



**A HEUSIRTIC APPROACH TO MINE SOFTWARE REVISION
HISTORIES FOR QUALITY DRIVEN SOFTWARE DEVELOPMENT**

A Dissertation Submitted by

Meghna Soni

11010056

To

Lovely School of Computer Science Engineering

In partial fulfillment of requirement for the

Award of the Degree of

Master of Technology in Computer Science

Under the Guidance of

Harshpreet Singh

Assistant Professor

May 2015



LOVELY
PROFESSIONAL
UNIVERSITY

Transforming Education. Transforming India.

School of Computer Science & Engineering

DISSERTATION TOPIC APPROVAL PERFORMANCE

Name of the Student: Meghna Loni Registration No.: 11010056
 Batch: 2010-2015 Roll No: RK2006032
 Session: 9th Sem (2014-2015) Parent Section: K2006
 Details of Supervisor:
 Name: Hareshpreet Singh Designation: A.P.
 U.I.D.: 17478 Qualification: M.E.
 Research Experience: 3.5 years
 SPECIALIZATION AREA: Software Engineering (pick from list of provided specialization areas by DAA)

PROPOSED TOPICS

1. Mining of Software Engineering data to ensure quality driven development.
2. Characterization and classification of software defects based on mining data.
3. Tool to mine data for software repositories.

Signature of Supervisor
Hareshpreet Singh
17478
2/09

PAC Remarks:

Topic 1 is approved.
 Publication can be expected.

APPROVAL OF PAC CHAIRPERSON:

Signature:

Date:

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor

ABSTRACT

Mining software engineering data aims at removing irrelevant data and outliers or noisy data from revision histories of various software applications by using various data mining approaches. It helps in improving software debugging and software maintenance at low cost and effort which prevents software failures and results in high quality of software.

Abundant research has been done on mining software engineering data for detecting error patterns but still software engineering data remains an important topic of research for various academicians due to unintentional violations made by some writers in the software data which leads to software failure. To maintain software through its lifetime, continuous evaluation of software is required, which can be accomplished by mining software engineering data.

An efficient data mining strategy will be applied to source code plugins and method pairs of revision histories of software applications which mine usage patterns and error patterns and finally validate the patterns. The proposed approach will be compared with the existing traditional algorithm in terms of memory acquired and execution time.

The proposed mining algorithm produced an efficient output when compared to Apriori algorithm in terms of memory used and time taken for mining. The number of methods mined by the new proposed algorithm is more as compared to Apriori algorithm. Different java source code files of jEdit versions are mined and it was found that the new proposed algorithm produced efficient results.

CERTIFICATE

This is to certify that Meghna Soni has completed dissertation proposal titled “**A Heuristic Approach to Mine Software Revision Histories for Quality Driven Software Development**” under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. This work has not been submitted elsewhere for any other degree.

The dissertation proposal is fit for the submission and the partial fulfilment of the conditions for the award of M.Tech Computer Science & Engg.

Date:

Signature of Advisor
Name: Harshpreet Singh
UID:17478
Designation: Asst. Professor
School of Computer Science
and Engineering

ACKNOWLEDGEMENT

I would like to express my profound sense of gratitude and respect to all those who helped me throughout the duration of this project. I am highly thankful to **Mr. Harshpreet Singh** for his support, valuable time and advice, guidance, sincere cooperation during the study and in completing the assignment of preparing the said project within the time stipulated.

This period proved for me one of the most productive and knowledgeable experiences of my career. It provided me an opportunity to upgrade my skills as well as sharpen my professional knowledge.

I would like to thank my Parents, God and friends. With their support and well wishes I am able to complete this project in time.

DECLARATION

I hereby declare that the dissertation proposal entitled, “**A HEUSIRTIC APPROACH TO MINE SOFTWARE REVISION HISTORIES FOR QUALITY DRIVEN SOFTWARE DEVELOPMENT**” submitted for the Master of Technology (M.Tech) Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date:

Investigator: Meghna Soni

Registration No: 11010056

TABLE OF CONTENTS

ABSTRACT.....	i
CERTIFICATE.....	iv
ACKNOWLEDGEMENT	v
DECLARATION	ivi
TABLE OF CONTENTS.....	vii
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER 1 INTRODUCTION	1
1.1 Software Engineering Data	1
1.1.1 Types of software repositories.....	1
1.2 Data mining.....	3
1.2.1 Type of data to be mined	4
1.2.2 Data Mining Tasks.....	4
1.3 Software Engineering and Data Mining.....	5
1.3.1 Association Rule and Frequent Pattern Mining.....	7
1.3.2 Classification	Error! Bookmark not defined.
1.3.3 Prediction.....	11
1.3.4 Clustering.....	12
1.3.5 Text Mining.....	14
CHAPTER 2 LITERATURE SURVEY.....	17
CHAPTER 3 PRESENT WORK.....	26
3.1 Objectives of Research.....	26
3.2 Research Plan	26
Chapter 4 RESULTS AND DISCUSSIONS	30
Chapter 5 CONCLUSION AND FUTURE SCOPE	37
REFERENCES.....	38
LIST OF PAPERS.....	41

LIST OF TABLES

Tables	Page No.
Table 1.1: Examples of software repositories	2
Table 1.2: Software Engineering Data, Mining Algorithms and Software Engineering Tasks	7
Table 1.3: Software Engineering Data Mined by Frequent Pattern Technique	7
Table 1.4: Software Engineering Data mined by Classification Techniques	9
Table 1.5: Software Engineering Data mined by Prediction Technique	11
Table 1.6: Software Engineering Data Mined by Clustering Technique	13
Table 1.7: Software Engineering Data mined by Text mining Technique	15
Table 4.1: Number of Methods Mined by Apriori Algorithm from jEdit4.3pre file	34
Table 4.2: Number of Methods Mined by New Algorithm of jEdit4.3 pre file	34
Table 4.3: Number of Methods mined by Apriori Algorithm from jEdit 5.0 file	35
Table 4.4: Number of Methods mined by New Algorithm from jEdit 5.0 version file	35

LIST OF FIGURES

Figure	Page No.
Figure 1.1: Data Mining Process	4
Figure 1.2: Process of Mining Software Repositories	7
Figure 1.3: Classification of Data Mining Techniques	14
Figure 1.4: Process of Mining Unstructured Data	14
Figure 3.1: Flowchart of Methodology	29
Figure 4.1: Browsing the jEdit version files	30
Figure 4.2: Mining Status Dialog Box appeared	31
Figure 4.3: Methods mined by Apriori algorithm	31
Figure 4.4: Methods mined by new algorithm	32
Figure 4.5: Methods mined by Apriori algorithm	32
Figure 4.6(a): Methods Mined by New Algorithm from jEdit 5.0 Version file	33
Figure 4.6(b): Methods mined by New algorithm	33
Figure 4.7: Number of methods mined by New Algorithm	34
Figure 4.9: Comparison in terms of Time	35
Figure 4.10: Graphical Representation of Time	36
Figure 4.11: Graphical Representation of Memory Required	36

Chapter 1

INTRODUCTION

This chapter gives the introduction about the software engineering data and a brief overview about data mining. The chapter also contains the introduction to techniques of data mining that are used to mine software engineering repositories.

1.1 Software Engineering Data

Software engineering is a constructive approach which involves systematic and cost effective techniques for software development. Software systems are very complex due to which it sometimes slows down the maintenance activities which lead to defects in system and finally result in high cost of software.

Software ‘artifacts’ are the products which are produced during software development processes which describes the functionality and design of software and also explains the development process. The examples of artifacts are class diagrams, use case diagram, project plans and business cases. The major challenge faced in software development is that these software artifacts are generally lost and thus it becomes difficult to provide full history of software system and its development process.

Software engineering data contains information about software and is also termed as **software repositories** [Ahmed E. Hassan, 2008]. It comprises of source code, execution traces, historical code changes, mailing lists and bug databases. Software engineering data constitutes information about status and progress of the software projects with the help of which practitioners can depend less on their intuition and experience and depend more on historical data. It is used to keep record of the software projects and used to support decision making process.

1.1.1 Types of software repositories

Software engineering data or software repositories can be classified into following types [Ahmed E. Hassan et al, 2007]:

- **Historical repositories:** Historical repositories are used to track the history of a bug or a feature. It contains the repositories such as source control, bug repositories and archived communications. It is used to record the development history of a software project, bug

reports and tracks the all the changes related to source code and various aspects of project all through its lifetime.

- **Run time repositories:** Runtime repositories keep track of information about the running state of software application. It consists of deployment logs which contain information about the execution and the usage of an application at a single or multiple deployment sites.
- **Code repositories:** Code repositories comprise of source code of various applications such as Sourceforge.net and Google code.

Software engineering data which is generally mined [Ahmed E. Hassan, 2008] is:

- Source code:** The repositories store all the changes that the different source code files undergo during the project. It also allows work to be done in parallel by different developers over the same source code tree.
- Issue tracking systems:** Issue tracking systems handles the bug defects and user requests, where users and developers can fill tickets describing the defects found, or a desired new functionality.
- Messages between developers and users:** In open source software, the projects are opened to world and the messages are communicated in the form of mailing lists, which can also be mined for research purposes.
- Meta data about the projects:** The meta data about the software projects can also be useful for research. It may include programming language, domain of application, license etc.

Table 1.1: Examples of Software Repositories

Repository	Description
Source control repositories	These record the development history of a project. They track all the changes in the source code along with meta data about each change. Ex: change units such as files, functions, CVS
Bug repositories	They keep the record of history of bug reports of software project all through its life. Ex: Bugzilla

Deployment logs	They record the information about the execution of a single deployment of a software application or different deployments of the same applications. Example: it may record the error messages reported by the application at various deployment sites.
Archived communications	The development team use some form of electronic communication to have discussions. This repository keeps record of the discussions about various aspects of software project throughout its lifetime. Examples are: mailing lists, emails, IRC chats etc.

Majority of software development organizations make use of revision control software such as CVS, subversion to manage the development of digital assets. Revision control software keeps a historical record of each revision and makes users to access previous versions. It also helps in analyzing historical artefacts produced during software development, such as number of lines written, authors which wrote particular line or any software metrics. Many organizations also make use of defect tracking software such as Bugzilla, JIRA which associates bugs with meta information such as status, comments dates which helps to detect defect prone modules.

