Combining Association Mining with Topic Modeling to Discover More File Relationships

Namita Dave, Karen Potts, Vu Dinh, and Hazeline U. Asuncion
School of Science, Technology, Engineering, and Mathematics
University of Washington Bothell
Bothell, WA, USA
{namitad , pottsk2 , vdinh143, hazeline}@u.washington.edu

Abstract— **Software maintenance tasks require familiarity with the entire software system to make proper changes. Often, maintenance engineers who did not develop the software are tasked with corrective or adaptive maintenance tasks. As a result, modifying the software becomes a time-consuming process due to their lack of familiarity with the source code. To help software engineers locate relevant files for a maintenance task, association mining has been used to identify the files that frequently change together in a software repository. However, association mining techniques are limited to the amount of project history stored in a software repository. We address this difficulty by using a technique that combines association mining with topic modeling, referred to as Frequent Pattern Growth with Latent Dirichlet Allocation (FP-LDA). Topic modeling aims to uncover file relationships by learning semantic topics from source files. We validated our technique via experiments on seven open source projects with different project characteristics. Our results indicate that FP-LDA can find more related files than association mining alone. We also offer lessons learned from our investigation.**

Keywords-Association mining; Topic Modeling; Software Engineering.

I. INTRODUCTION

Software maintenance has been known to incur the highest cost among the different phases in the software lifecycle [1, 2]. This may be due to an engineer's unfamiliarity with the software to modify, requiring more time to understand the source code [3]. Maintenance tasks also become more difficult as the complexity of the code increases and as code degradation occurs over time, due to patches and workarounds [4].

To assist with software maintenance tasks, various techniques have been proposed to find related source code, including static and dynamic analyses, recommendation systems, and code search techniques. Static analysis techniques, more specifically, dependency analysis, provide file relationships based on call graphs [5]. Dynamic analysis tools, meanwhile, are able to identify relationships between files based on execution traces [6]. These techniques, however, are generally language-specific. Recommendation systems, meanwhile, provide possible files of interest based on a developer's past activities, textual similarity, check-in records, or email records [7, 8]. These systems generally use information retrieval techniques, along with user context, to provide files of interest. Code search techniques find related code based on syntactic or structural matches [9].

Association mining is another technique used to find related files. Association mining uncovers relationships between files, based on files that have been modified together in the past. This technique generates rules, which specify which files are frequently changed together. Unlike the other techniques, association mining is not specific to the programming language used or restricted to syntactic or structural matches of a query.

The most commonly used algorithms for association mining are Apriori [10] and Frequent Pattern Growth (FP-Growth) [11]. While these algorithms provide some level of accuracy, they are highly dependent on the project history. If there are not enough modifications in the software project, or if the modifications are sparse throughout the software system, there are fewer chances that association mining will result in correct rules.

Meanwhile, machine learning techniques, such as Latent Dirichlet Allocation (LDA), allow us to automatically detect relationships between files based on semantic similarity. LDA is an unsupervised statistical approach for learning semantic topics from a set of documents [12]. It is a fully automated approach that does not require training labels. It only requires a set of documents and number of topics to learn.

Thus, we aim to address the challenges of association mining by combining it with LDA. Our technique, Frequent Pattern Growth with Latent Dirichlet Allocation (FP-LDA), allows us to achieve better recall results than solely using association mining. By combining these two techniques, we are able to overcome the limitations of each technique. LDA allows us to find file associations even with limited modification history. Association mining, meanwhile, allows us to find associations among files where semantic similarities may not be readily apparent. We previously introduced FP-LDA [1] but this paper provides more details regarding our technique.

The contributions of this research paper are as follows: (1) combination of association mining and topic modeling to identify file relationships in a software project, (2) experiments on seven open source projects, and (3) lessons learned in effectively using these techniques. We also created a set of tools that automates the entire process—from pre-processing the data to querying related files.

We envision our technique being used in the following scenarios: a developer presented with a modification task knows at least one file to change and would like to know what other files to change; a technical lead wishes to ensure that changes performed by a teammate are complete; a maintenance engineer needs to perform impact analysis to determine the feasibility of changing a section of the code.

The rest of the paper is organized as follows. Section II covers background on association mining and Section III covers background on topic modeling. In Section IV, we present our combined approach, FP-LDA. We then validate our approach in Section V. Section VI covers lessons learned. We conclude with future work.

II. Association Mining

This section covers background on association mining, selection of the association mining technique, application, limitations, and related work.

A. Background

Association rule mining is a method used to discover patterns in large data sets. Initially, it was used in Market Basket Analysis to find how items bought by customers are related [13]. Rules are mined from the dataset, such as "Customers who bought item A also bought item B". In the case of mining file associations in software projects, rules such as "Developers who modified file A also modified file B" are mined [14]. In order to mine these rules, patterns must be analyzed in the dataset.

We now discuss the main ideas of association mining based on work by Agrawal and Srikant [10], as applied to software development.

$$I = \{i_1, i_2, ..., i_m\} \quad (1)$$

Let (1) represent the total set of items. In this paper, the files in the repository are items. T represents a set of transactions

$$T = \{t_1, t_2, ..., t_n\}, \quad (2)$$

which are in the software repository being mined. Each transaction t is a set of items such that $t \subseteq I$. In this paper, t represents one atomic commit.

Given the set of transactions T (see (2)), the goal of association mining is to find all the association rules that have support and confidence greater than the user specified threshold values. An itemset is a collection of items. The support is defined as the fraction of transactions that contain the itemset and from which the rule is derived. The confidence denotes the strength of a rule. An association rule is represented as

$$X \rightarrow Y \text{ [support = 20\%, confidence = 80\%]} \quad (3)$$

In this notation, itemset X is called the antecedent and itemset Y is called the consequent such that $X, Y \subseteq I$. Both antecedent and consequent are comprised of one or more items. Assume that both X and Y consist of one file, each namely x and y, respectively. Then, this rule says that in 20% of the check-in transactions, both x and y files are modified and the transactions, which changed file x, also changed file y 80% of the time.

The threshold support value specified by the user is called minimum support. This is an important element that makes association mining practical. It reduces the search space by limiting the number of rules generated [15]. The threshold confidence value specified by the user is called minimum confidence [15].

There are two types of measures for association mining: objective and subjective. Support and confidence, which we just discussed, are objective measures of association mining [16]. Subjective measures are unexpectedness and actionability [17]. The generated rules are "unexpected" or surprising if the relationship is not obvious to the user. For example a file customers.h is, most of the time, going to change if customers.c is modified. Such a rule, though valid has little usefulness to the developer. However, if the recommendations help the developer to perform her task effectively, then such rules have high actionability. Actionability refers to the capability of the approach to yield a rule that can be acted upon with some advantage.

B. Selection of Association Mining Technique

There are two commonly used association mining techniques. The first sequential pattern mining algorithm used to mine rules was Apriori algorithm [10]. Later, the Frequent Pattern Growth Algorithm, or FP-Growth, was introduced [11]. We now discuss the ideas behind these two techniques and the rationale for selecting FP-Growth.

