Measuring Defect Datasets Sensitivity to Attributes Variation

Izzat M. Alsmadi
Texas A&M University-San Antonio, ialsmadi@tamusa.edu

Measuring Defect Datasets Sensitivity to Attributes Variation

Izzat Alsmadi

*Computer Information Systems Department
Yarmouk University, Irbid, Jordan
ialsmadi@yu.edu.jo*

Abstract

The study of the correlation between software project and product attributes and its modules quality status (faulty or not) is the subject of several research papers in the software testing and maintenance fields. In this paper, a tool is built to change the values of software data sets' attributes and study the impact of this change on the modules' defect status. The goal is to find those specific attributes that highly correlate with the module defect attribute. An algorithm is developed to automatically predict the module defect status based on the values of the module attributes and based on their change from reference or initial values. For each attribute of those software projects, results can show when such attribute can be, if any, a major player in deciding the defect status of the project or a specific module.

Results showed consistent, and in some cases better, results in comparison with most surveyed defect prediction algorithms. Results showed also that this can be a very powerful method to understand each attribute individual impact, if any, to the module quality status and how it can be improved.

Keywords: *Software metrics, software mutation, software metrics, and software testing.*

1. Introduction

The main goal of research projects in software testing is to first find techniques to automate software testing activities and second to improve the ability of the generated test cases to find unique defects and hence increase the overall coverage and quality. Coverage can be measured based on one or more attributes or concerns of the software code, requirements or design. This may include: path, statement, decisions, etc. coverage.

Software datasets include information about historical software projects. In software repositories, those datasets are usually collected based on a focus concern (e.g. cost estimation of defect prediction datasets). Attributes that are collected depend on the focus or the concern of those datasets. Each record in the dataset represents a particular module or instance. Collectors of those datasets try to include attributes that can help researchers understand the attributes that may impact the results of the software module or instance. Each module usually has an attribute called: class. The class attribute is the goal of the module. For example, in defect datasets, the class for each module is either defect or not (usually a Boolean attribute to indicate whether this particular module was defective or not during its usage).

The general assumption in studying correlations in defect datasets is that a module that is faulty should be distinguished from the other module(s) that are not by one or more attributes in the project or the product. Usually, it is assumed that data sets' collectors gathered all relevant information correctly. If some important attributes were not collected, this may impact the developed algorithms' ability to predict correctly what happened on those modules. Similarly, if attribute values were not accurate, or were relative or subjective, this may also risk the validity of any proposed prediction algorithm.

Some attributes that are related to the development process, the developers' characteristics, the business and the environment can be subjective and hence can be hard to gather or quantify. However, it is assumed that correlation can be made from the available information and attributes. It is also assumed that those attributes are correctly measured and collected. Another constraint is that those datasets usually come from different domains. Domain related attributes are not included in those datasets. In some cases, those attributes may have a higher impact on the predicted class relative to 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. It is used in statistics and several other fields to study the degree of cohesion or connectedness between two or more attributes. Correlation coefficients can range from -1.00 to +1.00. The value of -1.00 represents a perfect negative correlation while a value of +1.00 represents a perfect positive correlation. A value of 0.00 represents a lack of correlation between the two evaluated attributes.

In a high positive correlation, if we randomly select two samples from a dataset and evaluate those two related attributes, then if the first attribute is increasing from sample A to B then the second attribute will also *always* be increasing and vice versa. On the other hand, 100 % negative correlation means that those two attributes are always out of synch where if the first attribute increases then the second attribute will always decrease and vice versa. In the middle of the correlation line, the number zero means that there is no correlation at all between those two attributes and that we can't 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 methods and data mining techniques as usually the process of classification, prediction and association in data mining will also try to look for relations between the different attributes and the goal attribute (i.e. the class attribute). 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, in correlation, the process compares two value attributes (also called nominal attributes). However, in this paper, 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.

