Using Latent Dirichlet Allocation for Automatic Categorization of Software

Kai Tian, Meghan Revelle, and Denys Poshyvanyk
Computer Science Department
The College of William and Mary
Williamsburg, VA 23185
{ktian, meghan, denys}@cs.wm.edu

Abstract

In this paper, we propose a technique called LACT for automatically categorizing software systems in open-source repositories. LACT is based on Latent Dirichlet Allocation, an information retrieval method which is used to index and analyze source code documents as mixtures of probabilistic topics. For an initial evaluation, we performed two studies. In the first study, LACT was compared against an existing tool, MUDABlue, for classifying 41 software systems written in C into problem domain categories. The results indicate that LACT can automatically produce meaningful category names and yield classification results comparable to MUDABlue. In the second study, we applied LACT to 43 software systems written in different programming languages such as C/C++, Java, C#, PHP, and Perl. The results indicate that LACT can be used effectively for the automatic categorization of software systems regardless of the underlying programming language or paradigm. Moreover, both studies indicate that LACT can identify several new categories that are based on libraries, architectures, or programming languages, which is a promising improvement as compared to manual categorization and existing techniques.

1. Introduction

Open-source software repositories such as SourceForge.net maintain massive amounts of source code and software artifacts. To facilitate easier browsing and searching of such repositories, software systems are placed into categories (e.g., text editors, anti-virus, databases, etc). These categories group systems by their functionality, and classification is performed manually by users or administrators. This labor-intensive categorization is time-consuming and requires an understanding of the underlying functionalities of the software systems in the repository.

Automatic categorization is a desirable alternative to the current practice since it eliminates manual effort. An existing research prototype, *MUDABlue* [6], has successfully used Latent Semantic Indexing (LSI) [3], an Information Retrieval (IR) technique, to automatically categorize software systems in open-source software repositories. Latent Dirichlet Allocation (LDA) [2] is an alternative IR approach in which documents can be viewed as a mixtures of topics, which may make it more amenable to software categorization than LSI. If we consider a software system in an open-source repository to be a document, the distribution of topics in that document can be used to automatically place the software system into categories. In this paper, we propose a novel technique called *LACT* for automatically classifying software systems in open-source repositories. *LACT* works by using LDA's topic-document distributions that are gleaned from comments and identifiers in source code. We conducted two initial studies, one aimed at comparing *LACT* with *MUDABlue* on a previously published dataset, and the other studying *LACT* when categorizing software systems written in different programming languages. The next sections present the details of *LACT* and the results of our studies.

2. Using Latent Dirichlet Allocation for Software Categorization

LDA is a probabilistic topic model originally used in natural language processing, but it has also been applied to software artifacts [1, 8-10, 12]. In LDA, documents are represented as mixtures over latent topics, and each topic is characterized by a distribution over words [2]. Given a corpus of documents, LDA identifies a set of topics, associates a set of words with each topic, and defines a finite mixture of these topics for each document. Our proposed technique of *LACT* utilizes LDA as described in the following steps:

1. **Parse software systems**. We consider a software system as a collection of words (i.e., identifiers and comments). Each system is parsed and represented as a document in a corpus (see Table 1).
2. **Index corpus with LDA**. We use GibbsLDA++[1] to index the resulting corpus. Topic-document or

[1] http://gibbslda.sourceforge.net/ (accessed and verified on 03/01/09)

Table 1. Using LDA for software categorization

LDA Model	Source Code Entities
word	Identifiers and comments are extracted from source code to form a vocabulary set. This set is refined to exclude programming language keywords, stop words, and punctuation. All compound identifiers are split based on observed naming conventions. $V=\{w_1, w_2,..., w_v\}$.
document	A software system is treated as a document which can be expressed as *n* identifies and comments from a vocabulary. $s_i=(w_1,w_2,...,w_n)$
corpus	A corpus consists of a set of indexed software systems. C = $\{S_1, S_2, ..., S_z\}$

topic-software matrices are obtained from LDA in which each document (i.e., software system) is probabilistically associated with a set of topics. The number of topics generated is a parameter of LDA, and we explore how the number of topics impacts results in our studies (see Section 3.1).

