International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 02 Issue: 02 | May-2015 www.irjet.net p-ISSN: 2395-0072

PREDICTION OF SOFTWARE DEFECT USING LINEAR TWIN CORE VECTOR MACHINE MODEL

Dr.P.Ganeshkumar, S.Kalaivani

[1] *Assistant Professor, Department of Information Technology, Anna University Regional center, Tamilnadu , India*
[2] *Student, Department of Information Technology, Anna University Regional center, Tamilnadu , India*

---***---

Abstract - Software defect prediction is the most prominent system in the program testing, in which a defect has to be predicted exactly in order to keep away from the most risk factors. The classification of error rates in the program defect prediction module need to be done carefully to keep away from the misclassification of error rates. In the existing work, two-stage cost sensitive learning system is used to foretell the program defects in which misclassification rate is reduced considerable by introducing the idea of thinking about the cost factor in feature choice stage and as well as in classification stage. However this lacks in performance due to its more time consumption to foretell the program defects and also the existing system cannot support the large scale of knowledge effectively as like tiny scale of knowledge. To overcome this issue, in this work, linear twin core vector machine idea is introduced. This proposed work is used to pick the optimal features and also it can select both linear type of knowledge and non linear type of knowledge. The experimental results show that the proposed methodology is better than the existing methodology.

Key Words
Application defect prediction, Cost sensitive learning, Feature choice, linear twin core vector machine

1. INTRODUCTION

A application bug is a mistake, flaw, failure, or fault in a computer program or technique that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most bugs arise from mistakes and errors made by people in either a program's source code or its design, or in frameworks and operating systems used by such programs, as well as a few are caused by compilers producing incorrect code. A program that contains a huge number of bugs, and/or bugs that seriously interfere with its functionality, is said to be buggy. Reports detailing bugs in a program are often known as bug

Reports, defect reports, fault reports, issue reports, trouble reports, change requests, and so forth.

Bugs trigger errors that can in turn have a wide selection of ripple effects, with varying levels of inconvenience to the user of the program. Some bugs have only a subtle effect on the program's functionality, and may thus lie undetected for a long time. More serious bugs may cause the program to crash or freeze. Others qualify as security bugs and might for example enable a malicious user to bypass access controls in order to receive unauthorized privileges.

How bugs get in to application

In application development projects, a "mistake" or "fault" can be introduced at any stage in the work of development. Bugs are a consequence of the nature of human factors in the programming task. They arise from oversights or mutual misunderstandings made by a application team in the work of specification, design, coding, information entry and documentation. For example, in making a comparatively simple program to sort a list of words in to alphabetical order, one's design might fail to think about what ought to happen when a word contains a hyphen. Perhaps, when converting the abstract design in to the selected programming language, might inadvertently generate an off-by-one error and fail to sort the last word in the list. Finally, when typing the resulting program in to the computer, might accidentally type a "<" where a ">" was intended, perhaps leading to the words being sorted in to reverse alphabetical order.

Another section of bug is called a race condition that can occur when programs have multiple parts executing simultaneously, either on the same method or across multiple systems interacting across a network. If the parts interact in a different order than the developers intended, it may break the logical flow of the program. These bugs can be difficult to detect or anticipate, since they may not occur in the work of every execution of a program.

Programming techniques

Bugs often generate inconsistencies in the internal knowledge of a jogging program. Programs can be written to check the consistency of their own internal knowledge while jogging. If an inconsistency is encountered, the program can immediately halt, so that the bug can be located & fixed. Alternatively, the program can basically

© 2015, IRJET.NET- All Rights Reserved

tell the user, try to correct the inconsistency, & continue jogging.

Development methodologies

There's several schemes for managing programmer activity, so that fewer bugs are produced. Plenty of of these fall under the discipline of program engineering (which addresses program design issues as well). For example, formal program specifications are used to state the exact behavior of programs, so that design bugs can be eliminated. Regrettably, formal specifications are impractical or impossible for anything but the shortest programs, because of issues of combinatorial explosion & indeterminacy.

Nowadays, popular approaches include automated unit testing & automated acceptance testing (sometimes going to the extreme of test-driven development), & agile program development (which is often combined with, or even in some cases mandates, automated testing). All of these approaches are supposed to catch bugs & poorly-specified requirements soon after they are introduced, which ought to make them simpler & cheaper to fix, & to catch at least a number of them before they enter in to production use.