Sensitivity is a term used in prediction and statistics to indicate the percentage of TP (i.e. True Positive classes, classes that were defective and predicted as predictive) relative to the total of TP and FN (False Negative, which means that the class is not defective and is correctly predicted as not defective):

$$\text{Sensitivity} = (TP/(TP+FN)) \text{-----------------}$$

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. It is the probability that a test is positive. However, in this paper, by sensitivity, indicates studying the variation in value of the dataset attributes and evaluate its impact on the module class. The rest of paper is organized as the following: The next section shows relevant related papers. Later on, goals and approaches section is presented to demonstrate the technique used to mutate dataset attributes along with results and statistics. The last section summarizes summary and possible future work.

2. Related Works

Several papers are presented about software fault prediction based on studying defect datasets. Kagdy et al presented a comprehensive literature survey on approaches for Mining Software Repositories (MSR) in the context of software evolution processes [1].

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 predict faulty modules). The module will then be classified according to the possibility with high probability. Menzies et al In [2], 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]. In [2], Menzies et al 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 quality of the proposed prediction algorithms.

A group of researchers conducted manual software reviews to find defective modules. They found that approximately 60 percent of defects can be detected manually [4]. Raffo found that the accuracy of correctly classified instances in defect detection of industrial review methods is relatively small [5].

Ostrand et al designed a scheme to evaluate the effectiveness of the current and proposed software development, validation, and maintenance techniques [6]. 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 [7]. They presented some correlations between module size and fault-proneness and the relationships between pre- and post-release faults in modules.

Andersson et al [8] conducted a replicated study of an earlier one by Fenton et al [9] 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 et al [9] 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.

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 module 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.

Table 1. The Details of the Experimental Data Sets used in this Paper

Dataset	NO of records	No of attributes	No of Defects	Def %
JM1	10884	22	2106	20
AR3	63	30	8	12.7
40 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

It is noticed that prediction accuracy can be 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. However, it is noticed that a good TP predictor is usually a week TN predictor. As a result, it is also 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).

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

- Average: The value calculates the overall average for the selected attribute (i.e. from all instances).
- AverageTrue: This metric calculates the average values for the selected 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, the metric calculates the average of all instances which have the class as false.
- Total: Calculates the summation of all attribute values from all instances.
- TotalTrue: Calculates the total for the selected attribute in which the class is true.
- TotalFalse: Calculates the total for the selected 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.

Based on those previous metrics, a correlation factor is then developed for each attribute of the dataset to show the correlation between the attribute and the module class.

A thorough statistical investigation for all datasets is developed to produce prediction metrics based on the best combination of the previously described metrics that are gathered from the dataset and all attributes.

A program is developed to first collect simple statistical metrics for each attribute (i.e. values total, max, min, average, t otal, count) in addition to the previously described metrics. Those are calculated two times for each attribute, onetime when the class is true and another time when the class is false. Based on these statistical metrics, an initial correlation factor is developed for each attribute in e very selected dataset.

Distance similarity metrics (e.g. Euclidian, Cosine, etc.) are then used to find the best value that represent all modules for the evaluated datasets. 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: First, TPCorr. This is a correlation factor that indicates the percentage of true positive classes that were successfully predicted using this attribute. Second, 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 have the highest TPCorr and TNCorr values. Table 2 shows an example of the attribute correlation for one of the tes ted datasets (JM1). As mentioned earlier and can be seen in Table 3, a good TN predictor is usually a weak TP predictor.

Table 2. JM1 Dataset Attributes Prediction

Attribute	TN Prediction	TP Prediction
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	0.696	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