3. **Retrieve categories**. To group similar topics around categories, we compute cosine similarities between each pair of topics. If a cosine similarity between two topics is greater than *0.8*, we cluster them into the same category. Note that a topic may belong to several different categories. The result of this step is a listing of categories and topics.
4. **Categorize software systems**. Once the categories are populated with topics, the software systems are assigned to the categories. Since we obtain topic-software matrices in Step 2, we can derive a list of topics for each software system as follows: if one of the category topics belongs to a software system with a probability above a certain distribution threshold, then the software system is assigned to that category. This criterion does not preclude assigning a software system to multiple categories (e.g., *KOffice* may be placed into the *text editor* and *spreadsheet* categories). We study the impact of various distribution thresholds in Section 3.1.

3. Case Studies

We performed two initial studies to evaluate *LACT*. In the first, we compare *LACT* head-to-head with *MUDABlue* [6] on a set of 41 open-source software systems written in C. In the second case study, we apply *LACT* to a set of 43 open-source software systems written in different programming languages and compare the results to the manual categorizations provided on SourceForge. The results of these studies are discussed below. The full results from our case studies are available as an online appendix.[2]

[2] http://www.cs.wm.edu/~denys/data/msr09/msr09-appendix.htm

Table 2. Precision and recall for various distribution thresholds and number of topics

# of topics	Distribution Threshold					
	0.001	**0.005**	**0.01**	**0.02**	**0.05**	**0.1**
10	0.54,0.52	0.56,0.53	0.57,0.52	0.59,0.54	0.58,0.56	0.56,0.54
20	0.57,0.55	0.58,0.56	0.61,0.58	0.64,0.63	0.65,0.63	0.62,0.59
30	0.62,0.61	0.62,0.61	0.64,0.63	0.69,0.70	0.68,0.65	0.68,0.64
40	0.66,0.65	0.69,0.66	0.71,0.70	**0.74,0.72**	0.73,0.73	0.69,0.69
50	0.63,0.61	0.62,0.62	0.65,0.64	0.68,0.70	0.68,0.69	0.66,0.65
60	0.64,0.63	0.66,0.65	0.69,0.68	0.69,0.70	0.64,0.63	0.69,0.68
70	0.68,0.67	0.71,0.70	0.74,0.73	0.73,0.73	0.69,0.68	0.62,0.64
80	0.56,0.57	0.73,0.72	0.64,0.63	0.70,0.69	0.64,0.63	0.68,0.67

3.1. Comparison with *MUDABlue*

We evaluated *LACT* on the same 41 software systems used to evaluate *MUDABlue* in [6]. We explored the categorization results in terms of the number of topics and the distribution thresholds. The number of topics ranged from 10 to 80, and the distribution thresholds were between 0.001 and 0.1. For each configuration, precision and recall was computed using the SourceForge categorizations as the ideal set. *Precision* is the number of systems correctly categorized by a technique divided by the total number of systems categorized, while *recall* is the number of correctly categorized systems divided by the total number of ideal categorizations. Table 2 shows *LACT*'s precision and recall results. The performance of our technique varies greatly in terms of the number of topics and the distribution threshold. However, *LACT*'s performance is generally comparable with that of *MUDABlue* [6]. The highest precision and recall values come from 40 topics with distribution threshold of 0.02. When the number of topics is too large, *LACT* generates very fine-grain categories, which means each category has only one or two software systems.

Using 40 topics and a distribution threshold 0.02, *LACT* generates 33 categories. As compared to the manual categorization on SourceForge, *LACT* generates 19 of the same categories. In addition, *LACT* generates 14 new categories which are not defined on SourceForge. Among the new categories, seven are based on libraries or architectures. The other new categories do not represent meaningful concepts. Identifying and eliminating these hollow categories is part of the future work of this paper.

We compared *LACT*'s results under its best configuration directly with *MUDABlue*. While *MUDABlue* generated 40 categories for these systems, *LACT* generated 33 categories. The average number of software systems in each category is 2.6 and 3.125 for *MUDABlue* and *LACT* respectively. *MUDABlue* generated 18 of the same categories defined on

Table 3. 43 software systems used in programming language-independent categorization study

Category	C/C++	PHP	Java	Perl	C#
Game	gcells-0.4, freedroid-1.0.2, fuzzy_adventure-0.5	nomic-1.3, Netrisk 2.0, deep dungeons-0.1	Bubblebreaker-0.1	fortune	SharpTTT-1.2.1
Editor	pdfedit-0.4.2, npp-5.0, AkelPad-3.6.4	ontext-0.3, FCKeditor 2.6.3	jedit-4.2,VietPad-2.0, rtext 0.9.9.7	Kephra-0.4	
Database	postgresql-8.3.0	phpMyAdmin-3.0.1.1	smallsql-0.20, DBEdit2.1.5	SequelExplorer0.7	
Terminal	anyterm-1.0.1,cgterm-1.6, putty-0.60, tatelnet-1.1.1	ajaxphpterm, phpterm-0.3.0	jalita-1.0, j2ssh-0.2.9		
E-mail	jwsmtp-1.32.15, hotpop3.0.0.1	squirrelmail-1.4.17, emailer-2.3.1	OpenJMail 1.0.9	Mercury-0.10	ILKMail v1.2, cses-0.3
Chat	gib-0.2	Web2IRC	exb-2.1.0-20020504	pircd-beta-one	

