Journal of Computer Science 3 (5): 281-288, 2007
ISSN 1549-3636

Intelligence System for Software Maintenance Severity Prediction

[1]Parvinder Singh Sandhu, [2]Sunil Kumar and [3]Hardeep Singh
[1]Department of Computer Science and Engineering, Guru Nanak Dev Engineering College, Ludhiana, Punjab, India
[2]Department of Computer Science and Engineering Lala Lajpat Rai Institute of Engineering and Technology, Moga, India
[3]Department of Computer Science and Engineering Guru Nanak Dev University, Amritsar, Punjab, India

Abstract: The software industry has been experiencing a software crisis, a difficulty of delivering software within budget, on time, and of good quality. This may happen due to number of defects present in the different modules of the project that may require maintenance. This necessitates the need of predicting maintenance urgency of the particular module in the software. In this paper, we have applied the different predictor models to NASA five public domain defect datasets coded in C, C++, Java and Perl programming languages. Twenty one software metrics of different datasets and Java Classes of thirty five algorithms belonging to the different learner categories of the WEKA project have been evaluated for the prediction of maintenance severity. The results of ten fold cross validation are recorded in terms of *Accuracy*, Mean Absolute Error (*MAE*) and Root Mean Squared Error (*RMSE*) for different project datasets. The results show that logistic model Trees (LMT) and Complimentary Naïve Bayes (CNB) based Model provide a relatively better prediction consistency compared to other models and hence, can be used for the maintenance severity prediction of the software. The developed system can also be used for analysis and to evaluate the influence of different factors on the maintenance severity of different software project modules.

Key words: Prediction Models, Metrics, Accuracy, Maintenance Severity, MAE, RMSE

INTRODUCTION

Software maintenance is defined as the process of modifying existing operational software after delivery to the customer to correct faults, to improve performance, and/or to adapt the product to a changed environment. Maintenance is inevitable for almost any kind of product. However, most products need maintenance due to the wear and tear caused by use. On the other hand, software products do not need maintenance on this count, but need maintenance to correct errors, enhance features, port to new platforms etc. Maintenance requests[1] can be of corrective, perfective, adaptive, user support and preventive types. The software maintenance life cycle (SMLC) concept recognizes four stages [2, 3] in the life of an application software system: introduction, growth, maturation, and decline.

The software industry has been experiencing a software crisis, a difficulty of delivering software within budget, on time, and of good quality. At the same time, the industry has experienced a dramatic increase in the software life cycle costs of maintenance. Pigoski [4] illustrates that the percentage of the industry's expenditures used for maintenance purposes was 40 percent in the early 1970s, 55 percent in the early 1980s, 75 percent in the late 1980s, and 90 percent in the early 1990s. Given its dominance in the industry, the study of software maintenance is increasingly prudent. It has also been noted [5] that over 50% of programmer effort is dedicated to maintenance. According to Mall [12] the effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratios. Given this high cost, some organizations are beginning to look at their maintenance processes as areas for competitive advantage.

With real-time systems becoming more complex and unpredictable, partly due to increasingly sophisticated requirements, traditional software development techniques might face difficulties in satisfying these requirements. Future real-time software systems may need to dynamically adapt themselves based on the run-time mission-specific requirements and operating conditions. This involves dynamic code synthesis that generates modules to provide the functionality required to perform the desired operations in real-time. However, this necessitates the need to develop a real-time assessment technique that classifies these dynamically generated systems as being faulty / maintenance free [6].

A variety of software maintenance predictions techniques have been proposed, but none has proven to be consistently accurate. These techniques include

Corresponding Author: Parvinder Singh Sandhu, Assistant Professor, Department of Computer Science and Engineering, Guru Nanak Dev Engineering College, Ludhiana(Punjab)- 141 0006 India. Tel: 9855532004

statistical method, machine learning methods, parametric models and mixed algorithms. Therefore, there is a need to find the best prediction technique for a given maintenance prediction dataset (MP) to calculate the maintenance severity. In this paper we have proposed a prediction model for quantifying the impact of defects on the overall environment by predicting maintenance severity.

