ON THE EFFECT OF DATA MINING TECHNIQUES ON RECOMMENDING SOURCE CODE CHANGES

by

Saeed KHALILAZAR

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A. Sc.

MONTREAL, MARCH 22, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Ms. Latifa Guerrouj, Thesis Supervisor
Department of Software and IT Engineering, École de technologie supérieure

Mr. Luc Duong, President of the Board of Examiners
Department of Software and IT Engineering, École de technologie supérieure

Mr. Ali Ouni, Member of the jury
Department of Software and IT Engineering, École de technologie supérieure

THIS THESIS WAS PRENSENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

22 MARCH 2022

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

SUR L'EFFET DES ALGORITHMES DE FORAGE DE DONNÉES SUR LES RECOMMANDATIONS DES CHANGEMENTS DE CODE SOURCE

Saeed KHALILAZAR

RESUME

Des recherches passées et récentes ont tiré parti du forage de données pour créer des approches et des techniques qui peuvent guider les développeurs lors de leurs modifications du code source. A notre connaissance, très peu de travaux ont étudié des techniques avancées d'exploration de données (par exemple autres qu'Apriori) et/ou ont comparé leurs résultats avec d'autres algorithmes ou un référentiel de base.

Dans cet article, nous proposons une approche automatique pour recommander des changements de code source à l'aide de quatre algorithmes de forgae de données, Apriori, FP-Growth, Eclat et Relim. Nous considérons Apriori, l'algorithme de forage de données largement adopté, comme notre référentiel de base. Nous recommandons non seulement des changements de code source en utilisant ces quatre algorithmes de data mining, mais nous fournissons également une évaluation empirique de leurs performances en utilisant différentes configurations et explorons comment ces différentes configurations affectent la pertinence des recommandations produites.

Notre étude empirique implique sept projets open-source à partir desquels nous avons extrait l'historique des changements de source au niveau du fichier. Nous avons comparé les résultats en termes de précision, de rappel et de F-mesure en considérant comme référence l'algorithme Apriori.

Nos résultats apportent des preuves empiriques sur le fait que bien que certains algorithmes avancés puissent dans certains cas être plus performants que des algorithmes de base tels qu'Apriori, les résultats dépendent de l'historique des changements, du type de techniques d'exploration de données appliquées, de la nature et des caractéristiques des projets, y compris

du total. nombre de transactions. Nous pensons que la communauté des chercheurs travaillant dans ce domaine peut tirer parti de ces résultats lors de la sélection d'algorithmes de forage de données pour créer leurs recommandations pour les changements du code source.

Mots-clés: Changements du code source, Systèmes de recommandation, forage de données, historique des changements du code source, Apriori, FP-Growth, Eclat, ReLim, étude empirique.

ON THE EFFECT OF DATA MINING TECHNIQUES ON RECOMMENDING SOURCE CODE CHANGES

Saeed KHALILAZAR

ABSTRACT

Past and recent research has leveraged data mining to build approaches and techniques that can guide developers during their source code changes. To the best of our knowledge, very few works have investigated advanced data mining techniques (e.g., FP-Growth, Relim, or Eclat, etc.) and--or compared their results with other algorithms or a baseline.

In this paper, we suggest an automatic approach to recommend source code changes using four data mining algorithms, Apriori, FP-Growth, Eclat and Relim. We consider Apriori, the widely-adopted data mining algorithm, as our baseline. We not only recommend source code changes using these four data mining algorithms, but we also provide an empirical evaluation of their performances using different configurations and explore how these different configurations affect the relevance of the produced recommendations.

Our empirical study involves seven open-source projects from which we have extracted source change history at the file level. We have compared the results in terms of precision, recall, and F-measure by considering as our baseline, the Apriori algorithm.

Our findings bring empirical evidence on the fact that although some advanced algorithms may, in some cases, perform better than basic algorithms such as Apriori, the results depend on the change history, type of applied data mining techniques, the nature and characteristics of the projects including total number of transactions. We believe the research community working in this area can leverage these findings when selecting data mining algorithms to build their recommenders for source code changes.

Keywords: Source code changes, Recommendation systems, data mining, source code change history, Apriori, FP-Growth, Eclat, ReLim, empirical study.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABREVIATIONS

ETS	École de technologie supérieure
SVM	Support Vector Machine
MI	Mining programmer Interaction histories
API	Application Programming Interface
PPM	Prediction by Partial Matching
MAPO	Mining API usage Pattern from Open-source repositories
MDGs	Module Dependency Graph
WMC	Weighted Methods per Class
CBO	Coupling Between Objects
CVS	Concurrent Versioning System
MSR	Mining Software Repositories
IDE	Integrated development environment
AST	Abstract Syntax Tree
OSS	Open-Source Software
EM	Expectation-Maximization
PDGs	Program-Dependence Graphs
NB	Naive Bayes
MRR	Mean Reciprocal Rank
SVN	Apache Subversion

INTRODUCTION

In recent years, developers accomplish their tasks and do their activities by means of Recommendation Systems. These kinds of tools prepare and provide sufficient data for developers which are appropriate and relevant to the context of their tasks. In order to prevent re-inventing the wheel, some piece of valuable information such as reusable code snippets, method invocations from external libraries and resolution of reported bugs could be provided to developers with the help of recommendation systems. Recommendation systems ability to explore through large volume of produced information enhances its capability to present customized contents for individual user of recommendation system.

Exploring research contributions presents multiple approaches with different goals and functionalities such as traditional static and dynamic analysis techniques, IR-based methods and mining software repositories.

The utilization of data mining algorithms for the purpose of recommending source code changes has been continuously expanding in many areas including industrial, scientific and commercial ones (Borg et al., 2017; Robillard et al., 2009; Ponzanelli et al., 2017). Software developers must consider modification tasks in software systems in terms of dependencies between different parts of the source code such as files that were changed together regularly (Agrawal & Horgan, 1990; Parnas, 1972; Weiser, 1984). To supplement these considerations, we investigate various advanced association rules mining algorithms that produce recommendations for source code changes using four data mining algorithms, ie Apriori, FP-Growth, Relim, and Eclat.

Our work is inspired by Zimmerman et al (2004) and Ying et al (2004) and it is an extension of the work done by our lab member, Pierre Poilane in this area (Pierre Poilane, 2017). Zimmerman et al (2004) and Ying et al (2004) used Apriori and FP-Growth algorithms respectively to predict file change patterns, while Pierre Poilane has leveraged both Apriori and FP-Growth to recommend source code changes. Similarly, to these works, we predict source code file changes by leveraging data mining. However, we explore four advanced data mining techniques, as well as different configurations of each algorithm along with the

performances corresponding to each configuration. Furthermore, we perform a comparison between our examined algorithms by considering Apriori (used in Zimmerman et al (2004)) as our baseline. To the best of our knowledge, we are the first to investigate the use of four different advanced data mining algorithms: Apriori, FP-Growth, Relim, and Eclat to predict changes in source code files.

In this research, we address two main research questions:

- **RQ1:** How do the four studied advanced data mining techniques, i.e. Apriori, FP-Growth, Eclat and Relim perform in terms of Precision, Recall and F-measure?
- **RQ2:** Are there any differences between the four studied advanced data mining techniques, i.e. Apriori, FP-Growth, Eclat and Relim in terms of their performances when recommending source code file changes?

Our empirical research consists of seven open-source projects: Eclipse, ElasticSearch, Rhino, SWT, Kotlin, Guava and JabRef, from which we extract the change history at the file level. We chose these projects because they are different in size and domain. In addition, they have a reasonable source code change history that can be used for our study purposes.

We show through our empirical investigation that while some advanced algorithms can perform better in some cases than basic algorithms such as Apriori, the results depend on the change history, type of applied data mining techniques, the nature and characteristics of the projects including total number of transactions.

In thesis, we will discuss some principal concepts of recommendation systems in software engineering, literature review, their goals and functionality as well as different approaches and algorithms used in developing recommendation systems. Furthermore, we will introduce some proposed implemented recommendation systems in further details as well as their comparison considering used approaches and results.

We can categorize recommendation in terms of their goals and functionality to different categories such as supporting developers in change tasks, API usage, reusable software

components, etc. More technically, it can be clustered based on recommendation filtering techniques as follows (Isinkaye, Folajimi, & Ojokoh, 2015):

- Content-based filtering technique,
- Collaborative filtering technique: Model-based filtering techniques (Clustering techniques, Association techniques, Bayesian network, Neural network), Memory-based filtering techniques (User-based, Item-based),
- Hybrid filtering technique.

Malheiros et al., (2012) propose Mentor which helps newcomers in order to resolve change requests by source code file recommendation. It utilizes the Prediction by Partial Matching (PPM) algorithm as well as some heuristics with purpose of data mining within version control system and change request analysis in order to provide relevant source code recommendations that assist developers to handle change requests. The programming language is not important for the tool whether it is written in C, C++, Java or mix programming languages. In order to find similar change requests, it makes use of the Support Vector Machine (SVM) classifier (Malheiros, Moraes, Trindade, & Meira, 2012).

One of the important aspects of recommendation systems to consider is the level of granularity which shows what level and details will a recommendation system addresses. Lee & Kim (2015) explored characterization of the contexts of files to edit with the help of histories with purpose of using finer-grained association rules. They extended ROSE, an edit-based recommendation system and developed MI (Mining programmer Interaction histories). There are two paradigms for history-based recommendation systems, one is based on mining software revision histories. Zimmerman et al (2004) and Ying et al (2004) proposed their recommendation systems which recommends files to be edited considering mined software revision histories. Association rules among most edited files together are mined then file-to-edit are recommended (Lee & Kim, 2015). One pitfall in this approach is failure in recommendation when the programmer has not edited a file up to moment. In MI as a recommender system, files are recommended based on both the viewed files and the edited files. Furthermore, they have extended MI capability to mine programmer interaction histories

that means files to edit are recommended using mining association rules in programmer interaction in either viewed of edited files. Mining rules using MI outperforms the rules mined from edit histories in ROSE which shows that MI provides higher recommendation accuracy than ROSE (Zimmerman et al., 2004).

Another level of granularity examined by researchers is the API methods level. In effect, (Zhong, et al., 2009) have developed an API usage mining tool called MAPO (Mining API usage Pattern from Open-source repositories) which automatically mines API usage patterns. MAPO clusters code snippets based on the level of similarity to each other which shows various usages of different clusters. Also, MAPO organizes API usage patterns and recommends code snippets based on that organization. Holmes et al., (2005) proposed Strathcona which suggests code snippet by comparing developing and stored code snippet in repository and lists relevant code snippets (Holmes et al., 2005).

We focus on source code changes issued for developers in this study, we investigate various advanced data mining algorithms that produce recommendations for source code changes using four data mining algorithms, followed by an empirical study in order to measure the performance of those techniques with seven different projects. In the next chapter, we investigate four data mining algorithms.

CHAPTER 1

TECHNICAL BACKGROUND

In this section, we provide an overview of the data mining algorithms that we have applied in our study, i.e., Apriori, FP-Growth, Eclat and Relim.

1.1 Apriori

Apriori was the first data mining algorithm that was proposed for frequent itemset mining by Agrawal and Srikant (1994). It is known to be a reference and one of the most used association mining rule algorithms to find sets of items from transactions (Agrawal and Srikant, 1994). This algorithm uses two steps “join” and “prune” to reduce the search space. It is a level-wise iterative approach to discover the most frequent itemset (Iqbal et al., 2008). Apriori starts by generating candidates, i.e., potential sets of frequent items. The list of candidates in the second step are taken as input and checks if they are common. To do this, each of the candidates from the transactions is verified by the calculation of its support and if equal to or greater than the minimum support defined, the candidate will be considered a frequent item and will be part of the list of frequent sets. An extended list of frequent candidates is then generated and the process continues (starting from step 1) to build the next list of candidates, and so on. Finally, when the last generated list of candidates is empty, the algorithm ends.

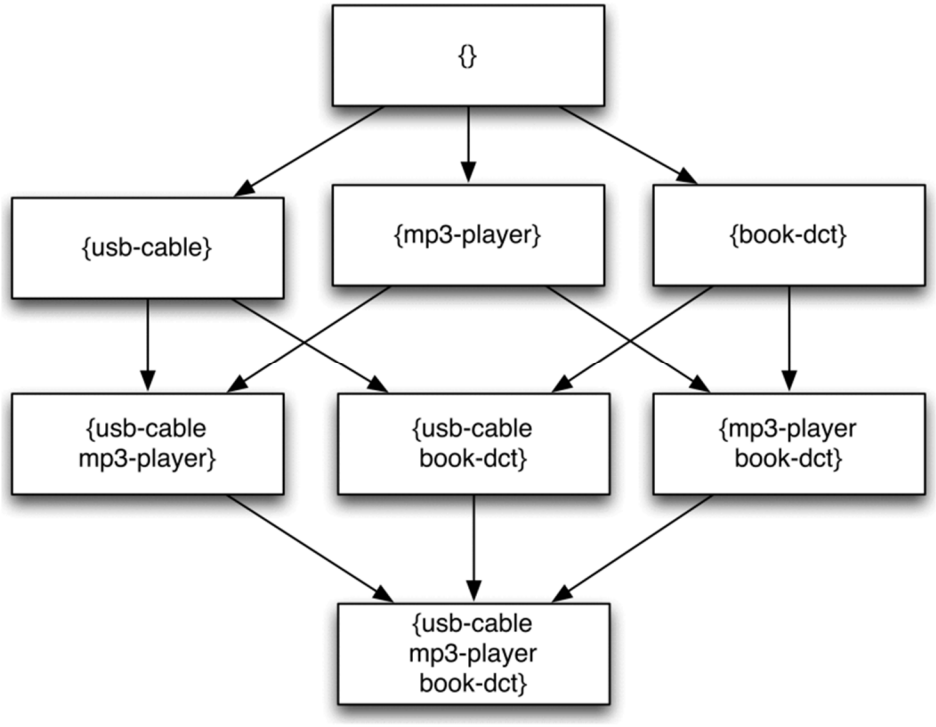

Figure 1.1 Apriori breadth-first search extracted from Heaton (2016)

In Figure 1.1, a real example of Apriori technique is illustrated. As it can be seen, Apriori uses a breadth first search for scanning the itemsets and begins by an empty candidate set. Then, a list of all singleton itemsets is created. A combination of the singleton itemsets builds the next set of candidates frequent itemsets. The itemsets including minimum support will be added to the candidate set. For example, usb-cable, mp3-player, and book-dct are all singleton itemsets with sufficient support. Minimum support for usb-cable, mp3-player, and book-dct are 4, 3 and 3, respectively. As mentioned earlier, K+1 itemset are built based on K itemset from previous layer as shown in the figure. Combinations of all singleton itemsets are created in the next layer. In this example, all 2-itemsets combinations have minimum support as follows. Minimum support for “usb-cable, mp3-player”, “usb-cable, book-dct” and “mp3-player, book-dct” are 3, 2 and 2, respectively which is sufficient to proceed to the next level. Then in the next layer, a triplet itemset with all three items for 3-itemsets will be evaluated in terms of minimum support in order to decide whether to end the algorithm or to continue. In this

example, minimum support is set two and “usb-cable, mp3-player, book-dct” hold sufficient minimum support of two (Heaton, 2016).

1.2 FP-Growth

This algorithm is an improvement to the Apriori method introduced by Han et al. (2000). The algorithm follows divide-and-conquer strategy (Han and Kamber, 2006). A frequent pattern is generated without the need for candidate generation. FP-Growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree. It constructs an FP Tree rather than using the generate and test strategy of Apriori. As illustrated in Figure 1.2, FP tree consists of a null root node and child nodes (Alzahrani et al., 2015). The focus of the FP Growth algorithm is on fragmenting the paths of the items and mining frequent patterns.

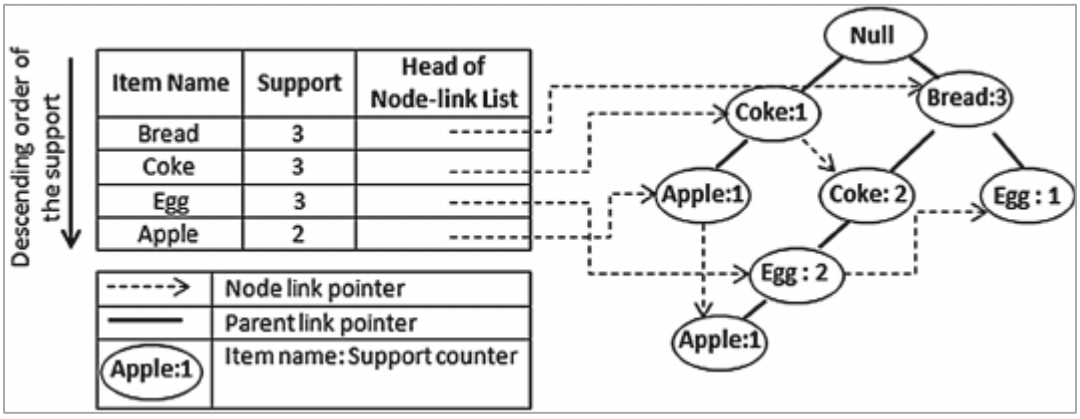

Figure 1.2 A FP-tree extracted from Alsulim et al. (2015)

1.3 Eclat

(Zaki, 2000) introduced Eclat (Equivalence Class Transformation) which is a mining frequent itemsets algorithm. Using a vertical data format, Eclat, mines and groups all the transactions having particular itemset into the same record. In the first step, database is scanned and data will be transformed from horizontal to vertical format. This process is illustrated in the following tables (Table 1.1, Table 1.2, Table 1.3)

Table 1.1 Transactional data in vertical data format
extracted from Chee et al. (2019)

Itemset	TID set
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Table 1.2 Two-Itemsets in vertical data format
extracted from Chee et al. (2019)

Itemset	TID set
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

Table 1.3 Three-itemsets in vertical data format
extracted from Chee et al. (2019)

Itemset	TID set
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

Then, by means of intersecting the transactions of the frequent k-itemsets, it produces frequent (k+1)-itemsets. Once there will be no more itemset generated, it will continue iterations. Unlike other algorithms, it is not necessary for database to be scanned multiple times. Eclat scans the database only once and data will be transformed from horizontal to vertical (Chee et al., 2019). Using a vertical database layout, this depth-first search algorithm can easily calculate the support of an itemset (Pramod & Vyas ,2010).

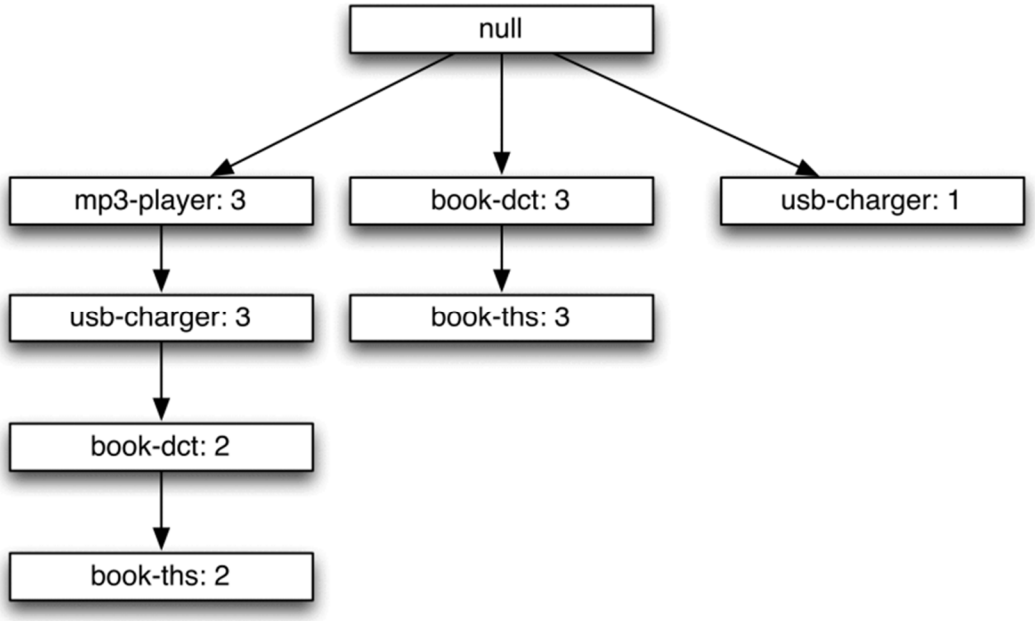

Figure 1.3 An example of Eclat extracted from Heaton (2016)

This process is better illustrated using an example in Figure 1.3. Eclat uses a structure called a trie graph, in which the root node is always empty. Itemsets are inserted into the trie graph from left, in which the left-most item corresponds to a child of the root node. Mp3-player and usb-charger are having minimum support of three, while usb-charger is holding 1 as the value of minimum support. Likewise, mp3-player, usb-charger, book-dct are identified as frequent itemset due to the support value of two. All paths are created based on encountered transactions. Eclat traverses the whole graph in the same manner and extracts all frequent itemsets. For example, as it can be seen, 4-itemsets of “mp3-player, usb-charger, book-dct, book-ths” which is the left-most path in the trie graph, is representing 4-itemsets with minimum support of two. Likewise, second path in the graph is presenting 2-itemsets of “book-dct, book-ths” with minimum support of three (Heaton, 2016).

