidence in the comparative studies of software prediction models (Myrtveit et al., 2005). Furthermore, D'Ambros et al. state that defect prediction is a field where external validity is very hard to achieve (D'Ambros et al., 2012). Menzies et al. show that an approach useful in global context is often irrelevant in local contexts in defect prediction studies (Menzies et al., 2011). Rule induction (Shepperd and Kadoda, 2001), regression (Shepperd and Kadoda, 2001; Ekanayake et al., 2009), case-based reasoning (CBR) (Khoshgoftaar et al., 1997; 2000; Shepperd and Kadoda, 2001), decision tree approaches like C4.5 (Song et al., 2006), random forest (Kim et al., 2015; Guo et al., 2004; Kaur and Malhotra, 2008), linear discriminant analysis (Munson and Khoshgoftaar, 1992), artificial neural networks (Khoshgoftaar et al., 1995; Thwin and Quah, 2002; Kaur et al., 2009; Shepperd and Kadoda, 2001), *k*-nearest neighbor (Boetticher, 2005), *K*-star (Koru and Liu, 2005), Bayesian networks (Fenton et al., 2002; Pai and Dugan, 2007; Zhang, 2000; Okutan and Yildiz, 2012) and support vector machine based classifiers (Lessmann et al., 2008; Hu et al., 2009; Jin and Liu, 2010; Shivaji et al., 2009; Xing et al., 2005; Gondra, 2008) are machine learning algorithms that are used in the fault prediction literature.

In our previous work, we proposed a novel kernel to predict the number of defects. We showed that support vector machines (SVM) with a precomputed kernel matrix performs as good as the SVM with linear or RBF kernels, in terms of the root mean square error (RMSE). We also have shown that the proposed regression method is comparable with other regression methods like linear regression and KNN (Okutan and Yildiz, 2013). In this paper, we extend our study and focus on the software defect prediction as a classification problem and consider the similarities of code patterns among different files of a software system to predict defectiveness. Although the classification method we suggest can be used to predict defectiveness, the novelty of our work is the proposition of a new kernel rather than a new defect prediction method. To reveal the relationship between the source code similarity and defectiveness, the precomputed kernel matrix is used with *K*-nearest neighbor classifier in addition to SVM. Furthermore, to extract similarity between any pair of files and generate kernels, clone detection and information retrieval techniques are used.

This paper is organized as follows: In Section 2, we give a background on kernel machines. In Section 3, we present a brief review of previous work on kernel methods and software defect prediction in general. We explain our proposed approach in Section 4 and give the experiment results in Section 5 before we conclude in Section 6.

2. Kernel machines

2.1. Support vector machines

Let's assume that we have a training set

$$S = \left\{(\mathbf{x}_i, y_i), \mathbf{x}_i \in R^t, y_i \in \{-1, 1\}\right\}_{i=1}^{N} \tag{1}$$

where y_i's are either +1 (positive class) or -1 (negative class) and each $\mathbf{x}_i$ vector (with t entries) belongs to one of these classes. Based on this assumption, a hyperplane is defined as

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} - b \tag{2}$$

where $\mathbf{w}$ shows the vector normal to the optimal hyperplane and $b/\|\mathbf{w}\|$ is the offset of the hyperplane from the origin on the direction of $\mathbf{w}$.

Support vector machines generate the optimal hyperplane (or hypersphere, depending on the kernel) that can be used for classification or regression (Cortes and Vapnik, 1995). The optimal hyperplane is found by maximizing the margin which is defined as the distance between two nearest instances from either side of the hyperplane. As it is shown in the Fig. 1, the hyperplane is $\frac{1}{\|\mathbf{w}\|}$ away from each class and it has a margin of $\frac{2}{\|\mathbf{w}\|}$.

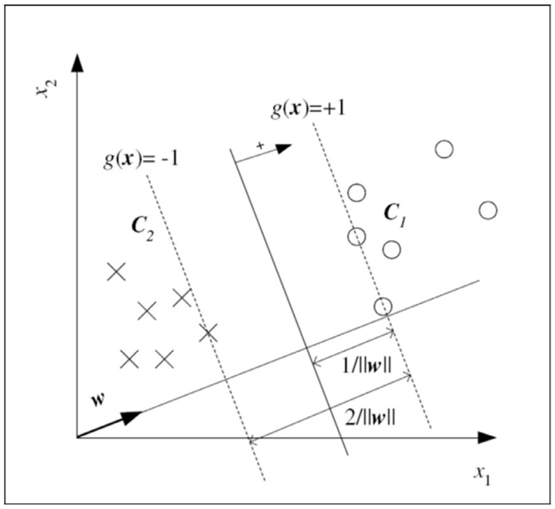

Fig. 1. An optimal separating hyperplane (Alpaydın, 2004).

In order to calculate the margin of the optimum hyperplane, we need to find the boundaries of the two classes as hyperplanes first and then take the distance among these two hyperplanes. The boundary hyperplane of the positive class can be written as $g(\mathbf{x}) = 1$ whereas the boundary hyperplane of the negative class is $g(\mathbf{x}) = -1$.

Assuming that we have a linearly separable data set, the distance to the origin is $(b+1)/\|\mathbf{w}\|$ for the first hyperplane and $(b-1)/\|\mathbf{w}\|$ for the second one. The distance between these two hyperplanes is $2/\|\mathbf{w}\|$. The optimum hyperplane separating these two classes maximizes this margin or minimizes $\|\mathbf{w}\|$.

For positive instances we have

$$\mathbf{w}^T\mathbf{x}_i - b \geq 1 \tag{3}$$

and for negative instances we have

$$\mathbf{w}^T\mathbf{x}_i - b \leq -1 \tag{4}$$

These two constraints can be combined as $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$. Now our problem becomes an optimization problem of

$$\text{Minimize } \|\mathbf{w}\| \quad \text{s. t. } y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 \tag{5}$$

To control the sensitivity of SVM and tolerate possible outliers, slack variables (ξ_i) are introduced. After adding constant $C > 0$, which determines the relative importance of maximizing the margin and minimizing the amount of slack, the problem changes slightly and becomes an optimization problem of

$$\begin{aligned} &\text{Minimize } \|\mathbf{w}\| + C\sum \xi_i \\ &\text{s. t. } y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 - \xi_i \end{aligned} \tag{6}$$

In this new representation, we observe that if $0 < \xi_i \leq 1$, the data point lies between the margin and the correct side of the hyper plane. On the other hand if $\xi_i > 1$, then the data point is misclassified. Data points that lie on the margin are called support vectors and regarded as the most important data points in the training set.

If the data points are not linearly separable, one can use a suitable transformation function to carry the data points to a higher dimension where linear separation is possible. The transformation is based on the dot product of two vectors as $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i)^T\mathbf{x}_j$. Assuming that the transformation function θ is defined as θ: $\mathbf{x} \rightarrow \Phi(\mathbf{x})$, our new dot product in the high dimensional space becomes $K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \Phi(\mathbf{x}_i)^T, \Phi(\mathbf{x}_j) \right\rangle$. So, the kernel function can be defined