Reducing False Alarms in Software Defect Prediction by Decision Threshold Optimization

Ayşe Tosun[1], Ayşe Bener[2]
Software Research Laboratory,
Computer Engineering Department,
Bogazici University
Istanbul, Turkey
{[1]ayse.tosun, [2]bener}@boun.edu.tr

Abstract

Software defect data has an imbalanced and highly skewed class distribution. The misclassification costs of two classes are not equal nor are known. It is critical to find the optimum bound, i.e. threshold, which would best separate defective and defect-free classes in software data. We have applied decision threshold optimization on Naïve Bayes classifier in order to find the optimum threshold for software defect data. ROC analyses show that decision threshold optimization significantly decreases false alarms (on the average by 11%) without changing probability of detection rates.

1. Introduction

Defect prediction is a powerful technique proposed so far by many researchers in order to estimate the potential defective modules (i.e. methods, files or classes) in the software systems [2,3,6,9,12]. Erroneous predictions of defect-free modules as defective (false alarms) force testers to inspect "safe" modules and waste their precious time. On the other hand, missing defective modules (false negatives) would cause more expensive and hard-to-fix failures in the final software product.

Software defect data generally has an imbalanced nature with very few number of defective modules compared to defect-free ones [5]. Misclassification costs are unequal and class distributions are highly skewed. There are studies aiming to estimate the relative costs of false positive errors vs. false negative errors in software defect prediction [4]. However, it is difficult and much debatable to define these misclassification costs for different software systems. Various sampling strategies have been applied in order to overcome the imbalanced software data problem [6,7,12]. Kamei et al. used four sampling strategies on legacy software and concluded that sampling techniques could not always improve the prediction performance of classifiers [7]. Turhan et al. proposed nearest neighbor sampling [6] on cross-company data in the absence of within-company defect data. Results show that nearest neighbor sampling is statistically better than random sampling approach [6]. In a recent study, Maloof showed that Receiver Operating Characteristics (ROC) curves produced by sampling strategies such as over-sampling or under-sampling are similar to those produced by varying the decision threshold or the cost matrix [8]. ROC curve plots the true positive rate vs. false positive rate by adjusting the decision threshold [1]. ROC analysis provides statistically consistent and more representative performance assessment than misclassification rate. Lessmann et al. used Area Under Curve measure using ROC curves to assess the performance of various classification algorithms on software defect prediction [9]. Menzies et al. also utilized ROC curves to discuss the effects of increasing the information content of software data [12]. Decision threshold optimization on two dimensional ROC curves has been investigated in order to balance this tradeoff between TP and FP rate in various machine learning applications, such as pattern recognition [8].

In this paper, we have proposed a simple and powerful analysis on ROC curves to improve the prediction performance of Naïve Bayes classifier. We have adjusted the hyper parameter of Naïve Bayes classifier, which is the decision threshold, on thirteen public software datasets. Our results show that default decision threshold (0.5) is not always the best threshold for imbalanced software data. Decision threshold optimization on Naïve Bayes significantly

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

decreases false alarms from 34% to 23%, while keeping detection rate and balance rate the same.

2. Data

In this study, thirteen software datasets are used, all of which are publicly available in Promise repository [5]. First five datasets are from a white-goods manufacturer that operates in embedded industry, while the rest of them are from NASA MDP repository. Defect rates are around 6% to 32%, showing the imbalanced nature in all datasets. Number of static code attributes, i.e. size and complexity metrics extracted from the source code, varies from 21 to 39. Finally, they all have a class label, either 1 as defective and 0 as defect-free. Table 1 shows the software datasets, number of defective instances, number of defect-free instances, overall defect rate (%) and the number of attributes, respectively.

Table 1. Datasets Used In This Study

	# defectives	#defect-free	defect rate	# attributes
ar1	2103	8777	19.33	21
ar3	326	1783	15.46	21
ar4	107	415	20.50	21
ar5	43	415	9.39	39
ar6	52	109	32.30	39
jm1	9	112	7.44	29
kc1	8	55	12.70	29
kc2	20	87	18.69	29
kc3	8	28	22.22	29
mc2	15	86	14.85	29
pc1	77	1032	6.94	37
pc3	160	1403	10.24	37