1.2 Data mining

Data mining came into view around 1990. It can be expressed as knowledge mining from data by applying intelligent methods for extraction. Data mining is a knowledge discovery process which is used to reveal hidden patterns and trends in large datasets. It is the extraction of hidden predictive information from large databases and transforms it into understandable structure for further use. These discovered interesting patterns are presented to the user and may be stored as new knowledge in knowledge base.

It is a technology with great potential to help companies focus on the most important information in their data warehouses. Data mining techniques can be implemented rapidly on existing software and hardware platforms to enhance the value of existing information resources, and can be integrated with new products and systems as they are brought on-line. Data mining tools can answer business questions that traditionally were very much time

consuming to resolve. It polishes databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectation.

1.2.1 Type of data to be mined

- **Database data:** Data base management system consists of collection of interrelated data and software programs to access and manage this data. Relational databases can be mined by searching for trends and data patterns.
- **Data Warehouses:** Data warehouse is a repository of information collected from various sources and stored in a uniform manner. They are developed by data cleaning, data integration, data transformation, data loading and periodic data refreshing.
- **Transactional data:** Transactional database comprises of transaction such as airline booking
- **Other:** Other kinds of data beside the above mentioned data that can be mined are historical records, stock exchange data, time series and biological sequence data, spatial data, engineering design data, media data, graph and networked data.

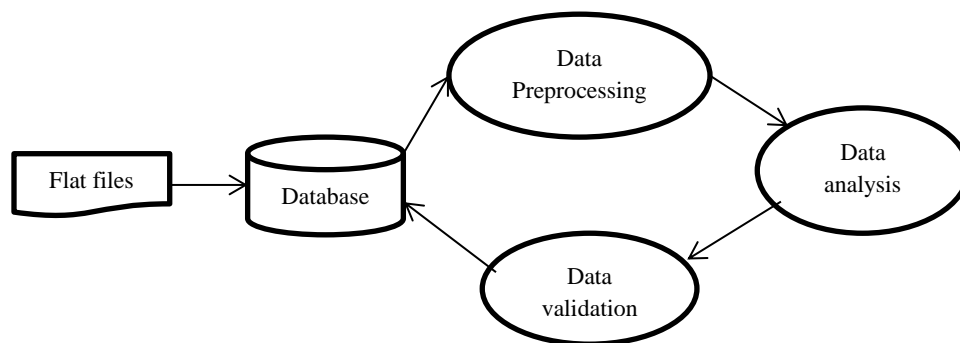


Figure 1.1: Data Mining Process

1.2.2 Data Mining Tasks

There are six common tasks involved in data mining:

- **Classification:** Classification is the process of examining features of newly presented objects and assigning them to a predefined class which consists of training sets.
- **Prediction:** Prediction is the process of getting valued outcomes for some unknown continuous variables. It finds a function which models the data with the least error.

- **Association Rules:** The rules that assign any association relation between set of objects i.e. they occur together
- **Clustering:** Division of elements into similar groups or clusters is called clustering,
- **Outlier Analysis:** identification of unusual data such as noise or exceptions is outlier analysis.
- **Description and Visualization:** This includes (i) verification of user's hypothesis (ii) discovery of patterns to find the future behavior of elements and representation of data sets.

1.3 Software Engineering and Data Mining

In order to improve the software quality, data mining techniques have been used to mine software engineering data which store the information about any software system related to its source code, bug history [Ahmed E. Hassan, 2008]. Software metrics has been used to calculate the reliability and performance of software, but there are some challenges faced while using metrics for estimating the quality of software. Metrics can be uninformative, invalid, irrelevant and can be difficult to change and compute [Quinn Taylor et. al, 2010]. Due to these challenges, data mining concepts provide better solution for computing quality of software applications.

Mining Software Repositories Workshop was established in 2004 with an aim to improve the quality of software engineering data by using different data mining procedures [Ahmed E. Hassan, 2007]. Mining software engineering data finds its use in the following [Quinn Taylor et. al, 2010]:

i. Development: Mining software engineering data makes development process of software effortless because enough relevant data can be provided that will help in guiding development.

ii. Management: Project managers have the task to prevent the introduction of faults and ensure that these faults are discovered quickly so that they can use the software engineering data to fulfill the requirements of customers.

iii. Research: From the research point of view, data mining helps in gaining insight about different software projects so that a better guidance can be provided for its development.

Mining software engineering data helps in converting the static records of software applications into active repositories which helps in decision making processes for modern software projects [Ahmed E. Hassan, 2007]. It focuses on mining frequent patterns from revision histories of various software applications by using pertinent data mining strategies along with the removal of outlier and noisy data. It helps in software maintenance at low cost and effort which prevents software failures and results in high quality software.

Mining information about software systems can help the developers to better understand the system and propagate changes in other software systems [Ahmed E. Hassan, 2006]. Mining software engineering data also helps in code reuse as extracting information from repositories helps in producing frequent information used to develop software and can be used as reusable component.

Software plugins are the additional features provided by software applications which are written by programmers who develop code for software [Benjamin Livshits et al, 2006]. Plugins and method calls in the source code of software applications may sometimes contain common errors which can be extracted by using data mining approach. Many software applications such as jEdit, Eclipse, Mozilla Firefox and many more consist of many plugins. The mining of patterns from large datasets can be more time consuming, therefore extraction of patterns by using a fast approach and that needs less space is a challenge [Daniel Rodriguez et al, 2012].

Software specific bugs are more likely to occur therefore huge amount of attention has been given in finding those bugs or errors. Error patterns also occur in the source code revision histories of many application systems due to programmers violating the coding rules [Rakesh Agarwal et al, 1994]. Mining software revision histories for frequent error patterns in large applications can be done using many data mining approaches which can be more time consuming and has less memory usage for mining the source code repositories and to uncover the work carried in mining software engineering data. Figure 1.2 shows the process of mining software engineering data which includes extracting data set by pre-processing the

repositories and then applying the mining strategy to get the desired mining result which will be used in different manners to improve the software quality.

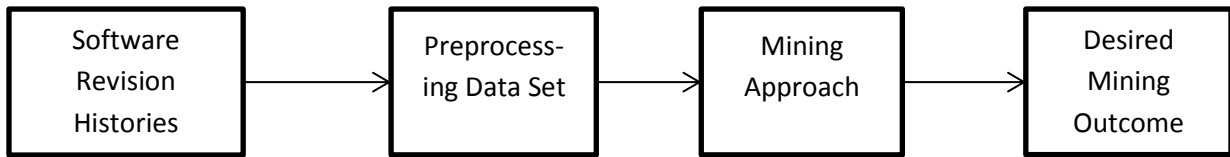


Figure 1.2: Process of Mining Software Repositories

Table 1.2: Software Engineering Data, Mining Algorithms and Software Engineering Tasks

Software Engineering Data	Mining Algorithms	Software Engineering Tasks
Sequences: execution/static traces, co-changes.	Association rule mining, frequent itemset/subsequence/partial-order mining, sequence matching/clustering/classification	Programming, maintenance, bug detection debugging
Graphs: dynamic/ static call graphs, program dependence graphs	Frequent subgraph mining, graph matching/clustering/classification	Bug detection, debugging
Text: bug reports, emails, code comments, documentations	Text matching/clustering/classification	Maintenance, bug detection, debugging

1.3.1 Association Rule and Frequent Pattern Mining

Association rule and frequent pattern mining technique was proposed by Agrawal et al. in 1994. It is the method of achieving the relationship between the variables.

Table 1.3: Software Engineering Data Mined by Frequent Pattern Technique

Information sources	Technique and Tools used	Task	Result	Reference
Open source NASA dataset	Classification model based on relational association rule(DPRAR)	Predicted whether a software module is defective or non-defective	The proposed classifiers predicted the defective modules	Gabriela Czubala et al. 2014

Open source projects(YARI, Zest, JUnit, JFreeChart, AgroUML)	Sub graph mining, AST, UML	Detecting identical data structures in object oriented systems	Detected many identical structures that can be classified as software design clones.	Umut Tekin et al. 2014
Data Streams	Frequent pattern mining algorithms	Comparative study of each algorithm	RARM and ASPMS perform better than other.	Shamila Nasreen et al. 2014
Real world data	Mining algorithm and tools, frequent pattern mining, Coding Tracker	Mining unknown frequent code change patterns	Algorithm was scalable efficient, mined half a million of instances in less than six hours.	Stas Negara et al. 2013
Histories from Eclipse project	Kenyon framework, Bird Data	Mining crashes or bugs, identifying fix patterns and fixing them	Identified three major crashes and found that there are fix patterns for crashes	Jaechang Nam et al. 2013
Eclipse plugin	GraPacc tool, frequent pattern mining	Mining API usage patterns	Mines graph based patterns and completes code	A. T. Nguyen et al. 2012
Health care software system	Program dependence graph, spatial pattern search, graph based pattern mining, false positive pruning	Pattern mining of cloned codes	Mined patterns were used for bug discovery, pattern analysis	Wei Qu et al. 2010

Classification is a predictive modeling technique which assigns an object to a certain class based on its similarity. It is done by constructing a model or classifier to predict class and assign a particular object to that class. Data classification is a two-step process which consists of learning step and classification step. The procedure is given as follows:

Procedure: In the learning step, classification algorithm builds a classifier with the help of a training set. A tuple X , represented by n dimensional attribute vector $X=(x_1, x_2, \dots, x_n)$, shows n measurements made on the tuple from m database attributes, A_1, A_2, \dots, A_m . Each tuple X belongs to a predefined class determined by class label attribute and individual tuples make a training set called training tuples. In classification step, the model or classifier is used to predict class labels for given data. Test set which comprises of test tuples and their associated class labels is compared with the learned classifier's class prediction

Table 1.4: Software engineering data mined by classification techniques

Information sources	Technique and Tools used	Task	Result	Reference
Eclipse version control system, Bugzilla	Eclipse metric plugin, classifier algorithm, R 2.15 tool	Predicting software defect classification by prediction models	Models predicted the defects and measure the performance of models	Hui Wang 2014
PC1 dataset of NASA's MDP	Fuzzy c-means clustering approach, k-Nearest Neighbors Classifiers and combination of Fuzzy c-mean and Genetic algorithm, MATLAB R2010	Comparing performance of software fault prediction system	Combination of Fuzzy c-mean and Genetic algorithm is more efficient.	Anil Kumar Singh et al. 2014
NASA dataset	Naive Bayes, Neural Networks, Association Rules, Decision Tree	Predict the defective state of software modules	Best performance in predicting defective modules is of Naïve Bayes then neural network then decision tree.	Ahmed H. Yousef 2014