Apriori is a classic algorithm for learning association rules over transactional databases for sample collections of items bought by customers [10]. It works in two steps. In the first step, it generates the candidate itemsets. These are the set of items that have the minimum support. In the second step, association rules are generated. Apriori uses the property that any subsets of a frequent itemset are also frequent. The essential idea behind Apriori algorithm is that it iteratively generates candidate itemsets of length (k + 1) from frequent itemsets of length k and then tests their corresponding frequency in the database. Apriori is not efficient when used with large data sets, as generation of candidate item sets and support counting is very expensive, as confirmed in [18].

FP-Growth is a faster and more scalable approach to mine a complete set of frequent patterns by pattern fragment growth. This can be achieved by using a compact prefix tree structure for storing a transaction dataset [11]. This algorithm operates in two steps. In the first step, it creates a compact Frequent Pattern tree to encode the database. The construction of an FP-tree begins with pre-processing the input data with an initial scan of the database to count support for single items. The single items that do not meet the threshold support values are eliminated. The database is then scanned for the second time to produce an initial FP-tree. The second step runs a depth first recursive procedure to mine the FP-tree for frequent itemsets with increasing cardinality. The FP tree stores a single item at each node. The root node of an FP tree is empty. The path from the root to a node in the FP tree is a subset of the transactions database. The items in the path are in decreasing order of support. In the second step, the algorithm examines a conditional-pattern base for each itemset starting with length 1 and then constructs its own conditional FP-tree. Unlike the Apriori algorithm, it avoids generating expensive candidate itemsets. Each conditional FP-tree is recursively mined to generate frequent itemsets. The algorithm uses a divide-and-conquer approach to decompose the mining task into smaller tasks of mining the confined conditional databases. Interested readers can refer to work by Han, Pei and Yin for more information [11].

C. Application

Association mining has been used in the past to support various software engineering tasks. Some approaches to accomplish these tasks rely on structural analysis of code while others rely on textual mining. In mining associations from software projects, two data sources are primarily used as sequence-sources: the project history and the code structure. In cases where history or documentation is unavailable, the structure of the software may be analyzed by breaking it into groups, further decomposing them into entities, and then mining association rules from the entity sets [19]. MAPO, a tool for suggesting API usage patterns, analyzes sequences in code structure found within open-source repositories [20]. Clustering techniques have also been explored to find similarities in program entities in order to support software maintenance [21]. Techniques that rely on the structure of the software are useful in cases where one language is used or when the interoperation of processes is not a concern. In analyzing open source repositories, we have found that several types of code may be checked-in together. In addition, interoperating processes may not share dependencies in source descriptions, and yet they may pass messages, and thus rely upon each other.

Another source of data for mining associations between source files is found in the history logs of configuration management systems, such as those found in the open source repositories that we have mined. Association mining with FP-Growth has been applied to these change histories [14]. We build on this approach and we enhance this technique with the use of topic modeling. An example of a tool that performs association mining on history logs is Rose [19]. It is a tool that parses syntactic entities from the committed source code, such as classes, functions, and fields. Association mining is applied to this parsed version of the history data set. The association rules obtained could predict that programmers who changed a given entity also changed the recommended entities. Our approach is similar, in that we mine rules from the history of the repository. However, we do not provide the fine granularity of connection provided by parsing syntactic entities, for the reasons outlined above, relating to techniques that rely on software structure to mine associations. Instead, we use topic modeling techniques to find associations between source files based on variable names, comments, and other information available as plain text. By restricting the words we use in our topic model, this technique is applicable to any source code language. Later, we discuss our approach to pre-processing source code and our approach to language-specific keywords (see Section IV.A). Association rules have also been mined from repository histories in order to find traceability links [22]. As pointed out by David et al., mining the project history has the benefit of reducing the need to rely on the content of the data in instances where it may be sparse or where the content of related artifacts is not related. While we do not rely on the content, we do leverage it where appropriate with topic modeling. It has been pointed out that temporal information can also be useful in eliminating false-positive recommendations [23]. However, this was not applicable to our approach, since we consider both the history of aggregated commits as well as the content of the source files.

D. Limitations

Association mining is useful in finding patterns in the data that satisfy minimum support and minimum confidence constraints. However, some researchers have shown that association mining often results in redundant and unimportant rules. A drawback is that it is difficult to eliminate insignificant rules [24].

In this research, the number of association rules generated depends on the amount of modification history of a project. Also, there is a possibility that not all modules or files may be changed during a software maintenance phase. This can affect the number of rules generated.

E. Related Work

Our work is most closely related to previous work in mining frequently changed files from a software repository [14, 25]. We used association mining as other software engineering researchers have used this technique in the past. We build on top of this existing work and examine the benefits of combining association mining with topic modeling. While others have used collaborative filtering [26], we use topic modeling, which is a probabilistic version of matrix factorization over the word-document matrix. In this paper, we use topic modeling to analyze the semantic content of source code and commit comments. In previous work, we have used topic modeling to identify associations between various software files and architecture components [27]. In the future, we plan to use topic modeling to identify associations between files and authors. Our work is also related to other techniques that seek to identify relationships between software files, such as recommendation systems, code search techniques, and dependency analysis.

Recommendation systems for software engineering may also recommend files for modification. Not all recommendation systems use association rule mining, but eRose a plugin for Eclipse does [8]. The common factor among all recommendation systems for software engineering is that they rely on the user's context in order to provide recommendations. While recommendation systems may help find related files in source code, the issue of user context is outside of the scope of our work.

Code search techniques may also be used to find source files that are related to one another. These techniques have their roots in traditional information retrieval methods [28]. An equivalency study was undertaken to compare various IR methods in the area of traceability recovery [29]. The results of this study showed that while Latent Semantic Indexing (LSI), Jensen-Shannon (JS), and Vector Space Model (VSM) provided higher accuracy in identifying related files, LDA was able to capture associations, which the other methods could not. Recent work in code search has been performed to enhance the accuracy of these methods by allowing the user to specify both the syntactic and semantic properties of a search [28]. Code search techniques, however, fall short in finding relationships between project files, which are not semantically or syntactically related. Meanwhile, our technique finds these relationships based on the change history of the project and semantic relationship.

Dependency analysis tools may be used to find relationships between source files based on call graphs [5]. By making use of the project histories, we can mine relationships between any files that are checked-in together, as opposed to simply analyzing the code structure. As discussed previously, we also have the ability to find relationships between source code files written in different languages. Most importantly, this approach helps to detect cross cutting concerns in which there may be a relationship between two files, but no relationship in a call-graph. For example, a project created for multiple operating systems may contain two source files, which accomplish the same task, but have no relationship in the calling tree. In this case, dependency analysis cannot detect these relationships, but our approach can, because of the semantic similarity between files.

III. Topic Modeling

This section covers background on topic modeling, how we selected the topic modeling technique, application, and limitations.

A. Background

LDA is an unsupervised statistical approach for learning semantic topics from a set of documents [12]. Since it is an unsupervised machine learning technique, no training labels are necessary. This is a fully automated approach that only requires a set of documents and the number of topics to learn

LDA is a generative Bayesian topic model for a corpus of documents. The basic concept behind LDA is that it discovers topics. Then, it associates a set of words with each topic. Lastly, it defines each document as a probabilistic mixture of these topics. Thus, each document can belong to multiple topics. Additional details regarding LDA's generative process are in [12].

