A practical method for the software fault-prediction

Zhan Li, Marek Reformat
ECE Department, University of Alberta
{zhanli, reform}@ece.ualberta.ca

Abstract

In the paper, a novel machine learning method, SimBoost, is proposed to handle the software fault-prediction problem when highly skewed datasets are used. Although the method, proved by empirical results, can make the datasets much more balanced, the accuracy of the prediction is still not satisfactory. Therefore, a fuzzy-based representation of the software module fault state has been presented instead of the original faulty/non-faulty one. Several experiments were conducted using datasets from NASA Metrics Data Program. The discussion of the results of experiments is provided.

1. Introduction

It is obvious that the reliability is an important facet to evaluate software. However, processes of identifying and locating faults in software projects are difficult, especially when the project size grows. Today, more and more efforts have been put in analyzing and testing software. Several different techniques are used to perform these tasks, for example extreme point combination (EPC) strategy, weak n x 1 strategy, and cause-effect graphs in the case of black box testing; and flow graph (CFG), and data dependency graph (DDG) for the white box testing [1]. The results of these strategies and techniques depend a lot on the knowledge, skills and experience of test engineers.

The increasing popularity and effectiveness of machine learning algorithms have created possibilities for researchers to focus on building software fault-prediction models. The inputs of these models are software metrics derived from a source code. Software metrics quantify different aspects of a product generated during a software project [2]. The well-known metrics are Halstead metrics [3], McCabe metrics [4], system design metrics [5], and functional metrics [6]. The model representing relations between software metrics and defects found in the software modules is constructed using datasets from previous software projects. When new datapoints representing modules are input to the trained model, the modules are classified to be faulty or non-faulty ones.

There is a growing concern that the metrics are not good enough to make reliable predictions. Some researchers suspect that the relation between software metrics and many facets that are used as outputs of models is not strong enough [7].

However, a model predicting software faults should be a good supporting tool for human-based testing procedures. The goal of our research is to establish a practical method, which could be used to extract information for identifying defects in software modules based on source code features.

The paper is organized as follows: Section 2 provides detailed description and analysis of the dataset used for the experiments. The proposed method is presented in sections 3 and 4. In the Section 5, the description of experiments and their results are provides. Section 6 contains the conclusions.

2. Software datasets

The datasets from NASA's Metrics Data Program (MDP) are used in our research. The repository at NASA IV & V Facility MDP provides a number of datasets representing non-specific software projects. Each dataset includes Halstead metrics, McCabe metrics, line of code metrics, and the errors state data at the module level. In our software defect prediction, we use the error state data as class labels for prediction, i.e., faulty modules as one class, and non-faulty modules as the other class. This feature is called the target feature, and the metrics representing a module are called input features.

2.1 Description of the datasets

The MDP dataset contains data from thirteen software projects. The two we selected for our experiments are JM1 and PC1.

In each project, the data have been presented by eight files, which are "*PN_product_hierarchy.csv", "PN_defect_product_relations.csv","PN_dynamic_defect_data.csv",* "*PN_static_defect_data.csv", "PN_requirement_product_relations.csv*", *"PN_product_module_metrics.csv", "PN_odc_activity.csv*", and *"PN_odc_target.csv".*

In the file names, the "PN" stands for the name of the project. For example, for the JM1 project the name of the first file would be *jm1_product_hierarchy.csv*. The detailed information about each file can be found at *http://mdp.ivv.nasa.gov/about.html*.

1-4244-1500-4/07/$25.00 ©2007 IEEE

The two files, which are of interest for us are *PN_product_hierarchy.csv* and *PN_defect_product_relations.csv*. The *PN_product_hierarchy.csv* file represents the relationship between the metrics of the modules and their error states. Each data point, i.e. each row in the file, sets the *ModuleID* as its primary key. Namely, each row in this file contains metrics and the error state representing one module. On the other hand, the *PN_static_defect_data.csv* provides the information about each defect in each module, i.e., the relationship between the *ModuleID* and the *DefectID*. Note that one module could have several defects; so one *ModuleID* could be connected to multiple *DefectIDs*.

