Mining Software Repositories Using Topic Models

Stephen W. Thomas
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Kingston, ON, Canada
sthomas@cs.queensu.ca

ABSTRACT

Software repositories, such as source code, email archives, and bug databases, contain unstructured and unlabeled text that is difficult to analyze with traditional techniques. We propose the use of statistical topic models to automatically discover structure in these textual repositories. This discovered structure has the potential to be used in software engineering tasks, such as bug prediction and traceability link recovery. Our research goal is to address the challenges of applying topic models to software repositories.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

General Terms

Experimentation, Measurements

Keywords

topic models, mining software repositories, LDA

1. INTRODUCTION

Understanding and maintaining the source code of a software project is difficult for both developers and managers. The complexity of source code tends to increase as it evolves, leading to bugs and increased maintenance costs.

Recently, the field of software engineering (SE) has attempted to combat this problem by mining the repositories related to a software project, such as source code changes, email archives, bug databases, and execution logs [7, 10]. Research shows that interesting and practical results can be obtained from these repositories, allowing developers and managers to better understand their projects and ultimately increase the quality of their products [18].

However, techniques to understand the textual data in these repositories are still relatively immature and are a current research challenge, because the textual data is unstructured, unlabeled, and noisy [10]. The sheer size of this data, coupled with its lack of structure, makes analysis difficult or impossible.

Topic Models. *Statistical topic models*, such as latent Dirichlet allocation (LDA), are statistical models that provide a means to automatically index, search, cluster, and structure unstructured and unlabeled documents [3,4]. Topic models accomplish these tasks by discovering a set of *topics* within the documents, where a topic is a collection of co-occurring words. For example, a topic might contain the words {*mouse click left right move scroll*}, because these words tend to occur in the same documents. Each document is assigned a set of topics that describe it. Using this discovered structure, documents can be linked through the topics they contain, and metrics can be calculated for each topic, such as popularity or scatter [16].

Originally developed in the field of information retrieval, topic models have been successfully adapted to other domains such as social sciences [14] and computer vision [2].

2. RESEARCH OVERVIEW

We hypothesize that statistical topic models can be used to automatically discover structure in the textual data found in software repositories. Further, we think that this discovered structure can directly contribute to the solution of practical software engineering tasks, such as bug prediction, software evolution, and traceability link recovery.

In addition to discovering structure, topic models are promising for several reasons. First, topic models require no training data, which makes them easy to use in practical settings. Second, topic models operate directly on the raw, unstructured text without expensive data acquisition or preparation costs. And finally, topic models have proven to be fast and scalable to millions of documents or more [13].

Topic models can be applied to any of several software repositories of interest, such as the identifier names and comments within the source code, bug reports in a bug database, email archives, and execution logs.

Research Goals. The overarching goal of our research is to address the challenges of bringing topic models into the domain of software engineering. Specifically, this includes:

– *Understanding the peculiarities of the data in our domain.* Topic models have been developed and optimized for free-flowing natural text, such as articles, books, and blogs. However, the datasets in our domain differ. For example, source code contains programming language constructs and identifier names; bug reports contain a mix of natural text, source code, and stack traces.

– *Optimizing techniques for our domain.* The preprocessing of source code for analysis with topic models is an important step, but there is no standard within our community. It is currently not clear whether identifier names, comments, sting literals, or some combination should be considered. Additionally, should we remove stop words, split identifier names into multiple words, apply word stemming, and remove common and rare words?

In addition, choosing the right parameters (e.g., number of topics) for the topic model is an open problem [8]. The number of topics has a big impact on the results of topic models, but there is no consensus in our community as to which values are best. Some initial work has provided guidance [8], but more work needs to be performed.

– *Making results of topic models easier to interpret.* Researchers in our domain have treated topic models as a "black box". It is not clear how to interpret the results of topic models, and thus it is not easy to determine whether one topic model is better than another. We aim to make topic models more transparent to our community so that topic models can be better utilized in practical settings.

– *Analyzing successive versions of source code.* Several different *topic evolution models* have been proposed to characterize how a topic evolves over time, but were built for data different from source code, such as yearly conference proceedings or blog posts. Our initial results show that this difference can negatively affect results [15].

– *Adapting topic models to solve practical software engineering problems.* We aim to use the above results to apply topic models to practical SE tasks, such as bug prediction, software evolution, and traceability link recovery.