Open source NASA dataset	Classification model based on relational association rule(DPRAR)	predicted whether a software module is defective or non-defective	The proposed classifiers predicted the defective modules	Gabriela Czibula et al. 2014
jEdit	Genetic algorithm, Chidamber & Kemerer metrics suite	Finding classes and metrics that are fault prone	The result was measured in terms of accuracy and found the proposed approach efficient.	Aditi Puri et al.2014
KC1 dataset of NASA metric data program	SVM(classifier algorithm), WEKA	Investigated accuracy of SVM	SVM with Poly kernel is most accurate	P.A. Selvaraj et al. 2013
Linux kernel, Mozilla, and Apache	Randomly sampling bugs, machine learning techniques	Study the characteristics of software bugs	Studied the root cause and impact of bugs.	Lin Tan et al. 2013
NASA software repository	Rule-based classification, enhanced RIDOR algorithm	Classify the software modules as fault prone or not fault prone.	Accurately mined defects from datasets	Hassan Najadat et al. 2012
Eclipse versions, Equinox	SVM and Fuzzy classification	Predicting defective modules	Achieved 76.5 mean recall and 34.65 mean false alarm rate.	Bharavi Mishra et al. 2012
NASA MDP	COCA algorithm	Predict software defects.	The algorithm is more effective	Xiao-dong Mu et al. 2012
Mozilla project, CVS	Bugzilla, machine learning, stratification based resampling	Mining dataset from project	Computed 18 object oriented metrics,found Heteroskedastic	Lourdes PELAYO et al. 2009

			ity and skewness problems	
NASA software projects	AntMiner+ ant colony optimization based on classification	Prediction of erroneous software modules	The approach is accurate and efficient in predicting error modules in software projects.	Olivier Vandecruys et al.2008
Eclipse plugin	Change classification(SVM)	Predicting bugs inside an IDE	Effective in classifying changes.	Madhavan et al. 2007

1.3.3 Prediction

Prediction is a method in which the identity of one attribute is predicted based on the description of another attributes that are interrelated. It is a supervised learning task in which the data, without creating any explicit models, is used to predict the class value of a new instance.

Procedure: Let us assume that we have a dataset where each record has attributes X_1, \dots, X_n and Y . The goal of prediction is to learn a function and use this function to predict Y attribute for input record $(X_1 \dots X_n)$. Predicted values are usually continuous and here Y is a continuous attribute. A model or predictor is constructed to compute the predicted value and prediction techniques are based on fitting a curve by finding the relationship from predictor to predicted value.

Table 1.5: Software engineering data mined by prediction technique

Information sources	Technique and Tools used	Task	Result	Reference
JHotDraw, JEdit	Latent Dirichlet allocation (statistical modeling)	Evaluating topic models in the analysis of software evolution.	Topics in both the systems grow as the size of source code grows. Topic evolutions of JHotDraw are more active than those of jEdit	Stephen W. Thomas et al.2014

Rhino, ajc, Lucene	FindBugs tool	Study static correspondence and statistical correlation between defects and warning	No statistical correspondence but moderate statistical correlation between defects and warning	Cesar Couto et al. 2013
25 projects of telecommunication system(GSM operator), NASA MDP	Defect prediction model, static call graph based ranking (CGBR), nearest neighbor sampling.	Prediction of fault prone modules in large system using data mining.	70% of the defects were detected by the proposed approach.	Burak Turhan et al.2008
Windows XP OS	Machine learning algorithm(logistic regression), MATLAB	Predict risk of software metrices.	70% of engineers use regression risk reports while testing	Alexander Tarvo 2008

1.3.4 Clustering

Clustering is the process of partitioning a set of data objects into subsets called a cluster. Objects in a cluster are similar to one another and dissimilar to objects in other cluster. Clustering methods are classified into following categories:

- **Partitioning methods:** Let there be set of n objects such that k partitioning of data is constructed where each partitioning represent a cluster and $k \leq n$. Most partitioning methods are distance based. There are two methods of partitioning:
 - k-Means: Let D be a data objects containing n objects. Divide the objects into k clustes, C_1, C_2, \dots, C_k such that $C_i \in D$ and $C_i \cap C_j = \Phi$. C_i is centroid of cluster.

$$E = \sum_{i=1}^k \sum_{p \in C_i} dist(p, c_i)^2$$

- k-Medoid: In this method, absolute error criterion is used which is defined as:

$$E = \sum_{i=1}^k \sum_{p \in C_i} dist(p, o_i)$$

where p is all objects in dataset and o_i is the representative object of C_i . This is the basis of k-medoid method,

which groups n objects into k clusters by minimizing the absolute error.

- **Hierarchical methods:** Hierarchical method creates hierarchical decomposition of data objects. There are two sub categories of this method: agglomerative and divisive approach (AGNES and DIANA). AGNES is a bottom-up approach and DIANA is a top-down approach.
- **Density-based methods:** Density based method is based on density. The basic idea of this method is to continue growing a given cluster as long as the density in the neighborhood exceeds some threshold. The techniques used in this method are: Density Based Clustering on Connected Regions with High Density (DBSCAN), Ordering Points to Identify the Clustering Structure (OPTICS) and Clustering Based on Density Distribution Functions (DENCLUE).

Table 1.6: Software engineering data mined by clustering technique

Information source	Technique and Tools used	Task	Result	Reference
Connection-oriented telecommunication data	k-Means and Fuzzy C-Means	Compared the performance of clustering algorithms.	The computational time of k-Means algorithm is less than FCM algorithm.	Velmurugan T 2014
3 PROMISE repository AR3, AR4 and AR5	Algorithms such as DBSCAN,OPTICS, X-means,CLOPE	Developed a fault prediction model which select number of clusters without experts.	Performance of X-means is better than that of other.	Mikyeong Park et al 2014
Repository of machine learning database	Advanced clustering algorithms,K-means,WEKA tool.	Proposed algorithm to deal with problem of curse of dimensionality.	Advanced clustering algorithms have less execution time and are more accurate.	Amanpreet Kaur Toor et al. 2014
4 datasets	k-Means clustering	Proposed a	The proposed	Amita

Tunedit Repository	algorithm, clustering tool WEKA.	method for making the K-Means algorithm more effective and efficient	algorithm has been found effective	Verma et al. 2014
--------------------	----------------------------------	--	------------------------------------	-------------------

1.3.5 Text Mining

Text Mining is a process of automatically extracting information from a large amount of different unstructured textual resources. In this method, the patterns are extracted from natural language text rather than database which result in improving the decisions and predictions about given data. The data can be classified as structured data and unstructured data. Language is usually unambiguous which makes text mining a hard process.

Procedure: Figure 1.4 below describes the process followed while mining textual data. It includes the activities:

- i) Document Collection
- ii) Preprocessing the data
- iii) Analysis of textual data
- iv) Management Information System
- v) Extracting knowledge

Figure 1.3 below shows the classification of different data mining techniques used to mine software engineering data.

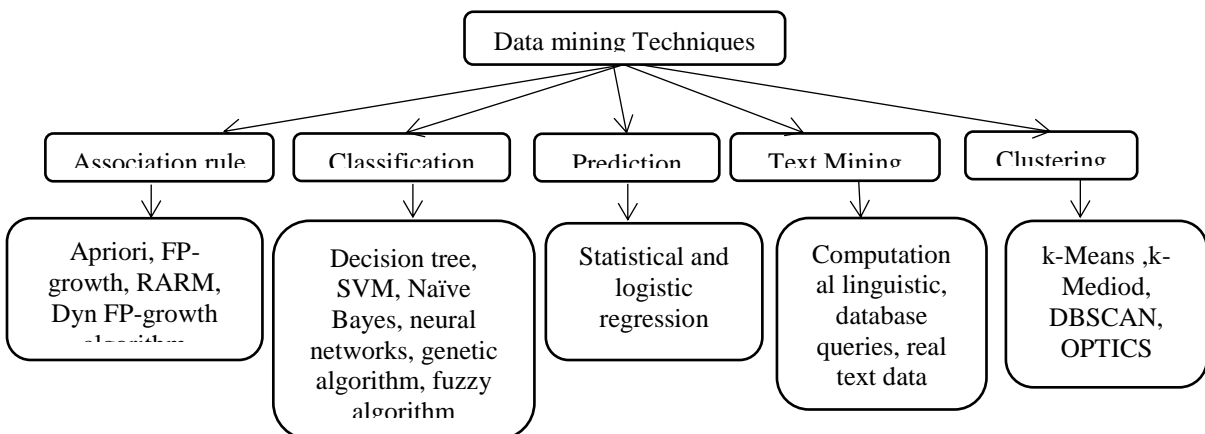


Figure 1.3: Classification of Data Mining Techniques Applied on Software Engineering Data

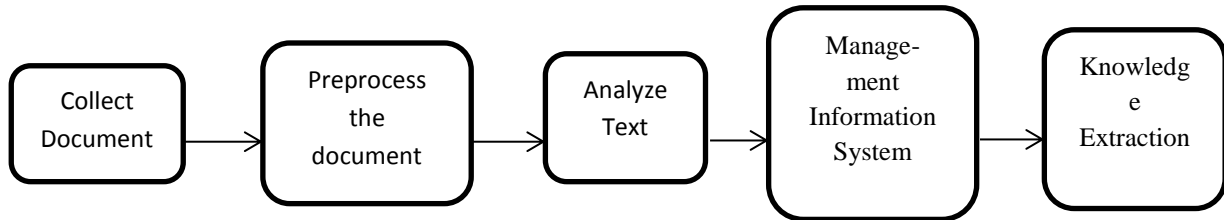


Figure 1.4: Process of Mining Unstructured Data

Table 1.7: Software engineering data mined by text mining technique

Information sources	Technique and Tools used	Task	Result	Reference
ANDROID OS malware families.	Text mining and information retrieval techniques	Proposed a text mining approach to automatically classify smartphone malware samples.	Technique is fast, scalable and very accurate.	Guillermo Suarez-Tangil et al 2014
3 open source web applications	Text Mining techniques and software metrics	Compared vulnerability prediction models based on text mining with models using software metrics as predictors.	Text mining provided better recall performance at same cost.	James Walden et al 2014
Eclipse bug report and code revision of Android	Sequential pattern query language	Proposed a pattern matching approach for effective and efficient query of sequential software engineering data.	Analysed that there are 5.84 % files that have gone through more than 10 revisions.	Chengnian Sun et al.2014
RCV1 data collection and TREC topics	Pattern deploying algorithms, pattern mining.	Proposed a pattern discovery technique to extract relevant information.	Proposed PTM outperforms in terms of F-measures.	Ning Zhong et al.2012

C++ source code	Text mining source code management tool	Mining program source code for debugging	Code maintenance , outputs code into other format	A.V.Krish na et al. 2010
-----------------	--	--	---	--------------------------------

Chapter 2

LITERATURE SURVEY

This chapter introduces the studies that have been already been used to mine software engineering data. It introduces the techniques and tools that have been used by academicians for mining software engineering data.