Obviously the information embedded inside the *PN_product_hierarchy.csv* is important for our software defect prediction research. It is the basis for constructing models to find the relations/rules between the software metrics and the error states. Almost all of the research papers (Section 2.2) that are related to this NASA data are based only on this file.

However, the information embedded in the file *PN_static_defect_data.csv* is also useful. In Section 6 we provide detailed description on how to use it.

2.2 Related works and the existed problems

There are a number of published papers that use the NASA dataset for the research experiments. Taghi M. Khoshgoftaar, et al [8] used the JM1 dataset to construct 25 different prediction models. He used supervised machine learning algorithms: several different decision tree algorithms, several types of neural networks and SVM. Shi Zhong, et al. [9], on the other hand, applied clustering (unsupervised) instead of classification (supervised) methods to process the same dataset. A. Gunes Koru, et al. [7] tried to find a relationship between the size of module and the accuracy of prediction. These papers indicated difficulties in processing the data, and obtaining satisfactory results. There are a few possible reasons for that.

First, the data is highly skewed. In JM1 dataset, after prepossessing (elimination of erroneous datapoints), there are 7675 non-defective modules compared with 1732 defective ones. Therefore, even if all of the modules are recognized as the non-defective ones, the accuracy could still be over 77%. To better explain this, please refer to Table 1.1, which provides an example of confusion matrix.

Table 1.1 Confusion matrix

Actual \ Predicted	Faulty	Non-faulty
Faulty	***a***	***b***
Non-faulty	***c***	***d***

Based on the confusion matrix, we can define sensitivity, specificity and accuracy.

Sensitivity= a/(a+b)
Specificity=d/(d+c)
Accuracy= (a+d)/(a+b+c+d)

In the software fault prediction project, it is obvious that the sensitivity is more crucial than the specificity. In the real software project, the result of misclassification of non-faulty modules is not as critical as of the faulty modules. However, the consequence of the skewed data is that the models experience relatively high accuracy, but their sensitivity is very low.

Highly skewed data sets seem to be the main obstacle in obtaining better results. In this paper, a new method is proposed that addresses this issue and is able to efficiently decrease the effect of the unbalanced dataset. The detailed description of the method is presented in Section 3.

Second, as we mentioned in the introduction, some researchers suspect that the relationship between the software metrics and the faulty/non-faulty feature is not strong enough for accurate predictions.

The faults in the modules are generated very subjectively, i.e., human is the only generator of both a source code and errors. Nevertheless, researchers try to separate the modules by the objective features, i.e., software metrics. The metrics may not be enough to support a process of classification modules as non-faulty ones and faulty ones. Given two similar modules that are generated by a skilled engineer and a newbie, they can belong to different categories (faulty/non-faulty). Kanglin Li said that "… Any software product, no matter how accomplished by today's testing technologies, has bugs. …" [10]. Thus, it might be not practical to classify the modules into only two categories faulty and non-faulty, because all of them are faulty ones.

In the case of the JM1 dataset, Gary D. Boetticher [11] conducted several experiments concentrating on the input features, i.e., software metrics, instead of target feature, i.e., faulty/non-faulty. He found out that some modules are close to each other (by Euclidean distance) in the input space (features) but have different target features. Based on the experiment of separating the dataset into two halves, the conclusion was "the model would be unable to recognize that 18 of the test vectors are classified with 100 percent accuracy" [11]. This statement can be seen as one more argument indicating that the software metrics are not enough to built very accurate fault prediction models.

However, software metrics could still provide us with valuable information regarding the possibility of modules being faulty or non-faulty, which can be useful in the real world.

In addition, the knowledge of the *DefectID* might increase the accuracy. It seems that the defects instead of modules are the atoms that are represented by software metrics. In many cases one module has several different defects. Therefore, the relation between defects and software metrics is valuable as well.

As we mentioned earlier, the *DefectID* is the ID for different kinds of defects, and the *ModuleID* is the ID for each module. In our experiments, the *DefectID* and the *ModuleID* are combined in the training dataset, while the testing datasets are kept unchanged.

3. Explanation of the method

As we mentioned earlier, in the dataset JM1, the rate of the number of faulty modules and non-faulty ones is less than 2:8, which is a big obstacle for obtaining models with good accuracy. Moreover, the faulty modules are more crucial.