Software Defect prediction

Identifying and locating defects in software projects is a difficult work. Especially, when project sizes grow, this task becomes expensive with sophisticated testing and evaluation mechanisms. On the other hand, measuring software in a continuous and disciplined manner brings many advantages such as accurate estimation of project costs and schedules, and improving product and process qualities. Detailed analysis of software metric data also gives significant clues about the locations of possible defects in a programming code.

According to a survey carried out by the Standish Group, an average software project exceeded its budget by 90 percent and its schedule by 222 percent (Chaos Chronicles, 1995). This survey took place in mid 90s and contained data from about 8-000 projects. These statistics show the importance of measuring the software early in its life cycle and taking the necessary precautions before these results come out. For the software projects carried out in the industry, an extensive metrics program is usually seen unnecessary and the practitioners start to stress on a metrics program when things are bad or when there is a need to satisfy some external assessment body.

On the academic side, less concentration is devoted on the decision support power of software measurement. The results of these measurements are usually evaluated with naive methods like regression and correlation between values. However models for assessing software risk in terms of predicting defects in a specific module or function have also been proposed in the previous research (Fenton and Neil, 1999). Some recent models also utilize machine-learning techniques for defect predicting (Neumann, 2002). But the main drawback of using machine learning in software defect prediction is the scarcity of data. Most of the companies do not share their software metric data with other organizations so that a useful database with great amount of data cannot be formed. However, there are publicly available well-established tools for extracting metrics such **as size, McCabe's cyclamate complexity, and Halstead's program vocabulary. These tools help** automating the data collection process in software projects.

A well established metrics program yields to better estimations of cost and schedule. Besides, the analyses of measured metrics are good indicators of possible defects in the software being developed. Testing is the most popular method for defect detection in most of the **software projects. However, when projects' sizes grow in** terms of both lines of code and effort spent, the task of testing gets more difficult and computationally expensive with the use Identifying and locating defects in program projects is a difficult work. , when project sizes grow, this task becomes pricey with sophisticated testing and evaluation mechanisms. On the other hand, measuring program in a continuous and disciplined manner brings lots of advantages such as correct estimation of project costs and schedules, and improving product and method qualities. Detailed analysis of program metric knowledge also gives significant clues about the locations of feasible defects in a programming code.

According to a survey carried out by the Standish Group, an average program project exceeded its budget by 90 percent and its schedule by 222 percent (Chaos Chronicles, 1995). This survey took place in mid 90s and contained knowledge from about 8-000 projects. These statistics show the importance of measuring the program early in its life cycle and taking the necessary precautions before these results come out. For the program projects carried out in the industry, an extensive metrics program is usually seen unnecessary and the practitioners start to stress on a metrics program when things are bad or when there is a necessity to satisfy some outside assessment body.

On the academic side, less concentration is devoted on the decision support power of program measurement. The results of these measurements are usually evaluated with naive methods like regression and correlation between values. However models for assessing program risk in terms of predicting defects in a specific module or function have also been proposed in the earlier research (Fenton and Neil, 1999). Some recent models also utilize machine-learning techniques for defect predicting (Neumann, 2002). But the main drawback of using machine learning

© 2015, IRJET.NET- All Rights Reserved

in program defect prediction is the shortage of knowledge. Most of the companies do not share their program metric knowledge with other organizations so that a useful database with large amount of knowledge cannot be formed. However, there are publicly available well-established tools for extracting metrics such as size, McCabeâs cyclamate complexity, and Halsteadâs program vocabulary. These tools help automating the knowledge collection method in program projects.

A well established metrics program yields to better estimations of cost and schedule. Besides, the analyses of measured metrics are nice indicators of feasible defects in the program being developed. Testing is the most popular technique for defect detection in most of the program projects. However, when projectsâ sizes grow in terms of both lines of code and hard work spent, the task of testing gets more difficult and computationally pricey with the use

of sophisticated testing & evaluation procedures. Nevertheless, defects that are identified in earlier segments of programs can be clustered according to their various properties & most importantly according to their severity. If the relationship between the program metrics measured at a sure state & the defectsâ properties can be formulated together, it becomes feasible to foretell similar defects in other parts of the code written.

