A systematic overview of software defect prediction using ML methods

Meer Tauseef Ali[1], Dr .Syed Abdul Sattar[2]

1. Research Scholar, (PP.COMP.SCI.O398) Dept. of Computer Science, Rayalaseema University, Kurnool, A.P.

2. Professor, Principal, Nawab Shah Alam Khan College of Engg. & Tech., Hyderabad.

Abstract- The quality of a software product is related to the number of defects and is limited by time and cost. Today software is an important factor in any environment. Predicting defects can reduce development time, reduce costs, reduce rework, increase customer satisfaction and create reliable software. Errors are inevitable, but try to minimize them. Therefore, practicing error prediction is important to achieve software quality and learn from past mistakes. This document reviews the literature of the last two decades and discusses recent progress in error forecasting.

I. Introduction

Project teams always try to produce a high quality software product with zero or few errors. The quality of a software product cannot be achieved by identifying errors and mistakes only in the test phase. Errors are automatically introduced at every stage of the software development life cycle. Software defects are errors, vulnerabilities, bugs, errors and malfunctions in computer programs and software that can lead to inaccurate or unexpected results.

High risk components in a software project should be stopped as soon as possible to improve the quality of the software. Therefore, error detection must be carried out at all stages, not just during testing.

While it is virtually impossible to eliminate every single error, it is possible to mitigate the extent of the defects and their negative impact on the project. Software defects constantly increase the cost and time required to complete a software product of the expected quality. In addition, defect identification and correction is one of the most expensive and expensive software processes.

Early detection of software errors can reduce development costs and rework effort and potentially provide more reliable software. Predicting software errors is the process of locating faulty modules in the software. To produce high quality software, the number of defects in the final product should be as low as possible. Many organizations want to predict the number of failures (errors) in their software systems before implementation in order to assess the quality and maintenance efforts that could be made. Therefore, fault prediction investigation is important to improve software quality.

2. Related verification work

In this section we have conducted a review of the literature on models and techniques for predicting software errors. Below is a brief overview of the work of many of the people working in the field of error prediction, as described above, with the aim of identifying future strategies in this area.

Norman Fenton et al. (1999) [1] describe a probabilistic model for the prediction of software errors. The aim is to design various forms of combination models, often accidental, based on the evidence available in software development, so that they can work more naturally and efficiently than before. Here is a critical review of a series of software metrics and statistical models, and the state of the art. The metrics are used in most models used to predict the extent and complexity of errors.

Others are based on test data, quality development processes and/or multivariate styles. A probabilistic graphical model (also known as the Bayesian Belief Network) has been used to authenticate this approach. To develop the stochastic model, we used the subjective judgment of an expert, an experienced project manager, to predict the error model. This was applied throughout the project life cycle.

The model can be used to evaluate ongoing projects and to consider the potential impact of many activities to improve software processes. If the costs are linked to process improvement and the benefits for the expected software quality improvements can be assessed, the model can be used to support informed decision making for SPI (software process improvement).

Ahmet Okutan, et. al. (2012) [2] proposed a new method to investigate the relationship between software metrics and error vulnerabilities using Bayesian networks. Using nine data sets from the Promise Data Repository, they show that RFC, LOC and LOCQ have higher error rates. We also proposed two additional metrics, NOD, which represents the number of developers, and LOCQ, which represents source code quality. Finally, we discussed the probability of marginal software error, its effective indicators and their relationships.

N. Fenton, et al. (2008) [3] used Bayesian networks to predict software reliability and defects. He used qualitative and quantitative measurements as well as random process factors and therefore took into account the limitations of traditional software. The use of dynamic discretization techniques leads to an improved model for predicting software errors.

Jie Xu, et al. (2010) [4] used various statistical and machine learning methods to vary the validity of models in order to predict software errors. A neurophysical approach has been used here. ISBSG data were used for the work.