Here are some concepts used in LDA:

- A word is a basic unit of discrete data.
- A document is characterized by a vector of word counts.
- A corpus has a total of W words in its vocabulary.
- D documents placed side by side, gives W x D matrix of counts.
- A topic is a probability distribution over W words.
- Each document is associated with a probability distribution over T topics.

To obtain a semantic interpretation of a topic, we simply examine the highest-probability words in that topic. For example, if a topic has high probability words "window", "dialog", "height", "width", "button", we can infer the topic to be related to the user interface of the software.

As we discuss in the next section, we use LDA to determine possible relationships between source code files through their topic distributions. Each source code file equates to a document in LDA.

B. Selection of Topic Modeling Technique

Topic modeling algorithms generally fall under two categories: sampling-based and variational methods [12]. Sampling-based algorithms collect samples to approximate the posterior with an empirical distribution. Variational methods, meanwhile, use a parameterized family of distributions and then find the member of the family that is closest to the posterior. In this paper, we use a fast version of Collapsed Variational Inference (CVB0) for LDA [30], which has been shown to be among the fastest and most accurate methods for learning topic models.

C. Application

Topic modeling has generally been used to analyze unstructured text [31]. In software engineering, topic modeling has been used to relate code topics to authors [32], to enhance the prospective capture of traceability links [27], to derive coupling metrics between classes [33], to find duplicate bug reports [34], and to analyze code fragmentation on the Android framework [35, 36]. Other studies have also shown that LDA can capture associations between software files that are not captured by other IR techniques [37, 29]. In this paper, we use topic modeling to obtain additional file associations to those that can be acquired from mining a project history.

D. Limitations

LDA has generally been applied to unstructured text [31]. Meanwhile, source code is a highly structured text that has a limited range of semantic concepts. The results are also subject to parameters used in LDA. As a result, researchers have examined ways to fine-tune the parameters [36].

We processed the source code prior to running LDA such that reserved words are removed and only semantically meaningful words are used. Our pre-processing technique is similar to the pre-processing technique described here [38].

IV. Combined Approach

Our technique, FP-LDA, aims to lower the dependency of the result on the project history and to provide an alternative means of uncovering related files. FP-LDA consists of the following steps: (1) data extraction and pre-processing, (2) association data mining, (3) topic modeling, and (4) result querying. Figure 1 shows a high level process

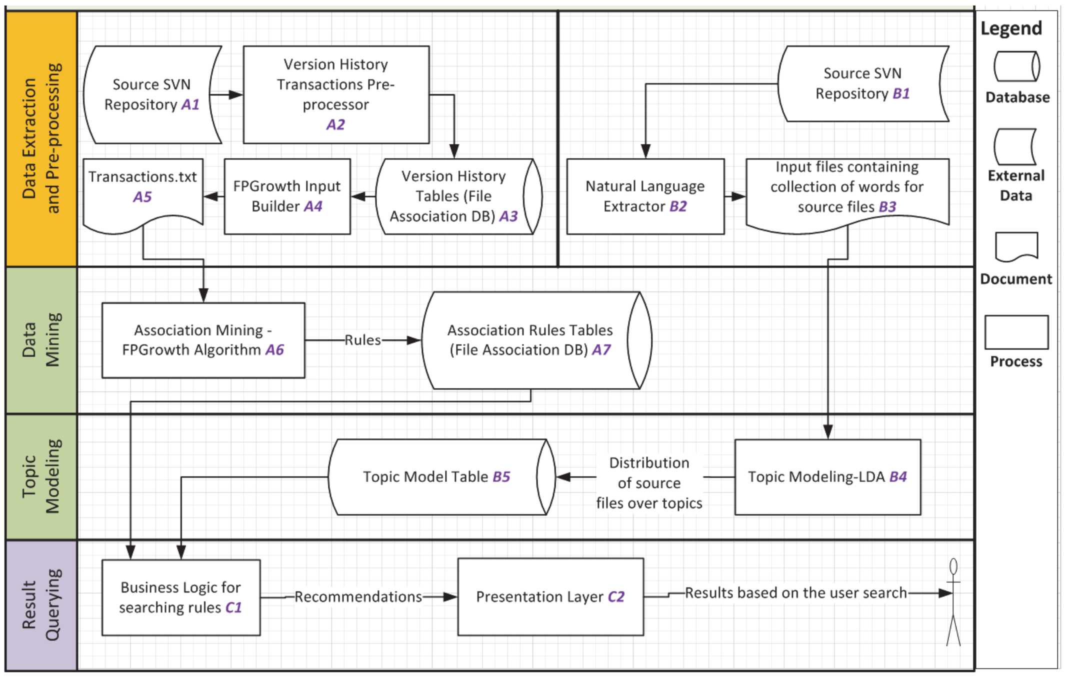

Data Extraction and Pre-processing
Data Mining
Topic Modeling
Result Querying
Source SVN Repository A1
Version History Transactions Pre-processor A2
Transactions.txt A5
FPGrowth Input Builder A4
Version History Tables (File Association DB) A3
Source SVN Repository B1
Natural Language Extractor B2
Input files containing collection of words for source files B3
Association Mining - FPGrowth Algorithm A6
Rules
Association Rules Tables (File Association DB) A7
Topic Model Table B5
Distribution of source files over topics
Topic Modeling-LDA B4
Business Logic for searching rules C1
Recommendations
Presentation Layer C2
Results based on the user search
Legend
Database
External Data
Document
Process

Figure 1. FP-LDA data flow to find file dependencies.

of our technique. Each layer in Figure 1 corresponds to each of these steps. All the processes represented by a rectangle have been implemented.

A. Data Extraction and Pre-processing

Pre-processing version history for data mining. This first step involves extracting the version history of an open source project and preparing the data to be fed as input to the mining algorithm (Step A2). We created a tool that accesses the version history of the project, processes it, and stores the history data in MySQL database. For projects using Subversion (SVN), we used SVNKit application programming interfaces (APIs) [39] to access the version history of the project. SVNKit is an open source Java-based SVN library. For projects using Git, we used JavaGit [40] API to access the version history.

Data pre-processing is an important step in that it removes all unwanted data that may impact data mining (A2). In our technique, our goal is to create a generic pre-processing step to support different open source projects. Thus, we used the following conditions when determining the type of transactions to include in our association mining. Similar to [14], we do not include transactions with more than one hundred files since these transactions may contribute to noise. Such commits may be due to specialized tasks, such as formatting all source code files and then checking-in all files together. We also removed transactions that do not assist in identifying relationships between files, such as single file commits, non-source code commits (e.g., graphic files), and commits of deleted files. The remaining valid transactions are then stored in a database (A3). This is the dataset that will be analyzed by the mining algorithm. We then transform this dataset into a file format that conforms to expected format of the mining algorithm (A4).

Pre-processing source files for LDA. While the mining algorithm examines the entire commit history, we use topic modeling to extract topics from the latest version of the source code. We extracted from the source code semantically meaningful text, such as comments, identifier names and string literals. These words provide clues on the purpose or functionality of the code (B2).

