2008 International Conference on Emerging Technologies
IEEE-ICET 2008
Rawalpindi, Pakistan, 18-19 October, 2008

Software Quality Prediction Techniques: A Comparative Analysis

Sana Shafi, Syed Muhammad Hassan, Afsah Arshaq, Malik Jahan Khan, Shafay Shamail
Department of Computer Science
Lahore University of Management Sciences
Lahore - Pakistan
{sana.shafi, 07030019, 07030020, jahan, sshamail}@lums.edu.pk

***Abstract*—There are many software quality prediction techniques available in literature to predict software quality. However, literature lacks a comprehensive study to evaluate and compare various prediction methodologies so that quality professionals may select an appropriate predictor. To find a technique which performs better in general is an undecidable problem because behavior of a predictor also depends on many other specific factors like problem domain, nature of dataset, uncertainty in the available data etc. We have conducted an empirical survey of various software quality prediction techniques and compared their performance in terms of various evaluation metrics. In this paper, we have presented comparison of 30 techniques on two standard datasets.**

I. Introduction

Software quality prediction is helpful for better utilization of resources and to minimize error. Hence it helps in reducing the cost by early assessment of faults. It also helps in planning more thorough tests for modules that might have defects [35]. Many organizations and institutions are involved in the study of software quality prediction techniques and much work has been done to develop and refine models. These models have also been compared and benchmarked using different data sets [4], [25]. But still no study has concluded a best technique that always outperforms other techniques overall. Therefore more research is required to analyze more results and comparisons.

In this paper we have compared software quality prediction techniques on two recently available data sets in the PROMISE repository [4]. They are AR3 [4] and JEditData [4]. These data sets contain 63 instances with 30 attributes and 369 instances with 9 attributes respectively. Each instance is a module and is classified either as faulty or not faulty. These data sets were then analyzed and statistical analysis techniques were used to clean the data. The resultant datasets were used and all the techniques were benchmarked using these datasets. Rest of the paper is organized in the following manner. The Software Quality Prediction Techniques are given in section II. Section III describes different software quality prediction techniques which we have benchmarked. In section IV results and analysis of our work is given. Section V contains the conclusion and future work.

II. Related Work

Fenton et al. [13] in their paper provided a critical review of different models available and metrics for software quality prediction. Their work has suggested that some models do not consider the relationship between defects and failures, hence they do not model underlying problem correctly and sometimes miss-specify it.

Guo et al. [19] have proposed a model based on statistical techniques of Mixture model analysis for prediction of a module as fault prone and non-fault prone. To built the model they used Expectation Maximum Likelihood Algorithm along with size and complexity metrics. They also used Akaike Information Criterion (AIC) [2] in their model and claimed that this technique can be used to develop a model for predicting software quality even when prior knowledge about fault count and modules is unavailable.

Khoshgoftaar et al. [26] assessed quality of seven software quality prediction techniques, namely Classification and Regression Trees ($CART$), S-PLUS, the Sprint-Sliq algorithm, the C4.5 algorithm, the Treedisc algorithm, case base reasoning and logistic regression. The assessment was done based on a single criterion of the expected cost of misclassifications. They found that other criteria may help decide which modeling technique is better than the others as a single criteria is not enough.

Challagulla et al. [5] compared five metric types for error prediction which are McCabe, Halstead, Line Count, Operator and Branch Count. They were further subdivided into 18 definitions, such as, Cyclometric Complexity, Essential Complexity, Design Complexity, Lines of Code, Length, Volume, Difficulty, Intelligent Content, Effort, Error Estimate, Level, Programming time, Lines of Blank, Lines of Comment, Lines of Code and Comment, Unique Operands, Unique Operators, Total Operands, Total Operators and Branch Count. They used seven predictive models for comparison which are, Decision Trees, Naive-Bayes, Logistic Regression, Nearest Neighbor, 1-Rule, Regression and Neural Networks. They used both the continuous and the discrete values. They have evaluated different predictor models on four different real-time software defect data sets. The results indicate that a combination of 1R approach and Instance-based Learning algorithm along with the Consistency-based Subset Evaluation technique performed better than other techniques. They concluded that size and complexity metrics are not sufficient for accurately predicting real-time software defects.