1.4 ReLim

Borgelt et al., 2005 developed the Relim algorithm, like numerous other algorithms in the search for frequent elements, based on recursive deletion of elements from transactional database (Borgelt et al., 2005). ReLim begins by forming a frequency table using the frequency of each item, then infrequent items are dropped because those are not among repeated items (Figure 1.4).

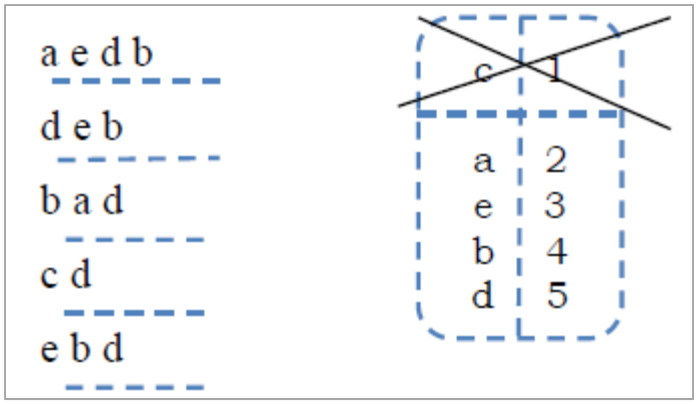

Figure 1.4 Transaction table (left), frequent table (right)
extracted from Fakir et al. (2020)

Next step is to create initial database based on frequency of the first item of each itemset that includes the first item of each itemset as well as its frequency. In the next step, the items with the lowest frequency are eliminated. The above-mentioned process repeats until reaching a final list which is keeping the null frequencies except the last one which finishes the process. This process is depicted in Figure 1.5.

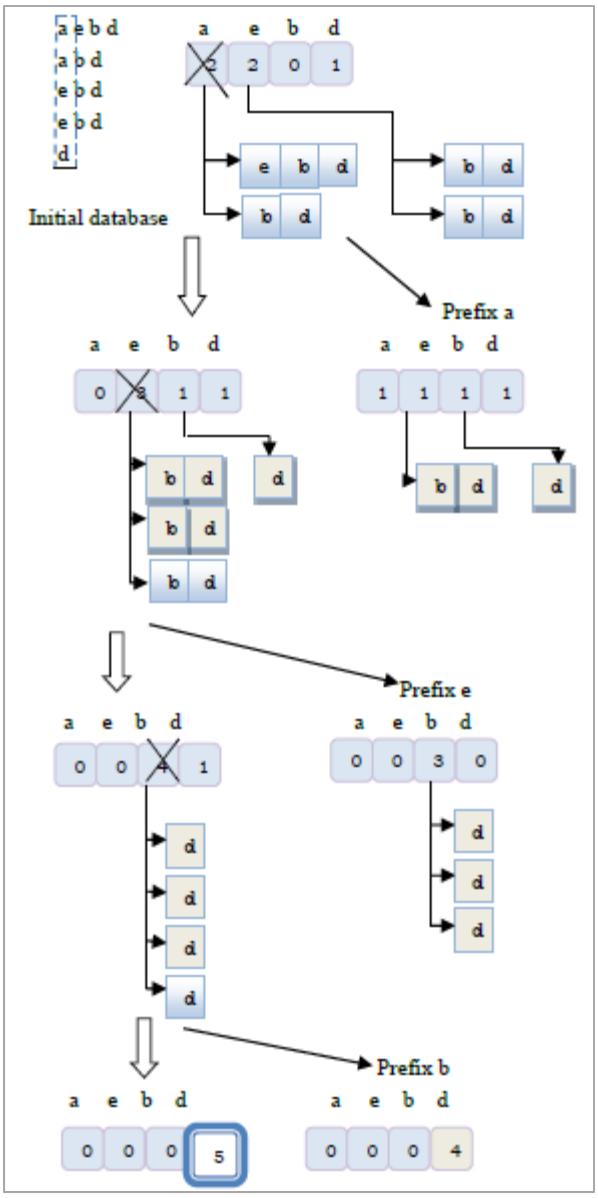

Figure 1.5 The process of the Relim algorithm
extracted from Fakir et al. (2020)

ReLim's major strength is the simplicity of its structure since everything is done in a recursive function (Fakir et al., 2020). On the other hand, the structure of ReLim is built in a time-consuming manner as well as consuming many system resources. ReLim algorithm is different from other extracting frequent items algorithms in terms of simple processing and its data structure (Fakir et al., 2020).

1.5 Comparison of Frequent Pattern Mining algorithms

There are different kinds of association rule mining depending upon the type of itemsets generated during the itemset generation phase e.g., Frequent itemset generation, closed frequent itemset generation, Maximal frequent itemset generation (Borah & Nath, 2021). Among the Frequent itemset generation algorithms, Apriori (Agrawal & Srikant, 1994) is the first algorithm generated in 1994. For this study, we considered Apriori as the baseline and three more algorithms for comparison purposes, i.e., FP-Growth, Eclat and Relim. These algorithms were selected because they are among the most frequently cited studies referenced in the literature (Bayardo, 1998; Borah & Nath, 2017; Borah & Nath, 2018).

Various researchers have done many experimental tests in order to evaluate the performance of Frequent Pattern Mining algorithms considering advantages and disadvantages in terms of execution time, memory consumption and number of generated itemsets. Additionally, important performance measures including precision and recall are critical to evaluate those algorithms. In our study, four algorithms have been chosen from different categories to evaluate on seven different datasets from popular projects. Advantages and disadvantages of the chosen algorithms are illustrated in following in Table 1.4. Generally, three main classifications of Frequent Pattern Mining algorithms are Join-Based, Tree-Based, and Pattern Growth. In this study, all the categories are covered with Apriori which is Join-Based in nature, Eclat is classified as Tree-Based algorithm, as well as FP-Growth which is considered as a Pattern Growth algorithm (Chee et al., 2019).

Table 1.4 Advantages and disadvantages of investigated algorithms

Investigaed Algorithms	Advantages	Disadvantages
Apriori (Agrawal and Srikant, 1994)	- Applying an iterative level-wise search technique which results in (k+1)-itemsets discovery from k-itemsets (Chee et al., 2019)	- Being time-consuming while generating candidate - Needing numerous scans on the databases while performing - Generating redundant rules (Mythili & Shanavas, 2013)
Eclat (Zaki, 2000)	- The database is not required to be scanned multiple times in order to identify the (k+1)-itemsets (Chee et al., 2019). - The database is also not required to be scanned multiple times in order to identify the support count of every frequent itemset (Chee et al., 2019).	- Take extensive memory space and processing time for intersecting the itemsets (Chee et al., 2019).
FP-Growth (Han and Pei, 2000)	- Maintaining the association information of all itemsets - Reducing target data to be scanned and searched (Chee et al., 2019)	- For large datasets, FP-tree construction is time-consuming (Chee et al., 2019)
ReLim (Fakir et al., 2020)	- Simplicity of its structure (Applying recursive function) (Fakir et al., 2020)	- Consuming many system resources - Time-consuming (Fakir et al., 2020)

(Pramod & Vyas ,2010) showed that Relim is outperforming FP-Growth in terms of execution time for various support thresholds. (Mythili & Shanavas, 2013) revealed that Apriori consumes large memory in comparison to FP-Growth due to candidate generation. Additionally, they evaluated the performance of Apriori and FP-Growth algorithms and revealed that the FP-Growth is outperforming Apriori in terms of execution time for various values of support, based on their two datasets.

CHAPTER 2

RELATED WORK

A large body of past and recent research works have leveraged data mining to gain knowledge and insights from various data sources. In the following, we divide our related work into two sections. The first section presents the relevant research works that have leveraged source code change history, while the second section states the most relevant works to our research, from the software engineering literature, on recommendation systems that have been developed to guide software developers during their engineering tasks.

2.1 Research works that leveraged source code change history

(Kovalenko, Palomba, & Bacchelli, 2018) defined and approach which tries Mining Software Repository (MSR) using two different algorithms. One algorithm only follows the first parent of each commit when traversing the repository, while the other returns the full modification history of a file across all branches The influence of the file history retrieval method on accuracy of recommendations is evaluated based on the history of changes using Mean Reciprocal Rank (MRR) and top-k precision. Inference of association rules are implemented using the Apriori algorithm.

McIntosh et al. (2014) proposed Basic Model Attributes in which Number of Files, Prior Build Co-Changes, (Source/Test) File added, deleted or renamed are measures for improving existing prediction models (Macho et al., 2016). In our research work, our granularity level is the file level and we have selected projects that include thousands of files. Developers have to be vigilant while changing files in each commit. Additionally, we investigate four advanced data mining algorithms, in particular, Apriori, FP-Growth, ReLim, and Eclat which operate at the file level since built on top of a prioro work that has been already started by the research group and that the aimis to extend it to other fine-grained levels such as the method level.

(Kovalenko, Palomba, & Bacchelli, 2018) defined and approach which tries Mining Software Repository (MSR) using two different algorithms:

- One algorithm only follows the first parent of each commit when traversing the repository;
- The other returns the full modification history of a file across all branches.

The influence of the file history retrieval method on accuracy of recommendations is evaluated based on the history of changes using the following metrics: Mean Reciprocal Rank (MRR) and top-k precision. Inference of association rules are implemented using the Apriori algorithm. In our study, four advanced data mining algorithms have been empirically investigated and their performance has been measured in terms of Precision, Recall and F-measure.

(Canfora & Cerulo, 2005) proposed a method to derive the set of source files impacted by a proposed change request. In order to link between change request and revisions of source file changes in a versioning system, namely CVS, and in a bug tracking system, Bugzilla, this method relies on information retrieval algorithms. Also, based on explicit traceability relationship, they achieved traceability analysis which identifies affected software objects.

With evolution of software systems, there will be an increase in the number as well as the complexity of interactions in code which results in, challenges for developers, when identifying the impact of changes. One solution is using change impact analysis aims to find artifacts (e.g., files, methods, classes) affected by a given change.

Canfora and Cerulo considered the file level but suggest that considering different levels of granularity of impacted entities such as program and architectural entities such as modules improves the results. In our study, we also investigate source code changes at the file level of granularity with four advanced data mining techniques in order to evaluate their performance. The results show that their approach got a precision that ranges between 30% and 78%. There are not many papers which infer and use traceability links between artifacts in bug repositories and source code artifacts via Mining Software Repositories (MSR).

In (Canfora & Cerulo, 2005), it is assumed that there is a dependency between artifacts which is a limitation of existing impact analysis techniques. In order to overcome such a limitation,

approaches such as those based on information retrieval that leverage the textual content of the artifacts have been proposed. Also, this approach has a weakness which is failing in exploring links in between when similarity is low, or finding false positives when artifacts seem similar but in fact, they are not relevant. It seems that approach used in this research work is (formal) language/artifact centric (e.g., static, and dynamic dependencies such as call graphs) which could be replaced by developer/human centric approaches (e.g., comments and identifiers, and commit practices). In our study, we consider investigation of the performance of four data mining algorithms when recommending source file changes.

Other studies went further by using the context of the developer's work. For example, (Ponzanelli, et al., 2017) used web browsing data navigated by developers along with source code changes, as well as the semantic relationships of the resources used by developers to help build a holistic recommendation system that can assist software developers. However, in our study, association rules are generated and reported based on changes committed by developers according to various advanced data mining algorithms using several configurations of each data mining techniques and comparisons among them.

(Sisman & Kak, 2012) focuses on finding bugs using the development history of projects. Specifically, the authors focused on the frequency of files that could be associated with defects and changes from history, which can be then used to construct estimates of the prior probability that a given file would be the source of bugs. We also leverage source code change history but for the purpose of recommending source code file changes using different advanced data mining techniques with different configurations.

(Malik & Shakshuki, 2010) have relied on different heuristics, including the analysis of related entities if they have similar files. By combining these heuristics, the researchers have demonstrated that source code changes can be used to generate recommendations for functions. In our study, recommendations are generated based on files changes rather than functions, using advanced data mining techniques rather than heuristics. Using frequent itemsets mining techniques such as Apriori, FPGrowth, Eclat and ReLim, we are able to identify files changed together.

(Heo, Oh, & Yang, 2019) proposed an algorithm for learning a controller for abstraction coarsening. The learning algorithm is inspired by batch mode reinforcement learning. The algorithm solves the problem in approximately using heuristics from the reinforcement learning community. Using the decision tree algorithm in the scikit-learn package, they have implemented supervised learner.

(Malheiros, Moraes, Trindade, & Meira, 2012) propose Mentor which helps newcomers in order to resolve change requests by source code file recommendation. It uses the Prediction by Partial Matching (PPM) algorithm as well as some heuristics with purpose of data within version control system and change request analysis in order to provide relevant source code recommendation that assists developers to handle change requests. In order to find similar change requests, it makes use of the Support Vector Machine (SVM) classifier.

The above-mentioned studies have not considered the temporal dimension. However, (Robbes, Pollet, & Lanza, 2010) have conducted research in which they recorded all developer interactions with the IDE, allowing them to track back the time spent by each developer to perform his/her changes.

We share with these research works the idea that recommendation systems can be useful to guide and support developers when performing software maintenance and evolution tasks. Similarly, to these works, we leverage source code change history to build our recommenders.

2.2 Recommendation systems developed to guide software developers

(Ying, Murphy, Ng, & Chu-Carroll, 2004) developed an approach using FP-Growth to predict file change patterns. Several other works leveraged change history to guide developers during their software changes. In effect, (Uddin, Dagenais, & Robillard, 2011) has developed a recommender that identifies API changes using a temporary API usage pattern. Likewise, in our thesis, the performance of Apriori, FP-Growth, Relim and Eclat algorithms, are measured in terms of Precisions and Recall, and F-measure using various configurations of support and confidence for each examined algorithm.

Zimmermann et al., (2005) proposed ROSE which analyzes the full history of a project to predict the location of most likely further changes and uses the Apriori Algorithm in order to compute association rules. This algorithm computes the set of all association rules, taking a minimum support and minimum confidence. Through this approach, they could detect coupling between program parts which are developed in different languages that shows it is not limited to a specific programming language. They developed an Eclipse plug-in that explores source code changes retrieved from repositories in order to help developers with possible co-changes (Zimmermann, Zeller, Weissgerber, & Diehl, 2005).

They proposed $P(A \rightarrow B) = \frac{N(A \cap B)}{N(A)}$ to compute the probability of impacted source code unit in terms of association rule. Because it assumes that support value term only includes intentional co-changes, A and B were modified at the same time for the same reason. Considering a situation in which A and B might be related to separate features that are included in the same release shows that the aforementioned assumption is not held necessarily. It is possible that, A and B edit at the same time, is not associated with A change because it has been changed several times in the past.

Using historical data occurs at different granularity levels, such as coupling between files and classes as well as detecting coupling between fine-grained program entities functions and variables which is implemented in (Zimmermann, Zeller, Weissgerber, & Diehl, 2005) research work. These data enable us to propose suggestion about possible relevant source code to a developer during her task accomplishment.

They suggested the usage of fine-grained co-change dependencies on Module Dependency Graph (MDGs) which is a graph representation of software. Using data mining to extract the co-change coupling, some quantitative code metrics such as LOC, Weighted Methods per Class (WMC) and Coupling Between Objects (CBO) using static impact analysis have been provided.

The results show that for stable systems, ROSE delivers many and precise suggestions: 44% of related files and 28% of related entities can be predicted, with a precision of about 40% for each single suggestion, and a likelihood of over 90% for the three topmost suggestions. But

for changing and evolving systems will deliver better suggestions at the file level. When ROSE warns it should be considered seriously because just 2% of all transactions results in false alarms.

In our study, we have increased the number of data mining algorithms to four by adding FP-Growth, Relim, and Eclat, whose performance is measured in terms of precision, recall, and F-measure using various configurations of support and confidence. Additionally, we have empirically evaluated our approach using seven different open-source projects.

(Martinez & Monperrus, 2019) presented, Coming, which is a tool that takes as input a Git repository and mines instances of code change patterns present on each commit. It first applies an AST-diff algorithm to compute the code changes between them at a fine-grained level. It does a fine-grained comparison of each revision pair by comparing their ASTs (Abstract Syntax Tree).

Several works in the software engineering literature have leveraged development history to build data mining-based recommenders. (Palomba, et al., 2017) have used it for the detection of bad smells in the source code. Their system for called HIST (Historical Information for Smell Detection), it uses data from change history in order to look for the presence of bad practices in the source. The results generated by HIST have shown that the use of development history as a data source provide promising results.

(Dotzler, Kamp, Kreutzer, & Philippsen, 2017) proposed ARES which gets higher accuracy because of using algorithms that take care of code movements when creating patterns and recommendations. It uses a tree differencing algorithm that extracts the differences between two abstract syntax trees (ASTs) as well as MTDIFF, the currently most precise tree differencing algorithm that considers code movements.

(Zhang, Upadhyaya, Reinhardt, Rajan, & Kim, 2018) designed ExampleCheck, an API usage mining framework that extracts patterns through combining frequent subsequence mining and SMT-based guard condition mining to retain important API usage features. Global-Aware Recommendations for Repairing Violations in Exception Handling The authors presented

RAVEN, a heuristic strategy aware of the global context of exceptions that produces recommendations of how violations in exception handling may be repaired.

(Martinez & Monperrus, 2019) presented, Coming, which is a tool that takes as input a Git repository and mines instances of code change patterns presented on each commit. It first applies a ASTdiff algorithm to compute the code changes between them at a fine-grained level. It performs a fine-grained comparison of each revision pair by comparing their ASTs (Abstract Syntax Tree). (Nguyen, et al., 2019) presented FOCUS which is a Context-aware collaborative-filtering recommender system that mines open-source software (OSS) repositories to provide developers with API FunctiOn Calls and USage patterns. It makes use of the structural Expectation-Maximization (EM) algorithm.

In another research by (Nguyen, et al., 2019), the researchers proposed CPATMINER that detects unknown repetitive changes through mining fine-grained semantic code change patterns from many repositories. It relies on fine-grained change graphs to capture program dependencies. Results show that this tool using graph-based approach, detects 2.1x more meaningful patterns in comparison with AST-based technique. They proposed a graph-based program representation that captures the control/data dependencies as in Program-Dependence Graphs (PDGs) and uses the API elements as in Object Usage Graphs (GROUMs).

(Huang, Xia, Xing, Lo, & Wang, 2018) presented BIKER (Bi-Information source-based Knowledge Recommendation), an API recommendation approach aims to bridge the lexical gap using word embedding technique to calculate the similarity score between two text descriptions. Using NLTK package, the sentence is tokenized. Based on several heuristic rules to extract API entities from each question's answers, Asymmetric similarity and IDF-weighted sum of similarities, BIKER recommends APIs at method-level.

(Liu, Huang, & Ng, 2018) presented, RecRank, which is an approach and tool that applies a novel ranking-based discriminative approach leveraging API usage path features to improve top-1 API recommendation that uses the graph-based statistical model to recommend top-10 API candidates effectively. This approach does not rely on code change history. RecRank employs a discriminative re-ranker that is trained to re-rank Gralan's top-10 candidate APIs:

A novel kind of features are proposed for use in conjunction with discriminative re-ranker, API usage path-based features. Re-ranking systems are either trained using the Naive Bayes (NB) generative model or the Support Vector Machine learner.

(Nagashima & He, 2018) comprises two main parts, naming PaMpeR's preparation phase and its recommendation phase. PaMpeR first preprocesses the database and generates a database for each proof method. Then, PaMpeR applies a regression algorithm to each database and creates a regression tree for each proof method. This regression algorithm attempts to discover combinations of features useful to recommend which proof method to apply.

Zhong et al. (2009) have developed an API usage mining tool called MAPO (Mining API usage Pattern from Open-source repositories), which automatically mines API usage patterns. MAPO clusters code snippets based on the level of similarity to each other, which shows various usages of different clusters. Also, MAPO organizes API usage patterns and recommends code snippets based on that organization (Zhong, et al., 2009). Holmes et al (2005) proposed Strathcona, which suggests code snippet by comparing of developing and stored code snippet in repository and lists relevant code snippets (Holmes et al., 2005).

CHAPTER 3

METHODOLOGY

In this section, we describe the methodology followed to address our research problem. As shown in Figure 3.1, our methodology consists of the following main steps. The first step consists of reviewing the literature. Based on the literature review, we stated the research problems. Next step consists of studying, understanding and investigating the advanced data mining techniques used in our study. This step is followed by our empirical study, which is composed of four main steps: definition, planning, implementation and interpretation of the obtained results. This phase is followed by conclusions drawn from our findings, which lead us to write our thesis to convey our message to the research community.

