Advances in Engineering Software 42 (2011) 535–546

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Evaluating the change of software fault behavior with dataset attributes based on categorical correlation

Izzat Alsmadi [a,*], Hassan Najadat [b]

[a] Yarmouk University, Faculty of CS and IT, Jordan
[b] Jordan University of Science and Technology, Computer and IT Faculty, Jordan

ARTICLE INFO

Article history:
Received 23 January 2011
Received in revised form 7 March 2011
Accepted 14 March 2011
Available online 19 May 2011

Keywords:
Correlation
Data mining
Software quality
Software mining
Fault prone modules
Prediction algorithms
Clustering

ABSTRACT

Utilization of data mining in software engineering has been the subject of several research papers. Majority of subjects of those paper were in making use of historical data for decision making activities such as cost estimation and product or project attributes prediction and estimation. The ability to predict software fault modules and the ability to correlate relations between faulty modules and product attributes using statistics is the subject of this paper. Correlations and relations between the attributes and the categorical variable or the class are studied through generating a pool of records from each dataset and then select two samples every time from the dataset and compare them. The correlation between the two selected records is studied in terms of changing from faulty to non-faulty or the opposite for the module defect attribute and the value change between the two records in each evaluated attribute (e.g. equal, larger or smaller). The goal was to study if there are certain attributes that are consistently affecting changing the state of the module from faulty to none, or the opposite. Results indicated that such technique can be very useful in studying the correlations between each attribute and the defect status attribute. Another prediction algorithm is developed based on statistics of the module and the overall dataset. The algorithm gave each attribute true class and faulty class predictions. We found that dividing prediction capability for each attribute into those two (i.e. correct and faulty module prediction) facilitate understanding the impact of attribute values on the class and hence improve the overall prediction relative to previous studies and data mining algorithms. Results were evaluated and compared with other algorithms and previous studies. ROC metrics were used to evaluate the performance of the developed metrics. Results from those metrics showed that accuracy or prediction performance calculated traditionally using accurately predicted records divided by the total number of records in the dataset does not necessarily give the best indicator of a good metric or algorithm predictability. Those predictions may give wrong implication if other metrics are not considered with them. The ROC metrics were able to show some other important aspects of performance or accuracy.

1. Introduction

As software testing and maintenance consume a large percent of software projects' resources, software quality is always a concern while developing or buying software products. For companies that develop software for years, it is important and useful for them to study their previous projects in trying to study what went wrong and what could be done better. Generally, one of the challenges in software cost estimation and fault prediction is in the ability to apply attributes or correlations from one domain to another and from one company to another. Studying correlations between the software and the project attributes and their goal attributes (such as the development duration for cost estimation and fault modules for defect prediction) can help in looking for commonalities in all those projects rather than differences. This is why such process will always be called prediction or estimation meaning that it will never be very accurate. However, an accuracy percentage such as 80% or 90% is considered acceptable.

The general assumption in studying correlations is that a module that is faulty should be distinguished from the other modules that are not by one or more attributes in the project or the product. One constraint of such studies is that open datasets cannot and do not contain all types of required attributes to see their correlation with the selected attribute. Some attributes that are related to the development process, the developers' characteristics, the business and the environment may hard to be gathered and quantified. However, it is hoped that we can make correlations from the available attributes. It is also hoped that those attributes are correctly

* Corresponding author.
E-mail addresses: ialsmadi@yu.edu.jo (I. Alsmadi), najadat@just.edu.jo (H. Najadat).

0965-9978/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2011.03.010

measured and collected. Another constraint is that those datasets usually come from different domains. There are no domain-related attributes to be collected and evaluated. Those attributes may have higher impact on the predicted class (i.e. fault modules) more than those known attributes.

Correlation describes the strength of a link or a connection between two or more attributes. If two attributes are mutually correlated, then any change in one attribute, will cause a change in the other attribute. In general, the value of the correlation coefficient ranges from −1.00 to +1.00. The value of −1.00 represents an ideal negative correlation while a value of +1.00 represents an ideal positive correlation. A value of 0.00 represents a lack of correlation. In statistics, there are several ways to calculate correlation and hence several types of correlation such as: Pearson product-moment, biserial, tetrachoric, polychoric, point-biserial, point-polyserial, and phi correlation. Pearson's Product-Moment Correlation (PPMC) is the most commonly used. It gives an indication of the relative overlap of a test sample when it is administered two different times or of two tests when they are administered concurrently. When measured in a population PPMC is designated by the Greek letter rho (ρ). When computed in a sample, it is designated by the letter r and is sometimes called "Pearson's r."

It can be given by the equation:

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n-1)S_x S_y}$$

where $\bar{x}$ and $\bar{y}$ are the sample means of x_i and y_i, s_x and s_y are the sample standard deviations of x_i and y_i and the sum is from $i = 1$ to n.

Correlation is used in statistics and several other fields to study the degree of cohesion between two or more attributes. In general, a correlation factor takes values between −1 and 1, where 1 means that the two attributes are always positively related. This means if we randomly select two samples from a dataset and evaluate those two related attributes, then if attribute one is increasing from sample A to B then attribute two will also *always* increase and vice versa. On the other hand, 100% negative correlation means that those two attributes are always out of synch as if attribute one increases then attribute two will always decrease and vice versa. In the middle of the correlation, the number zero means that there is no correlation at all between those two attributes and that we cannot make sense of any type of relation or correlation between the two attributes. In reality, attributes will fall between +1 and −1 where there is some weak or strong positive or negative correlation between the two attributes. There are similarities between correlation and data mining techniques as usually classification mining techniques will also try to look for relations between the different attributes and the goal attribute (i.e. the class). In software fault prediction, the class is the type of the module whether it is faulty or not. In software cost estimation, the class can be the project duration, size, cost, complexity, etc.

Typically correlations can be measured comparing two value attributes. However, in this project, we are trying to evaluate correlation between the numerical attributes in the datasets and the class attribute (i.e. a faulty or not faulty module) which is a categorical variable.

2. Related works

Several papers are presented about using mining for software fault prediction. Some of those papers discussed methods for fault prediction such as size and complexity metrics, multivariate analysis, and multi-co-linearity using Bayesian belief networks. Naïve Bayes technique is widely used for building classifiers due to its simplicity and optimal accuracy that it delivers based on Bayes theorem. Kagdy et al. presented a broad literature survey on approaches for Mining Software Repositories (MSR) within the framework of software evolution processes [1].

Many real life problems such as; credit evaluation, medical diagnosis or expert systems, speech, voice or image recognition are considered as classification problems. Throughout the years, several data classification techniques have been proposed. They include: decision tree induction, Bayesian classification, *K*-Nearest Neighbor, case-based reasoning, genetic algorithms, neural networks, rough set approach, and Support Vector Machines (SVMs) [2].

When developing a defect predictor, the probability of each class is calculated, given the attributes extracted from a module, such as Halstead and McCabe attributes (i.e. attributes that are relevant to predicting faulty modules). The module will then be classified according to the possibility with high probability. In [4], predictors with Naïve Bayes (NB) are developed for fault characteristics. Olivier et al. have used the Ant Colony Optimization (ACO) algorithm, and the Max–Min Ant System to develop the AntMiner+ model that classifies the dataset records into either faulty or non-faulty modules [3]. Paper [4] used Receiver Operating Characteristic (ROC) as a predictor evaluator. Similar to this paper, the paper evaluated prediction based on static code attributes. In this paper, we used ROC and several other prediction evaluators to evaluate the proposed prediction algorithms.

Predictors that are built using the previous techniques, suffer from high errors in assigning records to the correct class. As it will be shown in the results section, NB provides high number of incorrectly classified modules. As a result, many algorithms were built to overcome the significant drawbacks of NB. One of these algorithms that improve the accuracy relative to NB technique is Lazy Bayes Rules (LBR), which prevents the problem of high errors in NB while having comparable accuracy to NB. In other words, LBR algorithm produces a classification algorithm with very low errors. LBR has high computational overheads, due to high computational complexity required at classification time. This will reduce its usefulness as an alternative to NB. It is feasible only when the number of records in the training set is small.