Mohamad Mahdi Askari and Vahid Khatibi Bardsiri (2014) [5] used artificial neural networks to predict software errors in order to improve the algorithm's generalization capability. In addition to learning algorithms and evolutionary methods, vector machine techniques were used to support them. This has helped to maximize the scope of the classification and prevent overfitting problems. The algorithm was tested using 11 machine learning models from the NASA dataset. The conclusion was that it was more accurate and precise than the other models.

Karpagavadivu.K, et al. (2012) [6] analyzed the performance of the different methods used to predict software errors. They also described different algorithms and their applications. They found that the goal of predicting modules subject to errors using data mining is to improve the quality of the software development process. Using this technique, software managers can allocate resources effectively. The overall error rates of all techniques are compared and the benefits of all techniques are analyzed.

Yi Liu et al. (2010) [7] studied the problem of modeling software quality classification using the history of metric data sets derived from a single software project. Classification models derived from a single dataset are generally insufficient to generate powerful and accurate models.

To solve this problem, we conducted software quality classification modeling using multiple datasets from different software projects.

Previous studies have shown that the use of multiple data sets for validation can lead to robust models based on genetic programming. In this paper, we examine in detail the effectiveness of multi-set modeling. In addition, we propose a new general classifier consisting of training, multi-set validation and tuning phases. The data set used in the experiments comes from a NASA software project. The performance of the proposed classifier is compared with the results of 17 other data mining techniques. The results of the comparison show that the proposed approach is more effective and accurate when multiple datasets are used.

Tom M. Mitchell (1997) [8] has attempted to classify data using the Naïve Bayesian algorithm. Naïve Bayesian is one of the most popular learning algorithms in data mining and machine learning. It is popular because it is an efficient and effective inductive learning algorithm. The classification based on the Naïve Bayesian algorithm provides very competitive performance through the assumption of conditional independence.

Others are based on test data, quality development processes and/or multivariate styles. A probabilistic graphical model (also known as the Bayesian Belief Network) has been used to authenticate this approach. To develop the stochastic model, we used the subjective judgment of an expert, an experienced project manager, to predict the error model. This was applied throughout the project life cycle.

The model can be used to evaluate ongoing projects and to consider the potential impact of many activities to improve software processes. If the costs are linked to process improvement and the benefits for the expected software quality improvements can be assessed, the model can be used to support informed decision making for SPI (software process improvement).

3. Software defect prediction techniques based on ML

Various data mining techniques can be applied to different areas of software engineering to improve the effectiveness and quality of software development and to predict software errors. the most common SDP techniques are data mining techniques, and Figure 1.1 shows machine learning Methods.

Machine learning is the study and construction of algorithms capable of learning from sample data sets and making data-driven predictions and decisions about new data. It is similar to data mining by developing computer programs that change and learn when exposed to new data. Both systems scan data and search for models. But while data mining extracts data for human understanding, machine learning uses this data to identify patterns in the data and adjust program behavior accordingly.

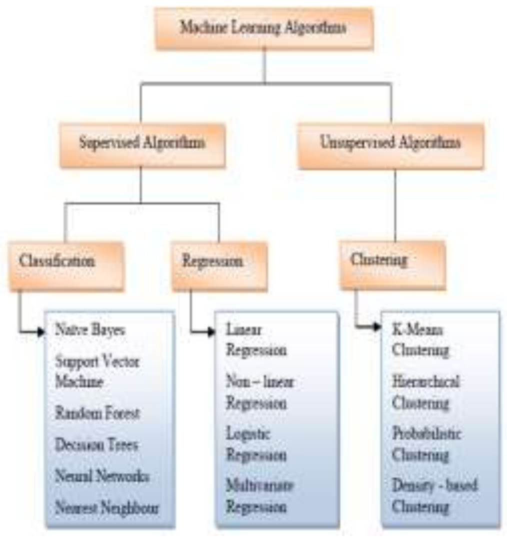

Figure 1.1: ML algorithm overview

Learning the machine is always based on observations, data, direct experience and instructions. In general, machine learning is about learning from past experiences and making them better in the future. The objective is to develop

learning algorithms that learn automatically, without the need for human intervention or help. Machine learning is the heart of artificial intelligence.