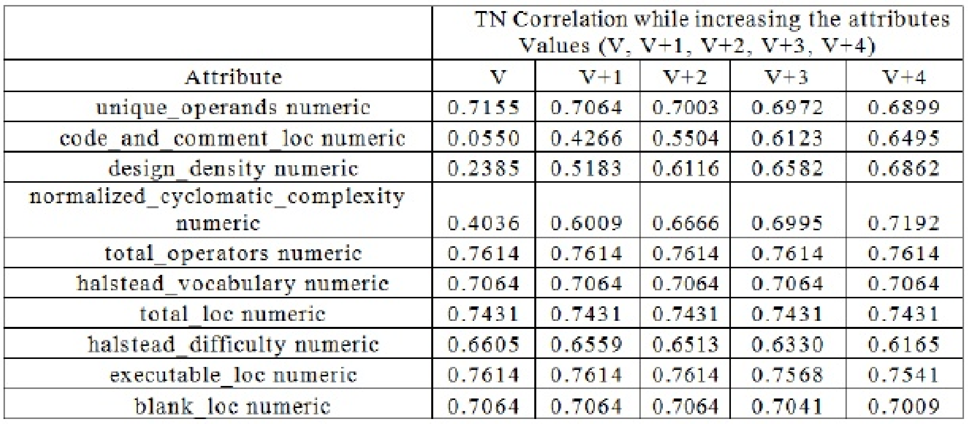
Table 3 below shows the summary of metrics results based on TP and TN correlation calculations. For each dataset, two attributes are selected. Those are the ones that show the highest TP and TN values. The Table below shows the prediction metrics for the statistical algorithm proposed. It shows significant 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 representatives are selected for the evaluation. If both are true the class is predicted as true, if both are false, the class is sel ected 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).

Table 3. ROC Metrics for all Tested Datasets

Data Set	PD	PF	Precision	Accuracy	Sensitivity
4_final	0.84	0.45	0.63	0.69	0.84
AR3	1	0.82	0.16	0.29	1
Retr.	1	1	0.91	0.91	1
Jm1	0.55	0.36	0.27	0.62	0.55
AR6	0.93	0.73	0.18	0.37	0.93
AR4	0.95	0.74	0.27	0.38	0.95
AR5	1	0.5	0.36	0.61	1

Tables 3 and 4 show samples from AR4 dataset in studying the TN correlation between attributes and the class of their module. This correlation is measured while changing the value of the attribute. It can be noticed that there are columns or attributes that are not affected through this specific addition on contrary to the others. This can help us understand the sensitivity of those attributes relative to the class value.

Table 4. Attributes Correlation in Response to Increasing their Values



	TN Correlation while increasing the attributes Values (V, V+1, V+2, V+3, V+4)				
Attribute	V	V+1	V+2	V+3	V+4
unique_operands numeric	0.7155	0.7064	0.7003	0.6972	0.6899
code_and_comment_loc numeric	0.0550	0.4266	0.5504	0.6123	0.6495
design_density numeric	0.2385	0.5183	0.6116	0.6582	0.6862
normalized_cyclomatic_complexity numeric	0.4036	0.6009	0.6666	0.6995	0.7192
total_operators numeric	0.7614	0.7614	0.7614	0.7614	0.7614
halstead_vocabulary numeric	0.7064	0.7064	0.7064	0.7064	0.7064
total_loc numeric	0.7431	0.7431	0.7431	0.7431	0.7431
halstead_difficulty numeric	0.6605	0.6559	0.6513	0.6330	0.6165
executable_loc numeric	0.7614	0.7614	0.7614	0.7568	0.7541
blank_loc numeric	0.7064	0.7064	0.7064	0.7041	0.7009

Looking at the two tables, there are several comments to notice.

1. The tables show the TN correlation between the attributes and the modules classes. This is calculated based on the percentage of correct predictions of the correct modules (i.e. predicted non defective and it is non defective). TP predictions are calculated for all datasets but are not shown due to paper size limitations. TP results are usually different and in most cases opposite to those of TN. A high TN value indicates a high correlation and high ability of prediction of the attribute of correct modules.
2. The first columns in the two tables show the value of TN when taking the actual attribute value. The small variation in some of the attributes is due to the fact that the process of calculating TN correlation is statistical and different rounds may produce slightly different results.
3. In the studied attributes, there are some attributes that showed no sensitivity or response to increasing or decreasing their values (e.g. total operators_numeric). For this specific attribute (total operators numeric), its correlation value is high: 0.7614. On the other hand, other attributes such as: code_and_comment_loc numeric show a very high sensitivity or instability through varying its value up and down. It should be mentioned that adding or decreasing 1,2,3, etc maybe the reason that some attributes are sensitive or not. It is noticed that some attributes require higher values in magnitude to be affected.
4. For some attributes, they are either affected by either increasing or decreasing the value of their attributes.