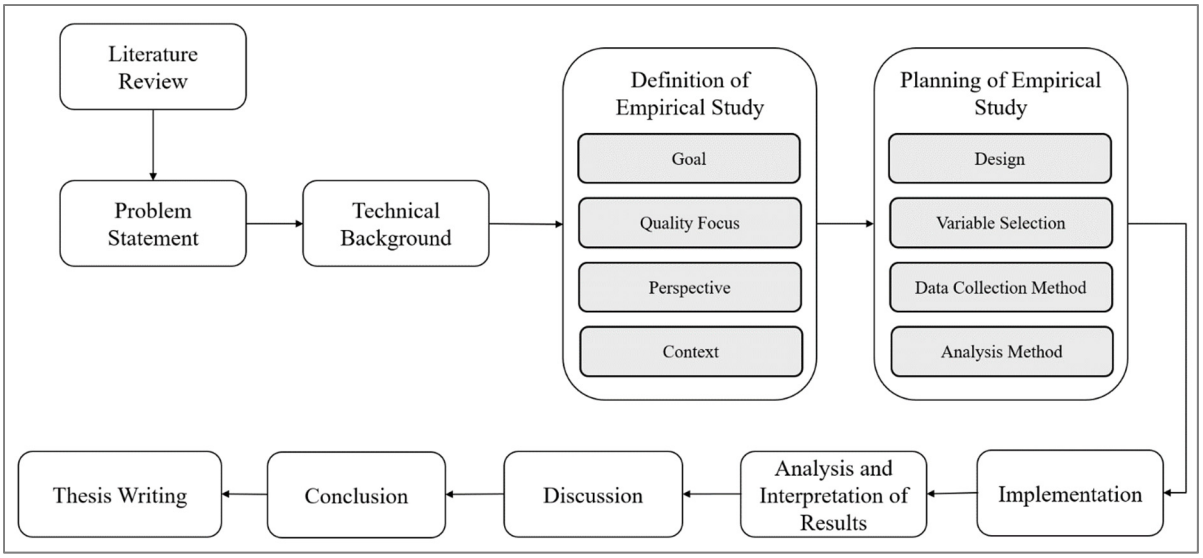

Figure 3.1 Overview of the methodology followed in our thesis

Since our approach is empirical, we illustrate first the main steps of our empirical approach as illustrated in Figure 3.2 in the following sections of this chapterwhile the other phases of the overall methodology presented in Figure 3.1 will be discussed separately in the following chapters of the thesis for organization purposes.

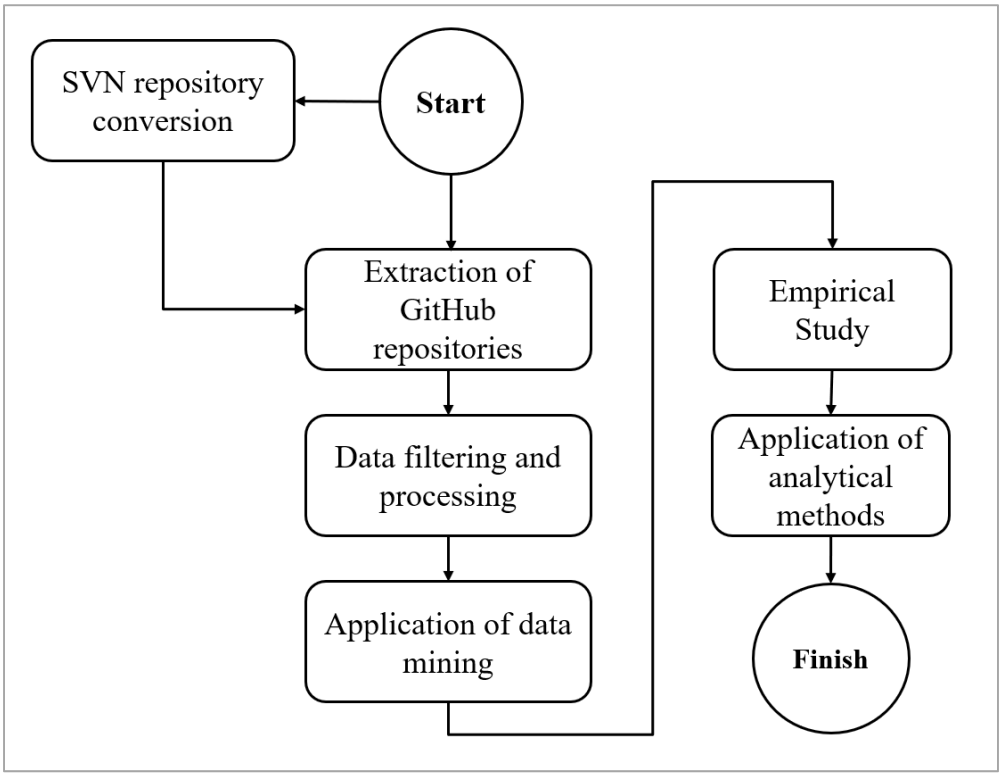

Figure 3.2 Overview of our Empirical Approach

As it can be noticed from Figure 3.2, the first step consists of extracting the source code change of each studied project and saving this information in a PostgreSQL database for data organization. The second step consists of filtering and creating the training and test data. Finally, the third step corresponds to the application of the four data mining algorithms investigated on the training data to generate recommendations of source code change files. The generated recommendations are empirically validated against the test data. After that, a comparison between the four algorithms, Apriori, FP-Growth, Eclat and Relim is performed, while considering the simple Apriori as a baseline.

3.1 Extracting source code change history

The first step of our methodology consists of generating a list of all Java source code file changes from a project repository through a change extraction program using the JGit tool.

Prior to that, if a project was only available in SVN, we had to migrate from SVN to Git using the svn2git tool. This step begins with launching a local repository under Git for the studied project. Then, the history extraction program goes through all the changes that have been made to the repository in the past and when the change concerns one or more Java files (.Java files), the change is selected by the program. Various information concerning the files are then mined, in particular, the change ID, the change log, the change date, the type of modification performed (add, modify, rename or delete), the file name for each modified file and a unique index for each file. Finally, each selected file is inserted into a PostGreSQL database with the various extracted information. We hence obtain a database, for each project, containing all the Java files that have been modified, in the form of transactions. A transaction consists of a set of files that have changed in the same commit (Zimmermann, Zeller, Weissgerber, & Diehl, 2005). Each file is identified by the same author, same date, and a message under Git.

3.2 Data filtering and processing

The second step of our methodology consists of filtering and processing the transactions retrieved from the PostGreSQL databases of each project, using a Perl script and the DBI module and generating the training and test data in two separate files. For each project from which the local repository has been extracted in PostGreSQL, a Perl script will go through all the project transactions, start by filtering them, then end by processing the transactions in order to organize them into transactions belonging to the training file and transactions belonging to the test file.

The first filter applied concerns transactions linked to only one file. Whether be it for the training or test data, transactions with only one file are not helpful to draw any predictions since the used data mining algorithms use the different files occurring in the same transactions to extract frequent sets of files. The second filter ignore transactions with more than 100 files as used in other studies such as (Hassan & Xie, 2010), (Ying, Murphy, Ng, & Chu-Carroll, 2004). These types of changes and the information they provide are not relevant to the

predictions we target since the files within these transactions are, for example, not necessarily linked to the files that have been modified to implement a new feature.

Once the filters are applied, the processing phase selects the filtered transactions in order to split them into the training and test data. To this aim, we mirrored previous (Zimmermann, Zeller, Weissgerber, & Diehl, 2005), for each studied change history of each project, we examined a number of full months containing the last 1,000 transactions, but not more than 50% of all transactions (Zimmermann, Zeller, Weissgerber, & Diehl, 2005). In this period, called in the mining jargon, the evaluation period, we verified for each transaction, whether its files can be predicted from earlier change history, from the training data.

For some recent projects, we have noticed that when selecting the last 1,000 transactions, the latter are spread over only a few months, while for older projects, the latter 1000 transactions may extend over several years. To overcome this challenge, we verified, for each project, if the last 1,000 transactions go beyond one year. If this is case, the script only selects transactions from the last year. If, however, the transactions do not exceed one year, then the script rounds the date to the full month based on the earliest date in the thousand selected and most recent transactions. As a result of this step, we create the training and test data for each examined project.

3.3 Recommending source code file changes

To explore the relationships between files in our transactional databases and generate the predictions of file changes, we have applied four different data mining algorithms, Apriori, FP-Growth, Eclat and ReLim on our training data set, previously produced in the second step of our process, using different parameters and configurations for each studied project. The recommendations generated for files have been checked against the test data.

When generating the change file recommendations, the minimum support and confidence parameters (Webb, 1989) must be considered according to the kind of algorithm used. For each project, we have empirically investigated different values of support and confidence. The

reason is that we aimed to compare the four studied algorithms in terms of their performance using different configurations.

It is important to mention that we empirically noticed during our different experiments, that the lower the selected support and confidence parameters are, the greater the number of generated association rules will be, but this is at the expense of an increase in the computation time. The same applies to the number of transactions since the computation time significantly increases when dealing with a large volume of data. Since the main purpose of producing these file change recommendations is to support and guide developers during their software development and maintenance activities, computation time must be normally as short as possible if real-time suggestions are to be made.

The results of this step consist of generating, for each studied algorithm, a list of files that change together. After generating source code change files recommendation for each project, our last task consists of verifying the relevance of the recommendations made using each of our considered algorithms.

For each recommendation, we have compared if a correspondence exists in the test data of the concerned project. From these various comparisons, statistics are generated, and the performance of our algorithms is measured in terms of precision, recall, and F-measure. Other analytical methods, presented in « Chapter 4 Empirical Evaluation », are then used to quantify these results and to represent them graphically, facilitating our discussion in Section Results and Discussion.

CHAPTER 4

EMPIRICAL EVALUATION

To address our research questions, we have performed an empirical study following the Basili framework described in Wohlin (Wohlin, et al., 2012). This framework consists of definition, planning, operation and interpretation (Basili et al., 1986) which are described in this section.

4.1 Definition and Planning of the study

The *Goal* of the study (Basili et al., 1986) is to compare the effect of different advanced data mining techniques algorithms, when recommending source code file changes.

The *Quality focus* (Basili et al., 1986) is represented by the performance of the four studied algorithms, Apriori, FP-Growth, Eclat and Relim measured in terms of precision, recall, and F-measure using different configurations.

The *Perspective* (Basili et al., 1986) concerns researchers and developers that are interested in having recommendations systems that can guide them through their software maintenance and development activities. The *Context* consists of seven open-source projects: Eclipse[1], ElasticSearch[2], Rhino[3], SWT[4], Kotlin[5], Guava[6], JabRef[7], and their corresponding source code history. The main characteristics of the projects under study are summarized in Table 3.1. As it can be noticed, this table shows the examined projects, their evaluation period, which means the time period corresponding to the considered change history, the number of transactions,

[1] https://github.com/eclipse/eclipse.jdt.core
[2] https://github.com/elastic/elasticsearch
[3] https://github.com/mozilla/rhino
[4] https://www.eclipse.org/swt
[5] https://github.com/JetBrains/kotlin
[6] https://github.com/google/guava
[7] https://github.com/JabRef/jabref

the number of files that changed together in the same commit, as well as the number of contributors to each project.

4.2 Research Questions

We address the main following research questions in this research work:

RQ1: How do the four studied advanced data mining techniques, Apriori, FP-Growth, Eclat and Relim perform in terms of Precision, Recall and F-measure?

RQ2: Are there any differences between the four studied advanced data mining techniques, Apriori, FP-Growth, Eclat and Relim in terms of their performances when recommending source code file changes?

We derived five specific research questions from the above-mentioned research questions. These specific research questions along with their corresponding null and alternative hypothesis are as follows:

RQ_{11}: How does the Apriori algorithm perform in terms of Precision, Recall and F-measure?

RQ_{12}: How does the FP-Growth algorithm perform in terms of Precision, Recall and F-measure?

RQ_{13}: How does the Eclat algorithm perform in terms of Precision, Recall and F-measure?

RQ_{14}: How does the Relim algorithm perform in terms of Precision, Recall and F-measure?

RQ_{21}: How does the Apriori perform compare to FP-Growth when recommending source code file changes?

- $H_{0\text{-}21}$: There is no statistically significant difference between Apriori and FP-Growth in terms of their performance when recommending source code file changes.
- $H_{a\text{-}21}$: There is statistically a significant difference between Apriori and FP-Growth in terms of their performance when recommending source code file changes.

RQ_{22}: How does the ReLim perform compare to FP-Growth when recommending source code file changes?

- $H_{0\text{-}22}$: There is no statistically significant difference between ReLim and FP-Growth in terms of their performance when recommending source code file changes.
- $H_{a\text{-}22}$: There is statistically a significant difference between ReLim and FP-Growth in terms of their performance when recommending source code file changes.

RQ_{23}: How does the ReLim perform compare to Apriori when recommending source code file changes?

- $H_{0\text{-}23}$: There is no statistically significant difference between ReLim and Apriori in terms of their performance when recommending source code file changes.
- $H_{a\text{-}23}$: There is statistically a significant difference between ReLim and Apriori in terms of their performance when recommending source code file changes.

RQ_{24}: How does the Apriori perform compare to Eclat when recommending source code file changes?

- $H_{0\text{-}24}$: There is no statistically significant difference between Apriori and Eclat in terms of their performance when recommending source code file changes.
- $H_{a\text{-}24}$: There is statistically a significant difference between Apriori and Eclat in terms of their performance when recommending source code file changes.

RQ_{25}: How does the ReLim perform compare to Eclat when recommending source code file changes?

- $H_{0\text{-}25}$: There is no statistically significant difference between ReLim and Eclat in terms of their performance when recommending source code file changes.
- $H_{a\text{-}25}$: There is statistically a significant difference between ReLim and Eclat in terms of their performance when recommending source code file changes.

RQ_{26}: How does the FP-Growth perform compare to Eclat when recommending source code file changes?

- $H_{0\text{-}26}$: There is no statistically significant difference between FP-Growth and Eclat in terms of their performance when recommending source code file changes.
- $H_{a\text{-}26}$: There is statistically a significant difference between FP-Growth and Eclat in terms of their performance when recommending source code file changes.

4.3 Variables selection

In this section, we present the variables selection related to our empirical investigation according to Basili Framework (Basili et al., 1986) along with the metrics used to measure the performance of our studied algorithms. The main independent variable of our study is the kind of advanced data mining algorithm being used. There are four different values for this factor: Apriori, FP-Growth, Eclat and ReLim.

The dependent variable considered in our study is the accuracy, performance of the advanced data mining algorithms measured in terms of precision, recall, and F-measure.

We compute these measures following the method used in past research (Zimmermann, Zeller, Weissgerber, & Diehl, 2005), (Ying, Murphy, Ng, & Chu-Carroll, 2004). In fact, for each project, for each transaction of the test data, we evaluated a number of queries. A query is simply a file from a transaction in the test data. Each query q has an expected result B_q that is all items, files in those transactions but not the query. Let us consider that A_q represents the rules/predictions generated by one of our data mining algorithms (Apriori, FP-Growth, Eclat or Relim) for the query q. We assess q using the precision P_q, recall R_q, and F-Measure F_q, which is the harmonic mean of precision and recall (van Rijsbergen, 1979):

$$P_q = \frac{|Aq \cap Bq|}{|Aq|}, R_q = \frac{|Aq \cap Bq|}{|Bq|}, F_q = \frac{2.Pq.Rq}{Pq+Rq}$$

To measure the overall performance of an algorithm, we computed the mean value (of performances computed for all queries as above) of the precision, recall, and F-measure triples as in (Zimmermann, Zeller, Weissgerber, & Diehl, 2005).

4.4 Analysis method

Our analysis method is based on both descriptive statistics and statistical analyses. As per descriptive statistics, we have used the boxplots, which is a simple way to present the results (McGill, Tukey, & Larsen, 1978).

For comparisons purposes, we have compared the precision, recall, and F-Measure using a non-parametric test for pair-wise median comparison, namely the Wilcoxon paired test (Wilcoxon, 1945).

We used a paired test as our samples are dependent, as we generate, for each source code file among the test data, the precision, recall and F-Measure of the predictions made by each type of studied algorithm using the training data. The Wilcoxon test reports whether the median difference between the two advanced data mining algorithms is zero. Since the Wilcoxon test has been used multiple times, p-values must be adjusted. To this aim, the Holm correction method was used (Holm, 1979). This procedure sorts the results of p-values from n tests in ascending order of values, multiplying the smallest by n, the next by n-1, and so on. Results are interpreted as statistically significant at $\alpha = 5\%$. Then, the Cliff's delta is calculated in order to measure the effect size between two different applied data mining techniques. Cliff's delta ranges from -1 to 1 and its values signify for various effectiveness levels, in which, $0.474 \leq |d|$ is considered as large, $0.33 \leq |d| < 0.474$ is considered as medium, $0.147 \leq |d| < 0.33$ is considered as small and $|d| < 0.147$ will be considered as negligible (Chen et al., 2019).

Table 4.1 Characteristics of the investigated projects

Project	Evaluation period	Records	Contributors
Eclipse	27/01/2001- 24/04/2017	22,509	56
ElasticSearch	08/02/2010 – 01/06/2017	28,078	854
Rhino	21/04/1999 – 19/12/2016	3,383	31
SWT	02/08/2012 – 26/06/2017	25,585	53
Kotlin	10/12/2010 – 31/05/2017	35,115	197
Guava	15/09/2009 – 31/05/2017	5,654	189
JabRef	16/10/2003 – 25/04/2017	16,489	175

Characteristics of the investigated projects are illustrated in Table 4.1, in which, evaluation period signifies for the date of first and last records in databases of each project. Total number of records are presented by the Records column, which is the total number of extracted records before transforming the commits data to transactions in the database corresponding to each examined project, while total number of contributors are shown in Contributor's column. Then, in Table 4.2, after filtering each project using above-mentioned procedure, total number of transactions, number of transactions after filtering, division of transactions to training and test transactions are illustrated. As can be seen, some projects, naming Eclipse, ElasticSearch and Kotlin are maintaining more than ten thousand of transactions.

Table 4.2 Number of transactions for each project

Project	**Total transactions**	**After filtering**	**Training transactions**	**Test transactions**
Eclipse	17,985	10,320	7,224	3,096
ElasticSearch	20,659	12,414	8,690	3,724
Rhino	2,840	1,406	984	422
SWT	21,588	8,260	5,782	2,478
Kotlin	25,710	15,115	10,580	4,535
Guava	3,887	2,399	1,679	720
JabRef	7,074	4,229	2,960	1,269

In this section, all seven projects mentioned above are investigated statistically in terms of Number of releases (Releases), Number of Contributors (Contributors), Number of Files (Files), Number of Commits (Commits) and Number of Branches (Branches) are illustrated in Table 4.3. Level of granularity in this study is file therefore the most important measure in order to compare projects and number of transactions will be files. According to this table, Elasticsearch and Kotlin are largest projects in terms of files and contributors.

Table 4.3 Projects statistics

Project	Releases	Contributors	Files	Commits	Branches
Eclipse	87	325	8.6k	1230	129
ElasticSearch	74	1712	22.5k	62868	163
Rhino	15	76	4.6k	4090	51
SWT	2	12	1k	466	10
Kotlin	179	532	65.6k	91069	4k
Guava	34	271	3k	5735	19
JabRef	33	403	13k	16888	20

CHAPTER 5

RESULTS AND DISCUSSION

To verify if our data follows a normal distribution or not, we have used the Shapiro-Wilk normality test (Shapiro & Wilk, 1965). According to this test, when the p-value is greater than 0.05, this implies that the distribution of the data is not significantly different from normal distribution. In other words, almost all of the performance measures of the different projects that we have examined are not following a normal distribution. We therefore apply a non-parametric statistical test, i.e., Wilcoxon paired test and non-parametric effect-size measure, namely Cliff's Delta. Table 5.1 illustrates the output of the Shapiro-Wilk normality test:

Table 5.1 Results of the Shapiro-Wilk normality test

Project	Precision	Recall	F-score
Eclipse	<0.0001	0.1103	0.02614
ElasticSearch	<0.0001	0.0009518	<0.0001
Guava	<0.0001	<0.0001	<0.0001
Jabref	<0.0001	<0.0001	<0.0001
Kotlin	0.00708	0.01017	0.001215
Rhino	0.007473	0.001503	0.1342
SWT	0.0001218	<0.0001	0.003301

5.1 Results of RQ1

RQ1: *How do the four studied advanced data mining techniques, Apriori, FP-Growth, Eclat and Relim perform in terms of Precision, Recall and F-measure?*

In this section, we present the descriptive statistics of four data mining algorithms that we have investigated, namely Apriori, FPGrowth, Eclat and ReLim, as well as their performance measured in terms of precision, recall and F-measure for each studied project. We organize the results as follows. For each project of the seven studied projects, we show the descriptive statistics of precision, recall, and F-measure, namely Min, Mean, Sd, Median and IQR. Min shows the minimum value of measure for each data mining technique. Mean reveals the average of obtained values for every algorithm. Sd illustrates the standard deviation which

shows amount of variation in each dataset. Median represents the middle number in dataset and IQR is showing the interquartile range which contains the second and third quartiles of dataset.