To extract these words, we run each file through a tokenizer. The tokenizer aids in splitting words with underscore or in camel case to obtain the name of objects or variables. We also specified a set of stop words that are programing language-reserved words, and high frequency terms in a software project (see Lessons Learned in Section VI for a detailed discussion). We also removed words like “get” and “set” since source files contain methods that start with these words. This requires some knowledge of the programming language syntax. Another option is to generate the Abstract Syntax Tree using tools like ANTLR [41] to support multiple languages. The generated tree can then be explored to extract the comments and identifiers inside the source code.

B. Association Data Mining

Once the data is preprocessed, we run the data mining algorithm (A6). We used Frequent Pattern Growth (FP-Growth) algorithm for association mining, more specifically, the Liverpool University Computer Science – Knowledge Discovery in Data (LUCS-KDD) implementation of FP-Growth. This Java implementation uses tree structures for association mining [42]. This algorithm requires an input file for the transactions to be analyzed. Each line in the input

Table I. Open Source Project Characteristics.

Project	Repos	Total LOC	Java LOC	No of Files	No of Commits	Years of History
ArchStudio5	Git	838675	194766	2406	596	3.8
ArgoUML	SVN	432521	328494	2254	14922	9.3
EclipseFP	Git	453362	106798	1435	2419	9
Eu_Geclipse	Git	503152	333389	3196	3356	7.5
Lucene	SVN	3893658	1058795	15463	11374	4.5
Thrift	Git	332158	27468	1561	3723	6.3
Xerces	SVN	1266177	260242	2497	5434	14.8

file constitutes one transaction. Each item in the line denotes a file ID. This implementation only works on numeric data. The input file was generated in the previous step. The minimum support and minimum confidence values can be passed as input parameters to the algorithm. The algorithm then generates all the association rules.

In our approach, we did not restrict the frequent itemsets generated to two so that we could uncover more complex relationships. In this case, rules are produced with more than one item in the antecedent and consequent. A file can be related to different files in a different way. If we input only one file, it will give all the recommendations that many not be valid for a certain transaction. However, if the user knows more than one file to be modified for a task, we can refine the predictions. For example, using complex rules we can find out which files change given that two input files are modified together. We store the generated frequent itemsets in a database (A7).

C. Topic Modeling

Once the source files are pre-processed, we extract semantic topics using LDA (B4). We used the CVB0 implementation of LDA [30]. Our implementation of LDA has the following parameters: number of topics and number of iterations. Number of topics is the number of topics we specify. The greater the number of topics, the more fine-grained will be the generated topics. Number of iterations is the number of times the algorithm will run. The higher number of iterations increases the likelihood that the topics will converge. We observed that it is sufficient to run the topic model using 1000 iterations.

D. Result Querying

The last step is to query the results of both the rules generated from association mining and the document relationship to topics (C1). We assume that the user is aware of at least one file that has to be modified for a given modification task. This file is used as the input. The output will show all the files that are recommended or predicted to change along with the input file.

V. Validation

In this section, we discuss how we assess our technique. We cover the setup of our experiment,

A. Experiment with Open Source Projects

In order to validate the ability of FP-LDA to identify relevant files to modify, we conducted experiments on open source projects. We compared FP-LDA with our baseline, FP-Growth.

1) Experiment Setup

We conducted an experiment on seven open source projects that use SVN or Git repositories (see Table I). We selected these projects because these are active projects with different lengths of time (ranging from 3.8 years to almost 15 years) and different range of files (ranging from one thousand files to more than fifteen thousand files).

For each project, we used the same set of parameters. For association mining, we used minimum support of 10 and 15 and confidence value of 40. For topic modeling, we used 25, 50, and 100 topics. For all topic model runs, we used 1000 iterations. We also used topic cutoff values of 10%, 25%, 50%, and 75%. The LDA recommendations were calculated by returning files that have a topic distribution percentage higher than the cutoff for a given topic. We assumed that a file with a higher distribution is semantically closer to a given topic. The validation was performed for these four cutoff percentages. For this experiment, we used Java source code for the topic model.

2) Procedure

To measure the effectiveness of our approach, we used precision and recall. Precision measures the conciseness of a recommendations provided by the approach. Recall measures how many relevant recommendations are made by using this approach. We followed the same approach as used by Ying et al [14]. In this case study, we have assumed that developer is aware of at least one file for a given modification task. Therefore, we specified only one file f_s for generating recommendations for a modification task m. As explained

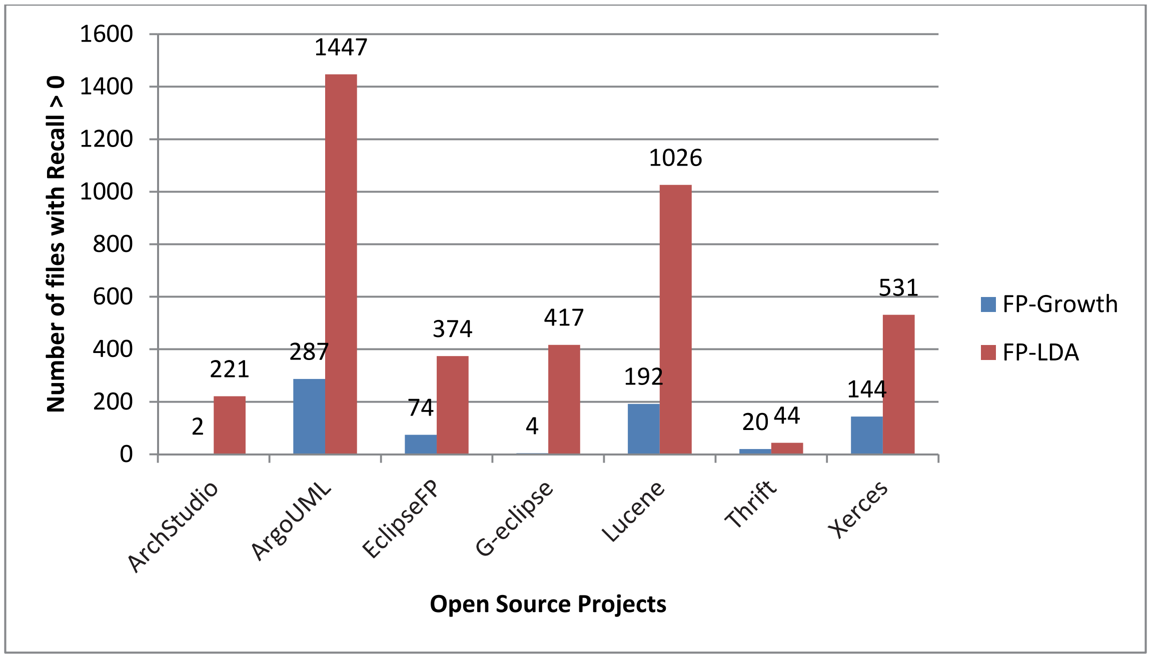


Figure 2. Comparison of recall count between FP-Growth and FP-LDA for the different open source projects.

in [14], the precision $precision(m, f_s)$ of a recommendation $recom(f_s)$ is the fraction of files that are predicted correctly and are part of the solution $f_{sol}(m)$ for the modification task m. The recall $recall(m, f_s)$ of a recommendation $recom(f_s)$ is the fraction of files recommended out of $f_{sol}(m)$.