Madhavan et al. 2007 predicted bugs inside an IDE by using change classification in Eclipse plugin. Change Classification uses Support Vector Machines (SVM), a machine learning classifier algorithm, to classify changes to projects mined from their configuration management repository. A Change Classification plugin for Eclipse was based on client-server architecture and it was found that the approach was effective in classifying changes.

Olivier Vandecruys et al.2008 proposed a novel approach named AntMiner+, to predict erroneous software modules. The approach used Ant Colony Optimization (ACO) to infer rules from the data. AntMiner+ was applied on three publicly available datasets of NASA software projects³: PC1, PC4 and KC1. To compare the results of AntMiner+, a benchmarking study was performed that includes commonly used state-of the-art classification techniques C4.5, RIPPER, logistic regression, 1-nearest neighbor, support vector machine. It was found that the approach is accurate and efficient in predicting error modules in software projects.

Sunghun KIM et al. 2008 proposed a technique for finding latent software bugs, called change classification. Change classification makes the use of machine learning classifier algorithm to determine whether a new software change is buggy or clean change. The file change histories and features from source code of 12 open source projects were extracted from their software configuration management systems. The classification accuracy, recall, and precision were evaluated for each project. Support vector machine classifies file changes as buggy or clean with 78% accuracy. The classification model was evaluated using 10 fold cross validation method.

Alexander Tarvo 2008 predicted risk of regression by software metrics by using historical data on changes in Windows XP operating system. The data collected included post-release changes, or *fixes*, which are made after software system is released to the market. Machine learning algorithm (logistic regression) was used to perform the experiment using MATLAB tool. Analysis of system's accuracy shows it could be successfully used to predict software regressions. It was concluded that 70% of engineers use regression risk reports while testing to support efficient testing.

Burak Turhan et al. 2008 predicted fault prone modules in large system using data mining techniques. 25 projects of a large telecommunication system were analysed. To predict defect proneness of modules, models on publicly available Nasa MDP data were trained. In the experiments, static call graph based ranking (CGBR) as well as nearest neighbor sampling was used for constructing method level defect predictors. The results suggested that for the analyzed projects, at least 70% of the defects can be detected by inspecting only 6% of the code using a Naïve Bayes model, 3% of the code using CGBR framework.

Lourdes PELAYO et al. 2009 extracted and mined a large scale software prediction dataset from an open source software system, Mozilla project. The focus of research work was to extract object oriented software metrics from the C++ classes contained in Mozilla and relate them to defects found in those classes and report by Bugzilla defect tracking system. 18 metrics were computed for each class and joined these metrics to the number of defects reported for each class. A statistical dataset of this dataset was constructed. It appeared that Heteroskedasticity and skewness were significant problems within this dataset. This skewness was removed by employing stratification based resampling. Further, it was stated that this analysis could also be combined with additional evidence from other software artifacts.

Wei Qu et al. 2010 proposed a framework for pattern mining of cloned codes by using a joint space-logic-domain analysis. Different programs of health care software system were used to perform this experiment. In this novel approach, graph matching was used to recover lost information and enhance mining accuracy. Both false positive pruning and pattern

composition were further adopted to improve the pattern mining performance. Finally the result was compared with the methods using only graph matching and it was found that this approach reduces the high computational complexity and thus greatly improves the algorithm's efficiency.

Stephen W. Thomas et al. 2011 evaluated topic models in the analysis of software evolution by performing a detailed manual analysis on the source code histories of two systems, JHotDraw and jEdit. Latent Dirichlet allocation, a statistical topic modeling technique, was applied to the release history of the source code, several metrics on the discovered topics were computed and investigated the source code and project documentation to verify that the evolution of metric values is useful and consistent with the actual change activity in the source code. It was found that topics in both the systems grow as the size of source code grows and topic evolutions of JHotDraw are more active than those of jEdit.

A.T. Nguyen et al. 2012 proposed GraPacc, a graph based, pattern-oriented, context-sensitive code completion approach to mine API usage patterns. GraPacc represented and managed the API usage patterns of multiple variables, methods, and control structures via graph-based models. It extracted the context-sensitive features from the code e.g. the API elements on focus and their relations to other code elements. The features were then used to search and rank the patterns that are best matched with the current code. The proposed approach was performed on Eclipse plugin. Empirical evaluation results showed that GraPacc can achieve high accuracy in code completion up to 95% precision, 92% recall, and 93% f-score.

Bharavi Mishra et al. 2012 predicted defective modules by proposing an approach named as Support Vector based Fuzzy Classification System. In the proposed model an initial rule set was constructed using support vectors and Fuzzy logic and rule set optimization was done using Genetic algorithm. The experiment was performed on three versions of Eclipse and Equinox datasets. It was found that the prediction performance of SVFCS approach is generally better than other prediction approaches. The approach achieved 76.5 mean recall and 34.65 mean false alarm rate on three versions of Eclipse and Equinox software bug data

sets. The experimental results revealed the effectiveness of SVFIS in fault prone module prediction.

Hassan Najadat et al. 2012 predicted the fault prone modules using data mining techniques to help in building better designs in future systems and avoid error prone modules. Different data mining rule based classification techniques were applied on the datasets of NASA software repository which is composed of several static code attributes. The objective of the work was to classify the software modules to either fault prone or not fault prone modules. The selected datasets were: PC1, PC2, PC3, PC4, CM1, MW1, KC3, and KC4. A new enhanced RIDOR algorithm was introduced which combines RIDOR and CLIPPER mining algorithms and predicts defect using mining of static code attributes.

Ning Zhong et al. 2012 proposed a pattern taxonomy model to extract relevant information from RCV1 data collection and TREC topics datasets. The model was based on pattern deploying and pattern evolving approaches. It was found that proposed PTM outperforms and resulted in 0.440 F-measure which is a performance measure.

Xiao-dong Mu et al. 2012 proposed an approach named, competitive organization coevolutionary algorithm to improve the accuracy of prediction for software defect in large data sets. In COCA, the individual's fitness is calculated not only by a population but also relying on competition among species. In this algorithm, population is divided into two competitive parts which are species training data (STRD) and species test data (STED). The experiment was based on the five datasets from NASA which were used to validate the method. The experimental results showed that the proposed method is effective.

Jaechang Nam et al. 2013 reviewed the bug fixing histories extracted from Eclipse project. All the commits related to fixing some bugs are extracted. Some common fix types are revealed which were collected by Bird data and was used to fix certain kinds of program exceptions. Kenyon framework was used to extract the data and get the major crashes of Eclipse and 1999 fixes were found. It was summarized that there are three major crashes of the Eclipse project: NullPointerException, IndexOutOfBoundsException and ClassCastException. It was identified that changes can be made to files to fix the crash and

was analyzed that by this methodology 80% of fixes require one file modification which help developers to see only a small set of files for fixing a particular crash.

Lin Tan et al. 2013 identified the characteristics of software bugs by randomly sampling 2,060 real world bugs in three large open-source projects: the Linux kernel, Mozilla, and Apache. The root cause, impacts and components of bugs were studied. The method consisted of preprocessing bug reports step, training, evaluating classification performance and applying classification models on the entire Bugzilla databases. It was found that semantic bugs are the dominant root cause of bugs in software projects.

Cesar Couto et al. 2013 reviewed static correspondence and statistical correlation between defects and warning by using FindBugs which is a bug finding tool used in Java systems. FindBugs detected and removed defects in Rhino, ajc and Lucene systems. It was found that no static correspondence exist between defects and warning but statistical tests showed that there is a moderate level of correlation between warnings and software defects.

P.A. Selvaraj et al. 2013 investigated the accuracy of Support Vector Machine for software defect prediction using different kernels. KC1 datasets which is a NASA metric data program was used to predict defects. Several classifiers were discussed such as decision stumps, Naïve Bayes and SVM. Weka, a machine learning software was used to carry out classification. Experiments revealed that SVM with Poly kernel achieves best performance in predicting defects.

Aditi Puri et al. 2014 proposed an approach, genetic algorithm, to find classes and metrics that are fault prone. The raw data was collected in the form of structural codes, source code of open source system, jEdit and the evaluation of this data obtained was done on Chidamber & Kemerer metrics suite. The source code of jEdit was collected and analyzed and fed as input into JArchitect which is astatic analysis tool for JAVA code. The parameters calculated are number of public methods, cyclomatic complexity, LOC, efferent coupling, relational cohesion, depth of inheritance tree. Genetic algorithm was applied on the metric values which predicted the fault prone metrics.

Ahmed H. Yousef 2014 predicted the defective state of software modules by using Naive Bayes, Neural Networks, Association Rules and Decision Tree methods. The algorithms were performed on the NASA dataset which is available online. The prediction results were measured with the parameters precision, recall, accuracy and F-measure. McCabe's metrics, Halstead Metrics, branch-count and five different measures representing the lines of code were used for measurement. Comparison between accuracy of different models was done using lift chart technique and classification metric technique. It was found that the best performance in predicting defective modules is of Naïve Bayes then neural network and then decision tree method.

Amanpreet Kaur Toor et al. 2014 proposed an algorithm to deal with problem of “curse of dimensionality” in large dataset using advanced clustering algorithms and k-means algorithm. Repository of machine learning database was that the advanced Clustering Algorithm can improve the execution time and are more effective. It was found that SOM has 297 ms execution time and forms 6 clusters.

Amita Verma et al. 2014 proposed a modification in the simple K-means algorithm by changing the distance similarity measure to make it more effective and efficient. The experiment was performed on four datasets of Tunedit repository and it was found that proposed algorithm was effective when compared with standard k-means clustering algorithm in terms of execution time and was found that the enhanced K-means takes 0.03 execution time. Table 2.4 provides the review of clustering algorithm for mining software repositories.