5.1.1 Descriptive statistics of Eclipse

Table 5.2 Descriptive statistics of Eclipse

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.433	0.452	0.018	0.454	0.034
FPGrowth	0.26	0.391	0.036	0.400	0.020
ReLim	0.349	0.398	0.017	0.402	0.007
Eclat	0.281	0.395	0.070	0.412	0.104
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.885	0.937	0.058	0.926	0.107
FPGrowth	0.453	0.654	0.091	0.639	0.119
ReLim	0.434	0.660	0.099	0.666	0.122
Eclat	0.552	0.735	0.109	0.762	0.147
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.604	0.609	0.004	0.609	0.008
FPGrowth	0.377	0.486	0.042	0.496	0.044
ReLim	0.404	0.494	0.037	0.507	0.034
Eclat	0.373	0.507	0.0624	0.503	0.106

Regarding the precision, as it can be noticed from Table 5.2, the min of precision varies from 0.26 to 0.433 and the mean of the precision varies from 0.39 to 0.45, while the sd ranges from 0.17 to 0.7. Also, median changes from 0.4 to 0.454, while IQR differs from 0.007 to 0.104. For Recall, the min of recall varies from 0.434 to 0.885 and the mean of the recall varies from 0.654 to 0.937, while the sd ranges from 0.058 to 0.109. Also, median changes from 0.639 to 0.926, while IQR differs from 0.107 to 0.147. Accordingly, for F-score, the min of F-score varies from 0.373 to 0.604 and the mean of the F-score varies from 0.486 to 0.609, while the sd ranges from 0.004 to 0.0624. Also, median changes from 0.496 to 0.609, while IQR differs from 0.008 to 0.106.

5.1.2 Descriptive statistics of ElasticSearch

Table 5.3 Descriptive statistics of ElasticSearch

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.5	0.5	0	0.5	0
FPGrowth	0.097	0.366	0.122	0.400	0.0582
ReLim	0.097	0.388	0.106	0.413	0.0620
Eclat	0.097	0.404	0.126	0.439	0.0952
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.75	0.958	0.102	1	0
FPGrowth	0.394	0.618	0.159	0.689	0.290
ReLim	0.394	0.628	0.147	0.689	0.178
Eclat	0.41	0.638	0.154	0.712	0.278
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.6	0.656	0.0272	0.667	0
FPGrowth	0.173	0.428	0.119	0.471	0.0992
ReLim	0.173	0.457	0.108	0.488	0.0851
Eclat	0.173	0.466	0.116	0.503	0.129

Regarding the precision, as it can be noticed from

Table 5.3, the min of precision varies from 0.097 to 0.5 and the mean of the precision varies from 0.366 to 0.5, while the sd ranges from 0 to 0.126. Also, median changes from 0.4 to 0.5, while IQR differs from 0 to 0.0952. For Recall, the min of recall ranges from 0.394 to 0.75 and the mean of the recall varies from 0.618 to 0.958, while the sd ranges from 0.102 to 0.159. Also, median changes from 0.689 to 1, while IQR differs from 0 to 0.29. Accordingly, for F-score, the min of Fscore ranges from 0.173 to 0.6 and the mean of the F-score varies from 0.428 to 0.656, while the sd ranges from 0.0272 to 0.119. Also, median changes from 0.471 to 0.667, while IQR differs from 0 to 0.129.

5.1.3 Descriptive statistics of Guava

Table 5.4 Descriptive statistics of Guava

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.378	0.397	0.0141	0.4	0.0216
FPGrowth	0.111	0.375	0.150	0.4	0.163
ReLim	0.111	0.359	0.164	0.4	0.310
Eclat	0.12	0.384	0.152	0.41	0.167
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.7	0.838	0.117	0.875	0.131
FPGrowth	0.333	0.797	0.264	1	0.344
ReLim	0.333	0.760	0.280	0.875	0.466
Eclat	0.34	0.804	0.259	1	0.330
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.519	0.536	0.0163	0.539	0.0281
FPGrowth	0.167	0.508	0.193	0.547	0.203
ReLim	0.167	0.484	0.209	0.522	0.387
Eclat	0.177	0.518	0.192	0.560	0.207

Regarding the precision, as it can be noticed from Table 5.4, the min of precision ranges from 0.111 to 0.378 and the mean of the precision varies from 0.359 to 0.397, while the sd ranges from 0.0141 to 0.164. Also, median changes from 0.4 to 0.41, while IQR differs from 0.0216 to 0.31. For Recall, the min of recall ranges from 0.333 to 0.7 and the mean of the recall varies from 0.76 to 0.838, while the sd ranges from 0.117 to 0.28. Also, median changes from 0.875 to 1, while IQR differs from 0.131 to 0.466. Accordingly, for F-score, the imn of Fscore ranges from 0.167 to 0.519 and the mean of the F-score varies from 0.484 to 0.536, while the sd ranges from 0.0163 to 0.209. Also, median changes from 0.539 to 0.56, while IQR differs from 0.0281 to 0.387.

5.1.4 Descriptive statistics of Jabref

Table 5.5 Descriptive statistics of Jabref

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.442	0.457	0.0206	0.444	0.0276
FPGrowth	0.152	0.372	0.0852	0.402	0.116
ReLim	0.242	0.394	0.0765	0.431	0.104
Eclat	0.402	0.417	0.0206	0.404	0.0276
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.692	0.745	0.0607	0.722	0.101
FPGrowth	0.2	0.646	0.180	0.691	0.183
ReLim	0.2	0.633	0.190	0.676	0.157
Eclat	0.65	0.705	0.0609	0.685	0.105
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.541	0.565	0.0184	0.575	0.0279
FPGrowth	0.25	0.458	0.108	0.488	0.132
ReLim	0.239	0.477	0.110	0.525	0.139
Eclat	0.499	0.522	0.0188	0.531	0.0281

Regarding the precision, as it can be noticed from Table 5.5, the min of precision ranges from 0.152 to 0.442 and the mean of the precision varies from 0.372 to 0.457, while the sd ranges from 0.0206 to 0.0852. Also, median changes from 0.402 to 0.444, while IQR differs from 0.0276 to 0.116. For Recall, the min of recall ranges from 0.2 to 0.692 and the mean of the recall varies from 0.633 to 0.745, while the sd ranges from 0.0607 to 0.19. Also, median changes from 0.676 to 0.722, while IQR differs from 0.101 to 0.183. Accordingly, for F-score, the min of Fscore ranges from 0.239 to 0.541 and the mean of the F-score varies from 0.458 to 0.565, while the sd ranges from 0.0184 to 0.11. Also, median changes from 0.488 to 0.575, while IQR differs from 0.0279 to 0.139.

5.1.5 Descriptive statistics of Kotlin

Table 5.6 Descriptive statistics of Kotlin

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.435	0.453	0.0247	0.444	0.0177
FPGrowth	0.085	0.247	0.110	0.281	0.184
ReLim	0.0776	0.264	0.103	0.302	0.117
Eclat	0.087	0.244	0.0938	0.273	0.128
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.933	0.989	0.0272	1	0
FPGrowth	0.722	0.838	0.0528	0.838	0.0646
ReLim	0.722	0.842	0.0416	0.842	0.0646
Eclat	0.722	0.831	0.0483	0.829	0.0641
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.606	0.620	0.0172	0.615	0.0169
FPGrowth	0.154	0.368	0.137	0.427	0.241
ReLim	0.141	0.391	0.127	0.449	0.143
Eclat	0.156	0.367	0.117	0.413	0.154

Regarding the precision, as it can be noticed from Table 5.6, the min of precision ranges from 0.0776 to 0.435 and the mean of the precision varies from 0.244 to 0.453, while the sd ranges from 0.0247 to 0.11. Also, median changes from 0.273 to 0.444, while IQR differs from 0.0177 to 0.184. For Recall, the min of recall ranges from 0.772 to 0.933 and the mean of the recall varies from 0.831 to 0.989, while the sd ranges from 0.0272 to 0.0528. Also, median changes from 0.829 to 1, while IQR differs from 0 to 0.0646. Accordingly, for F-score, the min of Fscore ranges from0.141 to 0.606 and the mean of the F-score varies from 0.367 to 0.62, while the sd ranges from 0.0172 to 0.137. Also, median changes from 0.413 to 0.615, while IQR differs from 0.0169 to 0.241.

5.1.6 Descriptive statistics of Rhino

Table 5.7 Descriptive statistics of Rhino

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.417	0.427	0.0128	0.420	0.0190
FPGrowth	0.25	0.416	0.0701	0.415	0.0799
ReLim	0.308	0.426	0.0560	0.413	0.0842
Eclat	0.43	0.437	0.00404	0.438	0.00483
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.491	0.643	0.126	0.633	0.197
FPGrowth	0.4	0.702	0.180	0.768	0.281
ReLim	0.4	0.706	0.173	0.75	0.201
Eclat	0.52	0.679	0.134	0.665	0.201
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.451	0.510	0.0490	0.504	0.0751
FPGrowth	0.364	0.508	0.0712	0.523	0.0984
ReLim	0.364	0.519	0.0640	0.516	0.0624
Eclat	0.482	0.543	0.0438	0.537	0.0567

Regarding the precision, as it can be noticed from Table 5.7, the min of precision ranges from 0.25 to 0.43 and the mean of the precision varies from 0.416 to 0.437, while the sd ranges from 0.00404 to 0.0701. Also, median changes from 0.42 to 0.438, while IQR differs from 0.00483 to 0.0842. For Recall, the min of recall ranges from 0.4 to 0.52 and the mean of the recall varies from 0.643 to 0.706, while the sd ranges from 0.126 to 0.18. Also, median changes from 0.633 to 0.768, while IQR differs from 0.197 to 0.281. Accordingly, for F-score, the min of Fscore ranges from 0.364 to 0.482 and the mean of the F-score varies from 0.508 to 0.543, while the sd ranges from 0.0438 to 0.0712. Also, median changes from 0.504 to 0.537, while IQR differs from 0.0567 to 0.0984.

5.1.7 Descriptive statistics of SWT

Table 5.8 Descriptive statistics of SWT

Precision					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.467	0.474	0.00394	0.474	0.000892
FPGrowth	0.069	0.265	0.130	0.247	0.226
ReLim	0.069	0.284	0.173	0.248	0.253
Eclat	0.0707	0.267	0.133	0.247	0.258
Recall					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.824	0.858	0.0338	0.848	0.0530
FPGrowth	0.583	0.757	0.0888	0.774	0.159
ReLim	0.656	0.795	0.0699	0.814	0.0994
Eclat	0.529	0.735	0.108	0.758	0.171
F-score					
Algorithms	**Min**	**Mean**	**Sd**	**Median**	**IQR**
Apriori	0.596	0.610	0.00941	0.609	0.0102
FPGrowth	0.127	0.372	0.141	0.372	0.241
ReLim	0.13	0.398	0.151	0.374	0.252
Eclat	0.127	0.366	0.139	0.371	0.231

Regarding the precision, as it can be noticed from Table 5.8, the min of precision ranges from 0.069 to 0.467 and the mean of the precision varies from 0.265 to 0.474, while the sd ranges from 0.00394 to 0.173. Also, median changes from 0.247 to 0.474, while IQR differs from 0.000892 to 0.258. For Recall, the min of recall ranges from 0.529 to 0.824 and the mean of the recall varies from 0.735 to 0.858, while the sd ranges from 0.0338 to 0.108. Also, median changes from 0.758 to 0.848, while IQR differs from 0.053 to 0.171. Accordingly, for F-score, the min of Fscore ranges from 0.127 to 0.596 and the mean of the F-score varies from 0.372 to 0.61, while the sd ranges from 0.00941 to 0.151. Also, median changes from 0.371 to 0.609, while IQR differs from 0.0102 to 0.252.

According to the descriptive statistics illustrated in Table 5.2 to Table 5.8, we can notice that Apriori is performing better than the other investigated data mining techniques in terms of precision, recall and f-score for Eclipse, Elasticsearch, Jabref, Kotlin and SWT, however, for Rhino and Guava, there is no considerable difference between four investigated data mining techniques. While these trends are based on descriptive statistics for each project, more statistical tests are needed to examine the differences in terms of performance for the four

investigated data mining techniques. In the following, we present, for each project, the results of comparisons between the four data mining techniques measured in terms of precision, recall, and F-measure.

5.2 Results of RQ2

RQ2: *Are there any differences between the four studied advanced data mining techniques, Apriori, FP-Growth, Eclat and Relim in terms of their performances when recommending source code file changes?*

In this section, we use boxplots in order to display the distribution of data in a standard and visual way. Every boxplot represents five values, naming minimum, first quartile, median, third quartile, and maximum, as well as outliers, if exist. Additionally, we present all the adjusted p-values and Cliff's Delta values for performance measures, namely the Precision, Recall and F-score for all the studied projects.

5.2.1 Eclipse

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for Eclipse?

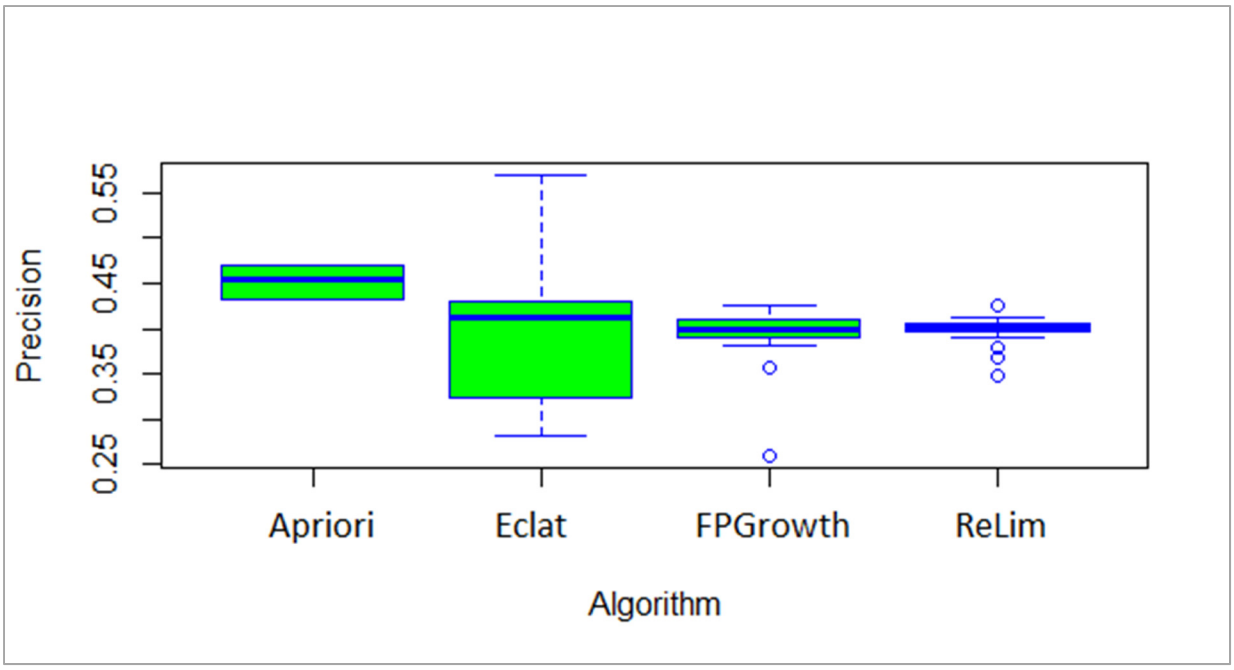

Figure 5.1 Boxplots of precision for the Eclipse project

The boxplots in Figure 5.1 show that Apriori is producing better results in terms of precision, compared to other three data mining techniques. While, Eclat, FPGrowth and ReLim are almost similar and they are only slightly different.

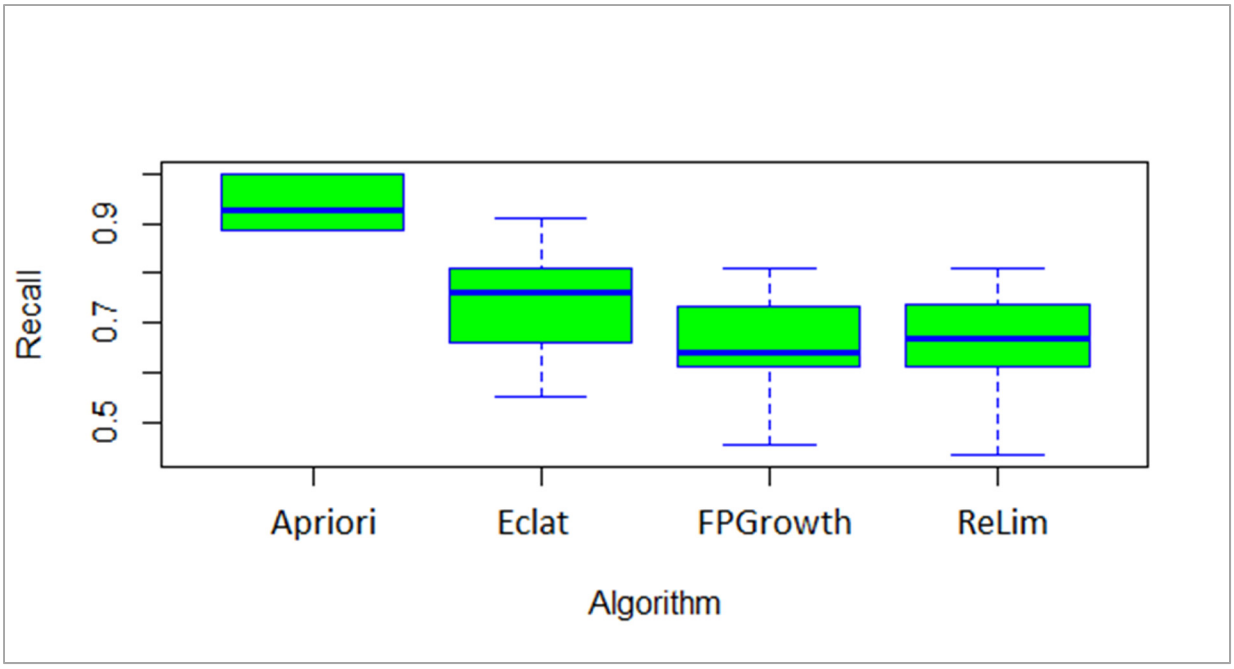

Figure 5.2 Boxplots of recall for the Eclipse project

The boxplots in Figure 5.2 show that Apriori is producing better results in terms of recall. Additionally, it can be noticed that Eclat is performing better than ReLim and FPGrowth, while there is no considerable difference between ReLim and FPGrowth.

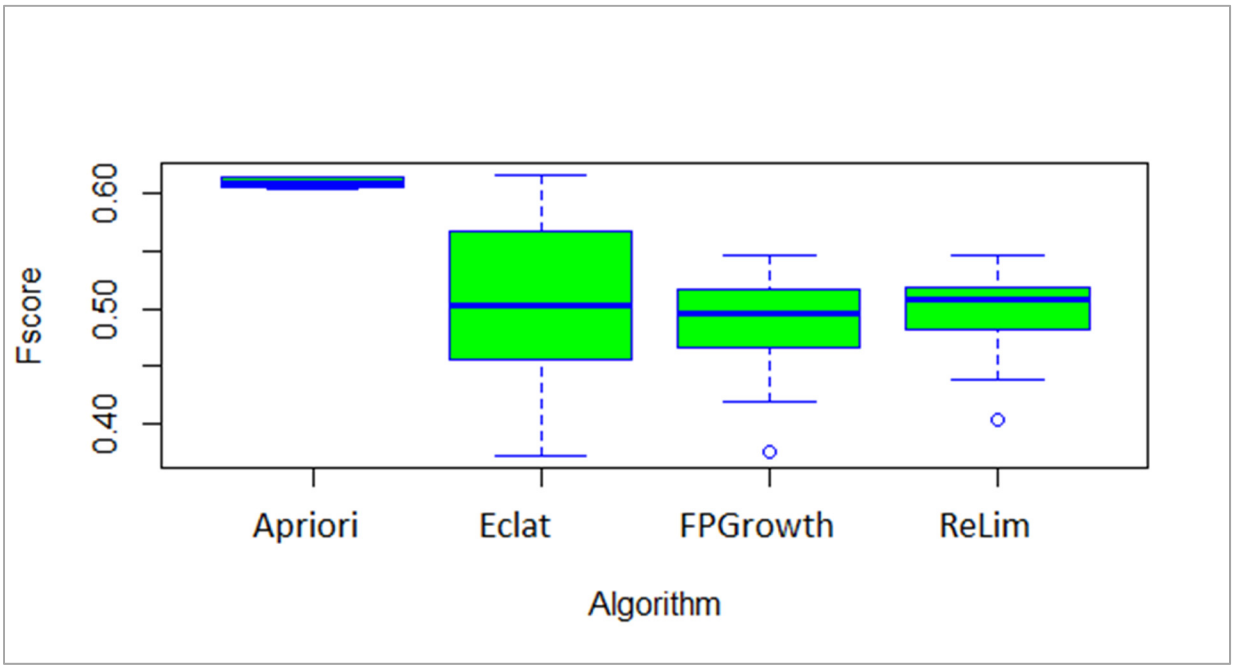

Figure 5.3 Boxplots of F-measure for the Eclipse project

The boxplots in Figure 5.3 show that Apriori is producing better results in terms of Fscore, compared to other three data mining techniques. While, Eclat, FPGrowth and ReLim are almost similar and they are only slightly different.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.9 P-values and Cliff's Delta for paired algorithms for Eclipse

Algorithms	Precision		Recall		F-score	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	**0.031**	**1 (large)**	**0.031**	**1 (large)**	**0.031**	**1 (large)**
ReLim - FPGrowth	0.62	0.04 (negligible)	0.95	0.059(negligible)	0.95	0.12 (negligible)
ReLim - Apriori	**0.031**	**-1 (large)**	**0.031**	**-1 (large)**	**0.031**	**-1 (large)**
Apriori - Eclat	0.43	0.79 (large)	**0.031**	**0.95 (large)**	0.06	0.89 (large)
ReLim - Eclat	0.74	-0.2 (small)	**0.0007766**	**-0.38 (medium)**	0.52	-0.15 (small)
FPGrowth - Eclat	0.92	-0.23(small)	**0.0007482**	**-0.41 (medium)**	0.14	-0.18 (small)

For the Eclipse project, regarding the precision, as it can be noticed from Table 5.9 that presents the p-values and Cliff's Delta values for each pair of comparison, there are statistically significantly differences between Apriori and FPGrowth (p-value = 0.031 and Cliff's delta is large) as well as statistically significantly differences between ReLim and Apriori (p-value = 0.031 and Cliff's delta is large).