3. Methodology

We have used Naive Bayes classifier as the prediction algorithm due to several reasons: a) it is simple, easy to interpret and a robust method, b) the outputs of the classifier are posterior probabilities which can be directly used in our approach, and c) recent study by Lessmann et al. [9] shows that most of the machine learning algorithms are significantly indifferent from each other in software defect prediction. Naive Bayes assumes that each attribute is independent, normally distributed and equally important [11]. It is derived from simple Bayes Theorem such that posterior probability of an instance x being in class Ci is proportional to prior probability of Ci and the likelihood p(x|Ci). It is normalized using evidence, p(x):

$$p(C_i|x) = \frac{p(x|C_i)\,p(C_i)}{p(x)} \tag{1}$$

Eq. (2) shows the classification task for binary classification problem such that an instance *x* is assigned to the first class, in our case, *defective* class, if its posterior probability is greater than or equal to the default threshold, i.e. 0.5, otherwise, it is assigned to the second class, i.e. *defect-free* class. In cases when class distributions are imbalanced, using default decision threshold would degrade the classification performance. Thus, it is critical to find the optimum decision threshold for each dataset and to improve the prediction performance in software defect prediction.

$$\begin{cases} C_1, & \text{if } \; p(C_1|x) \geq 0.5 \\ C_2, & \text{otherwise} \end{cases} \tag{2}$$

3.1. ROC Analysis

In machine learning, the use of ROC analysis has increased after the realization of the weakness of simple error rate as a performance measure [8]. In this study, we have used ROC curves to evaluate the discriminative performance of Naïve Bayes classifier. Given a classifier and an instance, the prediction outcomes, depending on actual class labels of instances, can be represented as a confusion matrix in Table 2. Common classifier performance metrics have been derived from the confusion matrix [11].

Table 2. Confusion Matrix

Actual	**Predicted**	
	Defective	**Defect-free**
Defective	TP	FN
Defect-free	FP	TN

TP Rate is a measure of accuracy for correctly detecting defective instances and is equal to the ratio of the number of true positives (TP) over the sum of true positives (TP) and false negatives (FN). TP Rate corresponds to *probability of detection rate (pd)* in software defect prediction.

$$\text{TP Rate}\,(pd) = TP/(TP + FN) \tag{3}$$

FP Rate represents the number of false alarms that is the false positives (FP) over the sum of false positives (FP) and true negatives (TN). FP Rate also corresponds to *probability of false alarms (pf)* in software defect prediction, as similar to various machine learning studies.

$$\text{FP Rate}\,(pd) = FP/(FP + TN) \tag{4}$$

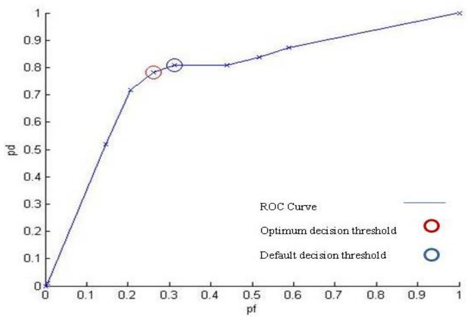


Figure 1. ROC curve for kc1 dataset

When using Naive Bayes classifier, *pd* and *pf* are been calculated for a single decision threshold (0.5) that maps to a single point on the ROC curve. However, Provost clearly defined that, "*when studying problems with imbalanced data, using the classifiers produced by standard machine learning algorithms without adjusting the output threshold may well be a critical mistake*" [10]. Since, software data has imbalanced class distributions, it is necessary to evaluate the performance of classification for different thresholds. To do this, we have changed Eq. (2) as follows:

$$\begin{cases} C_1, & \text{if} \quad p(C_1|x) \geq t \\ C_2, & \text{otherwise} \end{cases} \tag{5}$$

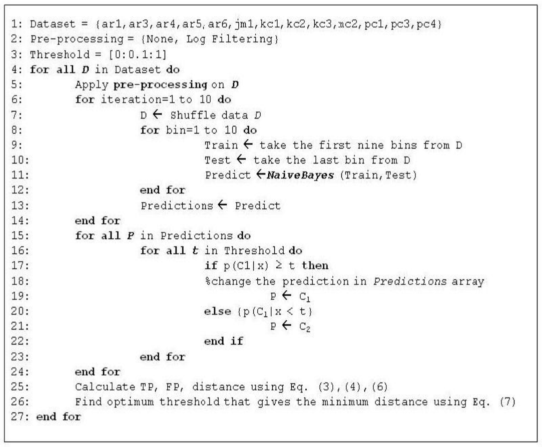
All *pd* and *pf* values have been calculated by varying the decision threshold, *t,* in the range of [0:0.1:1]. The resulting set of (*pd, pf*) pairs are plotted as a 2D ROC curve (Figure 1) that takes into account all possible solutions of the threshold variation.