It is highly unlikely that you can build an intelligent system capable of performing complex tasks such as speech and vision without using learning to get there. These challenges are simply too difficult to solve otherwise. Moreover, without the ability to learn, we do not think of systems as truly intelligent systems; although it is a sub-discipline of AI, machine learning also overlaps heavily with other domains, especially statistics, but also mathematics, physics and theoretical computing.

Machine learning methods are currently used to solve problems common to all industrial sectors; the primary objective of machine learning research is to develop algorithms that are generally of practical value and efficiency. In the context of learning, we must consider the amount of data required by the learning algorithm, as well as its temporal and spatial efficiency. Learning algorithms must serve a general purpose to solve problems that can be easily applied to a large class of learning problems, such as those described above. The most important thing is that the learning outcome should be an accurate prediction rule for making predictions about new data. Sometimes we may be interested in the interpretability of forecasting rules created by learning.

The main advantage of machine learning over static programming is that machine learning algorithms are data driven and can examine large amounts of data, so machine learning results are often more accurate than static programming results. On the other hand, human experts writing static programs are more likely to be led to inaccurate impressions or to examine a relatively small number of examples or data. Figure 1 shows a common process that occurs in a typical machine learning model.

Another reason to look at machine learning is that we expect it to give us an overview of the general phenomenon of learning. Some of the details we might learn are the intrinsic nature of a given learning problem, which allows us to know in advance whether the problem is difficult or easy to solve and what we will learn to learn effectively.

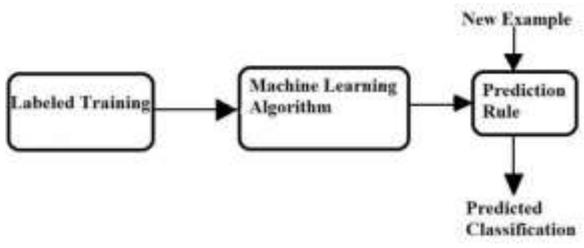

Figure1.2 Outline of the general machine learning process.

In this study, we are interested in designing the machine learning, but we want to analyses it mathematically to understand its efficiency. With the help of theory, we hope to understand the inherent difficulties of a particular learning problem and to be able to explain the phenomena observed in real-world experiments with a learning algorithm.

3.1 Overview of machine learning

Machine learning is essentially the transformation of data into information. The knowledge and insights you seek from the raw data cannot be acquired simply by looking at the data. Machine learning methods also use statistics, which can be used to interpret the data and then act on the facts learned to solve the problems that need to be solved when determining a new data set. Static programs are usually used to solve deterministic problems with a certain method of solution, but for problems that are not deterministic and for which there is insufficient data available on the problem, an approach called machine learning is used.

Initially, the lack of data sets to train the algorithms made it difficult to make realistic decisions using machine learning. However, with the renaissance of sensors and the availability of Internet access, the real challenge now is to efficiently order the abundant free data and use it to train machine learning algorithms. Current trends in developments such as mobile computing and the Internet of Things will lead to increasingly useful data generation in the future. Much of our business depends on data and we cannot afford to lose ourselves in it. Therefore, machine learning methods can help us to penetrate this data and extract important information from it.

The machine learning algorithm learns certain relationships between the characteristics and the target variable and tries to predict the target variable for new data. In this example, the target variable is just a value, i.e., every time a new result is entered into the machine, the machine measures the characteristic and predicts a value, and this new measured value is called a test set.

The precision of an algorithm can be calculated by comparing the actual values with the target variables provided by the algorithm. The process of observation of data learned from the algorithm is called knowledge representation, which can be a set of rules, a probability distribution function, or an example from a training set.

3.2 Types of automatic learning algorithms

Machine learning algorithms can model each problem differently based on input data. So, before looking at the algorithms, let's take a quick look at the different learning styles commonly used. By organizing the machine learning algorithms in this way, we can choose the right algorithm to tackle a given problem based on the available input data sets and the modeling process to achieve efficient results. Machine learning algorithms can be divided into three groups according to their learning styles.