The basic hypothesis of software quality prediction is that a module currently under development has defects if a module with the similar product or process metrics in an earlier project (or release) developed in the same environment had defects [7]. Therefore, the information available early within the current project or from the previous project can be used in making predictions. This methodology is very useful for the large-scale projects or projects with multiple releases.

Maintenance managers can apply existing techniques that have been traditionally been used for other types of applications. One system is not enough for prediction purposes. The empirical study detailing software maintenance for web based java applications can be performed to aid in understanding and predicting the software maintenance category and effort [8].

With the advent of Total Quality Management, organizations are using metrics to improve quality and productivity [9]. Software maintenance organizations are no exception. In 1987, the U.S. Navy established centralized Software Support Activity (SSA) to provide software maintenance for cryptologic systems. At that time two systems were supported and a software maintenance metrics program was established to support the goals of the SSA.

Visual approach [10] can be used to uncover the relationship between evolving software and the way it is affected by software bugs. By visually putting the two aspects close to each other, we can characterize the evolution of software artifacts.

Software maintenance is central to the mission of many organizations. Thus, it is natural for managers to characterize and measure those aspects of products and processes that seem to affect cost, schedule, quality, and functionality of a software maintenance delivery [13]. The importance o software maintenance in today's software industry can not be overestimated.

Statistical, machine learning, and mixed techniques are widely used in the literature to predict software defects. Khoshgoftaar [14] used zero-inflated Poisson regression to predict the fault-proneness of software systems with a large number of zero response variables. He showed that zero-inflated Poisson regression is better than Poisson regression for software quality modeling. Munson and Khoshgoftaar [15,16] also investigated the application of multivariate analysis to regression and showed that reducing the number of “independent” factors (attribute set) does not significantly affect the *Accuracy* of software quality prediction.

Menzies, Ammar, Nikora, and Stefano[17] compared decision trees, naïve Bayes, and 1-rule classifier on the NASA software defect data. A clear trend was not observed and different predictors scored better on different data sets. However, their proposed ROCKY classifier outscored all the above predictor models. Emam, Benlarbi, Goel, and Rai [18] compared different case-based reasoning classifiers and concluded that there is no added advantage in varying the combination of parameters (including varying nearest neighbor and using different weight functions) of the classifier to make the prediction *Accuracy* better.

Bayesian Belief Networks (also known as Belief Networks, Causal Probabilistic Networks, casual Nets, Graphical Probability Networks, Probabilistic Cause-Effect Models, and Probabilistic Influence Diagrams)[19] have attracted much recent attention as a possible solution for the problems of decision support under uncertainty. Although the underlying theory (Bayesian probability) has been around for a long time, the possibility of building and executing realistic models has only been made possible because of recent algorithms and software tools that implement them. Clearly defects are not directly caused by program complexity alone. In reality the propensity to introduce defects will be influenced by many factors unrelated to code or design complexity.

Many modeling techniques have been developed and applied for software quality prediction. These include logistic regression, discriminant analysis [20, 21], the discriminative power techniques, Optimized Set Reduction, artificial neural network [22-23], fuzzy classification Bayesian Belief Networks (Fenton & Neil, 1999), recently Dempster-Shafer Belief Networks. For all these software quality models, there is a tradeoff between the defect detection rate and the overall prediction *Accuracy*. The software quality may be analyzed with limited fault proneness data [24].

METHODOLOGY

The following steps are proposed for the prediction of maintenance severity:

1. Deciding the relevant attributes of software maintenance prediction and choosing the metric corresponding to the selected attribute that could have contribution towards prediction of maintenance urgency/severity.
2. The Collection of sampled relevant MP data, analyze and refine metrics data for different projects.
3. Evaluate different prediction techniques and selecting the best technique based on *Accuracy* Percentage, Mean Absolute Error (*MAE*) and Root Mean Squared Error (*RMSE*).

Table 1: Details of the Projects Datasets used in the study

Sr.No	Project	# of instances	# of instances with defects	Preprocessing	Source Code
1	KC1	2107	293	Missing values removed	C++
2	JM1	10878	2102	Missing values removed	C
3	PC4	370	178	Missing values removed	C
4	KC3	458	29	Missing values removed	Java
5	KC4	125	60	Missing values removed	Perl

4. Developing an intelligence system using the best technique as evaluated in the previous step.
5. Testing of the developed system.

