Investigating the Effect of Software Metrics Aggregation on Software Fault Prediction

A Dissertation

Submitted for the partial fulfilment of the requirements for the award of the degree

of

Master of Technology

in

Computer Science and Engineering

Submitted By

Deepanshu Dixit

M.Tech CSE

Enrolment No. 16535009

Department of Computer Science and Engineering,

Indian Institute of Technology, Roorkee,

Roorkee- 247667, India.

May, 2018

ABSTRACT

In inter-releases software fault prediction, the data from the previous version of the software that is used for training the classifier might not always be of same granularity as that of the testing data. The same scenario may also happen in the cross project software fault prediction. So, one major issue in it can be the difference in granularity ,i.e., training and testing datasets may not have the metrics at the same level. Thus, there is a need to bring the metrics at the same level. In this work, eight different aggregation techniques are explored. In addition to Median and Summation aggregation techniques that have been used earlier in Software Fault Prediction, three other aggregation techniques ,i.e., Average Absolute Deviation (AAD), Median Absolute Deviation (MAD) and Interquartile Range (IQR) that have not been used in Software Fault Prediction so far are also explored in this work. Three novel aggregation techniques ,i.e., Average of Quarter Medians (QM_AVG), Median of Quarter Medians (QM_MED) and Sum of Quarter Medians (QM_SUM) are also explored in this work.

AUTHOR’S DECLARATION

I, hereby declare that the work presented in this dissertation ***Investigating the Effect of Software Metrics Aggregation on Software Fault Prediction*** towards the fulfilment of the requirements for the award of the degree of ***Master of Technology in Computer Science and Engineering***, submitted to the Department of Computer Science and Engineering, ***Indian Institute of Technology Roorkee, India***, is an authentic record of my own work carried out during May 2017 to May 2018, under the guidance of ***Dr. Sandeep Kumar, Assistant Professor***, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India.

The content presented in this dissertation has not been submitted by me for the award of any other degree of this or any other institute.

Date:

Place:

Deepanshu Dixit

M.Tech, C.S.E.,

Indian Institute of Technology, Roorkee

CERTIFICATE

This is to certify that the statement made by the author in the above declaration is correct to the best of my knowledge and belief.

Date:

Place:

Dr. Sandeep Kumar

Assistant Professor, C.S.E.,

Indian Institute of Technology, Roorkee

ACKNOWLEDGEMENTS

I would first like to thank my supervisor Dr. Sandeep Kumar, Assistant Professor, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, for his thoughtful encouragement and careful supervision during my research work. His enthusiasm for Software Engineering kept me constantly engaged with my research work and his personal generosity helped make my time at I.I.T. Roorkee enjoyable.

I sincerely thank my family and friends for directly or indirectly helping me and giving me moral support that motivated me during the course of the work. I am also thankful to all the staff members of the Department of Computer Science and Engineering for their support.

CONTENTS

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Software Fault Prediction is the mechanism to predict whether in a software the modules are going to be faulty or non faulty, before even applying the testing mechanism. In other words, Software Fault Prediction is a way to find the fault proneness of the software modules during the early stages of software development life cycle process. This prediction has a great role to play in improving the quality of the software as well as reducing the time and efforts needed in the testing phase of the development life cycle of the software. This chapter describes the basic terminologies and briefs about the Software Fault Prediction mechanism.

1.1 SOFTWARE FAULT PREDICTION

Now a days software are being used in almost every field and they play an important role in our lives. The software testing is an important as well as a costly task, both in terms of time and efforts. Maintaining the quality of the software is now a days of prime importance and so the testing phase is paid much more attention. As the software testing phase is a costly task, it is better to have an estimate about the fault proneness of the software modules before applying the testing efforts. This can heavily reduce the efforts required in testing the software modules. Software fault prediction mechanism predicts whether the software module is faulty or not before applying the testing mechanism. More testing efforts are made in a module which is predicted as faulty as compared to the one predicted as non faulty [1]. In many software systems like banking, financial systems, medical systems, satellite systems, etc., if any bug is left undetected then severe damages can be caused. Hence, testing is indeed very important phase in the development of such software systems [2].

1.2 NEED FOR SOFTWARE FAULT PREDICTION

Performing high end testing of all the modules is a costly task hence software fault prediction techniques help in predicting whether the modules are faulty or not. This saves time as all modules are not evenly faulty and now it is known which modules are more fault prone hence more focus can be given only to those modules. Software fault prediction techniques help in capturing the faulty modules even before the testing phase (during early phases of software development cycle). Many prediction models, which describe the relationship between different software metrics and software defects, have been proposed [3], but till now, no such metric can be used all-alone for correctly predicting the modules. Studies are still going on to find the best metric which would predict correctly the fault proneness of software modules.

1.3 SOFTWARE FAULT PREDICTION METHODOLOGY

Software Fault Prediction process involves two steps ,i.e., training and testing. A prediction model is developed using the data and metrics from previous versions of software during training phase and this model is used to predict the presence of faulty modules in the new versions of the software during testing phase.

The data from the previous version of the software (inter-release experiments) or from some other software belonging to the related domain (cross project software fault prediction) is used to train the prediction model. A suitable classifier is trained using this data. This trained prediction model is now fed with some new data over which testing mechanism is to be performed. In case of a binary classification of software fault prediction, the prediction model gives the output in terms of module being faulty or non faulty. In case of predicting the number of faults in software fault prediction, the prediction model predicts the number of faults that are likely to be present in that software module. Depending upon the fault proneness of a software module, the testing efforts are made to remove the faults in the software modules. Higher the fault proneness, more the efforts needed and lower the fault proneness, lesser are the efforts required in the testing phase. Hence, this prediction mechanism helps in determining the amount of efforts required in testing of a particular software module and reduces the unnecessary testing efforts and resources in testing the modules that are non faulty or likely to be non faulty.

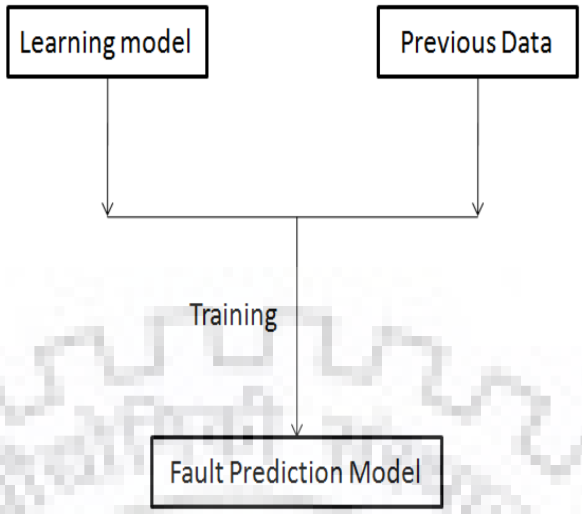

Figure 1.1: Training Phase in Software Fault Prediction

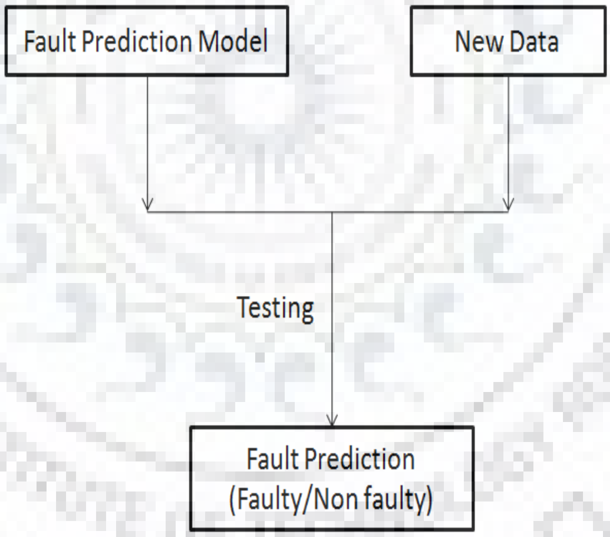

Figure 1.2: Testing Phase in Software Fault Prediction

At present there exists no software metric and no learning model that always performs accurately for all types of data sets. For varying data sets the efficiency shown by different learning models using different software metrics varies. The aim is to find software metrics that could perform well for most of the data sets in general.

1.4 BINARY CLASSIFICATION IN SOFTWARE FAULT PREDICTION

In software fault prediction, prediction is to be made about the module under consideration whether it is going to be faulty module or non faulty module, even when testing mechanism is not applied. The prediction outcome of this mechanism gives information about the fault proneness of the software module. Binary classification in software fault prediction means that either the module under consideration will be labeled as faulty or non faulty. There are only two outcomes possible for this type of prediction.

In binary classification of software fault prediction, if the faults are more than a particular threshold value then that module is labeled as faulty and if the faults are less than the particular threshold then that module is labeled as non faulty. Choosing the correct threshold is of prime importance in such prediction mechanism. If the threshold value is too high, most of the modules will be forced to be labeled as non faulty module, while if the value of the threshold is too low the most of the modules will be forced to be labeled as faulty modules. Hence, a balanced threshold values is required for a particular module under consideration to be labeled either as faulty or non faulty module.

1.5 PREDICTING THE NUMBER OF FAULTS IN SOFTWARE FAULT PREDICTION

In software fault prediction mechanism, the fault proneness of the module is predicted using some classifier. This fault proneness can be in terms of binary classification or in terms of the number of faults present in the module. Finding the number of faults in a module gives more accurate information about the fault proneness of the given module. It is better than just having the information whether a module is faulty or non faulty. Binary classification of fault proneness does not give the exact information about how less or more the module is fault prone. Thus, finding the number of faults in a software module rather than just finding it being faulty or non faulty is more helpful in reducing the testing efforts. On the other hand, finding number of faults is more complex and difficult as compared to finding whether a module is faulty or not.

1.6 INTER RELEASE SOFTWARE FAULT PREDICTION

In inter release software fault prediction, the previous version of software is used for the training of the prediction model. This trained model is then used for testing the later version of the same software. Thus the same software, but different releases of it are used for applying the software fault prediction mechanism. It is expected that the later release of a software will continue to have the characteristics of the previous version of the same software. Thus, the previous version of the software serves as a good training data and will train the classifier well. The prediction model then makes prediction on the later releases of that software.

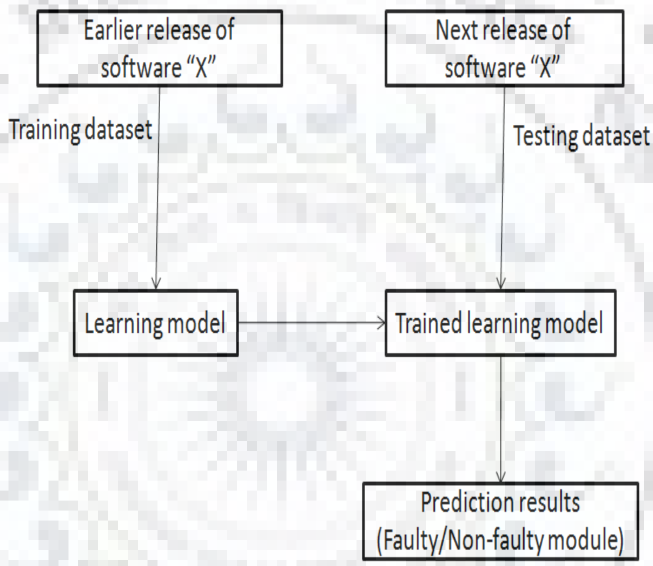

Figure 1.3: Inter-Release Software Fault Prediction

1.7 INTRA RELEASE SOFTWARE FAULT PREDICTION

In intra-release software fault prediction, the same dataset is used for training as well as for the testing purpose. K-fold cross validation experiment is performed in intra-release software fault prediction.The dataset is divided into K partitions. These partitions are known as the folds and the partition of the dataset is done in such a manner that the size of each fold is the same. (K-1) folds are used to train the classifier while the remaining 1 fold is used for the testing purpose. In order to avoid any biasing, the experiment is repeated K times. Each time a new fold is chosen as the testing set and rest (K-1) folds are used to train the classifier. The final predicted value is obtained by taking the average of all the values obtained in performing the experiments K times.

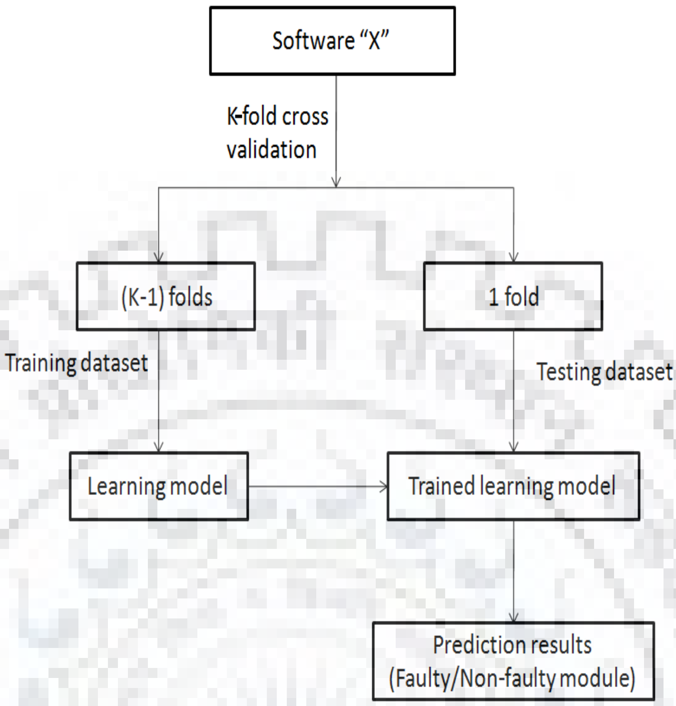

Figure 1.4: Intra-Release Software Fault Prediction

1.8 CROSS PROJECT SOFTWARE FAULT PREDICTION

Many times the data of the previous version of the software is not available, either due to unavailability of the data of the previous version of the software or the previous version of the software does not exist. In such cases a different software of same domain is chosen to train the classifier. Thus, the training and the testing datasets belong to different softwares. The software fault prediction mechanism conducted on different training-testing datasets is known as the cross project software fault prediction.

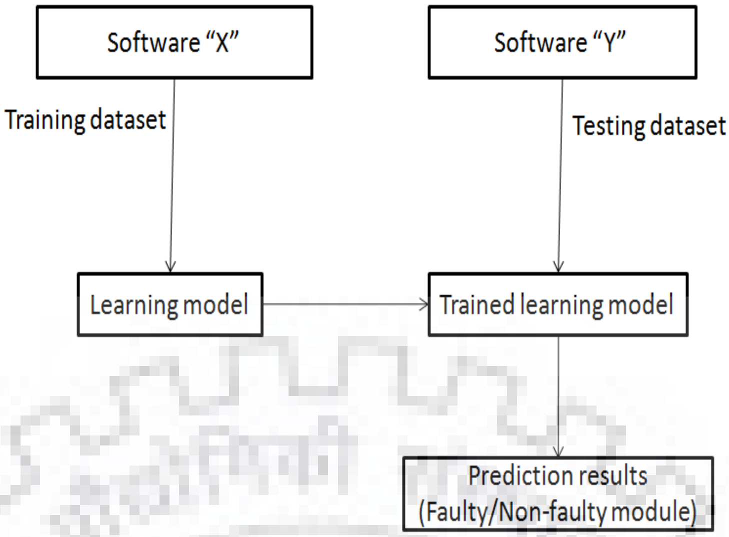

Figure 1.5: Cross Project Software Fault Prediction

1.9 SOFTWARE METRICS AGGREGATION

In cases of inter-releases software fault prediction, the data from the previous version of the software that is used for training the classifier might not always be of same granularity as that of the testing data, which can be a major issue. The same scenario may also happen in the cross project fault prediction. Thus, there is a need to bring the metrics at the same level. In this work, the software metrics available at the class and file level are aggregated to package level by using eight different aggregation techniques i.e AAD, MAD, IQR, MED, SUM, QM_AVG, QM_MED and QM_SUM .

1.10 NEED FOR SOFTWARE METRICS AGGREGATION

Following are some of the reasons why there can arise a need to aggregate the software metrics form one level to some other higher level for software fault prediction mechanism:

a) In inter-releases software fault prediction, the granularity level at which the metrics of the training dataset is available might not be same as that of the metrics of the testing dataset. Hence before applying fault prediction mechanism, it is required to bring the metrics of training and testing datasets at same level of granularity. Metrics aggregation can be helpful in such scenario.

b) The same scenario may happen in case of Cross project software fault prediction where the training and testing datasets are from different projects. Hence they might not have the metrics available at the same level of granularity. Before applying the fault prediction mechanism, the metrics of the training and testing datasets will have to be brought at the same level. Metrics aggregation will be helpful in doing so.

c) Generally the modules in a software program are small in size. The statistical measures like Lines of Code etc. which depend upon the size of a module show little variations. As the number of small modules is comparatively larger than the number of large modules, the classifiers become biased and they are not able to significantly distinguish between small defective and non defective modules. Thus the classifiers mistakenly predict small defective modules also as non defective modules because the number of non defective modules is more than the number of small defective modules. Software metric aggregation techniques help in aggregation the metrics of smaller modules to a large module [4].

d) There are certain metrics that can be calculated at a particular aggregated level only e.g. class cohesion and inheritance are meaningful only at the class level and not below it. If the customer reviews are needed then the complete aggregated software is needed for the customer to present the reviews and it cannot be done at any lower level. Thus metrics aggregation is needed in such cases to bring the metrics form a lower level to some higher level where it is meaningful for certain phenomena [5].

1.11 CONCLUSION

Software fault prediction is a mechanism to predict whether a module is going to be faulty or non faulty. The performance of the prediction model depends upon many factors such as the datasets used, classifiers used, performance evaluation measures used etc. At present there exists no software metric and no learning model that always performs accurately for all types of data sets. For varying data sets the efficiency shown by different learning models using different software metrics varies. The aim is to find software metrics that could perform well for most of the data sets in general. This prediction can be done in binary form or in the form of predicting the number of faults, either as inter-release or intra-release experiments.

CHAPTER 2

LITERATURE SURVEY

Several works have been done till now in predicting whether the software module is faulty or non faulty, using different classifiers on different datasets. The performances vary on using different classifiers on different datasets. Till now, there is no particular prediction model that uses a specific classifier on a specific set of metrics that performs well on every dataset in general. The aim is to build some prediction model that could be used as an universal model for every kind of dataset. A lot of work and empirical analysis has been done in the field of software fault prediction also , in analysing different metrics and classifiers on different datasets.

2.1 AGGREGATION TECHNIQUES IN SOFTWARE FAULT PREDICTION

Following are some of the works in which different aggregation techniques have been used in the field of software engineering to predict fault proneness of the modules.

Zhang et al. [6] addressed the problem of difference in granularity ,i.e., the difference in the levels at which software metrics are collected. They aggregated the data metrics from method level to file level. They analyzed eleven aggregation techniques on 255 open source projects. Experiments were conducted using ten-fold cross validation technique. In ten fold cross validation process, the entire dataset under consideration was partitioned into ten equal parts known as the folds. Nine out of the ten folds were used to train the classifier while the tenth fold was used for the testing purpose. This process was repeated several times and the final resultant value was obtained by taking average of all the experimental values. Four defect prediction models were dealt with: defect proneness model, in which random forest was used and all schemes gave best results; defect rank model, in which logistic regression was used and all schemes gave best results; defect count model, in which logistic regression was used and summation scheme for aggregation was found to be the best and effort aware model, in which again logistic regression was used and median technique of metric aggregation was found to give the best results among all used aggregation schemes.

Zimmermann et al. [7] worked on three releases of publicly available eclipse datasets and mapped the packages and classes to the number of bugs that were reported before and after the release. Post release bugs are the actual ones that matter for the users of the software program. They used version archives and bug tracking systems to find the failed modules in the system. The keywords like bug, fixed etc. were captured in the version archives to locate the bugs. They computed the metrics at method, class and file level and aggregated them to higher levels ,i.e., file and package level. The aggregation techniques used were average, total and maximum values of the metrics. Logistic regression was used as the machine learning technique. Binary classification of software fault prediction was dealt with. A module was considered faulty even if it contained a single bug and was considered non faulty if it contained no bug.

Herzig [8] used test execution metrics and studied their effectiveness in building the models for predicting pre-release defects and post-release defects. He conducted experiments and found that the test execution metrics give promising results in terms of precision and recall performance evaluation measures in predicting the pre-release and post-release defects in a software. Summation, median, mean and maximum value were used as the metric aggregation techniques in software fault prediction mechanism in this work.

Posnett et al. [5] used summation aggregation technique to aggregate the metrics from the file level to package level. AUC ROC (Area Under Curve Receiver Operating Characteristic) and AUC CE (Area Under Curve Cost Effectiveness) were used in this work as performance evaluation measures. The aggregation was performed for fault prediction process. In this work, the effect of changes in a particular phenomenon was studied at aggregated as well as at disaggregated level. There are certain process and phenomenon that are valid only at some particular level. Similarly, there are certain metrics that can be calculated at a particular aggregated level only e.g. class cohesion and inheritance are meaningful only at the class level and not below it. If the customer reviews are needed then the complete aggregated software is needed for the customer to present the reviews and it cannot be done at any lower level. According to this work, metrics aggregation is needed in such cases to bring the metrics form a lower level to some higher level where it is meaningful for certain phenomena.

Koru and Liu [4] used minimum, maximum, summation and average for the aggregation of metrics in software fault prediction in their work. Aggregation was done from method to class level. F-measure was used as the performance evaluation measure. Generally, the modules in a software program are small in size. The statistical measures like Lines of Code etc. which depend upon the size of a module show little variations. As the number of small modules is comparatively larger than the number of large modules, the classifiers become biased and they are not able to significantly distinguish between small defective and non defective modules. Thus the classifiers mistakenly predict small defective modules also as non defective modules because the number of non defective modules is more than the number of small defective modules. In this work, software metric aggregation techniques are considered helpful in aggregating the metrics of smaller modules to a larger module.

2.2 AGGREGATION TECHNIQUES IN OTHER FIELDS OF SOFTWARE ENGINEERING

According to Vasilescu et al. [9], the software metrics are generally collected at the micro level such as method, class and package level but in order to have a view from the macro level ,i.e., system level, these metrics have to be aggregated. There are mainly two categories of the aggregation techniques: traditional and econometrics aggregation techniques. Traditional techniques of aggregation consist of mean, median and summation techniques. Econometrics techniques of aggregation consist of Gini, Theil, Kolm, Atkinson and Hoover inequality indices. In their work, the traditional and econometrics aggregation techniques were studied to analyze the correlations amongst them. SLOC metric was aggregated from class to package level. They concluded that Gini, Theil, Atkinson and Hoover aggregation techniques show high correlation amongst them, correlation between mean and Kolm aggregation technique was very high, and median showed high correlation with the mean technique.

Serebrenik and van den Brand [10] were the first to apply a famous econometric measure of inequality, Theil index, in the field of software metric aggregation. There are several other techniques for aggregation of metrics from lower to higher level but have some or the other shortcomings in them. Mean technique of aggregation smoothens the values and does not give an insight of the large variations in the values. Gini coefficient has a shortcoming that it is not decomposable while on the other hand Theil index is decomposable.