Gyimthy et al. [20] assessed fault prediction for open source software. They compared different versions of Mozilla by

978-1-4244-2211-1/08/$25.00 ©2008 IEEE

consulting bug database of Bugzilla. They used the following Object Oriented metrics for comparison: Weighted Methods per Class, Depth of Inheritance Tree, Response for a Class, Number of Children, Coupling between Object Classes, Lack of Cohesion on Methods, Lack of Cohesion on Methods allowing Negative Value and Lines of Code. They concluded that Coupling between Object classes is the best in predicting fault proneness and all others are fairly good.

In 2006, Khan et al. [25] compared six Artificial Intelligence based techniques, Bayesian Belief Networks (BBN), Neural Networks (NN), Fuzzy Logic, Support Vector Machines (SVM), Expectation Maximum (EM) Algorithm and Case-Based Reasoning (CBR). They compared the techniques in terms of predicting a faulty module as non-faulty and non-faulty as faulty and decided which techniques required time, data, binary classification, n-class classification and expert knowledge.

Grunske [18] said that component based software engineering is used to develop complex embedded systems which needs to fulfil requirements regarding quality attributes such as safety, reliablity, avaliablity, maintainablity, performance, security and temporal correctness. He suggested a generic framework for predicting quality properties based on component based architectures. His model comprised of four elements, namely encapsulated evaluation models, operational/usage profiles, composition algorithms and evaluation algorithms.

III. Software Quality Prediction Techniques

In our research, we selected 30 different techniques for comparison. The following section briefly describe these techniques.

A. Case-Based Reasoning

Case-Based Reasoning (CBR) [27] uses computational intelligence for automated reasoning processes. A case based repository is made based on the previously made modules and according to this knowledge faults in new modules are predicted. It has a case-base library, which has all the cases already learnt using a predictive algorithm. Then a similarity function can be used to find the nearest neighbors. Then a solution algorithm is used to find the similar solution according to the case-based library built in the first step. The Ripple Down Rule *Ridor* [9] is similar to Case-Based Reasoning in which expert knowledge is incrementally added to the system. It handles similar cases and tries to make rules to minimize misclassification.

B. Bayesian Network

Bayesian Belief Network (*BBN*), $BayesNet$ [1], [10] is a probabilistic graphical model used to represent set of variables. It is composed of a Directed Acyclic Graph (*DAG*) [11] having nodes as variables and edges as conditional independencies. Naive Bayes (*NB* [24] algorithm implements Naive Bayes classifier using estimator classes. The choice of numeric estimator precision values is based on study of the training data.

C. Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) algorithm uses John Platt's algorithm [32] for training a Support Vector Classifier (*SVC*) [32]. It replaces all missing values globally and converts nominal attributes into binary. Attributes are normalized by default and pairwise classification is used to work out the solutions to multi-class problems.

D. Voted Perceptron

Voted Perceptron algorithm globally restores all missing values and converts nominal attributes into binary attributes. This algorithm has been implemented by Freund et al. [17].

E. Instance Based Learning

Instance Based Learning, $IB1$ algorithm [31] classifies according to nearest neighbor. Training instance nearest to the given test instance can be found using normalized Euclidean distance and the first instance found is used in case multiple instances have the same distance. IBk is another variation of IB1. In this algorithm attributes are normalized by default and appropriate values of k are chosen on the basis of cross-validation.

F. Nearest Neighbor Based Learning

Nearest Neighbor ($NNge$) [36] is a nearest neighbor based technique to classify modules as faulty and not faulty. The $KStar$ algorithm [7] classifies a given data instance to the most similar instance which is provided in the training set. The similarity is calculated with the help of some distance function, normally Euclidean distance.

G. Locally Weighted Learning

Locally Weighted Learning (LWL) [3] is a lazy learning algorithm that uses memory based learning and locally weighted linear regression. It uses similarity functions to find the nearest neighbors and then by using weighted average it finds the nearby points.

H. Attribute Selected Classifier

Attribute Selected Classifier ($AttributeClassifier$) algorithm reduces dimensionality of training and test data before the data is passed on to a classifier [30].

I. Regression Method

Regression Method algorithm performs classifications using *Regression*. This algorithm is binarized and builds one regression model for each value [14]. $Vote$ algorithm [12] combines classifiers using unweighted average of numeric predictions.

J. Decision Trees

Decision Stump [38] algorithm uses combination of a boosting algorithm and regression. *Decision Stump* performs regression based on mean-squared error as well as classification based on entropy while treating missing values separately. $J48$ is an extension of C4.5 decision tree algorithm used for weighted data.