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised learning

Supervised learning occurs when the algorithm learns from input data (also known as training data). This input data has a known target response or label and can be a number or a string of numbers or a character string. The training or learning process creates a pattern that predicts the correct response when a new example is given. The monitoring approach is further divided into two areas. Classification and regression. In classification, the algorithm predicts the class to which a given test corresponds and in regression it predicts the value of the variable of interest.

Unsupervised learning

Unsupervised learning occurs when the algorithm learns from input data without labels and has no unique result, so that the algorithm can determine the data pattern independently. It learns the characteristics present in the input data to extract the general rules and create a model. This is done through a mathematical process to reduce redundancy and organize the data by similarity.

Unsupervised learning is used in two main forms: clustering, which groups similar elements, and density estimation, which finds a statistic representing a date. Recommendations are based on the estimation of the groups of clients to whom one is most similar and the derivation of one's preferences from those groups.

Reinforcement learning

Learning amplification allows the machine to automatically determine its behaviour in a given context to maximize its performance. To learn its behaviour, known as the reinforcement signal, simple reward feedback is required. This learning occurs when an unlabeled example is presented to the algorithm, as in unsupervised learning.

However, positive or negative feedback on the solution proposed by the algorithm may be accompanied by an example. Reinforcement learning, unlike unsupervised learning, refers to applications where the algorithm has to make decisions that have consequences in the real world. It is considered as learning by trial and error.

3.3 Steps in the development of machine learning algorithms

In this report, there are six general steps in the implementation of a machine learning algorithm, which are briefly described below.

1**. Collect data.** Collecting data is a tedious task and can be done by reading some websites and extracting from them, obtaining information from RSS feeds, collecting readings from any sensor or anything on the internet. To simplify the process, we have used publicly available data in this document.

2. **Prepare the input data**. Once the data is available, converting it into a format that the algorithm will accept will help the different algorithms to use the same set of information. However, the specific formats of the algorithms are usually trivial compared to data collection.

3. **Analyze the data entered**. Review the data from the previous task to ensure that the text file created in steps 1 and 2 is a valid job data that meets expectations. You can also search for recognizable templates and display them in two or three dimensions for further analysis. If there are multiple features in the data, these are reduced and presented to three key features.

4. **Algorithm formation**: this step is sometimes defined as the learning process. The combination of training and testing of an algorithm is the "heart" of any learning process of the machine. Valid analytical data, called training sets, are fed to the algorithm to extract knowledge and information. This knowledge is often stored in a format that can be easily used by the machine for the next two stages. In the case of For unsupervised learning there are no learning steps because there are no target values. Everything is used in the next phase.

5. **Test the algorithm**. Here we use the information obtained during the training process. When evaluating an algorithm, you need to test it to determine its accuracy. In the case of supervised learning, the target variable of the test data used to evaluate the algorithm is known. In the case of unsupervised learning, you may need to use several other metrics to assess success. In both cases, if the efficiency is not satisfactory, we must go back to step 4 and repeat the learning process with different and more precise training data to retest the algorithm.

6**. Use.** At this stage, algorithms are used to make decisions and foresee solutions. If you are not satisfied with the accuracy, retrace the process from the first step and use more data, as learning the machine is a continuous development process.

These six steps are great for creating algorithms for machine learning, but for each algorithm based on the problem to be solved you need to insert small changes or some steps between them to facilitate the data as needed. As already mentioned, this document focuses mainly on machine learning classification algorithms. The next part of the document clearly describes the general process of creating a supervised learning model.

3.4 Supervised learning

In this section, we examine a detailed flow chart of supervised learning for both classification and regression (see Figure 1.2). You do not want to have missing data in a random sample. So, if the data is less scarce, you can remove samples with missing values from the dataset or replace them with some statistics instead of removing the missing values. The third stage of sampling is the process of randomly splitting the dataset into a training dataset and a test dataset. The training dataset is used to train the algorithm and the test dataset is used to evaluate the efficiency of the algorithm at the end. The next process is called cross-validation and is used to evaluate different combinations of feature selection, dimensionality reduction and training algorithms.