The program metric information gives us the values for specific variables to measure a specific module/function or the whole program. When combined with the weighted error/defect information, this information set becomes the input for a machine learning process. A learning process is defined as a process that is said to learn from experience with respect to some class of tasks & performance measure, such that its performance at these tasks improves with experience (Mitchell, 1997). To design a learning process, the information set in this work is divided in to parts: the training information set & the testing information set. Some predictor functions are defined & trained with respect to Multi-Layer Perception & Decision Tree algorithms & the results are evaluated with the testing information set.

2. RELATED RESEARCH

2.1 Cost-sensitive boosting neural networks for SDP

Software defect predictors are tools to deal with this issue in a cost-effective way. Application defect predictors which classify the application modules in to defect-prone & not-defect-prone classes are effective tools to maintain the high quality of application products. In the work of the application defect prediction process, types of misclassification errors can be encountered. The type I misclassification happens when a not-fault-prone module is predicted as fault-prone while a sort II misclassification is that a fault prone module is classified as not-fault-prone. A sort I misclassification will lead to the waste of time & resources to review a non-faulty module. A sort II misclassification ends in the missed opportunity to correct a defective module that the faults may appear in the method testing or even in the field. The most often used measure is the misclassification rate which is defined as the ratio of the number of wrongly classified modules to the total number of modules.

2.2 Applying Novel Re-sampling Strategies to SDP

Accurate defect prediction is enormously important, because of the large economic impact of defective program. In program, a common rule of thumb is that 80% of the issues reside in only 20% of the modules. When they try to foretell the occurrence of faults in program where the giant majority of modules are fault-free, the classifier is often unable to detect the defective modules. This is a widely known issue in machine learning, often known as learning from imbalanced datasets. A knowledge set that is heavily skewed toward the majority class will sometimes generate classifiers that never predict the minority class. Due to the tremendous complexity & sophistication of program, improving program reliability is an enormously difficult task. Learning from imbalanced datasets is a current, active topic of research in the machine learning & knowledge mining communities. Imbalanced class distributions are a major issue in applying machine learning techniques to program defect prediction, & there has been tiny inquiry of stratification as a solution to this important issue.

2.3 Software Defect Association Mining and Defect Correction Effort Prediction

The success of a program process depends not only on cost and schedule, but also on quality. Among plenty of program quality characteristics, residual defects has become the de facto industry standard. The defect associations can be used for purposes: First, find as plenty of related defects as feasible to the detected defect(s) and, consequently, make more-effective corrections to the **program. Second, help evaluate reviewers' leads to the** coursework of an inspection. Third, to assist managers in improving the program process through analysis of the reasons some defects often occur together. For the defect association prediction, the length-first strategy permits us to find out as plenty of defects as feasible that coincide with known defect(s), thus stopping errors due to incomplete discovery of defect associations. For the defect correction hard work prediction, the length-first strategy permits us to receive more-accurate rules, thus improving the hard work prediction accuracy.

© 2015, IRJET.NET- All Rights Reserved

2.4 Assessing Predictors of Software Defects

When learning defect detectors from static code measures, NaiveBayes learners are better than entrophy-based decision-tree learners. Also, accuracy is not a useful way to evaluate those detectors. Further, those learners need no over 200-300 examples to learn adequate detectors, when the information has been heavily stratified; i.e. divided up in to sub-sub-sub systems (and by âadequateâ, they mean that those detectors perform as well slower, more pricey manual inspections). If defect detectors are fascinating, they must somehow be better than known baselines in the literature. For example, think about manual code reviews. These reviews are labour intensive. Various assessment measures exist for information miners including readability (neural networks cannot succinctly document their theories in a human readable form); repeatability (genetic algorithms can return different theories after different runs); to name a few. If the objective of learning is to generate models that have some useful future validity, then the learned theory ought to be tested on information not used to build it. In a commercial setting, accessing information is difficult. Information miners need to know how small information they need to accomplish lovely results

2.5 An Experimental Evaluation of an Experience-BasedCapture-Recapture Method

Inspections are accepted widely in the program engineering community as efficient contributors to improved program quality and reduced costs. The efficiency of inspections are seldom questioned anymore. However for a specific inspection there is a necessity for quantitative methods to analyses the result of an inspection. The knowledge usually obtainable after an inspection are the number of defects. If the knowledge on remaining defects was obtainable, it could be applied to control the development method, in order to utilize the development resources in the most cost-effective manner. On longer knowledge runs, the window is to be tuned, keeping it as short as feasible to stay sensitive to environmental and team composition factors versus keeping it long to keep away from statistical variations. The programs solve small knowledge structure, statistics and numerical issues. The defects are hence not seeded in the code, but are actual defects introduced in the coursework of program development. Functional defects as well as cosmetic ones, like misspelled comments, are counted as defects.