The real-time defect data sets used in this paper has been accessed from the NASA's MDP (Metric Data Program) data repository. The KC1 data is obtained from a science data processing project coded in C++, containing 2107 modules. Out of these 293 modules have defects. The JM1 data is obtained from a predictive ground system project, written in C, containing 10878 modules. Out of these 2102 modules have defects. The PC4 data is collected from a software system coded in C, containing 370 modules. Out of these 178 modules have defects. The KC3 data is collected from a software system coded in Java, containing 458 modules. Out of these 29 modules have defects. The KC4 data is collected from a software system coded in Perl, containing 125 modules. Out of these 60 modules have defects as shown in Table 1. All these data sets varied in the percentage of defect modules, with the KC3 dataset containing the least number of defect modules and the JM1 dataset containing the largest.

The Table 2 shows the different types of predictor software metrics (independent variables) used in our analysis. These complexity and size metrics include well known metrics, such as Halstead, McCabe, line count, operator/operand count, and branch count metrics. Halstead metrics are sensitive to program size and help in calculating the programming effort in months. The different Halstead metrics include length, volume, difficulty, intelligent count, effort, error, and error estimate. McCabe metrics measure code (control flow) complexity and help in identifying vulnerable code. The different McCabe metrics include cyclomatic complexity, essential complexity, design complexity and lines of code. The target metric (dependent variable) is the "Severity".

RESULTS AND DISCUSSIONS

The Severity value quantifies the impact of the defect on the overall environment with 1 being most severe to 5 being least severe. For, example severity 1 may imply that the defect caused a loss of functionality without a workaround where severity 5 may mean that the impact is superficial and did not cause any disruptions to the system.

Table 2: Details of the Metrics Group used in the study

Metric Type	Metric	Definition
McCabe	v(G)	Cyclomatic Complexity (CC)
	ev(G)	Essential Complexity(EC)
	iv(G)	Design Complexity(DC)
	ELOC	Lines of Code Executable
Derived Halstead	N	Length
	V	Volume
	L	Level
	D	Difficulty
	I	Intelligent Count
	E	Effort
	B	Effort Estimate
	T	Programming Time
Line Count	LOCode	Lines of Code
	LOComment	Lines of Comment
	LOBlank	Lines of Blank
	LOCodeAndComment	Lines of Code and Comment
Basic Halstead	UniqOp	Unique Operators
	UniqOpnd	Unique Operands
	TotalOp	Total Operators
	TotalOpnd	Total Operands
Branch	BranchCount	Total Branch Count

Table 3: Maintenance Severity values used in this study

S.No	Project	Number of instances having maintenance severity value				
		1	2	3	4	5
1	KC1	48	207	28	8	2
2	JM1	343	163	1146	64	386
3	PC4	58	40	80	0	0
4	KC3	0	25	3	0	1
5	KC4	3	23	31	0	3

The Table 3 shows the no of modules with defect associations of different projects having maintenance severity value of 1, 2, 3, 4 and 5 respectively. We have used MATLAB 7.2 and Java Classes of Weka Project [11] to conduct these experiments. Thirty five algorithms belonging to the six learner categories of the WEKA project have been evaluated on five projects for the prediction of maintenance severity. The ten fold cross validation results are recorded in terms of *Accuracy*, *MAE* and *RMSE* for different project as specified earlier. Table

4, Table 5 and Table 6 are derived from Table 7 and Table 8 by selecting the best algorithm from each category based on *Accuracy*, *MAE* and *RMSE*. The detailed tables implementing all the prediction algorithms on five different projects are shown in the Appendix section of the paper.