Anil Kumar Singh et al. 2014 compared the performance of software fault prediction system using three methods: Fuzzy c-means clustering approach, k-Nearest Neighbors Classifiers and a combination of Fuzzy c-means and Genetic Algorithm. The defect dataset was taken from NASA's MDP software repository named as PC1. Suitable metric value was set in database created in MATLAB R2010 as 0 for data with fault and 1 for data without

fault. A software fault prediction module was developed using the three data mining techniques mentioned above. The results after classification of software fault data was in form of efficiency parameters like Accuracy, Reliability, Mean Absolute Error, and Root Mean Squared Error for the comparison of all the approaches. It was found that the hybrid method gives more accuracy and less error as compared to Fuzzy C-means clustering and k-Nearest Neighbors Classifier.

Chengnian Sun et al. 2014 proposed a pattern matching approach and designed a sequential patterns query language to query sequential software engineering data. The query was applied on Eclipse bug report and code revision of Android and explored the approach in software development and maintenance. It was analyzed that there were 5.84 % files that have gone through more than 10 revisions. The query searched for the Android files that are modified by more than 10 developers within a month and was found that it takes 43 seconds to execute this query, and 14 files were returned. It was also found that the experiment on Android file revision history takes longer time than Eclipse report.

Gabriela Czibula et al. 2014 proposed a novel classification approach based on relational association rule mining, called DPRAR, to predict whether a software module is defective or non-defective. The main focus of this approach is on binary classification problem. The evaluation was performed on open source NASA datasets. The proposed approach followed the idea of representing the entities of a software system as multidimensional vector, whose elements are the values of different software metrics applied to the given entity. Training and testing phases that reflect the principles of a supervised learning algorithm were used for this process. Data preprocessing, training the DPRAR classifier, testing or classification was performed for the classification of modules as defective or not. For evaluating the performance of the DPRAR classifier, a cross-validation using a “leave-one-out” methodology was applied. It was found that the proposed method was efficient in predicting the defective modules.

Guillermo Suarez-Tangil et al. 2014 introduced DENDROID, a system based on text mining and information retrieval techniques to automatically classify smartphone malware

samples. The work proposed is motivated by a statistical analysis of the code structures found in a dataset of ANDROID OS malware families. It was found that the technique is fast, scalable and very accurate in discarding 16 out of the 49 families as they only contain one specimen each, resulting in a final dataset of 1231 malware samples grouped into 33 families.

Hui Wang 2014 proposed software defect classification prediction based on software development repository using the data from Eclipse version control system and bug tracking system. The data was used to calculate the software metrics for files and packages by Eclipse metrics plugin. The defect information in defect tracking system bug-Info was mapped to version control system to obtain defect version location information and thus found the relevant version of the software defect statistics. The datasets was analyzed, preprocessed and classifier algorithm was used to build prediction model with the help of R 2.15 tool to predict the software defects. Moreover it was stated that this model could be extended to more software repositories.

James Walden et al. 2014 compared vulnerability prediction models based on text mining with models using software metrics as predictors using text mining techniques and software metrics on three open source web applications Drupal, PHPMyAdmin and Moodle which contain 223 vulnerabilities. It was found that text mining provided better recall performance at same cost.

Mikyeong Park et al. 2014 developed an unsupervised fault prediction model using clustering algorithms which select the number of clusters without experts. The clustering algorithms used are DBSCAN, OPTICS, X-means, CLOPE and the experiment was performed on three PROMISE repositories AR3, AR4 and AR5. The performance of these algorithms was measured in terms of precision, recall and accuracy and the experimental results showed that X-means algorithm outperforms other clustering algorithm.

Shamila Nasreen et al. 2014 mined unknown frequent code change patterns in real world data and presented a comparative study of Apriori algorithm, Frequent Pattern (FP) Growth algorithm, Rapid Association Rule Mining (RARM), ECLAT algorithm and Associated

Sensor Pattern Mining of Data Stream (ASPMS) frequent pattern mining algorithms. The performance of every algorithm was compared using parameters like database scan, execution time. It was concluded that RARM and ASPMS has better performance than other algorithms.

Umut Tekin et al. 2014 detected identical data structures in object oriented systems by using sub graph mining approach for improving the knowledge about the software architecture. The method was performed in three steps; abstract syntax tree and UML based design, graph partitioning and sub graph mining algorithm. The experiment was performed on five open source projects YARI, Zest, JUnit JFreeChart and AgroUML. It was concluded that by the proposed algorithm we can detect many identical structures that can be classified as software design clones, reused design structures between projects, common design patterns, domain-specific patterns, copy–paste-modify structures or repeated design disharmonies within and between projects.

Velmurugan T 2014 compared the performance of partitioning based clustering algorithms, k-Means and Fuzzy C-means algorithm in terms of computational time on the telecommunication data source. It was found that the results obtained from these algorithms were more accurate, easy to understand and the time taken to process the data was high in Fuzzy C-Means algorithm than the k-Means.

Chapter 3

PRESENT WORK

This chapter gives a brief introduction about the objectives of research work and description of the proposed research methodology.

3.1 Objectives of Research

1. To study and analyze software engineering data and various data mining techniques applicable for mining software engineering data.
2. To study various frequent pattern mining approaches to mine software engineering data.
3. To propose an algorithm using evolutionary techniques for mining software engineering data.
4. To implement the proposed data mining algorithm to mine usage patterns from processed data of revision histories and validate the performance by comparing with existing algorithms.

3.2 Research Plan

The main aim of this research is to prove that the proposed approach is better in mining the usage patterns from revision histories than the existing traditional approach.

Step 1: The raw data will be collected in the form of source code check-ins and method calls of revision histories of some known applications like jEdit/Eclipse. The data is preprocessed and cleaned by including only method pairs and discard the remaining data.

Step 2: The data collected is mined for likely usage patterns and error patterns by applying the proposed algorithm in which the concepts of Association rule Mining and heuristic concept of Ant Colony Optimization algorithm has been used.

- **Association rule and frequent pattern mining:**

Association rule and frequent pattern mining technique was proposed by Agrawal et al. [6] in 1994. It is the method of achieving the relationship between the variables. It extracts association rules with the help of the frequent patterns mined. Let T transaction contains an item x if $x \in T$. Let an itemset X occurs in a transaction T if $X \subseteq T$. Let a dataset D of transactions and itemset X be given. The dataset cardinality is denoted by $|D|$. The count of X in D is denoted by $count^D(X)$. It is the number of

transactions in D that contains X . The support of X in D is denoted by $\text{sup port}^D(X)$. It is the percentage of transactions in D that contain X .

$$\text{sup port}^D(X) = \frac{|\{T \in D \mid X \subset T\}|}{|D|} \quad \dots(i)$$

An association rule is a pair as, $X \rightarrow Y$ where X and Y are two item sets and $X \cap Y = \Phi$. The item set X is called the antecedent of the rule. The item set Y is called the consequent of the rule.

- Support: Support can be defined as, $\text{sup port}^D(X \rightarrow Y) = \text{sup port}^D(X \cup Y)$
- Confidence: Confidence of the rule is defined as percentage of transactions in D containing X that also contain Y .

$$\text{confidence}^D(X \rightarrow Y) = \frac{\text{sup port}^D(X \cup Y)}{\text{sup port}^D(X)} = \frac{\text{count}^D(X \cup Y)}{\text{count}^D(X)} \quad \dots(ii)$$

The advantages of Apriori algorithm is that it is easily parallelized and it is easy to implement.

On the other hand, there are many disadvantages also:

The major weakness of Apriori algorithm is producing large number of candidate itemsets. Apriori algorithm requires multiple database scans which is equal to maximum length of frequent itemset and the mining becomes slow and expensive.

- **Ant colony optimization:**

Ant colony optimization is a metaheuristic approach that is derived from the behavior of real ant colonies and which is used to solve complex optimization problems. ACO is a class of algorithms, whose first member, called Ant System, was proposed by Colomi, Dorigo and Maniezzo. The main idea of the algorithm is based on the behavior of real ants that make a parallel search over several tours based on local problem data and on a dynamic memory structure containing information on the quality of previously. The ants move through states of problem by making

probabilistic decisions. This decision policy depends on two factors: heuristic value and trail.

Heuristic value: heuristic value (η_i) indicates the a priori desirability of the move.

Pheromone Trail: Trail value (τ_i) indicates how proficient it has been in the past to make that particular move. It represents a-posteriori indication of desirability of that move. When all ants have completed a solution, the pheromone trail is updated by:

$$\tau_{xy} \leftarrow (1 - \rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

Proposed algorithm is the improvement of Apriori algorithm which uses the heuristic approach for data mining. The proposed algorithms observe the pattern of the mined data and update the support vector over time to time.

Pseudo code of Mining Algorithm:

Begin

//Initialization Step

Support=20%

If(transactionID > Support)

 Select item in mining frequent item set in DB

//Transformation Step

For k=i to n //mining in vertical data

 If(transactionID < support)

 Learn item sets

//Final Reduction

 Total result & o/p

End

Step 3: The patterns that are mined will be ranked and validated by applying various ranking strategies (finding support and confidence/association rules, dynamic count). Support count and confidence level values will be taken with a proper ratio, so that error pattern can be generated by applying the algorithm on datasets. After the insertion of support count and confidence values, patterns will be ranked using ranking approach.

Step 4: The performance of the algorithm and the existing standard algorithms is compared in terms of execution time and memory acquired.

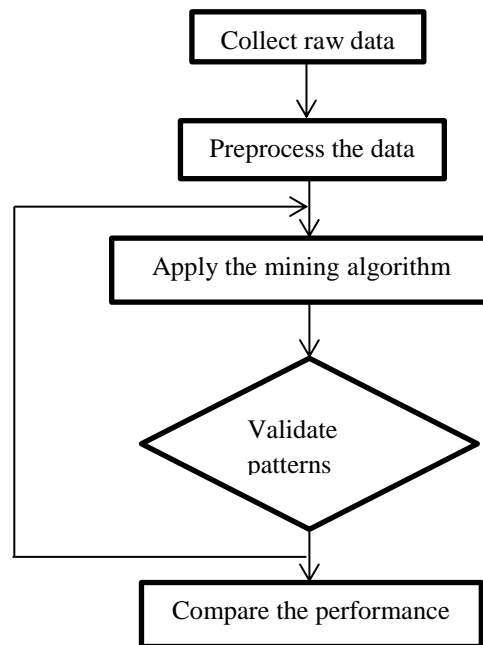


Figure 3.1 Flowchart of Research Methodology

3.3 Tools Used

Eclipse IDE:

Eclipse is an integrated development environment provides users with a base workspace and plug-in system for modifying the environment. Eclipse jee includes an official support of java 8 in java development tools, object teams, eclipse communication framework, wed tools platform, plugin development tools, memory analyzer etc.

