ITMViz: Interactive Topic Modeling for Source Code Analysis

Amir M. Saeidi, Jurriaan Hage, Ravi Khadka, Slinger Jansen
Department of Information and Computing Sciences, Utrecht University
{a.m.saeidi, j.hage, r.khadka, slinger.jansen}@uu.nl

***Abstract*—Topic modeling has seen a surge in use for software comprehension. Although the models inferred from the source code are a great source of knowledge, they fail to fully capture the conceptual relationships between the topics. Here we investigate the use of interactive topic modeling for source code analysis by feeding-in information from the end-users, including developers and architects, to refine the inferred topic models. We have implemented a web-based toolkit called *ITMViz* to provide support to interpret the topic models, and use the results to cluster modules together. A medium-sized Java project is used to evaluate our approach in understanding the software system.**

I. Introduction

As software evolves, the structure of the system becomes more complex and hence more difficult to understand. The heterogeneity of programming languages and technologies used in the application makes it even harder to develop tools for deep analysis. These issues give rise to a necessity to develop tools that can be easily extended to accommodate for heterogeneous software systems and help get a high-level overview of the underlying structure of the system. Information retrieval techniques are one such method adopted for source code analysis in order to identify the latent relationships between software modules based on a common use of vocabulary, and to derive a set of themes (topics) that run through the corpus. These techniques have shown to give results that are helpful in different phases of the maintenance lifecycle [1], [2].

The topics discovered by topic modeling techniques do not always make sense [3]. The problem stems from the underlying assumptions about the nature of relationship between terms in the corpus, and whether such heuristics conform with human judgements of what terms should constitue a topic. The problem is exacerbated when dealing with a source code corpus for program understanding [4]. For example, the naming convention used in the corpus may be arbitrary and contain noise[1]. Furthermore, the bag-of-words approach loses all the structure of the source code, and hence the context of data items that comes with it. To overcome the aforementioned problems, there has been an effort to feed-in information by a human expert about the orientation of topics to infer more reasonable topics.

A common topic modeling approach used for source code analysis is Latent Dirichlet Allocation (LDA) [5], which identifies the topics that permeate a corpus of text documents. Standard LDA lacks the capability to encode such information, so to help encode domain knowledge in the LDA framework, Andrzjeweski et al. [6] propose to use Dirichlet tree priors to incorporate such information. The domain knowledge is encoded in terms of two primitives: 1) Must-Link constraints, the set of words that should belong to the same topic, and 2) Cannot-Link constraints, pairs of words that should not appear together in the same topic. Hu et al. [3] extends this approach to fully capture the dynamic nature of interactive accomodation of constraints in the LDA.

We adopt this approach [3] to build an interactive environment for source code analysis called *ITMViz* for interactive topic modeling (ITM) as well as visualization, analysis and interpretation of inferred topic models. There exist other LDA-based tools such as *TopicXP* [7] and *LDA Analyzer* [8] for exploring topics in the source code; the notable difference with *ITMViz* is the element of user-supervision. The interactive topic modeling allows users (e.g. architects and developers) to iteratively refine the topics discovered by imposing constraints on the terms. Our approach is evaluated using a medium-sized Java open source project to 1) examine the alignment of discovered topics by unconstrained LDA with that of the inherent structure of the system, and 2) investigate whether the introduction of constraints helps to better capture the high-level structure of the system. In Section II, we outline the high-level architecture of *ITMViz*, and discuss each component in detail. We proceed by evaluating the toolkit in Section III. In Section IV, we conclude and outline future work.