Table 4: Accuracy Percentage for different projects shown by different prediction algorithms (PA)

PA	Accuracy Percentage for Different Projects Considered				
	KC1	JM1	KC3	KC4	PC4
CNB	62.4573	53.8535	41.3793	65	52.2472
LWL	69.2833	54.6622	82.7586	55	53.9326
CVR	70.9898	56.5176	86.2069	58.3333	54.4944
LMT	70.3072	55.3283	86.2069	65	58.9888
RBF	66.8942	54.0913	79.3103	53.3333	53.9326
SL	70.3072	54.9001	86.2069	65	57.3034

Table 5: Mean Absolute Error (*MAE*) for different projects shown by different prediction algorithms

PA	*MAE* for Different Projects Considered				
	KC1	JM1	KC3	KC4	PC4
CNB	0.1502	0.1846	0.2345	0.14	0.191
LWL	0.1855	0.2513	0.106	0.2207	0.2329
CVR	0.1784	0.2416	0.1011	0.2255	0.2257
LMT	0.1887	0.246	0.1127	0.2145	0.2242
RBF	0.1833	0.2516	0.0914	0.2147	0.2344
SL	0.1887	0.248	0.1172	0.2145	0.2251

Table 6: Root Mean Squared Error (*RMSE*) for different projects shown by different prediction algorithms (PA)

PA	*RMSE* for Different Projects Considered				
	KC1	JM1	KC3	KC4	PC4
CNB	0.3875	0.4296	0.4842	0.3742	0.437
LWL	0.3085	0.3552	0.2787	0.3548	0.348
CVR	0.3064	0.3506	0.2317	0.3396	0.3425
LMT	0.3076	0.3524	0.2224	0.3285	0.3395
RBF	0.3133	0.3563	0.2588	0.3694	0.3514
SL	0.3076	0.3533	0.2235	0.3285	0.3371

We have used abbreviations in the tables and figures to represent different predictive methods and other terms: Complement Naive Bayes (CNB), Logistic Model Trees (LMT), Classification via Regression (CVR), RBFNetwork (RBF), Simple Logistic (SL), Predictive Algorithm (PA), Mean Absolute Error (*MAE*) and Root Mean Squared Error (*RMSE*).

Prediction based on Accuracy Percentage: The Fig. 1 derived from Table 4 shows that Classification via Regression (CVR) performed better for KC1 and JM1 data and LMT performed better for KC4 and PC4 data sets. For KC3 data sets the both have performed equally. But, there is very less difference in *Accuracy* values of LMT and CVR for KC1 and JM1 as compared to *Accuracy* values for KC4 and PC4. Though, CVR produced good *Accuracy* results, yet LMT is much better for maintenance severity analysis based on *Accuracy* percentage results.

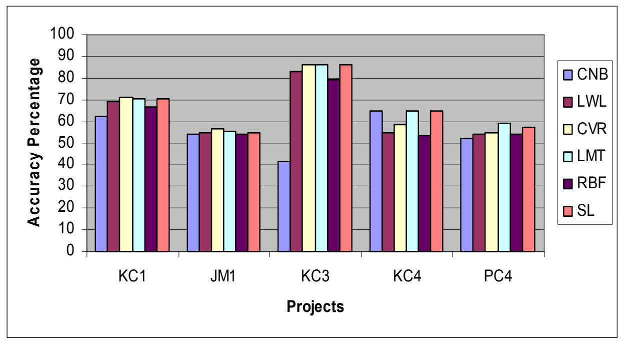

Fig. 1: Accuracy Percentage vs. Projects with different prediction techniques

Prediction based on Mean Absolute Error: The Fig. 2 derived from Table 5 shows that Complement Naive Bayes (CNB) performed better for KC1, JM1, KC4 and PC4 data sets. For KC3 data sets RBF Network has performed better. Though, RBF produced good results, yet CNB is much better for maintenance severity analysis based on mean absolute error calculations. It suggests the use of CNB as one of the foremost technique for maintenance severity prediction