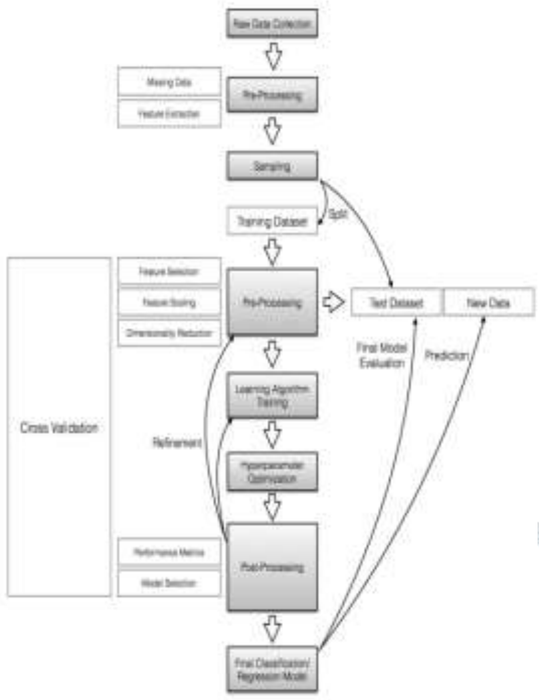

Figure 1.3: Diagram for Supervised Learning Process Workflow

The most common is the k-fold cross-validation, which divides the training data set into subassemblies k (for training the subassembly k-1 is used and for testing 1 subassembly is used). This partitioning helps to calculate the average error rate on completion of training. Normalization is performed to give each characteristic of the data record the same meaning, because each characteristic may have a different range of values during training and decision making. There are many types of learning algorithms described in this document, but we will talk about k-Nearest Neighbor.

In the following sections, Decision Trees and Naive Bayes. In post-processing, the accuracy of the algorithm can be evaluated against the test data, and if the accuracy does not meet the expected processing, the process can be restarted at any time providing the algorithm with more accurate and richer data. In addition, by perfecting the techniques for collecting and preparing input data sets, we can achieve better results. Once the algorithm has reached the expected accuracy, it can be used to predict the actual data.

The classification is part of the supervised learning model, where the algorithm predicts the class to which the new data corresponds, but the class does not have to be a number. In this article we will examine three important classification algorithms.

- K-nearest.
- Decision tree
- Naive baize
- Logistic regression
- baggage

3.6. K-nearest

Compared to other machine learning algorithms, it is easier to understand and implement. This section starts with a description of the basic operational concepts of the algorithm, followed by a flowchart explaining the procedure step by step.

The k-Nearest Neighbor (kNN) algorithm is one of the most widely used classification algorithms because of its simplicity and ease of implementation. It is also used as a basic classifier in many domain problems. The kNN algorithm is a traditional non-parametric classifier, usually used for classification and regression problems. We start with a data set with a data point and a known class and divide them into two subgroups, called training data and test data.

The process of forecasting a new class of data based on the class of available training data is called the classification problem. kNN is a type of delayed learning because kNN does not need to train the algorithm and the classification phase goes through all the training data and calculates the distance to the input data and predicts the class of input data.

Once the distance between the input data and the training data is measured with the above equation, the k number of points closest to the input data is selected and the majority class of the selected neighbours is the predictive class of the new data. So, it is called K-nearest neighbours. The Euclidean distance calculation applies to categorical and numerical data sets, but not to mixed data sets.

3.6.2 Flow chart

In the k-Nearest Neighbors (kNN) algorithm we have a training dataset and a test dataset. Since we only talk about label-based classification here, each instance of the training data will have different characteristics and labels. Thus each instance of data knows the corresponding labels. The general objective in developing this algorithm is to identify the labels of a new data without having its own labels, and the overall process involved is shown in the workflow diagram in Figure 1.3.This is the simple process behind the most widely used and powerful classifier in machine learning models.

3.7. Decision tree

This section describes the following decision tree classification algorithms. This is one of the most common methods in machine learning.