SourceForge, and *LACT* generated 19. For the new, non-SourceForge categories generated by both techniques, *MUDABlue* generated 11 meaningful categories while *LACT* generated seven. Among these categories, *LACT* had three categories in common with *MUDABlue*: *GTK*, *YACC*, and *SSL*. *LACT* generated two additional categories, *XML* and *SQL*, which *MUDABlue* did not. On the other hand, *MUDABlue* generated some categories that *LACT* did not, but the meaning of a number of these categories is difficult to interpret. *LACT* also generated six categories with titles from which it is difficult to discern meaning.

MUDABlue generated 11 categories whose titles contain identifiers such as "*X*" and "*a*". Outside of the source code, identifier names such as these have little to no meaning. As for the category titles, *LACT* provided significant improvement over *MUDABlue* because *LACT* pre-processes all identifiers to remove non-literals, keywords, tokens, and split identifiers to generate meaningful category names. As a result, the names of categories generated by *LACT* are more readable than those of *MUDABlue*.

3.2. Language-Independent Categorization

We also evaluated the performance of *LACT* using 43 open-source software systems written in various programming languages. Among these software systems, 14 software systems are written in C/C++, 10 are in Java, 11 in PHP, 5 in Perl, and 3 in C#. The systems, listed in Table 3, were randomly selected from six categories on SourceForge: *Game*, *Editor*, *Database*, *Terminal*, *E-mail*, and *Chat*. These systems belong to other manually assigned categories as well.

After running *LACT* using 45 topics and a distribution threshold of 0.05, 34 categories were generated. Compared to the manual categorization on SourceForge, *LACT* found nine of the same categories and generated 25 new categories. Among the new categories, 15 are meaningful as they are either based on libraries or architectures (*GTK*, *MFC*, *SSL*, and *XML)* or are based on certain programming languages. The other 10 new categories are not meaningful, and future work will involve eliminating such categories.

The categorization results of using these 43 software systems written in different languages are quite different from the previous study where the software system were all written in C. One possible reason is that a programming language acts as a noise factor in categorization, and sometimes *LACT* categorizes the software systems written in the same language into a certain category. *LACT*'s ability to categorize software systems based on programming language is more flexible and applicable than existing approaches.

3.3. Threats to Validity

Threats to external validity include to what degree the software systems in this case study are representative of all software systems. In the programming language-independent case study, we used software systems written in several popular programming languages from only a few domains. While meaningful categories were found, a larger sample of systems is needed to ensure that the results are applicable to all software systems. Threats to internal validity include choosing the number of topics and the distribution threshold to generate the best categorization results. To minimize this threat, we studied a range of values. We also acknowledge the cosine similarity value (i.e., 0.8) used to cluster topics into categories. Other similarity criteria could have resulted in different category-topics configurations. Another threat to internal validity exists in the subjectivity of judging categories. When calculating precision and recall, we manually determined correspondence of generated categories to ideal ones. An alternative way to conduct the study would be to collect categorizations from a group of diverse people and calculate precision and recall values based on the group's agreement. Another threat to validity is the manual evaluation and comparison of the categorization results generated by *MUDABlue* and *LACT*. However, this is a common problem in evaluating software categorization systems. To the best of our knowledge, there are no good, existing metrics or criteria to quantify the quality of software categorization results.

4. Related Work

Information Retrieval has previously been used to categorize software systems. Kawaguchi et al. [6] applied LSI to categorize software systems automatically with *MUDABlue*. They use source code identifiers and ignore comments, while *LACT* uses LDA, employs comments, and pre-processes identifiers to produce more legible category names.

