Software Defect Prediction Modeling

Burak Turhan
Department of Computer Engineering, Bogazici University
turhanb@boun.edu.tr

Abstract

Defect predictors are helpful tools for project managers and developers. Accurate predictors may help reducing test times and guide developers for implementing higher quality codes. We propose a statistical defect predictor model with two major differences from the existing ones. Our model will use static code measures as input, since they can easily be collected with automated tools and prevent any kind of human subjectivity. There are two major differences between our proposed model and the existing ones. While existing defect predictors treat the software metrics equally, we claim that the effect of different metrics on prediction performance may vary. Thus, we propose a weighted prediction model where the weights are estimated according to the relative importance of metrics. Furthermore, our proposed model assumes that the software metrics are not statistically independent from each other and incorporates the relations between metrics into prediction process. Finally we propose a framework that can use existing set of software metrics and produce more accurate estimates of module complexities by using software system structure information.

1. Introduction

Quality of software is usually measured by the number of defects in the final product. Minimizing the number of defects requires a thorough testing of the software in question. On the other hand, testing phase requires approximately 50% of the whole project schedule [8, 24]. This means testing is the most expensive, time and resource consuming phase of the software development lifecycle. An effective test strategy should therefore consider minimizing the number of defects while using resources efficiently. Defect prediction models are helpful tools for software testing. Accurate estimates of defective modules may yield decreases in testing times and project managers may benefit from defect predictors in terms of allocating the limited resources effectively [23].

In this research, we will investigate two main assumptions of existing defect predictors and propose a novel model for enhancing the performance of defect prediction. These assumptions state that the inputs for the defect predictors are equally important and independent of each other. In reality these assumptions rarely hold [6, 10]. We will show evidence for these cases, propose our models and compare their performances on datasets collected from industry and open source projects. We will use public data repositories as well [2, 19]. Another contribution will be the construction of a framework that makes use of the existing software metrics and system architecture in order to make more accurate estimations of module complexities. We will base our empirical work on a well established experimental setup which will enable us to discuss the validity and generalization capabilities of our proposed model.

2. Related Work

Munson and Khoshgoftaar investigate linear regression models and discriminant analysis [16, 17]. They report that the performance of the discriminant analysis is better. They use Principal Component Analysis (PCA) as a pre-processing step in order to eliminate the co-linearity in software metrics. Nagappan et.al. also uses linear regression analysis with PCA for the STREW metric suite [18]. Decision tree learning is another common method that is preferred for its rule generation capabilities [12, 13]. Such rules are easier to explain to non-technical people [3].

Though some research stated against using static code measures [4, 22], a recent research showed that using a Naïve Bayes classifier with log-filtered static code measures yields significantly better results than rule based methods like decision trees [14]. The
number of researches for relaxing the assumptions of the method has significantly increased in recent years. These researches focused on modifications to break the conditional independence assumption and weighting attributes [5, 6, 10, 25, 26]. All studies reported results that are generally ‘not worse’ than the standard Naïve Bayes, while preserving the simplicity of the model.

As for the attributes that are used for constructing predictors, some researches prefer ranking the features for feature subset selection [14, 15], and there are also researches on using the ranking criteria for feature weight assignment [1, 25]. In fact, feature subset selection corresponds to ‘hard’ weighting of features, i.e. assigning 0 or 1 for feature weights. We should note that neural network based methods assign weights internally but does not have this information apriori [9, 20].

In this research, we aim to combine the best practices of the defect prediction and machine learning disciplines for constructing robust and accurate defect predictors. We will focus on the Naïve Bayes classifier and investigate the affects of relaxing its assumptions. These are the conditional independence of features and the equal importance of features assumptions. In order to overcome the first assumption, interactions between the features should be included in the model. For the second assumption, the features may be weighted according to their estimated relative importance. The feature weighting process includes constructing heuristics for accurate feature weight assignment.