For example, let us consider a modification task that requires changing files {a, b, c, d}. In addition, let us assume that the recommendations obtained for file b using our approach are files {a, c}. In this case, the precision for file b in this modification task is 100% as the approach recommended correct files. The recall value for file b for same modification task is 66.67% because the approach could predict only two files {a, c} out of {a, c, d}.

In order to determine the effectiveness of our prediction algorithm, we generated FP-Growth rules using 90% of the commit transactions. We then calculated the precision and recall rates of the generated rules on the remaining 10% of the commit transactions (the withheld set). We split the dataset based on time, since this simulates actual practice. Then, we calculated precision and recall for both FP-Growth and FP-LDA for each file in each transaction. We calculated precision and recall for the different parameter combinations. Finally, we counted the number of files in the 10% commit that was able to predict at least one relevant file across the different parameters settings.

3) *Experiment Results*

As Figure 2 shows, FP-LDA consistently improves FP-growth's ability to identify related files. This is best illustrated in the case of the ArchStudio project where only four rules were produced by FP-Growth using minimum support of 10 and confidence of 40. FP-Growth did not produce any rules with minimum support of 15 and confidence of 40. The small number of rules is due to the fact that ArchStudio is a young project that does not have sufficient history to determine file associations. In this case, FP-LDA improves the ability to identify relevant files by two orders of magnitude. The Thrift project, which shows the least number of improvements of 100%, is due to the fact that this project contains the least number of Java source files in comparison to the other projects, only 1/10th of the project's source code. FP-LDA does not only benefit new projects, but older projects like Xerces, which has almost 15 years of history. In this project, we see an almost 300% increase in its ability to find relevant files.

B. *Experiment on Two Open Source Projects*

In order to validate the ability of FP-LDA to rank relevant files to modify, we conducted experiments on two open source projects: ArgoUML and EclipseFP. We compared FP-LDA with two baselines, FP-Growth and LDA.

1) *Experiment Setup*

We selected ArgoUML and EclipseFP projects for implementing ranking.

For ranking the association rules, we used confidence of the rule as a measure to return the recommended files. The higher the confidence, the higher are the chances that predicted files co-occur with the input file in the commit transactions. We selected the top 5 recommendations from FP-Growth. Based on our experience with these projects, the number of association rules generated is not many.

Therefore, by selecting top 5 predictions we covered almost all the predictions that can be provided by FP.

Then for LDA, we used cosine similarity measure to assess the similarity between the source files. The higher value of cosine between two files, the stronger correlation exists between the files. To compare the files using this method we represented each file with total number of frequent words. This is calculated by multiplying the distribution percentage with total number of words in a preprocessed file. We selected the top 10 recommendations from LDA. Next section discusses cosine similarity implementation in detail.

2) Procedure

Though the distribution percentage gives an indication of semantic closeness of a file and topic, the number of recommended files was very high. Therefore, for LDA, we decided to rank the recommended files and return only a limited number of recommendations.

Cosine similarity is often used in text mining applications to assess the similarity between two files [43]. Mathematically, cosine similarity is a measure of how similarity between two vectors and is measured by the cosine of the angle between them.

The cosine similarity between two vectors $\vec{v}$ and $\vec{w}$ of dimension N is calculated

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}.\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v^2{}_i}\sqrt{\sum_{i=1}^{N} w^2{}_i}} \quad (4)$$

The vectors are derived by multiplying the distribution percentage of a file for a topic with total number of words in a preprocessed file. This gives us a total number of frequent words from a topic present in the file. We assume two files are most similar if the cosine value between them is highest. To obtain topic model recommendations, for a given input file, we recommend files in order of highest cosine similarity. We limit the number of recommendations to 15 so that the precision calculated with FP-LDA is not less than the one obtained using just FP-Growth.

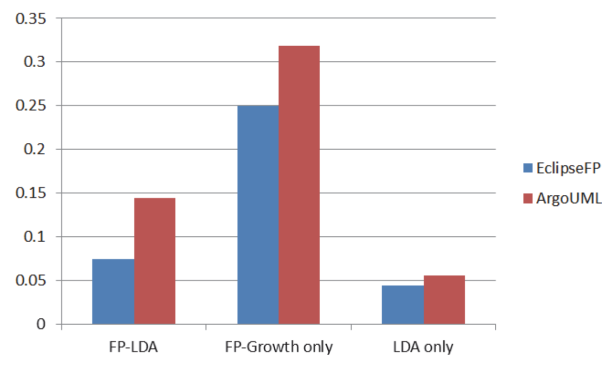


Figure 3. Average precision for precision at 15.

In this setup, we calculated precision and recall for the transactions where FP-Growth resulted in at least one prediction. We also calculated the number of files for all the transactions in test dataset that has at least one correct recommendation.

3) Results

Figure 3 shows the average precision obtained using FP-Growth, FP-LDA and LDA for ArgoUML and EclipseFP. We return at most 15 recommendations in this setup. A low value of precision for FP-LDA may be a result of the fact that most transactions do not consist of 15 files. However, we can see from Figure 4 that the total number of files for the transaction dataset that have at least one correct prediction is greater with FP-LDA than with FP-Growth alone. This means there are more recommendations obtained by using FP-LDA.

C. Examples

In addition to comparing the number of files in the withheld set, which resulted in at least one relevant recommendation, we also examined specific transactions to compare FP-LDA with our baseline. We selected one example that best illustrates each case.

Case 1: FP-Growth recall > 0 and FP-LDA recall > FP-Growth recall: In the Lucene project, we see an example of this in revision ID 1580463 (see Table II). In this example, FP-Growth was run with minimum support of 10 and a minimum confidence of 40%. The topic distribution cutoff was kept at 75%. Figure 5 shows the files committed as part of this transaction. We used Overseer.java as an input file for the techniques.

The association mining alone is able to predict two out of these eight files resulting in a recall value of 0.29. Figure 5 shows the predictions obtained using FP-Growth.

With combined FP-LDA, there are four correct recommendations resulting in a higher recall value of value of 0.6. The precision (0.2) of FP-LDA is low because there were 24 files predicted. The ranking approach we used in Section V.B allows recommendations to be returned that are semantically closer to the topic first and limits the total recommendations. Figure 5 also shows the recommendations from using FP-LDA.