Alternating Decision Tree ($ADTree$) [16] is another type of decision tree consisting of two types of nodes: decision nodes to identify a predict condition and prediction nodes which declare a value to add to the score based on the result of the decision node. Naive Bayes Trees ($NBTree$) [21] uses decision tree with *Naive Bayes* [24] classifiers. Quinlan [34] simplified the decision tree to another type named Reduced Error Pruning Tree ($REPTree$). It builds a decision tree with less errors and every subtree is replaced by the best leaf node with minimum error and then a bottom up approach is used to prune the nodes until the error does not increase.

K. Repeated Incremental Pruning

Repeated Incremental Pruning to Produce Error Reduction ($RIPPER$) implements a propositional rule learner, which was proposed by Cohen [8], [33] as an optimized version of Reduced Error Pruning (REP). This algorithm can also be used to predict the fault proneness in modules. It gives more efficient results on large samples.

Partial Decision Tree, $PART$ [15] is also a decision tree based algorithm to classify modules as faulty and not faulty. It builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule. Classification is done based on these rules. It combines two algorithms C4.5 and Repeated Incremental Pruning to Produce Error Reduction ($RIPPER$), but does not perform global optimization. $UserClassifier$ [39] builds user defined decision trees. It is an Interactive machine learning techniques which gives the liberty to users build classifiers.

L. Simple Logistic

Simple Logistic algorithm is a combination of *Linear Logistic Regression* and *Tree Induction* models [29]. It predict both nominal classes and numeric values. To give more accurate prediction in numeric values it creates trees with linear regression functions at the leaves.

M. Trees

Logistic Model Trees (LMT) [29] algorithm builds a tree dealing with binary and multi-class target variables, numeric and missing values. *Random Tree* [23] algorithm builds a tree that takes K randomly chosen attributes at each node. Pruning cannot be achieved using this algorithm. *Random Forest* [22] consists of many decision trees and outputs the class that is the mode of the classes output by individual trees.

N. Conjunctive Rule

Conjunctive Rule Learner ($ConjunctiveRule$) algorithm can predict nominal class labels based on rules constructed earlier by the learner. The rule consist of attribute values ANDed together that result in the class it belongs to [6].

O. Decision Table

decision table technique [28] uses majority classifier to predict modules as faulty and non-faulty .

P. ZeroR and OneR

Simplistic rule classifiers [37] are used to classify a module as faulty or non-faulty based on the extracted rules. It uses rules based on different values of attributes. The rule with the maximum similarity is chosen to classify a new module. The simplest of these rule classifiers is the majority class classifier, called $ZeroR$ [37]. This classifier takes a look at the target attribute and will output the value that is most commonly found in that attribute.

Another modified form of this algorithm is $OneR$ [37], in which an attribute is selected and then the best rule for that value of the attribute predicts the faultiness of the module.

IV. Implementation and Analysis

A. Implementation

The datasets used in this study were taken from PROMISE data repository [4]. These datasets were statistically checked for redundant data and missing values. The proportionality was also checked for the datasets to check for the normalization of the data. The first dataset named JEditData [4] is of a software jEdit [40]. It contains 369 instances and 9 attributes. The last attribute tells if the module had bugs or not. The second dataset named AR3 has 63 instances with 30 attributes each. The last attribute of this dataset also classified the module to be faulty or not faulty.

The datasets were divided into two parts, 66% of the instances were used as training set and 34% were used as testing data. Then each of the techniques was run on these datasets. We constructed 2x2 confusion matrix for each iteration of experiments and classified the prediction of each instance into one of the following cells of the confusion matrix.

- Classifying faulty module as faulty (True positive)
- Classifying faulty module as not faulty (False negative)
- Classifying not faulty module as not faulty (True negative)
- Classifying not faulty module as faulty (False positive)

We calculated following statistical evaluation metrics from the confusion matrix to compare the performance.

- *Precision (P)*: It is the proportion of positives correct predictions among all positive predictions.
- *Recall (R)*: It is the proportion of positives correct predictions among all positive examples.
- *Specificity (S)*: It is the proportion of negative correct predictions among all negative examples.
- *Accuracy (A)*: It is the proportion of correct predictions among all predictions.

The results are shown in Table I for jEdit [4], [40] and AR3 [4] datasets, represented as dataset 1 and dataset 2 respectively.