II. ITMViz Toolset

The *ITMViz* toolkit[2] is a web-based application developed as part of the Gelato toolset [9] for facilitating program comprehension and transformation of legacy software systems. The infrastructure of the application is developed using the Shiny package[3] in R combined with the D3 visualization library[4]. For performance reasons, the collapsed Gibbs sampling for posterior inference is implemented in C++. The architecture of *ITMViz* consists of three components: 1) Data Preprocessing, 2) Interactive Topic Modeling and 3) Visualization and Interpretation. Figure 1 depicts the interactive topic modeling process for source code analysis. Initially, the

[1]Terms used to implement a functionality (such as built-in or third-party library functions) that do not reflect domain and application concepts

[2]The screencast is available here: http://youtu.be/Is4ywW5oiUI

[3]http://shiny.rstudio.com/

[4]http://d3js.org/

source code corpus is scanned to build a bag-of-words. The resulting bag-of-words is fed into the topic modeler to build the posterior distribution of terms over topics. The output can be visualized to provide a global view of topics as well as how source code units are related through the discovered topics. After inspecting the results, the user may engage in refining the topics by providing constraints as well as tuning the parameters. This refinement is repeated until satisfactory results are obtained.

A. Data Preprocessing

The topic models' vision of the source code corpus is a 'bag-of-words', representing the set of source code units and their term occurrences. Constructing the bag-of-words involves applying a preprocessor to extract vocabulary terms from comments, identifier names and literals in the source code, followed by breaking up composite terms built using a standard way such as CamelCase convention. It proceeds by eliminating common terms that occur in a natural language (for instance, "the" and "is"), as well as keywords and reserved words of the programming language (such as "extends" and "implements" in Java). The resulting set of terms is normalized by applying a stemmer to emit a common radix. Each resulting document-term pair is normalized by the *term frequency-inverse document frequency* weighing mechanism to measure how prevalent terms are throughout the corpus and those terms that are lower than a user-defined threshold are eliminated from the produced bag-of-words[5].

B. Interactive Topic Modeling

Topic modeling discovers a set of distributions over terms for each topic and the association of topics with each document, performed through a posterior inference. Each topic is a multinomial distribution over terms, explaining how terms are associated with each topic. A standard procedure used for inferring the posterior distributions in LDA is collapsed Gibbs sampling [10].

To perform the posterior inference, LDA contains serveral hyper-parameters that needs to be set: 1) tc representing the number of topics in the corpus, and 2) α and β specifying Dirichlet priors about the uniformity and sparsity of topics over documents. The collapsed Gibbs sampler also has to be configured with a number of parameters including the number of *burn-in* iterations, the number of samples, and the sampling interval. These parameters have a significant impact on the resulting topic model in LDA and are subject of studies [11], [12] to help researchers configure LDA when used in the context of source code analysis. Interactive topic modeling has another parameter η which can be used to control the strength of domain knowledge on the inferred topic models, allowing for overriding the user-specified constraints if the underlying data strongly suggests otherwise. Albeit there is no rule of thumb to determine what the good choices for the parameters are, the visualization provides an effective approach to inspect the results and re-adjust the parameteres, if deemed necessary.

The feedback from a human expert is encoded via a set of 'Must-Link' and 'Cannot-Link' constraints. A Must-Link constraint is a set of terms whose probability should be correlated in a topic. Must-link contraints are transitive and are propagated to obtain the maximal set of words that should appear together. On the other hand, Cannot-Link constraints define pairs of terms whose probability should be uncorrelated across the topics. Before producing the Dirichlet tree priors, the constraints are checked to ensure no conflicts exist; otherwise the user is notified of the existence of conflicts and can revise the constraints. Enforcing constraints on the topic modeling changes the dynamics of how source code units are related to each other.

C. Visualization and Interpretation

The visualization consists of two parts, one for the global view of topics and topic-term relationships, and one to give a perspective on document-topic relationships and prevalent topics in each document. The visualization facilities in *ITMViz* are borrowed from *LDAviz* [13] which allows for a deep inspection of the latent relationships between terms and each individual topic.

III. Empirical Case Study

We have performed topic analysis on a Java open source project called jEdit[6], a text editor for programming. The system consists of 538 classes. We have studied the documentation of jEdit to recover the high-level architecture of the system and the functionalities it provides. This domain knowledge is translated into a set of constraints which are iteratively imposed on the LDA.