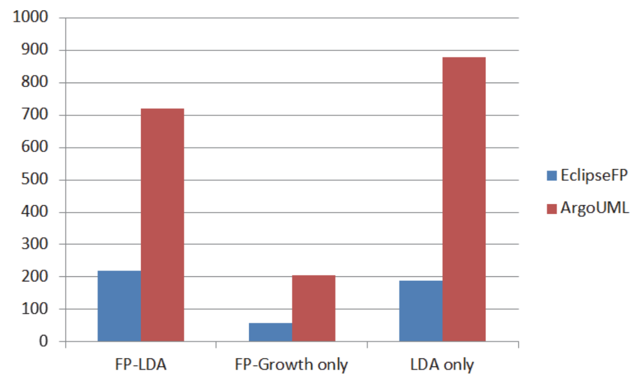
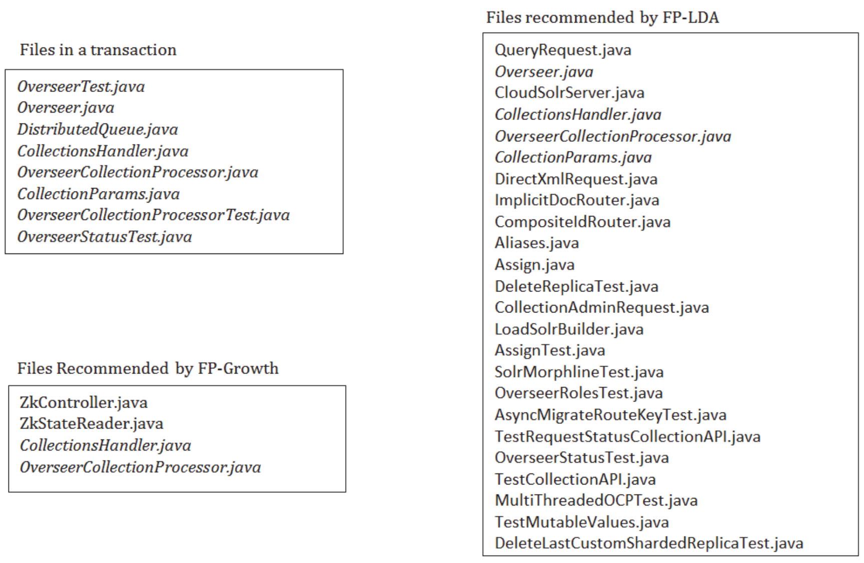


Figure 4. Number of files per transaction in the withheld transaction dataset that has at least one correct recommendation.



Files in a transaction

OverseerTest.java
Overseer.java
DistributedQueue.java
CollectionsHandler.java
OverseerCollectionProcessor.java
CollectionParams.java
OverseerCollectionProcessorTest.java
OverseerStatusTest.java

Files Recommended by FP-Growth

ZkController.java
ZkStateReader.java
CollectionsHandler.java
OverseerCollectionProcessor.java

Files recommended by FP-LDA

QueryRequest.java
Overseer.java
CloudSolrServer.java
CollectionsHandler.java
OverseerCollectionProcessor.java
CollectionParams.java
DirectXmlRequest.java
ImplicitDocRouter.java
CompositeIdRouter.java
Aliases.java
Assign.java
DeleteReplicaTest.java
CollectionAdminRequest.java
LoadSolrBuilder.java
AssignTest.java
SolrMorphlineTest.java
OverseerRolesTest.java
AsyncMigrateRouteKeyTest.java
TestRequestStatusCollectionAPI.java
OverseerStatusTest.java
TestCollectionAPI.java
MultiThreadedOCPTest.java
TestMutableValues.java
DeleteLastCustomShardedReplicaTest.java

Figure 5. Example for Case 1: Recall for FP-Growth > 0 and FP-LDA recall > FP-Growth recall.

Case 2: FP-Growth recall =0, FP-LDA recall> 0: This study was done on ArgoUML revision ID 19876. Figure 6 shows the files that are checked in as part of this revision ID.

The minimum support and confidence values used for FP are 10% and 40%, respectively. The percentage cutoff distribution is 75%.

Using FP-Growth technique gives no recommendations therefore recall is 0. FP-LDA provides 11 recommendations. Four recommendations are correct for this transaction giving a recall of 0.5 and precision of 0.4. Figure 6 also shows the files recommended by FP-LDA.

D. *Discussion*

The calculation of precision and recall gives a general understanding of how the approach fares in finding relationships. We assumed that each of these transactions was a task presented to a developer. For each file in the test transaction, we calculated precision and recall values to see if the tool can predict the remaining files.

We have shown in the example for Case 2 a case where FP-Growth is not able to find file relationships, but FP-LDA overcomes this shortcoming. Because the number of total recommendations increases with topic modeling, the precision has a tendency to decrease, as shown in the example. However, FP-LDA does achieve the same precision rates for the same recall as FP-Growth alone. A higher recall value shows that there is an increase in the number of relevant files predicted. This indicates that number of correct recommendations increases with LDA. The utility of the approach lies in the fact that a developer needs to search only the set of recommended files, and not the entire source code base. Moreover, since we solely base our precision and recall on actual check-in records in the latter 10% of the history record, it is entirely possible that two files are related, but they may not have been checked-in together within this subset of the data, within the same transaction.

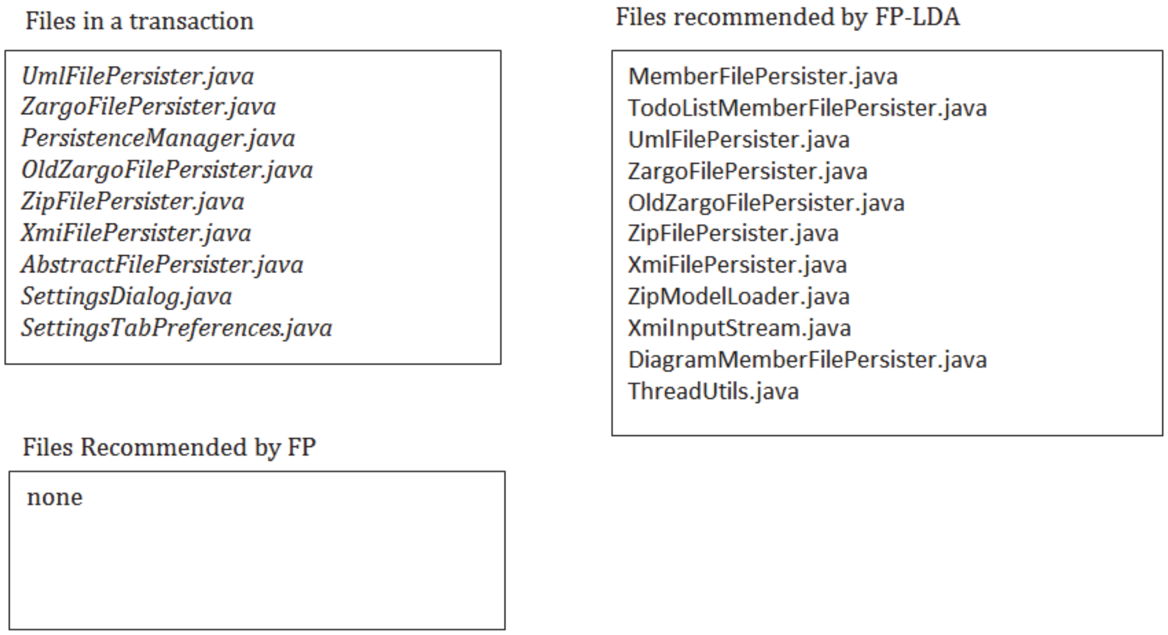
We have also shown in the experiment on two open source projects that FP-LDA is able to overcome the limitations of both FP-Growth and LDA only. FP-Growth can provide high precision rates but can recommend a very small number of files. Meanwhile, LDA only can recommend large number of files but with much lower precision. FP-LDA achieves a better balance between precision and the number of correct recommendations.