The problem of unbalanced data is not unique for software data, and researchers had already proposed several methods to handle it. The most popular methods are over/under sampling method [12, 13] and cost matrix method. However, both of these two methods have their own drawbacks. The over/under sampling changes the distribution of the training set to balance the data, while the cost matrix needs a additional process of adjusting matrix parameters.

In this paper, we proposed a method, which is called SimBoost, to handle the highly skewed data. The main idea behind this algorithm is to separate the data set into several subsets. In some of these subsets, the data is so skewed that we need only pay attention to a dominating class; while in others, the data is much more balanced. Those balanced subsets could be further process with different machine learning algorithms. Furthermore, the process could be iterated till the subsets are "good" enough to be accepted.

The method is presented in Figure 3.1. There are basically two units: *New Target Feature Assignment Unit* (NFAU), and *Model Processing Unit* (MPU). The NFAU assigns a new target feature to the data set, and the MPU performs prediction

The NFAU pre-processes the original data points. To accomplish that, a model, called *Model_pre*, is constructed. First of all, the *Model_pre* is built based on the training dataset. For each data point, the predicted value is compared with the original one, and a new feature *f'* is assigned to this data point based on the result of this comparison. In our experiments, the original target feature is the error state of the module, i.e., faulty/non-faulty, and the new target feature is presented Table 3.1. In short, this *Model_pre* performs separation of the training dataset, based on the assumption ***that some non-faulty modules, which induce the data to be unbalanced, have some obvious characteristics to be recognized***. Therefore, the non-faulty modules that are classified as non-faulty ones are assigned the class label "*0*" as the new target feature. The rest of the module have the label "*1*". The data points with the new target feature are the input to the MPU.

Table 3.1 Definition of the new feature

Actual target feature	Predicted target feature	*f'*
Non-faulty	Non-faulty	0
Faulty	Faulty	1
Faulty	Non-faulty	1
Non-faulty	Faulty	1

The MPU, in the Figure 3.2, contains two models *Model_1* and *Model_2* (please refer to Section 5 for the details on how to build these two models). The *Model_1* performs classification based on the new feature *f'*. In a nutshell, the *Model_1* separates the modules/data points that are "obviously" non-faulty ones from the rest of data points. The outputs of *Model_1* are two datasets, *dataset_EASY* and *dataset_HARD*.

Most of data points of the sub-dataset *dataset_EASY* belong to a dominated class. This means that this data set is very skewed, and no model will be built using that data.

The sub-dataset *dataset_HARD*, on the other hand, should be a much more balanced dataset. *Model_2*, Figure 3.2, is to handle this balanced dataset. Moreover, in the *Model_2*, the original target feature (faulty/non-faulty) is restored. So, *Model_2* is the model that does the faulty/non-faulty predictions on the less skewed sub-dataset (*dataset_HARD*) of the original dataset.

4. Principles of fuzziness

Fuzzy set theory introduced by Zadeh in 1965, provides an arithmetic description on the real world [14]. In the classic sets theory, everything belongs to a crisp set. For example, if A is an apple, it is not an orange, i.e., if A belongs to apple set, it means A does not belong to the orange set. However, in the real world, this is not always the case. For example, let us look at two sets of people – young and old. A 10-year-old boy belongs to the set young, and a 90-year-old man belongs to the set old, what about a 40-year-old person? The benefit of fuzzy sets is that they provide a mathematical way of expressing "**DEGREE OF MEMBERSHIP**". With the help of fuzzy sets, we could say that the 40-year-old person belongs to the set young with a degree of 0.6, and to the set old with a degree of 0.4.