Chapter 4

RESULTS AND DISCUSSIONS

This chapter gives the results obtained after the simulation of the proposed approach.

Step 1: The data for the research is taken as the source code of the open source java project, jEdit. The source code of the two versions of jEdit, jEdit 4.3pre and jEdit 5.0 is collected and analyzed. The java source code files of both versions are preprocessed and a single source file is prepared for mining purpose from which all the methods in the source code will be mined.

Step 2: The new mining algorithm is applied on both the versions of jEdit and the algorithm resulted in number of methods and all the frequent methods in both the versions. The jEdit versions contain java source code files which are mined by both Apriori algorithm and the new proposed algorithm.

Figure 4.1 represents the interface for mining the jEdit files by browsing the files from the system. Figure 4.2 represents the dialog box appearing when the mining of files is successful and shows the status of mining.

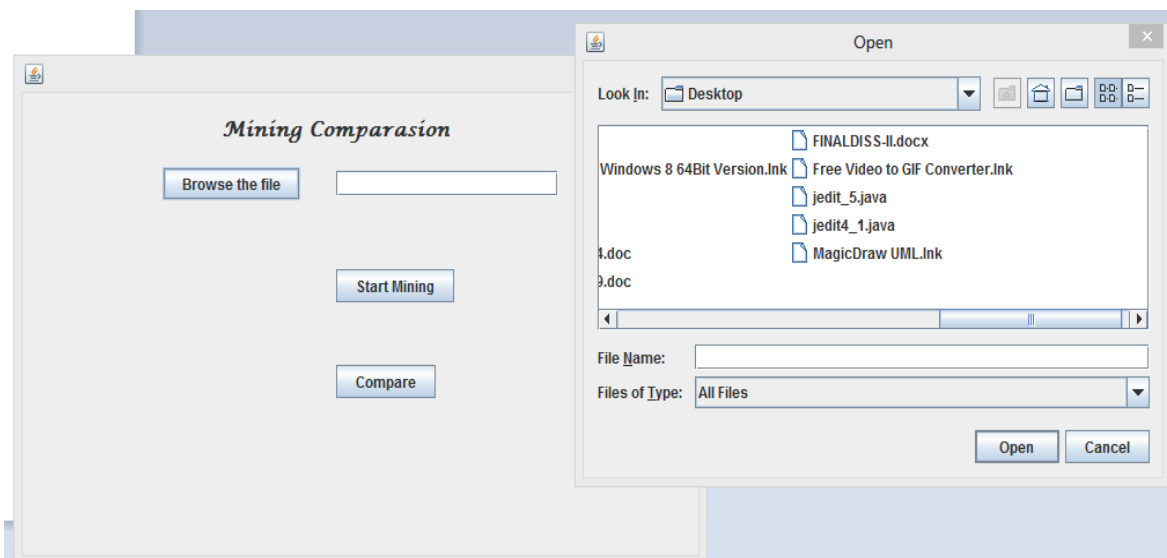


Figure 4.1: Browsing the jEdit Version Files

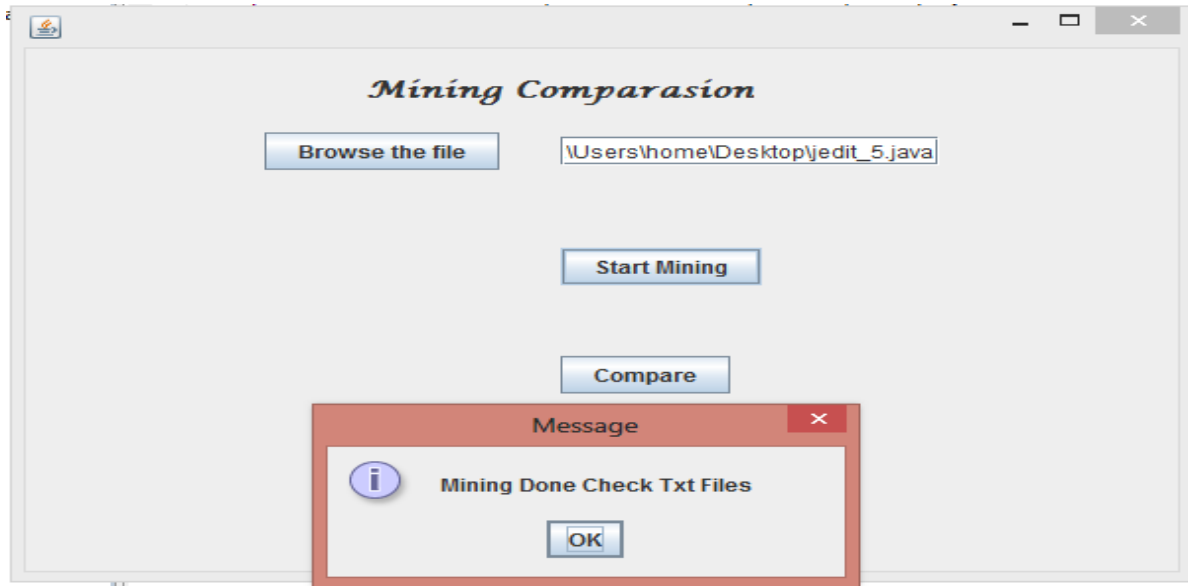


Figure 4.2: Mining Status Dialog Box Appeared

Results: The methods mined from jEdit files are:

Figure 4.3 shows the frequent methods mined by Apriori algorithm from jEdit4.3 pre version file. Figure 4.4 shows the frequent methods mined by New proposed algorithm from jEdit4.3 pre version file and shows the methods under different category.

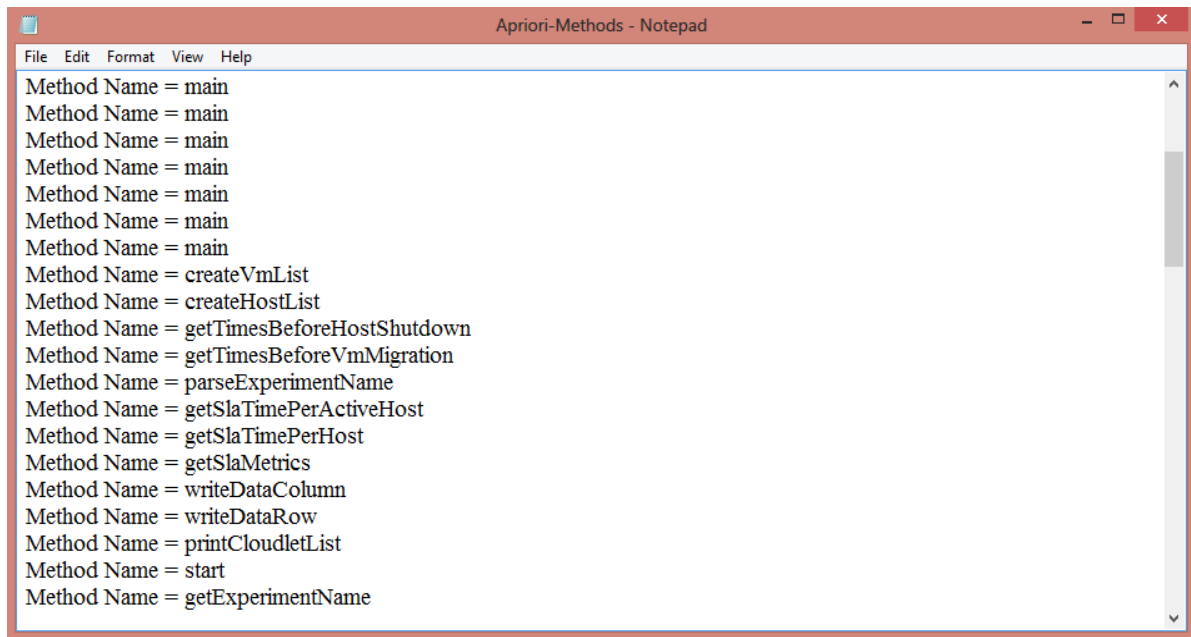
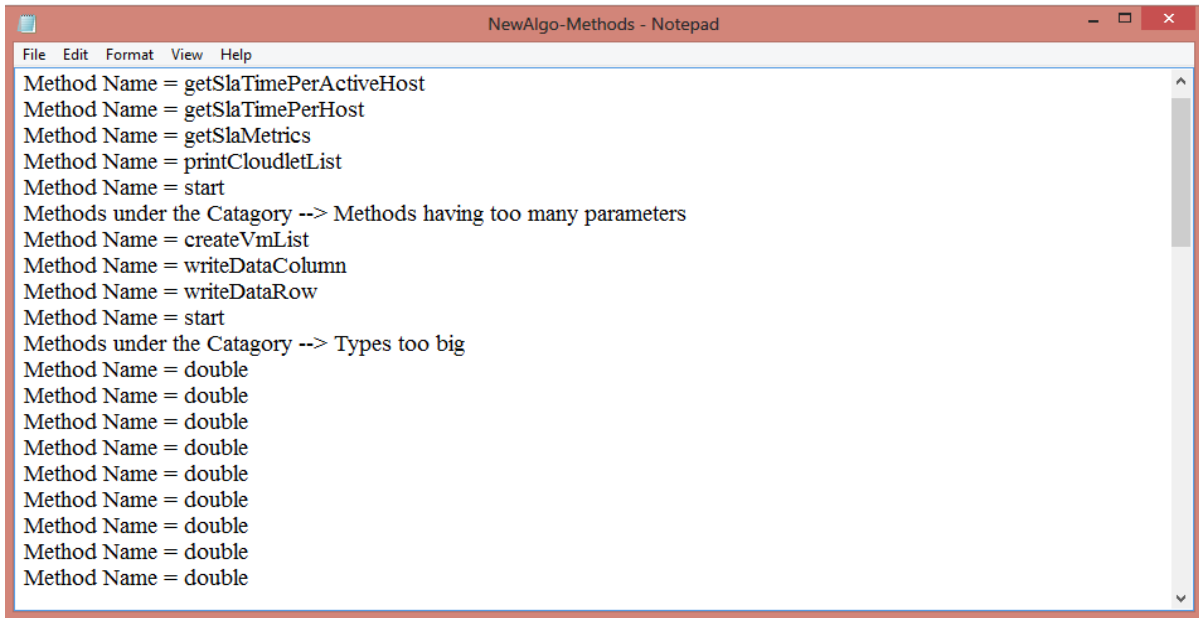