According to Manet et al. [11], the software metrics are calculated individually for every software module and they do not give enough information from higher level perspective. Hence the software metrics need to be aggregated from lower to higher level to give enough information at the system level. Metrics such as SLOC, cyclomatic complexity, inheritance depth etc. was used for aggregation. Simple and weighted average technique of aggregation have shortcomings as they dilute the bad values and do not provide enough information about the extreme or the bad values present in the set. In this paper, Manet et al. gave an empirical model for continuous and weighted metric aggregation termed as Squale quality model which ensures that the computed metrics at higher level are grounded by concrete repeatable measures to give fairly good enough overview of the system quality.

Walter et al. [12] used mean, standard deviation, Gini index, Theil index, Atkinson index, Kolm index, Hoover index and mean logarithmic deviation in software quality model. The data was first normalised to a range of 0-1 before applying the aggregation process. It was analysed that mean value is not sufficient to represent all the metric values as an aggregated value. Mean of a set of value normalises the variations in the values present in that set.

Ivan et al. [13] used summation and product for metric aggregation in software quality model. Weights were used to develop an aggregate indicator to study the effect on quality of software modules. The importance of these weights and changes in the performances on using different techniques were also studied in this work.

Sanz-Rodriguez et al. [14] used weighted mean, the Choquet integral and multiple linear regression for the aggregation of metrics to analyze the effect of aggregation in selecting the reusable educational materials from repositories on the web. The different aggregation techniques were analysed to study the changes in the significance in determining the reusability.

Vasa et al. [15] applied Gini index as the aggregation technique to study the effect on the information the metrics give about the software system. In this work, Gini index was applied on several projects that were object oriented in nature, developed using Java and C# programming languages. Gini index is widely used statistic in the field of economics to analyse the wealth distribution.

2.3 CONCLUSION

Most of these available works present sum, mean, median, maximum, standard deviation, Gini index, Theil index, Atkinson index and Hoover index as the aggregation methods and only a few of them have used aggregation in software fault prediction. The effects of Sum and Median and three other aggregation techniques AAD, MAD and IQR are studied in our work. To the best of our knowledge, AAD ,MAD and IQR aggregation methods have not been explored so far for software fault prediction, but have been used in other fields [16], [17], [18], [19], [20], [21]. Three novel techniques ,i.e.,QM_AVG, QM_MED and QM_SUM that have not been used so far in any of the fields, are also explored in our work. These three novel techniques try to overcome the limitations of summation, median and average methods of aggregation.

CHAPTER 3

METHODOLOGY

In software metrics, there are various granularities such as method level, class level, file level, package level, etc. [7] [22]. In this work, the metrics in the dataset are aggregated from the class (or file level) to package level. Class to package level aggregation is done for all sixteen datasets of PROMISE data repository, one apache dataset ,i.e., lucene and four other publicly available eclipse projects ,i.e., eclipse JDT CORE, eclipse PDE UI, equinox framework and mylyn. File to package level aggregation is done for the three releases of eclipse dataset ,i.e., eclipse 2.0, eclipse 2.1 and eclipse 3.0.

3.1 APPROACH OF FAULT PREDICTION MECHANISM

Figure 3.1 shows the work flow of activities in the approach proposed in this paper. Following steps are followed in the proposed approach:

Step1: For all the classes (or files) that belong to the same package, the metric values are aggregated using an aggregation technique. Class to package level aggregation is done for all sixteen datasets of PROMISE data repository, one apache dataset ,i.e., lucene and four other publicly available eclipse projects ,i.e., eclipse JDT CORE, eclipse PDE UI, equinox framework and mylyn. File to package level aggregation is done for the three releases of eclipse dataset ,i.e., eclipse 2.0, eclipse 2.1 and eclipse 3.0.

Step2: Generally, in every software system, the number of faulty modules is lesser than the number of non faulty modules. This creates an imbalance in the dataset, having more number of instances with non faulty label as compared to instances with faulty label. The classifier training becomes biased when the training dataset is facing the problem of class imbalance, leading to

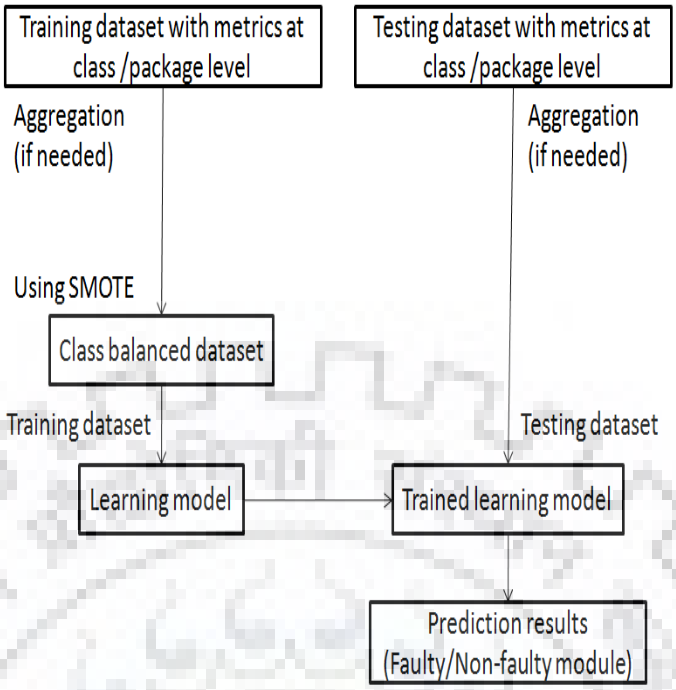

Figure 3.1: Approach of fault prediction mechanism used in this work.

inaccurate fault prediction. Inorder to remove class imbalance problem, SMOTE (Synthetic Minority Over-sampling Technique [23]) is used in this work. Synthetic instances are created using this technique to have almost equal number of non faulty and faulty modules in the dataset.

Step3: Earlier version of the dataset is used for training and the later version is used for testing in inter-release experiments. K-fold Cross validation is done in case of intra-release experiments. Sixteen releases of eight datasets form publicly available PROMISE data repository and three releases of publicly available Eclipse dataset is used in our work for inter-release experimentation. Three releases of Eclipse and five other publicly available datasets are used for intra-release experimentation.

Step4: Perform fault prediction mechanism using the training and testing datasets, generated in previous step. In case of binary classification of software fault prediction, five different classifiers are used: Decision Tree, Logistic Regression, Naive Bayes, Random Forest and Support Vector Machine. For predicting the number of faults, three different classifiers are used: Linear Regression,Decision Tree Regression and Multilayer Perceptron.

All the implementation is performed using R programming language version 3.4.0. It is a widely used programming language for data analysis and software fault predictions.

3.2 AGGREGATION PROCESS USED

The fundamentals of this work lies in *"what to aggregate" and "how to aggregate"*. Here, for each of the metric value, aggregation is done from class (or file) level to package level. All the metric values of the classes (or files) belonging to the same package are aggregated using one of the aggregation technique and brought to the package level. Figure 3.2 shows the aggregation process used in this work in detail, considering datasets of PROMISE data repository as an example for inter-release experiment.

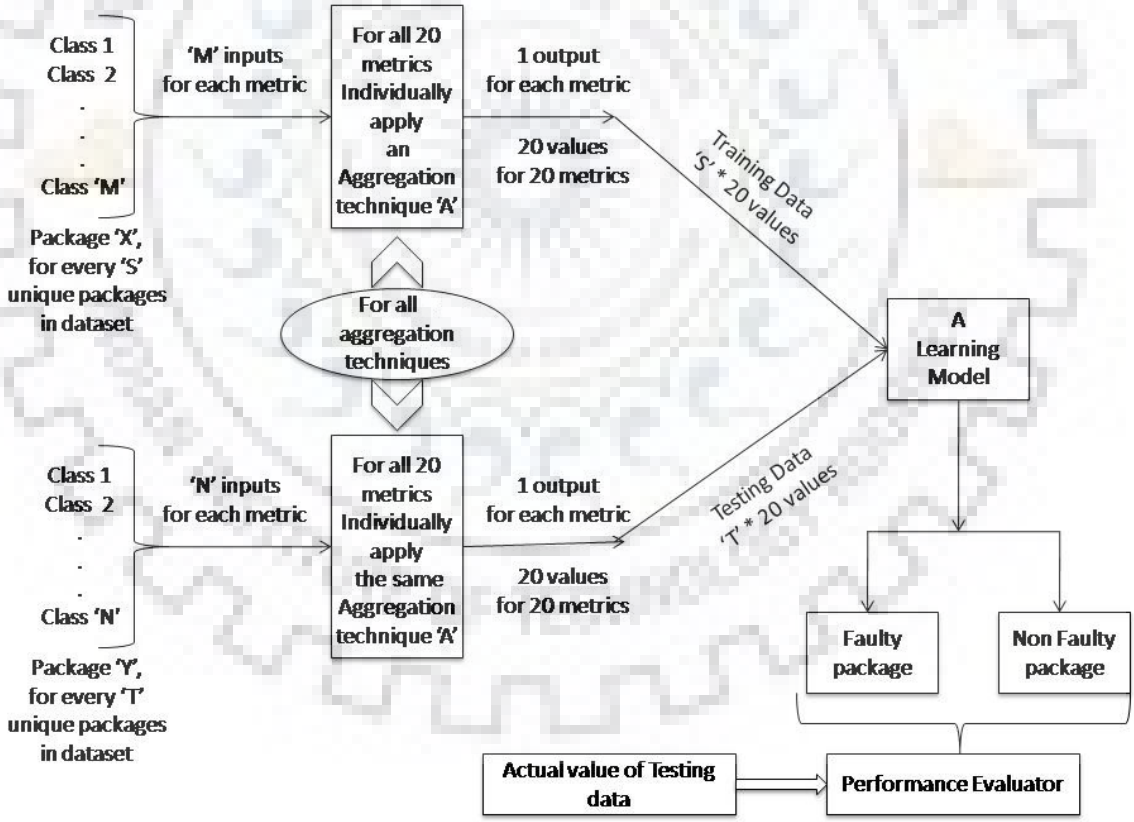

Figure 3.2: Details of Aggregation process used in this work.

For the PROMISE data repository, the aggregation is done from the class level to the package level. 20 metrics are used in the aggregation process. The classes which belong to the same package are aggregated together. For every distinct package in a dataset,for all the 20 metrics, the metrics values of all the classes are aggregated together. For all the 20 metrics, the aggregation technique is applied individually. For every distinct package there will be 20 aggregated values and if there are "S"(let) distinct packages in a certain dataset then there will be S*20 values. Let this be the training dataset, thus S*20 values will be used for training the learning model. Similar work is done for the testing dataset also. For every distinct package in a dataset,for all the 20 metrics, the metrics values of all the classes are aggregated together. For all the 20 metrics, the aggregation technique is applied individually. Let there be "T" distinct packages in the testing dataset, then there will be a total of T*20 vales for testing.

3.3 WITHOUT AGGREGATION METHOD OF SOFTWARE FAULT PREDICTION

In this work, eight different aggregation techniques are analysed to study their effects on the performance of software fault prediction. The metric values are aggregated from class (or file) level to package level and then the training and testing of the classifier is done. The performances of these aggregation techniques in fault prediction mechanism are also compared with the performance of the fault prediction mechanism when no aggregation technique is used. In "without aggregation method", the metric values are not aggregated. The original metric values of the dataset is used for training and testing the classifier.

3.4 CONCLUSION

The aggregation process is performed to bring together all the classes (or files) that belong to the same package. In this work eight different aggregation techniques ,i.e. AAD, IQR, MAD, MED, SUM, QM_AVG, QM_MED and QM_SUM are used to aggregate the classes (or files) which belong to the same package, turn by turn for analysing their effect on fault prediction mechanism. Inter-release and intra-release experiments are performed in binary classification software fault prediction and also in predicting the number of faults in software fault prediction. The performance of the aggregation techniques are compared with the performance of "without aggregation method" also.

CHAPTER 4

EMPIRICAL STUDY OF EXISTING AGGREGATION TECHNIQUES

Aggregation means to combine several values together into a single value. Aggregation of metric values means to combine several metric values together based on some criteria, and bring them from some lower level of granularity to some higher level of granularity. In this chapter, an empirical study of the five used existing aggregation techniques is done and their performances are compared with "without aggregation method" of software fault prediction. The five existing aggregation techniques used in this work are Average Absolute Deviation (AAD), Median Absolute Deviation (MAD), Interquartile Range (IQR), Median (MED) and Summation (SUM). The performances of these five techniques are also compared with each other. Inter-release and intra-release experiments are performed using binary classification and predicting number of faults in software fault prediction.

4.1 INTRODUCTION

Software Fault Prediction is the mechanism to predict whether in a software the modules are going to be faulty or non faulty, before even applying the testing mechanism. In cases of inter-releases software fault prediction, the data from the previous version of the software that is used for training the classifier might not always be of same granularity as that of the testing data, which can be a major issue. The same scenario may also happen in the cross project fault prediction. Thus, there is a need to bring the metrics at the same level before applying the prediction mechanism. Aggregation of metrics is helpful in such scenarios where the metric values are to be combined together to bring them from some lower level to some higher level of granularity. In this work, the software metrics available at the class and file level are aggregated to package level by using different aggregation techniques.

4.2 RELATED WORKS

The cases where the training and testing datasets are not having the metrics at the same level of granularity, aggregation of metrics is needed to bring them to the same level of granularity. Till now, several works have been done in the filed of software fault prediction but there has not been much work in using the aggregation techniques in the field of software fault prediction.

4.2.1 Aggregation used in the field of Software Fault Prediction

Following are the some of the works done in software fault prediction related to aggregation techniques.

Zhang et al. [6] addressed the problem of difference in granularity ,i.e., the difference in the levels at which software metrics are collected. They aggregated the data metrics from method level to file level. They analyzed eleven aggregation techniques on 255 open source projects. Experiments were conducted using ten-fold cross validation technique. Four defect prediction models were dealt with: defect proneness model, in which random forest was used and all schemes gave best results; defect rank model, in which logistic regression was used and all schemes gave best results; defect count model, in which logistic regression was used and summation scheme for aggregation was found to be the best and effort aware model, in which again logistic regression was used and median technique of metric aggregation was found to give the best results among all used aggregation schemes.

Zimmermann et al. [7] worked on three releases of publicly available eclipse datasets and mapped the packages and classes to the number of bugs that were reported before and after the release. Post release bugs are the actual ones that matter for the users of the software program. They used version archives and bug tracking systems to find the failed modules in the system. The keywords like bug, fixed etc. were captured in the version archives to locate the bugs. They computed the metrics at method, class and file level and aggregated them to higher levels ,i.e., file and package level. The aggregation techniques used were average, total and maximum values of the metrics. Logistic regression was used as the machine learning technique. A module was considered faulty even if it contained a single bug.

Herzig [8] used summation, median, mean and maximum value as the metric aggregation techniques in software fault prediction mechanism in his work. Posnett et al. [5] used summation while Koru and Liu [4] used minimum, maximum, summation and average for the aggregation of metrics in software fault prediction in their works.

4.2.2 Aggregation used in other fields

Aggregation of metrics have also been used in other fields of software engineering other than software fault prediction such as aggregation of metrics in software quality models.

According to Vasilescu et al. [9], the software metrics are generally collected at the micro level such as method, class and package level but in order to have a view from the macro level ,i.e., system level, these metrics have to be aggregated. There are mainly two categories of the aggregation techniques: traditional and econometrics aggregation techniques. Traditional techniques of aggregation consist of mean, median and summation techniques. Econometrics techniques of aggregation consist of Gini, Theil, Kolm, Atkinson and Hoover inequality indices. In their work, the traditional and econometrics aggregation techniques were studied to analyze the correlations amongst them. SLOC metric was aggregated from class to package level. They concluded that Gini, Theil, Atkinson and Hoover aggregation techniques show high correlation amongst them, correlation between mean and Kolm aggregation technique was very high, and median showed high correlation with the mean technique.

Serebrenik and van den Brand [10] were the first to apply a famous econometric measure of inequality, Theil index, in the field of software metric aggregation. There are several other techniques for aggregation of metrics from lower to higher level but have some or the other shortcomings in them. Mean technique of aggregation smoothens the values and does not give an insight of the large variations in the values. Gini coefficient has a shortcoming that it is not decomposable while on the other hand Theil index is decomposable.

According to Manet et al. [11], the software metrics are calculated individually for every software module and they do not give enough information from higher level perspective. Hence the software metrics need to be aggregated from lower to higher level to give enough information at the system level. Metrics such as SLOC, cyclomatic complexity, inheritance depth etc. was used for aggregation. Simple and weighted average technique of aggregation have shortcomings as they dilute the bad values and do not provide enough information about the extreme or the bad values present in the set. In this paper, Manet et al. gave an empirical model for continuous and weighted metric aggregation termed as Squale quality model which ensures that the computed metrics at higher level are grounded by concrete repeatable measures to give fairly good enough overview of the system quality.

Walter et al., [12] used mean, standard deviation, Gini index, Theil index, Atkinson index, Kolm index, Hoover index and mean logarithmic deviation while Ivan et al. [13] used summation and product for metric aggregation in software quality model. Sanz-Rodriguez et al. [14] used weighted mean, the Choquet integral and multiple linear regression for the aggregation of metrics to analyze the effect of aggregation in selecting the reusable educational materials from repositories on the web. Vasa et al. [15] applied Gini index as the aggregation technique to study the effect on the information the metrics give about the software system.

Most of these available works present sum, mean, median, maximum, standard deviation, Gini index, Theil index, Atkinson index and Hoover index as the aggregation methods and only a few of them have used aggregation in software fault prediction. However, to the best of our knowledge, AAD ,MAD and IQR aggregation methods have not been explored so far for software fault prediction, but have been used in other fields [16], [17], [18], [19], [20], [21].

4.3 AGGREGATION TECHNIQUES USED

In the inter-releases prediction and cross project fault prediction, the granularity of training and testing dataset metrics might not always be the same and when they are needed to be brought at the same level, then aggregation of the metrics can be used. In a particular package there exist several classes (or files). The metric values of all those classes (or files) which belong to the same package are combined together by using aggregation technique to give one value per metric for every package. It needs to be done for all the classes (or files) and packages. In this work, the existing aggregation techniques ,as listed in Table 4.1, are used for analyzing their effect on the software fault prediction performance:

a) Summation: It is one of the simplest way of finding the cumulative value of a given set of values. A module obtained after summation aggregation will contain larger metric value as compared to the smaller modules which are aggregated which is in accordance with the fact that as the size of the module increases the chances of it being faulty also increases. Summation has been used in many of the works related to software fault prediction e.g. [6], [7], [8], [5], [4].

$$SUM = \sum_{i=1}^{n} x_i \tag{4.1}$$

Where "n" is the number of values to be summed up.

S.No.	Aggregation Technique	Formula
1	Summation	$\sum_{i=1}^{n} X_i$
2	Median	$X_{(n+1)/2}$; n is odd $\frac{1}{2}(X_{(n)/2} + X_{(n+2)/2})$; otherwise
3	Average Absolute Deviation	$\frac{1}{n}\sum_{i=1}^{n} \lvert X_i - mean(X)\rvert$
4	Median Absolute Deviation	median($\lvert X_i - median(X)\rvert$)
5	Interquartile Range	$Q3-$Q1

Table 4.1: List of the existing Aggregation Techniques used.
n: number of classes, X_i:value of i^{th} module metric Q3:third quartile,Q1:first quartile.

b) Median: It is one of the mostly used measures of central tendency which gives an accumulative effect of the values present in a distribution. Median is calculated by finding the middle value in the sorted list of the given set of values which separates the first half from the second half of the given values. Median has been used in other works also (e.g. [6], [9]]). It is one of the traditional and easiest techniques to use.

$$MEDIAN = \begin{cases} x_{(n+1)/2}, & \text{if "n" is odd} \\ \frac{1}{2}(x_{n/2} + x_{(n+2)/2}), & \text{otherwise} \end{cases} \tag{4.2}$$

Where "n" is the number of values whose median is to be calculated.

c) Average Absolute Deviation: AAD depicts the average value of the absolute deviations of a given set of values $\{x_1, x_2, ...x_n\}$ from a central point. The central point is the average of the given set of values [16].

$$AAD = \frac{1}{n}\sum_{i=1}^{n} |x_i - A(X)| \tag{4.3}$$

Where A(X) is the average of the set of values $\{x_1, x_2, ...x_n\}$.

d) Median Average Deviation: MAD depicts the median value of the absolute deviations of a given set of values $\{x_1, x_2, ...x_n\}$ from a central point. The central point is the median of the given set of values [17], [18], [19].

$$MAD = Median(|x_i - Median(X)|) \tag{4.4}$$

Where Median(X) is the median of the set of values $\{x_1, x_2, ...x_n\}$.

e) Interquartile Range: IQR is a measure of statistical dispersion, which is the difference between the third and the first quartile, for a given set of values [19], [20], [21].

$$IQR = Q3 - Q1 \tag{4.5}$$

Where Q3 is the third quartile and Q1 is the first quartile.

4.4 DATASETS USED

Sixteen releases of datasets from the PROMISE data repository, three releases of publicly available eclipse dataset, one apache dataset and four other publicly available eclipse datasets have been used for experimentation [7], [24].

4.4.1 Inter-release experiments

The earlier release of a dataset is used for training purpose to predict the fault proneness for the later release that is used as testing dataset. There are eight pairs of training-testing datasets in our experiments. Table 4.2 provides the details of the used datasets.

S.No.	Training Dataset	Testing Dataset
1	ant 1.6	ant 1.7
2	camel 1.4	camel 1.6
3	ivy 1.4	ivy 2.0
4	poi 2.5	poi 3.0
5	synapse 1.1	synapse 1.2
6	velocity 1.5	velocity 1.6
7	xalan 2.5	xalan 2.6
8	xerces 1.3	xerces 1.4
9	eclipse 2.0	eclipse 2.1
10	eclipse 2.0	eclipse 3.0
11	eclipse 2.1	eclipse 3.0

Table 4.2: Training-Testing datasets used for Inter-release experiments.

4.4.2 Intra-release experiments

Table 4.3 shows the list of datasets used for performing the intra-release experiment. 10 fold cross validation techniques is used . The datasets is partitioned into 10 equal parts called folds, each fold having almost equal number of faulty and non faulty instances. Thus each fold is free from class imbalance problem. 9 out of 10 folds are used to train the classifier and testing is done on the 10th fold. This is repeated for ten times, making every fold as the testing data once.

S.No.	Dataset
1	eclipse JDT CORE
2	eclipse PDE UI
3	equinox framework
4	lucene
5	mylyn
6	eclipse 2.0
7	eclipse 2.1
8	eclipse 3.0

Table 4.3: Datasets used for Intra-release experiments.

4.5 BINARY CLASSIFICATION IN SOFTWARE FAULT PREDICTION

Binary classification in software fault prediction means that either the module under consideration will be labeled as faulty or non faulty. There are only two labels possible for prediction. In binary classification of fault prediction, if in a package even a single faulty class (or file) is present then that package is declared to be faulty otherwise non faulty [6], [7], [25].

4.5.1 Machine Learning Techniques used

Five machine learning techniques used are naive bayes (Yang et al., 2017), (Turhan et al., 2013), logistic regression (Arar and Ayan, 2016), (Zhao et al., 2017), support vector machine (Erturk and Sezer, 2015), decision tree (Ghotra et al., 2015) and random forest (Kamei and Shihab, 2016).