3. ARCHITECTURE

3.1 System Architecture

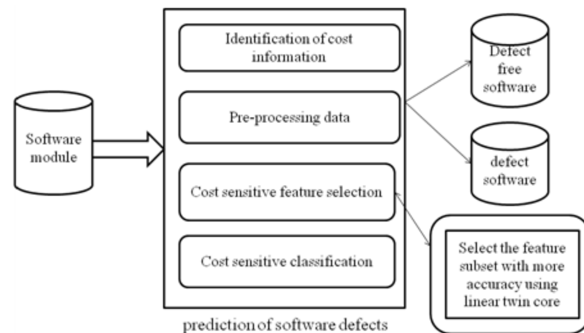

Figure 1

Initially program programs are enter in to system after that program defect prediction system will be processed in this cost information will be processed & then preprocessing will be processed in this system the program programs are separated as defective program & defect free program after this system feature choice will be completed in this system more accuracy is calculated by using linear twin core model

3.2 java platform

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, in lieu of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users often use a Java Runtime Surroundings (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets. Standardized libraries provide a generic way to access host-specific features such as graphics, threading, & networking.

A major benefit of using byte code is porting. However, the overhead of interpretation means that interpreted programs always run more slowly than programs compiled to native executables would. Just-in-Time compilers were introduced from an early stage that compiles byte codes to machine code in the coursework of runtime. Over the years, this JVM built-in feature has been optimized to a point where the JVM's performance competes with natively compiled C code

Object oriented

To be an Object Oriented language, any language must follow at least the four characteristics.

© 2015, IRJET.NET- All Rights Reserved

- Inheritance : It is the process of creating the new classes and using the behavior of the existing classes by extending them just to reuse the existing code and adding the additional features as needed.
- Encapsulation: It is the mechanism of combining the information and providing the abstraction.
- Polymorphism : As the name suggest one name multiple form, Polymorphism is the way of providing the different functionality by the functions having the same name based on the signatures of the methods.
- Dynamic binding: Sometimes we don't have the knowledge of objects about their specific types while writing our code. It is the way of providing the maximum functionality to a program about the specific type at runtime.

Multithreaded

As all of us know several features of Java like Secure, Robust, and Transportable, dynamic. Java is and a multithreaded programming language. Multithreading means a single program having different threads executing independently simultaneously. Multiple threads execute instructions according to the program code in a method or a program. Multithreading works the similar way as multiple processes. Multithreading programming is a fascinating idea in Java. In multithreaded programs not even a single thread disturbs the execution of other thread. Threads are obtained from the pool of obtainable prepared to run threads and they run on the method CPUs. This is how Multithreading works in Java which you will soon come to know in details in later chapters.

Interpreted

We all know that Java is an interpreted language as well. With an interpreted language such as Java, programs run directly from the source code. The interpreter program reads the source code & translates it on the fly in to computations. Thus, Java as an interpreted language depends on an interpreter program. The flexibility of being platform independent makes Java to outshine from other languages. The source code to be written & distributed is platform independent. Another advantage of Java as an interpreted language is its error debugging quality. Due to this any error occurring in the program gets traced. This is the way it is different to work with Java.

4. IMPLEMENTATION AND RESULTS

Software defect prediction can be done in the more correct manner without losing any defect knowledge. The optimized application prediction is done with the help of the optimized feature subset. These features are extracted by using the linear twin core support vector machine. & also optimized feature subset choice is done based on the f-score measure.

Software Defect Prediction

Based on the processes mentioned above, application defect is predicted & then the knowledge of that is used for the further processing. The application defect prediction method is described in the following algorithm.

4.1. Module Description

Cost information identification

In this module the cost information associated with the each misclassification of software modules are assigned through cost matrix.

There are three types of cost are classified. Those are

C_{OI} – Misclassifying the sample from the out group class as being from the in group class
C_{IO} – Misclassifying the sample from the in group class as being from the out group class
C_{II} – Misclassifying the sample from the one in group class as being from another in group class

Pre-processing Data

In this module, the whole historical knowledge will be divided in to the partitions namely training knowledge & check knowledge.