The jEdit system provides the following set of functionalities:

- GUI components comprising of text editor, menu, messaging, layout
- Text editing features including syntax highlighting, indentation and tabbing, commenting out code, abbreviations and folding the code
- Working with files
- Input/output for data transfer
- Add-on plugins
- Options for customizing the jEdit environment
- BeanShell module for writing macros and startup scripts

We will use the aforementioned functionalities to extract meaningful topics by guiding the ITM.

A. Data Preprocessing

The jEdit project is implemented in Java and the natural language used to write comments, literals and identifier names is English. The naming convention used, as promoted for good programming in Java, is CamelCase, which we use to break up the composite terms. The total number of words in the bag-of-words is over 2500. To eliminate prevalent terms throughout

[5]One such example is the set of terms specific to the copyright and disclaimer notices in Java source files

[6]http://www.jedit.org/

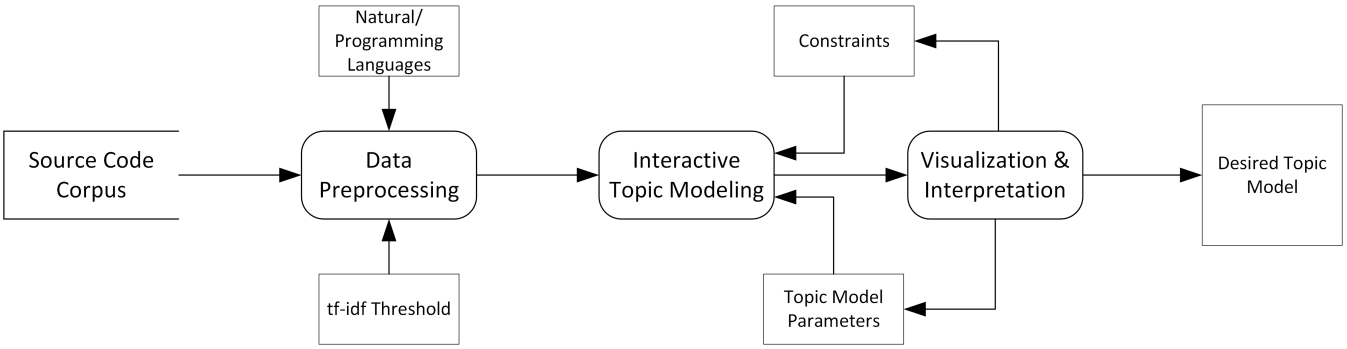

Fig. 1. Interactive Topic Modeling Architecture Overview

the corpus, we impose a minimum threshold of 20 on the *tf-idf* score of words, reducing the total number of words by more than a half to about 1100 words.

B. Interactive Topic Modeling

We have chosen the following configuration to instantiate the ITM framework: $tc = 15$, $\alpha = 0.5$, $\beta = 0.1$, and $\eta = 10000$. The number of burn-in iterations is set to 10, and the number of samples to 200. We run the LDA before estimating the topics from the final sample. Once the topic model is produced, the visualized results are assessed to determine further steps.

C. Visualization and Interpretation

The Visualization and Interpretation facility in *ITMViz* consists of two components: 1) Topic Visualization & Analysis: used for inspection of term-topic relationship in the source code as well as analysis of individual topics followed by associating domain concepts to a topic (*labelling*), and 2) Clustering: used to investigate the association between the topics and documents throughout the system.

1) Topic Visualization & Analysis: Figure 2 depicts the global view of term-topic relationship of the jEdit system on the left, while on the right the term barcharts show the ranking of the terms. The size of each topic is proportional to the relative prevalence of the topics throughout the corpus [13]. The default technique for mapping the topics into two dimensions is Principal Component Analysis; other techniques are also implemented. The ranking of terms is based on the measurement proposed in [13] where a term is ranked based on its *relevance* to a topic. It is possible to adjust the ranking of the terms to allow users to examine the usefulness of terms in the interpretation of topics. Furthermore, there is an option to cluster the topics together based on their scaled two-dimensional locations using the k-means clustering algorithm.