4.5.2 Performance Evaluation Measures used

In binary classification of fault prediction, if in a package, even a single faulty class (or file) is present then that package is declared to be faulty otherwise non faulty [26], [7], [25]. This concept is used for calculation of values of performance measures. Four different performance evaluation measures have been used as discussed below:

Accuracy: It denotes the percentage of correctly classified instances to the total number of instances.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} * 100 \quad (4.6)$$

Precision: It denotes the number of correctly classified faulty instances amongst the total number of instances classified as faulty.

$$Precision = \frac{TP}{TP+FP} \quad (4.7)$$

Recall: It denotes the number of correctly classified faulty instances amongst the total number of instances which are faulty.

$$Recall = \frac{TP}{TP+FN} \quad (4.8)$$

F-measure: It denotes the harmonic mean of the precision and recall values.

$$F-measure = \frac{2 * precision * recall}{precision + recall} \quad (4.9)$$

Where TP represents True Positive, FP represents False Positive, TN represents True Negative and FN represents False Negative.

4.6 NUMBER OF FAULTS IN SOFTWARE FAULT PREDICTION

In software fault prediction mechanism, the fault proneness of the module is predicted using some classifier. This fault proneness can be in terms of binary classification or in terms of the number of faults present in the module. Finding the number of faults in a module gives more accurate information about the fault proneness of the given module. It is better than just having the information whether a module is faulty or non faulty. Binary classification of fault proneness does not give the exact information about how less or more the module is fault prone.

4.6.1 Machine Learning Techniques used

Three machine learning techniques are used in the experimentations for predicting the number of faults in software fault prediction. These techniques are linear regression , multilayer perceptron and decision tree regression [1], [27], [28], [29].

4.6.2 Performance Evaluation Measures used

Following are the performance evaluation measures used in finding the number of faults in software fault prediction:

Average Absolute Error: It calculates the difference in the predicted and actual values and takes the average value considering all the instances. Its value ranges from 0 to 1. Lower the AAE better is the prediction.

$$AAE = \sum_{i=1}^{n} |X_i - Y_i| \tag{4.10}$$

Here n is the number of instances, X_i is the predicted value and Y_i is the actual value of an instance.

Average Relative Error: It calculates the ratio of the difference in the predicted and actual values to the actual value of an instance and then finds the average value for all the instances. Its value ranges from 0 to 1. Lower the ARE better is the prediction.

$$ARE = \sum_{i=1}^{n} (|X_i - Y_i|/Y_i + 1) \tag{4.11}$$

Here n is the number of instances, X_i is the predicted value and Y_i is the actual value of an instance. Sometimes the value of Y_i can be 0, making the fraction undefined. In order to avoid such situations an additional 1 is added in the denominator value [30].

Prediction at level 'l': It calculates the number of predictions having the predicted value within l% of the actual value. It calculates the number of predictions which have the ARE value under a certain predefined threshold value, generally taken to be 30%. Thus it calculates the percentage of the number of predictions whose ARE value is lesser than or equal to 0.3 [31].

$$Pred(l) = k/n \tag{4.12}$$

Here n is the total number of modules while k is the number of those modules which have the predicted value less than or equal to 'l'.

Measure of Completeness: It depicts the ratio of the number of faults predicted to the actual number of faults present in the overall modules. It is a measure to find how complete a model is in finding the number of faults as compared to the actual number of faults present.

$$MOC = \frac{Predicted\ number\ of\ faults}{Actual\ number\ of\ faults\ present} \tag{4.13}$$

4.7 EXPERIMENTAL RESULTS AND ANALYSIS

Table 4.4- Table 4.13 show the experimental results obtained for binary classification of software fault prediction. Five classifiers used are Decision Tree, Logistic Regression, Naive Bayes, Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure. Following observations can be made from these tables.

4.7.1 Inter-release Binary Classification

For Promise datasets, it can be observed from Table 4.4- Table 4.13 that AAD performs the best for Naive Bayes and Random Forest classifiers while Summation performs the best for Decision Tree and Logistic Regression classifiers, for all four performance evaluation measures used ,i.e., Accuracy, Precision, Recall and F-measure.

For Eclipse datasets, it can be observed from Table 4.4- Table 4.13 that none of the aggregation technique could outperform "without aggregation method" in terms of Accuracy. Summation gives the best results for all five classifiers used in terms of Precision. Median gives the best results in terms of Recall for Decision Tree, Logistic Regression and Naive Bayes while Summation gives the best results for Logistic Regression, Random Forest and Support Vector Machine classifiers.

Thus, AAD and Summation outperform other aggregation techniques for inter-release binary classification of software fault prediction for above mentioned scenarios.

Table 4.4: Performance of Decision Tree in terms of Accuracy % and Precision.

	Dataset	Accuracy %						Precision					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	75.168	67.164	62.687	49.254	52.239	64.179	0.453	0.636	0.625	0.452	0.491	0.578
	camel1.4-camel1.6	77.927	81.6	84.8	83.2	84	78.4	0.429	0.649	0.727	0.724	0.667	0.585
	ivy1.4-ivy2.0	82.67	59.615	61.538	55.769	63.462	73.077	0.2	0.417	0.462	0.389	0.5	0.667
	poi-2.5-poi3.0	41.403	80	90	85	75	85	0.612	0.882	1	1	0.929	1
	synapse1.1-synapse1.2	69.531	54.545	54.545	63.636	69.697	63.636	0.557	0.611	0.625	0.889	0.8	0.769
	velocity1.5-velocity1.6	57.205	88	80	84	80	76	0.429	0.833	0.812	0.824	0.812	0.737
	xalan2.5-xalan2.6	57.853	83.333	80.952	80.952	78.571	85.714	0.541	0.919	0.917	0.917	0.914	0.921
	xerces1.3-xerces1.4	39.456	68.421	76.316	76.316	68.421	78.947	0.872	1	1	1	1	1
	eclipse2.0-eclipse2.1	80.325	70.492	65.369	64.754	57.992	69.672	0.247	0.639	0.568	0.567	0.506	0.627
	eclipse2.0-eclipse3.0	78.524	67.258	64.939	64.256	56.889	68.486	0.312	0.665	0.6	0.6	0.528	0.674
	eclipse2.1-eclipse3.0	79.911	66.166	62.619	64.802	59.209	61.528	0.291	0.65	0.681	0.664	0.578	0.828
Intra	eclipse JDT CORE	94.861	97.157	94.412	94.412	97.157	96.569	0.981	0.971	0.933	0.958	0.98	0.962
	eclipse PDE UI	92.385	94	77	86.333	69.667	84.667	0.967	0.983	0.735	0.94	0.942	0.966
	equinox framework	81.067	93.833	92.667	87.667	96.833	97.667	0.759	0.971	0.975	0.933	0.971	1
	lucene	95.585	97.5	95.667	95.833	94	99.167	0.971	0.971	0.986	0.943	0.944	0.986
	mylyn	93.435	91.003	83.882	85.725	81.07	77.046	0.966	0.984	0.85	0.941	0.784	0.852
	eclipse2.0	92.939	76.121	74.871	77.076	73.371	82.72	0.973	0.815	0.746	0.796	0.74	0.814
	eclipse2.1	94.198	71.16	71.277	76.706	71.489	68.342	0.979	0.684	0.828	0.781	0.696	0.775
	eclipse3.0	91.49	71.932	67.862	69.763	68.145	74.047	0.982	0.737	0.619	0.647	0.678	0.711

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.5: Performance of Decision Tree in terms of Recall and F-measure.

	Dataset	Recall						F-measure					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.554	0.677	0.484	0.452	0.839	0.839	0.499	0.656	0.545	0.452	0.619	0.684
	camel1.4-camel1.6	0.404	0.706	0.706	0.618	0.824	0.706	0.416	0.676	0.716	0.667	0.737	0.64
	ivy1.4-ivy2.0	0.175	0.263	0.316	0.368	0.421	0.526	0.187	0.323	0.375	0.378	0.457	0.588
	poi-2.5-poi3.0	0.214	0.882	0.882	0.824	0.765	0.824	0.317	0.882	0.938	0.903	0.839	0.903
	synapse1.1-synapse1.2	0.453	0.579	0.526	0.421	0.632	0.526	0.5	0.595	0.571	0.571	0.706	0.625
	velocity1.5-velocity1.6	0.769	1	0.867	0.933	0.867	0.933	0.55	0.909	0.839	0.875	0.839	0.824
	xalan2.5-xalan2.6	0.611	0.895	0.868	0.868	0.842	0.921	0.574	0.907	0.892	0.892	0.877	0.921
	xerces1.3-xerces1.4	0.217	0.613	0.71	0.71	0.613	0.742	0.348	0.76	0.83	0.83	0.76	0.852
	eclipse2.0-eclipse2.1	0.399	0.713	0.804	0.751	0.804	0.722	0.305	0.674	0.665	0.646	0.621	0.671
	eclipse2.0-eclipse3.0	0.376	0.606	0.749	0.708	0.749	0.633	0.341	0.634	0.667	0.65	0.619	0.653
	eclipse2.1-eclipse3.0	0.249	0.601	0.379	0.501	0.475	0.224	0.269	0.624	0.487	0.571	0.522	0.353
Intra	eclipse JDT CORE	0.919	0.967	0.967	0.933	0.947	0.967	0.948	0.963	0.942	0.937	0.96	0.957
	eclipse PDE UI	0.884	0.9	0.86	0.786	0.423	0.726	0.923	0.933	0.786	0.852	0.562	0.821
	equinox framework	0.941	0.912	0.892	0.838	0.975	0.958	0.837	0.935	0.923	0.878	0.971	0.977
	lucene	0.941	0.983	0.933	0.983	0.95	1	0.956	0.976	0.956	0.96	0.943	0.992
	mylyn	0.903	0.841	0.845	0.771	0.873	0.634	0.933	0.905	0.843	0.845	0.825	0.714
	eclipse2.0	0.885	0.669	0.748	0.702	0.721	0.834	0.927	0.728	0.742	0.743	0.719	0.821
	eclipse2.1	0.908	0.708	0.499	0.699	0.728	0.47	0.942	0.687	0.608	0.724	0.693	0.576
	eclipse3.0	0.848	0.633	0.863	0.827	0.645	0.778	0.91	0.68	0.718	0.724	0.659	0.738

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.6: Performance of Logistic Regression in terms of Accuracy % and Precision.

	Dataset	Accuracy %						Precision					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	73.154	50.746	53.731	55.224	47.761	65.672	0.432	0.462	0.5	0.517	0.45	0.667
	camel1.4-camel1.6	60.622	85.6	80.8	88.8	72.8	80.8	0.253	0.69	0.632	0.812	0.5	0.614
	ivy1.4-ivy2.0	77.273	73.077	57.692	61.538	55.769	73.077	0.065	0.619	0.421	0.476	0.429	0.727
	poi-2.5-poi3.0	66.29	80	50	55	90	45	0.758	1	1	1	1	1
	synapse1.1-synapse1.2	62.891	57.576	39.394	39.394	66.667	75.758	0.455	0.667	0.462	0.462	0.75	0.867
	velocity1.5-velocity1.6	61.135	60	68	64	76	80	0.456	0.778	0.889	0.75	0.8	0.857
	xalan2.5-xalan2.6	56.384	69.048	61.905	78.571	54.762	85.714	0.537	0.931	0.958	0.939	0.913	0.9
	xerces1.3-xerces1.4	47.619	60.526	63.158	50	63.158	44.737	0.901	1	1	0.929	1	0.917
	eclipse2.0-eclipse2.1	75.228	67.418	63.934	65.164	61.68	70.082	0.24	0.632	0.594	0.597	0.541	0.668
	eclipse2.0-eclipse3.0	75.767	64.529	62.892	63.165	61.937	70.668	0.32	0.648	0.643	0.625	0.578	0.776
	eclipse2.1-eclipse3.0	75.333	63.029	61.937	61.255	60.982	68.895	0.316	0.662	0.665	0.651	0.635	0.798
Intra	eclipse JDT CORE	77.542	98.824	86.078	89.412	73.725	99.412	0.848	0.971	0.883	0.933	0.75	0.983
	eclipse PDE UI	69.061	71.667	77	75.667	70	84.667	0.754	0.746	0.79	0.806	0.76	0.904
	equinox framework	71.527	89.667	94.5	88.833	94.667	93	0.75	0.921	1	0.938	0.969	0.983
	lucene	66.775	78	78.167	81	83	93.333	0.764	0.821	0.85	0.872	0.879	0.952
	mylyn	68.761	72.629	75.786	71.24	65.61	73.293	0.775	0.787	0.818	0.801	0.744	0.805
	eclipse2.0	69.256	68.674	66.667	67.326	65.621	75.265	0.792	0.73	0.705	0.702	0.632	0.798
	eclipse2.1	66.441	68.351	62.558	64.472	62.307	70.234	0.768	0.738	0.678	0.761	0.693	0.773
	eclipse3.0	66.518	66.106	63.753	64.019	63.211	72.42	0.771	0.692	0.682	0.673	0.645	0.797

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.7: Performance of Logistic Regression in terms of Recall and F-measure.

	Dataset	Recall						F-measure					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.651	0.387	0.548	0.484	0.581	0.516	0.519	0.421	0.523	0.5	0.507	0.582
	camel1.4-camel1.6	0.521	0.853	0.706	0.765	0.706	0.794	0.34	0.763	0.667	0.788	0.585	0.692
	ivy1.4-ivy2.0	0.075	0.684	0.421	0.526	0.632	0.421	0.07	0.65	0.421	0.5	0.511	0.533
	poi-2.5-poi3.0	0.69	0.765	0.412	0.471	0.882	0.353	0.723	0.867	0.583	0.64	0.938	0.522
	synapse1.1-synapse1.2	0.523	0.526	0.316	0.316	0.632	0.684	0.486	0.588	0.375	0.375	0.686	0.765
	velocity1.5-velocity1.6	0.731	0.467	0.533	0.6	0.8	0.8	0.562	0.583	0.667	0.667	0.8	0.828
	xalan2.5-xalan2.6	0.438	0.711	0.605	0.816	0.553	0.947	0.483	0.806	0.742	0.873	0.689	0.923
	xerces1.3-xerces1.4	0.332	0.516	0.548	0.419	0.548	0.355	0.485	0.681	0.708	0.578	0.708	0.512
	eclipse2.0-eclipse2.1	0.594	0.574	0.498	0.574	0.699	0.598	0.342	0.602	0.542	0.585	0.61	0.631
	eclipse2.0-eclipse3.0	0.568	0.531	0.466	0.531	0.688	0.525	0.409	0.583	0.541	0.574	0.628	0.626
	eclipse2.1-eclipse3.0	0.573	0.429	0.376	0.37	0.391	0.449	0.408	0.52	0.48	0.472	0.484	0.575
Intra	eclipse JDT CORE	0.681	1	0.867	0.867	0.747	1	0.754	0.983	0.859	0.882	0.73	0.991
	eclipse PDE UI	0.607	0.706	0.78	0.737	0.657	0.791	0.67	0.714	0.773	0.749	0.68	0.829
	equinox framework	0.698	0.879	0.9	0.842	0.929	0.892	0.716	0.89	0.942	0.884	0.946	0.931
	lucene	0.496	0.717	0.717	0.733	0.8	0.917	0.601	0.752	0.763	0.79	0.828	0.932
	mylyn	0.548	0.666	0.698	0.592	0.51	0.623	0.641	0.712	0.749	0.674	0.599	0.694
	eclipse2.0	0.529	0.575	0.538	0.575	0.703	0.653	0.634	0.632	0.607	0.63	0.662	0.714
	eclipse2.1	0.503	0.483	0.379	0.359	0.369	0.507	0.607	0.578	0.476	0.463	0.467	0.605
	eclipse3.0	0.487	0.514	0.464	0.486	0.53	0.561	0.596	0.588	0.545	0.562	0.575	0.655

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.8: Performance of Naive Bayes in terms of Accuracy % and Precision.

	Dataset	Accuracy %						Precision					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	77.718	61.194	49.254	47.761	46.269	67.164	0.5	0.576	0.474	0.466	0.459	0.846
	camel1.4-camel1.6	73.575	84	82.4	38.4	53.6	78.4	0.321	0.733	0.7	0.295	0.312	0.733
	ivy1.4-ivy2.0	82.955	76.923	69.231	40.385	40.385	78.846	0.321	0.64	0.588	0.342	0.333	0.9
	poi-2.5-poi3.0	47.964	70	85	70	45	60	0.823	1	1	1	0.875	1
	synapse1.1-synapse1.2	66.406	69.697	54.545	54.545	54.545	69.697	0.5	0.909	0.6	0.577	0.577	0.909
	velocity1.5-velocity1.6	67.686	88	88	76	76	84	0.534	0.833	0.833	0.846	0.765	0.867
	xalan2.5-xalan2.6	61.921	76.19	66.667	50	30.952	76.19	0.708	0.912	0.9	0.87	0.846	0.967
	xerces1.3-xerces1.4	40.476	84.211	84.211	84.211	71.053	57.895	0.958	1	1	0.963	1	1
	eclipse2.0-eclipse2.1	85.028	64.754	65.164	64.139	53.689	67.623	0.312	0.667	0.651	0.576	0.478	0.726
	eclipse2.0-eclipse3.0	83.65	60.437	61.801	63.574	55.662	65.075	0.427	0.675	0.668	0.612	0.516	0.809
	eclipse2.1-eclipse3.0	84.32	61.392	62.892	59.618	54.161	64.256	0.446	0.683	0.654	0.634	0.506	0.829
Intra	eclipse JDT CORE	63.864	88.824	83.725	68.824	62.451	89.412	0.865	0.955	0.842	0.967	0.579	1
	eclipse PDE UI	61.362	62.667	59.333	66	55	69	0.745	0.767	0.558	0.86	0.717	0.842
	equinox framework	66.484	87.167	88.667	83	89	89.5	0.829	0.893	0.937	0.838	0.939	0.975
	lucene	58.918	67.667	77.167	62.667	61.333	83.833	0.743	0.866	0.865	0.582	0.573	0.899
	mylyn	60.845	69.885	66.267	64.81	67.554	66.551	0.785	0.735	0.766	0.78	0.691	0.848
	eclipse2.0	62.316	60.068	59.962	65.477	60.205	64.47	0.833	0.704	0.674	0.682	0.557	0.811
	eclipse2.1	58.632	67.636	64.156	59.619	53.139	63.26	0.819	0.703	0.658	0.709	0.5	0.723
	eclipse3.0	59.802	59.332	59.819	62.825	54.058	66.418	0.819	0.678	0.696	0.68	0.513	0.827

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.9: Performance of Naive Bayes in terms of Recall and F-measure.

	Dataset	Recall						F-measure					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.59	0.613	0.871	0.871	0.903	0.355	0.541	0.594	0.614	0.607	0.609	0.5
	camel1.4-camel1.6	0.319	0.647	0.618	0.912	0.588	0.324	0.32	0.688	0.656	0.446	0.408	0.449
	ivy1.4-ivy2.0	0.45	0.842	0.526	0.684	0.632	0.474	0.375	0.727	0.556	0.456	0.436	0.621
	poi-2.5-poi3.0	0.231	0.647	0.824	0.647	0.412	0.529	0.361	0.786	0.903	0.786	0.56	0.692
	synapse1.1-synapse1.2	0.593	0.526	0.632	0.789	0.789	0.526	0.543	0.667	0.615	0.667	0.667	0.667
	velocity1.5-velocity1.6	0.397	1	1	0.733	0.867	0.867	0.456	0.909	0.909	0.786	0.812	0.867
	xalan2.5-xalan2.6	0.307	0.816	0.711	0.526	0.289	0.763	0.428	0.861	0.794	0.656	0.431	0.853
	xerces1.3-xerces1.4	0.208	0.806	0.806	0.839	0.645	0.484	0.342	0.893	0.893	0.897	0.784	0.652
	eclipse2.0-eclipse2.1	0.317	0.354	0.402	0.617	0.885	0.392	0.315	0.462	0.497	0.596	0.621	0.509
	eclipse2.0-eclipse3.0	0.305	0.297	0.364	0.603	0.851	0.332	0.356	0.413	0.472	0.608	0.642	0.471
	eclipse2.1-eclipse3.0	0.247	0.327	0.44	0.324	0.834	0.297	0.318	0.442	0.526	0.429	0.63	0.438
Intra	eclipse JDT CORE	0.341	0.813	0.847	0.393	0.827	0.78	0.488	0.856	0.831	0.472	0.676	0.829
	eclipse PDE UI	0.389	0.389	0.96	0.437	0.169	0.457	0.51	0.497	0.703	0.527	0.259	0.569
	equinox framework	0.467	0.846	0.85	0.85	0.85	0.821	0.591	0.865	0.878	0.835	0.888	0.888
	lucene	0.28	0.483	0.667	1	0.967	0.767	0.403	0.593	0.726	0.734	0.718	0.821
	mylyn	0.32	0.655	0.532	0.45	0.692	0.424	0.454	0.691	0.619	0.562	0.688	0.555
	eclipse2.0	0.316	0.34	0.341	0.564	0.901	0.345	0.458	0.45	0.449	0.608	0.686	0.481
	eclipse2.1	0.254	0.507	0.484	0.457	0.862	0.325	0.387	0.582	0.547	0.492	0.624	0.442
	eclipse3.0	0.27	0.283	0.281	0.431	0.86	0.372	0.405	0.395	0.396	0.523	0.64	0.51

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

4.7.2 Intra-release Binary Classification

For Eclipse datasets, it can be observed from Table 4.4- Table 4.13 that Summation gives the best results in general, for Logistic Regression and Random Forest classifiers, in terms of all the four performance measures used.

For remaining four Eclipse and one Apache datasets, it can be observed from Table 4.4- Table 4.13 that Summation gives the best results in general, for all the five used classifiers in terms of Accuracy and Precision. Summation also gives the best results in terms of Recall and F-measure for Decision Tree and Logistic Regression classifiers. IQR gives the best results in general, in terms of Recall and F-measure for Naive Bayes, Random Forest and Support Vector Machine classifiers.

Thus, Summation outperforms all other techniques for intra-release binary classification of software fault prediction for above mentioned scenarios.

Table 4.10: Performance of Random Forest in terms of Accuracy % and Precision.