A group of researchers conducted manual software reviews to find defective modules. They found that approximately 60% of defects can be detected manually [11]. Raffo found that the accuracy of correctly classified instances in defect detection of industrial review methods is relatively small [12].

One of the main factors that affect the prediction and accuracy of faulty modules is the kind and accuracy of dataset attributes collected. Static code attributes such as Lines of Code (LOC) and the McCabe/Halstead complexity attributes can be automatically collected, even for very large systems [9].

However, on the contrary, other methods, such as manual code reviews, are expensive to collect and labor intensive. It is estimated that 8–20 LOC/min can be inspected by humans [13]. In many studies defect predictors are trained using static code attributes defined by McCabe [7] and Halstead [6]. McCabe and Halstead are module based metrics, where a module is the smallest unit of functionality in the system as a whole. Code-based models can be used to increase the performance of design-level models and, thus, increase the efficiency of the verification and validation activities late in the development lifecycle [14]. In another aspect, some related works concludes that models use a combination of design and code level metrics outperform models which use only one of them [14].

In order to overcome using conventional (i.e. non-fuzzy) classification methods such as decision trees, and present defect predictor that improves the accuracy and ability of detecting defective modules for decision trees and Naïve Bayes based predictors, a new defect predictor is presented to improve predicted accuracy

[15]. Furthermore, the software attributes are chosen among the static code attributes. The advantage of those attributes is that they can easily be extracted from a source code, which prevents the consideration of human errors or subjectivity. These attributes are preprocessed with feature selection techniques to select the most relevant attributes for prediction.

Ostrand and Weyuker designed a scheme to evaluate the effectiveness of the current and proposed software development, validation, and maintenance techniques [16]. The scheme attempts to identify the fault characteristics in several areas with several possible values to describe the fault. Later on, same authors worked in the same field to describe the use of fault data from successive releases of commercial systems [17]. They presented some correlations between module size and fault-proneness and the relationships between pre- and post-release faults in modules.

Andersson and Runeson [18] conducted a replicated study of an earlier one by Fenton and Ohlsson [19] to study and correlate fault distribution on complex software. Their studies made some indications that some faults may not be always correlated to attributes. They may be due to a thorough process of testing and evaluation that help exposing such defects. Results showed also that majority of faults usually exist in a small number of modules. Size of the modules may not be always directly proportional to the number of faults.

Fenton and Pfleeger [26] showed that that static code attributes can never accurately be a certain indicator of the presence or absence of a fault. However, they are useful as probabilistic statements that the frequency of faults tends to increase in code modules that trigger the predictor.

2.1. The contribution of Menzies and coauthors

Tim Menzies (http://menzies.us/) along with several coauthors has several published papers in this subject. We referenced to some of them in this research [4,5,8,10,12,13]. His focus was in using AI and data mining in solving real world problems. In IEEE transactions on software engineering 2007, volume 33, a paper on fault prone prediction was written by Menzies et al. [4] and critiqued by Zhang and Zhang in the same volume [24]. In the same journal volume of [24], Menzies responded to the comments in the paper [25]. The major comment on the paper is that "because of the small percentage of defective modules, using probability of detection (PD) and Probability of False Alarm (PF) as accuracy measures may lead to impractical prediction models". Zhang et al. claimed that a precision of 25% in average for all evaluated datasets is very low although PD is high. In the later paper [25], Menzies et al. argued that to achieve high precision, this will impact probability of false (PF) and cause a large possible number of false alarms.

3. Methodology

In order to study the relations between dataset attributes and the dataset class (i.e. whether the module is faulty or not), a method is developed to measure the correlation between each attribute and the class. Several datasets are selected from PROMISE repository defect prediction section (i.e. http://promisedata.org/?cat=4). Table 1 shows a summary of the tested datasets.

3.1. Selecting the test data

In typical data mining processes, the evaluated dataset is divided into a part for training and another part for testing. The training dataset A machine learning algorithm is build based on the training set through an iterative process that is repeated until the error is reduced to a level below a certain threshold. This is usually repeated many times, with various parameter values, and often randomizing the order of the input in the goal of finding the best design for a given problem. Once the solution design is set, the test data that is unknown to the algorithm are applied through the architecture, and the error rate is recorded. Typical methods to choose which data to seize for the test dataset include using a hold out ratio for the test data (e.g., 1/3) and using a cross-fold validation such as 10-fold, where 10% of the dataset is randomly chosen for the test dataset, the algorithm is repeated 10 times, and the average validation error is recorded.

Table 1
Experimental datasets' details.

Dataset	No. of records	No. of attributes	No. of defects	Def %
JM1	10,884	22	2106	20
AR3	63	30	8	12.7
4_0_preprocessed	273	9	113	41.3
AR4	108	30	20	18.7
AR5	35	30	8	22.22
AR6	101	30	15	14.85
Datatrieve	129	9	11	8.5

In this research however, A correlation factor is built from all the dataset records based on simple algebraic algorithms that calculate the following metrics for each attribute:

- *Average*: The value calculates the overall average for this attribute (i.e. from all instances).
- *AverageTrue*: This metric calculates the average value for the attribute for all those instance in which the class (i.e. the state of the module whether defective or not) is true.
- *AverageFalse*: Similar to the previous one, calculates the average of all instances which have the class is false.
- *Total*: Calculates the summation of all attribute values from all instances.
- *TotalTrue*: Calculates the total for the attribute in which the class is true.
- *TotalFalse*: Calculates the total for the attribute in which the class is false.
- *CountTrue*: This metric calculate the total number of instances in which the class is true. This metric is fixed for all dataset attributes.
- *CountFalse*: It calculates the total number of instances in which the class is false. This metric is fixed for all dataset attributes.
- A correlation factor is developed for each attribute of the dataset to show the correlation between the attribute and the module class. More details on the developed algorithms are described later.

3.2. Correlation Algorithm 1

A tool is developed to generate a large number of randomly selected modules from the selected datasets. In every time, two records are selected. If the two modules have the same class (i.e. both classes are either faulty or not), then the selection is repeated. The selection is only considered when the two classes are different. The comparison is then made accordingly between all values of the attributes in those two records. A correlation factor is calculated based on evaluating the attributes' values between the two records where the class changes from faulty to none or the opposite. The process is repeated depending on the number of the sample taken.

The general steps for the developed algorithm are summarized below:

- From each selected dataset, two records are selected randomly with the following conditions: for the two records, the class should be different (i.e. one faulty and one not). If in any time the two classes are the same, one of the two records is reselected.
- One the two records with different class values are selected, for the attribute under evaluation, the two values of the two records are compared. The output of the comparison will have one of three possible results:

○ If the two values are equal, then there is no correlation, which means that in this selection, we cannot predict a correlation between the way the class is changing and the selected attribute.
○ If value 2 > value 1 and test 2 = false, test 1 is true (i.e. when value 1 increases the module became faulty, this is defined as negative correlation).
○ If (value 2 > value 1 and test 2 is true, test 1 is false, this is a positive correlation because when value 1 increases the module became better or not faulty.
○ If (value 2 < value and test 2 is false while test 1 was true, then this is also a positive correlation as when value 1 decreases the module became worse or faulty.
○ If (value 2 < value 1 and test 2 is true (while test 1 is false), this is assumed negative correlation.
○ In each case, 10, 100, and 1000 cycles are selected. Increasing the number of times that this process is repeated will improve the confidence in the correlation value that is measured. However, in some datasets, selecting 1000 cycles may take very long time especially if the size of the dataset is small (as the process requires two different selections every time). Tables 2–4 shows attributes' correlations for 100 and 1000 cycles respectively. Accuracy in many cases is improved when increasing the number of cycles. However, this is usually comes at the cost of performance. Accuracy is measured through evaluating predicted class value with the actual value in each row.