Above mentioned researches treat the set of selected software metrics equally and assume their independence. On the other hand, we propose to treat each metric based on their estimated importance, include information about the interactions among metrics and search for empirical evidence for the validity of our approach. Incorporating weights will enable us to scale and compare metric values, and make sensitivity analysis of their effect on software defect prediction. Furthermore, we will propose a framework to include inter-module information for estimating module complexities, using the existing software metrics.

3. Research Objectives, Questions and Hypothesis

The goal of this research is to come up with a novel model for software defect prediction. Statistical models in machine learning have been used in other domains and specialized models are constructed to use domain related information, i.e. computer vision. We aim to construct a model that specializes for software domain. Furthermore, we plan to develop a tool for automated defect prediction, which can be used to help planning test strategies in order to reduce test costs. Another tool will also be implemented for guiding developers to increase the quality of their code that can pass the test of our proposed model. We will collect several metrics from different projects and plan to publish a public repository that other researchers can also use. Our proposed framework will use existing software metrics and include inter-module relations for estimating module complexities. Finally, by using weighted software metrics, we will be able to scale metric values and make sensitivity analysis of their effect on the number of defects for unit changes in their values.

**Research Question 1:** Existing models for defect prediction assume that all software metrics used in the predictor model have equal contribution to the prediction. Can the performance of defect prediction models increase by using more realistic approaches such as weighting software metrics?

Although software metrics are usually filtered by choosing a subset, the resulting metrics are treated equally throughout the existing models. Our hypothesis is that all software metrics do not contribute equally to the prediction of defects. Thus, we will assign weights to the software metrics and construct heuristics for this purpose. Then we will construct predictor models that use the metric weight information.

**Research Question 2:** Existing models for defect prediction assume that all software metrics used in the predictor model are independent. Can the performance of defect prediction models increase by means of incorporating the linear and non-linear relations/dependencies among software metrics?

Like in many problems, software metrics are assumed to be independent of each other; however this is not always the case. Modeling the relations between software metrics will increase the model complexities since this will require more parameters. Our hypothesis is that incorporating linear and non-linear metric relations in the model can increase the prediction results significantly, with the cost of more complex models.

**Research Question 3:** Widely used software metrics for estimating module complexities use McCabe, Halstead and LOC metrics [7, 11]. These evaluate each module independently. Can the performance of defect prediction models increase by means of using a framework (without introducing new metrics), that incorporates software system structure information (i.e. call graphs) to a widely used set of existing software metrics?
Metrics that count the number of calls to other modules are based on the fact that as the number of external calls of a module is relatively high, then the probability of encountering a defect is also higher due to possible defects in the external modules. An external module may be called once and another can be called many more times. Our hypothesis is that pure counting of the number of calls gives an idea of module complexity, but a better estimate would be based on the relative frequency of specific external calls and the module complexities of the called modules. This requires a recursive definition of complexity for which we will provide mathematical background in our proposed framework.

4. Empirical Study Design

An overall view of the planned research is given in Figure 1. The research will consist of the following steps:

i. Choosing the industry and open source data to use in the research: Public data sets provided by NASA MDP and PROMISE databases will be used in earlier stages [2, 19]. We are in the process of gathering metrics data from local companies. Once a trusted local dataset is established, we will include their projects in our model. For constructing the open source dataset, selected software from sourceforge.net will be used. The selection criteria are not clear at this stage.

ii. Extracting pre-defined set of software metrics from project source codes: Existing tools for metric collection may be used at this stage. For proposed software metrics, implementation of a new tool would be necessary. Corresponding defect information should also be collected together with the software metrics.

iii. Pre-processing Software Metrics: After this step, we focus on data mining principles. We will investigate the methods for feature subset selection, feature extraction, feature ranking and feature weighting methods in detail.