	Dataset	Accuracy %						Precision					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	77.852	59.701	67.164	59.701	50.746	67.164	0.503	0.548	0.615	0.545	0.478	0.629
	camel1.4-camel1.6	79.689	89.6	86.4	85.6	89.6	88.8	0.475	0.839	0.774	0.808	0.862	0.833
	ivy1.4-ivy2.0	86.364	80.769	75	67.308	71.154	75	0.3	0.8	0.714	0.6	0.667	0.714
	poi-2.5-poi3.0	62.67	90	90	85	85	85	0.734	1	1	1	0.938	0.938
	synapse1.1-synapse1.2	69.531	54.545	45.455	54.545	63.636	63.636	0.574	0.625	0.533	0.643	0.733	0.769
	velocity1.5-velocity1.6	59.825	88	84	92	88	88	0.453	0.833	0.824	0.882	0.833	0.833
	xalan2.5-xalan2.6	67.91	85.714	85.714	85.714	85.714	85.714	0.64	0.921	0.921	0.921	0.921	0.921
	xerces1.3-xerces1.4	40.136	73.684	73.684	78.947	78.947	68.421	0.947	1	1	1	1	1
	eclipse2.0-eclipse2.1	83.253	72.336	71.721	68.648	67.418	72.951	0.297	0.649	0.658	0.61	0.602	0.661
	eclipse2.0-eclipse3.0	81.242	67.804	65.484	65.621	60.709	70.532	0.367	0.656	0.629	0.623	0.573	0.685
	eclipse2.1-eclipse3.0	82.3	68.213	67.121	63.574	61.664	68.486	0.355	0.702	0.689	0.635	0.617	0.711
Intra	eclipse JDT CORE	97.687	98.824	97.745	97.745	97.745	99.412	1	0.971	0.958	0.958	0.958	0.983
	eclipse PDE UI	97.278	95	98.333	96	96	97.333	1	0.958	0.988	0.963	0.963	1
	equinox framework	97.605	95.5	94.833	92.167	98.833	99.667	1	1	1	1	1	1
	lucene	97.799	99	100	100	98.167	98.333	1	1	1	1	1	1
	mylyn	96.474	95.969	95.169	93.991	93.78	94.478	0.991	1	1	1	1	1
	eclipse2.0	96.722	96.598	96.841	95.894	96.189	96.894	0.999	0.962	0.964	0.967	0.956	0.96
	eclipse2.1	94.88	95.372	95.974	94.329	94.42	97.255	0.991	0.961	0.972	0.963	0.958	0.967
	eclipse3.0	94.588	94.991	94.863	92.556	93.714	96.168	0.998	0.968	0.968	0.961	0.957	0.968

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.11: Performance of Random Forest in terms of Recall and F-measure.

	Dataset	Recall						F-measure					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.572	0.742	0.774	0.774	0.71	0.71	0.535	0.63	0.686	0.64	0.571	0.667
	camel1.4-camel1.6	0.404	0.765	0.706	0.618	0.735	0.735	0.437	0.8	0.738	0.7	0.794	0.781
	ivy1.4-ivy2.0	0.15	0.632	0.526	0.316	0.421	0.526	0.2	0.706	0.606	0.414	0.516	0.606
	poi-2.5-poi3.0	0.648	0.882	0.882	0.824	0.882	0.882	0.688	0.938	0.938	0.903	0.909	0.909
	synapse1.1-synapse1.2	0.36	0.526	0.421	0.474	0.579	0.526	0.443	0.571	0.471	0.545	0.647	0.625
	velocity1.5-velocity1.6	0.872	1	0.933	1	1	1	0.596	0.909	0.875	0.938	0.909	0.909
	xalan2.5-xalan2.6	0.708	0.921	0.921	0.921	0.921	0.921	0.672	0.921	0.921	0.921	0.921	0.921
	xerces1.3-xerces1.4	0.206	0.677	0.677	0.742	0.742	0.613	0.338	0.808	0.808	0.852	0.852	0.76
	eclipse2.0-eclipse2.1	0.399	0.77	0.708	0.742	0.703	0.756	0.34	0.705	0.682	0.67	0.649	0.705
	eclipse2.0-eclipse3.0	0.367	0.656	0.638	0.671	0.633	0.685	0.367	0.656	0.634	0.646	0.601	0.685
	eclipse2.1-eclipse3.0	0.24	0.557	0.542	0.522	0.478	0.551	0.287	0.621	0.607	0.573	0.539	0.621
Intra	eclipse JDT CORE	0.955	1	1	1	1	1	0.977	0.983	0.977	0.977	0.977	0.991
	eclipse PDE UI	0.949	0.94	0.98	0.96	0.951	0.946	0.973	0.946	0.982	0.96	0.957	0.97
	equinox framework	0.957	0.912	0.904	0.858	0.979	0.996	0.978	0.951	0.944	0.914	0.989	0.998
	lucene	0.957	0.983	1	1	0.967	0.967	0.978	0.991	1	1	0.982	0.982
	mylyn	0.94	0.925	0.91	0.884	0.882	0.891	0.965	0.96	0.951	0.937	0.936	0.941
	eclipse2.0	0.936	0.97	0.966	0.947	0.961	0.976	0.966	0.965	0.965	0.955	0.958	0.967
	eclipse2.1	0.91	0.918	0.933	0.889	0.909	0.966	0.949	0.938	0.951	0.924	0.93	0.966
	eclipse3.0	0.895	0.922	0.921	0.878	0.904	0.945	0.944	0.944	0.944	0.917	0.929	0.956

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.12: Performance of Support Vector Machine in terms of Accuracy % and Precision.

	Dataset	Accuracy %						Precision					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	73.691	70.149	64.179	44.776	50.746	65.672	0.442	0.634	0.581	0.406	0.483	0.682
	camel1.4-camel1.6	70.57	87.2	88	83.2	80	78.4	0.336	0.737	0.744	0.76	0.592	0.606
	ivy1.4-ivy2.0	77.557	61.538	71.154	65.385	61.538	71.154	0.132	0.455	0.75	0.529	0.471	0.75
	poi-2.5-poi3.0	62.896	70	90	95	90	55	0.739	0.867	1	1	0.941	1
	synapse1.1-synapse1.2	63.281	57.576	60.606	66.667	63.636	69.697	0.452	0.632	0.667	0.833	0.769	0.909
	velocity1.5-velocity1.6	55.895	84	92	92	48	84	0.421	0.824	0.882	0.882	0.667	0.867
	xalan2.5-xalan2.6	67.797	73.81	69.048	69.048	76.19	73.81	0.646	0.909	0.931	0.903	0.889	1
	xerces1.3-xerces1.4	50.34	73.684	76.316	73.684	76.316	52.632	0.919	1	1	1	1	1
	eclipse2.0-eclipse2.1	70.449	67.828	63.525	63.32	61.68	70.902	0.214	0.613	0.562	0.558	0.542	0.654
	eclipse2.0-eclipse3.0	72.161	66.576	64.666	64.393	61.937	69.577	0.301	0.644	0.617	0.61	0.584	0.704
	eclipse2.1-eclipse3.0	72.406	67.394	63.847	65.484	58.799	67.258	0.3	0.724	0.684	0.692	0.618	0.767
Intra	eclipse JDT CORE	81.225	92.745	88.235	84.412	89.412	83.235	0.868	0.958	0.933	0.892	0.892	0.867
	eclipse PDE UI	75.529	75.333	77	76.333	71.333	76.333	0.798	0.773	0.814	0.873	0.767	0.863
	equinox framework	73.704	89.333	90.667	85.5	96.833	87.5	0.741	0.927	0.938	0.918	0.965	0.98
	lucene	78.759	86.333	90.667	86.333	81.833	90	0.826	0.908	0.98	0.932	0.922	0.907
	mylyn	74.725	77.215	80.474	79.539	76.863	76.972	0.804	0.849	0.865	0.842	0.87	0.811
	eclipse2.0	72.011	72.621	68.924	74.023	71.22	73.818	0.774	0.737	0.679	0.716	0.684	0.754
	eclipse2.1	69.099	71.537	65.45	71.264	63.623	71.537	0.746	0.753	0.721	0.774	0.765	0.783
	eclipse3.0	70.067	71.767	71.437	68.201	67.549	71.078	0.741	0.725	0.707	0.692	0.681	0.773

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.13: Performance of Support Vector Machine in terms of Recall and F-measure.

	Dataset	Recall						F-measure					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.687	0.839	0.806	0.419	0.903	0.484	0.538	0.722	0.676	0.413	0.629	0.566
	camel1.4-camel1.6	0.521	0.824	0.853	0.559	0.853	0.588	0.408	0.778	0.795	0.644	0.699	0.597
	ivy1.4-ivy2.0	0.175	0.263	0.316	0.474	0.421	0.316	0.151	0.333	0.444	0.5	0.444	0.444
	poi-2.5-poi3.0	0.644	0.765	0.882	0.941	0.941	0.471	0.688	0.812	0.938	0.97	0.941	0.64
	synapse1.1-synapse1.2	0.442	0.632	0.632	0.526	0.526	0.526	0.447	0.632	0.649	0.645	0.625	0.667
	velocity1.5-velocity1.6	0.782	0.933	1	1	0.267	0.867	0.547	0.875	0.938	0.938	0.381	0.867
	xalan2.5-xalan2.6	0.679	0.789	0.711	0.737	0.842	0.711	0.662	0.845	0.806	0.812	0.865	0.831
	xerces1.3-xerces1.4	0.364	0.677	0.71	0.677	0.71	0.419	0.521	0.808	0.83	0.808	0.83	0.591
	eclipse2.0-eclipse2.1	0.648	0.675	0.675	0.689	0.679	0.679	0.322	0.642	0.613	0.617	0.603	0.667
	eclipse2.0-eclipse3.0	0.668	0.638	0.644	0.662	0.647	0.603	0.415	0.641	0.631	0.635	0.614	0.65
	eclipse2.1-eclipse3.0	0.647	0.49	0.423	0.472	0.312	0.431	0.41	0.584	0.523	0.562	0.415	0.552
Intra	eclipse JDT CORE	0.745	0.9	0.813	0.8	0.9	0.713	0.801	0.907	0.859	0.82	0.89	0.763
	eclipse PDE UI	0.709	0.717	0.746	0.657	0.623	0.631	0.749	0.737	0.762	0.71	0.676	0.714
	equinox framework	0.794	0.858	0.888	0.792	0.979	0.779	0.761	0.886	0.905	0.847	0.971	0.862
	lucene	0.736	0.817	0.833	0.8	0.733	0.9	0.778	0.855	0.898	0.854	0.79	0.899
	mylyn	0.667	0.684	0.759	0.749	0.654	0.727	0.729	0.753	0.8	0.79	0.74	0.762
	eclipse2.0	0.628	0.704	0.683	0.774	0.768	0.687	0.693	0.709	0.677	0.742	0.719	0.713
	eclipse2.1	0.61	0.574	0.432	0.526	0.35	0.521	0.67	0.645	0.53	0.607	0.459	0.621
	eclipse3.0	0.632	0.659	0.694	0.598	0.618	0.56	0.682	0.689	0.696	0.64	0.644	0.644

* w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.14 - Table 4.19 show the experimental results obtained for number of faults of software fault prediction. Three classifiers used are Linear Regression, Decision Tree Regression and Multilayer Perceptron. Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC). Following observations can be made from these tables.

4.7.3 Inter-release experiments for Number of Faults Prediction

For Promise datasets, it can be observed from Table 4.14 - Table 4.19 that MAD outperforms the other techniques of aggregation in general, in terms of AAE, ARE and Pred(l) while Median outperforms the other techniques of aggregation in terms of MOC, for all three classifiers used.

For Eclipse datasets, it can be observed from Table 4.14 - Table 4.19 that AAD outperforms the other techniques of aggregation in general, in terms of AAE, ARE and pred(l) for Linear Regression and Multilayer Perceptron classifiers. MAD gives the best results in terms of MOC for Linear Regression and Decision Tree Regression classifiers.

Thus, MAD aggregation technique gives the best results in the above mentioned scenarios for inter-release experiments, for predicting number of faults.

Table 4.14: Performance of Linear Regression in terms of AAE and ARE.

	Dataset	Average Absolute Error						Average Relative Error					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.622	0.897	1.268	0.494	0.417	7.924	0.473	0.707	1.026	0.469	0.388	3.253
	camel1.4-camel1.6	1.038	0.355	0.508	0.313	0.595	6.399	0.776	0.266	0.414	0.299	0.565	3.618
	ivy1.4-ivy2.0	0.422	0.309	0.243	0.019	0.418	1.905	0.336	0.23	0.191	0.013	0.405	0.794
	poi-2.5-poi3.0	0.857	0.889	0.807	0.538	0.714	15.848	0.43	0.461	0.384	0.4	0.424	0.823
	synapse1.1-synapse1.2	0.746	0.464	0.888	0.386	0.735	3.706	0.503	0.333	0.628	0.301	0.567	1.156
	velocity1.5-velocity1.6	1.046	1.892	1.735	11.389	0.853	18.121	0.748	1.041	0.915	9.956	0.695	2.167
	xalan2.5-xalan2.6	0.667	0.242	0.636	0.402	0.699	15.537	0.43	0.158	0.348	0.269	0.448	0.643
	xerces1.3-xerces1.4	2.339	2.098	2.647	1.06	2.897	47.098	0.533	0.588	0.74	0.349	0.778	1.453
	eclipse2.0-eclipse2.1	0.621	0.234	0.514	0.362	0.502	3.923	0.569	0.203	0.48	0.355	0.476	1.852
	eclipse2.0-eclipse3.0	0.634	0.276	0.526	0.359	0.478	4.002	0.538	0.215	0.46	0.344	0.441	1.461
	eclipse2.1-eclipse3.0	0.617	0.234	0.4	0.327	0.496	3.236	0.517	0.168	0.331	0.312	0.459	0.843
Intra	eclipse JDT CORE	0.645	5.076	3.163	4.659	5.058	1.614	0.362	2.334	1.205	2.039	2.351	0.699
	eclipse PDE UI	0.602	0.186	0.419	0.341	0.289	2.099	0.382	0.151	0.287	0.255	0.207	1.071
	equinox framework	0.615	0.091	0.146	0.091	0.265	0.729	0.349	0.065	0.086	0.066	0.178	0.328
	lucene	0.597	3.019	3.094	2.073	2.391	0.725	0.378	1.435	1.973	0.805	0.98	0.428
	mylyn	0.551	0.17	0.275	0.214	0.351	1.243	0.353	0.136	0.205	0.168	0.257	0.594
	eclipse2.0	0.681	0.274	0.515	0.344	0.491	4.052	0.392	0.191	0.321	0.244	0.325	1.559
	eclipse2.1	0.603	0.173	0.353	0.303	0.469	2.602	0.38	0.135	0.25	0.216	0.33	1.085
	eclipse3.0	0.718	0.225	0.44	0.383	0.488	4.186	0.415	0.167	0.297	0.269	0.336	1.888

* AAE=Average Absolute Error, ARE=Average Relative Error, w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.15: Performance of Linear Regression in terms of Pred(l) and Measure of Completeness.

	Dataset	Pred(l)						Measure of Completeness					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	46.174	34.328	28.358	47.761	61.194	7.463	153.776	-28.785	49.935	-185.615	166.344	63.562
	camel1.4-camel1.6	28.083	70.4	53.6	64.8	32.8	15.2	186.747	99.667	171.227	201.862	383.479	71.827
	ivy1.4-ivy2.0	46.307	78.846	80.769	96.154	55.769	23.077	153.248	-4.658	51.1	0	595.417	12.62
	poi-2.5-poi3.0	52.262	30	50	50	40	25	91.785	43.598	67.96	-51.185	128.165	76.31
	synapse1.1-synapse1.2	33.594	69.697	42.424	60.606	42.424	24.242	118.085	27.446	59.116	3.938	119.303	129.053
	velocity1.5-velocity1.6	29.694	36	40	36	36	8	154.957	202.24	152.138	1450.916	196.804	205.428
	xalan2.5-xalan2.6	33.672	92.857	50	64.286	42.857	23.81	95.776	69.824	36.808	65.11	103.27	58.457
	xerces1.3-xerces1.4	32.313	36.842	28.947	60.526	34.211	7.895	31.297	-9.095	-1.32	15.954	-2.943	2.15
	eclipse2.0-eclipse2.1	22.896	81.557	38.73	50.615	21.516	19.672	440.173	179.615	537.193	1693.88	631.71	173.195
	eclipse2.0-eclipse3.0	25.979	78.035	44.065	53.752	25.375	20.873	260.38	116.049	294.314	767.731	434.319	101.629
	eclipse2.1-eclipse3.0	10.649	88.54	51.432	68.759	21.828	32.742	235.6	85.763	202.37	610.691	438.533	58.849
Intra	eclipse JDT CORE	51.384	14.51	17.843	15.588	14.281	42.941	95.201	117.383	119.453	149.343	192.252	97.902
	eclipse PDE UI	45.934	91.333	65.205	71.595	74.5	24.667	95.844	107.655	96.166	111.208	98.181	118.599
	equinox framework	56.199	98.571	92.857	95.625	88.571	67	97.798	99.941	100.242	96.877	98.437	101.48
	lucene	40.718	19.667	23.561	38.167	26.833	55.667	93.93	104	115.248	115.481	99.067	103.578
	mylyn	43.117	92.678	77.071	88.234	66.888	44.722	92.484	104.844	96.113	87.76	93.651	127.817
	eclipse2.0	42.788	81.932	59.91	68.768	48.612	22.318	93.081	118.29	92.281	91.142	92.524	108.983
	eclipse2.1	45.305	93.071	69.017	82.663	51.849	27.615	93.618	101.804	90.799	93.411	87.976	110.852
	eclipse3.0	42.007	86.561	60.386	66.576	50.188	14.346	92.934	104.556	92.105	97.846	95.796	104.538

* Pred(l)=Prediction at level "l",w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

4.7.4 Intra-release experiments for Number of Faults Prediction

For Eclipse datasets, it can be observed from Table 4.14 - Table 4.19 that AAD aggregation technique outperforms all other techniques in terms of all the four performance evaluation measures used, for all the three used classifiers.

For remaining four Eclipse and one Apache datasets, it can be observed from Table 4.14 - Table 4.19 that "without aggregation" method and AAD technique give comparable performance results and both outperform other aggregation techniques in terms of AAE, for Linear Regression and Multilayer Perceptron classifiers. AAD gives the best results in terms of ARE and pred(l) for Linear Regression and Multilayer Perceptron classifiers. Summation gives the best results in terms of MOC, for all three classifiers used.

Thus, AAD aggregation technique gives the best results in case of the above mentioned scenarios for intra-release experiments, for predicting number of faults.

Table 4.16: Performance of Decision Tree Regression in terms of AAE and ARE.

	Dataset	Average Absolute Error						Average Relative Error					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.56	0.294	0.468	0.073	0.293	3.162	0.395	0.205	0.366	0.061	0.27	1.296
	camel1.4-camel1.6	0.793	0.289	0.409	0.222	0.345	4.375	0.522	0.201	0.309	0.197	0.305	1.557
	ivy1.4-ivy2.0	0.312	0.197	0.128	0.019	0.269	1.085	0.24	0.138	0.074	0.013	0.256	0.442
	poi-2.5-poi3.0	0.828	0.345	0.666	0.251	0.502	13.046	0.396	0.164	0.316	0.156	0.351	1.283
	synapse1.1-synapse1.2	0.734	0.353	0.543	0.256	0.434	2.49	0.495	0.234	0.361	0.158	0.315	0.64
	velocity1.5-velocity1.6	1.061	0.496	0.545	0.454	0.809	8.706	0.76	0.312	0.343	0.421	0.693	2.134
	xalan2.5-xalan2.6	0.664	0.269	0.471	0.236	0.628	14.358	0.428	0.173	0.264	0.159	0.403	0.539
	xerces1.3-xerces1.4	2.413	1.351	1.783	1.16	1.896	37.299	0.495	0.372	0.43	0.318	0.41	1.028
	eclipse2.0-eclipse2.1	0.5	0.234	0.479	0.205	0.345	3.393	0.444	0.202	0.437	0.196	0.314	1.493
	eclipse2.0-eclipse3.0	0.529	0.27	0.494	0.21	0.357	3.522	0.421	0.208	0.418	0.189	0.307	1.106
	eclipse2.1-eclipse3.0	0.442	0.224	0.33	0.166	0.317	3.188	0.319	0.156	0.256	0.146	0.27	0.852
Intra	eclipse JDT CORE	0.453	3.293	3.042	3.817	3.399	2.255	0.24	1.38	1.344	1.324	1.203	0.691
	eclipse PDE UI	0.364	0.172	0.252	0.153	0.171	1.859	0.212	0.135	0.174	0.113	0.122	0.689
	equinox framework	0.483	0.081	0.123	0.126	0.245	0.817	0.28	0.054	0.072	0.082	0.154	0.262
	lucene	0.297	2.102	1.503	2.159	2.634	0.908	0.181	0.765	0.517	0.743	1.057	0.328
	mylyn	0.294	0.12	0.157	0.078	0.19	1.155	0.176	0.095	0.117	0.061	0.133	0.605
	eclipse2.0	0.659	0.27	0.314	0.156	0.497	3.107	0.379	0.187	0.196	0.11	0.329	1.072
	eclipse2.1	0.339	0.175	0.186	0.134	0.221	2.516	0.197	0.137	0.126	0.086	0.149	1.025
	eclipse3.0	0.508	0.202	0.352	0.14	0.246	2.854	0.27	0.151	0.237	0.096	0.162	1.034

* AAE=Average Absolute Error, ARE=Average Relative Error, w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.17: Performance of Decision Tree Regression in terms of Pred(l) and Measure of Completeness.

	Dataset	Pred(l)						Measure of Completeness					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	53.02	79.104	61.194	92.537	79.104	23.881	135.099	91.384	154.458	111.667	255.568	111.767
	camel1.4-camel1.6	47.047	72.8	60.8	78.4	64.8	34.4	126.583	96.268	146.276	207.461	249.339	96.738
	ivy1.4-ivy2.0	68.75	86.538	88.462	96.154	78.846	34.615	140.683	30.46	13.964	0	444.457	52.118
	poi-2.5-poi3.0	52.489	85	40	80	50	30	89.381	70.44	26.573	30.213	97.569	107.676
	synapse1.1-synapse1.2	32.031	69.697	57.576	81.818	57.576	30.303	121.92	63.591	55.43	26.395	99.537	81.933
	velocity1.5-velocity1.6	31.878	60	56	60	24	4	153.818	114.296	133.084	314.902	217.233	176.818
	xalan2.5-xalan2.6	31.864	88.095	57.143	80.952	28.571	16.667	95.981	63.756	57.577	74.612	85.957	49.813
	xerces1.3-xerces1.4	30.782	52.632	47.368	57.895	44.737	15.789	23.879	33.154	34.555	4.191	23.5	13.785
	eclipse2.0-eclipse2.1	48.783	82.787	44.057	82.787	58.402	25	348.461	179.133	496.469	927.725	401.856	170.641
	eclipse2.0-eclipse3.0	49.929	78.854	48.84	82.265	59.072	30.423	206.412	115.461	284.542	359.026	287.805	96.844
	eclipse2.1-eclipse3.0	60.436	85.948	63.847	83.356	63.847	32.606	137.736	79.364	146.162	289.281	231.825	61.25
Intra	eclipse JDT CORE	73.698	24.608	26.176	18.431	24.491	41.863	96.234	115.943	119.802	150.745	219.796	97.176
	eclipse PDE UI	76.787	92.667	84.035	94.024	92.667	33.667	93.439	103.754	98.573	100.436	96.756	118.258
	equinox framework	65.347	97.857	95	91.349	85	70	97.96	99.745	103.424	93.206	99.644	106.021
	lucene	81.85	47.333	56.136	35.667	13.5	65.833	92.541	104.565	99.831	104.544	95.355	102.645
	mylyn	82.848	97.09	88.124	98.053	88.804	31.944	92.489	102.1	97.301	97.215	95.499	132.08
	eclipse2.0	45.8	83.682	77.329	93.53	47.908	29.159	93.532	118.363	93.823	94.071	92.431	108.316
	eclipse2.1	80.09	93.738	89.091	92.16	86.08	29.632	93.498	101.792	93.198	91.816	91.038	109.25
	eclipse3.0	70.15	89.887	71.94	94.618	83.973	29.713	93.08	101.903	93.286	94.82	93.135	106.667

* Pred(l)=Prediction at level "l",w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.18: Performance of Multilayer Perceptron in terms of AAE and ARE.