In Figure 1, the lower left point (0,0) represents assigning all instances to defect-free class. Hence, there are no defective instances and this yields *pd* and *pf* to be 0. On the contrary, upper right corner (1,1) indicates assigning all instances to defective class. The upper left point (0,1) represents perfect classification. Therefore, the threshold value that gives the nearest point to (0,1) is accepted as the optimum decision threshold (*topt*). The distance between any (*pd, pf*) and (0,1) can be calculated using the below formula:

$$\text{distance(pd,pf)} = \sqrt{(1-pd)^2 + (0-pf)^2} \tag{6}$$

ROC curve demonstrates the effect of threshold optimization on the variation of *pd* and *pf*. Classification with the default decision threshold (0.5) produces *pd* and *pf* whose Euclidean distance to the ideal point (0,1) is 0.4201, while it is 0.4093 with t_{opt} = 0.6. Choosing a point on the left-hand side of the t_{opt} reduces *pf*, but often has lower *pd* as well. Thresholds on the right hand-side increase both *pd* and *pf*. In order to find the optimum decision threshold, Eq. (7) is used:

$$t_{opt} = \arg\min_t \text{distance}(\text{pd}, \text{pf}) \tag{7}$$



```
Dataset = {ar1,ar3,ar4,ar5,ar6,jm1,kc1,kc2,kc3,mc2,pc1,pc3,pc4}
Pre-processing = {None, Log Filtering}
Threshold = [0:0.1:1]
for all D in Dataset do
    Apply pre-processing on D
    for iteration=1 to 10 do
        D ← Shuffle data D
        for bin=1 to 10 do
            Train ← take the first nine bins from D
            Test ← take the last bin from D
            Predict ←NaiveBayes (Train,Test)
        end for
        Predictions ← Predict
    end for
    for all P in Predictions do
        for all t in Threshold do
            if p(C1|x) ≥ t then
            %change the prediction in Predictions array
                P ← $C_1$
            else {p($C_1$|x < t}
                P ← $C_2$
            end if
        end for
    end for
    Calculate TP, FP, distance using Eq. (3),(4),(6)
    Find optimum threshold that gives the minimum distance using Eq. (7)
end for
```

Figure 2. Pseudocode for the experiments

4. Experiments and Results

The pseudo code for decision threshold optimization using ROC curves can be seen in Figure 2. We have conducted our experiments using 10-fold cross validation to overcome sampling bias [11]. We have applied log filtering on software data to improve prediction performance of Naïve Bayes [2]. For all datasets, we have plotted ROC curves and taken the optimum threshold that is closest to the (0,1) point in terms of (pf,pd). Due to space limitations, we did not put all figures. Instead, we have presented our experimental results in terms of *pd, pf* and *bal* rates in Table 3. *Balance* measure is simply (1-*distance*) which is computed to assess how far our predictions are from the ideal case. The higher the *balance*, the better the prediction performance is.

Results in Table 3 show that optimal decision threshold changes between 0.5 and 0.8 depending on the class distribution in datasets. T-tests with 5% significance level indicates that *pd* and *bal* rates with and without threshold optimization are significantly indifferent from each other. However, *pf* rates are significantly greater when decision threshold optimization is not used. Although results show that we have decreased *pd* rates, on the average, from 79% to 70%, these changes are not statistically significant. Therefore, it is seen that using a standard machine learning algorithm (Naïve Bayes) with its default hyper-parameter, i.e. decision threshold as 0.5, is a critical mistake for software defect data.

5. Conclusion

This study proposes a two-dimensional ROC analysis to improve the prediction performance of Naïve Bayes classifier on software defect prediction.