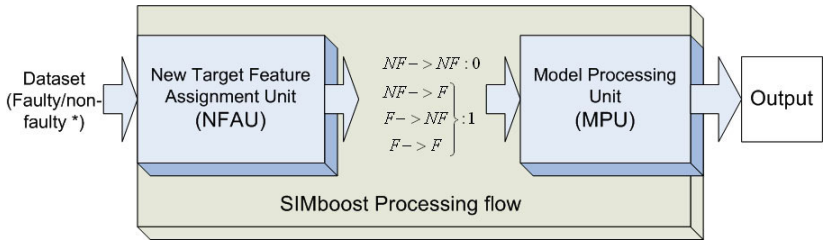

Figure 3.1 The process of the SimBoost

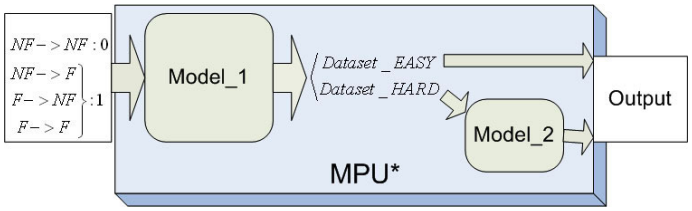

Figure 3.2 The detailed process on the MPU

There are lots of attributes that would affect the actual module error state, which cannot be represented by the software metrics. Examples of such attributes are: the difficulty of the programming objective, the background of the engineers, or the working environment. This means that an attempt to classify modules into only two classes (faulty/non-faulty) is almost impossible. To overcome that problem, we decide to use fuzzy categories – "Safe", "Confusing" and "Dangerous" as the target of our prediction process. As the names indicate "Safe" modules mean modules with less possibility of being faulty ones; "Dangerous" means modules that are very prone to have faults; and "Confusing" represents module that have faults, but the possibility that a module has faults is not as strong as in the case of "Dangerous" modules. When prediction models are built with such classes, an extra knowledge is provided to the manager/QA engineer to support their decisions. For instance, the module predicted as "Dangerous" deserves stricter and more comprehensive test cases, while the "Confusing" module may need only several basic tests.

Note that the definitions of these three categories are quite flexible. For example, if one company has enough resources (engineers, time and money) to perform extensive testing, it can apply strict requirements on how "Safe" modules are classified. This would lead to a small number of the "Safe" modules, but the company would have high confidence in prediction of these modules. On the other hand, if the company believes that their programmers are good, and there is no need to do too many tests, the requirements for "Dangerous" modules can be very strict. This would lead to high confidence in prediction of "Dangerous" modules, and their number would be small.

In our experiments, modules in the "Safe" category are those modules which are identified by *Model_1* as modules with the new target feature *f'=0*. These modules constitute the *dataset_EASY*, Figure 3.2. The *dataset_HARD* is further separated into "Dangerous" category and "Confusing" category.

In order to construct *Model_2*, Figure 3.2 (see details in Section 5), we set the target value as a numeric one instead of a nominal one, i.e., we assign "0" to non-faulty modules, and "1" to faulty ones. Therefore, the prediction based on the *Model_2* generates a number between "0" and "1". In this case, the value of a special threshold that is used to distinguish different categories should be set. For example, a module is a "Dangerous" one when the prediction value is over the threshold, while it is a "Confusing" one when the value is below the threshold.

5. Experiments and results

The JM1 dataset contains some noisy data points. After prepossessing, the number of data points suitable for experiments decreased from 11,470 to 9,407.

For the purpose of this paper, we conduct two series of experiments. In each series, there are ten independent experiments. The difference between these two series is the type of the *Model_pre* in the NFAU. We, select different models as the *Model_pre* to find out what influence a type of *Model_pre* has on the results. The two models that are considered are the SVM [15] and C4.5 [16]. All results are presented as the average of the ten single tests. C4.5 is also applied as the *Model_1* of the MPU in both series of experiments. In the case of the *Model_2* of the MPU, since the numeric output is required, a multi-layer perceptron network is used.