	Dataset	Average Absolute Error						Average Relative Error					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	0.639	0.36	0.536	0.213	0.296	3.968	0.449	0.279	0.435	0.203	0.269	1.812
	camel1.4-camel1.6	0.945	0.389	0.652	0.155	0.513	4.771	0.666	0.3	0.574	0.132	0.473	2.375
	ivy1.4-ivy2.0	0.358	0.216	0.215	0.019	0.204	1.091	0.282	0.153	0.163	0.013	0.184	0.519
	poi-2.5-poi3.0	0.865	0.433	0.547	0.247	0.475	14.199	0.394	0.243	0.297	0.14	0.305	0.837
	synapse1.1-synapse1.2	0.689	0.342	0.511	0.298	0.578	3.44	0.411	0.216	0.304	0.199	0.411	1.605
	velocity1.5-velocity1.6	1.065	0.514	0.669	0.585	0.812	9.097	0.609	0.293	0.379	0.562	0.742	2.4
	xalan2.5-xalan2.6	0.679	0.248	0.509	0.3	0.72	14.951	0.394	0.155	0.267	0.206	0.513	0.542
	xerces1.3-xerces1.4	2.242	1.867	2.264	1.017	1.92	39.456	0.673	0.538	0.587	0.278	0.414	0.767
	eclipse2.0-eclipse2.1	0.703	0.197	0.406	0.256	0.546	3.032	0.655	0.163	0.369	0.248	0.526	1.07
	eclipse2.0-eclipse3.0	0.714	0.257	0.437	0.285	0.538	3.669	0.622	0.186	0.362	0.264	0.507	0.868
	eclipse2.1-eclipse3.0	0.815	0.278	0.305	0.21	0.287	3.09	0.729	0.213	0.231	0.195	0.231	0.708
Intra	eclipse JDT CORE	0.915	5.076	3.163	4.659	5.058	1.614	0.673	2.334	1.205	2.039	2.351	0.699
	eclipse PDE UI	0.589	0.186	0.419	0.341	0.289	2.099	0.388	0.151	0.287	0.255	0.207	1.071
	equinox framework	0.636	0.091	0.146	0.091	0.265	0.729	0.419	0.065	0.086	0.066	0.178	0.328
	lucene	0.673	3.019	3.094	2.073	2.391	0.725	0.503	1.435	1.973	0.805	0.98	0.428
	mylyn	0.54	0.17	0.275	0.214	0.351	1.243	0.289	0.136	0.205	0.168	0.257	0.594
	eclipse2.0	0.654	0.274	0.515	0.344	0.491	4.052	0.343	0.191	0.321	0.244	0.325	1.559
	eclipse2.1	0.595	0.173	0.353	0.303	0.469	2.602	0.322	0.135	0.25	0.216	0.33	1.085
	eclipse3.0	0.71	0.225	0.44	0.383	0.488	4.186	0.32	0.167	0.297	0.269	0.336	1.888

* AAE=Average Absolute Error, ARE=Average Relative Error, w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

Table 4.19: Performance of Multilayer Perceptron in terms of Pred(l) and Measure of Completeness.

	Dataset	Pred(l)						Measure of Completeness					
		w/o agg	AAD	MAD	IQR	MED	SUM	w/o agg	AAD	MAD	IQR	MED	SUM
Inter	ant1.6-ant1.7	43.221	62.687	49.254	77.612	73.134	16.418	86.926	142.37	160.998	41.679	265.702	126.396
	camel1.4-camel1.6	42.487	52	47.2	85.6	55.2	10.4	147.635	50.914	249.33	127.479	410.202	110.329
	ivy1.4-ivy2.0	65.057	82.692	82.692	96.154	76.923	36.538	144.034	29.482	-15.762	0	348.957	68.049
	poi-2.5-poi3.0	47.964	65	50	75	55	10	86.791	61.842	52.738	19.144	91.814	100.474
	synapse1.1-synapse1.2	50.781	66.667	63.636	75.758	51.515	24.242	68.812	35.905	98.55	-8.841	130.005	104.691
	velocity1.5-velocity1.6	37.555	68	52	44	32	16	30.764	146.917	95.047	410.214	254.104	211.021
	xalan2.5-xalan2.6	25.085	90.476	66.667	83.333	33.333	19.048	75.911	63.301	38.099	97.751	130.737	48.301
	xerces1.3-xerces1.4	35.034	23.684	39.474	57.895	47.368	10.526	65.009	-1.689	10.677	17.888	19.285	6.3
	eclipse2.0-eclipse2.1	8.608	83.811	56.148	68.033	18.443	18.033	503.29	97.874	348.022	-64.526	718.879	77.91
	eclipse2.0-eclipse3.0	10.932	79.945	58.527	61.937	19.509	13.097	299.176	60.388	180.61	-117.705	513.675	28.41
	eclipse2.1-eclipse3.0	10.299	89.632	72.033	84.72	81.446	14.734	338.3	114.328	129.516	428.247	107.303	49.021
Intra	eclipse JDT CORE	37.983	14.51	17.843	15.588	14.281	42.941	172.484	117.383	119.453	149.343	192.252	97.902
	eclipse PDE UI	47.748	91.333	65.205	71.595	74.5	24.667	109.237	107.655	96.166	111.208	98.181	118.599
	equinox framework	39.731	98.571	92.857	95.625	88.571	67	114.796	99.941	100.242	96.877	98.437	101.48
	lucene	41.951	19.667	23.561	38.167	26.833	55.667	136.778	104	115.248	115.481	99.067	103.578
	mylyn	54.802	92.678	77.071	88.234	66.888	44.722	47.72	104.844	96.113	87.76	93.651	127.817
	eclipse2.0	47.624	81.932	59.91	68.768	48.612	22.318	73.145	118.29	92.281	91.142	92.524	108.983
	eclipse2.1	49.513	93.071	69.017	82.663	51.849	27.615	60.032	101.804	90.799	93.411	87.976	110.852
	eclipse3.0	51.539	86.561	60.386	66.576	50.188	14.346	48.363	104.556	92.105	97.846	95.796	104.538

* Pred(l)=Prediction at level "l",w/o agg.=Without Aggregation, AAD=Average Absolute Deviation, MAD=Median Absolute Deviation, IQR=Interquartile Range,SUM=Summation, MED=Median.

4.8 OBSERVATIONS

Table 4.4- Table 4.13 show the experimental results obtained for binary classification of software fault prediction. Five classifiers used are Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure.

Table 4.14 - Table 4.19 show the experimental results obtained for number of faults of software fault prediction. Three classifiers used are Linear Regression (LNR), Decision Tree Regression (DTR) and Multilayer Perceptron (MLP). Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC).

A comparative analysis of "without aggregation method" and five existing aggregation techniques ,i.e., AAD, IQR, MAD, MED and SUM is done.

4.8.1 Inter-release Binary Classification

For Promise datasets, it can be observed from Table 4.4- Table 4.13 that AAD performs the best for Naive Bayes (best in 38.46%, 30% and 41.66% cases in terms of accuracy, recall and F-measure respectively) and Random Forest (best in 20%, 22.22% ,25% and 28.57% cases in terms of accuracy, precision, recall and F-measure respectively) classifiers while Summation performs the best for Decision Tree (best in 25%, 28.57%, 40% and 50% cases in terms of accuracy, precision, recall and F-measure respectively) and Logistic Regression (best in 33.33%, 28.57%, 30% and 44.44% cases in terms of accuracy, precision, recall and F-measure respectively) classifiers.

The accuracy value ranges from 40.13% to 92%, precision value ranges from 0.3 to 1, recall ranges from 0.15 to 1 and F-measure ranges from 0.2 to 0.93 for Random Forest.

For Eclipse datasets, it can be observed from Table 4.4- Table 4.13 that none of the aggregation technique could outperform "without aggregation method" in terms of Accuracy. Summation gives the best results in 100% cases, for all five classifiers used in terms of Precision. Median gives the best results in 60% cases in terms of Recall for Decision Tree, Logistic Regression and Naive Bayes while Summation gives the best results in 60% cases, for Logistic Regression, Random Forest and Support Vector Machine classifiers in terms of F-measure.

The accuracy value ranges from 60.7% to 83.25%, precision value ranges from 0.29 to 0.71, recall ranges from 0.24 to 0.77 and F-measure ranges from 0.28 to 0.7 for Random Forest.

4.8.2 Intra-release Binary Classification

For Eclipse datasets, it can be observed from Table 4.4- Table 4.13 that Summation gives the best results in general, for Logistic Regression (best in 100%, 100% ,66.66% and 66.66% cases in terms of accuracy, precision, recall and F-measure respectively) and Random Forest classifier (best in 100%, 100% and 100% cases in terms of accuracy, recall and F-measure respectively).

The accuracy value ranges from 92.55% to 97.25%, precision value ranges from 0.95 to 0.99, recall ranges from 0.87 to 0.97 and F-measure ranges from 0.91 to 0.96 for Random Forest.

For remaining four Eclipse and one Apache datasets, it can be observed from Table 4.4- Table 4.13 that Summation gives the best results in general, for all the five used classifiers in terms of Accuracy and Precision in 60% cases. Summation also gives the best results in terms of Recall (50% cases) and F-measure (60% cases) for Logistic Regression classifier . IQR gives the best results in general, in 60% cases in terms of Recall and F-measure for Naive Bayes, Random Forest and Support Vector Machine classifiers.

The accuracy value ranges from 92.16% to 100%, precision value ranges from 0.95 to 1, recall ranges from 0.85 to 1 and F-measure ranges from 0.91 to 0.1 for Random Forest.

4.8.3 Inter-release experiments for Number of Faults Prediction

For Promise datasets, it can be observed from Table 4.14 - Table 4.19 that MAD outperforms the other techniques of aggregation in general, in terms of AAE, ARE and Pred(l) in 100% cases while Median outperforms the other techniques of aggregation in terms of MOC in 100% cases, for all three classifiers used. The AAE value ranges from 0.01 to 47 , ARE value ranges from 0.01 to 9.9, pred(l) ranges from 7.4 to 96.15 and MOC ranges from -185.6 to 1450 for Linear Regression.

For Eclipse datasets, it can be observed from Table 4.14 - Table 4.19 that AAD outperforms the other techniques of aggregation in general, in terms of AAE, ARE and pred(l) for Linear Regression and Multilayer Perceptron classifiers in 77.78% cases. MAD gives the best results in terms of MOC for Linear Regression and Decision Tree Regression classifiers in 66.66% cases. The AAE value ranges from 0.23 to 4 , ARE value ranges from 0.16 to 1.85, pred(l) ranges from 10.64 to 88.54 and MOC ranges from 58.84 to 1693 for Linear Regression.

4.8.4 Intra-release experiments for Number of Faults Prediction

For Eclipse datasets, it can be observed from Table 4.14 - Table 4.19 that AAD aggregation technique outperforms all other techniques in 100% cases, in terms of all the four performance evaluation measures used, for all the three used classifiers. The AAE value ranges from 0.17 to 4.18 , ARE value ranges from 0.13 to 1.88, pred(l) ranges from 14.34 to 93.07 and MOC ranges from 87.97 to 118.2 for Linear Regression.

For remaining four Eclipse and one Apache datasets, it can be observed from Table 4.14 - Table 4.19 that "without aggregation" method and AAD technique give comparable performance results and both outperform other aggregation techniques in 100% cases in terms of AAE, for Linear Regression and Multilayer Perceptron classifiers. AAD gives the best results in terms of ARE and pred(l) for Linear Regression and Multilayer Perceptron classifiers in 66.66% cases. Summation gives the best results in 75% cases in terms of MOC, for all three classifiers used. The AAE value ranges from 0.09 to 5.07 , ARE value ranges from 0.06 to 2.35, pred(l) ranges from 14.28 to 98.57 and MOC ranges from 87.75 to 192 for Linear Regression.

4.9 CONCLUSION

For binary classification in software fault prediction, AAD and Summation aggregation techniques outperform "without aggregation" and other aggregation techniques in the scenarios mentioned. Five classifiers used are Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure. For predicting number of faults in software fault prediction, MAD and Summation aggregation techniques outperform "without aggregation" and other aggregation techniques in the scenarios mentioned. Three classifiers used are Linear Regression (LNR), Decision Tree Regression (DTR) and Multilayer Perceptron (MLP). Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC).

CHAPTER 5

PROPOSED AGGREGATION TECHNIQUES

Aggregation techniques are useful in aggregating the software metrics available at different levels of granularity for training and testing datasets and thus bring the metrics at the same level of granularity. In this chapter, three novel aggregation techniques ,i.e., Average of Quarter Medians (QM_AVG), Median of Quarter Medians (QM_MED) and Summation of Quarter Medians (QM_SUM) are proposed and their performances are explored in the field of software fault prediction. The performances of these three techniques are also compared with the five existing aggregation techniques used in previous chapter. Performance of "without aggregation method" is also compared with the performances of these three techniques. Inter-release and intra-release experiments are performed using binary classification and predicting number of faults in software fault prediction.

5.1 INTRODUCTION

Software Fault Prediction is the mechanism to predict whether a software module is going to be faulty or non faulty. This prediction mechanism is applied before the testing phase in the development life cycle of the software. The prediction helps in deciding the amount of resources required in the testing phase. If the module is predicted to be faulty, then more resources are deployed for testing mechanism as compared to the resources deployed when a module is predicted to be non faulty. In case of inter-releases software fault prediction, the data that is used to train the classifier might not always be available at the same level of granularity as that of the data used for testing. The same scenario may also happen in the cross project fault prediction. Thus, there is a need to first bring the metrics at the same level of granularity and then apply the prediction mechanism. Aggregation of metrics is helpful in such scenarios where the metric values are to be combined together to bring them from some lower level to some higher level of granularity. In this work, the software metrics available at the class and file level are aggregated to package level by using different aggregation techniques.

5.2 RELATED WORKS AND MOTIVATION

Some of the works done on software metrics aggregation techniques in the filed of software fault prediction are as follows.

Zimmermann et al. [7] worked on three releases of publicly available eclipse datasets and mapped the packages and classes to the number of bugs that were reported before and after the release. Post release bugs are the actual ones that matter for the users of the software program. They used version archives and bug tracking systems to find the failed modules in the system. The keywords like bug, fixed etc. were captured in the version archives to locate the bugs. They computed the metrics at method, class and file level and aggregated them to higher levels ,i.e., file and package level. The aggregation techniques used were average, total and maximum values of the metrics. Logistic regression was used as the machine learning technique. A module was considered faulty even if it contained a single bug.

Herzig [8] used summation, median, mean and maximum value as the metric aggregation techniques in software fault prediction mechanism in his work. Posnett et al. [5] used summation while Koru and Liu [4] used minimum, maximum, summation and average for the aggregation of metrics in software fault prediction in their works.

Other than software fault prediction, aggregation techniques have been used in other fields also. There are mainly two categories of the aggregation techniques: traditional and econometrics aggregation techniques. Traditional techniques of aggregation consist of mean, median and summation techniques. Econometrics techniques of aggregation consist of Gini, Theil, Kolm, Atkinson and Hoover inequality indices. Vasilescu et al. [9] studied the traditional and econometrics aggregation techniques to analyze the correlations amongst them.

There exist several techniques for the aggregation of software metrics to bring them from lower level to a higher level, in which the most commonly used ones are sum, median, mean and maximum of the metric values. These techniques do not reveal much about the nature of distribution of values. Summation technique gives the accumulative effect of the set of values, revealing nothing about the range of values that exist in the set. Median just gives the central value when the values are arranged in a sorted order, telling nothing about the distribution of values that are present before and after it, in the sorted order of values. It just focuses on the central value and ignores its neighbouring values. Mean of the set of values smoothens the values [10]. It cannot differentiate between the two cases when all the values in the set are almost equal and when there are different values present in the set such that their average value is the same as that in the first case. Maximum value just gives one single value out of all the values present in the set, giving no information about the rest of the values. In order to include not only a specific point or a specific region in calculating the aggregated value but different regions in the distribution of values in the set, three aggregation techniques have been proposed in this work.

In order to focus on the complete set of values instead of just a central point or a specific region, the three aggregation techniques ,i.e., QM_AVG, QM_MED and QM_SUM are proposed. In all of these techniques the complete set (in sorted order) is divided into four equal parts, called quarters, and is then taken into consideration for calculating the final value. Thus, four different regions are considered together in defining the aggregated value which takes care of the distribution of values.

5.3 PROPOSED AGGREGATION TECHNIQUES

In the inter-releases prediction and cross project fault prediction, the granularity of training and testing dataset metrics might not always be the same and when they are needed to be brought at the same level, then aggregation of the metrics can be used. In a particular package there exist several classes (or files). The metric values of all those classes (or files) which belong to the same package are combined together by using aggregation technique to give one value per metric for every package. It needs to be done for all the classes (or files) and packages. In this work, three novel aggregation techniques ,as listed in Table 5.1, are used for analyzing their effect on the software fault prediction performance:

a) Average of Quarter Medians (QM_AVG): The complete set of values in sorted order is divided into four equal halves (quarters). QM_AVG is calculated by taking the average value of the median values of the four quarters.

S.No.	Aggregation Technique	Formula
1	Average of Quarter Medians	$\frac{1}{4} * (Med(QM1), Med(QM2), Med(QM3), Med(QM4))$
2	Median of Quarter Medians	$Median(Med(QM1), Med(QM2), Med(QM3), Med(QM4))$
3	Sum of Quarter Medians	$Med(QM1) + Med(QM2) + Med(QM3) + Med(QM4))$

Table 5.1: List of the Proposed Aggregation Techniques.
Med:Median,QM:Quarter Median

$$QM\_AVG = \frac{1}{4} * (Median(Q1), Median(Q2), Median(Q3), Median(Q4)) \tag{5.1}$$

Where Q1= first quarter, Q2= second quarter, Q3= third quarter, Q4= fourth quarter of the given set of values.

b) Median of Quarter Medians (QM_MED): The complete set of values in sorted order is divided into four equal halves (quarters). QM_MED is calculated by taking the median of the median values of the four quarters.

$$QM\_MED = Median(Median(Q1), Median(Q2), Median(Q3), Median(Q4)) \tag{5.2}$$

Where Q1= first quarter, Q2= second quarter, Q3= third quarter, Q4= fourth quarter of the given set of values.

c) Sum of Quarter Medians (QM_SUM): The complete set of values in sorted order is divided into four equal halves (quarters). QM_SUM is calculated by summing up the median values of the four quarters.

$$QM\_SUM = Median(Q1) + Median(Q2) + Median(Q3) + Median(Q4) \tag{5.3}$$

Where Q1= first quarter, Q2= second quarter, Q3= third quarter, Q4= fourth quarter of the given set of values.

5.4 DATASETS USED

Sixteen releases of datasets from the PROMISE data repository, three releases of publicly available eclipse dataset, one apache dataset and four other publicly available eclipse datasets are used for experimentation [7], [24].

5.4.1 Inter-release experiments

The earlier release of a dataset is used for training purpose to predict the fault proneness for the later release that is used as testing dataset. There are eight pairs of training-testing datasets in our experiments. Table 5.2 provides the details of the used datasets.

S.No.	Training Dataset	Testing Dataset
1	ant 1.6	ant 1.7
2	camel 1.4	camel 1.6
3	ivy 1.4	ivy 2.0
4	poi 2.5	poi 3.0
5	synapse 1.1	synapse 1.2
6	velocity 1.5	velocity 1.6
7	xalan 2.5	xalan 2.6
8	xerces 1.3	xerces 1.4
9	eclipse 2.0	eclipse 2.1
10	eclipse 2.0	eclipse 3.0
11	eclipse 2.1	eclipse 3.0

Table 5.2: Training-Testing datasets used for Inter-release experiments.

5.4.2 Intra-release experiments

Table 5.3 shows the list of datasets used for performing the intra-release experiment. 10 fold cross validation techniques is used . The datasets is partitioned into 10 equal parts called folds, each fold having almost equal number of faulty an non faulty instances. Thus, each fold is free from class imbalance problem. 9 out of 10 folds are used to train the classifier and testing is done on the 10th fold. This is repeated for ten times, making every fold as the testing data once.

S.No.	Dataset
1	eclipse JDT CORE
2	eclipse PDE UI
3	equinox framework
4	lucene
5	mylyn
6	eclipse 2.0
7	eclipse 2.1
8	eclipse 3.0

Table 5.3: Datasets used for Intra-release experiments.

5.5 BINARY CLASSIFICATION IN SOFTWARE FAULT PREDICTION

Binary classification in software fault prediction means that either the module under consideration will be labeled as faulty or non faulty. There are only two labels possible for prediction. In binary classification of fault prediction, if in a package even a single faulty class (or file) is present then that package is declared to be faulty otherwise non faulty [6], [7], [25].

5.5.1 Machine Learning Techniques used

Five machine learning techniques used are naive bayes (Yang et al., 2017), (Turhan et al., 2013), logistic regression (Arar and Ayan, 2016), (Zhao et al., 2017), support vector machine (Erturk and Sezer, 2015), decision tree (Ghotra et al., 2015) and random forest (Kamei and Shihab, 2016).

5.5.2 Performance Evaluation Measures used

In binary classification of fault prediction, if in a package, even a single faulty class (or file) is present then that package is declared to be faulty otherwise non faulty [26], [7], [25]. This concept is used for calculation of values of performance measures. Four different performance evaluation measures are used as discussed below:

Accuracy: It denotes the percentage of correctly classified instances to the total number of instances.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} * 100 \tag{5.4}$$

Precision: It denotes the number of correctly classified faulty instances amongst the total number of instances classified as faulty.

$$Precision = \frac{TP}{TP+FP} \tag{5.5}$$

Recall: It denotes the number of correctly classified faulty instances amongst the total number of instances which are faulty.

$$Recall = \frac{TP}{TP+FN} \tag{5.6}$$

F-measure: It denotes the harmonic mean of the precision and recall values.

$$F - measure = \frac{2 * precision * recall}{precision + recall} \tag{5.7}$$

Where TP represents True Positive, FP represents False Positive, TN represents True Negative and FN represents False Negative.

5.6 NUMBER OF FAULTS IN SOFTWARE FAULT PREDICTION

In software fault prediction mechanism, the fault proneness of the module is predicted using some classifier. This fault proneness can be in terms of binary classification or in terms of the number of faults present in the module. Finding the number of faults in a module gives more accurate information about the fault proneness of the given module. It is better than just having the information whether a module is faulty or non faulty. Binary classification of fault proneness does not give the exact information about how less or more the module is fault prone.

5.6.1 Machine Learning Techniques used

Three machine learning techniques have been used experimentations for predicting the number of faults in software fault prediction. These techniques are linear regression , multilayer perceptron and decision tree regression [1], [27], [28], [29].

5.6.2 Performance Evaluation Measures used

Following are the performance evaluation measures used in finding the number of faults in software fault prediction:

Average Absolute Error: It calculates the difference in the predicted and actual values and takes the average value considering all the instances. Its value ranges from 0 to 1. Lower the AAE better is the prediction.

$$AAE = \sum_{i=1}^{n} |X_i - Y_i| \tag{5.8}$$

Here n is the number of instances,X_i is the predicted value and Y_i is the actual value of an instance.

Average Relative Error: It calculates the ratio of the difference in the predicted and actual values to the actual value of an instance and then finds the average value for all the instances. Its value ranges from 0 to 1. Lower the ARE better is the prediction.

$$ARE = \sum_{i=1}^{n}(|X_i - Y_i|/Y_i + 1) \tag{5.9}$$

Here n is the number of instances,X_i is the predicted value and Y_i is the actual value of an instance. Sometimes the value of Y_i can be 0, making the fraction undefined. In order to avoid such situations an additional 1 is added in the denominator value [30].