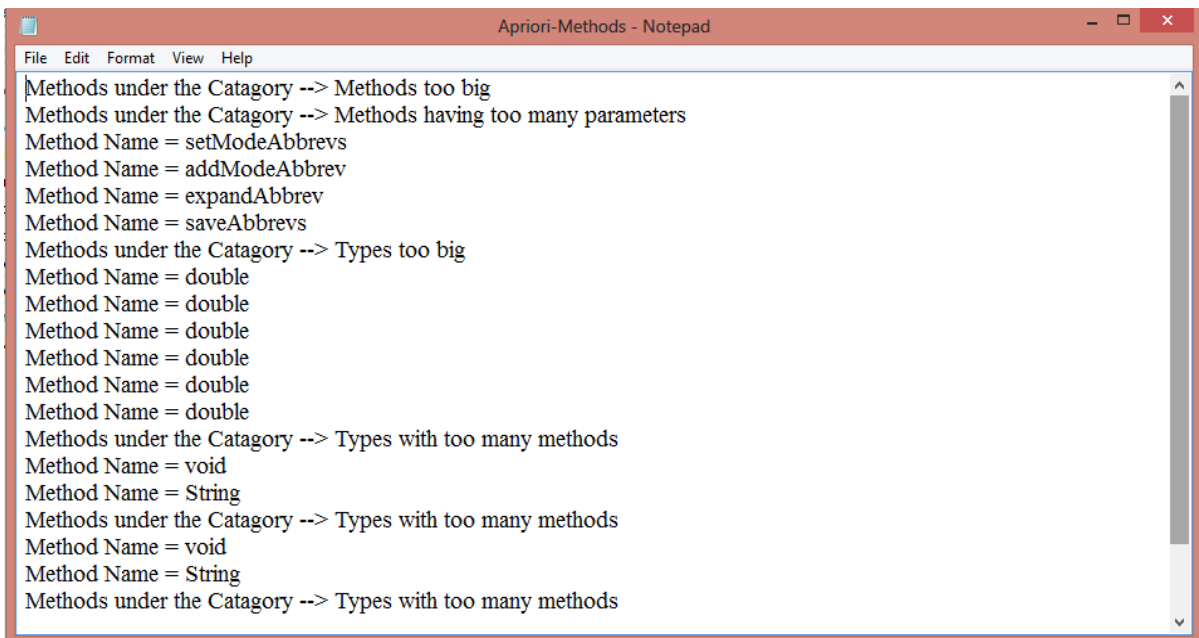
Fig 4.3: Methods mined by Apriori algorithm



```
File Edit Format View Help
Method Name = getSlaTimePerActiveHost
Method Name = getSlaTimePerHost
Method Name = getSlaMetrics
Method Name = printCloudletList
Method Name = start
Methods under the Catagory --> Methods having too many parameters
Method Name = createVmList
Method Name = writeDataColumn
Method Name = writeDataRow
Method Name = start
Methods under the Catagory --> Types too big
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
```

Figure 4.4: Methods mined by new algorithm

Figure 4.5 represents the frequent methods mined from jEdit5.0 version file using the Apriori algorithm. Figure 4.6(a) and Figure 4.6(b) represents the frequent methods mined from jEdit5.0 file using the proposed algorithm.



```
File Edit Format View Help
Methods under the Catagory --> Methods too big
Methods under the Catagory --> Methods having too many parameters
Method Name = setModeAbbrevs
Method Name = addModeAbbrev
Method Name = expandAbbrev
Method Name = saveAbbrevs
Methods under the Catagory --> Types too big
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Methods under the Catagory --> Types with too many methods
Method Name = void
Method Name = String
Methods under the Catagory --> Types with too many methods
Method Name = void
Method Name = String
Methods under the Catagory --> Types with too many methods
```

Figure 4.5: Methods mined by Apriori algorithm

```
File Edit Format View Help
Methods under the Catagory --> Methods too big
Method Name = start
Method Name = end
Method Name = getExecutionTimes
Method Name = sum
Method Name = listToArray
Methods under the Catagory --> Methods having too many parameters
Method Name = setModeAbbrevs
Method Name = addGlobalAbbrev
Method Name = addModeAbbrev
Method Name = expandAbbrev
Method Name = saveAbbrevs
Methods under the Catagory --> Types too big
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Methods under the Catagory --> Types with too many methods
```

Figure 4.6(a) Methods Mined by New Algorithm from jEdit 5.0 Version file

```
File Edit Format View Help
Methods under the Catagory --> Types too big
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Method Name = double
Methods under the Catagory --> Types with too many methods
Method Name = void
Method Name = String
Method Name = double
Methods under the Catagory --> Types with too many methods
Method Name = void
Method Name = String
Method Name = double
Methods under the Catagory --> Types with too many methods
Method Name = void
Method Name = String
Method Name = double
```

Figure 4.6(b): Methods mined by New algorithm

Table 4.1: Number of methods mined by Apriori Algorithm from jEdit4.3pre file

Methods	Number
Methods too big	17
Methods having too many parameters	4
Types too big	99
Types with too many methods	12

Table 4.2: Number of methods mined by New Algorithm of jEdit4.3 pre file

Methods	Number
Methods too big	45
Methods having too many parameters	2
Types too big	100
Types with too many methods	9

Figure 4.7 shows the number of frequent methods mined by new Algorithm from jEdit 5.0 version file. The methods of different categories are mined such as methods too big, methods having too many parameters, methods with types too big.

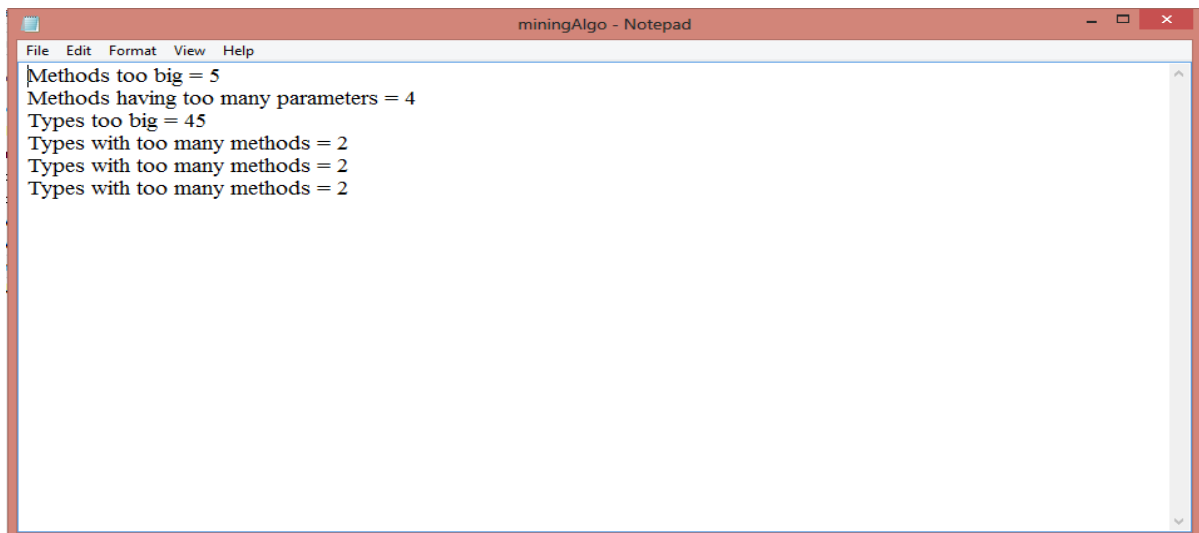


Figure 4.7: Number of methods mined by New Algorithm

Table 4.3: Number of methods mined by Apriori Algorithm from jEdit 5.0 file

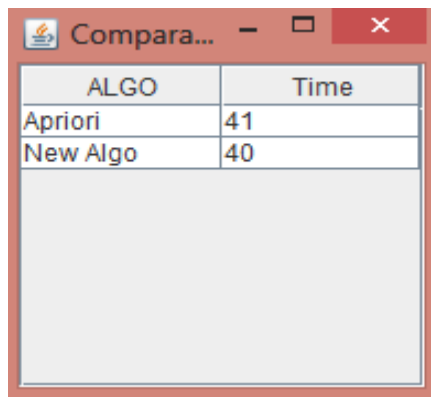
Methods	Number
Methods too big	5
Methods having too many paramaters	4
Types too big	45
Types with too many methods	2

Table 4.4: Number of methods mined by New Algorithm from jEdit 5.0 version file

Methods	Number
Methods too big	30
Methods having too many parameters	1
Types too big	45
Types with too many methods	2

Step 3: The patterns extracted are ranked by calculating the support count using heuristic approach in which the support count is modified every time the patterns are mined by considering the last support count and update the support vector.

Step 4: The different values which we get by mining using the new algorithm and Apriori is used to generate the comparison table for the consumption of space and time.



ALGO	Time
Apriori	41
New Algo	40

Figure 4.9: Comparison in terms of Time

Figure 4.9 shows the time taken to mine the jEdit version file by both Apriori algorithm and new algorithm and shows that new algorithm takes less time in mining. Figure 4.10 shows the graphical representation of comparison of time taken for mining by both Apriori algorithm and new proposed algorithm. Figure 4.11 shows the graphical representation of comparison of memory used for mining jEdit files by both Apriori algorithm and new proposed algorithm and shows that new algorithm used less memory as compared to Apriori algorithm.