Table 5.1 Prediction results for the *Model_1*

Experiment results						
	TrS2		TS		KCTS	
SeriesSVM						
Easy	20.78% ± 1.99%	347.1 ± 35.84	22.20% ± 2.80%	249.3 ± 28.89	43.65% ± 9.75%	257.1 ± 57.42
Faulty	22.39% ± 1.86%	77.7 ± 9.43	21.46% ± 2.79%	46 ± 11.07	27.77% ± 1.39%	71.4 ± 17.70
Non-faulty	77.60% ± 1.86%	269 ± 29.19	78.54% ± 2.79%	177 ± 20.03	72.23% ± 1.39%	185.7 ± 40.01
Hard	79.22% ± 1.98%	1321.9 ± 29.01	77.80% ± 2.80%	875 ± 42.14	56.35% ± 9.75%	331.9 ± 57.42
Faulty	49.78% ± 1.27%	657.9 ± 26.30	43.43% ± 1.16%	379.8 ± 15.39	40.19% ± 1.39%	132.6 ± 17.70
Non-faulty	50.22% ± 1.27%	664 ± 17.14	56.57% ± 1.16%	495.2 ± 30.46	59.81% ± 1.39%	199.3 ± 40.01
SeriesC4.5						
Easy	21.49% ± 1.71%	357.5 ± 28.25	23.61% ± 2.76%	263.8 ± 34.3	44.86% ± 10.79%	264.2 ± 63.57
Faulty	23.69% ± 2.28%	84.7 ± 9.82	23.05% ± 2.48%	60.8 ± 8.30	30.02% ± 2.63%	79.3 ± 17.22
Non-faulty	76.30% ± 2.28%	272.8 ± 21.22	76.95% ± 2.48%	203 ± 29.12	69.98% ± 2.63%	184.9 ± 47.34
Hard	78.51% ± 1.71%	1307.3 ± 46.30	76.39% ± 2.77%	852.2 ± 29.68	55.14% ± 10.79%	324.8 ± 63.57
Faulty	49.36% ± 1.15%	662.3 ± 33.54	43.87% ± 1.54%	373.9 ± 12.97	38.39% ± 2.80%	124.7 ± 17.21
Non-faulty	50.64% ± 1.15%	645 ± 20.63	56.13% ± 1.54%	478.3 ± 25.85	61.61% ± 2.80%	200.1 ± 47.34

In addition, for the purpose of our experiments we combined the *ModuleID* and the *DefectID* based on the files "*JM1_product_module_metrics.csv*" and "*JM1_defect_product_relations.csv*" in the training phase. The combination of *ModuleID* and *DefectID* resulted in a new ID for each data point, i.e., each data point has its unique combination of *ModuleID* and *DefectID*, instead of only *ModuleID*. On the other hand, in the testing phase, we keep the data set unchanged, since in the real software prediction application, the prediction is based on the modules.

The WEKA [17] has been applied for our experiments. WEKA is an open source machine learning software in Java. Because of our complicated process, we wrote our own program with the "inclusion" of the WEKA libraries to guarantee its flexibility.

5.1 Preparation of the training and testing dataset

The JM1 dataset is split into 3 parts, which are called training set 1(TrS1), training set 2(TrS2), and testing set (TS). The rate of the split is 30%, 30% and 40%, respectively. TrS1 and TrS2 are used to train all of the models in the Figure 3.2, while the TS is the dataset for testing.

Besides, to make the results more plausible, we also use KC1, which is a data package from another NASA project, as another testing dataset. This dataset is called KC testing set (KCTS).

Please note just as we mentioned before, in the TrS1 and TrS2, *ModuleID* and *DefectID* have been combined, while the TS and KCTS are unchanged.

5.2 Training *Model_pre* and generation of the new target feature

The task of *Model_pre* is to assign the new target feature *f'* (Section 3) for each module. The *Model_pre* is developed based on the training set 1 (TrS1).

Initially, the target feature is the defect state, i.e., faulty or non-faulty. From Table 3.1, it is obvious that the new target feature is also a binary output. When the original target feature of a module is non-faulty and the *Model_pre* predicts this module as a non-faulty one, the new target feature assigned to the module is "0"; while in the other situations, the new target feature is "1". Let us assume that the category that has "0" in the new target feature is defined as the *dataset_EASY*, and the one that has "1" as the *dataset_HARD*. We could find out that the data points in the *dataset_EASY* are basically non-faulty modules, and the data points in the *dataset_HARD* have to be further classified. Because a significant number of non-faulty modules have been moved out, the *dataset_HARD* data set is more balanced.