Fig. 1 shows the effect of increasing the cycle or the number of times correlation is measured on the attributes' correlations in each one of the tested datasets (JM1 dataset, 10, 100, and 500 cycles or loops). The horizontal line represented the dataset attributes. The figure shows that increasing the number improves the measurement of the correlation factor through eliminating the outliers' measures.

3.3. Correlation Algorithm 2

In the second correlation algorithm, correlation is evaluated between three selected records from the tested dataset rather than two records. For the three records, the selected attribute and the class are evaluated. Table 5 shows the correlation proposed values in the different possible cases (T: True, F: False, M: More, and L: Less). For records values, the first value is set as a reference, and then for the two next records, values are observed whether they are larger than value 1 (i.e. M), or less (i.e. L). Calculated correlation values here vary between −2 and +2. This is only an intermediate correlation value collected through the algorithm where +2 indicates stronger positive correlation than +1 and so on. Table 5

Table 2
Final_preprocessed dataset.

Attribute	Correlation	Prediction accuracy %	
		100	1000
LCOM	0.09	58.76	58.75
WMC	0.37	60.2	60.2
NPM	0.08	61.68	61.68
RFC	0.22	63.86	63.86
NOC	−0.05	45.6	45.6
DIT	0.21	40.5	40.5
LOC	0.21	71.89	71.89
CBO	−0.01	31.39	68.6
All		67.5	68.98

Table 3
Retrieve dataset.

Attribute	Correlation	Prediction accuracy %	
		100	1000
Diff block	−0.15	50.77	50.77
Dell LOC	−0.2	50	50
MOD rate	−0.21	26.15	26.15
Added LOC	−0.07	45.38	45.38
LOC_0	−0.27	60.76	60.76
Mod_lNow	−0.18	8.46	8.46
LOC_1	−0.06	60	60
Reuse LOC	−0.16	63	63
All		46.9	49

Table 4
AR3 dataset.

Attribute	Correlation	Prediction accuracy %	
		100	1000
Hal error	0.29	66.67	65.8
Cyc com	0.33	63.5	78.2
Cyc den	0.47	47.6	76
Mul con	0.38	74.6	74.9
Con count	0.15	63.5	73.8
Tot Op	0.34	66.67	79.8
Tot opn	0.41	65.08	84.6
Hal time	0.45	76.2	79.2
Hal vol	0.33	66.67	72.2
Hal eff	0.3	76.2	76.8
Nor Cyc	−0.18	47.6	83.2
Tot LOC	0.34	63.5	68
Hal diff	−0.26	34.9	78.6
Blk LOC	0.36	65.08	−69
Call pairs	0.37	74.6	70
Hal Voc	0.42	65.08	76
Form para	0.13	71.4	81.4
Uni opr	0.39	55.5	75.8
Dec den	0.24	41.27	78.4
Hal len	0.34	66.67	78.5
Hal level	0.41	62	30.9
Des comx	0.1	74.6	80.9
Com LOC	0.42	66.67	72.1
Des dens	0.12	76.2	79.6
Cod & Com	0.43	75.2	49
Un Opnd	0.45	60.3	19.9
Brnch CC	0.41	63.5	26.8
Exe LOC	0.35	65.08	−18.9
Dec count	0.37	63.5	19.6

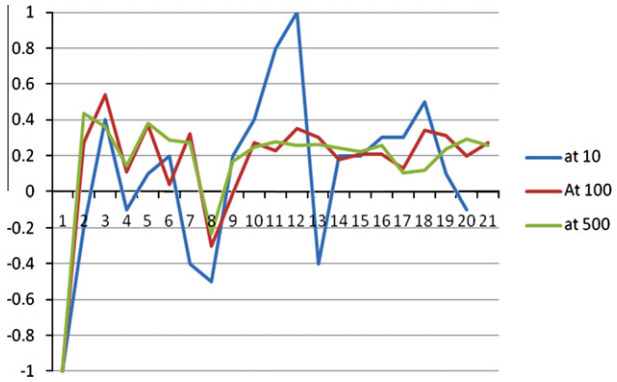

Fig. 1. Effect of changing the cycle size on jm1 dataset on prediction correlation.

Table 5
Correlation table for three records.

State 1	State 2	State 3	Value 2	Value 3	Correlation
T	T	T	Any	Any	Ignored
F	F	F	Any	Any	Ignored
T	T	F	M	M	+1
T	T	F	M	L	−1
T	T	F	L	M	+1
T	T	F	L	L	−1
T	F	T	M	M	+1
T	F	T	M	L	+1
T	F	T	L	M	−1
T	F	T	L	L	−1
T	F	F	M	M	+2
T	F	F	M	L	0
T	F	F	L	M	0
T	F	F	L	L	−2

shows correlation values for the attributes in Retrieve dataset after applying this algorithm. Some of the attributes have similar values to those in the previous algorithm (see Table 6).

3.4. Categorical or nominal correlation

In order to study the relation between values of each attribute and the class (i.e. defect or not), we have to consider techniques that deal with categorical (also called nominal or dichotomous) attributes' correlation as the class values fall into only two categories: defect or not defect.

Traditional correlation methods work for quantifiable data in which numbers are meaningful, usually quantities of some sort. There are many metrics such as average, mean, standard deviation that are meaningless in case of categorical variables. It cannot be used for purely categorical data, such as gender, brands purchased, or favorite color. Dealing with categorical variables when the categories are two (as in this study, faulty; True or false) is somewhat easy. It's easiest to follow this if you code the variable as 0–1. This could be done by replacing categorical variables with dichotomous dummy variables. A categorical variable with K levels can be replaced with $(K-1)$ binary dummy variables. Let's assume that we have two categorical variables, software project size (S) which may have three classes: small, medium, and large, and faulty case (F) which will have two classes: true, false. Table 7 shows how can those two categories be coded or transferred into dummy variables. Canonical correlation is used to deal with categorical variables.

In our research, categorical correlation is calculated as the following:

- First, for each attribute, we will calculate overall average of all attribute values, with the total number of values, and then calculate the total for values where class is false and another total where class is true.

Table 6
Retrieve dataset Algorithm 2.

Attribute	Correlation	Prediction accuracy %	
		100	1000
Diff block	−1	50.77	50.77
Dell LOC	−1	50	50
MOD rate	0.752	26.15	73.84
Added LOC	−1	54.6	45.38
LOC_0	−1	60.77	60.77
Mod_lNow	−1	8.46	8.46
LOC_1	−1	40	60
Reuse LOC	−1	63.07	63.07
All		39.23	53.84

Table 7
Categorical variables coding.

Row	Size	Dummy variables			
		D1	D2	F	D3
1	Small	0	0	Yes	1
2	Medium	0	1	Yes	1
3	Large	1	1	No	0
4	Small	0	0	No	0
5	Medium	0	1	Yes	1
6	Large	1	1	No	0

- Then correlation factor is calculated based on those three values.
- Later on, for each attribute value, class value is predicted (true or false) based on the above information.

3.5. Class prediction based on correlation algorithms

The class prediction is calculated based on the following information that is calculated from each one of the tested datasets: The attributes' average for all attributes in each dataset, total number of attributes where the class value is true, total number of attributes where the class value is false, the average value for all attributes where the class value is true and the average value where the class is false. Later on, for each record and in order to predict its class value, each attribute value in the record is evaluated to know its distance from the average values calculated for the class values false and true. The record predicted class will then be selected based on the attribute distance from the two averages where it will take the class value that it is closer to. Then this is calculated for all attribute values in the record and a prediction class for the record will be judged based on the majority vote (i.e. if there are more attributes with the class value true, then predicted class value is true and vice versa). The table below shows the accuracy in the prediction using this algorithm for the selected datasets.