For what concerns the recall, there are statistically significantly differences between Apriori and FPGrowth as well as ReLim and Apriori (p-value =0.03125 and Cliff's delta is large). Also, Table 5.9 shows statistically significantly differences between Apriori and Eclat (p-value = 0.031 and Cliff's delta is large) as well as ReLim and Eclat (p-value =0.0007766 and Cliff's delta is medium). Besides, there are statistically significantly differences between FPGrowth and Eclat (p-value =0.0007482 and Cliff's delta is medium).

Regarding the F-score, as it can be noticed from Table 5.9, there are statistically significantly differences between Apriori and FPGrowth as well as ReLim and Apriori (p-value = 0.031 and Cliff's delta is large).

For Eclipse, we can conclude that there are statistically significant differences between Apriori and FP-Growth as well as Apriori and ReLim for precision, recall, and F-measure with a large effect-size measure. Additionally, Eclat is performing better than ReLim and FPGrowth in terms of recall only with medium effect-size measure when recommending source code change files. Overall, we reject the null hypothesis $H_{0\text{-}21}$ and $H_{0\text{-}23}$ for the comparison between Apriori and FP-Growth as well as Relim and Apriori.

5.2.2 ElasticSearch

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for Elasticsearch?

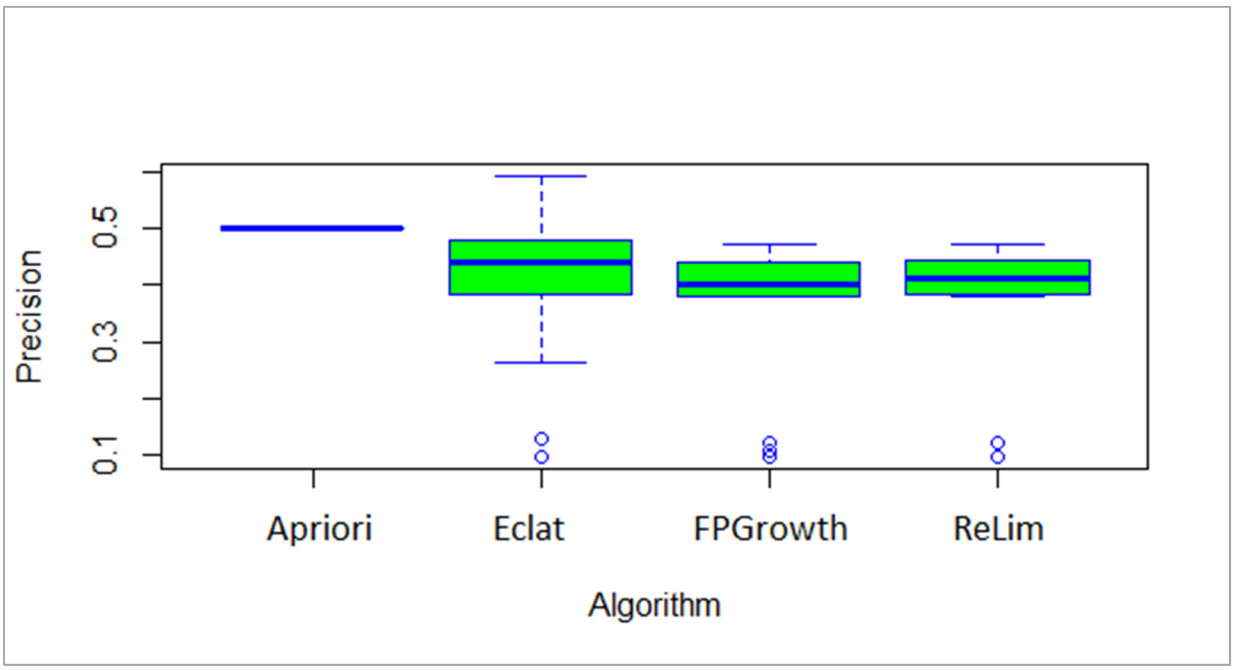

Figure 5.4 Boxplots of precision for the ElasticSearch project

The boxplots in Figure 5.4 show that Apriori is producing better results in terms of precision, compared to other three data mining techniques. Additionally, it can be noticed that both Eclat and ReLim are performing better than FPGrowth.

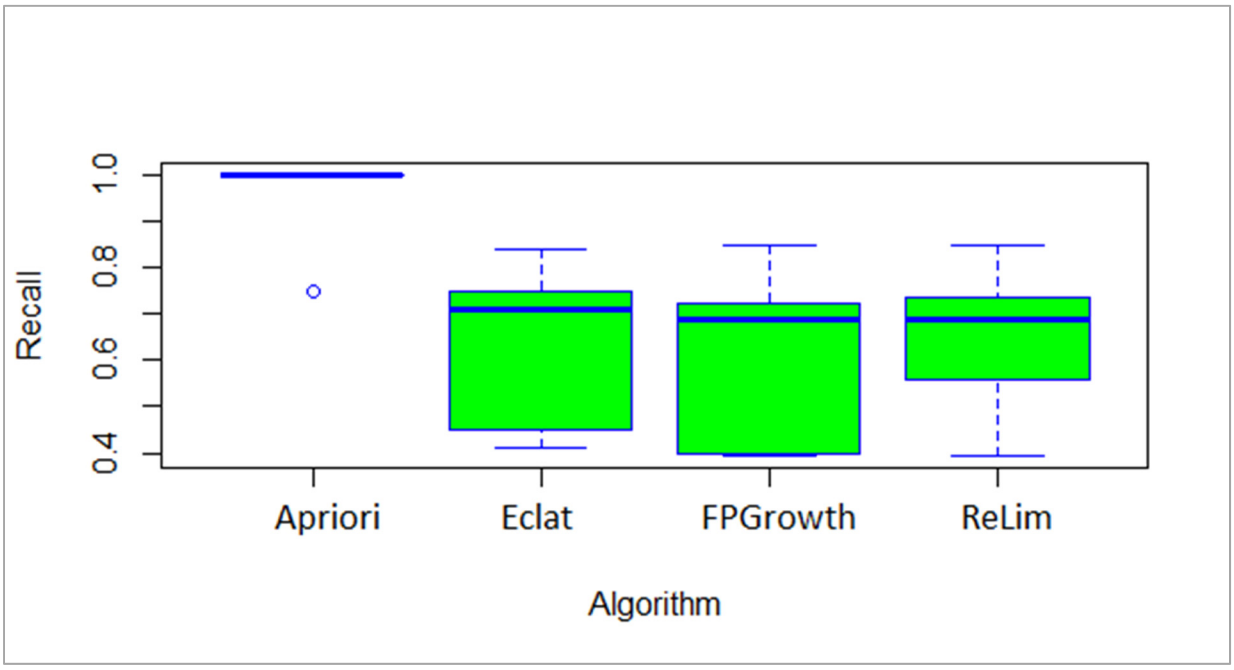

Figure 5.5 Boxplots of recall for the ElasticSearch project

The boxplots in Figure 5.5 show that Apriori is producing better results in terms of recall, compared to other three data mining techniques. Additionally, it can be noticed that there is no considerable difference between the performance of Eclat, FPGrowth and ReLim.

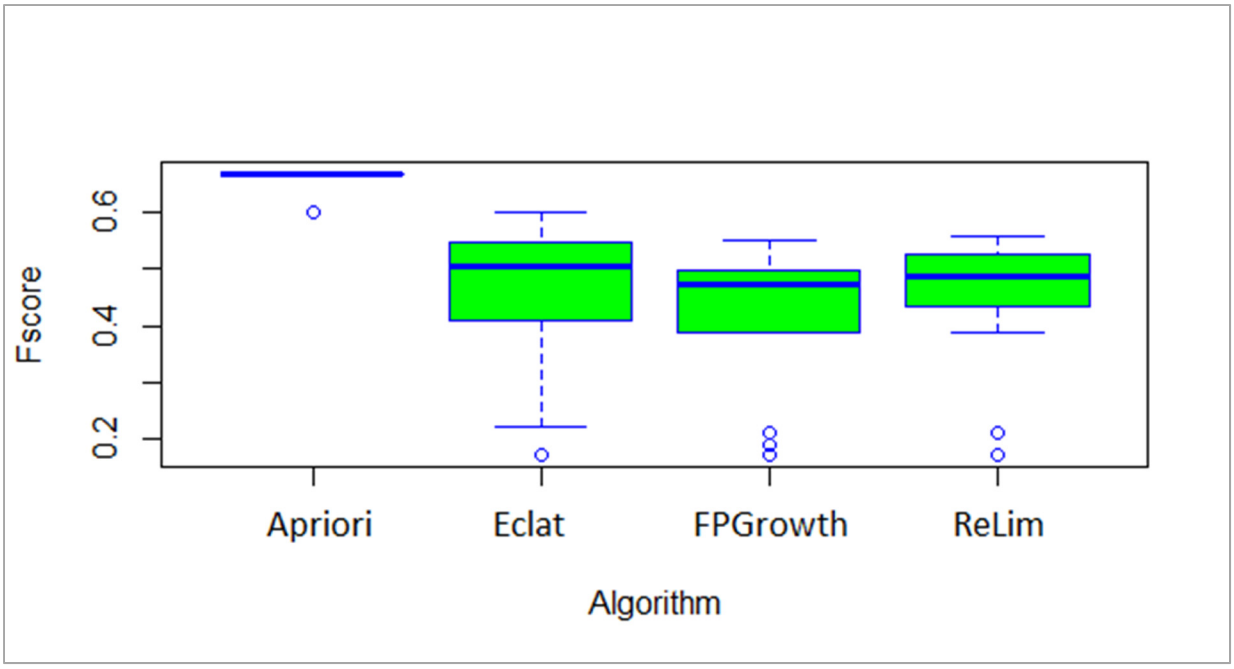

Figure 5.6 Boxplots of F-measure for the ElasticSearch project

The boxplots in Figure 5.6 show that Apriori is producing better results in terms of Fscore, compared to other three data mining techniques. Additionally, it can be noticed that both Eclat and ReLim are performing better than FPGrowth.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.10 P-values and Cliff's delta for paired algorithms for Elasticsearch

Algorithms	**Precision**		**Recall**		**F-score**	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	**0.031**	**1 (large)**	**0.031**	**0.95 (large)**	**0.031**	**1 (large)**
ReLim - FPGrowth	**0.037**	**0.17 (small)**	0.1	0.083 (negligible)	**0.03**	0.18 (small)
ReLim - Apriori	**0.031**	**-1 (large)**	**0.031**	**-0.96 (large)**	**0.031**	**-1 (large)**
Apriori - Eclat	**0.031**	**0.89(large)**	**0.031**	**0.93(large)**	**0.031**	**1 (large)**
ReLim - Eclat	0.21	-0.12 (negligible)	0.27	-0.12 (negligible)	0.06	-0.12 (negligible)
FPGrowth - Eclat	**0.009**	**-0.3 (small)**	**0.0076**	-0.18 (small)	**0.00058**	-0.26 (small)

For the Elasticsearch project: Regarding the precision: As it can be noticed from Table 5.10 that presents the p-values and Cliff's Delta values for each pair of comparison, there are statistically significantly differences between Apriori and FPGrowth (p-value =0.031 and Cliff's delta is large) as well as ReLim and FPGrowth (p-value =0.037 and Cliff's delta is small). Additionally, there are statistically significantly differences between ReLim and Apriori (p-value =0.031 and Cliff's delta is large) as well as Apriori and Eclat (p-value =0.031 and Cliff's delta is large). Furthermore, we obtained statistically significantly differences between FPGrowth and Eclat (p-value =0.009 and Cliff's delta is small).

For what concerns the recall, there are statistically significantly differences between Apriori and FPGrowth as well as ReLim and Apriori (p-value =0.031 and Cliff's delta is large). In addition, there are statistically significantly differences between Apriori and Eclat (p-value =0.031 and Cliff's delta is large) as well as FPGrowth and Eclat (p-value =0.0076 and Cliff's delta is small).

Regarding the F-score, Table 5.10 indicates that there are statistically significantly differences between Apriori and FPGrowth as well as ReLim and Apriori (p-value =0.031 and Cliff's delta is large) along with ReLim and FPGrowth (p-value =0.03 and Cliff's delta is small). Table 1-10 also shows that there statistically significantly differences between Apriori and Eclat (p-value =0.031 and Cliff's delta is large) as well as FPGrowth and Eclat (p-value =0. 00058 and Cliff's delta is small).

For Elasticsearch, we conclude that there are statistically significant differences for almost all instances of comparisons between algorithms for the triple precision, recall, and F-score with an effect-size ranging from small to large, apart from Relim and Eclat, for which no statistically significant differences have been obtained between these two data mining algorithms. We therefore reject the null hypothesis, for this project, for all instances of comparisons except for ReLim – Eclat.

5.2.3 Guava

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for Guava?

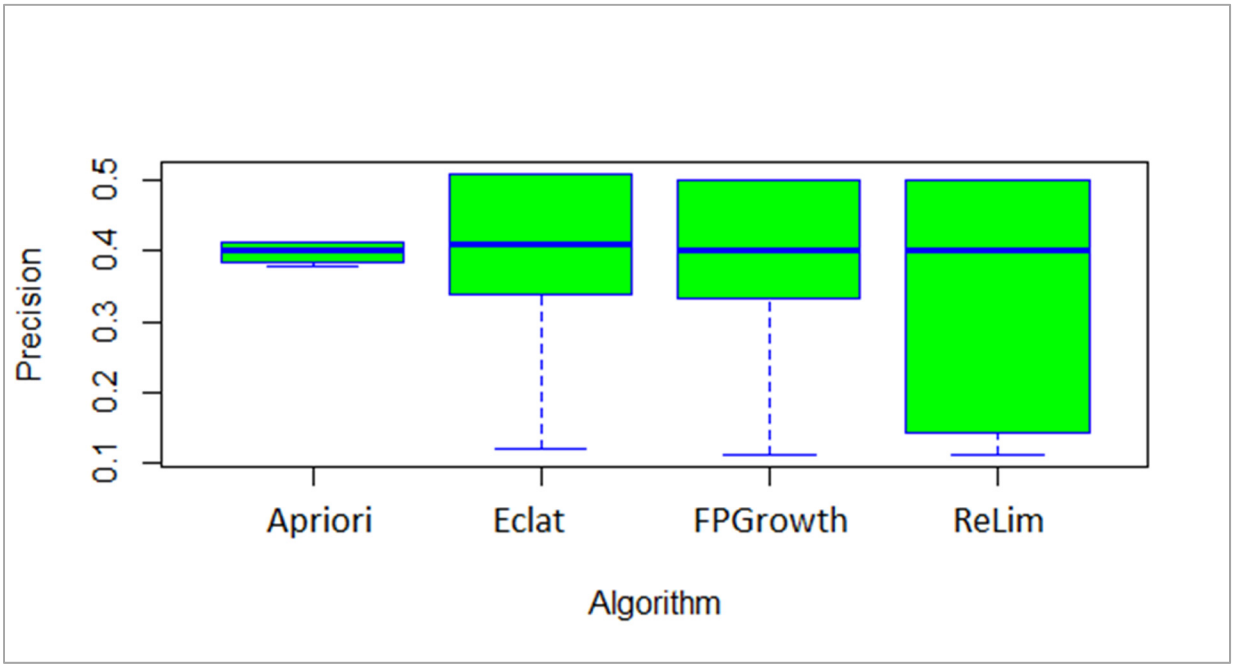

Figure 5.7 Boxplots of precision for the Guava project

The boxplots in Figure 5.7 show that there are no considerable differences between performance of four data mining techniques in terms of precision.

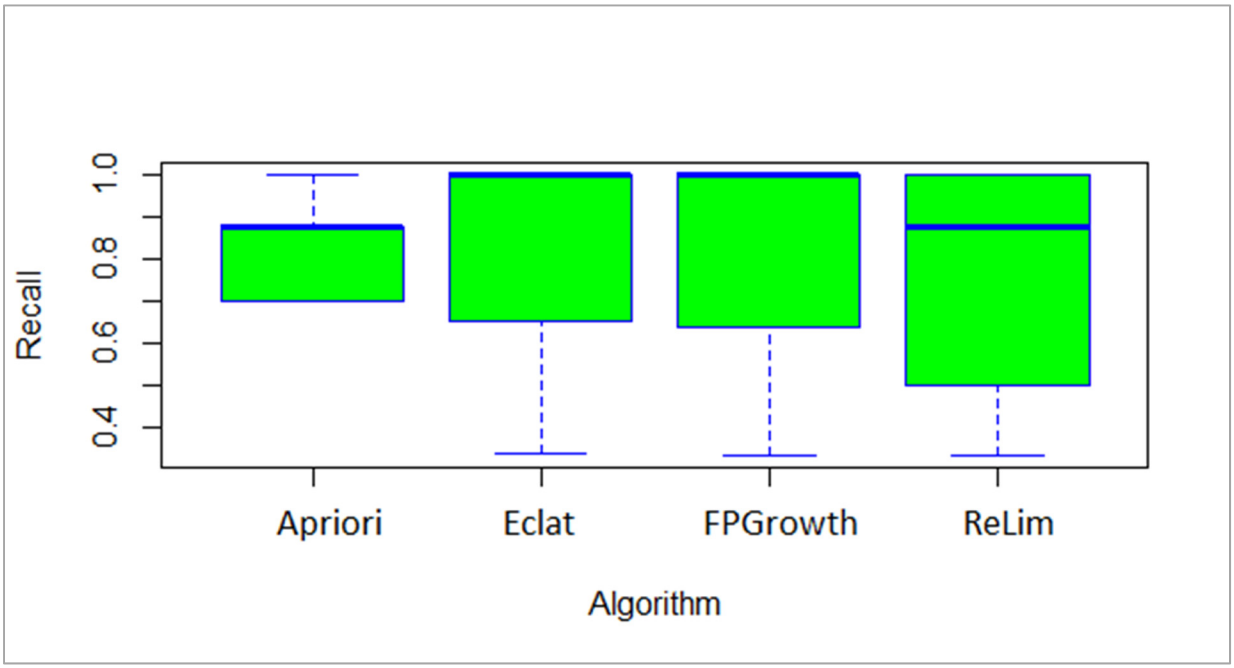

Figure 5.8 Boxplots of recall for the Guava project

The boxplots in Figure 5.8 show that Eclat and FPGrowth are performing better than ReLim and Apriori in terms of recall.

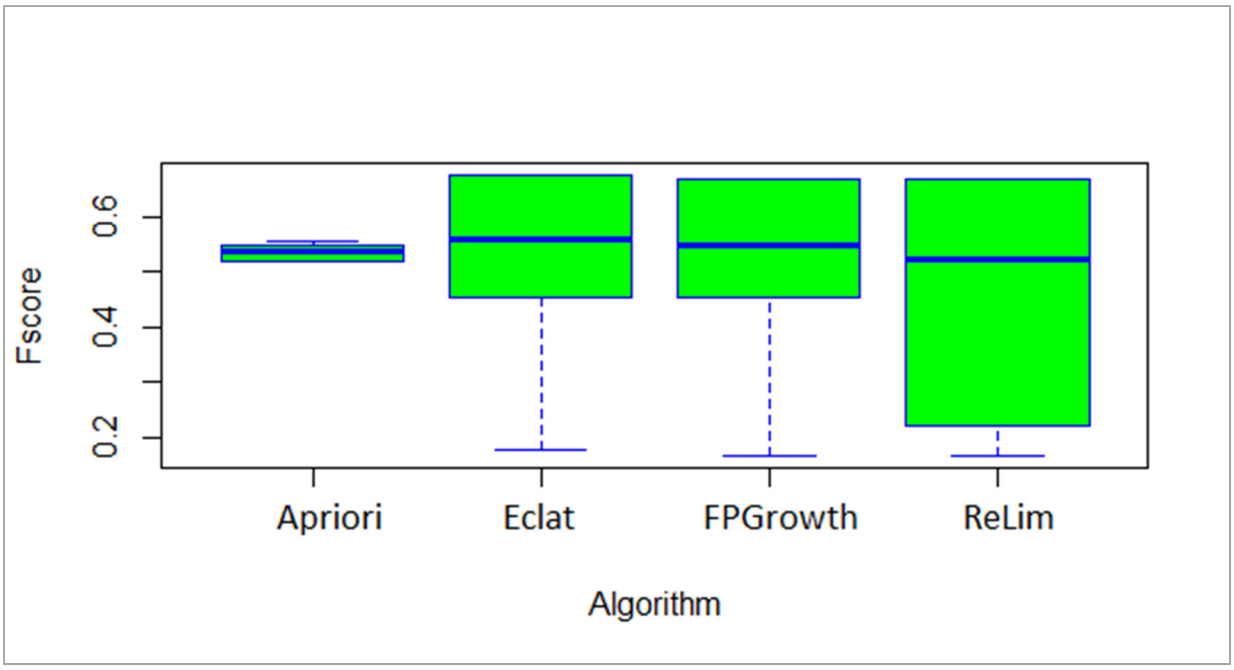

Figure 5.9 Boxplots of F-measure for the Guava project

The boxplots in Figure 5.9 show that there are no considerable differences between performance of four data mining techniques in terms of Fscore.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.11 P-values and Cliff's delta for paired algorithms for Guava

Algorithms	**Precision**		**Recall**		**F-score**	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	0.29	-0.09 (negligible)	**0.035**	**-0.13 (negligible)**	**0.035**	**-0.07 (negligible)**
ReLim - FPGrowth	0.37	-0.056 (negligible)	1	-0.07 (negligible)	0.37	-0.06 (negligible)
ReLim - Apriori	0.29	0.037 (negligible)	**0.035**	**0.028 (negligible)**	**0.035**	**-0.037 (negligible)**
Apriori - Eclat	0.28	-0.17 (small)	**0.035**	**-0.13 (negligible)**	**0.058**	**-0.11 (negligible)**
ReLim - Eclat	**0.00027**	**-0.31 (small)**	**0.009**	**-0.13 (negligible)**	**0.00018**	**-0.31 (small)**
FPGrowth - Eclat	**0.00027**	**-0.26 (small)**	**0.01**	**-0.05 (negligible)**	**0.00018**	**-0.25 (small)**

For the Guava project, regarding the precision: As it can be noticed from Table 5.11, there are statistically significantly differences between FPGrowth and Eclat (p-value =0.00027 and Cliff's delta is small) as well as statistically significantly differences between ReLim and Eclat (p-value =0.00027 and Cliff's delta is small).