Table 3. Experimental Results

	NB with default threshold (t_0=0.5)			NB with optimum threshold (t_{opt})			
	pd	**pf**	**bal**	**t_{opt}**	**pd**	**pf**	**bal**
ar1	0.70	0.33	0.56	0.5	0.70	0.33	0.56
ar3	1.00	0.35	0.65	0.8	0.80	0.13	0.76
ar4	0.80	0.39	0.56	0.6	0.70	0.31	0.57
ar5	0.90	0.23	0.75	0.6	0.80	0.10	0.78
ar6	0.65	0.42	0.45	0.6	0.55	0.31	0.45
jm1	0.50	0.30	0.42	0.5	0.50	0.30	0.42
kc1	0.82	0.37	0.59	0.7	0.65	0.20	0.60
kc2	0.85	0.26	0.70	0.6	0.84	0.21	0.74
kc3	0.90	0.33	0.66	0.6	0.83	0.25	0.70
mc2	0.60	0.40	0.43	0.6	0.50	0.18	0.47
pc1	0.73	0.32	0.58	0.6	0.70	0.22	0.63
pc3	0.85	0.37	0.60	0.6	0.76	0.27	0.64
pc4	0.91	0.30	0.69	0.7	0.82	0.17	0.75
avg	**0.79**	**0.34**	**0.59**	**-**	**0.70**	**0.23**	**0.62**

We have changed the decision threshold on Naïve Bayes and observed the changes in prediction performance measures. Results on thirteen public software data show that we have managed to keep *pd* and *bal* rates the same, while decreasing *pf* rates significantly. Using decision threshold optimization on Naïve Bayes classifier, *pf* rate has decreased, on the average, from 34% to 23%. *Balance* rate has increased from 59% to 62%, and the pd rates remained the same. These results are also validated using paired t-test.

This study also has practical contributions. Decision threshold optimization helps to decrease false alarms while keeping high pd rates. This would in turn decrease the verification cost, i.e. testing effort for finding the actual defective modules in the software systems. Furthermore, a simple technique applied on a simple and robust classifier is easier to be understood and interpreted by the practitioners, instead of more complex models. As a future direction, we will extend this study by introducing the reject option [13]. Although we have improved false alarm rates via decision threshold optimization, there are still instances whose posterior probabilities are not confident enough to make predictions. Reject option would help us to eliminate such less confident classifications.

Acknowledgment

This research is supported in part by Turkish Scientific Research Council, TUBITAK, under grant number EEEAG108E014.

6. References

[1] T. Fawcett, "An Introduction to ROC Analysis", Pattern Recognition Letters, vol. 27, issue 8, 2006, pp. 861-874.

[2] T. Menzies, J. Greenwald, A. Frank, "Data mining static code attributes to learn defect predictors", IEEE Transactions on Software Engineering vol. 33, issue 1, pp. 2-13, 2007.

[3] A. Tosun, B. Turhan, A. Bener, "Ensemble of software defect predictors: a case study", Proc. ESEM, October 2008, pp. 318-320.

[4] R. Moser, W. Pedrycz, G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction", Proc. ICSE, 2008, pp. 181-190.

[5] T. Boetticher, T. Menzies, T. Ostrand, "Promise repository of empirical software engineering data" http://promisedata.org/ repository (2007) West Virginia University, Department of Computer Science.

[6] B. Turhan, A. Bener, T. Menzies, "Nearest neighbor sampling for cross company defect predictors", (abstract only), Proc. DEFECTS, 2008, p. 26.

[7] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, K. Matsumoto, "The effects of over and under sampling on fault-prone module detection", Proc. ESEM, 2007, pp. 196-204.

[8] A.M. Maloof, " Learning when data sets are imbalanced and when costs are unequal and unknown" Workshop on Learning from Imbalanced Data Sets, 2003.

[9] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings", IEEE Trans. on Software Engineering, vol. 34, 2008, pp. 485-496.

[10] F. Provost, "Machine learning from imbalanced data sets 101", Proc. Working Notes AAAI00 Workshop Learning from Imbalanced Data Sets, 2000, pp. 1-3.

[11] Alpaydin, E., "Intoduction to Machine Learning", MIT Press, October 2004.

[12] B. Turhan, T., Menzies, A. Bener, J. Distefano, "On the Relative Value of Cross-company and Within-Company Data for Defect Prediction", Empirical Software Engineering Journal, 2009, in print. DOI 10.1007/s10664-008-9103-7.

[13] C. K. Chow, "On optimum recognition error and reject tradeoff", IEEE Transactions on Information Theory, volume 16, 1970, pp. 41-46.