iv. Constructing models for prediction:
   - Constructing a Weighted Naive Bayes model treating the metrics according to their importance and constructing heuristics for metric weight assignment. These heuristics will be based on feature ranking schemes that are used in data mining research.
   - Constructing a Naive Bayes model considering the relations among metrics (i.e. removing the independence assumption). This will be achieved by using multivariate distributions for modeling data. Since most of the candidate metrics are size based, strong correlations are expected. Analysis of these correlations may result in reducing the candidate metric list. We plan to investigate both linear and non-linear correlations among these metrics.
   - Other methods to evaluate are to be decided in later phases of the research. An alternative is to use decision trees, since they are commonly used because of their interpretability.
   - Model combination by using methods for combining classifiers like building ensembles and voting.

v. An experimental setup for validation and generalization of the results: A valid experimental setup that is agreed on machine learning literature will be used for the validation of results. The experiments will include replication of stratified holdout studies and be clear and well defined for
ease of replication by other researchers. The proposed prediction models will use software metrics as the dependent variables and corresponding defect information as independent variables.

vi. **Evaluation of the prediction results and an analysis of software metrics**: Suitable performance metrics will be used for performance analysis. Besides accuracy of estimates, measures like probability of detection and probability of false alarm will be used [14]. The reason is that software defects are not balanced among modules. In general only a relatively small portion of the software modules are defective. Thus, performance metrics like accuracy can be misleading [14]. Furthermore, depending on the domain specific properties, hits/misses of defective modules can have different costs. This requires a joint optimization of performance metrics. Considering these issues, prediction performances of existing defect predictors will be compared with the proposed model using statistical tests for determining significance. Visual analysis of results will be done by using common methods including ROC curves and box plots.

5. Definition of Metrics

We plan to use only static code measures in this research. The reasons for this choice are given below:

- Static code measures are well defined.
- Static code measures can be collected from source code easily via automated tools on site or remotely.
- They allow data collection from open source projects; hence a larger dataset can be constructed considering the difficulties in collecting data from industry, sometimes due to confidentiality concerns.
- Automated static code measure collection avoids human interaction in the process. Thus human error and subjectivity are easily avoided.

A list of candidate metrics is given in Table 1. These metrics can be classified into 3 categories: McCabe metrics, Halstead metrics and line of code (LOC) metrics [7, 11]. The list is based on NASA Metric Data Program [19]. They are collected on a module basis and reflect the estimated complexity of modules. These modules are treated individually and the fact that these modules are communicating entities of a system is ignored. But it is also important to integrate the inter-module relations for better estimates of overall system quality. We plan to propose a framework to fine tune these metrics concerning the system structure. Our proposed framework will be based on a ranking scheme that is used for assigning page ranks in the internet and evaluating reputation in multi agent systems.

These candidate metrics will be used as the independent variables of the experiments. We will also collect defect information from the corresponding software modules (i.e. defect count, defect severity), which will be the dependent variables.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Metric Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclometric Complexity</td>
<td>Call Pairs</td>
</tr>
<tr>
<td>Cyclometric Density</td>
<td>Condition Count</td>
</tr>
<tr>
<td>Decision Density</td>
<td>Decision Count</td>
</tr>
<tr>
<td>Design Density</td>
<td>Edge Count</td>
</tr>
<tr>
<td>Essential Complexity</td>
<td>Formal Parameter Count</td>
</tr>
<tr>
<td>Essential Density</td>
<td>Modified Condition Count</td>
</tr>
<tr>
<td>Global Data Density</td>
<td>Multiple Condition count</td>
</tr>
<tr>
<td>Module Design Complexity</td>
<td>Node count</td>
</tr>
<tr>
<td>Maintenance Severity</td>
<td>Number of Lines</td>
</tr>
<tr>
<td>Normalized Cyclometric Complexity</td>
<td>Number of Operators</td>
</tr>
<tr>
<td>Pathological Complexity</td>
<td>Number of Unique Operators</td>
</tr>
<tr>
<td>Halstead Length</td>
<td>Number of Unique Operands</td>
</tr>
<tr>
<td>Halstead Volume</td>
<td>Number of Executable of Lines of Code</td>
</tr>
<tr>
<td>Halstead Level</td>
<td>Number of Lines of Comment</td>
</tr>
<tr>
<td>Halstead Difficulty</td>
<td>Number of Lines of Code Code and Comment</td>
</tr>
<tr>
<td>Halstead Intelligent Content</td>
<td>Number of Lines of Comment</td>
</tr>
<tr>
<td>Halstead Programming Effort</td>
<td>Percent of Code that is Comments</td>
</tr>
<tr>
<td>Halstead Error Estimate</td>
<td>Total Number of Blank Lines</td>
</tr>
<tr>
<td>Halstead Programming Time</td>
<td>Total Number of Lines of Code</td>
</tr>
<tr>
<td>Branch Count</td>
<td></td>
</tr>
</tbody>
</table>
6. Data Analysis Methods