For what concerns the recall, Table 5.11 indicates that there are statistically significantly differences between Apriori and FPGrowth as well as ReLim and Apriori (p-value =0.035 with a negligible Cliff's delta). In addition, there are statistically significantly differences between Apriori and Eclat, ReLim – Eclat, as well as FPGrowth – Eclat with a negligible Cliff's delta.

Regarding the F-score, Table 5.11 shows that there are statistically significantly differences between Apriori and FPGrowth (p-value =0.035), ReLim and Apriori (p-value = 0.035), as well Apriori and Eclat (p-value =0.058) but with a negligible Cliff's delta. Additionally, there are statistically significantly differences between ReLim and Eclat (p-value = 0.00018) as well as FPGrowth – Eclat (p-value = 0.00018).

Overall, for Guava, we can conclude that there are statistically significant differences between ReLim and Eclat as well as FPGrowth and Eclat for precision, recall, and F-score but with a negligible to small effect-size measure. Other instances of comparisons are statistically significant differences for the recall and F-score parts but the effect-size measure is negligible.

5.2.4 Jabref

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for Jabref?

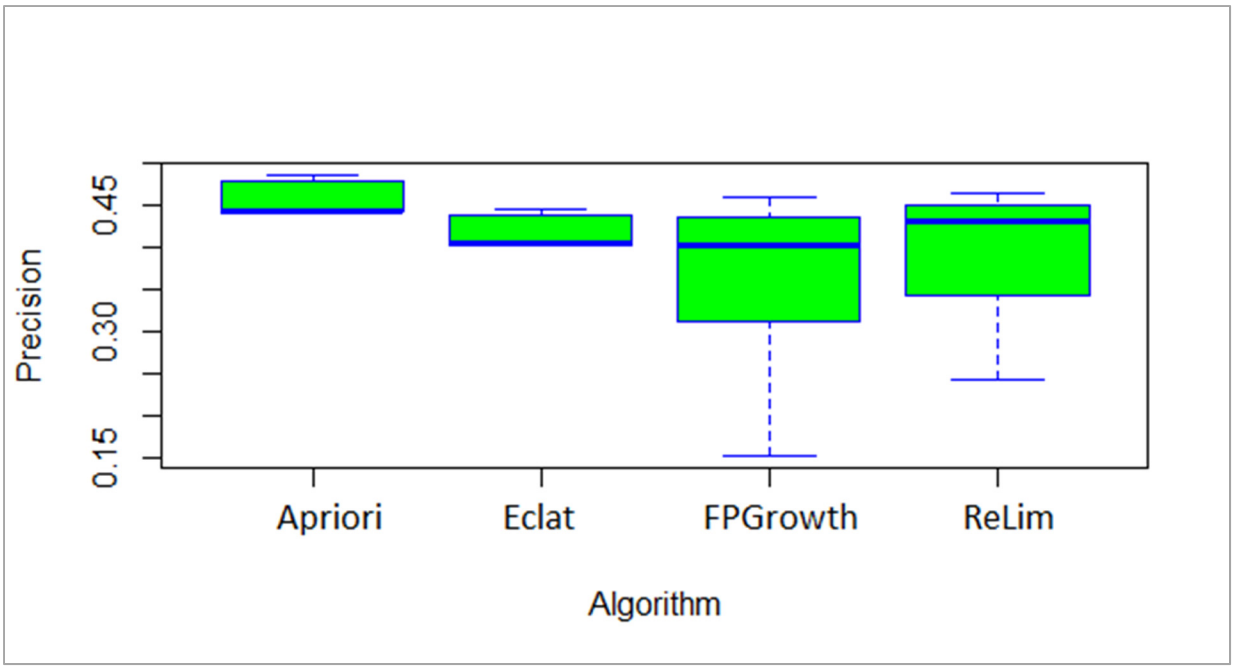

Figure 5.10 Boxplots of precision for the Jabref project

The boxplots in Figure 5.10 show that Apriori is producing better results than other data mining techniques in terms of precision.

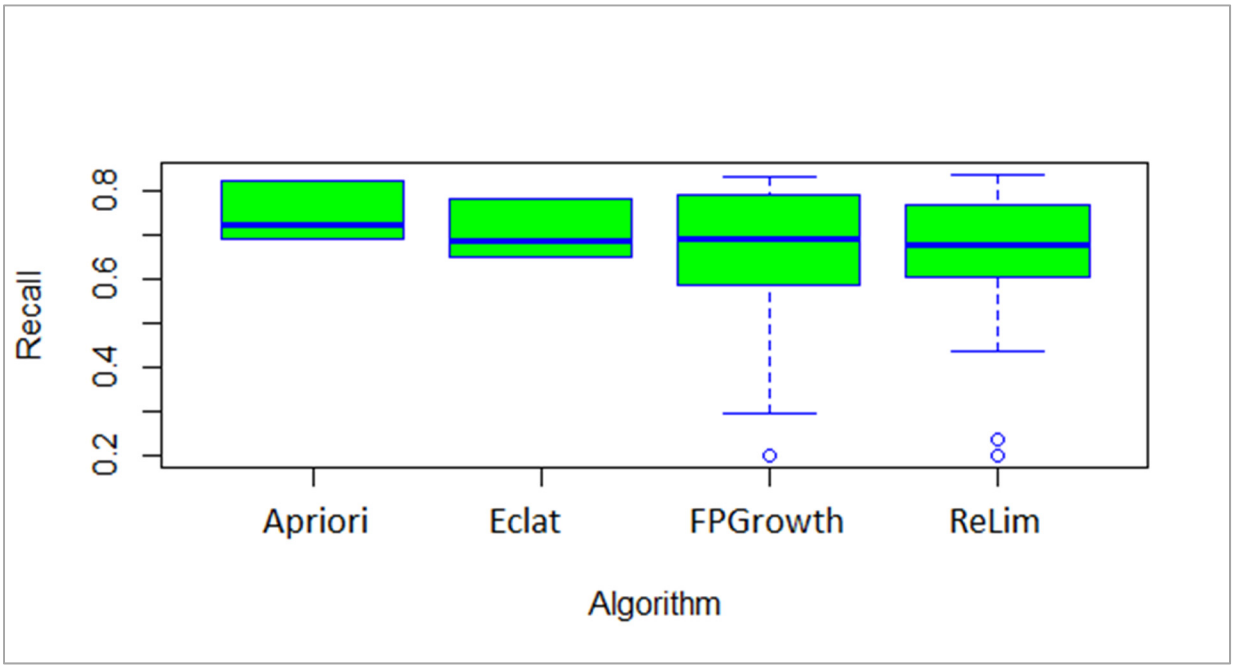

Figure 5.11 Boxplots of recall for the Jabref project

The boxplots in Figure 5.11 show that Apriori is producing better results than other data mining techniques in terms of recall.

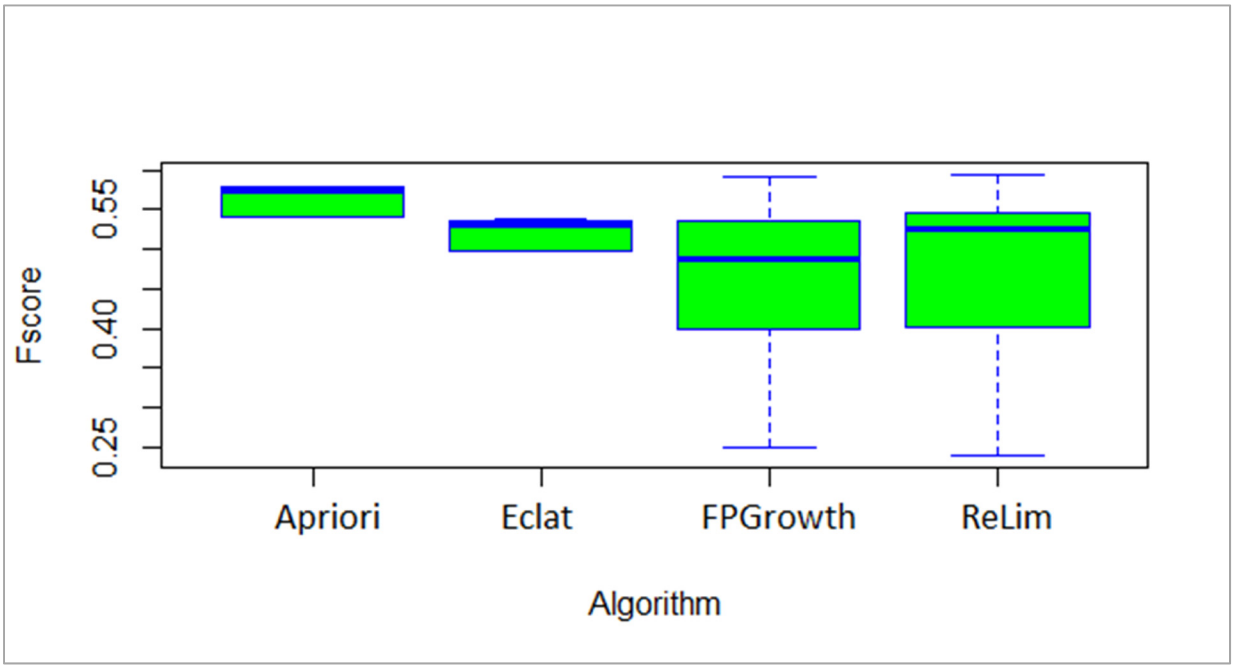

Figure 5.12 Boxplots of F-measure for the Jabref project

The boxplots in Figure 5.12 show that Apriori is producing better results than other data mining techniques in terms of Fscore.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.12 P-values and Cliff's delta for paired algorithms for Jabref

Algorithms	**Precision**		**Recall**		**F-score**	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	0.16	0.7 (large)	0.29	0.31 (small)	0.29	0.74 (large)
ReLim - FPGrowth	0.15	0.2(small)	0.13	0.09(negligible)	0.127	0.16 (small)
ReLim - Apriori	0.15	-0.48 (large)	**0.031**	**-0.28 (small)**	**0.031**	**-0.59 (large)**
Apriori - Eclat	**0.035**	**0.78(large)**	**0.035**	**0.44 (medium)**	**0.035**	**1 (large)**
ReLim - Eclat	0.84	0.17(small)	**0.031**	**0.06(negligible)**	0.31	0 (negligible)
FPGrowth - Eclat	0.84	-0.18 (small)	**0.031**	**-0.06 (negligible)**	0.22	-0.08 (negligible)

For the Jabref project: regarding the precision, as it can be noticed from Table 5.12, there are statistically significantly differences between Apriori and Eclat (p-value =0.035 and Cliff's delta is large).

For what concerns the recall, Table 5.12 indicates that there are differences between ReLim and Apriori (p-value =0.031 and Cliff's delta is small) as well as statistically significantly differences between Apriori and Eclat (p-value =0.035 and Cliff's delta is medium). Furthermore, there are differences between FP-Growth and Eclat as well as ReLim and Eclat (p-value =0.031 while Cliff's delta is negligible).

Regarding the F-score, as it can be noticed from Table 5.12, there are statistically significantly differences between Apriori and Eclat (p-value =0.035 and Cliff's delta is large) as well as ReLim and Apriori (p-value =0.031 and a large Cliff's delta).

Overall, for Jabref, we reject the null hypothesis $H_{0\text{-}24}$ for the comparison between Apriori and Eclat since there are statistically significant differences between these two algorithms in terms of precision, recall, as well as F-measure with a medium to large effect-size. Other instances such as the comparison between ReLim and Apriori yield statistically significant differences but this is limited to recall and F-measure.

5.2.5 Kotlin

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for Kotlin?

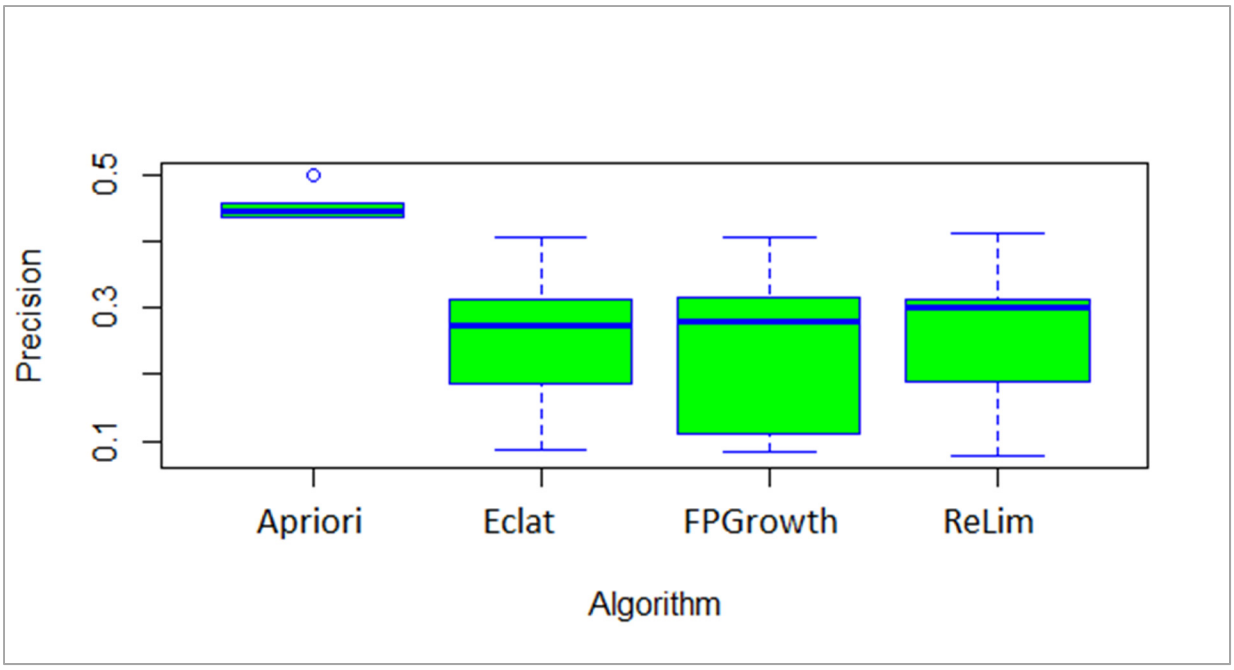

Figure 5.13 Boxplots of precision for the Kotlin project

The boxplots in Figure 5.13 show that Apriori is producing better results than other data mining techniques in terms of precision.

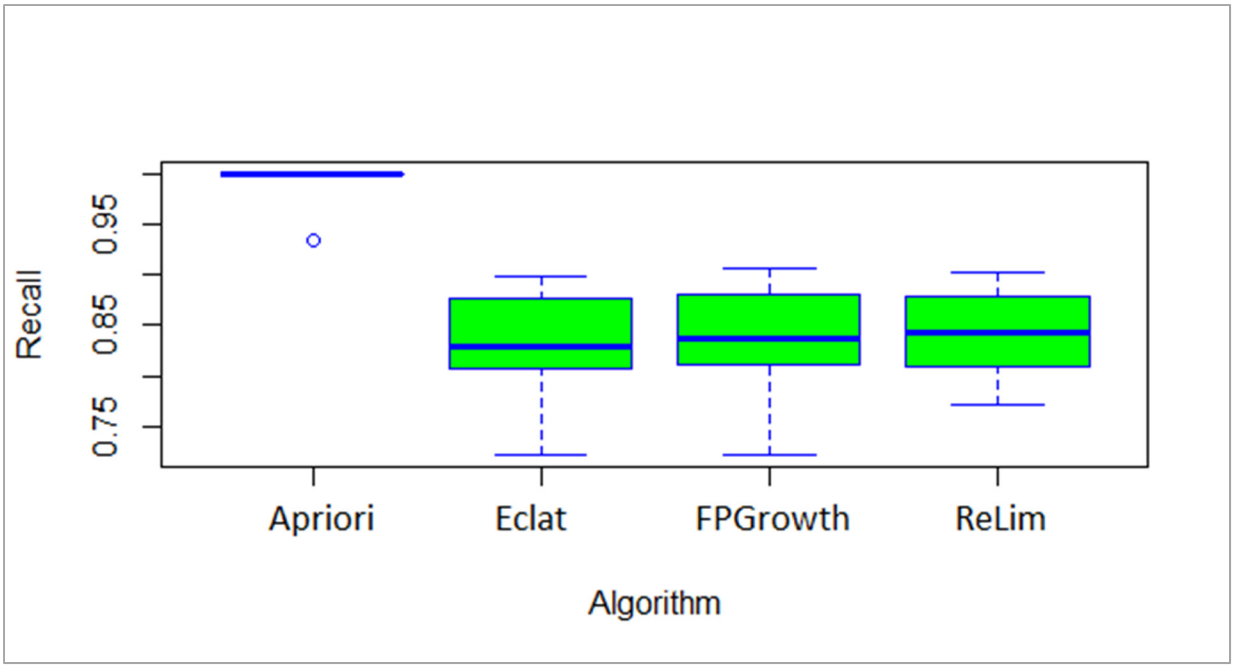

Figure 5.14 Boxplots of recall for the Kotlin project

The boxplots in Figure 5.14 show that Apriori is producing better results than other data mining techniques in terms of recall.

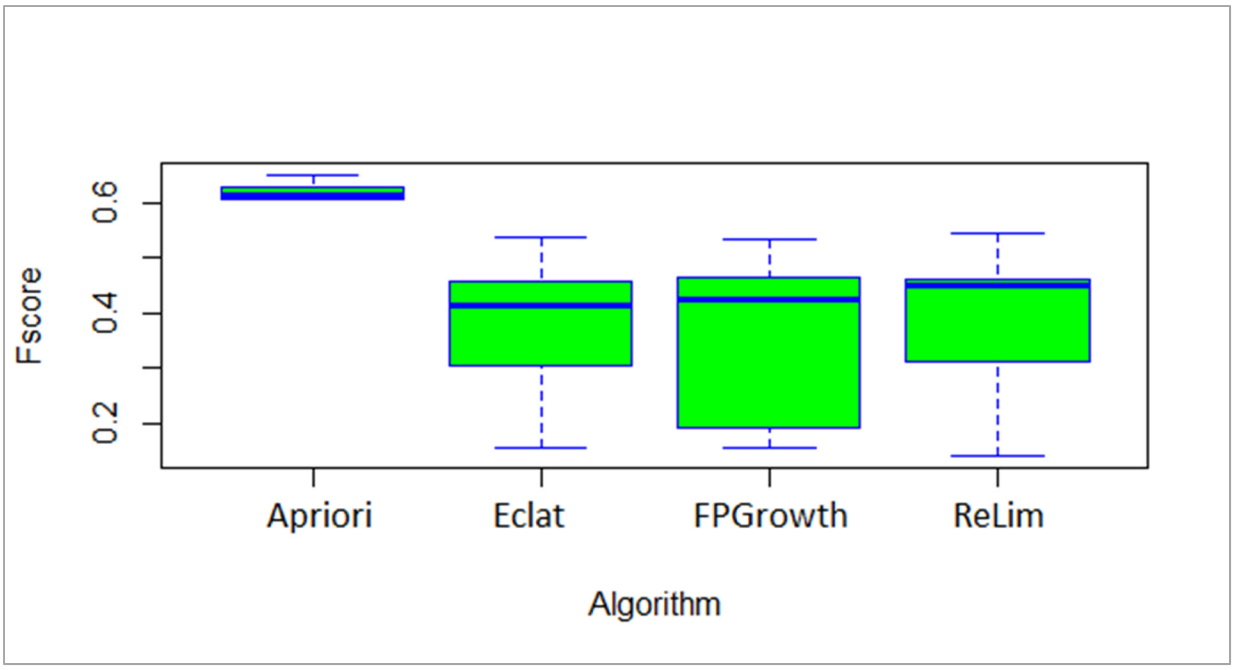

Figure 5.15 Boxplots of F-measure for the Kotlin project

The boxplots in Figure 5.15 show that Apriori is producing better results than other data mining techniques in terms of Fscore.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.13 P-values and Cliff's delta for paired algorithms for Kotlin

Algorithms	**Precision**		**Recall**		**F-score**	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	**0.031**	**1 (large)**	**0.031**	**1 (large)**	**0.031**	**1 (large)**
ReLim - FPGrowth	0.079	0.068 (negligible)	0.9	0.02 (negligible)	0.1	0.06 (negligible)
ReLim - Apriori	**0.031**	**-1 (large)**	**0.031**	**-1 (large)**	**0.031**	**-1 (large)**
Apriori - Eclat	**0.031**	**1 (large)**	**0.031**	**1 (large)**	**0.031**	**1 (large)**
ReLim - Eclat	0.1	0.1 (negligible)	0.53	0.14 (negligible)	0.06	0.13 (negligible)
FPGrowth - Eclat	0.83	0.02 (negligible)	0.31	0.13 (negligible)	0.38	0.06 (negligible)

For the Kotlin project, regarding the precision: As it can be noticed from Table 5.13, there are statistically significantly differences between Apriori and FPGrowth (p-value =0.03125 and Cliff's delta is large) as well as ReLim and Apriori (p-value =0.031 and Cliff's delta is large),

and between Apriori and Eclat (p-value =0.031 and Cliff's delta is large). The same trends are observed for recall as well as F-measure for these instances of comparisons.