The proposed developed algorithms study the overall characteristics of the dataset in order to help us predict the correlation between an attribute value and the class of the module. For each attribute, the attribute values are divided into two parts: those attributes that their class value is true, and another one for all attributes that their class is false. For each group, the average value for true (trueAverage) and false (falseAverage) are calculated. In the selection process, the instance value will be recorded to calculate its distance from both averages (i.e. distFromTrueAverage, and distFromFalseAverage).

In this first algorithm, the average and total are considered as the following (Pseudo code):

```
if (distFromTrue <= disFromFalse & totalTrue>=totalFalse) {
   predict = true; countTTrue=countTTrue+ N;}
else if (distFromTrue <= disFromFalse) {
   predict = true; countTTrue = countTTrue + 1;} else if
   (distFromTrue > disFromFalse & totalTrue < totalFalse) {
   predict = false; countTFalse= countTFalse+ N;}
      else if (distFromTrue > disFromFalse) {
predict = false; countTFalse = countTFalse + 1;}
```

where N is an integer value calculated based on the weight given when the two evaluated criteria are positive. The algorithm predict the class value for each attribute based on its distance from the true and the false averages and based on the total attribute true and false values.

At the end of each record class prediction, total true and false predictions are evaluated and the predicted class is given for the larger value. Table 8 and Fig. 2 show the prediction accuracy for all selected datasets along with studying the effect of changing the value of the constant *N*. This is a constant used in our developed algorithm. The constant values are selected to be: 1, 2, 4, and 10 respectively. In the algorithm, every time a positive correlation is found, the this number is added to the accumulated correlation accuracy. In many cases, increasing the value of the number *N* improved the prediction accuracy.

Table 9 shows selected attribute to the class correlation using SPSS. All datasets except ar6 shows somewhat consistent results.

3.6. Studying correlations between attributes

In previous section, the focus was on studying correlation between attributes and the class. In this section we will study the correlation between the different attributes. The goal of this part is to see if there are some dependencies between the software projects gathered attributes. For example, many software datasets collect several metrics related to complexity (those of McCabe and Halstead). This may raise a question in the possibility that some of those metrics are redundant and hence should be eliminated so that they will not produce biased results. The correlation between value attributes is calculated through Excel Pearson correlation formula. As amount of data is large and in order to show a summary of the results, the Table 10 below shows LOC total with some selected attributes among several datasets. The table shows that comparing the same attributes in the different datasets there are inconsistencies between the correlations between the same attributes. Although some variation is expected, however, some values show extreme change in correlation which is unexpected.

3.7. Comparison with selected data mining algorithms

Several known data mining algorithm are applied on the same fault prone datasets. The selected algorithms are widely used in literature and research papers. The goal was to evaluate whether the proposed statistical algorithms can compete with existed data mining algorithms. Table 11 shows a summary of the results of accuracy of data mining algorithms on the selected datasets. One problem with this accuracy is that it does not show the prediction accuracy of TN which is expected to be low. This will increase the percentage of false warnings.

Table 8
Prediction accuracy and the impact of modifying the constant (N).

Dataset	ACC (1)	ACC (2)	ACC (4)	ACC (10)
JM1	76.7	78.37	79.39	80.16
AR3	90.5	92	93.65	95.24
Preprocessed	71.16	71.16	71.16	59.85
AR4	84.11	86.92	86.92	85.98
AR5	88.89	86.11	86.11	86.11
AR6	75.25	77.22	81.18	85.15
Datatrieve	76.16	79.23	83.07	90.77

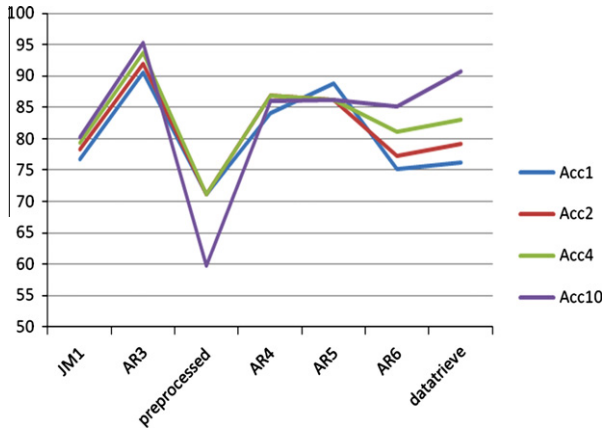

Fig. 2. Prediction accuracy and the impact of modifying the constant (N).

Table 9
SPSS nominal correlations for selected attributes.

Dataset	Dependent variable	Predictor	Correlation
ar3	total_loc	Def	0.627
ar3	hal_deff	Def	0.484
ar3	brnch_cnt	Def	0.653
ar3	cyc_com	Def	0.634
ar4	total_loc	Def	0.501
ar4	hal_deff	def	0.43
ar4	brnch_cnt	def	0.432
ar4	cyc_com	def	0.446
ar5	total_loc	def	0.695
ar5	hal_deff	def	0.605
ar5	brnch_cnt	def	0.632
ar5	cyc_com	def	0.668
ar6	total_loc	def	0.239
ar6	hal_deff	def	−0.07
ar6	brnch_cnt	def	0.419
ar6	cyc_com	def	0.366
jm1	loc1	def	0.308
jm1	cyc_com	def	0.257
jm1	hal_eff	def	0.143
jm1	brnch_cnt	def	0.257

Table 10
Attribute-to-attribute correlation.

Dataset/attribute	Attribute 2	Correlation
AR3/LOC1	Halstead difficulty	−0.4
AR3/LOC1	Halstead level	0.97
AR3/LOC1	Cyclic complexity	1
AR4/LOC1	Halstead difficulty	0.96
AR4/LOC1	Halstead level	0.90
AR4/LOC1	Cyclic complexity	−0.11
AR6/LOC1	Halstead difficulty	0.60
AR6/LOC1	Halstead level	−0.14
AR6/LOC1	Cyclic complexity	0.91

3.8. Evaluation measures and classification performance based on ROC

Receiver Operating Characteristic (ROC) is a generic graphical method to measure the classification performance. The ROC curves were originally developed in signal detection theory [23]. It has the advantage of being independent of class prior probabilities [21,22]. The ROC graph has two dimensions: the True Positive (TP: actual fault, correctly detected) rate at the *y*-axis and False Positive FP (datasets without defect that is evaluated as defected; false warning) rate at *x*-axis. The best classifiers have high TP rate and low FP rate [20]. Table 12 shows the truth table for the evaluated criteria.

Table 11
Datasets error prediction accuracy using selected data mining algorithms.

Dataset	Accuracy percentage %				
	DT J48	NB tree	Neural networks	SV	c4.5 PART
JM1	91.08	81.71	80.52	80.74	98.53
AR1	93.15	78.05	92.68	95.12	
PC1	93.32	89.17	93.59	92.96	93.68
AR6	0.8	0.84		0.85	0.85
AR5	94.44	86.1	100.00	83.33	

Here are some possible ideal situations.

- A system that can detect all true and false classes correctly will have FP and FN as zero values. In this system, PD will be one and PF will be zero which is the upper level ideal goal that we want while we may never reach.
- TP + FN complement each other and their total represents the number of faulty modules in the dataset. FP and TN also complement each other and their total represents the number of the correct modules. This means that the opposite of the first situation to have TP and TN as zeros. Consequently, PD will be zero and PF will be one; a predictor that cannot detect any instances correctly.