TABLE I
TECHNIQUES TESTED ON DATA SETS

Techniques	$RMSE_1$	$RMSE_2$	PI_1	PI_2	API
LWL	0.45	0.11	0.76	0.97	0.86
Regression	0.43	0.19	0.76	0.95	0.86
NBTree	0.46	0.05	0.72	0.97	0.84
Random Forest	0.46	0.16	0.68	0.97	0.82
Decision Stump	0.48	0.11	0.68	0.97	0.82
NaiveBayes	0.55	0.22	0.66	0.95	0.81
J48	0.51	0.19	0.66	0.95	0.8
PART	0.49	0.19	0.65	0.95	0.8
OneR	0.64	0.9	0.61	0.97	0.79
ADTree	0.48	0.09	0.61	0.97	0.79
Attribute Classifier	0.48	0.4	0.68	0.76	0.72
NNge	0.58	0.43	0.63	0.79	0.71
KStar	0.52	0.33	0.65	0.76	0.71
IB1	0.55	0.21	0.7	0.7	0.7
IBk	0.55	0.21	0.7	0.7	0.7
Random Tree	0.64	0.43	0.62	0.79	0.7
REPTree	0.49	0.34	0.69	0.68	0.69
Conjunctive Rule	0.48	0.34	0.68	0.68	0.68
BayesNet	0.48	0.34	0.68	0.68	0.68
JRip	0.45	0.34	0.67	0.68	0.68
SMO	0.59	0.21	0.65	0.7	0.68
Decison Table	0.48	0.34	0.62	0.68	0.65
Simimple Logistic	0.47	0.5	0.62	0.68	0.65
LMT	0.47	0.5	0.62	0.68	0.65
Ridor	0.58	0.37	0.58	0.68	0.63
ZeroR	0.5	0.34	0.54	0.68	0.61
User Classifier	0.5	0.34	0.54	0.68	0.61
Vote	0.5	0.34	0.54	0.68	0.61
CBR	0.5	0.34	0.54	0.68	0.61
Voted Perceptron	0.68	0.37	0.51	0.68	0.59

B. Analysis

On the basis of above mentioned evaluation metrics, we formulated Performance Index (PI) for each data set as given in equation 1.

$$PI_i = (P_i + R_i + S_i + A_i)/4 \quad (1)$$

Where P is precision, R is recall, S is specificity, A is accuracy and i is the data set number.

We calculated PI_i for both the data sets and further aggregated them to reach a consensus about the effectiveness of software quality prediction approaches. We calculated the Aggregated Performance Index (API) by taking simple arithmetic average as given in equation 2.

$$API = \frac{\sum PI_i}{n} \quad (2)$$

Where n is total number of data sets used for the experiments.

We also computed Root Mean Square Error $(RMSE)$ of prediction of each approach on each data set i as an evaluation metric, as given in equation 3.

$$RMSE_i = \sqrt{\frac{\sum_{j=1}^{n} e_{ij}^2}{n}} \quad (3)$$

$$e_{ij} = |predictedSolution_{ij} - desiredSolution_{ij}| \quad (4)$$

We used $RMSE_i$ and API_i to compare and analyze all aforementioned techniques. Results of both data sets show that LWL and the regression method outperform all other techniques. They give the highest value of API and their $RMSE_i$ is also relatively smaller than the other techniques. $NBTree$, $RandomForest$ and $DecisionStump$ algorithms are also comparatively good quality predictors as shown in Table I.

V. CONCLUSION AND FUTURE WORK

We used two new standard data sets of software quality and implemented 30 software quality prediction techniques and empirically evaluated them. The results of our empirical study show that the techniques classification via regression and LWL performed relatively better in both datasets.

In future we intend to run all these techniques on more datasets of even larger size and check the consistency of the results. Moreover, we intend to analyze the results for datasets which have relatively different number of fault prone and not fault prone instances.

REFERENCES

[1] S. Abe, O. Mizuno, T. Kikuno, N. Kikuchi, and M. Hirayama. Estimation of project success using bayesian classifier. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 600–603, New York, NY, USA, 2006. ACM.

[2] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, AC-19(6):716– 723, 1974.

[3] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artif. Intell. Rev.*, 11(1-5):11–73, 1997.

[4] G. Boetticher, T. Menzies, and T. Ostrand. Promise repository of empirical software engineering data http://promisedata.org/ repository. *West Virginia University, Department of Computer Science*, 2008.

[5] V. U. Challagulla, F. B. Bastani, and I.-L. Y. R. A. Paul. Empirical assessment of machine learning based software defect prediction techniques. In *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*.