5.3 Obtaining the *Model_1*

The construction of *Model_1* starts with preprocessing of data. This preprocessing is done using the *Model_pre*. In our experiments, the dataset TrS2 is being "pushed" through the *Model_pre*. As the result, the new target feature *f'* is assigned to the datapoints. The model built by C4.5 algorithm is our *Model_1*. Since *Model_pre* is built using TrS1, TrS2 is the testing set for that model. At the same time, TrS2 is the training set for the *Model_1*. The 10-fold cross validation is

applied during the training of *model_1*, in order to avoid over-fitting.

Model_1 is the model that classifies data points into *dataset_EASY* or *dataset_HARD*. Once the model is built both TS and KCTS1 are applied as testing sets. From the results of the two series, SeriesSVM and SeriesC4.5 of experiments, we find that highly skewed data set is not such an issue any more. Table 5.1 presents both average and standard deviation of the results of these experiments. In each single experiment, the datapoints of three datasets, TrS2, TS, and KCTS, have been split into *dataset_EASY* and *dataset_HARD*.

In the experiments, the rate between faulty and non-faulty modules in the *dataset_HARD* is in the range from about 2:3 to about 1:1 which is much more balanced than that in the original JM1 data set (around 2:8). Then, ***if the highly skewed data is the main issue for the accuracy of the prediction, we can expect better classification of the dataset_HARD.***

However, it turns out that the accuracy of the prediction of *dataset_HARD* is not as good as we expected. Table 5.2 shows the classification results for four different models based on the *dataset_HARD* of TrS2 for SeriesSVM. The models SVM, C4.5, Multilayer Perceptron(MLP), and Naïve Bayes classifier, are used as the *Model_2*. The presented results are based the 10-fold cross-validation.

Table 5.2 Results of four regular models

Model	SVM	C4.5	MLP	NaiveBayes
Accuracy	63.34%	68.81%	64.02%	61.92%

As it can be observed based on Table 5.2, the accuracy is less than 70%. Although considering the conclusions of Gary D. Boetticher's experiments (Section 2.2), they seem to be not too bad, but still unacceptable. After all, few software managers would like to depend on a less-than-70%-accuracy model.

5.4 Generation of the fuzzy Model_2

Table 5.1, indicates that the *dataset_EASY* contains around 75% of the modules that belong to the non-faulty category. This means that we can accept *dataset_EASY* as the "Safe" category of modules. Now, we concentrate on the *dataset_HARD*, in which the rate of faulty to non-faulty modules is closer to1:1.

Therefore, we need to generate a model that will assign fuzzy labels "Dangerous" and "Confusing" to datapoints from the *dataset_HARD*. The basic method of doing that is building a model, which has numerical outputs. Thus, the original target feature is changed from the nominal feature, faulty/non-faulty, to the numeric. The output values between "0" and "1" represent the degree of membership of a datapoint to the category faulty. The boundary value of "0" means non-faulty, and of "1" means faulty. Moreover, a threshold has been set to divide the output into two categories "Dangerous" and "Confusing". If the threshold is 0.5, then when a data point goes through the *Model_2* and the output is over 0.5 the label would be "Dangerous"; otherwise it would be "Confusing". In our experiments, we use the *dataset_HARD* part of TrS2 as the training set, and the sets TS and KCTS as the testing sets. Table 5.3 shows the results of the experiments.

A close inspection of Table 5.3 indicates that basically most of the modules in the "Dangerous" category are faulty ones. The only issue is the testing results for KCTS, where the "Dangerous" category only takes a very small part (less than 10%). It is obviously because the KCTS is another project, and the projects JM1 and KC1 have differences. For example, in JM1 there are over 10,000 modules while in KC1 there are only around 2,000 modules. Furthermore, the rates of faulty modules to non-faulty modules in these two projects are also different; JM1 has 22.5% of faulty modules, while KC1 has 26% of faulty modules. The distributions of the "Safe", "Confusing" and "Dangerous" modules for these two projects are also different. The analysis of the results lead to a very interesting conclusion regarding comparison of JM1 and KC1 projects. It seems that faulty modules of both projects vary a lot when compared with the non-faulty modules. This conclusion seems reasonable, since it is not difficult to imagine that the faulty modules are induced by many different factors when they belong to different projects.