We used the previous terminology to evaluate the effectiveness of the fault prediction in the previously described algorithms. Table 13 shows number of values for each of those four items in the ROC truth table taken from some attributes of AR3 dataset for an earlier algorithm. The table shows two important prediction quality metrics that will be described later (PD and PF). The table shows also that according to this algorithm some attributes can be very good predictors of the error state more than other attributes. The correlation between such attributes and the module class is usually high. For example, attribute 4 is a very good predictor with its high probability of detection (PD) and its low false alarms (PF). Note the consistency of results from the different attributes where FP and FN are always getting the majority of results. This means that the majority of the predictions are either: FP where there is no defect and incorrectly predicted as defect, or FN where it was false and correctly predicted as false. Ideally, detectors should have high PDs, low PFs, and low effort. However, such ideal state rarely happens. A high PD value comes at the cost of high PF values and a low PF value comes at the cost of low PD value.

Table 14 shows a summary of the overall ROC attributes for all evaluated datasets. Those values are calculated for the second correlation algorithm discussed earlier. The algorithm uses accumulative correlation from all attributes based on given higher weight for more relevant attributes.

Table 12
ROC truth table.

	Detected as:	
	Negative	Positive
Faulty module?		
Yes	FN(A) (undetected fault)	TP (D) (correct fault)
No	TN No fault-correctly detected (as not faulty)	FP (false Warning) not faulty but detected as faulty)

Table 13
ROC truth table for AR3 dataset.

ATT	TP	FP	FN	TN	PD	PF
1	6	19	2	36	0.75	0.345
2	1	33	7	22	0.125	0.6
3	1	23	7	32	0.125	0.418
4	7	15	1	40	0.875	0.272
5	1	33	7	22	0.125	0.6
6	2	36	6	19	0.25	0.654
7	2	35	6	20	0.25	0.636
8	2	42	6	13	0.25	0.763
9	2	36	6	19	0.25	0.654
10	6	13	2	42	0.75	0.236
11	3	28	5	27	0.375	0.509
12	2	34	6	21	0.25	0.618
13	6	20	2	35	0.75	0.363
14	6	20	2	35	0.75	0.363

Table 14
A summary of the overall average of ROC attributes for all evaluated attributes (calculated as percentage).

Dataset	TP	FP	FN	TN
4_final	35.5	15	14	35.5
AR3	10	5	3	82
Retrieve	76	5.5	15.5	3
Jm1	10.4	37.6	8.8	43
AR6	7	17	8	68
AR4	10	4	10	76
AR5	19.5	10.5	3	67

Fig. 3 shows the ROC curve between TP and FP rates. Rates should be percentages of 100. However, numbers are multiplied by 100 to better display results. A test line is drawn to show it as a reference with the results. The test line is a line drawn with slope = 1 (i.e. will have 0,0 and 1,1 in the line as known points). Typically when values are above the random test line and below the ideal or perfect line, this is a good indication of the validity of the prediction its accuracy.

3.9. ROC metrics

The first metric to consider is Recall (also called probability of detection, PD) which defines the ratio of the detected true defective modules in comparison to the total number of defective modules. It is given by the equation:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The second metric is Precision. Precision defines the ratio of correctly detected modules. It is given by the equation:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

A good prediction model should achieve high Recall and high Precision (see Table 15).

The third metric is Probability of false alarms PF. It is calculated as: FP/(TN + FP).

Table 16 shows the results of evaluating those three metrics from the tested datasets. Investigations showed that the defect detection capability of industrial review methods can vary from 35% to 65% [4]. In comparison to Menzies paper [4], for our evaluated average of PD and PF are: 0.63 and 0.21 respectively relative to 0.71 and 0.25 on theirs. However, the selected datasets in the two cases are different. Most of the datasets evaluated in Table 16 exceeded the standard accuracy cut off (i.e. 70%). However, PD calculations for three datasets caused the overall average to be low.

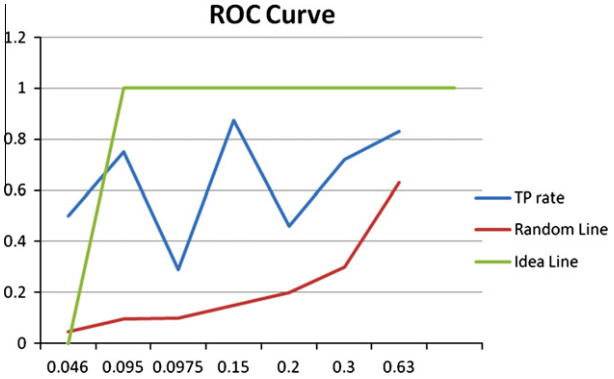

Fig. 3. ROC curve for all evaluated datasets in comparison with the random and ideal lines.

Table 15
TP and FP rates.

Dataset	TP	Total positive	TP rate	FP	Total negative	FP rate
4_final	97	134	0.72	42	140	0.3
AR3	6	8	0.75	5	53	0.095
Retrieve	99	119	0.83	7	11	0.63
Jm1	1142	5234	0.48	4092	5650	0.52
AR6	9.5	49.5	0.49	40	51.5	0.51
AR4	10	20	0.50	4	87	0.046
AR5	7	8	0.875	4	26	0.15

Table 16
Recall, precision and PF metrics.

Dataset	Recall-PD	Precision	PF
4_final	0.717172	0.70297	0.29703
AR3	0.769231	0.666667	0.057471
Retrieve	0.830601	0.932515	0.647059
JM1	*0.543*	0.27	0.46
AR6	*0.63*	0.226	0.465
AR4	*0.5*	0.714286	0.05
AR5	0.866667	0.65	0.135484
Avg	0.638018	0.626668	0.212258

Table 17
ROC metrics 1.

Dataset	TP	TN	FN	N	ACC %	Sensitivity %
4_final	97	98	37	274	71.16	72.38
AR3	6	52	2	63	92	75
Retrieve	99	4	20	130	79.23	83.19
Jm1	607	7923	1499	10,885	78.36	28.82
AR6	7	73	8	101	79.2	46.66
AR4	10	83	10	107	86.9	50
AR5	7	24	1	36	83.3	87.5

Table 18
ROC metrics 2.

DS	TN	TP	FN	FP	SPEC %	SEN %	EFF %	MCC
1	98	97	37	42	70	72.38	71.19	0.424
2	52	6	2	5	91.2	75	83.1	0.58
3	4	99	20	7	36.36	83.19	59.78	0.14
4	7923	607	1499	856	90.24	28.82	59.53	0.23
5	73	7	8	17	81.11	46.66	63.89	0.232
6	83	10	10	4	95.4	50	72.7	0.525
7	24	7	1	4	85.7	87.5	86.6	0.661

With the exception of AR6 and JM1, the algorithm precision is very good for the rest of the datasets. The overall precision average of 62.66% is very good relative to that of Menzies et al. algorithm of about 25% [4].

The fourth metric to consider is the accuracy. The accuracy is calculated for all correctly predictions to the total number of records. Or accuracy = TP + TN/*N*, where *N* is the total records in the dataset. Table 17 shows this metrics for all datasets. One issue with this metric is that it is highly dependent on the dataset distribution and may lead to wrong conclusions about the system performance.

The fifth metric is the sensitivity which is calculated as TP/(TP + FN) which is calculated in Table 17 above. The sensitivity metric indicates the large variation between the different datasets and the ability of the algorithm on correctly predicting the condition in cases it is really present.

The sixth metric is the Specificity (which is somewhat the opposite of sensitivity). This is calculated based on the equation:

$$\mathrm{SPEC} = \mathrm{TN}/(\mathrm{TN} + \mathrm{FP})$$

The seventh metric is the effort or efficiency which is calculated as:

$$\mathrm{EFF} = (\mathrm{SENS} + \mathrm{SPEC})/2$$

PD and effort are linked. The more modules that trigger the detector, the higher the PD is. However, effort also gets increases.

