Understanding Software Defect Prediction Models

Shrenuj Gandhi
Computer Science
North Carolina State University
sgandhi4@ncsu.edu

Neng Jiang
Computer Science
North Carolina State University
njiang@ncsu.edu

***Abstract*— Software defect prediction is one of the most assisting activities of the testing phase of software development lifecycle. It identifies modules that are defect prone and require extensive software testing. This way the testing resources can be used efficiently without violating the constraints.**

In this paper, we analyze a baseline study and replicate its results. We take a step back and look at the balance of data in our datasets. We also take a step further by diving into feature selection. The results from both these steps are interesting. Finally, we take this setup and apply it to different set of learners to gain insight into their performances.

***Index Terms*— Halstead metrics, McCabe metrics, naive Bayes, Information Gain, cross-validation, SMOTE, Logistic Regression**

I. INTRODUCTION

Software defect prediction strives to improve software quality and testing efficiency by constructing predictive classification models from code attributes to enable a timely identification of fault-prone modules.

Risk Analysis and Management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. A risk is a potential problemit might happen, it might not. But, regardless of the outcome, its a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan should the problem actually occur. Everyone involved in the software processmanagers, software engineers, and customersparticipate in risk analysis and management. To overcome risk, Software Defect Prediction can help to detect defects and identify potential defects using regression. Existing software defect prediction models that are optimized to predict explicitly the number of defects in a software module might fail to give an accurate order because it is very difficult to predict the exact number of defects in a software module due to noisy data. There are different fault prediction approaches available in the Software Engineering discipline. Such as fault prediction, security prediction, effort prediction, reusability prediction, test effort prediction and quality prediction. These approaches help us to minimize the cost of testing which minimize the cost of the project. Many research studies in a decade have focused on proposing new metrics to build prediction models. Widely studied metrics are Source Code and Process Metrics. Source Code metrics measure how source code is complex and the main rationale of the source code metrics is that source code with higher complexity can be more bug-prone. Process Metrics are extracted from software archives such as version control systems and issue tracking systems that manage all development histories. Process Metrics quantify many aspects of software development process such as changes of source code, ownership of source code files, developer interactions, etc. Usefulness of process metrics for defect prediction has been proved in many studies. Most defect prediction studies are conducted based on statistical approach, i.e. machine learning. Prediction models learned by machine learning algorithms can predict either bug-proneness of source code (classification) or the number of defects in source code (regression). Some research studies adopted recent machine learning.

```
M    = 10
N    = 10
All  = 38  # all the attributes
DATAS=(cm1 kc3 kc4 mw1 pc1 pc2 pc3 pc4) # data set list
FILTERS= (none logNums)                 # filter list
LEARNERS= (oneR j48 nb)                 # learner list

for data in DATAS
  for filter in FILTERS
      data' = filter(data)
      rank data' attributes via InfoGain # Equation 2
       for i = 1,2,3, All
          attribute' = the i-th highest ranked attributes
          data''     = select attributes' from data'
          repeat M times
              randomized order from data''
              generate N bins from data''
              for i in 1 to N
                tests        = bin[i]
                trainingData = data'' - tests
                for learner in LEARNERS
                  METHOD         = (filter attributes' learner)
                  predictor      = learner(trainingData)
                  RESULT[METHOD] = apply predictor to tests
```

Fig. 1. Data is filtered and the attributes are ranked using InfoGain. The data is then shuffled into a random order and divided into 10 bins. A learner is then applied to a training set built from nine of the bins. The learned predictor is tested on the remaining bin.

II. BASELINE STUDY

In the paper "Data Mining Static Code Attributes to Learn Defect Predictors", authors Tim Menzies, Jeremy Greenwald and Art Frank demonstrate that *naive Bayes* data miner with a *log filtering* preprocessor on the numeric data outperforms rule-based and decision-based learning methods.

For the evaluation method, quaritle charts are used. Recall and false alarm is measured

III. METHODOLOGY

As part of this project, we perform 4 experiments in order to understand the behavior of defect prediction models. evaluation measures

A. Reproducing Baseline Results

B. Performing SMOTE

The efficiency of Software Fault prediction models is greatly shaped by the class distribution of the training data24. Class distribution is described as the number of instances of each class in the training dataset. If the number of instances belonging to one class is much more than the number of instances belonging to another class, then the problem is known as class imbalance problem25. The class with more instances is called majority class and the one with lesser instances is called minority class. The problem widens when the class under consideration, i.e. the faulty class is represented by fewer instances. Various techniques have been proposed for addressing this problem and a few are discussed below.