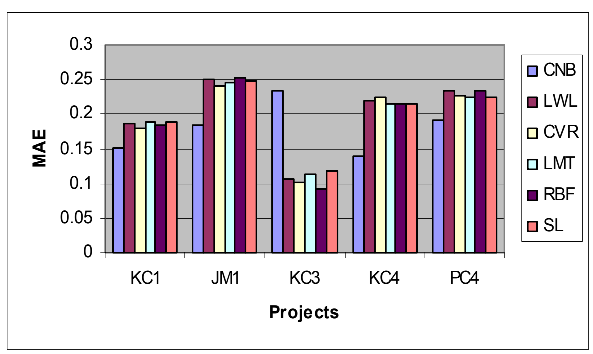

Fig. 2: *MAE* vs. Projects Graph with different prediction techniques

Prediction based on Root Mean Squared Error: The Fig. 3 derived from Table 6 shows that Classification via Regression (CVR) performed better for KC1 and JM1 data, LMT performed better for KC3 and KC4 data sets and SL performed better for KC4 and PC4 data sets . For KC4 data sets the LMT and SL have performed equally. But, LMT has performed better than SL for KC1, JM1 and KC3 datasets. Also, LMT has performed better than CVR for PC4 data sets and there is not much difference in results for KC1 and JM1 data sets. So, LMT is much better for maintenance severity Prediction based on *RMSE* results.

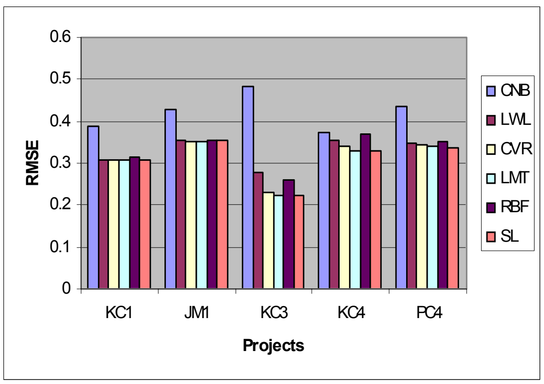

Fig. 3: *RMSE* vs. Projects Graph with different prediction techniques

CONCLUSION

We have compared different prediction models for predicting the maintenance urgency of different projects having modules with defects. We have seen that there in no particular predicting technique that performed the best for all the data sets based on *Accuracy*, Mean Absolute Error (*MAE*) and Root Mean Squared error (*RMSE*). However, Classification via Regression (CVR) and Logistic Model Trees (LMT) are the better methods that showed relatively better result consistency in predicting *Accuracy* percentage and *RMSE* value. CNB has shown better result consistency in predicting *MAE* value. But, logistic model Trees (LMT) and Complimentary Naïve Bayes (CNB) based Model provide a relatively better prediction consistency compared to other models and hence, can be used for the maintenance severity prediction of the software.

So, the predicted model can be used to automate the calculation of maintenance severity of defective modules .We can also prioritize that which module should be maintained first based on predicted maintenance severity value and this will reduce the amount of effort required to maintain that particular module. Hence, the productivity and ease of use of the software will be increased. In Future, the developed system can also be used for analysis and to evaluate the influence of different factors on the maintenance severity of different software project modules.

APPENDIX

Appendix has two tables: Table 7 and Table 8. The Table 7 shows the results of prediction algorithms on KC1, JM1 and KC3 projects datasets. The Table8 shows the results of prediction algorithms on KC4 and PC4 projects datasets. The name of all the 35 algorithms are BayesNet (BN) , Complement Naive Bayes (CNB), Naive Bayes (NB), Naive Bayes Multinomial (NBM), IB1, IBk, KStar, LWL, AdaBoostM1 (ABM1), Attribute Selected Classifier (ASC), Bagging, Decorate, Classification Via Regression (CVR), CVParameter Selection (CVPS), FilteredClassifier (FC), LogitBoost (LB), MultiBoostAB (MBAB), Ordinal Class Classifier (OCC), Raced Incremental LogitBoost (RILB), MultiClass Classifier (MCC), Random Committee (RC), HyperPipes (HP), VFI, J48, Decision Stump (DS), LMT, NBTree, RandomForest (RF), RandomTree (RT), REPTree, RBFNetwork (RBF), Logistic, Multilayer Perceptron (MP), Simple Logistic (SL), SMO. They all belong to the six categories Bayes, Function, Lazy, Meta, Miscellaneous and Trees.