The eighth metric is the Matthews correlation coefficient. This is a measure of the quality of two binary classifications that can be used even if the two compared classes have very different sizes. It returns a value between −1 and +1, in which a +1 coefficient represents a perfect prediction, 0 a random prediction, and −1 an inverse prediction. The metric is calculated as the following:

MCC = φ = (TP ∗ TN − FP ∗ FN)/
sqrt((TP + FP) ∗ (TP + FN) ∗ (TN + FP) ∗ (TN + FN)). Table 18 summarized the results of the last four metrics. For MCC all predicted values are in the positive side with a maximum value of 0.66 which means that all predictions were positive. Results from MCC showed also that traditional accuracy calculated in the first metric does not necessarily give the best indicator of a good metric or algorithm predictability. The later metrics were able to show some other aspects of performance or accuracy.

3.10. A new prediction algorithm based on statistics

Through the process of developing the previous prediction algorithms, it is noticed that prediction accuracy is usually divided into two parts: Prediction accuracy for the classes of the value "correct" i.e. predicting accurately that the defective class is defective and prediction accuracy for the classes of the value "false or incorrect" where the instance or the module is not faulty. Those are usually referred to as True Positive (TP) and True Negative (TP) respectively. (More elaboration on those prediction metrics will be discussed later.) However, it is noticed that a good TP predictor is a week TN predictor. As a result, it is noticed that in many prediction algorithms proposed in literature, considering an algorithm accuracy based on TP only is deceptive as high TP will often cause a large percent of false warning or alarms (i.e. FN).

We further did a thorough statistical investigation to produce a prediction metrics based on several statistical metrics that are gathered from the dataset and all attributes. A program is developed to first collected the simple statistical metrics described earlier for each attribute (i.e. values total, max, min, average, total, count, and average for each attribute in which the class is true and other in which the class is false). Based on these statistical metrics, an initial correlation factor is developed for each attribute in every selected dataset. The simplest effective method of prediction applied is reading each attribute value and measure its Euclidean distance from the average of the false (i.e. the average of the attribute values in which the class is false) and the average from true. If the value is closer to the true average, the class is predicted as true and vice versa.

Each attribute will have two correlation factors: TPCorr. This is a correlation factor that indicates the percentage of true positive classes that were successfully predicted using this attribute. TNCorr is another metric collected for every attribute to indicate its prediction accuracy in predicting true negative classes. In the final stage, two attributes are selected from each dataset to predict its classes. Those attributes are of the highest TPCorr and TNCorr values. Table 19 shows an example of the attribute correlation for one of the tested datasets. As noticed, a good TN predictor is usually a week TP predictor.

Table 20 below shows the summary of metrics results based on TP and TN correlation calculations. For each dataset, two attributes are selected. Those are the ones that show the highest TP and TN values. The table below shows the prediction metrics for the statis-

Table 19
JM1 dataset attributes prediction.

Attribute	TN pred	TP pred
t numeric	0.77	0.028
e numeric	0.77	0.028
locCodeAndComment numeric	0.727	0.041
lOComment numeric	0.706	0.055
v numeric	0.718	0.0586
b numeric	0.718	0.0586
eV(g) numeric	0.674	0.063
total_Op numeric	0.698	0.064
v(g) numeric	0.704	0.0648
n numeric	0.6969	0.065
iv(g) numeric	0.703	0.065
total_Opnd numeric	0.697	0.065
lOCode numeric	0.695	0.0666
branchCount numeric	0.692	0.0685
lOBlank numeric	0.683	0.071
loc numeric	0.703	0.0759
uniq_Opnd numeric	0.662	0.078
d numeric	0.609	0.084
i numeric	0.626	0.085
uniq_Op numeric	0.516	0.107
l numeric	0.316	0.155

Table 20
ROC metrics for all tested datasets.

Dataset	PD	PF	Precision	ACC	SENS	SPEC	EFF	MCC
4_fin.	0.84	0.45	0.63	0.69	0.84	0.54	0.69	0.40
AR3	1	0.82	0.16	0.29	1	0.18	0.6	0.17
Retr.	1	1	0.91	0.91	1	0	0.5	NaN
Jm1	0.55	0.36	0.27	0.62	0.55	0.64	0.6	0.16
AR6	0.93	0.73	0.18	0.37	0.93	0.27	0.6	0.17
AR4	0.95	0.74	0.27	0.38	0.95	0.25	0.6	0.19
AR5	1	0.5	0.36	0.61	1	0.5	0.75	0.43

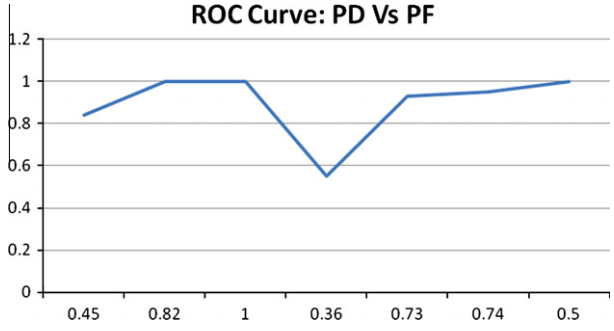

Fig. 4. ROC for the statistical algorithm.

tical algorithm proposed. It shows significant improvement over our previous proposed method and over several research surveyed methods. In some odd cases such as the third dataset where PF is 100%, this means that the system can detect all true faulty modules correctly; however, it has a very high amount of false alarms. We were able to modify this situation through selecting other alternative attributes for those selected in this experiment. In each dataset, one true and one false representative are selected for the evaluation. If both are true the class is predicted as true, if both are false, the class is selected as false. However, if there is a tie, we decided for most datasets to give the tie to the true class (i.e. faulty) as they are usually few. For those datasets that show PD and PF as 1, giving the equality to the false section will reverse the situation and TP zero and FP zero (instead of TN and FN).

Fig. 4 shows the overall ROC curve for the statistical algorithm which indicates that both PD and PF are high.

3.11. A comparison study

Results in this paper are compared with the fault prediction results from Menzies et al. paper [4]. However and in order to offset any possible inaccurate comparison due to the nature of the datasets selected, in this section we will evaluate our proposed algorithm on the same datasets used in [4]. Table 21 shows a summary of those datasets.

The statistical prediction metric described earlier is used to evaluate all dataset attributes. Each attribute correlation is divided into two parts. The first part is its TP prediction which shows its effectiveness in prediction faulty classes and TN which predict its accuracy in predicting in-faulty classes. Note that in dataset termi-

Table 21
Prediction accuracy summary [4].

Dataset	NO of modules	Percentage		
		Def	PD	PF
PC1	1108	6	48	17
MW1	404	7	52	15
KC3	459	9	69	28
CM1	506	9	71	27
PC2	5590	0.4	72	14
KC4	126	49	79	32
PC3	1564	10	80	35
PC4	1458	12	98	29
All			71	25

Table 22
Attributes predictors for PC4 dataset.

Attribute	Pred-TP	Pred-TN
BRANCH_COUNT numeric	0.047	0.608
CALL_PAIRS numeric	0.049	0.608
CONDITION_COUNT numeric	0.058	0.723
CYCLOMATIC_COMPLEXITY numeric	0.049	0.602
CYCLOMATIC_DENSITY numeric	**0.114**	0.417
DECISION_COUNT numeric	0.058	0.727
DECISION_DENSITY numeric	**0.107**	0.523
DESIGN_COMPLEXITY numeric	0.077	0.28
DESIGN_DENSITY numeric	0.0705	0.475
EDGE_COUNT numeric	0.0479	0.634
ESSENTIAL_COMPLEXITY numeric	0.1	0.21
ESSENTIAL_DENSITY numeric	0.066	0.117
HALSTEAD_CONTENT numeric	0.034	0.77
HALSTEAD_DIFFICULTY numeric	0.054	0.636
HALSTEAD_EFFORT numeric	0.0328	**0.782**
HALSTEAD_ERROR_EST numeric	0.038	0.753
HALSTEAD_LENGTH numeric	0.043	0.731
HALSTEAD_LEVEL numeric	0.091	0.36
HALSTEAD_PROG_TIME numeric	0.0328	**0.782**
HALSTEAD_VOLUME numeric	0.038	0.752
LOC_BLANK numeric	0.0575	0.683
LOC_CODE_AND_COMMENT numeric	0.0534	**0.812**
LOC_COMMENTS numeric	0.060	0.68
LOC_EXECUTABLE numeric	0.0568	0.696
LOC_TOTAL numeric	0.0568	0.72
MAINTENANCE_SEVERITY numeric	0.0938	0.41
MODIFIED_CONDITION_COUNT numeric	0.0609	0.706
MULTIPLE_CONDITION_COUNT numeric	0.0575	0.723
NODE_COUNT numeric	0.0486	0.639
NORMALIZED_CYLOMATIC_COMPLEXITY numeric	**0.1123**	0.412
NUM_OPERANDS numeric	0.0404	0.742
NUM_OPERATORS numeric	0.0445	0.723
NUM_UNIQUE_OPERANDS numeric	0.0479	0.697
NUM_UNIQUE_OPERATORS numeric	0.0705	0.546
NUMBER_OF_LINES numeric	0.0582	0.68
PARAMETER_COUNT numeric	0.07739	0.439
PERCENT_COMMENTS numeric	0.0767	0.651
Correlation	−0.726	