C. Tuning Feature Selection

There are lots of different software metrics discovered and used for defect prediction in the literature. Instead of dealing with so many metrics, it would be practical and easy if we could determine the set of metrics that are most important and focus on them more to predict defectiveness. source [http://dl.acm.org/citation.cfm?id=2578408]

D. Comparing Learners

Each dataset goes through the following procedure

E. Data Cleaning

Any string column is ignored. Only numeric columns are processed. The idea is to keep it simple. String columns will

F. Pre-processing

The numeric data varies a lot. The following figure shows the min and max values of all the columns in the dataset -. Due to high variance, we are applying a filtering layer. We applying log filtering and then normalize the data.

G. Processing Method

The dataset is broken into testing and training sets. We are using 10×10 cross evaluation i.e. 10 times we train on 90% of data and test on the remaining 10%. Then SMOTE is applied to training set (keeping the size constant). Figure x shows the defect percentage in the 7 datasets. Clearly there is a minority of the defective class. Due to this we apply the Synthetic Minority Over-Sampling Technique (SMOTE). (source - http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/) This is done by replacement (resulting in inflated training set), or by SMOTE and then reconstructing original bin size by random selection// How SMOTING works - It works by creating synthetic samples from the minor class instead of creating copies. The algorithm selects two or more similar instances (using a distance measure) and perturbing an instance one attribute at a time by a random amount within the difference to the neighboring instances.(source -http://www.jair.org/papers/paper953.html)

OR examine results of SMOTE and try re-sampling i.e. duplicating data Side-effect of SMOTE-ing You will have more and different data, but the non-linear relationships between the attributes may not be preserved.

The training and testing datasets are fed to the learners after SMOTE. For evaluating the effect of SMOTE with the baseline result, we have used Naive Bayes classifier. Naive Bayes uses Bayes Theorem to model the conditional relationship of each attribute to the class variable.

Prior works(mention references) have also used different learning methods. To get a good grasp of the performance of different learners, we have also used, in this experiment, Support Vector Machine (SVM), Random Forest, Classification and Regression Trees(CART), and Logistic Regression(LR).

SVM method uses points in a transformed problem space that best separate classes into two groups. Classification for multiple classes is supported by a one-vs-all method. SVM also supports regression by modeling the function with a minimum amount of allowable error[MLM]. Random Forest is an extension of Bootstrap Aggregation or bagging that in addition to building trees based on multiple samples of your training data, it also constrains the features that can be used to build the trees, forcing trees to be different. CART are constructed from a dataset by making splits that best separate the data for the classes or predictions being made. Logistic regression fits a logistic model to data and makes predictions about the probability of an event (between 0 and 1). Different learners (source-http://machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/)

Decision Trees perform well on imbalanced datasets.The splitting rules that look at the class variable used in the creation of the trees, can force both classes to be addressed.try a few popular decision tree algorithms like C4.5, C5.0, CART, and Random Forest.

The above logic is wrapped around a feature selection process. This spits out the top K attributes (based on info gain). This process is tuned to select top k attributes that provide the most recall value. (provide reason do doing this)

IV. Software Science vs Data Dabbling

The fundamental of this paper was to reproduce the baseline study results. While most of data mining is done through hypothesis testing, limited data sets, and ...., it is a good idea to check these results. it is all the more useful to test these, if they were performed using different tools. Like in our case the baseline study was carried our in WEKA, whereas we use sklearn libraries in python.

Our results are not golden. We invite fellow researchers to test our approach and notify us by raising issues in our gitHub repository. The steps for all the above experiments, along with sample outputs