Table 7: Accuracy and Errors shown by different models on predicting maintenance severity

Classification / Prediction Algorithm	KC1 Statistics after 10 fold Cross-Validation			JM1 Statistics after 10 fold Cross-Validation			KC3 Statistics after 10 fold Cross-Validation		
	Accuracy	*MAE*	*RMSE*	Accuracy	*MAE*	*RMSE*	Accuracy	*MAE*	*RMSE*
BN	69.9659	0.1904	0.3078	32.3977	0.2726	0.4692	86.2069	0.1198	0.2291
CNB	62.4573	0.1502	0.3875	53.8535	0.1846	0.4296	41.3793	0.2345	0.4842
NB	15.6997	0.336	0.5653	22.0742	0.3109	0.5471	48.2759	0.2069	0.4549
NBM	12.2867	0.3511	0.5922	20.5519	0.3183	0.5622	37.931	0.2483	0.4967
IB1	59.0444	0.1638	0.4047	46.5271	0.2139	0.4625	72.4138	0.1103	0.3322
IBk	58.0205	0.1698	0.4024	46.7174	0.214	0.4606	72.4138	0.144	0.311
KStar	54.6075	0.1844	0.4041	46.8126	0.2226	0.4262	65.5172	0.1351	0.3595
LWL	69.2833	0.1855	0.3085	54.6622	0.2513	0.3552	82.7586	0.106	0.2787
ABM1	69.9659	0.212	0.321	54.5195	0.2522	0.3557	86.2069	0.1287	0.255
ASC	70.6485	0.1857	0.3047	53.3302	0.2363	0.3738	86.2069	0.1006	0.2279
Bagging	69.6246	0.1836	0.3051	55.7088	0.2375	0.3488	86.2069	0.1101	0.2324
Decorate	62.7986	0.1794	0.3393	54.9477	0.2239	0.3573	72.4138	0.1275	0.2752
CVR	70.9898	0.1784	0.3064	56.5176	0.2416	0.3506	86.2069	0.1011	0.2317
CVPS	70.6485	0.1882	0.3048	54.5195	0.2544	0.3565	86.2069	0.1359	0.233
FC	70.6485	0.1857	0.3047	54.6622	0.2466	0.355	86.2069	0.1006	0.2279
LB	69.6246	0.1736	0.3082	55.471	0.244	0.3516	72.4138	0.1096	0.3195
MBAB	69.9659	0.212	0.321	54.5195	0.2522	0.3557	82.7586	0.0806	0.2675
OCC	64.8464	0.1915	0.3352	52.8069	0.2411	0.3674	72.4138	0.1418	0.3239
RILB	70.6485	0.1882	0.3048	54.6147	0.2423	0.3579	86.2069	0.1359	0.233
MCC	66.8942	0.3032	0.38	55.0428	0.3113	0.3896	72.4138	0.2972	0.3731
RC	63.8225	0.1773	0.3307	54.6147	0.2191	0.3579	75.8621	0.1269	0.3026
HP	20.1365	0.3017	0.385	16.6508	0.3196	0.3996	86.2069	0.1562	0.2727
VFI	14.6758	0.3124	0.4004	15.0333	0.3197	0.4005	44.8276	0.2272	0.3514
J48	62.116	0.18	0.367	48.2873	0.2269	0.4204	75.8621	0.1097	0.2982
DS	69.9659	0.1861	0.308	54.5195	0.2522	0.3557	86.2069	0.0949	0.2437
LMT	70.3072	0.1887	0.3076	55.3283	0.246	0.3524	86.2069	0.1127	0.2224
NBTree	68.942	0.1927	0.312	56.0419	0.2431	0.353	86.2069	0.1359	0.233
RF	67.9181	0.1664	0.3154	53.568	0.2233	0.3585	82.7586	0.1145	0.2779
RT	58.7031	0.1646	0.4031	45.0048	0.2202	0.4682	68.9655	0.1241	0.3523
REPTree	69.2833	0.1855	0.3184	55.3758	0.2369	0.3617	86.2069	0.1124	0.2286
RBF	66.8942	0.1833	0.3133	54.0913	0.2516	0.3563	79.3103	0.0914	0.2588
Logistic	65.529	0.1804	0.328	54.9001	0.2456	0.3532	68.9655	0.124	0.3516
MP	66.8942	0.175	0.32	55.0428	0.2461	0.3538	79.3103	0.1024	0.2571
SL	70.3072	0.1887	0.3076	54.9001	0.248	0.3533	86.2069	0.1172	0.2235
SMO	70.6485	0.2586	0.3443	54.6147	0.2732	0.365	86.2069	0.2294	0.3228