[6] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, New York, NY, USA, 1977. ACM.

[7] J. G. Cleary and L. E. Trigg. K*: an instance-based learner using an entropic distance measure. In *Proc. 12th International Conference on Machine Learning*, pages 108–114. Morgan Kaufmann Inc., 1995.

[8] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.

[9] P. Compton and R. Jansen. Knowledge in context: a strategy for expert system maintenance. In *AI '88: Proceedings of the second Australian joint conference on Artificial intelligence*, pages 292–306, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[10] G. F. Cooper, D. H. Dash, J. D. Levander, W.-K. Wong, W. R. Hogan, and M. M. Wagner. Bayesian biosurveillance of disease outbreaks. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 94–103, Arlington, Virginia, United States, 2004. AUAI Press.

[11] M. Crochemore and R. Vérin. Direct construction of compact directed acyclic word graphs. In *CPM '97: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, pages 116–129, London, UK, 1997. Springer-Verlag.

[12] O. Dekel, F. Fischer, and A. D. Procaccia. Incentive compatible regression learning. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 884–893, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[13] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Trans. Softw. Eng.*, 25(5):675–689, 1999.

[14] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten. Using model trees for classification. *Mach. Learn.*, 32(1):63–76, 1998.

[15] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[16] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 124–133, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[17] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learing Theory*, pages 209–217, 1998.

[18] L. Grunske. Early quality prediction of component-based systems - a generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[19] P. Guo and M. R. Lyu. Software quality prediction using mixture models with em algorithm. In *First Asia-Pacific Conference on Quality Software.*

[20] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[21] M. Hall. A decision tree-based attribute weighting filter for naive bayes. *Know.-Based Syst.*, 20(2):120–126, 2007.

[22] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):832–844, 1998.

[23] H. Iba. Random tree generation for genetic programming. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 144–153, London, UK, 1996. Springer-Verlag.

[24] G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. pages 338–345.

[25] M. J. Khan, S. Shamail, M. M. Awais, and T. Hussain. Comparative study of various artificial intelligence techniques to predict software quality. *Proceedings of International Multitopic Conference (INMIC).*

[26] T. M. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Softw. Engg.*, 9(3):229–257, 2004.

[27] T. M. Khoshgoftaar, N. Seliya, and N. Sundaresh. An empirical study of predicting software faults with case-based reasoning. *Software Quality Control*, 14(2):85–111, 2006.

[28] R. Kohavi. The power of decision tables. In *ECML '95: Proceedings of the 8th European Conference on Machine Learning*, pages 174–189, London, UK, 1995. Springer-Verlag.

[29] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Mach. Learn.*, 59(1-2):161–205, 2005.

[30] K. Messer and J. Kittler. A comparison of colour texture attributes selected by statistical feature selection and neural network methods. *Pattern Recogn. Lett.*, 18(11-13):1241–1246, 1997.

[31] S. Okamoto and N. Yugami. Effects of domain characteristics on instance-based learning algorithms. *Theor. Comput. Sci.*, 298(1):207–233, 2003.

[32] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods: support vector learning*, pages 185–208, 1999.

[33] J. Plisson, N. Lavrač, D. Mladenić, and T. Erjavec. Ripple down rule learning for automated word lemmatisation. *AI Commun.*, 21(1):15–26, 2008.

[34] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[35] Z. A. Rana, S. Shamail, and M. M. Awais. Towards a generic model for software quality prediction. In *WoSQ '08: Proceedings of the 6th international workshop on Software quality*, pages 35–40, New York, NY, USA, 2008. ACM.

[36] S. Roy and B. Martin. Instance-based learning : Nearest neighbor with generalization. Technical report, University of Canterbury, Christchurch, New Zealand, 2002.

[37] R. Russo and P. S. A. Alvarez. Bayesian and neural networks for motion picture recommendation. Technical report, Boston College, 2006.

[38] A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt. A decision procedure for an extensional theory of arrays. In *Logic in Computer Science*, pages 29–37, 2001.

[39] M. Ware, E. Frank, G. Holmes, M. Hall, and I. H. Witten. Interactive machine learning: letting users build classifiers. *Int. J. Hum.-Comput. Stud.*, 56(3):281–292, 2002.

[40] S. Watanabe, H. Kaiya, and K. Kaijiri. Adapting a fault prediction model to allow inter language reuse. In *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 19–24, New York, NY, USA, 2008. ACM.