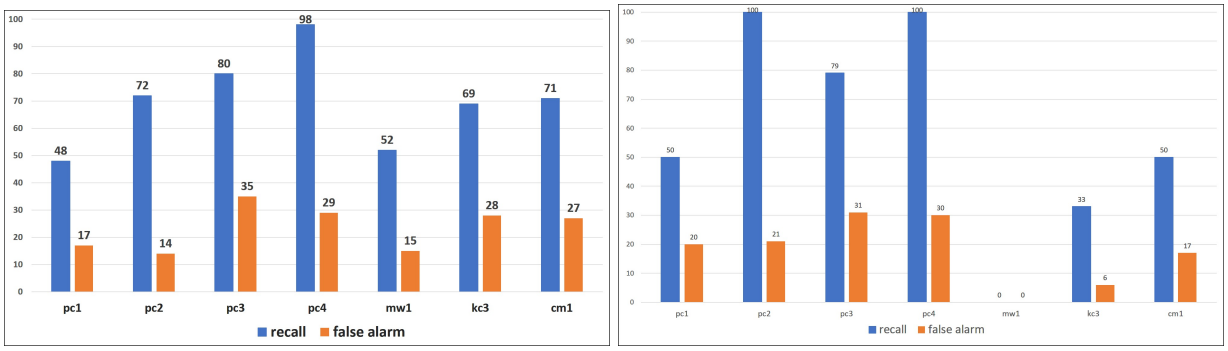

Fig. 2. The left hand side graph are the baseline results. It shows the average recall and average false alarm rate of Naive Bayes classifier with log filtering when top 3 features (based on InfoGain) are selected. The right hand side graph depicts the results we produced by following the steps in stated in Figure 1.

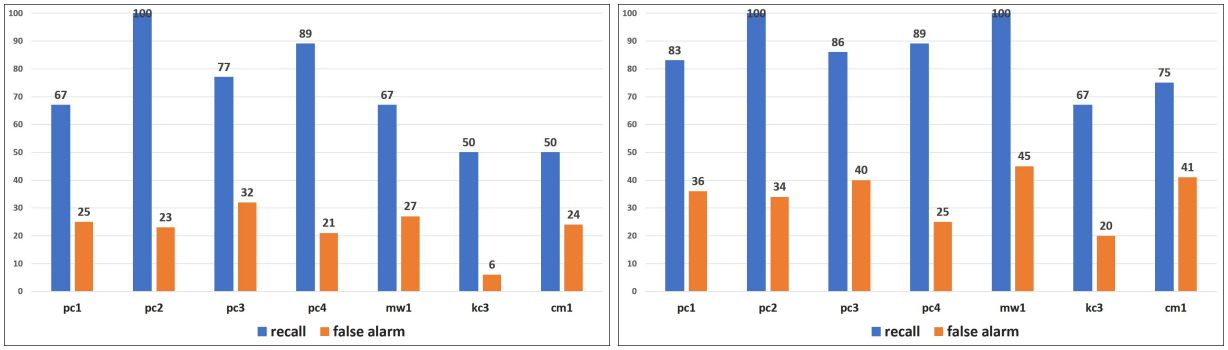

Fig. 3. The left hand side graph shows average recall and average precision values calculated by Naive Bayes after log filtering when top 5 features are selected based on InfoGain. The right hand side graph shows the average recall and average false alarm values when SMOTE is applied to the training set, while keeping the other procedure same.

and data sets is present on our repository. [source https://github.com/NeilBINGOHIT/fss16gNS/tree/master/project].

Unlike data dabblers, we provide a certification envelope that comments on the extent to which our approach is useful.

V. Conclusion

From these experiments we conclude that, software defect prediction is often faced with imbalanced class problem. A good solution is to apply SMOTE.

Defect prediction is data dependent. Thus if 3 attributes almost describe the distribution of class variable in one dataset, does not mean it will apply to different datasets. It is also interesting to note that not all attributes carry value. Via feature selection, we establish that only a small subset of attribute contribute to the answer of whether the module is defective or not. The size of the subset depends on the dataset.

VI. Future Work

Code the learners and tune their parameters USE alternatives of SMOTE- penalized models and re-sampling techniques

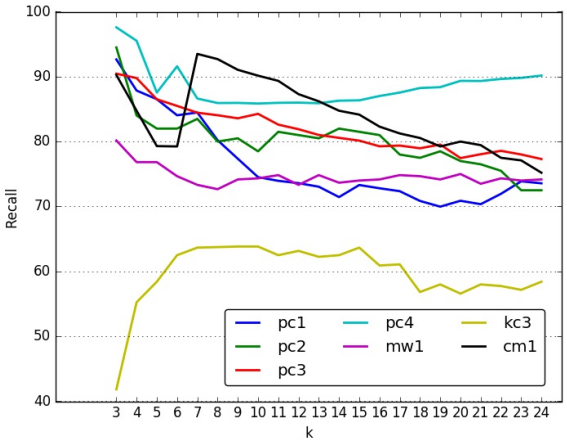

Fig. 4. The left hand side graph shows average recall and average precision values calculated by Naive Bayes after log filtering when top 5 features are selected based on InfoGain. The right hand side graph shows the average recall and average false alarm values when SMOTE is applied to the training set, while keeping the other procedure same.