We therefore reject the null hypothesis $H_{0\text{-}21}$, $H_{0\text{-}23}$ and $H_{0\text{-}24}$ for the comparisons between Apriori and FPGrowth, ReLim and Apriori, as well as Apriori and Eclat.

Overall, we can conclude, that for the Kotlin project, Apriori is outperforming Eclat, ReLim and FPGrowth in terms of precision, recall, as well as F-measure.

5.2.6 Rhino

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for Rhino?

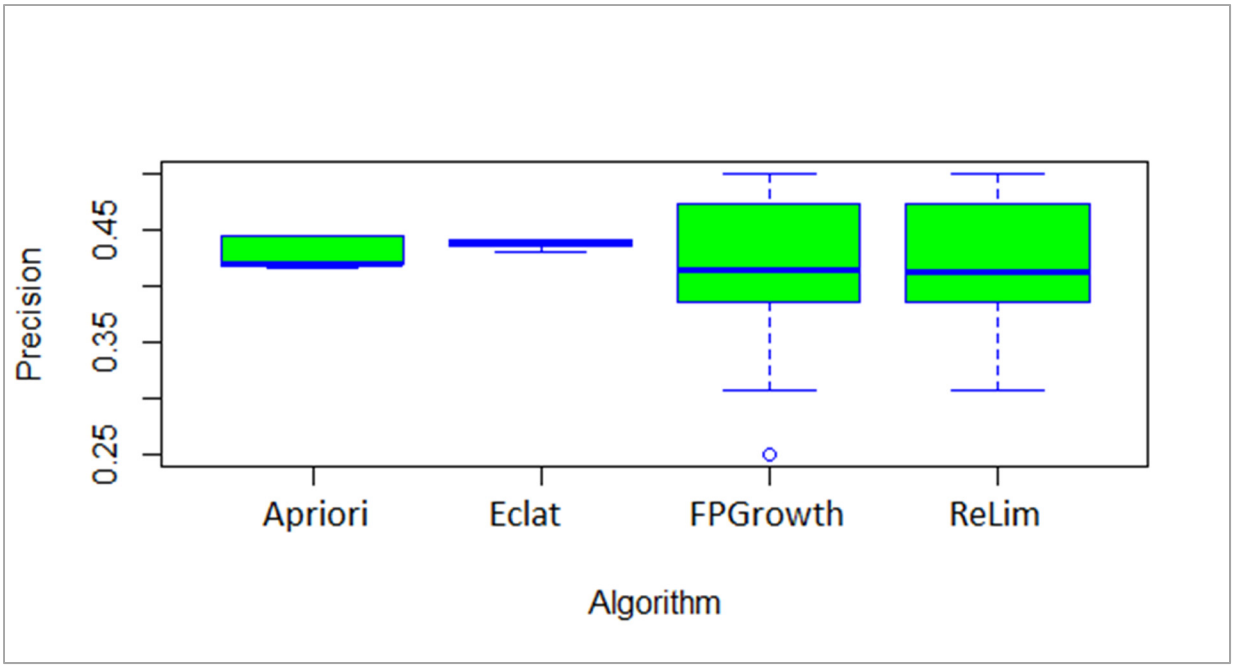

Figure 5.16 Boxplots of precision for the Rhino project

The boxplots in Figure 5.16 show that Eclat is slightly better than ReLim and FPGrowth in terms of precision.

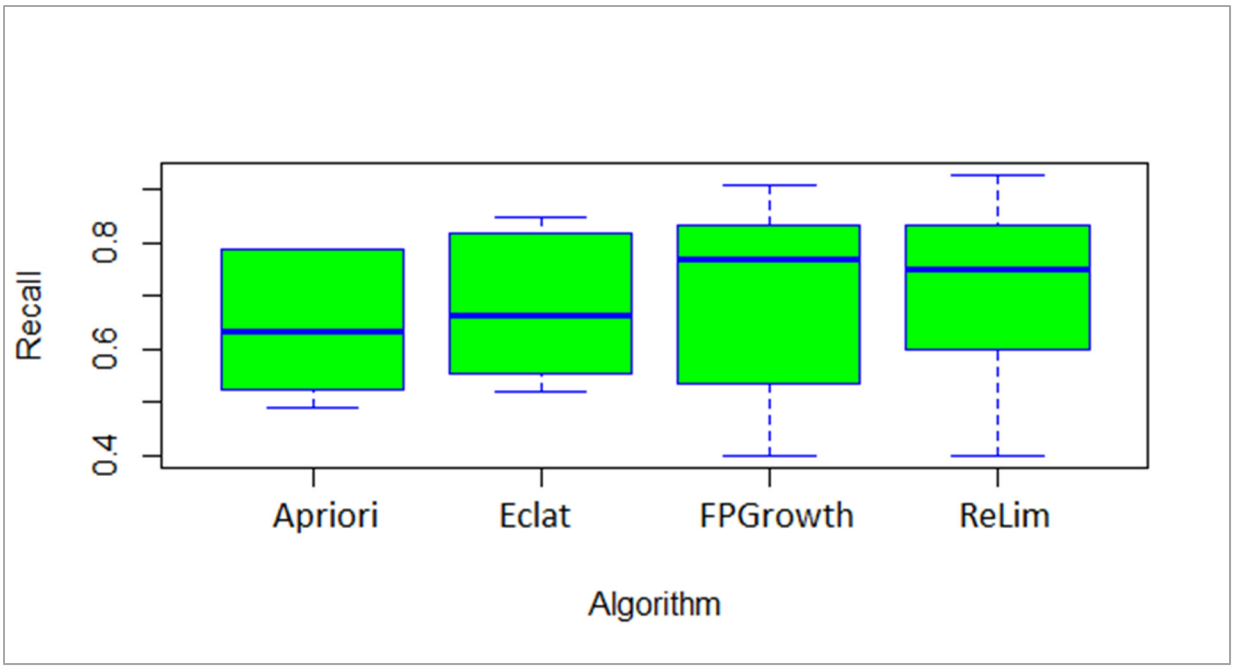

Figure 5.17 Boxplots of recall for the Rhino project

The boxplots in Figure 5.17 show that ReLim and FPGrowth are slightly better than Eclat and Eclat is slightly better than Apriori in terms of recall.

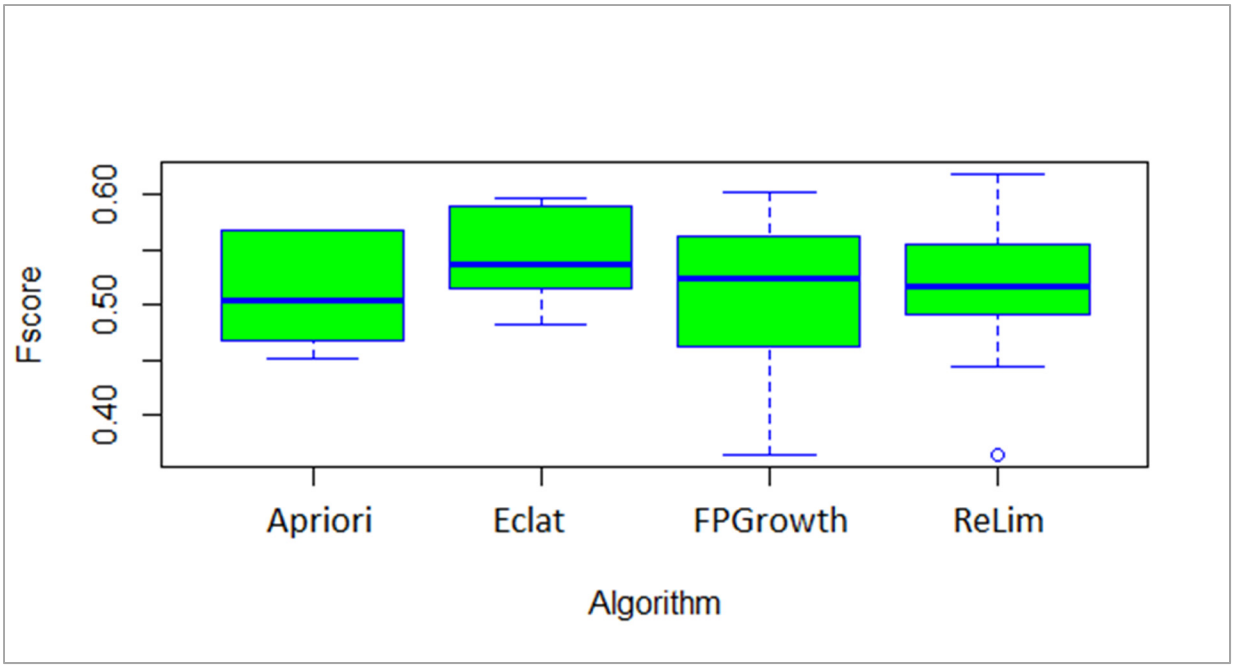

Figure 5.18 Boxplots of F-measure for the Rhino project

The boxplots in Figure 5.18 show that Eclat is performing slightly better than other data mining techniques.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.14 P-values and Cliff's delta for paired algorithms for Rhino

Algorithms	**Precision**		**Recall**		**F-score**	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	0.09	0.1388889 (negligible)	0.14	-0.26 (small)	0.84	-0.000923 (negligible)
ReLim - FPGrowth	0.81	0.009259259 (negligible)	0.59	-0.018 (negligible)	0.9	0.046 (negligible)
ReLim - Apriori	0.06	-0.1388889 (negligible)	0.14	0.26 (small)	0.84	0.11 (negligible)
Apriori - Eclat	0.16	-0.3333333 (medium)	**0.031**	**-0.28 (small)**	**0.031**	**-0.44 (medium)**
ReLim - Eclat	**0.031**	**-0.2222222 (small)**	0.56	0.13 (negligible)	0.56	-0.17 (small)
FPGrowth - Eclat	**0.031**	**-0.2222222 (small)**	0.31	0.13 (negligible)	0.69	-0.22 (small)

For the Rhino project, regarding the precision: As it can be noticed from Table 5.14, there are statistically significantly differences between ReLim - Eclat as well as FPGrowth and Eclat (p-value =0.031 and Cliff's delta is small).

For what concerns the recall, Table 5.14 shows that there are statistically significantly differences for only one instance of comparison, i.e., Apriori and Eclat (p-value =0.031 while Cliff's delta is small).

Regarding the F-score, Table 5.14 also indicates a statistically significantly difference, for one instance only, i.e., Apriori and Eclat (p-value =0.031 with a medium Cliff's delta).

For Rhino, we conclude that there are statistically significant differences for some instances of comparisons. Specifically, Eclat yield some better performances than ReLim and FPGrowth for precision (with a small effect-size). Additionally, it performs better than Apriori in terms of recall (small effect-size) and F-measure (medium effect-size).

5.2.7 SWT

Are there any differences between the four advanced data mining techniques considered, i.e., Apriori, FP-Growth, Eclat and ReLim in terms of their performance when recommending source code file changes for SWT?

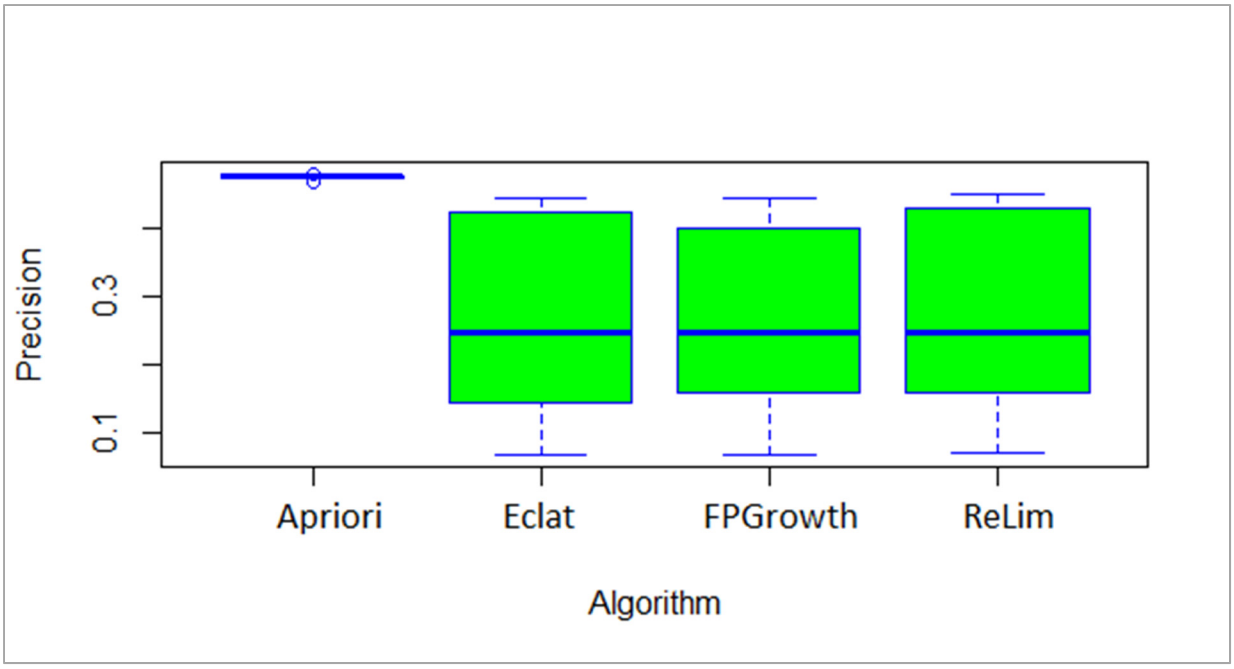

Figure 5.19 Boxplots of precision for the SWT project

The boxplots in Figure 5.19 show that Apriori is performing better than Eclat, FPGrowth and ReLim.

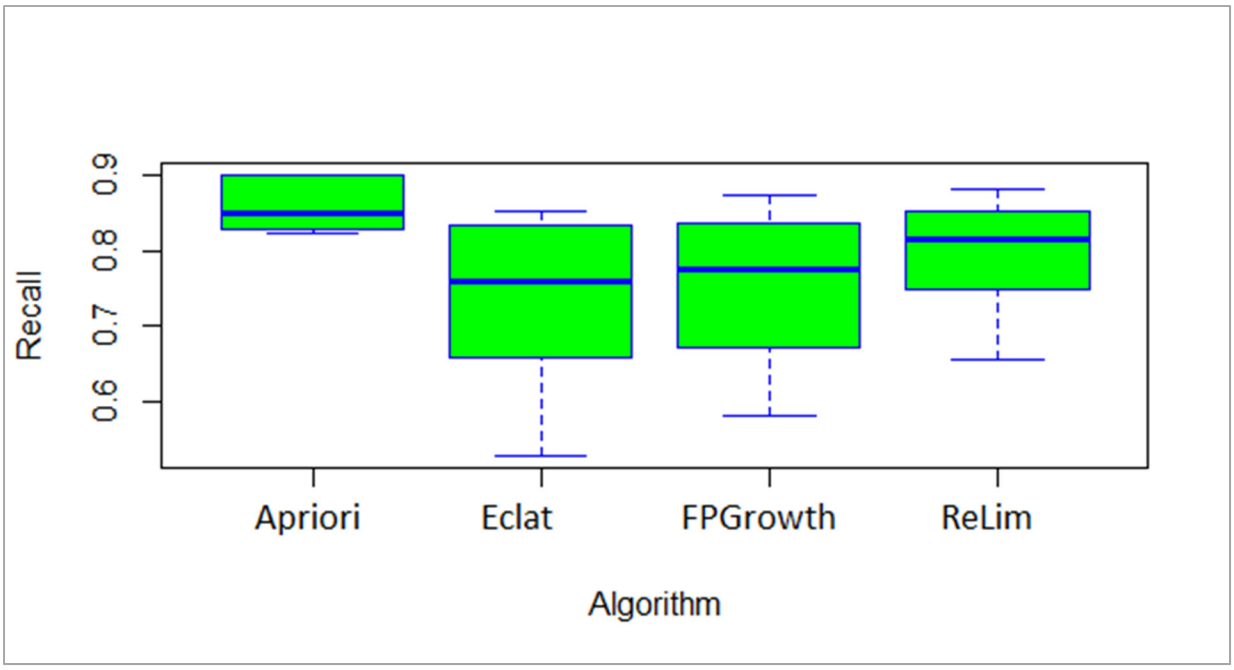

Figure 5.20 Boxplots of recall for the SWT project

The boxplots in Figure 5.20 show that Apriori and ReLim are performing slightly better than Eclat and FPGrowth.

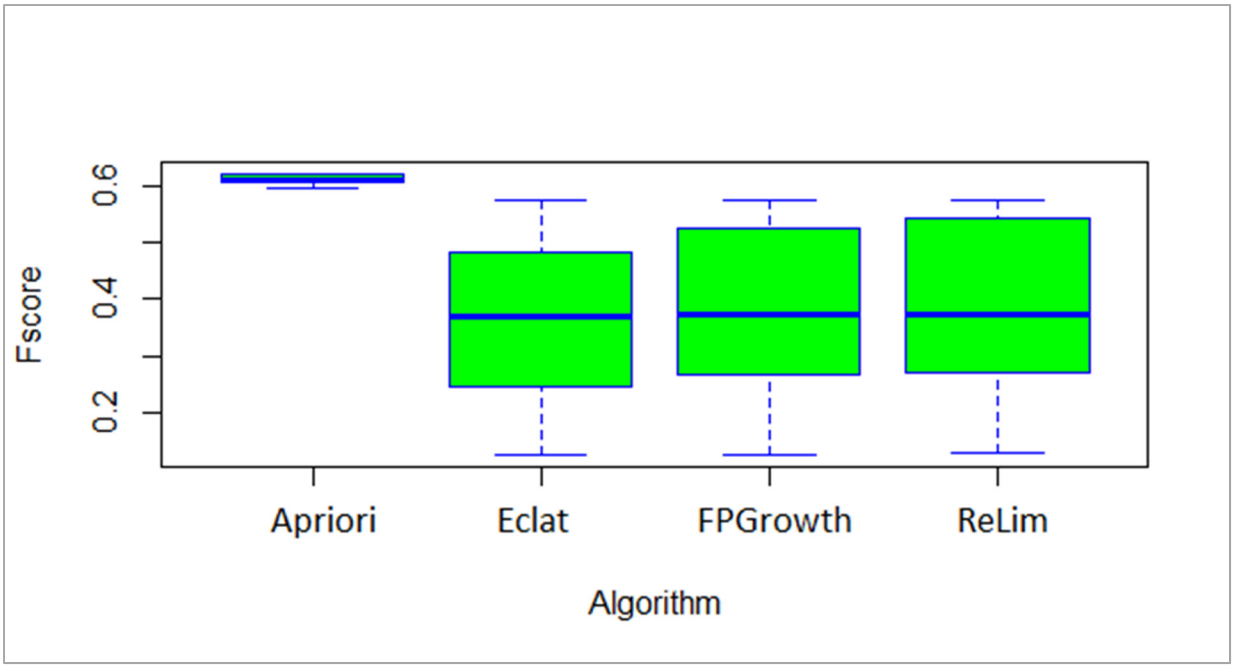

Figure 5.21 Boxplots of F-measure for the SWT project

The boxplots in Figure 5.21 show that Apriori is performing better than other data mining techniques. While, Relim is slightly better than FPGrowth and Eclat.

In the following, we will present the statistical tests corresponding to the different comparisons.