Table 7: Accuracy and Errors shown by different models on predicting maintenance severity

Classification / Prediction Algorithm	KC4 Statistics after 10 fold Cross-Validation			PC4 Statistics after 10 fold Cross-Validation		
	Accuracy	*MAE*	*RMSE*	Accuracy	*MAE*	*RMSE*
BN	51.6667	0.2372	0.3423	52.2472	0.1998	0.3782
CNB	65	0.14	0.3742	52.2472	0.191	0.437
NB	23.3333	0.281	0.4813	50.5618	0.1965	0.4289
NBM	65	0.1802	0.3719	46.0674	0.2168	0.4635
IB1	53.3333	0.1867	0.432	44.9438	0.2202	0.4693
IBk	53.3333	0.1995	0.4151	44.9438	0.2232	0.4623
KStar	51.6667	0.2105	0.3954	49.4382	0.2032	0.4302
LWL	55	0.2207	0.3548	53.9326	0.2329	0.348
ABM1	55	0.2527	0.3583	53.3708	0.3027	0.3862
ASC	48.3333	0.2397	0.3559	44.382	0.2423	0.4075
Bagging	63.3333	0.2186	0.3359	54.4944	0.2204	0.3385
Decorate	50	0.2264	0.3756	46.0674	0.2323	0.3738
CVR	58.3333	0.2255	0.3396	54.4944	0.2257	0.3425
CVPS	51.6667	0.2407	0.3426	44.9438	0.2585	0.3582
FC	51.6667	0.2333	0.3423	53.9326	0.2241	0.3462
LB	55	0.2125	0.3629	55.618	0.2194	0.353
MBAB	55	0.2527	0.3583	53.3708	0.3019	0.3856
OCC	55	0.2216	0.3635	46.0674	0.2265	0.3949
RILB	51.6667	0.2407	0.3426	44.9438	0.2585	0.3582
MCC	63.3333	0.3045	0.3815	55.0562	0.3084	0.3864
RC	50	0.1887	0.3822	49.4382	0.2283	0.368
HP	28.3333	0.2971	0.3838	34.8315	0.2648	0.3631
VFI	38.3333	0.2843	0.3769	53.3708	0.2584	0.3651
J48	55	0.219	0.3885	41.573	0.2448	0.4481
DS	55	0.2229	0.3557	53.3708	0.2366	0.3512
LMT	65	0.2145	0.3285	58.9888	0.2242	0.3395
NBTree	56.6667	0.2348	0.3448	49.4382	0.2296	0.3594
RF	55	0.2058	0.3626	52.809	0.2153	0.3524
RT	46.6667	0.2167	0.4637	45.5056	0.218	0.4669
REPTree	60	0.2236	0.3512	53.9326	0.2248	0.3576
RBF	53.3333	0.2147	0.3694	53.9326	0.2344	0.3514
Logistic	60	0.1934	0.3369	52.809	0.2222	0.3581
MP	60	0.2138	0.3307	50.5618	0.2167	0.3741
SL	65	0.2145	0.3285	57.3034	0.2251	0.3371
-SMO	53.3333	0.2667	0.3559	55.618	0.2509	0.3501