E. *Limitations of the study*

Our precision and recall numbers may be subject to the specific datasets we selected. However, since we selected projects with different characteristics and we observe the same trend across the different projects, this indicates that our results are applicable to other open source projects.

The number of topics used in LDA may also affect precision and recall rates. We ran our technique using different topic numbers and observed that the smaller the topic number, the higher the recall rates and the lower precision rates are generated. 50 and 100 topics are generally used by machine learning researchers. We added 25 topics to provide us a wider range of precision and recall values to examine. Also LDA may not always generate significant topics. The quality of generated topics has to be manually evaluated.

It is possible that our baseline, FP-Growth, could have produced more rules if we provided lower minimum support and confidence values. Choosing minimum support and minimum confidence for association mining is critical. A lower minimum support may yield more rules but not necessarily meaningful rules. Choosing the optimum value depends on the kind of dataset used. We decided the minimum support and confidence values based on our experience with analyzing projects and size of the version


Files in a transaction

UmlFilePersister.java
ZargoFilePersister.java
PersistenceManager.java
OldZargoFilePersister.java
ZipFilePersister.java
XmiFilePersister.java
AbstractFilePersister.java
SettingsDialog.java
SettingsTabPreferences.java

Files recommended by FP-LDA

MemberFilePersister.java
TodoListMemberFilePersister.java
UmlFilePersister.java
ZargoFilePersister.java
OldZargoFilePersister.java
ZipFilePersister.java
XmiFilePersister.java
ZipModelLoader.java
XmiInputStream.java
DiagramMemberFilePersister.java
ThreadUtils.java

Files Recommended by FP

none

Figure 6. Example for Case 2: Recall for FP-Growth = 0 and FP-LDA recall > 0.

history. However, even if more rules were produced, based on the examples shown, it is not possible for FP-Growth to predict certain files because the files modified in the most recent 10% of the history are not necessarily the same as the files modified in the first 90% of the history.

VI. LESSONS LEARNED

Lesson 1: Stop word selection. We observed that choice of the stop words affects the quality of topic model in a profound way. The words that need to be excluded from analysis depend largely on the use. For example, in this study we focused on finding the relationship between files. We do not want to know author file relationship. If we did not remove the author names from the processed source code files, we would get a topic with author names in it. In addition, the words we extracted for topic modeling represented different levels of abstraction. Thus, the highest level concepts (i.e., project-wide concepts) should also be added to the list of stop words. Since we are analyzing source code, the generally used list of stop words for natural language documents (e.g., a, an, the) are not applicable. At the same time, we wished to use a general approach for determining project-specific concepts that do not contribute to the meaning of each source code file. Thus, we used the following approach in creating our stop words list. We ran the corpus through a three-step process. First, we eliminated all language-specific reserved words. In Java, these include words such as “public”, “class”, “while”. After the language-specific words are eliminated from the corpus, we then analyzed the corpus for the highest frequency words for that project [44]. We took the 10% of the highest frequently occurring words in the corpus and used these as our second set of stop words. Lastly, we examined the generated topics manually and removed any more words that does not contribute to the meaning of the topics (e.g., copyright info).

Lesson 2: Aggregating commits. In our previous work [1], we considered each atomic commit as one transaction. This time, we logically grouped transactions to obtain more meaningful changed sets. These heuristics are time interval and author. We assumed that the commits within a time interval by the same author are related to each other. After examining certain transactions we decided the time interval to be one hour. However, using a fixed time interval may not be generalizable across different projects. In the future, we plan to determine how to create a generalizable heuristics for aggregating commits to get more relevant results.

VII. CONCLUSION AND FUTURE WORK

In this paper, we used association mining and topic modeling together to assist developers in software maintenance task. These techniques were used to uncover the source file dependencies within a software project. We applied association mining on version history of a project to find files that frequently change together. We complemented this technique by using topic modeling on the source code documents. We showed that using topic modeling could uncover file dependencies that are not captured due to lack of version history for those files. Our evaluation indicates that this combination of techniques increases the number of relevant files obtained by at least a 100%, based on the seven open source projects we analyzed.

In the future, we would like to explore various options that can measure the usefulness of this approach. We plan to analyze more open source projects as well as conduct user studies to determine whether our approach reduces the time required for impact analysis or any maintenance task. In

Table II. Summary of precision and recall for the two cases examined.

Project	FP-Growth		FP-LDA		Topic Modeling Files Recommended
	Precision	Recall	Precision	Recall	
Case 1: FP-Growth has no recommendations while FP-LDA gives recommendations					
Argo UML	0	0	0.4	0.5	Revision id = 19876: 11 with 4/8 correct
Lucene	0.5	0.3	0.2	0.6	Revision Id = 1580463 24 with 4/8 correct
Case 2: FP-LDA has higher number of correct recommendations than FP-Growth					
Lucene	0.5	0.3	0.5	1	Revision Id = 1610028 9 with 0 correct

addition, we can use an approach to automatically rank the LDA topics based on their semantic importance to eliminate insignificant topics [45].

ACKNOWLEDGMENT

We thank Arthur U. Asuncion for his insights on LDA and providing the CVB0 implementation of LDA. We also thank Eamon Maguire for his assistance in extracting version histories and running the mining algorithm. We thank Delmar Davis with his assistance in running evaluations. We also thank Subha Vasudevan for her contributions, including analyzing generated topics and preparing the stop words for effective topic modeling. This material is based upon work supported by the National Science Foundation under Grant No. CCF-1218266. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] N. Dave, D. Davis, K. Potts, and H. U. Asuncion, "Uncovering file relationships using association mining and topic modeling," in *The Sixth International Conference on Information, Process, and Knowledge Management*, pp. 105–111, Mar 2014.

[2] S. S. Yau and J. S. Collofello, "Some stability measures for software maintenance," *Trans. on Software Engineering*, vol. SE-6, pp. 545–552, Nov. 1980. doi:10.1109/TSE.1980.234503.

[3] A. J. Ko, B. A. Myers, M. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *TSE*, vol. 32, pp. 971–987, Dec 2006. doi:10.1109/TSE.2006.116.

[4] R. N. Taylor, N. Medvidovic, and E. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

[5] M. Sharp and A. Rountev, "Static analysis of object references in RMI-based Java software," in *Proc of the Int'l Conference on Software Maintenance*, pp. 101–110, Sep. 2005. doi:10.1109/ICSM.2005.84.

[6] M. Eaddy, A. V. Aho, G. Antoniol, and Y. G. Gueheneuc, "Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis," in *Proc of the 16th Int'l Conference on Program Comprehension*, pp. 53–62, Jun. 2008. doi:10.1109/ICPC.2008.39.

[7] D. Cubranic, G. C. Murphy, J. Singer, and S. Booth Kellogg, "Hipikat: a project memory for software development," *Trans. on Software Engineering*, vol. 31, pp. 446–465, Jun. 2005. doi:10.1109/TSE.2005.71.

[8] M. P. Robillard, R. J. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, pp. 80–86, Jul-Aug. 2010. doi:10.1109/MS.2009.161.

[9] S. Bajracharya, J. Ossher, and C. V. Lopes, "Sourcerer - an infrastructure for large-scale collection and analysis of open-source code," *Science of Computer Programming*, vol. 79, pp. 241–259, Jan. 2014. doi:10.1016/j.scico.2012.04.008.