5.5 Additional verification experiments on the new SimBoost method

The output space of the SimBoost method is a 3-attribute set {Safe, Confusing, Dangerous}, while the regular method output belongs to a 2-attribute set {faulty, non-faulty}. Therefore, it is not straightforward to compare the new method with the existed ones. The results of several additional experiments are provided as an attempt to perform such comparison.

First of all, the datasets and results of the first experiment in the SeriesSVM have been selected for comparision. Like we have said, the JM1 data set is split into three sets: training set 1 (TrS1), training set 2 (TrS2), and testing set (TS). The training sets TrS1 and TrS2 have been combined and used as training set for building the "conventional" model. We decided, based on Table 5.2, to use C4.5 as the conventional model. Once the model C4.5 has been trained, it is used to classify points form the testing set (TS).

Table 5.3 Results on the *dataset_Hard*

SeriesSVM				
TrS2(hard)	Non-faulty	Faulty	Non-faulty	Faulty
Confusing	63.84% ± 2.02%	38.22% ± 2.02%	546 ± 27.70	326.9 ± 27.10
Dangerous	29.06% ± 0.41%	70.94% ± 0.41%	135.6 ± 9.84	331 ± 23.05
TS(hard)				
Confusing	64.50% ± 2.19%	35.50% ± 2.19%	388.9 ± 35.14	214 ± 15.56
Dangerous	39.07% ± 3.04%	60.93% ± 3.04%	106.3 ± 14.11	165.8 ± 13.71
KCTS(hard)				
Confusing	61.51% ± 1.93%	38.49% ± 1.93%	171.5 ± 36.70	107.3 ± 15.67
Dangerous	52.35% ± 4.73%	47.65% ± 4.73%	27.8 ± 5.18	25.3 ± 4.13
SeriesC4.5				
TrS2(hard)				
Confusing	61.45% ± 1.69%	38.55% ± 1.69%	535.5 ± 34.84	335.9 ± 26.53
Dangerous	29.09% ± 0.38%	70.91% ± 0.38%	126.8 ± 14.34	309.1 ± 38.21
TS(hard)				
Confusing	63.95% ± 1.91%	36.05% ± 1.91%	383.5 ± 22.93	216.2 ± 14.2
Dangerous	37.54% ± 2.42%	62.46% ± 2.42%	94.8 ± 19.52	157.7 ± 18.97
KCTS(hard)				
Confusing	64.26% ± 3.03%	35.74% ± 3.03%	177.3 ± 47.78	98.6 ± 17.15
Dangerous	46.63% ± 3.61%	53.37% ± 3.61%	22.8 ± 6.69	26.1 ± 6.57

Table 5.4 TS results on SimBoost and C4.5

C4.5 / *SimBoost*	Faulty	Non-faulty
Safe	52	230
Confusing	251	283
Dangerous	235	50

(a)

Actual->C4.5 / *SIMBoost*	NF->F	F->NF	F->F	NF->NF
Safe	*33*	*45*	*19*	*185*
Confusing	*121*	*75*	*130*	*208*
Dangerous	*94*	*18*	*141*	*32*

(b)

Because the testing set is the same as in the case of SimBoost model, we can trace each data point and calculate statistics required for comparison. Table 5.4 displays the results.

Table 5.4 (a) implies that most of the data points in the category "Safe", based on SimBoost method, are classified as non-faulty modules by the C4.5 method. While most of the data points in the "Dangerous" category are classified as faulty modules. The most interesting part in the experiment is related to the category "Confusing". In this part, the ratio faulty to non-faulty modules is almost 1:1, which could mean that the data points that has been recognized as "Confusing" represent "module characteristics" that are hard to judge. It seems more realistic to put them into a "Confusing" category and treat them differently than either faulty or non-faulty modules.

Table 5.4 (b) displays the prediction results of the C4.5-based model divided into each category of the SimBoost. A number of conclusions can be drawn based on the analysis of the results. First of all, let us look at the column NF->F. Out of 248 (33+121+94) points misclassified by C4.5, 94 points were classified as "Dangerous" by SimBoost. This represents about 38% of data points that were misclassified by C4.5. On the other hand, the column "F->NF" represents low sensitivity and "sneaking" faulty modules through (Section 2.2). In this case, SimBoost misclassified about 33% of the cases that C4.5 misclassified (45/(45+75+18)).