Related Work. Most work involving the unstructured text in software repositories thus far has used non-statistical information retrieval models, such as the vector space model (VSM) or latent semantic indexing (LSI). While these models have produced promising results, LDA has been shown to be superior to both VSM and LSI in terms of disambiguating synonyms and polysemes, and creating more coherent topics [9].

Indeed, researchers in our domain are beginning to use LDA to mine software repositories. These studies focus on locating concepts in the source code [5], constructing search engines of source code repositories [17], or recovering traceability links between software artifacts [1,12].

However, many opportunities are still unexplored. Several repositories remain largely unanalyzed by topic models, including email archives and bug repositories. The evolution of topics over time in source code has only received an initial treatment [11]. An initial study has used topic models to derive a coupling metric that was shown to correlate well with bugs [6], although many topic metrics remain unexplored.

Progress Thus Far. We have performed two initial qualitative studies on using topic models to detect the evolution of topics in source code [15,16]. Our results indicate that detected changes in topics recovered by LDA correspond well (92%) to the actual change activities in the source code [16], and that analyzing only the changes between source code versions, rather than the entire versions with duplication, leads to a more accurate model of topic evolution [15].

Evaluation Plan. Evaluating topic models is a difficult task, due to a lack of ground truth [19]. We will therefore rely on secondary evaluation techniques based on the specific SE tasks at hand. For example, we can evaluate the performance of bug prediction models using standard recall and precision measures on oracle datasets; we can measure the performance of software evolution models by manually comparing the detected evolution with existing project documentation [16]; and we can evaluate the performance of traceability link recovery using accuracy measures on oracle datasets.

3. REFERENCES

[1] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 95–104, 2010.

[2] K. Barnard, P. Duygulu, D. Forsyth, N. D. Freitas, D. M. Blei, and M. I. Jordan. Matching words and pictures. *The Journal of Machine Learning Research*, 3:1107–1135, 2003.

[3] D. M. Blei and J. D. Lafferty. Topic models. In *Text Mining: Theory and Applications*. Taylor and Francis, London, UK, 2009.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] B. Cleary, C. Exton, J. Buckley, and M. English. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, 14(1):93–130, 2008.

[6] M. Gethers and D. Poshyvanyk. Using relational topic models to capture coupling among classes in Object-Oriented software systems. In *Proceedings of the 26th International Conference on Software Maintenance*, 2010.

[7] M. W. Godfrey, A. E. Hassan, J. Herbsleb, G. C. Murphy, M. Robillard, P. Devanbu, A. Mockus, D. E. Perry, and D. Notkin. Future of mining software archives: A roundtable. *Software, IEEE*, 26(1):67–70, 2008.

[8] S. Grant and J. R. Cordy. Estimating the optimal number of latent concepts in source code analysis. In *Proceedings of the 10th International Working Conference on Source Code Analysis and Manipulation*, pages 65–74, 2010.

[9] T. L. Griffiths, M. Steyvers, and J. B. Tenenbaum. Topics in semantic representation. *Psychological Review*, 114(2):211–244, 2007.

[10] A. E. Hassan. The road ahead for mining software repositories. In *Frontiers of Software Maintenance*, pages 48–57, 2008.

[11] E. Linstead, C. Lopes, and P. Baldi. An application of latent dirichlet allocation to analyzing software evolution. In *Proceedings of the 7th International Conference on Machine Learning and Applications*, pages 813–818, 2008.

[12] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. *Information and Software Technology*, 52(9):972–990, 2010.

[13] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceeding of the 14th International Conference on Knowledge Discovery and Data Mining*, pages 569–577, 2008.

[14] D. Ramage, E. Rosen, J. Chuang, C. D. Manning, and D. A. McFarland. Topic modeling for the social sciences. In *NIPS 2009 Workshop on Applications for Topic Models: Text and Beyond*, 2009.

[15] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. DiffLDA: topic evolution in software projects. Technical Report 2010-574, School of Computing, Queen's University, 2010.

[16] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Validating the use of topic models for software evolution. In *Proceedings of the 10th International Working Conference on Source Code Analysis and Manipulation*, pages 55–64, 2010.

[17] K. Tian, M. Revelle, and D. Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pages 163–166, 2009.

[18] W. Tichy. An interview with prof. andreas zeller: Mining your way to software reliability. *Ubiquity*, 2010, Apr. 2010.

[19] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1105–1112, 2009.