At a finer granularity than software systems, machine learning techniques have been exploited to automatically categorize software modules [5]. Similarly, different IR techniques such as LSI, pLSI, and Naïve Bayes approaches have also been applied to categorize reusable software components [15]. Software components have also been retrieved using IR techniques for software reuse. CodeBroker [17] is a tool for retrieving reusable components from a software system using IR. SpotWeb [16] is a code search engine based approach that detects *hotspots* and *coldspots* in a given framework by mining open-source code on the web. Automated approaches for assessing reuse also exist for software libraries [11, 13], reuse repositories [14] and ranking components by their reusability [4]. Unlike *LACT*, these approaches concentrate on reuse of software components instead of categorization of software systems. Semantic clustering is a technique used to identify topics in source code [7] that uses LSI and clustering algorithms to group source artifacts that use a similar vocabulary. Semantic clustering, like *LACT*, defines linguistic topics which reveal the intention of code.

5. Conclusions

In this paper, we introduced *LACT* as a technique for automatically categorizing software systems using Latent Dirichlet Allocation. Our initial results indicate that *LACT* can effectively categorize systems implemented in different programming languages according to manual assignments as well as find new, useful categories. Additionally, *LACT* generates more comprehensible category names as compared with an existing automatic categorization system.

6. Acknowledgements

We acknowledge Dr. Shinji Kawaguchi and Dr. Katsuro Inoue for providing us with *MUDABlue*'s categorization results. We thank anonymous reviewers for detailed comments and helpful suggestions. This research was supported in part by the United States Air Force Office of Scientific Research under grant number FA9550-07-1-0030.

7. References

[1] Baldi, P., Linstead, E., Lopes, C., and Bajracharya, S., "A Theory of Aspects as Latent Topics", in Proc. of Conference on Object-Oriented Programming, Systems, Languages and Applications, 2008.
[2] Blei, D. M., Ng, A. Y., and Jordan, M. I., "Latent Dirichlet Allocation", *Journal of Machine Learning Research*, vol. 3, 2003,.
[3] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R., "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, vol. 41, 1990.
[4] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., and Kusumoto, S., "Ranking significance of software components based on use relations", *Trans. on Software Engineering*, vol. 31, no. 3, March 2005, pp. 213- 225.
[5] Kawaguchi, S., Garg, P. K., Matsushita, M., and Inoue, K., "Automatic Categorization Algorithm for Evolvable Software Archive", in Proc. of International Workshop on Principles of Software Evolution, 2003, pp. 195-200.
[6] Kawaguchi, S., Garg, P. K., Matsushita, M., and Inoue, K., "MUDABlue: An automatic categorization system for Open Source repositories", *Journal of Systems and Software*, vol. 79, no. 7, 2006.
[7] Kuhn, A., Ducasse, S., and Gîrba, T., "Semantic Clustering: Identifying Topics in Source Code", *Information and Software Technology*, vol. 49, no. 3, March 2007, pp. 230-243.
[8] Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P., "Mining concepts from code with probabilistic topic models", in Proc. of International Conference on Automated Software Engineering, 2007, pp. 461-464.
[9] Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P., "Mining Eclipse Developer Contributions via Author-Topic Models", in Proc. of International Workshop on Mining Software Repositories, 2007, pp. 30-33.
[10] Lukins, S., Kraft, N., and Etzkorn, L., "Source Code Retrieval for Bug Location Using Latent Dirichlet Allocation", in Proc. of Working Conference on Reverse Engineering, 2008, pp. 155-164.
[11] Maarek, Y. S., Berry, D. M., and Kaiser, G. E., "An Information Retrieval Approach for Automatically Constructing Software Libraries", *Trans. on Software Engineering*, vol. 17, no. 8, 1991.
[12] Maskeri, G., Sarkar, S., and Heafield, K., "Mining Business Topics in Source Code using Latent Dirichlet Allocation", in Proc. of 1st Conference on India Software Engineering Conference, Hyderabad, India, 2008, pp. 113-120.
[13] Michail, A. and Notkin, D., "Assessing software libraries by browsing similar classes, functions and relationships", in Proc. of International Conference on Software Engineering, 1999.
[14] Pan, Y., Wang, L., Zhang, L., Xie, B., and Yang, F., "Relevancy based semantic interoperation of reuse repositories", in Proc. of FSE, 2004, pp. 211-220.
[15] Sandhu, P. S., Singh , J., and Singh, H., "Approaches for Categorization of Reusable Software Components", *Journal of Computer Science*, vol. 3, no. 5, 2007, pp. 266-273.
[16] Thummalapenta, S. and Xie, T., "SpotWeb: Detecting Framework Hotspots and Coldspots via Mining Open Source Code on the Web", in Proc. of ASE, 2008.
[17] Ye, Y. and Fischer, G., "Reuse-Conducive Development Environments", *Journal Automated Software Engineering*, vol. 12, no. 2, 2005, pp. 199-235.