It is also reasonable to analyze the "Confusing" category and compare it with the "Safe" and "Dangerous" categories. We can find that 208 modules of the "NF->NF" column (properly classified by C4.5 as non-faulty) are in the "Confusing" category instead of

"Safe". Such distribution can be explained by the classification rate of *Model_1*. For "F->F" column, the scenario is similar – there are 130 modules in the category "Confusing" instead of "Dangerous". The interesting fact that can be observed for this column is that only 19 modules would be not recognized. As the result SimBoost would miss 64 (45+19) faulty modules. It is not bad but the price for that is a large number of modules in the category "Confusing".

7. Conclusion

In this paper, we present the result of our work on the NASA project datasets. The prediction of the faulty/non-faulty modules was far from perfect when conventional approaches were used to build prediction models. Therefore, we proposed a novel method, called SimBoost. The results of the experiments indicated that the method could remarkably reduce the effect of the skewed dataset, which at first was regarded as the primary reason of the non-good prediction results. However, the prediction on more balanced data was still not accurate. To address that issue we proposed "fuzzy" labels for classification purposes. The results of these experiments demonstrated that application of fuzzy labels gives interesting prediction results.

Reference

[1] Jeff Tian, *Software quality engineering: Testing, quality assurance, and quantifiable improvement*, John Wiley & Sons, Inc. Hoboken, New Jersey, 2005, pp132-134,pp135-137, pp177-188,pp188-194.

[2] Ince, D. C., "Software metrics-an introduction,Software Metrics", *IEE Colloquium on*,, Jan 1990, pp. 1 – 2.

[3] Halstead, M. H. "Element of software science*", Elsevier north-holland*, Amsterdam, Holland, 1977.

[4] McCabe, T.j. "A complexity measure", *IEEE Transactions on software engineering*, v2, 1976, pp.308-320.

[5] Kafura, D. and Henry, S.,"Software quality metrics based on interconnectivity", *Journal of systems and software*, v2,1981, pp. 121-131.

[6] Albrecht, A.J., "Measuring application development productivity", *Proceedings of the joint SHARE/GUIDE symposium*, 1979, pp.83-92.

[7] A. Gunes Koru and Hongfang Liu, "Building Effective Defect-Prediction Models in Practice", *Software*,2005, v 22, No. 6, pp.23-29

[8] Taghi M. Khoshgoftaar, Naeem Seliya, "The necessity of assuring", *Quality in Software Measurement Data*, 2004.

[9] Shi Zhong, Taghi M. Khoshgoftaar, and Naeem Seliya. "Analyzing Software Measurement Data with Clustering Techniques", *IEEE Intelligent system*, 2004, pp. 20-27.

[10] Li, Kanglin, "Effective software test automation: developing an automated software testing tool", *SYBEX,2004* San Francisco, Calif. London, 2004, chapter 1, pp. 1-2.

[11] Gary D. Boetticher, "Nearest Neighbor Sampling for Better Defect Prediction", *Proceedings of the 2005 workshop on predictor models in software engineering*, St. Louis, Missouri, May,2005, pp.1-6.

[12] G. Batista, R. C. Prati, and M. C. Monard. "A study of the behavior of several methods for balancing machine learning training data", *SIGKDD Explor. Newsl.*, 2004, 6(1):20–29.

[13] Lei Tang, Huan Liu, "Bias Analysis in Text Classification for Highly Skewed Data", *Proceedings of the Fifth IEEE International Conference On Data Mining (ICDM'05)*, 27-30 Nov. 2005, pp.781-784.

[14] L.A. Zadeh, "Fuzzy sets", *Information and control*, 1965, v 8, pp. 338-353

[15] Vladimir Vapnik, Steven E. Golowich, Alex Smola, "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing", *Advances in Neural Information Processing Systems*, . Morgan Kaufmann Publishers, San Mateo, CA, 1997, v 9, pages 281--287.

[16] J. Ross Quinlan, *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.

[17] Ian H. Witten and Eibe Frank, *Data Mining: Practical machine learning tools and techniques, 2nd Edition*, Morgan Kaufmann, San Francisco, 2005.