Prediction at level 'l': It calculates the number of predictions having the predicted value within l% of the actual value. It calculates the number of predictions which have the ARE value under a certain predefined threshold value, generally taken to be 30%. Thus it calculates the percentage of the number of predictions whose ARE value is lesser than or equal to 0.3 [31].

$$Pred(l) = k/n \tag{5.10}$$

Here n is the total number of modules while k is the number of those modules which have the predicted value less than or equal to 'l'.

Measure of Completeness: It depicts the ratio of the number of faults predicted to the actual number of faults present in the overall modules. It is a measure to find how complete a model is in finding the number of faults as compared to the actual number of faults present.

$$MOC = \frac{Predicted\ number\ of\ faults}{Actual\ number\ of\ faults\ present} \tag{5.11}$$

5.7 EXPERIMENTAL RESULTS AND ANALYSIS

Table 5.4- Table 5.13 show the experimental results obtained for binary classification of software fault prediction. Five classifiers used are Decision Tree, Logistic Regression, Naive Bayes, Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure. "Without Aggregation method" is compared with all eight other aggregation techniques in this section. Following observations can be made from these tables.

5.7.1 Inter-release Binary Classification

For Promise datasets, it can be observed from Table 5.4- Table 5.13 that QM_MED gives the best results for Recall and F-measure when Logistic Regression and Support Vector Machine classifiers are used. QM_AVG outperforms other techniques in terms of Accuracy, Recall and F-measure when Random Forest is used. It also outperforms other techniques in terms of Accuracy and F-measure when Support Vector Machine is used.

For Eclipse datasets, it can be observed from Table 5.4- Table 5.13 that no aggregation techniques could outperform "without aggregation method" in terms of Accuracy. Summation gives the best results for all the classifiers used in terms of all four performance evaluation measures used. MED and QM_MED give comparable results and outperform all other techniques when Naive Bayes classifier is used, in terms of Recall and F-measure.

Thus, QM_MED and QM_AVG outperform other techniques, in general in inter-release binary classification of software fault prediction for above mentioned scenarios.

Table 5.4: Performance of Decision Tree in terms of Accuracy % and Precision.

	Dataset	Accuracy %				Precision			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	75.168	58.209	58.209	58.209	0.453	0.533	0.537	0.533
	camel1.4-camel1.6	77.927	84	79.2	89.6	0.429	0.719	0.618	0.818
	ivy1.4-ivy2.0	82.67	78.846	73.077	63.462	0.2	0.75	0.692	0.5
	poi-2.5-poi3.0	41.403	90	90	90	0.612	0.941	0.941	0.941
	synapse1.1-synapse1.2	69.531	66.667	66.667	66.667	0.557	0.786	0.786	0.75
	velocity1.5-velocity1.6	57.205	76	80	76	0.429	0.8	0.812	0.8
	xalan2.5-xalan2.6	57.853	80.952	78.571	80.952	0.541	0.917	0.914	0.917
	xerces1.3-xerces1.4	39.456	68.421	65.789	60.526	0.872	1	1	1
	eclipse2.0-eclipse2.1	80.325	66.189	62.09	66.189	0.247	0.589	0.543	0.589
	eclipse2.0-eclipse3.0	78.524	63.165	60.3	63.165	0.312	0.592	0.562	0.592
	eclipse2.1-eclipse3.0	79.911	64.666	65.075	64.666	0.291	0.66	0.657	0.66
Intra	eclipse JDT CORE	94.861	93.333	89.804	95	0.981	0.935	0.919	0.95
	eclipse PDE UI	92.385	92.333	76	92.333	0.967	0.958	0.93	0.971
	equinox framework	81.067	98.333	96.5	97.667	0.759	0.986	0.986	0.982
	lucene	95.585	96.667	93.333	94.833	0.971	0.983	0.98	0.969
	mylyn	93.435	90.901	91.592	87.846	0.966	0.945	0.931	0.908
	eclipse2.0	92.939	78.879	70.053	79.326	0.973	0.77	0.658	0.766
	eclipse2.1	94.198	74.052	67.238	73.61	0.979	0.885	0.652	0.893
	eclipse3.0	91.49	78.283	71.161	79.872	0.982	0.745	0.79	0.77

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.5: Performance of Decision Tree in terms of Recall and F-measure.

	Dataset	Recall				F-measure			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	0.554	0.774	0.71	0.774	0.499	0.632	0.611	0.632
	camel1.4-camel1.6	0.404	0.676	0.618	0.794	0.416	0.697	0.618	0.806
	ivy1.4-ivy2.0	0.175	0.632	0.474	0.474	0.187	0.686	0.562	0.486
	poi-2.5-poi3.0	0.214	0.941	0.941	0.941	0.317	0.941	0.941	0.941
	synapse1.1-synapse1.2	0.453	0.579	0.579	0.632	0.5	0.667	0.667	0.686
	velocity1.5-velocity1.6	0.769	0.8	0.867	0.8	0.55	0.8	0.839	0.8
	xalan2.5-xalan2.6	0.611	0.868	0.842	0.868	0.574	0.892	0.877	0.892
	xerces1.3-xerces1.4	0.217	0.613	0.581	0.516	0.348	0.76	0.735	0.681
	eclipse2.0-eclipse2.1	0.399	0.699	0.727	0.699	0.305	0.639	0.622	0.639
	eclipse2.0-eclipse3.0	0.376	0.685	0.682	0.685	0.341	0.635	0.617	0.635
	eclipse2.1-eclipse3.0	0.249	0.504	0.531	0.504	0.269	0.572	0.587	0.572
Intra	eclipse JDT CORE	0.919	0.967	0.88	0.967	0.948	0.941	0.883	0.951
	eclipse PDE UI	0.884	0.886	0.574	0.886	0.923	0.917	0.684	0.917
	equinox framework	0.941	0.983	0.954	0.979	0.837	0.983	0.967	0.979
	lucene	0.941	0.95	0.883	0.933	0.956	0.965	0.925	0.948
	mylyn	0.903	0.872	0.91	0.848	0.933	0.903	0.919	0.876
	eclipse2.0	0.885	0.816	0.815	0.823	0.927	0.787	0.724	0.792
	eclipse2.1	0.908	0.531	0.627	0.527	0.942	0.637	0.632	0.633
	eclipse3.0	0.848	0.831	0.575	0.843	0.91	0.786	0.652	0.803

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.6: Performance of Logistic Regression in terms of Accuracy % and Precision.

	Dataset	Accuracy %				Precision			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	73.154	52.239	59.701	52.239	0.432	0.483	0.548	0.483
	camel1.4-camel1.6	60.622	76.8	70.4	77.6	0.253	0.553	0.469	0.568
	ivy1.4-ivy2.0	77.273	61.538	50	61.538	0.065	0.476	0.316	0.471
	poi-2.5-poi3.0	66.29	85	85	85	0.758	1	0.938	1
	synapse1.1-synapse1.2	62.891	69.697	60.606	69.697	0.455	0.8	0.75	0.737
	velocity1.5-velocity1.6	61.135	80	80	88	0.456	0.812	0.812	0.833
	xalan2.5-xalan2.6	56.384	73.81	83.333	78.571	0.537	0.909	0.919	0.892
	xerces1.3-xerces1.4	47.619	71.053	71.053	63.158	0.901	1	1	0.947
	eclipse2.0-eclipse2.1	75.228	60.451	62.09	60.451	0.24	0.533	0.551	0.533
	eclipse2.0-eclipse3.0	75.767	60.982	62.619	60.982	0.32	0.581	0.597	0.581
	eclipse2.1-eclipse3.0	75.333	60.709	59.618	60.709	0.316	0.637	0.614	0.637
Intra	eclipse JDT CORE	77.542	82.647	75.98	82.157	0.848	0.85	0.744	0.823
	eclipse PDE UI	69.061	80.667	80.333	81	0.754	0.839	0.836	0.834
	equinox framework	71.527	93	91.667	94.333	0.75	0.969	0.94	1
	lucene	66.775	81.5	81.333	79	0.764	0.872	0.856	0.855
	mylyn	68.761	63.421	63.32	62.486	0.775	0.724	0.67	0.685
	eclipse2.0	69.256	66.265	67.515	66.712	0.792	0.651	0.654	0.655
	eclipse2.1	66.441	63.918	63.623	62.251	0.768	0.714	0.694	0.673
	eclipse3.0	66.518	63.477	62.623	64.14	0.771	0.649	0.639	0.661

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.7: Performance of Logistic Regression in terms of Recall and F-measure.

	Dataset	Recall				F-measure			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	0.651	0.452	0.742	0.452	0.519	0.467	0.63	0.467
	camel1.4-camel1.6	0.521	0.765	0.676	0.735	0.34	0.642	0.554	0.641
	ivy1.4-ivy2.0	0.075	0.526	0.316	0.421	0.07	0.5	0.316	0.444
	poi-2.5-poi3.0	0.69	0.824	0.882	0.824	0.723	0.903	0.909	0.903
	synapse1.1-synapse1.2	0.523	0.632	0.474	0.737	0.486	0.706	0.581	0.737
	velocity1.5-velocity1.6	0.731	0.867	0.867	1	0.562	0.839	0.839	0.909
	xalan2.5-xalan2.6	0.438	0.789	0.895	0.868	0.483	0.845	0.907	0.88
	xerces1.3-xerces1.4	0.332	0.645	0.645	0.581	0.485	0.784	0.784	0.72
	eclipse2.0-eclipse2.1	0.594	0.622	0.617	0.622	0.342	0.574	0.582	0.574
	eclipse2.0-eclipse3.0	0.568	0.598	0.621	0.598	0.409	0.589	0.609	0.589
	eclipse2.1-eclipse3.0	0.573	0.373	0.37	0.373	0.408	0.471	0.462	0.471
Intra	eclipse JDT CORE	0.681	0.813	0.813	0.847	0.754	0.815	0.755	0.825
	eclipse PDE UI	0.607	0.786	0.771	0.791	0.67	0.794	0.788	0.803
	equinox framework	0.698	0.896	0.912	0.892	0.716	0.922	0.921	0.941
	lucene	0.496	0.767	0.783	0.733	0.601	0.811	0.81	0.779
	mylyn	0.548	0.491	0.563	0.504	0.641	0.574	0.605	0.578
	eclipse2.0	0.529	0.646	0.687	0.652	0.634	0.646	0.669	0.651
	eclipse2.1	0.503	0.402	0.398	0.383	0.607	0.494	0.495	0.48
	eclipse3.0	0.487	0.513	0.51	0.519	0.596	0.57	0.563	0.579

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.8: Performance of Naive Bayes in terms of Accuracy % and Precision.

	Dataset	Accuracy %				Precision			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	77.718	62.687	62.687	62.687	0.5	0.594	0.583	0.594
	camel1.4-camel1.6	73.575	77.6	65.6	76.8	0.321	0.65	0.371	0.609
	ivy1.4-ivy2.0	82.955	78.846	73.077	76.923	0.321	0.786	0.619	0.769
	poi-2.5-poi3.0	47.964	75	60	75	0.823	0.929	0.909	0.929
	synapse1.1-synapse1.2	66.406	54.545	54.545	57.576	0.5	0.6	0.583	0.619
	velocity1.5-velocity1.6	67.686	72	76	72	0.534	0.786	0.765	0.786
	xalan2.5-xalan2.6	61.921	54.762	50	57.143	0.708	0.88	0.87	0.885
	xerces1.3-xerces1.4	40.476	78.947	55.263	76.316	0.958	1	1	1
	eclipse2.0-eclipse2.1	85.028	52.049	52.664	52.049	0.312	0.466	0.471	0.466
	eclipse2.0-eclipse3.0	83.65	56.48	56.207	56.48	0.427	0.522	0.52	0.522
	eclipse2.1-eclipse3.0	84.32	53.752	52.251	53.752	0.446	0.503	0.494	0.503
Intra	eclipse JDT CORE	63.864	72.157	67.549	74.902	0.865	0.842	0.76	0.867
	eclipse PDE UI	61.362	67.667	63.667	67.667	0.745	0.757	0.635	0.767
	equinox framework	66.484	86.333	88.167	88.667	0.829	0.887	0.875	0.933
	lucene	58.918	75	71.167	73.833	0.743	0.74	0.78	0.72
	mylyn	60.845	62.907	56.518	61.585	0.785	0.617	0.548	0.602
	eclipse2.0	62.316	60.348	61.288	59.553	0.833	0.556	0.565	0.552
	eclipse2.1	58.632	53.082	51.403	53.082	0.819	0.496	0.487	0.498
	eclipse3.0	59.802	55.74	53.167	55.639	0.819	0.524	0.508	0.523

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.9: Performance of Naive Bayes in terms of Recall and F-measure.

	Dataset	Recall				F-measure			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	0.59	0.613	0.677	0.613	0.541	0.603	0.627	0.603
	camel1.4-camel1.6	0.319	0.382	0.382	0.412	0.32	0.481	0.377	0.491
	ivy1.4-ivy2.0	0.45	0.579	0.684	0.526	0.375	0.667	0.65	0.625
	poi-2.5-poi3.0	0.231	0.765	0.588	0.765	0.361	0.839	0.714	0.839
	synapse1.1-synapse1.2	0.593	0.632	0.737	0.684	0.543	0.615	0.651	0.65
	velocity1.5-velocity1.6	0.397	0.733	0.867	0.733	0.456	0.759	0.812	0.759
	xalan2.5-xalan2.6	0.307	0.579	0.526	0.605	0.428	0.698	0.656	0.719
	xerces1.3-xerces1.4	0.208	0.742	0.452	0.71	0.342	0.852	0.622	0.83
	eclipse2.0-eclipse2.1	0.317	0.833	0.842	0.833	0.315	0.598	0.604	0.598
	eclipse2.0-eclipse3.0	0.305	0.845	0.851	0.845	0.356	0.645	0.645	0.645
	eclipse2.1-eclipse3.0	0.247	0.848	0.869	0.848	0.318	0.632	0.63	0.632
Intra	eclipse JDT CORE	0.341	0.593	0.573	0.573	0.488	0.657	0.608	0.644
	eclipse PDE UI	0.389	0.563	0.643	0.569	0.51	0.626	0.63	0.628
	equinox framework	0.467	0.829	0.921	0.858	0.591	0.845	0.892	0.886
	lucene	0.28	0.833	0.633	0.867	0.403	0.775	0.689	0.775
	mylyn	0.32	0.783	0.95	0.762	0.454	0.685	0.693	0.67
	eclipse2.0	0.316	0.898	0.903	0.9	0.458	0.685	0.693	0.682
	eclipse2.1	0.254	0.856	0.851	0.866	0.387	0.622	0.613	0.624
	eclipse3.0	0.27	0.874	0.886	0.872	0.405	0.652	0.643	0.651

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

5.7.2 Intra-release Binary Classification

For Eclipse datasets, it can be observed from Table 5.4- Table 5.13 that QM_MED gives the best results in terms of Recall and F-measure when Naive Bayes classifier is used.

For four Eclipse and one Apache dataset, it can be observed from Table 5.4- Table 5.13 that QM_MED gives the best results in terms of Recall and F-measure when Naive Bayes classifier is used. It also gives the best results in terms of Precision, Recall and F-measure when Support Vector Machine classifier is used.

Thus, QM_MED outperforms other aggregation technique in the above mentioned scenarios for intra-release binary classification of software fault prediction.

Table 5.10: Performance of Random Forest in terms of Accuracy % and Precision.

	Dataset	Accuracy %				Precision			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
	ant1.6-1nt1.7	77.852	61.194	58.209	61.194	0.503	0.558	0.535	0.558
	camel1.4-camel1.6	79.689	90.4	86.4	89.6	0.475	0.893	0.793	0.862
	ivy1.4-ivy2.0	86.364	78.846	73.077	76.923	0.3	0.786	0.632	0.818
	poi-2.5-poi3.0	62.67	90	90	90	0.734	0.941	0.941	0.941
	synapse1.1-synapse1.2	69.531	63.636	63.636	63.636	0.574	0.706	0.733	0.706
Inter	velocity1.5-velocity1.6	59.825	84	88	80	0.453	0.824	0.833	0.812
	xalan2.5-xalan2.6	67.91	85.714	83.333	85.714	0.64	0.921	0.919	0.921
	xerces1.3-xerces1.4	40.136	76.316	73.684	76.316	0.947	1	0.957	1
	eclipse2.0-eclipse2.1	83.253	71.721	66.393	71.311	0.297	0.65	0.591	0.646
	eclipse2.0-eclipse3.0	81.242	64.393	62.756	64.802	0.367	0.613	0.597	0.616
	eclipse2.1-eclipse3.0	82.3	65.484	64.666	66.439	0.355	0.66	0.658	0.671
	eclipse JDT CORE	97.687	98.333	98.235	98.333	1	0.975	0.962	0.975
	eclipse PDE UI	97.278	97	94.333	97	1	0.983	0.98	0.983
	equinox framework	97.605	100	99.333	99.667	1	1	1	1
	lucene	97.799	98.333	99.167	98.333	1	1	1	1
Intra	mylyn	96.474	94.444	94.648	95.833	0.991	1	0.988	1
	eclipse2.0	96.722	96.045	96.636	96.447	0.999	0.954	0.961	0.954
	eclipse2.1	94.88	96.45	94.784	96.234	0.991	0.969	0.955	0.954
	eclipse3.0	94.588	95.781	93.668	96.315	0.998	0.96	0.96	0.967

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.11: Performance of Random Forest in terms of Recall and F-measure.

	Dataset	Recall				F-measure			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	0.572	0.774	0.742	0.774	0.535	0.649	0.622	0.649
	camel1.4-camel1.6	0.404	0.735	0.676	0.735	0.437	0.806	0.73	0.794
	ivy1.4-ivy2.0	0.15	0.579	0.632	0.474	0.2	0.667	0.632	0.6
	poi-2.5-poi3.0	0.648	0.941	0.941	0.941	0.688	0.941	0.941	0.941
	synapse1.1-synapse1.2	0.36	0.632	0.579	0.632	0.443	0.667	0.647	0.667
	velocity1.5-velocity1.6	0.872	0.933	1	0.867	0.596	0.875	0.909	0.839
	xalan2.5-xalan2.6	0.708	0.921	0.895	0.921	0.672	0.921	0.907	0.921
	xerces1.3-xerces1.4	0.206	0.71	0.71	0.71	0.338	0.83	0.815	0.83
	eclipse2.0-eclipse2.1	0.399	0.737	0.699	0.732	0.34	0.691	0.64	0.686
	eclipse2.0-eclipse3.0	0.367	0.65	0.63	0.659	0.367	0.631	0.613	0.637
	eclipse2.1-eclipse3.0	0.24	0.542	0.51	0.554	0.287	0.595	0.575	0.607
Intra	eclipse JDT CORE	0.955	1	1	1	0.977	0.986	0.977	0.986
	eclipse PDE UI	0.949	0.96	0.906	0.96	0.973	0.969	0.939	0.969
	equinox framework	0.957	1	0.992	0.996	0.978	1	0.996	0.998
	lucene	0.957	0.967	0.983	0.967	0.978	0.982	0.991	0.982
	mylyn	0.94	0.889	0.912	0.917	0.965	0.94	0.947	0.956
	eclipse2.0	0.936	0.966	0.959	0.976	0.966	0.959	0.959	0.964
	eclipse2.1	0.91	0.947	0.929	0.947	0.949	0.957	0.939	0.949
	eclipse3.0	0.895	0.953	0.904	0.951	0.944	0.956	0.93	0.958

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.12: Performance of Support Vector Machine in terms of Accuracy % and Precision.

	Dataset	Accuracy %				Precision			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	73.691	53.731	52.239	53.731	0.442	0.5	0.492	0.5
	camel1.4-camel1.6	70.57	83.2	72.8	86.4	0.336	0.686	0.5	0.73
	ivy1.4-ivy2.0	77.557	76.923	67.308	78.846	0.132	0.667	0.538	0.722
	poi-2.5-poi3.0	62.896	95	90	80	0.739	0.944	0.941	0.933
	synapse1.1-synapse1.2	63.281	63.636	63.636	63.636	0.452	0.706	0.667	0.706
	velocity1.5-velocity1.6	55.895	76	88	76	0.421	0.8	0.833	0.8
	xalan2.5-xalan2.6	67.797	73.81	85.714	73.81	0.646	0.909	0.921	0.909
	xerces1.3-xerces1.4	50.34	78.947	73.684	76.316	0.919	1	1	1
	eclipse2.0-eclipse2.1	70.449	65.779	62.91	65.779	0.214	0.586	0.554	0.586
	eclipse2.0-eclipse3.0	72.161	61.937	61.392	61.937	0.301	0.592	0.578	0.592
	eclipse2.1-eclipse3.0	72.406	62.892	62.756	62.892	0.3	0.665	0.673	0.665
Intra	eclipse JDT CORE	81.225	89.902	86.569	89.412	0.868	0.892	0.962	0.91
	eclipse PDE UI	75.529	77.667	93.667	77.667	0.798	0.87	1	0.867
	equinox framework	73.704	95	94.833	95.167	0.741	0.938	0.943	0.957
	lucene	78.759	87.333	87.167	87.333	0.826	0.923	0.907	0.907
	mylyn	74.725	74.824	69.045	72.703	0.804	0.769	0.71	0.756
	eclipse2.0	72.011	69.765	69.765	69.121	0.774	0.693	0.671	0.674
	eclipse2.1	69.099	66.957	62.853	66.606	0.746	0.739	0.681	0.716
	eclipse3.0	70.067	67.208	65.407	67.742	0.741	0.656	0.66	0.657

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.13: Performance of Support Vector Machine in terms of Recall and F-measure.

	Dataset	Recall				F-measure			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-1nt1.7	0.687	0.677	0.968	0.677	0.538	0.575	0.652	0.575
	camel1.4-camel1.6	0.521	0.706	0.735	0.794	0.408	0.696	0.595	0.761
	ivy1.4-ivy2.0	0.175	0.737	0.737	0.684	0.151	0.7	0.622	0.703
	poi-2.5-poi3.0	0.644	1	0.941	0.824	0.688	0.971	0.941	0.875
	synapse1.1-synapse1.2	0.442	0.632	0.737	0.632	0.447	0.667	0.7	0.667
	velocity1.5-velocity1.6	0.782	0.8	1	0.8	0.547	0.8	0.909	0.8
	xalan2.5-xalan2.6	0.679	0.789	0.921	0.789	0.662	0.845	0.921	0.845
	xerces1.3-xerces1.4	0.364	0.742	0.677	0.71	0.521	0.852	0.808	0.83
	eclipse2.0-eclipse2.1	0.648	0.684	0.689	0.684	0.322	0.631	0.614	0.631
	eclipse2.0-eclipse3.0	0.668	0.598	0.647	0.598	0.415	0.595	0.611	0.595
	eclipse2.1-eclipse3.0	0.647	0.417	0.397	0.417	0.41	0.513	0.499	0.513
Intra	eclipse JDT CORE	0.745	0.913	0.767	0.9	0.801	0.89	0.817	0.893
	eclipse PDE UI	0.709	0.671	0.871	0.671	0.749	0.747	0.928	0.752
	equinox framework	0.794	0.967	0.971	0.962	0.761	0.951	0.953	0.956
	lucene	0.736	0.833	0.85	0.85	0.778	0.872	0.87	0.868
	mylyn	0.667	0.723	0.695	0.695	0.729	0.744	0.695	0.719
	eclipse2.0	0.628	0.7	0.727	0.7	0.693	0.689	0.696	0.686
	eclipse2.1	0.61	0.479	0.388	0.479	0.67	0.568	0.482	0.563
	eclipse3.0	0.632	0.669	0.591	0.675	0.682	0.659	0.619	0.664

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians.