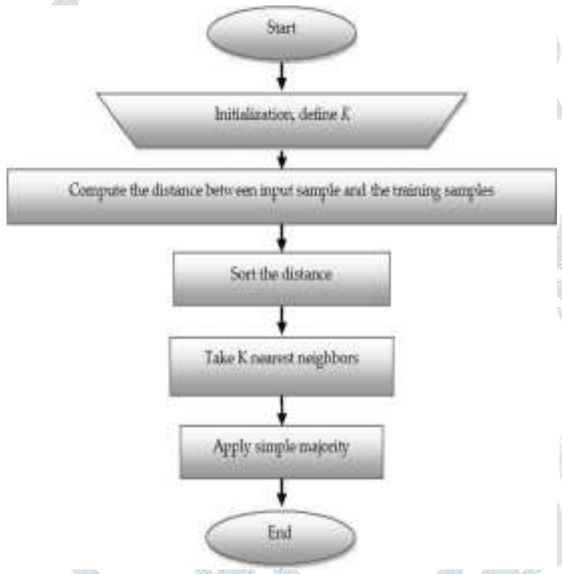

Figure 1.4: Flow diagram of the k-nearest-neighborhood algorithm.

Decision trees are still suitable for missing values and can handle irrelevant characteristics. The main disadvantage is the tendency to over-adaptation. The algorithm also works with numbers and nominal values, such as kNN.

3.7.1. Context

A decision tree is a representation of the design process to determine the class of a particular feature. Each node in the tree can be either a leaf node (or response node) containing the name of a class, or a non-leaf node (decision node) containing an attribute test with a different branch to a decision tree for each possible value of an attribute.

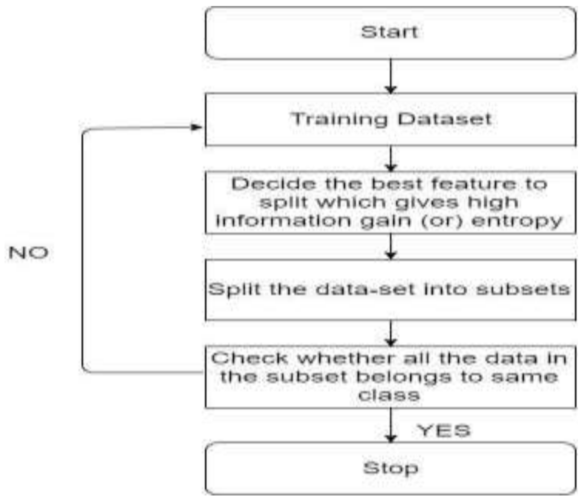
Start
Training Dataset
Decide the best feature to split which gives high information gain (or) entropy
Split the data-set into subsets
Check whether all the data in the subset belongs to same class
YES
NO
Stop

Figure1.5: Decision tree workflow diagram

In general, in the decision tree representation, leaf nodes are in the form of an ellipse, and decision nodes are rectangles with arrows indicating the connections between the appropriate node strategies, and the strategy behind this implemented decision tree considers the subdivision of the data set based on attributes and the appropriate place to end the subdivision. These processes are described below.

The higher the entropy, the more mixed the data set is. Therefore, the difference in entropy before and after the split is called information gain, and after all the algorithms in the decision tree for the classification of data instances belonging to the same class have been performed on the same leaf nodes, the split with the highest information gain is considered the characteristic to be split.

The decision tree algorithm is part of supervised learning methods and is the most commonly used method. The determination of the optimal characteristics to be partitioned and the actual partitioning process is performed using the ID3 algorithm, and the process of calculating the information in the data set is defined as Shannon entropy or entropy. The following steps are derived from the workflow diagram.

Check if all the data in the subset belong to the same class. If so, stop the splitting process. If a different class is available, start the process again, select the characteristics into which you want to split the data, and then further split the data set, which will result in a different subdivision. Therefore, this process continues until each end node of the tree has elements belonging to the same class.