nologies when the value for the defect class attribute is "true", this means that this instance or class is defective and vice versa. This finding shows an important reason why no attribute can be a perfect predictor by itself as this attribute can be usually a good predictor for one of the classes (i.e. either true or false). Looking at the attributes in Table 22 for PC4 dataset, we see that a good TP predictor is not usually a good TN predictor. Table 23 shows ROC metrics results on PC4 dataset based on different attributes selections. We evaluated using selecting different numbers of attributes based on TP and TN values shown in Table 22. The alternatives to consider where to select one or more representatives from each side. Comparison results showed that it is best to show only two attributes: Those that have the highest Pred-TP and Pred-TN values from each dataset. Further the algorithm evaluate those two attributes prediction of the class and decide the class (whether correct or not) based on the prediction of those two attributes. This is coded as:

```
if (countTrue >= countFalse) {
row1[table.Cols[table.Cols.Count-2]]="True";}
  else {
row1[table.Cols[table.Cols.Count-2]]="False";}
```

where countTrue and countFalse are the Boolean predictions of the TP and TN attributes. If TP and TN are both false, the class is predicted as false, same thing if they are both true. However, in case of a tie, it is best decided to predict the class as faulty (i.e. true) based on the fact that the whole subject focus is on predicting faulty classes. This means that it is better to mistakenly predict that class as faulty while it is not than to miss it.

Table 24 shows the overall ROC metrics for all evaluated datasets based on the proposed statistical algorithm. Based on tuning the algorithm to focus on TP prediction, results showed that the PD prediction is very high. Based on focusing on improving PD, and based on the fact that the number of faulty modules in any dataset are usually less than 10%, it is expected to get a high PF (i.e. many false warnings) value.

PC2 dataset has a unique situation. As the dataset contains very low number of faulty modules (23 out of 5590 modules), all attributes showed a higher percent TN (more than 0.95) and a very low TP percent. This may explain why its precision is very low (an algorithm that will be able to detect correctly all 23 faulty modules out of the large number of correct ones may have such precision as pay off of its high predictability ratio. In many of those datasets, LOC_CODE_AND_COMMENT_numeric attribute was the highest attribute in terms of the TN value. Others include some of the Halstead attributes. Table 22 shows the summary of results comparison with those from Ref. [4].

Table 25 shows results comparison for PD and PF metrics between our algorithm and that of Ref. [4]. KC4 is removed from the comparison as we could not get the dataset from the public repositories.

Comparison results showed that our statistical algorithm were able to achieve much higher PD values. However, this comes at the cost of PF where we have higher PF or false alarms values. It is noticed however through the several cycles of testing is that achieving such high PD values will always cause such impact specially when values get close to 1% or 100%. This is shown in MW1 dataset where PD and PF are both 100%. A major contribution in this research is in making clear distinction between an attribute ability to predict correct from incorrect classes. Results showed that a good TP predictor may not be a good TN predictor. On the contrary, results showed that those two values are always in contradiction (as can be seen from Table 22 where correlation is 72.6%).

Table 26 shows all selected attributes for the prediction algorithm based on those attributes that gave the highest TP and TN values.

Table 24
ROC metrics for the statistical algorithm.

Dataset	PD	PF	Precision	ACC	SENS	SPEC	EFF	MCC
PC1	0.83	0.65	0.09	0.38	0.83	0.35	0.59	0.09
MW1	1	1	0.08	0.08	1	0	0.5	NaN
KC3	0.95	0.63	0.14	0.42	0.95	0.36	0.66	0.19
CM1	0.99	0.90	0.77	0.77	0.98	0.09	0.54	0.18
PC2	0.96	0.45	0.01	0.54	0.95	0.54	0.74	0.06
PC3	0.88	0.70	0.12	0.35	0.87	0.29	0.58	0.11
PC4	0.96	0.92	0.13	0.18	0.96	0.07	0.51	0.04
All	0.94	0.75	0.19	0.38	0.93	0.24	0.58	0.11

Table 25
Results comparison with those of Ref. [4].

Dataset	PD %	PD [4]	PF %	PF [4]
PC1	83	48	65	17
MW1	1	52	1	15
KC3	95	69	63	28
CM1	99	71	90	27
PC2	96	72	45	14
PC3	88	80	70	35
PC4	96	98	92	29
All	94	71	75	25

Table 26
Attributes selected in all datasets.

Dataset	TN attribute	TP attribute
PC4	LOC_Code_And_Comment numeric	ESSENTIAL_DENSITY numeric
PC3	LOC_Code_And_Comment numeric	DECISION_DENSITY numeric
KC3	LOC_Code_And_Comment numeric	HALSTEAD_LEVEL numeric
CM1	ACTION numeric	OPTION numeric
PC2	HALSTEAD_EFFORT numeric	HALSTEAD_LEVEL numeric
MW1	LOC_Code_And_Comment numeric	ESSENTIAL_DENSITY numeric
PC1	B numeric	lOBlank numeric
AR5	comment_loc numeric	halstead_level numeric
AR4	halstead_time numeric	decision_density numeric
AR6	code_and_comment_loc numeric	formal_parameters numeric
Dtr	Mod_Know numeric	Diff_Block numeric
AR3	code_and_comment_loc numeric	formal_parameters numeric
4Fin	LOC real	NOC real
JM1	e numeric	l numeric

3.12. Final notes

The previously proposed prediction algorithms showed that probability and statistics can be effective tools in this field. However, there are several subjective factors that should be con-

Table 23
ROC metrics for PC4 dataset based on different attribute selections.

Selected attributes	PD	PF	Prec.
LOC_CODE_AND_COMMENT numeric, CYCLOMATIC_DENSITY numeric, countTrue >= countFalse	0.994	0.798	0.298
LOC_CODE_AND_COMMENT numeric,NORMALIZED_CYLOMATIC_COMPLEXITY numeric countTrue >= countFalse	0.921	0.53	0.525
LOC_CODE_AND_COMMENT numeric,NORMALIZED_CYLOMATIC_COMPLEXITY numeric countTrue > countFalse, TP:0, TN:1280, FP:0, FN: 178.	0	0	0.877

sidered when analyzing software datasets to evaluate maintainability prediction and their correlation with the dataset attributes. The first assumption is that dataset metrics are gathered and collected in a consistent and symmetric way. For example, the definition of LOC, comments, and several other code metrics are subjective and can be counted in different methods. If we talk several open source metric tools, we will find that those numbers can vary for the same source code from one tool to another.

The second assumption is that the dataset attributes are the major players in deciding the quality and error aspects in those projects. We know in reality that there are several non-product related attributes that can affect software maintainability and quality. Those are usually related to the people, process and project attributes. The conformance to design and coding standards, the level of rigorousness and consistency of the adapted software development process and the CASE tools are some examples of those external non-product related factors.