Table 5.14 - Table 5.19 show the experimental results obtained for number of faults of software fault prediction. Three classifiers used are Linear Regression, Decision Tree Regression and Multilayer Perceptron. Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC). "Without Aggregation method" is compared with all eight other aggregation techniques in this section. Following observations can be made from these tables.

5.7.3 Inter-release experiments for Number of Faults Prediction

For Promise datasets, it can be observed from Table 5.14- Table 5.19 that QM_MED gives best results in terms of MOC when Linear Regression and Decision Tree Regression classifiers are used.

For Eclipse dataset, it can be observed from Table 5.14- Table 5.19 that QM_AVG gives the best results in terms of AAE, ARE and pred(l) while QM_MED gives the best results in terms of MOC, when Multilayer Perceptron is used.

Thus, QM_MED outperforms other aggregation techniques in the above mentioned scenarios for Inter-release experiments in predicting the number of faults in software fault prediction.

Table 5.14: Performance of Linear Regression in terms of AAE and ARE.

	Dataset	Average Absolute Error				Average Relative Error			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	0.622	0.47	0.403	1.879	0.473	0.401	0.372	1.412
	camel1.4-camel1.6	1.038	0.324	0.431	1.334	0.776	0.273	0.405	1.015
	ivy1.4-ivy2.0	0.422	0.236	0.368	0.944	0.336	0.21	0.352	0.729
	poi-2.5-poi3.0	0.857	1.348	1.421	5.392	0.43	0.895	0.993	1.948
	synapse1.1-synapse1.2	0.746	0.786	0.965	3.119	0.503	0.579	0.773	1.761
	velocity1.5-velocity1.6	1.046	1.112	1.628	4.436	0.748	0.779	1.111	2.076
	xalan2.5-xalan2.6	0.667	0.658	0.511	2.615	0.43	0.354	0.31	0.768
	xerces1.3-xerces1.4	2.339	2.581	1.584	10.287	0.533	0.699	0.464	1.184
	eclipse2.0-eclipse2.1	0.621	0.248	0.378	0.979	0.569	0.216	0.355	0.757
	eclipse2.0-eclipse3.0	0.634	0.237	0.358	0.931	0.538	0.191	0.329	0.641
	eclipse2.1-eclipse3.0	0.617	0.255	0.434	1.033	0.517	0.209	0.406	0.747
Intra	eclipse JDT CORE	0.645	1.906	0.417	0.382	0.159	0.296	0.615	0.216
	eclipse PDE UI	0.602	0.206	0.428	0.349	0.158	2.588	0.597	0.234
	equinox framework	0.615	0.216	5.283	0.378	0.177	0.374	0.551	0.246
	lucene	0.597	0.234	0.647	0.353	0.177	0.472	0.681	6.003
	mylyn	0.551	0.246	1.019	0.392	2.496	1.473	0.603	0.206
	eclipse2.0	0.681	6.003	2.776	0.38	0.162	0.483	0.718	0.251
	eclipse2.1	0.603	0.206	0.79	0.415	0.171	0.472	5.791	3.558
	eclipse3.0	0.718	0.251	0.835	2.872	1.739	0.554	0.162	0.296

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians, AAE=Average Absolute Error, ARE=Average Relative Error.

Table 5.15: Performance of Linear Regression in terms of Pred(l) and Measure of Completeness.

	Dataset	Pred(l)				Measure of Completeness			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	46.174	59.701	68.657	25.373	153.776	177.277	253.5	177.277
	camel1.4-camel1.6	28.083	63.2	54.4	20	186.747	149.667	330.684	153.3
	ivy1.4-ivy2.0	46.307	75	61.538	40.385	153.248	25.242	33.153	25.357
	poi-2.5-poi3.0	52.262	50	35	40	91.785	245.297	166.921	245.297
	synapse1.1-synapse1.2	33.594	36.364	36.364	12.121	118.085	105.668	162.884	106.554
	velocity1.5-velocity1.6	29.694	44	48	28	154.957	144.25	951.725	147.403
	xalan2.5-xalan2.6	33.672	54.762	66.667	30.952	95.776	51.49	113.644	52.091
	xerces1.3-xerces1.4	32.313	18.421	36.842	5.263	31.297	-5.514	12.285	-5.853
	eclipse2.0-eclipse2.1	22.896	79.098	43.443	25	440.173	199.173	515.566	196.378
	eclipse2.0-eclipse3.0	25.979	82.265	46.385	30.559	260.38	137.941	383.013	134.686
	eclipse2.1-eclipse3.0	10.649	84.993	26.739	18.281	235.6	157.318	501.037	159.132
Intra	eclipse JDT CORE	51.384	23.667	53.111	95.844	99.203	94.815	56.199	88.106
	eclipse PDE UI	45.934	86.381	51.894	97.798	98.781	140.256	40.718	85.814
	equinox framework	56.199	88.106	21.176	93.93	103.557	90.212	43.117	84.588
	lucene	40.718	85.814	59.868	92.484	99.359	98.607	42.788	10.465
	mylyn	43.117	84.588	47.803	93.081	135.615	123.989	45.305	87.043
	eclipse2.0	42.788	10.465	28	93.618	98.015	100.697	42.007	81.857
	eclipse2.1	45.305	87.043	43.524	92.934	93.211	99.878	20.588	24.5
	eclipse3.0	42.007	81.857	44.258	150.539	130.945	101.775	91.875	73.439

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians, Pred(l)=Prediction at level "l".

Table 5.16: Performance of Decision Tree Regression in terms of AAE and ARE.

	Dataset	Average Absolute Error				Average Relative Error			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	0.56	0.325	0.269	1.299	0.395	0.26	0.25	0.867
	camel1.4-camel1.6	0.793	0.234	0.277	1.011	0.522	0.177	0.242	0.631
	ivy1.4-ivy2.0	0.312	0.144	0.117	0.588	0.24	0.125	0.101	0.449
	poi-2.5-poi3.0	0.828	0.51	0.538	2.042	0.396	0.364	0.372	0.949
	synapse1.1-synapse1.2	0.734	0.503	0.388	1.574	0.495	0.347	0.291	0.819
	velocity1.5-velocity1.6	1.061	0.548	0.655	2.836	0.76	0.443	0.601	1.636
	xalan2.5-xalan2.6	0.664	0.475	0.498	1.752	0.428	0.241	0.312	0.512
	xerces1.3-xerces1.4	2.413	1.785	1.562	7.322	0.495	0.424	0.437	0.742
	eclipse2.0-eclipse2.1	0.5	0.245	0.268	0.961	0.444	0.21	0.239	0.703
	eclipse2.0-eclipse3.0	0.529	0.234	0.251	1.018	0.421	0.187	0.217	0.692
	eclipse2.1-eclipse3.0	0.442	0.245	0.3	0.972	0.319	0.193	0.263	0.67
Intra	eclipse JDT CORE	0.453	1.254	0.212	0.212	0.087	0.138	0.483	0.161
	eclipse PDE UI	0.364	0.114	0.204	0.28	0.116	1.738	0.297	0.228
	equinox framework	0.483	0.161	3.954	0.181	0.174	0.291	0.294	0.188
	lucene	0.297	0.228	0.534	0.176	0.136	0.284	0.659	4.002
	mylyn	0.294	0.188	0.646	0.379	1.432	0.484	0.339	0.122
	eclipse2.0	0.659	4.002	1.493	0.197	0.095	0.259	0.508	0.203
	eclipse2.1	0.339	0.122	0.448	0.27	0.129	0.38	3.807	3.114
	eclipse3.0	0.508	0.203	0.687	1.553	1.169	0.505	0.126	0.12

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians, AAE=Average Absolute Error, ARE=Average Relative Error.

5.7.4 Intra-release experiments for Number of Faults Prediction

For Eclipse datasets, it can be observed from Table 5.14- Table 5.19 that AAD gives the best results in terms of all four performance evaluation measures used for Linear Regression and Multilayer Perceptron classifiers.

For four Eclipse datasets and one Apache dataset, it can be observed from Table 5.14- Table 5.19 that AAD gives the best results in general, in terms of ARE and pred(l) when Linear Regression and Multilayer Perceptron classifiers are used. Summation gives the best results in terms of MOC, for all the three classifiers used.

Thus, AAD outperforms other techniques in the above mentioned scenarios, in intra-release experiments for predicting the number of faults.

Table 5.17: Performance of Decision Tree Regression in terms of Pred(l) and Measure of Completeness.

	Dataset	Pred(l)				Measure of Completeness			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	53.02	64.179	76.119	37.313	135.099	114.46	269.292	114.46
	camel1.4-camel1.6	47.047	84	77.6	39.2	126.583	111.128	251.096	128.46
	ivy1.4-ivy2.0	68.75	86.538	96.154	50	140.683	158.58	76.408	154.18
	poi-2.5-poi3.0	52.489	60	50	45	89.381	103.668	99.055	104.079
	synapse1.1-synapse1.2	32.031	51.515	63.636	45.455	121.92	79.43	94.024	97.264
	velocity1.5-velocity1.6	31.878	32	28	16	153.818	218.403	486.341	249.898
	xalan2.5-xalan2.6	31.864	71.429	47.619	45.238	95.981	65.722	77.603	66.506
	xerces1.3-xerces1.4	30.782	36.842	39.474	5.263	23.879	19.892	17.949	19.277
	eclipse2.0-eclipse2.1	48.783	80.328	68.852	38.525	348.461	185.614	322.903	183.274
	eclipse2.0-eclipse3.0	49.929	84.584	73.124	39.154	206.412	135.108	250.574	141.228
	eclipse2.1-eclipse3.0	60.436	76.808	66.985	26.739	137.736	126.46	300.134	138.882
Intra	eclipse JDT CORE	73.698	60.5	86.97	93.439	94.575	93.799	65.347	93.515
	eclipse PDE UI	76.787	96.131	89.617	97.96	96.066	141.313	81.85	87.015
	equinox framework	65.347	93.515	18.922	92.541	104.753	90.128	82.848	89.461
	lucene	81.85	87.015	66.118	92.489	96.469	99.316	45.8	22.364
	mylyn	82.848	89.461	83.561	93.532	139.044	108.528	80.09	96.9
	eclipse2.0	45.8	22.364	58.333	93.498	101.55	97.602	70.15	87.571
	eclipse2.1	80.09	96.9	72.357	93.08	95.172	101.996	18.431	30.167
	eclipse3.0	70.15	87.571	57.045	120.768	121.361	101.863	97.5	96.004

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians, Pred(l)=Prediction at level "l".

Table 5.18: Performance of Multilayer Perceptron in terms of AAE and ARE.

	Dataset	Average Absolute Error				Average Relative Error			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	0.639	0.474	0.18	1.896	0.449	0.393	0.149	1.345
	camel1.4-camel1.6	0.945	0.309	0.374	1.327	0.666	0.256	0.348	0.954
	ivy1.4-ivy2.0	0.358	0.116	0.101	0.466	0.282	0.093	0.084	0.292
	poi-2.5-poi3.0	0.865	0.692	0.61	3.195	0.394	0.436	0.441	1.081
	synapse1.1-synapse1.2	0.689	0.447	0.538	2.049	0.411	0.297	0.421	0.911
	velocity1.5-velocity1.6	1.065	0.623	0.815	2.68	0.609	0.516	0.711	1.778
	xalan2.5-xalan2.6	0.679	0.565	0.494	2.356	0.394	0.31	0.318	0.733
	xerces1.3-xerces1.4	2.242	1.877	1.837	7.358	0.673	0.446	0.564	0.607
	eclipse2.0-eclipse2.1	0.703	0.192	0.468	0.769	0.655	0.155	0.448	0.51
	eclipse2.0-eclipse3.0	0.714	0.208	0.445	0.837	0.622	0.155	0.42	0.489
	eclipse2.1-eclipse3.0	0.815	0.277	0.693	1.094	0.729	0.231	0.673	0.812
Intra	eclipse JDT CORE	0.915	1.906	0.417	0.388	0.159	0.296	0.636	0.216
	eclipse PDE UI	0.589	0.206	0.428	0.419	0.158	2.588	0.673	0.234
	equinox framework	0.636	0.216	5.283	0.503	0.177	0.374	0.54	0.246
	lucene	0.673	0.234	0.647	0.289	0.177	0.472	0.654	6.003
	mylyn	0.54	0.246	1.019	0.343	2.496	1.473	0.595	0.206
	eclipse2.0	0.654	6.003	2.776	0.322	0.162	0.483	0.71	0.251
	eclipse2.1	0.595	0.206	0.79	0.32	0.171	0.472	5.791	3.558
	eclipse3.0	0.71	0.251	0.835	2.872	1.739	0.554	0.162	0.296

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians, AAE=Average Absolute Error, ARE=Average Relative Error.

Table 5.19: Performance of Multilayer Perceptron in terms of Pred(l) and Measure of Completeness.

	Dataset	Pred(l)				Measure of Completeness			
		w/o agg	QM_AVG	QM_MED	QM_SUM	w/o agg	QM_AVG	QM_MED	QM_SUM
Inter	ant1.6-ant1.7	43.221	64.179	82.09	17.91	86.926	206.069	119.361	206.069
	camel1.4-camel1.6	42.487	74.4	56.8	23.2	147.635	127.742	405.26	147.731
	ivy1.4-ivy2.0	65.057	92.308	88.462	67.308	144.034	36.911	200.559	103.821
	poi-2.5-poi3.0	47.964	40	55	15	86.791	66.021	91.534	39.831
	synapse1.1-synapse1.2	50.781	60.606	42.424	27.273	68.812	109.844	145.04	91.518
	velocity1.5-velocity1.6	37.555	52	28	16	30.764	134.531	422.898	167.278
	xalan2.5-xalan2.6	25.085	59.524	54.762	30.952	75.911	74.014	92.077	74.897
	xerces1.3-xerces1.4	35.034	34.211	13.158	26.316	65.009	12.568	-9.956	13.976
	eclipse2.0-eclipse2.1	8.608	88.73	29.713	37.705	503.29	84.393	700.386	83.313
	eclipse2.0-eclipse3.0	10.932	88.404	36.153	35.88	299.176	51.164	538.412	48.502
	eclipse2.1-eclipse3.0	10.299	93.724	16.508	21.692	338.3	176.61	873.412	173.544
Intra	eclipse JDT CORE	37.983	23.667	53.111	109.237	99.203	94.815	39.731	88.106
	eclipse PDE UI	47.748	86.381	51.894	114.796	98.781	140.256	41.951	85.814
	equinox framework	39.731	88.106	21.176	136.778	103.557	90.212	54.802	84.588
	lucene	41.951	85.814	59.868	47.72	99.359	98.607	47.624	10.465
	mylyn	54.802	84.588	47.803	73.145	135.615	123.989	49.513	87.043
	eclipse2.0	47.624	10.465	28	60.032	98.015	100.697	51.539	81.857
	eclipse2.1	49.513	87.043	43.524	48.363	93.211	99.878	20.588	24.5
	eclipse3.0	51.539	81.857	44.258	150.539	130.945	101.775	91.875	73.439

* w/o agg.=Without Aggregation, QM_AVG =Average of Quarter Medians, QM_MED=Median of Quarter Medians, QM_SUM= Sum of Quarter Medians, Pred(l)=Prediction at level "l".

5.8 OBSERVATIONS

Table 5.4- Table 5.13 show the experimental results obtained for binary classification of software fault prediction. Five classifiers used are Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure.

Table 5.14 - Table 5.19 show the experimental results obtained for number of faults of software fault prediction. Three classifiers used are Linear Regression (LNR), Decision Tree Regression (DTR) and Multilayer Perceptron (MLP). Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC).

A comparative analysis of "without aggregation method" and all eight other aggregation techniques ,i.e., AAD, IQR, MAD, MED, SUM, QM_AVG, QM_MED and QM_SUM is done.

5.8.1 Inter-release Binary Classification

For Promise datasets, it can be observed from Table 5.4- Table 5.13 that QM_MED gives the best results for Recall (in 40% cases) and F-measure (in 40% cases) when Logistic Regression and

Support Vector Machine classifiers are used. QM_AVG outperforms other techniques in terms of Accuracy (15% case), Recall (15.38% case) and F-measure (22.22% case) when Random Forest is used. It also outperforms other techniques in terms of Accuracy (20% case) and F-measure (22.22% case) when Support Vector Machine is used.

The range of accuracy is 44.77% to 95%, range of precision is 0.13 to 1, range of recall is 0.175 to 1 and range of F-measure is 0.15 to 0.97 for SVM.

For Eclipse datasets, it can be observed from Table 5.4- Table 5.13 that no aggregation techniques could outperform "without aggregation method" in terms of Accuracy. Summation gives the best results for all the classifiers used in terms of precision performance evaluation measure in 100% cases. MED and QM_MED give comparable results and outperform all other techniques when Naive Bayes classifier is used, in terms of Recall (50% cases) and F-measure (25% cases).

The range of accuracy is 52.04% to 85.02%, range of precision is 0.31 to 0.82, range of recall is 0.24 to 0.88 and range of F-measure is 0.31 to 0.64 for NB.

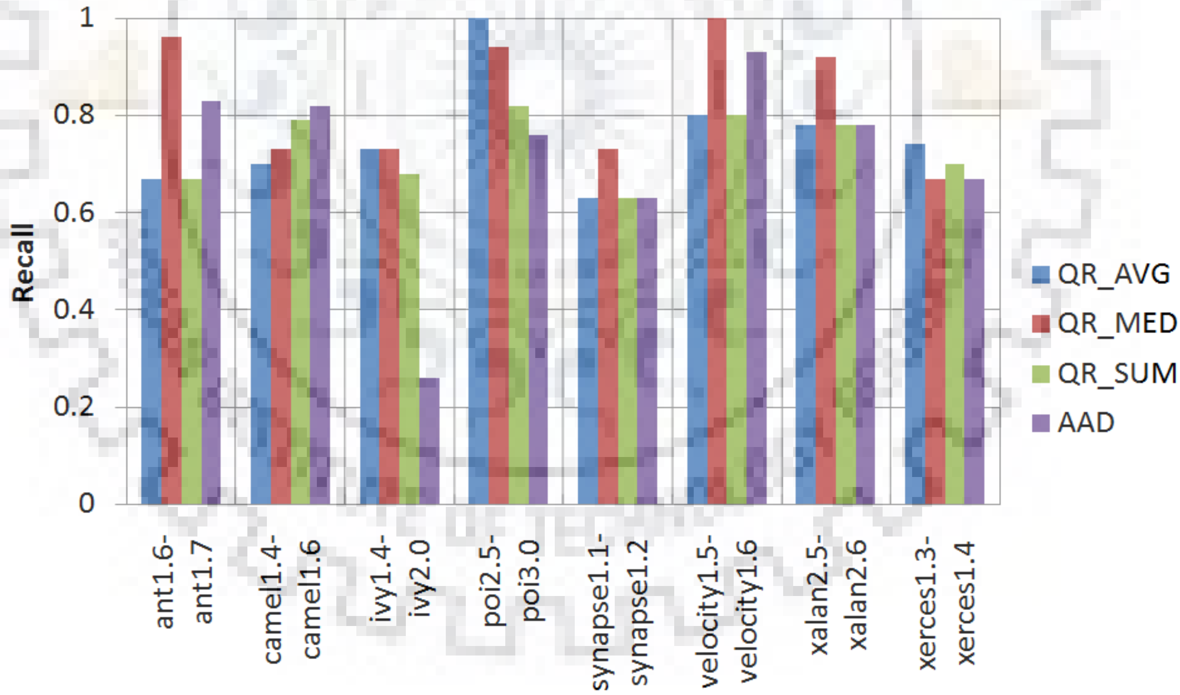

Figure 5.1: Comparative analysis of QM_AVG,QM_MED and QM_SUM with AAD using SVM and Recall,for Binary classification in Inter-Release Experiments.

Comparative analysis of QM_AVG, QM_MED and QM_SUM using SVM and Recall, for Binary classification in Inter-Release Experiments is done with the best technique amongst the five existing aggregation techniques used ,i.e., AAD in Figure 5.1. It can be observed from this figure that QM_MED aggregation technique shows the best performance amongst the other techniques in comparison.

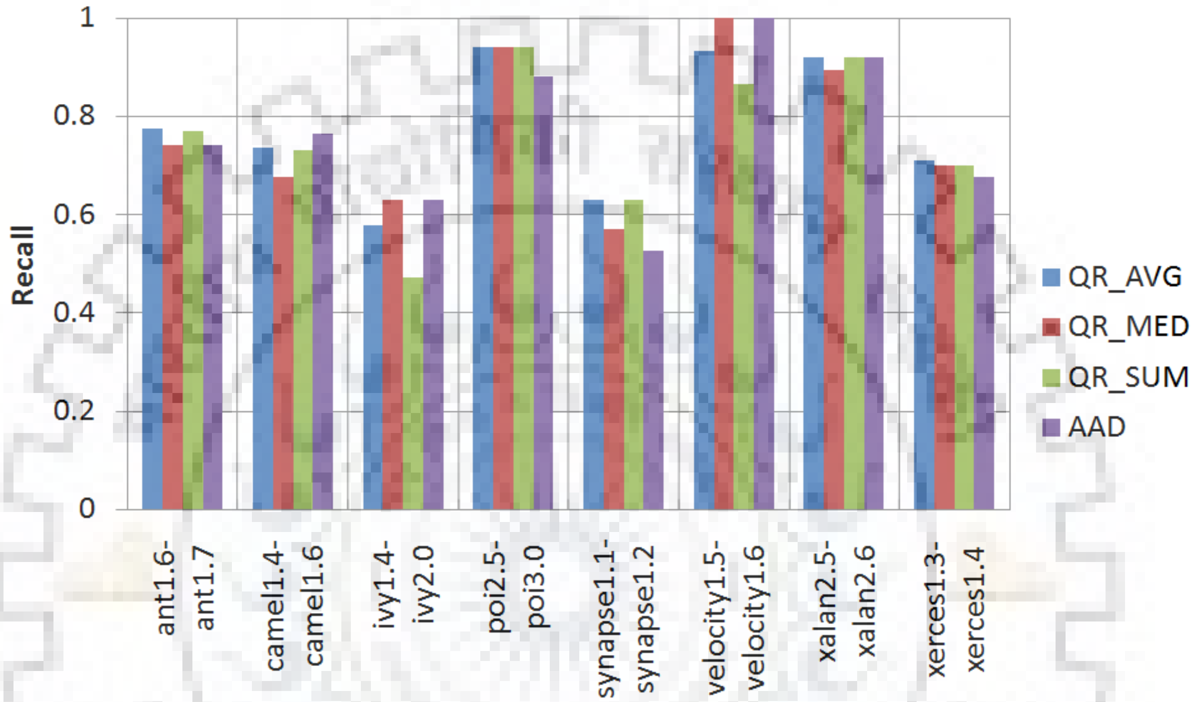

Figure 5.2: Comparative analysis of QM_AVG,QM_MED and QM_SUM with AAD using RF and Recall,for Binary classification in Inter-Release Experiments.