[10] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc of 20th Int'l Conference on Very Large Data Bases*, pp. 487–499, Sep. 1994.

[11] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc of the 2000 Int'l Conference on Mgmt of Data*, pp. 1–12, May 2000. doi:10.1145/342009.335372.

[12] D. M. Blei, "Probabilistic topic models," *Comunications of the ACM*, vol. 55, pp. 77–84, Apr 2012. doi:10.1145/2133806.2133826.

[13] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, pp. 207–216, Jun. 1993. doi:10.1145/170036.170072.

[14] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *Trans. on Software Engineering*, vol. 30, pp. 574–586, Sep. 2004. doi:10.1109/TSE.2004.52.

[15] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," in *Proc of the Fifth ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pp. 337–341, Aug. 1999. doi:10.1145/312129.312274.

[16] K. Lai and N. Cerpa, "Support vs confidence in association rule algorithms." http://www.researchgate.net/publication/233754781_Support_vs_Confidence_in_Association_Rule_Algorithms/file/9fcfd512a4907b8aca.pdf, 2001.

[17] B. Liu, W. Hsu, S. Chen, and Y. Ma, "Analyzing the subjective interestingness of association rules," *Intelligent Systems and their Applications*, vol. 15, pp. 47–55, 2000. doi:10.1109/5254.889106.

[18] J. Pei and et al., "Mining sequential patterns by pattern-growth: the PrefixSpan approach," *Trans. on Knowledge and Data Engineering*, vol. 16, pp. 1424–1440, Nov. 2004. doi:10.1109/TKDE.2004.77.

[19] C. Tjortjis, L. Sinos, and P. Layzell, "Facilitating program comprehension by mining association rules from source code," in *Proc of the International Workshop on Program Comprehension*, pp. 125–132, 2003. doi:10.1109/WPC.2003.1199196.

[20] T. Xie and J. Pei, “MAPO: mining API usages from open source repositories,” in *Proc of the 2006 Int’l Workshop on Mining Software Repositories*, pp. pages 54–57, 2006. doi:10.1145/1137983.1137997.

[21] D. Rousidis and C. Tjortjis, “Clustering data retrieved from java source code to support software maintenance: A case study,” in *Proc of the Ninth European Conference on Software Maintenance and Reengineering*, pp. 276–279, 2005. doi:10.1109/CSMR.2005.16.

[22] J. David, M. Koegel, H. Naughton, and J. Helming, “Traceability ReARMed,” in *Proc of the International Computer Software and Applications Conference*, pp. 340–348, 2009. doi:10.1109/COMPSAC.2009.52.

[23] H. Kagdi, S. Yusuf, and J. I. Maletic, “Mining sequences of changed-files from version histories,” in *Proc of the International Workshop on Mining Software Repositories*, pp. 47–53, 2006. doi:10.1145/1137983.1137996.

[24] B. Liu, W. Hsu, and Y. Ma, “Identifying non-actionable association rules,” in *Proc of Int’l Conference on Knowledge Discovery and Data Mining*, pp. 329–334, Aug. 2001. doi:10.1145/502512.502560.

[25] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, “Mining version histories to guide software changes,” *Trans. on Software Engineering*, vol. 31, pp. 429–445, Jun. 2005. doi:10.1109/TSE.2005.72.

[26] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proc of the 10th International Conference on World Wide Web*, pp. 285–295, May 2001. doi:10.1145/371920.372071.

[27] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, “Software traceability with topic modeling,” in *Proc of the Int’l Conference on Software Engineering*, vol. 1, pp. 95–104, May 2010. doi:10.1145/1806799.1806817.

[28] S. P. Reiss, “Semantics-based code search,” in *Proc of the Int’l Conference on Software Engineering*, pp. 243–253, May 2009. doi:10.1109/ICSE.2009.5070525.

[29] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, “On the equivalence of information retrieval methods for automated traceability link recovery,” in *Proc of the 16th Int’l Conference on Program Comprehension*, pp. 68–71, Jun-Jul. 2010. doi:10.1109/ICPC.2010.20.

[30] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, “On smoothing and inference for topic models,” in *Proc of Conference on Uncertainty in Artificial Intelligence*, pp. 27–34, Jun. 2009.

[31] B. Gretarsson and et al., “TopicNets: Visual analysis of large text corpora with topic modeling,” *Trans. on Intelligent Systems and Technology*, vol. 3, pp. 23:1–23:26, Feb. 2012. doi:10.1145/2089094.2089099.

[32] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, “Mining Eclipse developer contributions via author-topic models,” in *Proc of the Fourth International Workshop on Mining Software Repositories*, p. 30, 2007. doi:10.1109/MSR.2007.20.

[33] M. Gethers and D. Poshyvanyk, “Using relational topic models to capture coupling among classes in object-oriented software systems,” in *Proc of the International Conference on Software Maintenance*, pp. 1–10, 2010. doi:10.1109/ICSM.2010.5609687.

[34] A. Nguyen, T. T. Nguyen, H. Nguyen, and T. N. Nguyen, “Multi-layered approach for recovering links between bug reports and fixes,” in *Proc of the 20th International Symposium on the Foundations of Software Engineering*, p. 63, 2012.

[35] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, “Understanding Android fragmentation with topic analysis of vendor-specific bugs,” in *Proc of the Working Conference on Reverse Engineering*, pp. 83–92, 2012. doi:10.1109/WCRE.2012.18.

[36] A. Panichella and et al., “How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms,” in *Proc of the Int’l Conference on Software Engineering*, pp. 522–531, May 2013. doi:10.1109/ICSE.2013.6606598.

[37] A. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, “Duplicate bug report detection with a combination of information retrieval and topic modeling,” in *Proc. of the International Conference on Automated Software Engineering*, pp. 70–79, 2012. doi:10.1145/2351676.2351687.

[38] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk, “TopicXP: Exploring topics in source code using Latent Dirichlet Allocation,” in *Proc of the Int’l Conference on Software Maintenance*, pp. 1–6, Sep. 2010. doi:10.1109/ICSM.2010.5609654.

[39] TMate Software, “SVNKit.” http://svnkit.com/. 2014.12.17.

[40] “JavaGit.” http://javagit.sourceforge.net/. 2014.12.17.

[41] “ANTLR.” http://www.antlr.org/. 2014.12.17.

[42] F. Coenen, G. Goulbourne, and P. Leng, “Tree structures for mining association rules,” *Data Mining and Knowledge Discovery*, vol. 8, pp. 25–51, Jan. 2004. doi:10.1023/B:DAMI.0000005257.93780.3b.

[43] B. Li and L. Han, “Distance weighted cosine similarity measure for text classification,” in *Intelligent Data Engineering and Automated Learning* (H. Yin, K. Tang, Y. Gao, F. Klawonn, M. Lee, T. Weise, B. Li, and X. Yao, eds.), vol. 8206 of *Lecture Notes in Computer Science*, pp. 611–618, Springer Berlin Heidelberg, 2013.

[44] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[45] L. Alsumait, D. Barbará, J. Gentle, and C. Domeniconi, “Topic significance ranking of LDA generative models,” in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, pp. 67–82, 2009.