Table 5.15 P-values and Cliff's delta for paired algorithms for SWT

Algorithms	**Precision**		**Recall**		**F-score**	
	Adj. P-values	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**	**Adj. P-values**	**Cliff's Delta**
Apriori - FPGrowth	**0.031**	**1 (large)**	0.16	0.74(large)	**0.031**	**1 (large)**
ReLim - FPGrowth	**0.014**	**0.1 (negligible)**	**0.01**	**0.27(small)**	**0.012**	**0.1358025 (negligible)**
ReLim - Apriori	**0.031**	**-1 (large)**	0.22	-0.51 (large)	**0.031**	**-1 (large)**
Apriori - Eclat	**0.031**	**1 (large)**	0.06	0.72 (large)	**0.031**	**1 (large)**
ReLim - Eclat	0.12	0.07 (negligible)	**0.0045**	**0.35 (medium)**	**0.018**	**0.1358025 (negligible)**
FPGrowth - Eclat	0.68	-0.003 (negligible)	0.13	0.099 (negligible)	0.3	0.03703704 (negligible)

For the SWT project, regarding the precision: As it can be noticed from Table 5.15, there are statistically significantly differences between Apriori and FPGrowth, ReLim and Apriori, as

well as Apriori and Eclat (p-value = 0.031 and Cliff's delta is large). Additionally, there are differences between ReLim and FPGrowth (p-value = 0.014 but with a negligible effect-size).

For what concerns the recall, Table 5.15 reports that there are statistically significantly differences between ReLim and Eclat (p-value = 0.0045 and Cliff's delta is medium) as well as differences between ReLim and FPGrowth (p-value = 0.01 and Cliff's delta is small).

Regarding the F-score, similarly to the precision, we obtained statistically significantly differences between Apriori and FPGrowth, ReLim and Apriori, as well as Apriori and Eclat (p-value = 0.031 and Cliff's delta is large). Other instances of comparisons yield statistical differences such as ReLim and FPGrowth as well as ReLim and Eclat but the effect size is not significant.

For SWT, we conclude that, overall, Apriori yields better performances than Eclat, ReLim and FPGrowth since significant differences have been obtained with large effect-size for F-score.

Overall, the main results obtained can be summarized as follows.

Apriori yields better performances than Eclat, ReLim and FPGrowth in particular in terms of precision and Fscore as the case for Kotlin, Elasticsearch and SWT projects, and sometimes in terms of recall too (e.g., Eclipse and Jabref projects).

Eclat shows better performances than ReLim and FPGrowth in terms of precision and F-score for Rhino, it yields better performances than Aprior in terms of recall such as the case for Eclipse, as well as in terms of all performance measures as in the case of the Guava project. Additionally, Eclat performs better than FPGrowth for all performance measures in the Elasticsearch project. ReLim and FPGrowth have shown better performances than Apriori in terms of recall for the Guava project.

A possible explanation to the obtained findings is the fact that the Apriori algorithm generates less candidate set of itemsets while running, which results in better performance for projects such as Kotlin, Elasticsearch and SWT, having 15k, 12k and 8k transactions, respectively, in comparison with other projects with less than 3k transactions. Additionally, data mining

techniques including Apriori and Eclat are applying optimized frequent itemsets generation strategies such as pruning techniques in Apriori and vertical-based mining in Eclat, which results in a decrease in the number of candidates and comparisons, consequently it yields better performance for larger databases as mentioned.

Like any research work, our work has its limitations. One of the main limitations is that the granularity level of our approach is limited to the file level. Our approach could be extended to other levels such as the method level, which we consider as part of our future work. Another limitation is that our approach is limited to recommending source code changes for Java files only for instance, we therefore can address more programming language such as C# and Python. Clearly, our empirical study has its threats to validity also, which we discuss in details in the next section.

5.3 Threats to validity

There are threats to validity related to our empirical study in spite of the fact that we have evaluated it empirically and statistically. The most frequent ones are discussed in the following section:

5.3.1 Internal validity

Internal validity is the extent to which a study results are reliable or accurate (Godwin et al., 2003). We provided empirical evidence of the differences between traditional data mining algorithms and advanced ones. Possible threats could be related to the data filtering that we did when filtering source code file changes. Yet, when dealing with such a process, we mirrored widely-adopted filtering ways as in previous works (Zimmermann, Zeller, Weissgerber, & Diehl, 2005), (Ying, Murphy, Ng, & Chu-Carroll, 2004).

5.3.2 External validity

External validity is the extent to which a study results are generalizable (Godwin et al., 2003). Our study examined seven real open-source projects: Eclipse, ElasticSearch, SWT, Kotlin,

Guava, SWT and JabRef. The projects may be not representative of all software projects. However, they are different in size and domains. In the future, it would be interesting to consider more projects in other contexts as well, the industrial field.

5.3.3 Reliability validity

Reliability is the extent to which a study is producing analogous results in variuos conditions (Godwin et al., 2003). The source code repositories of Eclipse, ElasticSearch, Rhino, SWT, Kotlin, Guava and JabRef are publicly-available. The frame-times of the code histories that we investigated are reported in the thesis. Additionally, we used the apriori package of Python to run Apriori, as well as running FPGrowth and ReLim by pymining Python package (using the parameters reported in the thesis), while we used the source code of Eclat made available by their authors.

5.3.4 Conclusion Validity

Conclusion validity is the extent to which the conclusions of a study is arising out of sufficient data analysis and all the research questions are accurately and logically answered (MA García-Pérez, 2012). We have performed adequate tests, for the purpose of comparison, in order to statistically reject the null hypotheses. In particular, we used nonparametric tests, which do not make any assumption on the distributions of the data, and, specifically, the Wilcoxon pair-wise test. In addition, our conclusions have not been based only on the presence of significant differences between pairs of data mining algorithms, but also on the effect-size measure presence, i.e., the magnitude of the difference, between two algorithms. Furthermore, we dealt with issues related to performing multiple Wilcoxon tests using the Holm correction.

CONCLUSION

The use of advanced data mining techniques for generating predictions of phenomena or patterns or recommending source code file changes is becoming increasingly popular and has been quite common since the advent of big data. In this work, our focus is on four advanced data mining algorithms that mine development history to recommend source code file changes. To the best of our knowledge. while some research works have attempted to suggest recommendations for source code changes, they leveraged simple data mining algorithms such as Apriori, they have not investigated other advanced data mining techniques. Additionally, they have not investigated different configurations of support and confidence and they have not compared their work to any baseline and–or previous works. To the best of our knowledge, we are the first to suggest recommending source code file changes by applying four different algorithms and comparing their performances using a large number of configurations. We believe such research can be beneficial to both the research community and practitioners interested in leveraging appropriate data mining algorithms for their research context. In this paper, we applied four advanced mining algorithms: Apriori, FP-Growth, Eclat and Relim on the development history of seven widely-used open-source projects: Eclipse, ElasticSearch, Rhino, SWT, Kotlin, Guava and JabRef. We also compare the performance results of the four investigated algorithms in terms of their precision, recall, and F-measure, while using as our baseline the Apriori algorithm.

Our findings demonstrate that, Apriori is outperforming Eclat, ReLim and FPGrowth in particular in terms of precision and Fscore as the case for Kotlin, Elasticsearch and SWT projects, and sometimes in terms of recall too (e.g., Eclipse and Jabref projects). Eclat yields performances than Aprior in terms of recall such as the case for Eclipse, as well as in term of all performance measures as in the case of the Guava project. Also, Eclat outperforms FPGrowth for all performance measures in the Elasticsearch project. On the other hand, ReLim and FPGrowth have shown better performances than Apriori in terms of recall for the Guava project.

As future work, we plan to explore more projects with different programming languages other than Java, as well as extending the approach to more fine-grained levels such as methods. We also intend to improve the quality of our transactions by using different filters on our development history and considering metrics on the quality of transactions to identify only those that are relevant to the developer's task at hand. Finally, one idea could be to build new models of source code changes that consider the temporal aspect of change history, and in particular, the order of the source code changes.

APPENDIX

In Table A, some generated rules from seven projects are illustrated, in which frequent files changed together are shown.

Table A Generated rules that show changed files together for different projects

Eclipse
"frozenset({'org.eclipse.jdt.core/batch/org/eclipse/jdt/internal/compiler/batch/Main.java', 'org.eclipse.jdt.core/model/org/eclipse/jdt/core/JavaCore.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/problem/ProblemReporter.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/core/compiler/IProblem.java'})", "frozenset({'org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/problem/ProblemReporter.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/core/compiler/IProblem.java', 'org.eclipse.jdt.core/model/org/eclipse/jdt/core/JavaCore.java'})", "frozenset({'org.eclipse.jdt.core/model/org/eclipse/jdt/core/JavaCore.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/core/compiler/IProblem.java'})", "frozenset({'org.eclipse.jdt.core/model/org/eclipse/jdt/core/JavaCore.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/problem/ProblemReporter.java'})", "frozenset({'org.eclipse.jdt.core/batch/org/eclipse/jdt/internal/compiler/batch/Main.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/problem/ProblemReporter.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/core/compiler/IProblem.java'})", "frozenset({'org.eclipse.jdt.core/batch/org/eclipse/jdt/internal/compiler/batch/Main.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/core/compiler/IProblem.java'})", "frozenset({'org.eclipse.jdt.core/batch/org/eclipse/jdt/internal/compiler/batch/Main.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/problem/ProblemReporter.java'})", "frozenset({'org.eclipse.jdt.core/batch/org/eclipse/jdt/internal/compiler/batch/Main.java', 'org.eclipse.jdt.core/compiler/org/eclipse/jdt/core/compiler/IProblem.java', 'org.eclipse.jdt.core/model/org/eclipse/jdt/core/JavaCore.java'})", "frozenset({'org.eclipse.jdt.core/batch/org/eclipse/jdt/internal/compiler/batch/Main.java', 'org.eclipse.jdt.core/model/org/eclipse/jdt/core/JavaCore.java'})"
Elasticsearch
"frozenset({'src/main/java/org/elasticsearch/index/mapper/core/LongFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/ip/IpFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ShortFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/DateFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/DoubleFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/FloatFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ByteFieldMapper.java'})", "frozenset({'src/main/java/org/elasticsearch/index/mapper/core/DateFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/DoubleFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ByteFieldMapper.java',

'src/main/java/org/elasticsearch/index/mapper/core/FloatFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/ip/IpFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ShortFieldMapper.java'})", "frozenset({'src/main/java/org/elasticsearch/index/mapper/core/DoubleFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/ip/IpFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/FloatFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ByteFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ShortFieldMapper.java'})", "frozenset({'src/main/java/org/elasticsearch/index/mapper/core/ByteFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/FloatFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/ip/IpFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ShortFieldMapper.java'})", "frozenset({'src/main/java/org/elasticsearch/index/mapper/ip/IpFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/FloatFieldMapper.java', 'src/main/java/org/elasticsearch/index/mapper/core/ShortFieldMapper.java'})",
Kotlin
"frozenset({'compiler/frontend/src/org/jetbrains/jet/di/InjectorForLazyResolve.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForMacros.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerBasic.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForBodyResolve.java', 'compiler/frontend.java/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerForJvm.java'})", "frozenset({'compiler/frontend/src/org/jetbrains/jet/di/InjectorForMacros.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerBasic.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForBodyResolve.java', 'compiler/frontend.java/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerForJvm.java'})", "frozenset({'compiler/frontend/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerBasic.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForBodyResolve.java', 'compiler/frontend.java/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerForJvm.java'})", "frozenset({'compiler/frontend/src/org/jetbrains/jet/di/InjectorForMacros.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForBodyResolve.java', 'compiler/frontend.java/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerForJvm.java'})", "frozenset({'compiler/frontend/src/org/jetbrains/jet/di/InjectorForMacros.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerBasic.java', 'compiler/frontend.java/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerForJvm.java'})", "frozenset({'compiler/frontend/src/org/jetbrains/jet/di/InjectorForMacros.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForTopDownAnalyzerBasic.java', 'compiler/frontend/src/org/jetbrains/jet/di/InjectorForBodyResolve.java'})",
Guava
"frozenset({'guava/src/com/google/common/collect/Multimaps.java','guava-gwt/src- super/com/google/common/collect/super/com/google/common/collect/FluentIterable.java', 'guava/src/com/google/common/collect/Iterators.java'})", "frozenset({'guava/src/com/google/common/graph/Network.java', 'guava/src/com/google/common/graph/Graph.java', 'guava/src/com/google/common/collect/Maps.java'})",

"frozenset({'guava-gwt/src-super/com/google/common/collect/super/com/google/common/collect/Maps.java', 'guava/src/com/google/common/util/concurrent/Futures.java', 'guava-gwt/src-super/com/google/common/collect/super/com/google/common/collect/Sets.java'})"],
Jabref
"frozenset({'src/java/net/sf/jabref/JabRefFrame.java','src/java/net/sf/jabref/GUIGlobals.java', 'src/java/net/sf/jabref/JabRefPreferences.java'})", "frozenset({'src/main/java/net/sf/jabref/gui/BasePanel.java', 'src/main/java/net/sf/jabref/gui/actions/Actions.java'})", "frozenset({'src/main/java/net/sf/jabref/gui/BasePanel.java', 'src/main/java/net/sf/jabref/gui/JabRefFrame.java'})", "frozenset({'src/main/java/net/sf/jabref/gui/JabRefFrame.java', 'src/main/java/net/sf/jabref/sql/importer/DbImportAction.java'})" "frozenset({'src/java/net/sf/jabref/JabRefFrame.java','src/java/net/sf/jabref/BasePanel.java', 'src/java/net/sf/jabref/JabRefPreferences.java'})", "frozenset({'src/java/net/sf/jabref/JabRefFrame.java','src/java/net/sf/jabref/BasePanel.java', 'src/java/net/sf/jabref/GUIGlobals.java'})", "frozenset({'src/main/java/net/sf/jabref/gui/JabRefFrame.java', 'src/main/java/net/sf/jabref/gui/actions/Actions.java'})",
Rhino
"frozenset({'src/org/mozilla/javascript/Interpreter.java', 'src/org/mozilla/javascript/optimizer/Codegen.java'})", "frozenset({'src/org/mozilla/javascript/Token.java', 'src/org/mozilla/javascript/optimizer/Codegen.java'})", "frozenset({'src/org/mozilla/javascript/Token.java', 'src/org/mozilla/javascript/Interpreter.java'})", "frozenset({'src/org/mozilla/javascript/optimizer/Codegen.java', 'src/org/mozilla/javascript/Interpreter.java'})", "frozenset({'src/org/mozilla/javascript/IRFactory.java', 'src/org/mozilla/javascript/Interpreter.java'})", "frozenset({'src/org/mozilla/javascript/IRFactory.java', 'src/org/mozilla/javascript/optimizer/Codegen.java'})", "frozenset({'src/org/mozilla/javascript/optimizer/Codegen.java', 'src/org/mozilla/javascript/Parser.java'})", "frozenset({'src/org/mozilla/javascript/Interpreter.java', 'src/org/mozilla/javascript/Parser.java'})", "frozenset({'src/org/mozilla/javascript/Interpreter.java', 'src/org/mozilla/javascript/optimizer/Codegen.java'})", "frozenset({'src/org/mozilla/javascript/optimizer/Codegen.java', 'src/org/mozilla/javascript/ScriptRuntime.java'})", "frozenset({'src/org/mozilla/javascript/Interpreter.java', 'src/org/mozilla/javascript/ScriptRuntime.java'})",

"frozenset({'src/org/mozilla/javascript/Interpreter.java', 'src/org/mozilla/javascript/optimizer/Codegen.java'})"
SWT
"frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Table.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Control.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/List.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Table.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Tree.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Table.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Control.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Tree.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/List.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Tree.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/List.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Table.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Combo.java','bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Table.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Control.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Table.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Control.java'})", "frozenset({'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Combo.java', 'bundles/org.eclipse.swt/EclipseSWT/gtk/org/eclipse/swt/widgets/Control.java'})",

BIBLIOGRAPHY

Barbosa, E. A., & Garcia, A. (2017). Global-aware recommendations for repairing violations in exception handling. *IEEE Transactions on Software Engineering*, 44(9), 855-873.

Barbosa, E. A., & Garcia, A. (2017). Global-aware recommendations for repairing violations in exception handling. *IEEE Transactions on Software Engineering*, 44(9), 855-873.

Borg, M., Wnuk, K., Regnell, B., & Runeson, P. (2016). Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. *IEEE Transactions on Software Engineering*, 43(7), 675-700.

Canfora, G., & Cerulo, L. (2005). Impact analysis by mining software and change request repositories. *11th IEEE International Software Metrics Symposium (METRICS'05)* , (pp. 9-pp).

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3), 494.

Dotzler, G., Kamp, M., Kreutzer, P., & Philippsen, M. (2017). More accurate recommendations for method-level changes. *In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* , pp. 798-808.

Hassan, A. E., & Xie, T. (2010). Mining software engineering data . *In 2010 ACM/IEEE 32nd International Conference on Software Engineering* , (Vol. 2, pp. 503-504). IEEE.

Heo, K., Oh, H., & Yang, H. (2019). Resource-aware program analysis via online abstraction coarsening. *In Proceedings of the 41st International Conference on Software Engineering* , (pp. 94-104). IEEE Press.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, 65-70.

Huang, Q., Xia, X., Xing, Z., Lo, D., & Wang, X. (2018). API method recommendation without worrying about the task-API knowledge gap. *In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* , pp. 293-304.

Isinkaye, F. O., Folajimi, Y. O., & Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3), 261-273.

Kovalenko, V., Palomba, F., & Bacchelli, A. (2018). Mining file histories: should we consider branches? . *In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* , (pp. 202-213). ACM.

Lee, S., & Kim, S. (2015). The Impact of View Histories on Edit Recommendations. *IEEE Transactions on Software Engineering*, vol. 41, p. 314-30.

Liu, X., Huang, L., & Ng, V. (2018). Effective API recommendation without historical software repositories. *In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* , (pp. 282-292). ACM.

Malheiros, Y., Moraes, A., Trindade, C., & Meira, S. (2012). A source code recommender system to support new comers. *In Proceedings of the 36th Annual Computer Software and Applications Conference, (COMPSAC '12)*, p. 19-24.

Malik, H., & Shakshuki, E. (2010). Predicting function changes by mining revision history. . *In 2010 Seventh International Conference on Information Technology: New Generations* , (pp. 950-955). IEEE.

Martinez, M., & Monperrus, M. (2019). Coming: a tool for mining change pattern instances from git commits. *In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings* , (pp. 79-82). IEEE Press.

McGill, R., Tukey, J. W., & Larsen, W. A. (1978). Variations of box plots. *The american statistician*, 32(1), 12-16.

Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., & Marcus, A. (2015). How can I use this method? *IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1, pp. 880-890.

Nagashima, Y., & He, Y. (2018). PaMpeR: proof method recommendation system for Isabelle/HOL. *In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, (pp. 362-372).

Naghshzan, A., Guerrouj, L., & Baysal, O. (2021). Leveraging Unsupervised Learning to Summarize APIs Discussed in Stack Overflow. *In 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)* , (pp. 142-152).

Nguyen, H. A., Nguyen, T. N., Dig, D., Nguyen, S., Tran, H., & Hilton, M. (2019). Graph-based mining of in-the-wild, fine-grained, semantic code change patterns. *In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 819-830.

Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., & Di Penta, M. (2019). Focus: A recommender system for mining api function calls and usage patterns. *In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 1050-1060.

Palomba, F., Salza, P., Ciurumelea, A., Panichella, S., Gall, H., Ferrucci, F., & De Lucia, A. (2017). Recommending and localizing change requests for mobile apps based on user reviews. *In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* , (pp. 106-117). IEEE.

Ponzanelli, L., Scalabrino, S., Bavota, G., Mocci, A., Oliveto, R., Di Penta, M., & Lanza, M. (2017). Supporting software developers with a holistic recommender system. *In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, (pp. 94-105). IEEE.

Robbes, R., Pollet, D., & Lanza, M. (2010). Replaying ide interactions to evaluate and improve change prediction approaches. *In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, (pp. 161-170). IEEE.

Sisman, B., & Kak, A. C. (2012). Incorporating version histories in information retrieval based bug localization. *In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR)* , (pp. 50-59). IEEE.

Uddin, G., Dagenais, B., & Robillard, M. P. (2011). Analyzing temporal API usage patterns. *In 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* , (pp. 456-459). IEEE.

Webb, G. I. (1989). A machine learning approach to student modelling. *In Proceedings of the third Australian joint conference on artificial intelligence* , (pp. 195-205).

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *biometrics bulletin* , 1, 6 (1945), 80–83. URL http://www. jstor. org/stable/3001968.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering . *Springer Science & Business Media*.

Ying, A. T., Murphy, G. C., Ng, R., & Chu-Carroll, M. C. (2004). Predicting source code changes by mining change history . *IEEE transactions on Software Engineering*, 30(9), 574-586.

Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., & Kim, M. (2018). Are code examples on an online Q&A forum reliable?: a study of API misuse on stack overflow. *In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, (pp. 886-896). IEEE.

Zhong, Hao, Xie, T., Zhang, L., Pei, J., & Mei, H. (2009). "MAPO: Mining and Recommending API Usage Patterns". *ECOOP '09. ACM, New York, NY*, p. 318-343.

Zimmermann, T., Zeller, A., Weissgerber, P., & Diehl, S. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6), 429-445.