Table 5. Attributes Correlation in Response to Decreasing their Values

	TN Correlation while decreasing the attributes Values (V, V-1, V-2, V-3, V-4)				
Attribute	V	V-1	V-2	V-3	V-4
unique_operands numeric	0.7155	0.7155	0.7155	0.7178	0.7192
code_and_comment_loc numeric	0.0550	0.4082	0.5382	0.6032	0.6422
design_density numeric	0.2385	0.3944	0.5076	0.5756	0.6183
normalized_cyclomatic_complexity numeric	0.4036	0.6009	0.6666	0.6995	0.7192
total_operators numeric	0.7614	0.7614	0.7614	0.7614	0.7614
halstead_vocabulary numeric	0.7064	0.7064	0.7064	0.7087	0.7119
total_loc numeric	0.7431	0.7431	0.7431	0.7431	0.7431
halstead_difficulty numeric	0.6605	0.6651	0.6666	0.6743	0.6844
executable_loc numeric	0.7614	0.7614	0.7614	0.7568	0.7541
blank loc numeric	0.7064	0.7155	0.7186	0.7224	0.7247

All other datasets in the studied datasets were evaluated and results allowed us to understand how to optimize attributes ’ values through setting goals for typical or good quality software products.

4. Conclusion

In this paper, the study focused on studying the effect of changing the values of attributes on the class module to see their impact on the defect status of the module. Those values are then changed up and down to measure the sensitivity of the defect attribute with those changes. An algorithm is developed to evaluate the modules' correlation with all data sets' attributes. The approach can help us understand the attributes that are highly relevant in impacting the module defect status. The research divides attributes class prediction into TN and TP predictions based on defective and non-defective classes. In future, a thorough investigation will be developed through a large number of datasets. The power of this approach is that though which we can deeply understand variations in the software product attributes that can increase or decrease its overall quality.

References

[1] H. Kagdi, M. L. Collard and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution", Journal of Software Maintenance And Evolution: Research And Practice, **(2007)**.

[2] T. Menzies, J. Greenwald and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Transactions on Software Engineering, vol. 33, No. 1, **(2007)** January.

[3] O. Vanderuys, D.M. BartBaesens, C.Mues, M. De Backer and R. Haesen, "Mining Software Repositories for comprehensible Software Fault Prediction Models", The Journal of Systems and Software 81, **(2008)**, pp. 823-839.

[4] F. Shull, V. B. ad B. Boehm, A. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero and M. Zelkowitz, "What we have learned about fighting defects," in Proceedings of 8th International Software Metrics Symposium, Ottawa, Canada, **(2002)**, pp. 249–258, available from <http: //fc-md.umd.edu/fcmd/Papers/shull defects.ps>.

[5] T. Menzies, D. Raffo, S. on Setamanit, Y. Huand S. Tootoonian, "Model-based tests of truisms," in Proceedings of IEEE ASE 2002, **(2002)**.

[6] T. J. Ostrand and E. J. Weyuker, "Collecting and categorizing software error data in an industrial environment", The Journal of Systems and Software, **(1984)**, pp. 289-300.

[7] T. J. Ostrand and E. J. Weyuker, "The distribution of faults in a large industrial software system", In Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis, **(2002)**, pp. 55-64. ACM Press.

[8] C. Andersson and P. Runeson, "A replicated quantitative analysis of fault distributions in complex software systems", IEEE transactions on software engineering, Vol. 33, No. 5, **(2007)**.

[9] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system", IEEE transactions on software engineering, vol. 26, No. 8, **(2000)**, pp. 797-814.

Authors

Izzat Alsmadi

Izzat Mahmoud Alsmadi is an assistant professor in the department of computer information systems at Yarmouk University in Jordan. He obtained his Ph.D degree in software engineering from NDSU (USA), his second master in software engineering from NDSU (USA) and his first master in CIS from University of Phoenix (USA). He had B.sc degree in telecommunication engineering from Mutah University in Jordan. He has several published books, journals and conference articles largely in software engineering and information retrieval fields.