Reaching a predictability of PD = 1 and PF = 0 is largely theoretical. However, in this research we showed that knowing each attribute positive classes' prediction (i.e. TP) and negative classes' prediction (i.e. TN) can help us tune our PD and PF metrics according to the investigated situation. In most cases, we are more interested in TP or PD as those can show us those faulty instances and their relations with the dataset attributes. However, our research showed that taking this only into consideration may give us wrong implications. For example, in all previous experiments, for those results that gave us 100% TP prediction (i.e. PD = 1), PF is usually high. If we want to improve PF and lower its value, this is usually possible. However, it comes on the account of lowering the value of PD.

The main equation that we used in the statistical prediction where:

```
if (countTrue >= countFalse) {
row1[table.Cols[table.Cols.Count-2]]="True";}
  else {
row1[table.Cols[table.Cols.Count-2]]="False";}
```

As we only used two attributes (the highest TP and the highest TN), it was necessary to give the equality where there is a tie to one side. The decision to predict the class as true when TP and TN prediction are not similar is based on two assumptions. First is that in all evaluated datasets, the number of faulty classes (i.e., true, TP) are less than the number of the number of correct classes or modules (i.e. TN). Second, is that we are interested to improve PD on the account of PF. This is why results in tables showed high PD values and also high PF values (which is not a good indicator). Removing the equality from the equation (countTrue >= countFalse), caused both PD and PF to be decreased significantly. As described earlier, this can be tuned based on the problem context.

Further statistical algorithms will be developed to be able to make a smart decision on where to give the equality (i.e. who should decide) based on further statistical metrics for the module in particular and the whole dataset in general.

Correlation factors for attributes can help us predict those attributes that are more relevant and effect of the record class value. Correlation can be positive or negative. However, results show that perhaps there are some attributes that are not listed on evaluated datasets that may have impact on the class values. This may explain why in some cases, prediction of class values for some datasets may not be improved above values that are not close to 100%.

In most cases, performance was not a major challenge. All proposed algorithms did not show any performance issues to further consider.

4. Conclusion

Software quality is always a major concern in software projects. Predicting faults and faulty modules improve our knowledge of those faults and how could they be corrected whether in this project or in future ones. The goal of fault prone modules' prediction using data mining and other techniques is to improve the software development process. This enables the software managers to effectively allocate project resources toward those modules that require more effort. This will eventually enable the developers to fix the bugs before delivering the software product to end users.

In this research, we proposed techniques to evaluate correlation between numerical values and categorical variables of fault prone datasets in order to be able to automatically predict faulty modules based on attribute values. A tool is developed to generate a large number of randomly selected modules from tested datasets and make a comparison in every cycle between two modules where one is faulty and the other is not. A correlation factor is calculated based on evaluating the attributes' values between two records where the class changes from faulty to none or the opposite. Results showed relatively consistent high correlation between the developed techniques and some known data mining techniques or algorithms.

A prediction algorithm is also developed based on studying statistics of the whole dataset and each attribute to make a prediction of each module class. This algorithm proved to be very powerful in predicting faulty modules in comparison with previous studies. A major contribution in this subject is that we differentiate between the attribute ability to predict correct from incorrect classes where correlation shows that a good TP predictor is not a necessary good TN predictor and vice versa. The developed tool can gather all those information automatically and allows us to change the selection of attributes and continuously evaluate prediction performance. This allows us to change our preferences and see in each dataset the attributes that can improve PD on the account of PF or the opposite.

ROC metrics are used to overcome the possibility that traditional methods for measuring prediction accuracy may ignore some important factors about the quality of prediction. Results from ROC metrics gave us better implication on the quality of the proposed algorithms and on how they could be improved in future.

References

[1] Kagdi Huzefa, Collard Michael L, Maletic Jonathan I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J Softw Main Evol: Res Practice 2007.

[2] Han Jiawi, Kamber Micheline. Data mining concepts and techniques. 2nd ed. Morgan Kaufman Publishers; 2006.

[3] Vanderuys Oliver, Martens David, Baesens Bart, Mues Christophe, De Backer Manu, Haesen Raf. Mining software repositories for comprehensible software fault prediction models. J Syst Softw 2008;81:823–39.

[4] Menzies Tim, Greenwald Jeremy, Frank Art. Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng 2007;33(1) [January].

[5] Menzies T, DiStefano J, Orrego A, Chapman R. Assessing predictors of software defects. In: Proc workshop predictive software models; 2004.

[6] Halstead M. Elements of software science. Elsevier; 1977.

[7] McCabe T. A complexity measure. IEEE Trans Softw Eng 1976;2(4):308–20.

[8] Menzies T, DiStefeno JS, Chapman M, Mcgill K. Metrics that matter. In: Proc 27th NASA SEL workshop software eng; 2002.

[9] Nagappan N, Ball T. Static analysis tools as early indicators of pre-release defect density. In: Proc int'l conf software eng; 2005.

[10] Menzes T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y. Implications of ceiling effects in defect predictors. PROMISE, submitted for publication.

[11] Shull F, Basili V, Boehm B, Brown A, Costa P, Lindvall M, et al. What we have learned about fighting defect. In: Proceedings of 8th international software metrics symposium, Ottawa, Canada; 2002. p. 249–58. <http://fc-md.umd.edu/fcmd/Papers/shulldefects.ps>.

[12] Menzies T, Raffo D, on Setamanit S, Hu Y, Tootoonian S. Model-based tests of truisms. In: Proceedings of IEEE ASE 2002; 2002. <http://menzies.us/pdf/02truisms.pdf>.

[13] Jiang Yue, Cukic Bojan, Menzies Tim, Bartlow Nick. Comparing design and code metrics for software quality prediction. Morgantown: The Lane Department of Computer Science and Electrical Engineering West Virginia University.
[14] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In: PROMISE'07: international workshop on ICSE workshops 2007; May 2007. p. 9-9.
[15] Holte Robert C. Very simple classification rules perform well on most commonly used datasets. University of Ottawa, Ottawa, Canada K1N 6N5.
[16] Ostrand TJ, Weyuker EJ. Collecting and categorizing software error data in an industrial environment. J Syst Softw 1984;300:289.
[17] Ostrand Thomas J, Weyuker Elaine J. The distribution of faults in a large industrial software system. In: Proceedings of the 2002 ACM SIGSOFT international symposium on software testing and analysis. ACM Press; 2002. p. 55–64.
[18] Andersson Carina, Runeson Per. A replicated quantitative analysis of fault distributions in complex software systems. IEEE Trans Softw Eng 2007;33(5).
[19] Fenton NE, Ohlsson N. Quantitative analysis of faults and failures in a complex software system. IEEE Trans Softw Eng 2000;26(8):797–814.
[20] de Carvalho Andre B, Pozo Aurora, Vergilio Silvia, Lenz Alexandre. Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm. In: 20th IEEE international conference on tools with artificial intelligence; 2008.
[21] Castro Cristiano Leite, Braga Antonio Padua. Optimization of the area under the ROC curve. In: 10th Brazilian symposium on neural networks; 2008.
[22] Bradley Andrew. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recogn 1997;30(7):1145–59.
[23] Egan JP. Signal detection theory and ROC analysis. Academic Press; 1975.
[24] Zhang Hingyu, Zhang Xiuzhen. Comments on data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng 2007;33(9) [September].
[25] Menzies Tim, Dekhtyar Alex, Distefano Justin, Greenwald Jeremy. Problems with precision: a response to "Comments on Data Mining Static Code Attributes to Learn Defect Predictors". IEEE Trans Softw Eng 2007;33:9. September.
[26] Fenton NE, Pfleeger S. Software metrics: a rigorous and practical approach. International Thompson Press; 1997.