The most prevalent topic in jEdit is Topic 1, which comprises of terms including ‘eval’, ‘interpret’, ‘bean’ and ‘shell‘ corresponding to the BeanShell module for writing macros and scripts in jEdit. On the other hand, the least prevalent topic consists of terms such as ‘comment’, ‘parser’, ‘identifi’ and ‘literal’ which can be interpreted as syntax highlighting in jEdit. Once a decision is made about the meaning of a topic, it can be labelled to reflect its conceptual meaning. This is made possible through ‘Topic Analysis’ where topics are individually inspected and labelled. Labelling topics makes it possible to examine the corpus to see where in the system some concepts are prevalent.

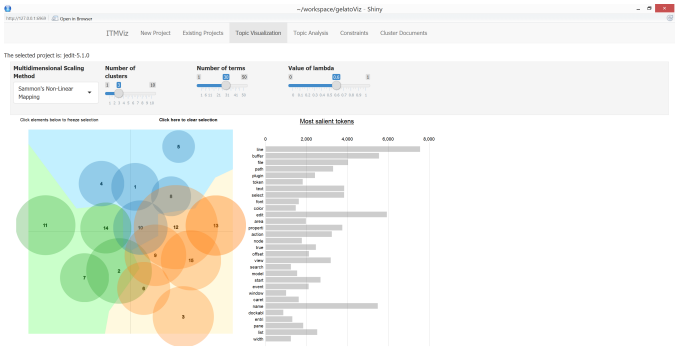

Fig. 2. The term-topic relationship of jEdit system

2) Clustering: The clustering facility provides a global perspective of topic-document relationship which then can be used to perform clustering of source code units. Figure 3 depicts the features that clustering provides to analyze the source modules across the system. Similar to ‘Topic Visualization’, the right panel reveals the most dominant topics in a document while the left panel gives the global overview of the relationship between the source code units based on the established topic-document relationship. Other features match those offered for visualization of topics. The size of the circle is proportional to the number of occurring terms in a source code unit relative to all term occurrences in the corpus.

D. Constraints

We encode the domain knowledge in terms of a set of constraints to manipulate the co-ocurrence of terms across topics. To bring together terms that should belong to the same topic, we introduce Must-Link constraints between them. On the other hand, Cannot-Link constraints are accumulated to pull apart terms across topics that should not come together. Overlapping topics can be split by imposing Cannot-Link constraints while similar topics can be merged by introducing Must-Link constraints on the terms. For instance, after

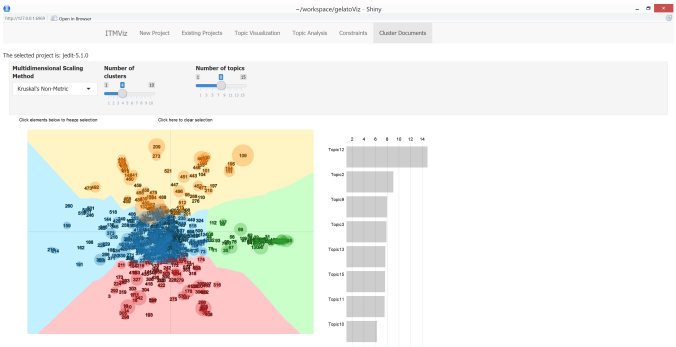

Fig. 3. The topic-document relationship of jEdit system

insepction of results generated from the unconstrained LDA, the terms 'plugin' and 'textarea' are uncorrelated by imposing a Cannot-Link between them. We also brought together terms associated with the editing features of text area under one topic by introducing Must-Link constraints between them.