6.1 Pre-processing Software Metrics:
In this step we will evaluate methods for feature subset selection, feature extraction, feature ranking and feature weighting.

Optimal subset selection requires an exhaustive search which is intractable. Forward selection, backward elimination and subset selection based on feature ranking are considered. In forward selection, one starts with an empty set of features and a feature is selected only if it increases the performance of the predictor, otherwise it is discarded. Backward elimination is similar and one starts with all features and a feature is removed if it does not affect the performance of the predictor. In feature ranking scheme, features are ordered and a similar procedure like forward selection is used. Using ranking is thought to be a better way than choosing features randomly.

In feature extraction, new features are formed by combining the existing ones. These new features can be interpreted as a set of hidden features. Principal Component Analysis (PCA) will be used as a feature extraction technique. PCA reveals the optimum linear relations within data. We plan to go one step further and try to reveal non-linear relations in software data if any exists. For this purpose we will use Isomap, which is a non-linear variant of multi dimensional scaling.

At this stage, we plan to use InfoGain and GainRatio methods for feature ranking step. Both are information theoretical measures, proposed for decision tree construction to determine the features that best split the data [21]. Finally, for feature weight assignment, we will construct heuristics based on the information gathered by feature ranking methods. We will also investigate methods for data transformation in order to fit our data into a suitable distribution that the prediction models use.

6.2 Models for Prediction:
Naïve Bayes will be in our main focus in earlier stages, since it is reported to produce the best results for defect prediction [14]. Naïve Bayes has two main assumptions: the features are conditionally independent and they have equal importance. The posterior probabilities can be estimated by using class conditional probabilities and modeling distributions for likelihood term in the famous Bayes rule. One advantage of Naïve Bayes is that it is a soft predictor, i.e. it returns posterior probability of a class, rather than hard classifiers like decision trees that make prediction of pure class memberships with no confidence information. By determining different likelihood distributions the independence assumption can be overcome. For instance, using a multivariate normal distribution rather than a univariate model is one option. In order to overcome the equal importance of features assumption, we will use a weighted version of the Naïve Bayes predictor. Other statistical models will be decided in later stages of the research.

7. Validity Threats and Control

Using only static code measures for our research avoids any kind of human error and subjectivity from the datasets. We will collect data both from industry and open source projects and use public software data repositories as well. By this way, we hope to cover a wide range of projects in terms of size and complexity. Most important threat to validity is the quality of the collected data. We must make sure that the collected defect information is tracked correctly in a disciplined manner. For this purpose, we will include NASA MDP datasets, which are used in previous research and accepted to reflect certain properties of software industry [14, 19]. We will also be in close contact with the local industry in order to ensure the validity of the data they provide.

The data mining experiment design will use standards accepted by machine learning community such as M by N fold cross validation and stratification of data. This assures that each data sample is used for both training and testing in different experiments, thus removes ordering effect and sampling bias for building models. In order to evaluate and compare the performances of different models, we will follow the notation of previous studies [14].

8. References