3.8 Naïve Bayes

In this section we describe the following classification algorithms from the two classification algorithms before Naïve Bayes; we estimated a class for the input data and calculated its error rate. However, in the case of naive Bayes, we estimate the best class and assign a probability to this best estimate. This section starts with a basic description of the algorithm,

3.8.1. Context

Naïve Bayes is a simple form of Bayesian classifier based on Bayesian decision theory. The Bayes theorem plays an important role in the classification process. Bayesian classifier assigns the most likely class to a given instance. The naive Bayes classifier assumes that the effect of an attribute on a class is statistically independent of all other attributes. This assumption is considered naive. Despite this assumption, its accuracy is comparable to other advanced classifiers and has been demonstrated in many practical applications. Naïve Bayes classifiers are increasingly popular and have gained wide acceptance due to their computational efficiency, simplicity and performance on real-world problems.

Before examining the example and its implementation, we discuss the mathematical foundations of Bayesian decision theory. That the probability that a data element belongs to class 1 is p1(x, y) and the probability that the same data element belongs to class 2 is p2(x, y). According to Bayesian decision theory, we select a class with a higher probability by following two rules.

$$\text{If } p1(x, y) > p2(x, y), \text{ class } 1$$

For p1(x, y) < p2(x, y), class 2

Bayesian rules are used to manipulate conditional probabilities, and the mathematical calculations behind the exchange of symbols within a conditional probability are as follows.

By combining conditional probability and Bayesian law from the above equation, we can rewrite the Bayesian classification law.

If p (c1 | x, y) > p (c2 | x, y), class 1

For p (c1 | x, y) < p (c2 | x, y), class 2

But since we don't have a value for p (we | x, y), applying the Bayesian rule we rewrite it as follows

3.3.9. Logistic regression for machine learning

Logistic regression is another technique borrowed from machine learning in the field of statistics. It is a "go-to" method for binary classification problems (problems with two class values).

- Many names and terms used to describe logistic regression
- Representations used in logistic regression models.
- A technique used to learn the coefficients of a logistic regression model from data.
- How to actually make forecasts in a trained logistic regression model
- Where you can get information when you want to dig a little deeper.

No background in linear or statistical algebra is required. Logistic regression, like linear regression, uses an expression as representation. Input values (x) are combined linearly using weights or coefficient values (called Greek capitalization betas) to predict output values (y). An important difference to linear regression is that the modeled output values are binary (0 or 1) rather than numerical.

Below is an example of a logistic regression equation.

$$y = e^n (b0 + b1*x) / (1 + e^n (b0 + b1*x)$$

Where y is the expected output, b0 is the bias or interception term, and b1 is the coefficient of the individual input values (x). Each column of input data is assigned a b coefficient (the real number of constants) which must be learned from the training data. The actual representation of the model to be stored in memory or on file consists of the coefficients (beta or b values) of the equation.

3.10 bagging machine

A bagging classifier is an ensemble meta stimulator that adjusts the base classifier to a random subset of the original data set and aggregates their individual forecasts (by collation or average) to form a final forecast. Such a meta-stimulator can be used as a way to reduce the variance of a black box estimator (e.g., a decision tree), typically by introducing randomization into its construction process to make a set.

Each basic classifier is trained in parallel with the generated training set by randomly drawing N examples (or data) for assignment from the original training data set (where N is the size of the original training set). Each base classifier training set is independent of the other. Much of the original data can be repeated in the resulting training sets, while other data can be omitted. Bagging reduces over-matching (variance) through averaging and tuning, which leads to increased distortion, but is offset by a reduction in variance.

How do you pack training records?

How does enveloping with a fictitious training data set work? Bagging replaces and resamples the original training records, so some cases (or data) may exist more than once and others may be omitted.

4. Results and discussion

Spending more time on defective modules and less time on non-defective modules allows better use of project resources and makes the maintenance part of the project easier for both the client and the project owner. Predicting defects can increase the probability of a module or file being re-tested by the event team, thus increasing the probability of failure.

I tend to examine publications related to the prediction of defects. Early research shows that much has been done with the characteristics of static code. There have been false and misleading assumptions about the representation and observation of defects. Your claims will be better understood if you know that some defects are described as observed defects and others are defined as residual defects.