E. Results & Discussion

To compare the quality of topic models produced from unconstrained LDA and those produced from ITM, we make the comparison based on the inferred topic-document relationship. We decompose the source code units at different stages of interaction and measure the authoritativeness of the produced clusters. The reference decomposition is produced from the package structure of jEdit. During each iteration, new constraints are introduced or removed, based on the results produced from the previous iteration. This process is repeated until the results align with our expectation. The clustering algorithm used here is k-means with the number of clusters correpsonding to the number of packages in the system, i.e., 35 clusters. Figure 4 shows the MoJo similarity measurement obtained for jEdit after 10 iterations. The MoJoSim achieved from unconstrained LDA is around 45%. We have injected domain knowledge stepwise into ITM over 10 iterations resulting in a final MoJoSim of around 55%.

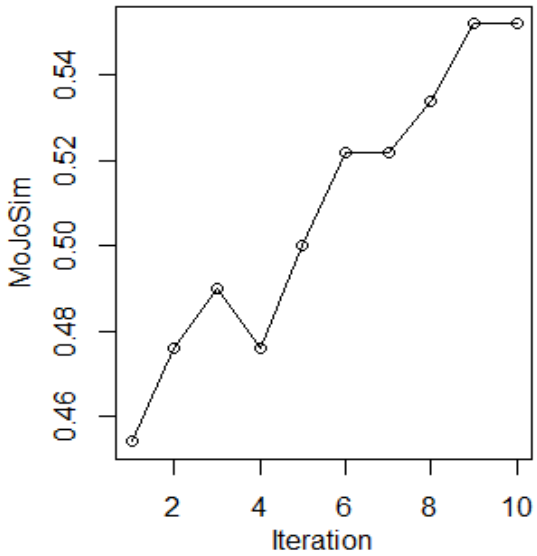

Fig. 4. The MoJoSim measurement for jEdit after 10 iterations

IV. Conclusion and Future Work

We have discussed *ITMViz*, an interactive topic modeling environment that enables users to supervise the topic modeling by enforcing constraints on the set of words in the corpus. The visualization feature of the toolkit allows for deep inspection and analysis of the discovered topic models to help users refine the results further. We present the results from the evaluation of a medium-sized Java open source project by manipulating results based on the domain knowledge extracted from the documentation of the system, and show that indeed this approach helps emit more meaningful topic models. We would like to apply our tooling to legacy systems in the industrial domain to see if supervision of topic modeling by a domain expert can help with the understanding of these systems.

References

[1] S. Grant, J. Cordy, and D. Skillicorn, "Using topic models to support software maintenance," in *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, March 2012, pp. 403–408.

[2] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. d. Lucia, "Improving software modularization via automated analysis of latent topics and dependencies," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 4:1–4:33, Feb. 2014.

[3] Y. Hu, J. Boyd-Graber, B. Satinoff, and A. Smith, "Interactive topic modeling," *Machine Learning*, vol. 95, no. 3, pp. 423–469, 2014.

[4] A. Hindle, C. Bird, T. Zimmermann, and N. Nagappan, "Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers?" in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, Sept 2012, pp. 243–252.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[6] D. Andrzejewski, X. Zhu, and M. Craven, "Incorporating domain knowledge into topic modeling via Dirichlet forest priors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 25–32.

[7] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk, "TopicXP: Exploring topics in source code using latent Dirichlet allocation," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*, Sept 2010, pp. 1–6.

[8] C. Zou and D. Hou, "LDA analyzer: A tool for exploring topic models," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 593–596.

[9] A. Saeidi, J. Hage, R. Khadka, and S. Jansen, "Gelato: GEneric LAnguage TOols for model-driven analysis of legacy software systems," in *20th Working Conference on Reverse Engineering (WCRE)*, Oct 2013, pp. 481–482.

[10] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[11] D. Binkley, D. Heinz, D. Lawrie, and J. Overfelt, "Understanding LDA in source code analysis," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: ACM, 2014, pp. 26–36.

[12] S. Grant and J. Cordy, "Examining the relationship between topic model similarity and software maintenance," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, Feb 2014, pp. 303–307.

[13] C. Sievert and K. Shirley, "LDAvis: A method for visualizing and interpreting topics," in *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*. Baltimore, Maryland, USA: Association for Computational Linguistics, June 2014, pp. 63–70.