Comparative analysis of QM_AVG, QM_MED and QM_SUM using RF and Recall, for Binary classification in Inter-Release Experiments is done with the best technique amongst the five existing aggregation techniques used ,i.e., AAD in Figure 5.2. It can be observed from this figure that QM_AVG aggregation technique shows the best performance amongst the other techniques in comparison.

5.8.2 Intra-release Binary Classification

For Eclipse datasets, it can be observed from Table 5.4- Table 5.13 that QM_MED gives the best results in terms of Recall (66.66% cases) and F-measure (33.33% cases) when Naive Bayes classifier is used. The range of accuracy is 51.4% to 67.63%, range of precision is 0.48 to 0.83, range of recall is 0.25 to 0.90 and range of F-measure is 0.38 to 0.69 for NB.

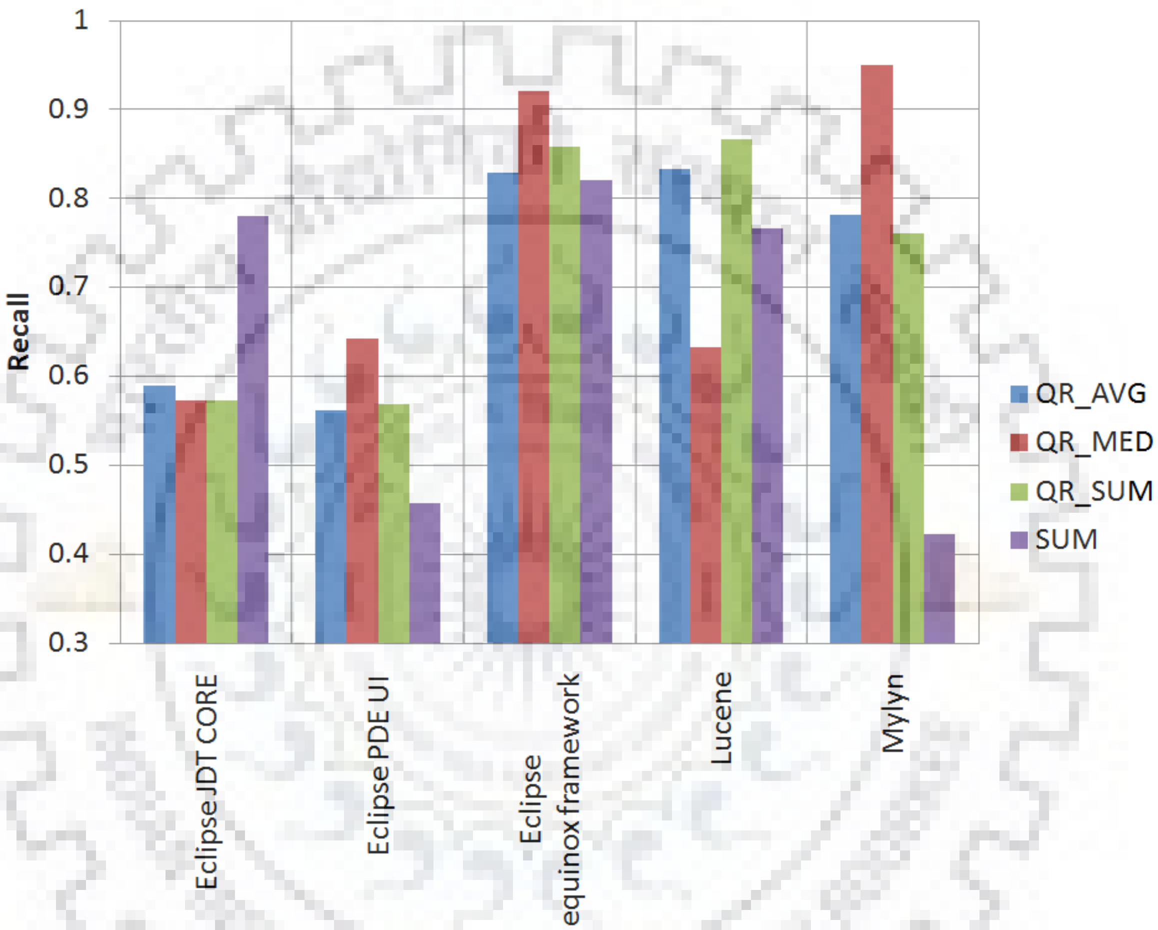

Figure 5.3: Comparative analysis of QM_AVG,QM_MED and QM_SUM with SUM using NB and Recall,for Binary classification of Intra-Release Experiments.

For four Eclipse and one Apache dataset, it can be observed from Table 5.4- Table 5.13 that QM_MED gives the best results in terms of Recall (40% cases) and F-measure (40% cases) when Naive Bayes classifier is used. It also gives the best results in terms of Precision (40% cases), Recall (20% cases) and F-measure (20% cases) when Support Vector Machine classifier is used. The range of accuracy is 55% to 89.5%, range of precision is 0.54 to 1, range of recall is 0.16 to

1 and range of F-measure is 0.25 to 0.89 for NB.

Comparative analysis of QM_AVG, QM_MED and QM_SUM using NB and Recall, for Binary classification in Intra-Release Experiments is done with the best technique amongst the five existing aggregation techniques used ,i.e., SUM in Figure 5.3. It can be observed from this figure that QM_MED aggregation technique shows the best performance amongst the other techniques in comparison.

5.8.3 Inter-release experiments for Number of Faults Prediction

For Promise datasets, it can be observed from Table 5.14- Table 5.19 that QM_MED gives best results in terms of MOC when Linear Regression (in 33.3% cases) and Decision Tree Regression (in 37.5% cases) classifiers are used. The range of AAE is 0.01 to 37.29, range of ARE is 0.01 to 2.13, range of pred(l) is 4 to 96.15 and range of MOC is 0 to 486.34 for DTR.

For Eclipse dataset, it can be observed from Table 5.14- Table 5.19 that QM_AVG gives the best results in terms of AAE (66.66% cases), ARE (66.66% cases) and pred(l) (100% cases) while QM_MED gives the best results in terms of MOC (66.66% cases), when Multilayer Perceptron is used. The range of AAE is 0.19 to 3.66, range of ARE is 0.15 to 1.06, range of pred(l) is 8.6 to 93.72 and range of MOC is -117 to 873 for MLP.

Comparative analysis of QM_AVG, QM_MED and QM_SUM using MLP and AAE, for predicting number of faults in Inter-Release Experiments is done with the best technique amongst the five existing aggregation techniques used ,i.e., AAD in Figure 5.4. It can be observed from this figure that QM_AVG aggregation technique shows the best performance amongst the other techniques in comparison.

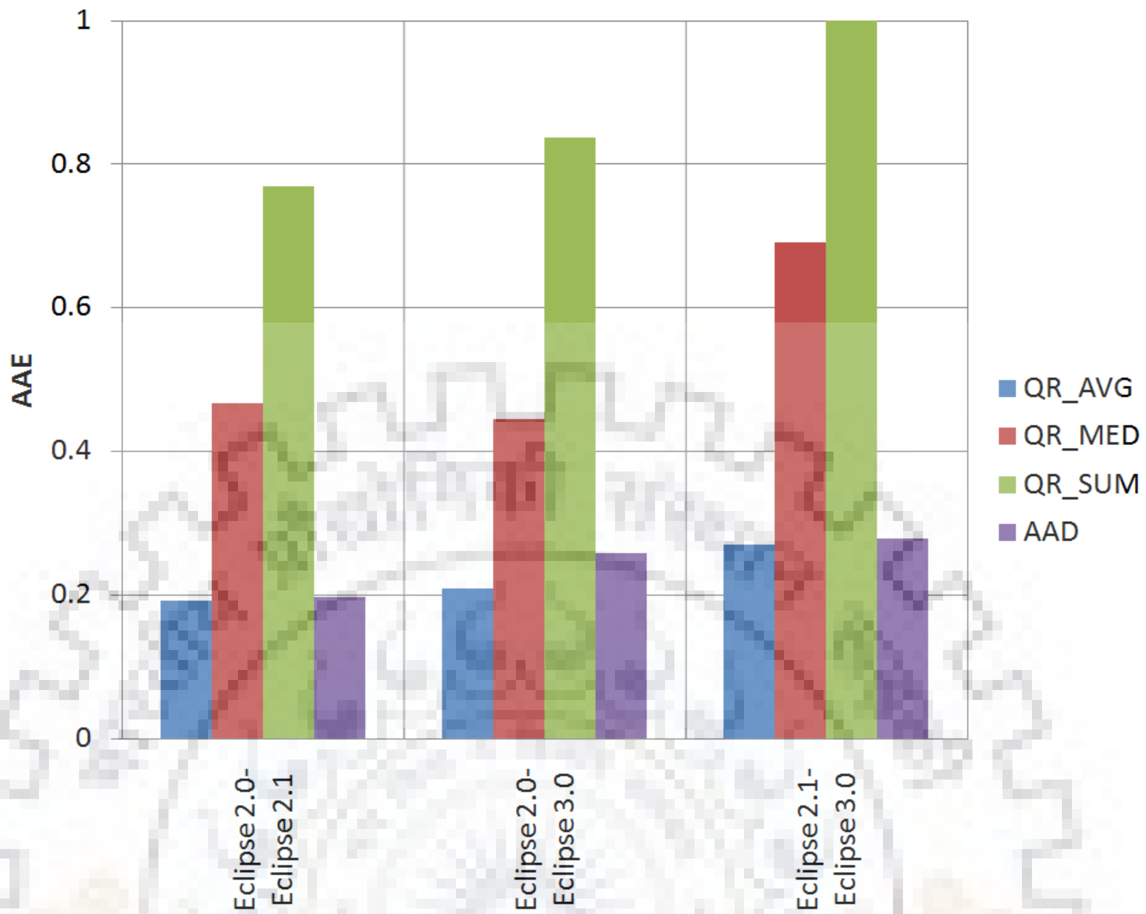

Figure 5.4: Comparative analysis of QM_AVG,QM_MED and QM_SUM with AAD using MLP and AAE,for Inter-Release Experiments in predicting number of faults.

5.8.4 Intra-release experiments for Number of Faults Prediction

For Eclipse datasets, it can be observed from Table 5.14- Table 5.19 that AAD gives the best results in terms of all four performance evaluation measures used in 100% cases for Linear Regression and Multilayer Perceptron classifiers.

The range of AAE is 0.17 to 4.18, range of ARE is 0.13 to 1.88, range of pred(l) is 14.34 to 93.07 and range of MOC is 48.36 to 118 for MLP.

For four Eclipse datasets and one Apache dataset, it can be observed from Table 5.14- Table 5.19 that AAD gives the best results in general, in terms of ARE and pred(l) when Linear Regression and Multilayer Perceptron classifiers are used (in 50% cases). Summation gives the best results (in 75% cases) in terms of MOC, for all the three classifiers used.

The range of AAE is 0.09 to 6, range of ARE is 0.06 to 2.87, range of pred(l) is 10.46 to 98.57 and range of MOC is 47.71 to 192.25 for MLP.

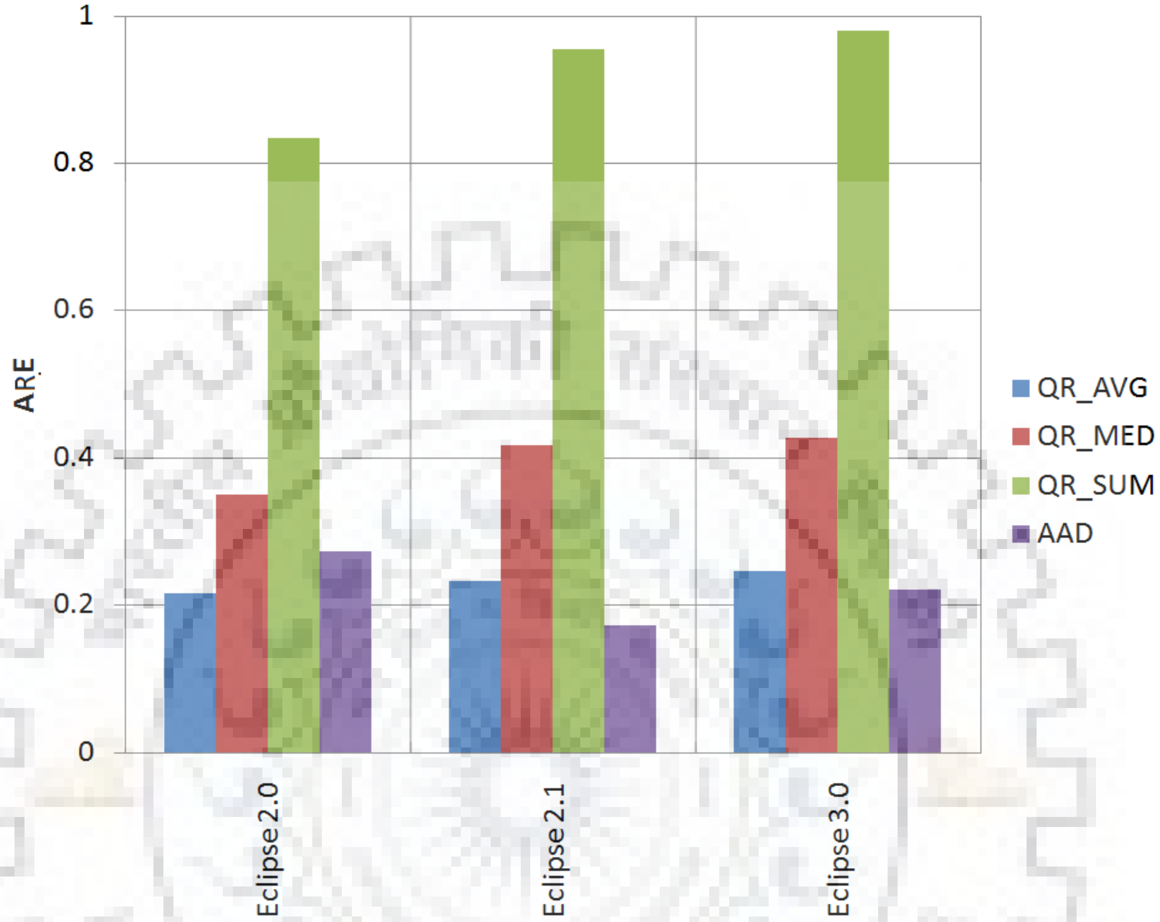

Figure 5.5: Comparative analysis of QM_AVG,QM_MED and QM_SUM with AAD using MLP and ARE,for Intra-Release Experiments in predicting number of faults.

Comparative analysis of QM_AVG, QM_MED and QM_SUM using MLP and ARE, for predicting number of faults in Intra-Release Experiments is done with the best technique amongst the five existing aggregation techniques used ,i.e., AAD in Figure 5.5. It can be observed from this figure that AAD aggregation technique shows the best performance amongst the other techniques in comparison.

5.9 CONCLUSION

For binary classification in software fault prediction, QM_MED and QM_AVG aggregation techniques outperform "without aggregation" and other aggregation techniques in the scenarios mentioned. Five classifiers used are Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure. For predicting number of faults in software fault prediction, QM_MED and AAD aggregation techniques outperform "without aggregation" and other aggregation techniques in the scenarios mentioned. Three classifiers used are Linear Regression (LNR), Decision Tree Regression (DTR) and Multilayer Perceptron (MLP). Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC).

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

Software fault prediction is the mechanism to predict the fault proneness of a module before testing mechanism is applied. Software fault prediction can be either binary classification fault prediction or can be prediction of the number of faults in the software.

For binary classification of software fault prediction, five classifiers used are Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Random Forest and Support Vector Machine. Performance evaluation measure used are Accuracy, Precision, Recall and F-measure. For predicting the number of faults in software fault prediction, three classifiers used are Linear Regression (LNR), Decision Tree Regression (DTR) and Multilayer Perceptron (MLP). Performance evaluation measure used are Average Absolute Error (AAE), Average Relative Error (ARE), Prediction at level "l" (pred(l)) and Measure of Completeness (MOC). Publicly available Promise datasets, Apache dataset and Eclipse datasets have been used. A comparative analysis of "without aggregation method" and all eight other aggregation techniques ,i.e., AAD, IQR, MAD, MED, SUM, QM_AVG, QM_MED and QM_SUM is done.

6.1 CONCLUSIONS

Aggregation may need to be performed in inter-releases and cross project prediction scenarios where the granularity of the training dataset and the target testing dataset is of different level.

• Five existing aggregation techniques have been explored in this work ,i.e., (AAD), Median Absolute Deviation (MAD), Interquartile Range (IQR), Median (MED) and Summation (SUM).

Out of these five techniques, AAD, IQR and MAD have not been used so far in the field of software fault prediction. From the experimental analysis, it is observed that the performance of software fault prediction is comparable with "without aggregation method" or even improved after applying the aggregation of metrics. Considering "without aggregation method" and all the five existing aggregation techniques used in this work, AAD and SUM gave better performances.

• In this work, Average of Quarter Medians (QM_AVG), Median of Quarter Medians (QM_MED) and Sum of Quarter Medians (QM_SUM) are the three novel techniques explored that have not been explored so far in any of the fields. Eight aggregation techniques ,i.e., Average Absolute Deviation (AAD), Median Absolute Deviation (MAD), Interquartile Range (IQR), Median (MED), Summation (SUM), Average of Quarter Medians (QM_AVG), Median of Quarter Medians (QM_MED) and Sum of Quarter Medians (QM_SUM) are investigated for their effect on the software fault prediction and "without aggregation technique" is also compared with them. The performance of fault prediction mechanism using aggregation techniques have shown comparable and even better performance as compared to the performance of fault prediction mechanism when no aggregation method was used. QM_AVG and QM_MED gave better performance amongst "without aggregation method" and all the eight other aggregation techniques used in this work.

The performance of aggregation techniques vary on using different classifiers and different datasets.

6.2 FUTURE WORK

Following are some of the possible areas where this work can be further explored.

• In future, more datasets can be used on which this work can be replicated to check the consistency of the results obtained.

• Apart from the eight machine learning techniques used in this work, some other advanced techniques could be used to see the difference in the performance of the fault prediction mechanism.

• Some other existing or new aggregation techniques could be thought of to compare their performances with the performance of the aggregation techniques used in this work.

• Ensemble of machine learning techniques could be explored to see what impact it will have on the performance of the software fault prediction mechanism.

REFERENCES

[1] S. S. Rathore and S. Kumar, “Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems,” *Knowledge-Based Systems*, vol. 119, pp. 232–256, 2017.

[2] Ö. F. Arar and K. Ayan, “Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies,” *Expert Systems with Applications*, vol. 61, pp. 106–121, 2016.

[3] Y. Kamei and E. Shihab, “Defect prediction: Accomplishments and future challenges,” in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 5. IEEE, 2016, pp. 33–45.

[4] A. G. Koru and H. Liu, “Building effective defect-prediction models in practice,” *IEEE software*, vol. 22, no. 6, pp. 23–29, 2005.

[5] D. Posnett, V. Filkov, and P. Devanbu, “Ecological inference in empirical software engineering,” in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 362–371.

[6] F. Zhang, A. E. Hassan, S. McIntosh, and Y. Zou, “The use of summation to aggregate software metrics hinders the performance of defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 43, no. 5, pp. 476–491, 2017.

[7] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Proceedings of the third international workshop on predictor models in software engineering*. IEEE Computer Society, 2007, p. 9.

[8] K. Herzig, “Using pre-release test failures to build early post-release defect prediction models,” in *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*. IEEE, 2014, pp. 300–311.

[9] B. Vasilescu, A. Serebrenik, and M. van den Brand, “You can’t control the unfamiliar: A study on the relations between aggregation techniques for software metrics,” in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 2011, pp. 313–322.

[10] A. Serebrenik and M. van den Brand, “Theil index for aggregation of software metrics values,” in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–9.

[11] K. Mordal-Manet, J. Laval, S. Ducasse, N. Anquetil, F. Balmas, F. Bellingard, L. Bouhier, P. Vaillergues, and T. J. McCabe, “An empirical model for continuous and weighted metric aggregation,” in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, 2011, pp. 141–150.

[12] B. Walter, M. Wolski, P. Prominski, and S. Kupiński, “One metric to combine them all: experimental comparison of metric aggregation approaches in software quality models,” in *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2016 Joint Conference of the International Workshop on*. IEEE, 2016, pp. 159–163.

[13] I. Ivan, A. Zamfiroiu, M. Doinea, and M. L. Despa, “Assigning weights for quality software metrics aggregation,” *Procedia Computer Science*, vol. 55, pp. 586–592, 2015.

[14] J. Sanz-Rodriguez, J. M. Dodero, and S. Sanchez-Alonso, “Metrics-based evaluation of learning object reusability,” *Software Quality Journal*, vol. 19, no. 1, pp. 121–140, 2011.

[15] R. Vasa, M. Lumpe, P. Branch, and O. Nierstrasz, “Comparative analysis of evolving software systems using the gini coefficient,” in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 179–188.

[16] X.-L. Sun, H.-L. Wang, Y.-G. Zhao, C. Zhang, and G.-L. Zhang, “Digital soil mapping based on wavelet decomposed components of environmental covariates,” *Geoderma*, vol. 303, pp. 118–132, 2017.

[17] U. G. Sefercik and A. Atesoglu, "Three-dimensional forest stand height map production utilizing airborne laser scanning dense point clouds and precise quality evaluation," *iForest-Biogeosciences and Forestry*, vol. 10, no. 2, p. 491, 2017.

[18] J. Jeong, E. Park, W. S. Han, K. Kim, S. Choung, and I. M. Chung, "Identifying outliers of non-gaussian groundwater state data based on ensemble estimation for long-term trends," *Journal of Hydrology*, vol. 548, pp. 135–144, 2017.

[19] S. S. Ghannadpour and A. Hezarkhani, "Comparing u-statistic and nonstructural methods for separating anomaly and generating geochemical anomaly maps of cu and mo in parkam district, kerman, iran," *Carbonates and evaporites*, vol. 32, no. 2, pp. 155–166, 2017.

[20] C. Pirlet, L. Pierard, V. Legrand, and O. Gach, "Ratio of high-sensitivity troponin to creatine kinase-mb in takotsubo syndrome," *International journal of cardiology*, vol. 243, pp. 300–305, 2017.

[21] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[22] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 91–100.

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[24] T. Menzies, R. Krishna, and D. Pryor, "The promise repository of empirical software engineering data (2015)," 2015. [Online]. Available: http://openscience.us/repo

[25] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on software engineering*, vol. 32, no. 10, pp. 771–789, 2006.

[26] Y. Zhao, Y. Yang, H. Lu, J. Liu, H. Leung, Y. Wu, Y. Zhou, and B. Xu, "Understanding the value of considering client usage context in package cohesion for fault-proneness prediction," *Automated Software Engineering*, vol. 24, no. 2, pp. 393–453, 2017.

[27] 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*, vol. 31, no. 10, pp. 897–910, 2005.

[28] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[29] S. S. Rathore and S. Kumar, "An empirical study of some software fault prediction techniques for the number of faults prediction," *Soft Computing*, vol. 21, no. 24, pp. 7417–7434, 2017.

[30] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.

[31] S. G. MacDonell, "Establishing relationships between specification size and software process effort in case environments," *Information and Software Technology*, vol. 39, no. 1, pp. 35–45, 1997.