Research on software error prediction has been criticized and several studies in the literature argue that the problem of error prediction depends on the solution However, in addition to the impact of static code metrics on error prediction, it was later realized that various measures such as process metrics are also valid and can be considered as examples, and argues that static code measures alone are not sufficient to accurately predict software defects.

In support of this concept, it is argued that if there is a defect in the software, it can be linked to one of all of the following

I don't think the developers are competent enough for this project. Project management has its disadvantages, and software lifecycle methods may not be followed correctly.

As the software has not been fully tested, defects may not be corrected during this period. It may have a poor style, may not cover all requirements, or may not reflect some requirements correctly. It is possible that the project specifications are wrong.

Each of these uncertain factors influences the detection and correction of errors at all stages of the development lifecycle, from initial requirements to product delivery. None of the above factors are related to code metrics and all factors can be affected by errors. So the question is whether factors and metrics influence vulnerability and how to measure their impact. Software efforts tend to pay attention to three fundamental issues

1. Time estimate from the point of view of system reliability

2. Verify the impact of the design and testing process on the number of defects and their density.

3. Prediction of errors that is abundant in the software.

5. Conclusion

Some use the characteristics of the working product, while others only require error data. All methods have their advantages and disadvantages, which depend on the quality of the input used for the forecast. Defect prediction techniques vary depending on the type of data required; some require only a small amount of data, while others require more data.

In exploring the strengths and weaknesses of any method of error detection, these factors must be selected at a certain level to guide the evaluation of empirical evidence. A problem arises in the choice of error detection methods. The choice of fault detection methods depends on factors such as artifacts, the types of faults they contain, who makes the detection and how, for what purpose and for what activities. The factors also include the criteria used to conduct the assessment. These factors show that many variations must be taken into account.

With the help of error forecasting technology, the quality and reliability of the software can be improved. Error prediction also leads to high quality software, cost savings, reduced maintenance and increased customer satisfaction. Fault prediction technology is very useful for producing quality software. Future research should focus on classifiers specialized in data pre-processing and defect prediction of software modules.

References:

[1] Norman Fenton, Paul Krause and Martin Neil, (1999), "A Probabilistic Model for Software Defect Prediction", For submission to IEEE Transactions in Software Engineering.

[2] Ahmet Okutan, Olcay Taner Yildiz,(2012) "Software defect prediction using Bayesian networks", Empirical Software Eng (2014) 19:154–181 © Springer Science+Business Media, LLC.

[3] N. Fenton and M. Neil (2008) "Using Bayesian networks to predict software defects and reliability", Proc. IMechE Vol. 222 Part O: J. Risk and Reliability, pp 702-7.

[4] Jie Xu, [2]Danny Ho and [1]Luiz Fernando Capretz, "An Empirical Study on the Procedure Drive Software Quality Estimation Models", International journal of computer science & information Technology (IJCSIT) Vol.2, No.4, (2010).

[5] Mohamad Mahdi Askari and Vahid Khatibi Bardsiri (2014), "Software Defect Prediction using a High Performance Neural Network", International Journal of Software Engineering and Its Applications Vol. 8, No. 12 (2014), pp. 177-188.

[6] Karpagavadivu.K, et.al. (2012), "A Review of Different Software Fault Prediction Using Data Mining Techniques Methods", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 1, Issue 8, pp 1-3.

[7]Yi Liu et al., (2010) investigated the problem of software quality classification modeling

[8]Tom M. Mitchell (1997) attempted to classify data using Naïve Bayesian algorithm

[9]Yi Liu et al., (2010) investigated the problem of software quality classification modeling

[10] Kim et al., (2008) proposed change classification method for predicting dormant software bugs. Change classification was based on machine learning classifiers,

1.

Meer Tauseef Ali,
Research Scholar,
(PP.COMP.SCI.O398)
Dept. of Computer Science,
Rayalaseema University,
Kurnool, A.P.

2.

Dr .Syed Abdul Sattar,
Professor, Principal,
Nawab Shah Alam Khan
College of Engg. & Tech.,
Hyderabad.