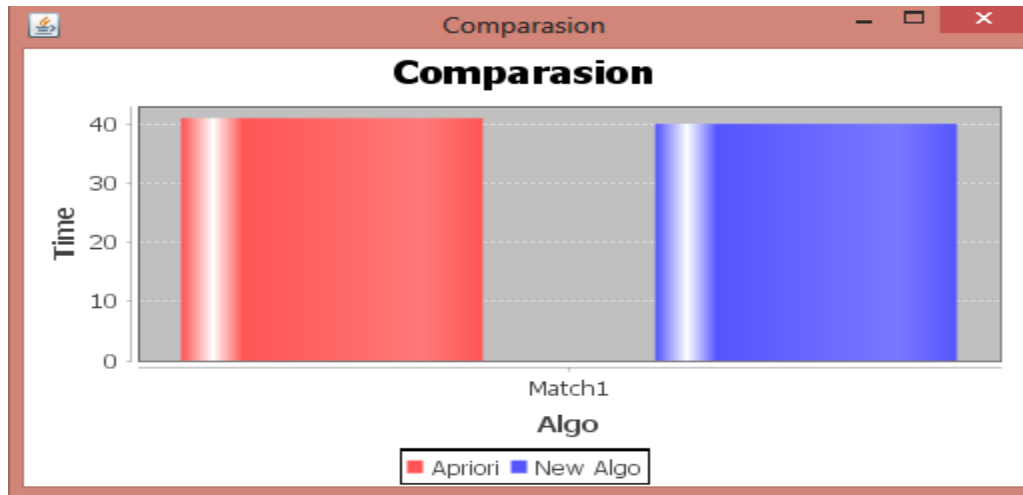


Figure 4.10: Graphical Representation of Time

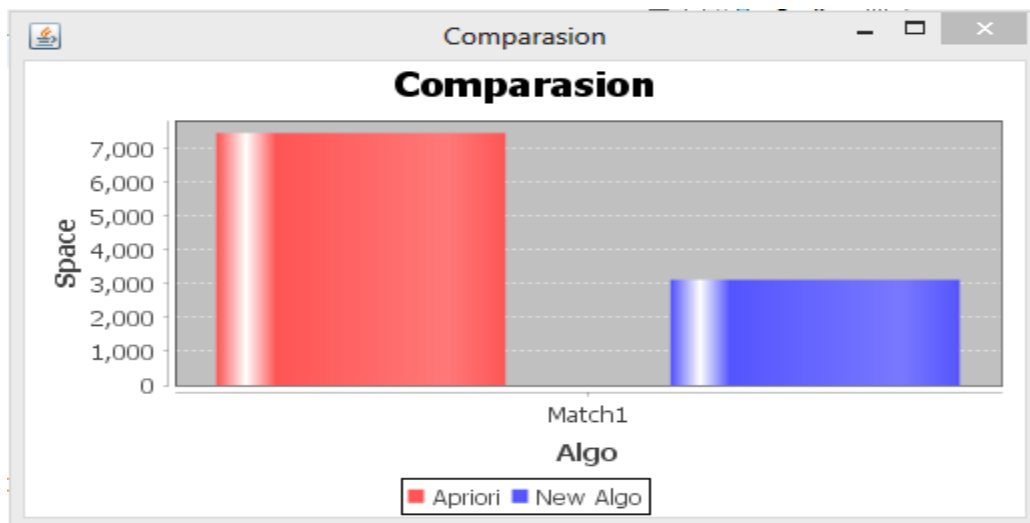


Figure 4.11: Graphical Representation in terms of Memory Required

5.1 Conclusion

The overview of software engineering data and various data mining techniques has been given in this report. It is widely described how the data mining techniques are used in software engineering for mining software engineering data to achieve software of good quality and high reliability.

A brief literature survey of the data mining techniques used to mine various software repositories has been provided. The report concludes with the brief objectives and methodology used to mine likely usage patterns by using an efficient data mining approach that has less execution time and requires less memory.

The new proposed algorithm used heuristic approach for mining usage patterns from code revision files and it was found that the number of methods mined by new algorithm is more as compared to Apriori algorithm. The new algorithm required less time and less memory for mining.

5.2 Future Work

- The work can be extended by using other pattern mining algorithms.
- Source code files of many open software revision histories in different languages can be mined for frequent usage methods.

REFERENCES

- Agrawal, R., & Srikant, R. (1994, September). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB* (Vol. 1215, pp. 487-499).
- Couto, C., Montandon, J. E., Silva, C., & Valente, M. T. (2013). Static correspondence and correlation between field defects and warnings reported by a bug finding tool. *Software Quality Journal*, 21(2), 241-257.
- Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences*, 264, 260-278.
- Hassan, A. E. (2006, September). Mining software repositories to assist developers and support managers. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on* (pp. 339-342). IEEE
- Hassan, A. E. (2008, September). The road ahead for mining software repositories. In *Frontiers of Software Maintenance, 2008. FoSM 2008.* (pp. 48-57). IEEE.
- Kim, S., Whitehead, E. J., & Zhang, Y. (2008). Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2), 181-196.
- Koçak, G., Turhan, B., & Bener, A. B. (2008). Predicting Defects in a Large Telecommunication System. In *ICSOFT (SE/MUSE/GSDCA)* (pp. 284-288).
- Livshits, B., & Zimmermann, T. (2006). DynaMine: Finding Common Error Patterns by Mining Software Revision Histories. *ACM Transactions on Programming Languages and Systems*, 2(3), 09.
- Madhavan, J. T., & Whitehead Jr, E. J. (2007, October). Predicting buggy changes inside an integrated development environment. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange* (pp. 36-40). ACM.
- Mishra, B., & Shukla, K. K. (2012). Defect Prediction for Object Oriented Software using Support Vector based Fuzzy Classification Model. *International Journal of Computer Applications*, 60(15).
- Mu, X. D., Chang, R. H., & Zhang, L. (2012). Software Defect Prediction Based on Competitive Organization CoEvolutionary Algorithm. *Journal of Convergence Information Technology, AICIT*, 7(5), 325-332
- Najadat, H., & Alsmadi, I. (2012). Enhance Rule Based Detection for Software Fault Prone Modules. *International Journal of Software Engineering and Its Applications*, 6(1), 75-86.

- Negara, S., Codoban, M., Dig, D., & Johnson, R. E. (2013). Mining fine-grained code changes to detect unknown change patterns.
- Nasreen, S., Azam, M. A., Shehzad, K., Naeem, U., & Ghazanfar, M. A. (2014). Frequent Pattern Mining Algorithms for Finding Associated Frequent Patterns for Data Streams: A Survey. *Procedia Computer Science*, 37, 109-116.
- Pelayo, L., & Dick, S. (2009). Predicting Object-Oriented Software Quality: A Case Study International Journal of Intelligent Control and Systems
- Prasad, A. K., & Krishna, S. R. (2010). Data Mining for Secure Software Engineering-Source Code Management Tool Case Study. *International Journal of Engineering Science and Technology (IJEST) ISSN, 0975-5462*.
- Park, M., & Hong, E. (2014). Software Fault Prediction Model using Clustering Algorithms Determining the Number of Clusters Automatically. *International Journal of Software Engineering & Its Applications*, 8(7).
- Puri, A., & Singh, H. (2014). Genetic Algorithm Based Approach For Finding Faulty Modules In Open Source Software Systems. *International Journal of Computer Science & Engineering Survey*,5(3).
- Qu, W., Jia, Y., & Jiang, M. (2010). Pattern mining of cloned codes in software systems. *Information Sciences*.
- Rodriguez, D., Herraiz Tabernero, I., & Harrison, R. (2012). On software engineering repositories and their open problems.
- Selvaraj, P. A., & Thangaraj, P.(2013) Support Vector Machine for Software Defect Prediction.
- Singh, A. K., Goel, R., Kumar, P., & NIET, G. N. Fault Prediction using Hybrid Fuzzy C-Means with Genetic Algorithm and KNN Classifier.
- Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Blasco, J. (2014). Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, 41(4), 1104-1117.
- Sun, C., Zhang, H., Lou, J. G., Zhang, H., Wang, Q., Zhang, D., & Khoo, S. C. (2014, November). Querying sequential software engineering data. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 700-710). ACM.
- Tarvo, A. (2008, November). Using statistical models to predict software regressions. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on* (pp. 259-264). IEEE

- Taylor, Q., Giraud-Carrier, C., & Knutson, C. D. (2010). Applications of data mining in software engineering. *International Journal of Data Analysis Techniques and Strategies*, 2(3), 243-257.
- Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., & Zhai, C. (2013). Bug characteristics in open source software. *Empirical Software Engineering*, 1-41.
- Tekin, U., & Buzluca, F. (2013). A graph mining approach for detecting identical design structures in object-oriented design models. *Science of Computer Programming*.
- Thomas, S. W., Adams, B., Hassan, A. E., & Blostein, D. (2014). Studying software evolution using topic models. *Science of Computer Programming*, 80, 457-479.
- Toor, A. (2014). An Advanced Clustering Algorithm (ACA) for Clustering Large Data Set to Achieve High Dimensionality. *Global Journal of Computer Science and Technology*, 14(2).
- Velmurugan, T. (2014). Performance based analysis between k-Means and Fuzzy C-Means clustering algorithms for connection oriented telecommunication data. *Applied Soft Computing*, 19, 134-146.
- Verma, A., & Kumar, A (2014). Performance Enhancement of K-Means Clustering Algorithms for High Dimensional Data sets. *International Journal of Advanced Research in Computer Science and Software Engineering* 4(1), pp. 791-796
- Walden, J., Stuckman, J., & Scandariato, R. (2014). Predicting Vulnerable Components: Software Metrics vs Text Mining. *status: accepted*.
- Wang, H. (2014). Software Defects Classification Prediction Based On Mining Software Repository.
- Xie, T., Pei, J., & Hassan, A. E. (2007, May). Mining software engineering data. In *Software Engineering-Companion, 2007. ICSE 2007 Companion. 29th International Conference on* (pp. 172-173). IEEE.
- Yousef, A. H. (2014). Extracting software static defect models using data mining. *Ain Shams Engineering Journal*.
- Zhong, N., Li, Y., & Wu, S. T. (2012). Effective pattern discovery for text mining. *Knowledge and Data Engineering, IEEE Transactions on*, 24(1), 30-44.
- Zimmermann, T., Zeller, A., Weissgerber, P., & Diehl, S. (2005). Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6), 429-445.

LIST OF PAPERS

Meghna Soni, Harshpreet Singh and Nisha Sethi. Article: A state of the art survey of Data mining techniques for Software Engineering Data. *International Journal of Applied Engineering and Research (IJAER)*. (Accepted)