After that the pre-processing will be applied to both training knowledge & testing knowledge to convert in to the form that can be used in further stages

Cost Sensitive Classification

In this module classification will be done on test **data's based on optimal features that are selected from the** training data.

LSTVM based Feature Subset Selection

Feature Selection, also known as attribute selection, is one of the significant issues in the construction of classification model. Feature selection is used to reduce the number of input features and select relevant features for a classifier to improve its predictive performance. FS is responsible for obtaining relevant data for future analysis, as per problem formulation. Since there are lots of software metrics available in software dataset repository, so FS select significant feature which in turn will reduce the total project cost. F-score is one of the simple and significant feature selection techniques which

© 2015, IRJET.NET- All Rights Reserved

is mostly used in machine learning. It calculates the discrimination between two sets of real numbers. Let number of +ve and –ve samples are symbolized by 'm' and 'n' respectively and xk is any training vectors, then the F-score froth feature is evaluated as

$$F(i) = \frac{(\bar{x}_i^{(+)} - \bar{x}_i)^2 + (\bar{x}_i^{(-)} - \bar{x}_i)^2}{\frac{1}{m-1}\sum_{k=1}^{m}(x_{k,i}^{(+)} - \bar{x}_i^{(+)})^2 + \frac{1}{n-1}\sum_{k=1}^{n}(x_{k,i}^{(-)} - \bar{x}_i^{(-)})^2}$$

Where

$\bar{x}_i, \bar{x}_i^{(+)}, \bar{x}_i^{(-)}$ = the mean of the total ith features, mean of positive ith feature and mean of negative ith feature respectively.

$x_{k,i}^{(+)}, x_{k,i}^{(-)}$ = i^{th} feature of k-positive and k-negative samples correspondingly.

The larger value of F-score indicates that the corresponding feature is more discriminative or highly significant

Algorithm

Step1: Load the Software defect dataset from PROMISE repository.

Step2: Perform pre-processing of the dataset.

Step3: Divide the dataset using k-fold cross validation process.

Step4: Calculate the F-score for each feature and arrange them in descending order.

Step5: Generate new dataset with N features, where N=1… m, m is the total number of feature.

Step6: Train the model for each feature subset.

Step7: Compare the results with different feature subset and with other existing data mining approaches.

Step 8: Select the feature subset showing highest accuracy.

5. CONCLUSION

Software defect prediction is the most important method in the program development which need to be completed with most concentration. In this work, linear twin core expertise is implemented in order to deduct the defect fastly with more accuracy. This linear twin score methodology is based on the F-score feature choice technique which is used to pick significant feature which are helpful to foretell defects in program modules. There is a significant difference in classifier performance which is

Developed using new feature subset as compared to the classifier built on complete feature set. This research discloses the effectiveness of proposed feature selection based LSTSVM approach in predicting defective software modules and suggests that the proposed model can be useful in predicting software quality.

6. REFERENCES

1. B.W.Boehm and P.N.Papaccio (1988), Understanding and controlling software costs', IEEE Tran, Software eng., vol. 14, pp. 1462-1477.

2. J.Zheng (2010), 'Cost-sensitive boosting neural networks for software defect prediction', Expert Systems with Appl., vol.37, pp. 4537-4543.

3. L.C.Briand, K.E.Emam, et al (2000), 'A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content', IEEE Transactions on Software Engineering, Vol. 26, pp.518-540.

4. Lourdes Pelayo and Scott Dick (2007), 'Applying Novel Resembling Strategies to Software Defect Prediction,' in proc.North Amr.Fuzzy Inf. Processing Society, pp. 69 – 72.

5. Mingxia Liu, Linsong Miao et al (2014), 'Two-Stage Cost-Sensitive Learning forSoftware Defect Prediction', IEEE Trans. Reliability, Vol. 63, pp.679-684.

6. P. Runeson and C. Wohlin (1998), 'An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections', Empirical Software Engineering., vol. 3, pp.381–406.

7. Q. Song, Z. Jia et al (2011), 'A General Software Defect- Proneness Prediction Framework', IEEE Transactions On Software Engineering, Vol. 37, pp.356-370.

© 2015, IRJET.NET- All Rights Reserved