REFERENCES

1. Abrand, A. and H. Nguyenkim, 1991. Analysis of Maintenance Work Categories Through Measurement. Proceedings of IEEE Conference on Software Maintenance, Sorrento, Italy: IEEE, pp: 104-113.
2. Kung, H. and C. Hsu, 1998. Software Maintenance Life Cycle Model. Proceedings International Conference on Software Maintenance, IEEE 1998, Washington D.C.
3. Hsiang-Jui, Kung, 2004. Quantitative Method to Determine Software Maintenance Life Cycle. Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE, 2004.
4. Pigoski, T. M., 1997. Practical Software Maintenance. Wiley computer publishing, 1997.
5. Gibson, V. and J. Senn, 1989. System Structure and Software Maintenance Performance. Comm. of ACM, 27(3): 347-358.
6. Venkata, U.B., B. Challagulla, B. Bastani Farokh, Y. I-Ling, 2005. Empirical Assessment of machine Learning based Software Defect Prediction Techniques. Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05), IEEE 2005.
7. Khoshgoftaar, T. M., E. B. Allen, F. D. Ross, R. Munikoti, N. Goel, and A. Nandi, 1997. Predicting fault-prone modules with case-based reasoning. ISSRE-1997 the Eighth International Symposium on Software Engineering IEEE Computer Society, pp: 27-35.
8. Lee, Min-Gu and L. Jefferson Theresa, 2005. An Empirical Study of Software Maintenance of a Web-based Java Application. Proceedings of the 21st IEEE
9. International Conference on Software Maintenance (ICSM'05), IEEE-2005.
10. Thomas M. Pigoski and Lauren E. Nelson, 1994. Software Maintenance Metrics: A Case Study. Proceedings of IEEE Conference on Software Maintenance, IEEE (1994).
11. Ambros, Marco D' and Michle Lanza, 2006. Software Bugs and Evolution: A Visual Approach to uncover their relationship. Proceedings of IEEE Conference on Software Maintenance and Reegineering (CSMR' 06), IEEE (2006).
12. www.cs.waikato.ac.nz/~ml/weka/
13. Mall, Rajib, 2005. Fundamentals of software Engineering. Prentice Hall of India Publishers, pp: 10-20.
14. Stark, George E., 1996. Measurements for Managing Software Maintenance. IEEE computer Society (1996).
15. Khoshgoftaar, T.M., K. Gao and R. M. Szabo, 2001. An Application of Zero-Inflated Poisson Regression for Software Fault Prediction. Software Reliability Engineering, ISSRE 2001. Proceedings of 12th International Symposium on, 27-30 Nov. (2001), pp: 66 -73.
16. Munson, J. and T. Khoshgoftaar, 1990. Regression Modeling of Software Quality: An Empirical Investigation. Information and Software Technology, 32(2): 106 - 114.
17. Khoshgoftaar, T. M. and J. C. Munson, 1990. Predicting Software Development Errors using Complexity Metrics. IEEE Journal on Selected Areas in Communications, 8(2): 253 -261.
18. Menzies, T., K. Ammar, A. Nikora, and S. Stefano, 2003. How Simple is Software Defect Prediction? Journal of Empirical Software Engineering, October (2003).
19. Eman, K., S. Benlarbi, N. Goel and S. Rai, 2001. Comparing case-based reasoning classifiers for predicting high risk software components. Journal of Systems Software, 55(3): 301 – 310.
20. Fenton, N.E. and M. Neil, 1999. A critique of software defect prediction models. IEEE Trans. on Software Engineering, 25(5): 675 -689.
21. Hudepohl, J. P., S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. E. Mayrand, 1996. Software Metrics and Models on the Desktop. IEEE Software, 13(5): 56-60.
22. Khoshgoftaar, T. M., E. B. Allen, K. S. Kalaichelvan, and N. Goel, 1996. Early quality prediction: a case study in telecommunications. IEEE Software (1996), 13(1): 65-71.
23. Khoshgoftaar, T. M. and N. Seliya, 2002. Tree-based software quality estimation models for fault prediction. METRICS 2002, the Eighth IIIE Symposium on Software Metrics, pp: 203-214.
24. Seliya N., T. M. Khoshgoftaar, S. Zhong, 2005. Analyzing software quality with limited fault-proneness defect data. Ninth IEEE international Symposium, Oct 12-14, (2005).
25. Munson, J. C. and T. M. Khoshgoftaar, 1992. The detection of fault-prone programs. IEEE Transactions on Software Engineering, 18(5): 423-433.