Mining Eclipse Developer Contributions via Author-Topic Models

Erik Linstead†, Paul Rigor†, Sushil Bajracharya*,
Cristina Lopes*, Pierre Baldi†
†Institute for Genomics and Bioinformatics *Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
{elinstea,prigor,sbajrach,lopes,pfbaldi}@ics.uci.edu

Abstract

We present the results of applying statistical author-topic models to a subset of the Eclipse 3.0 source code consisting of 2,119 source files and 700,000 lines of code from 59 developers. This technique provides an intuitive and automated framework with which to mine developer contributions and competencies from a given code base while simultaneously extracting software function in the form of topics. In addition to serving as a convenient summary for program function and developer activities, our study shows that topic models provide a meaningful, effective, and statistical basis for developer similarity analysis.

1 Introduction

As the availability of open source software repositories continues to grow, so does the need for tools that can automatically analyze these repositories on an increasingly larger scale. The ability to mine source code for functionality, team structure, and developer contributions is also of great interest to private industry, where these tools can be applied to such problems as in-house code reuse and project staffing. While accomplishing these tasks for arbitrarily large code repositories has proven challenging, recent progress in statistical topic modeling provides another direction for research in this area.

Automated topic modeling has been successfully used in the text mining and information retrieval communities where it has been applied to the problem of summarizing large text corpora. Recent techniques include Latent Dirichlet Allocation (LDA), which probabilistically models text documents as mixtures of latent topics, where topics correspond to key concepts presented in the corpus [3]. Author-Topic (AT) modeling is an extension of topic modeling that captures the relationship of authors to topics in addition to extracting the topics themselves. A recent advancement in this area is the probabilistic Author-Topic model, an augmentation of LDA that models the distribution of authors over topics in addition to topics over documents [7]. The end result is the ability to extract the most likely authors for topics mined from the corpus, where the list of topics is determined by the algorithm. These more recent approaches have been found to produce better results than traditional methods such as latent semantic analysis (LSA).

Despite the capabilities of the probabilistic author-topic model, its applications have generally been limited to traditional text corpora such as academic publications, news reports, and historical documents [5]. At the lowest level, however, a code base is nothing more than a text corpus where source files are analogous to documents and developers to authors. Though syntax and convention differentiate a programming language from a natural language, the tokens present in a source file are still indicative of its function (ie. its topics).

For the 2007 MSR challenge we develop and apply author-topic models to software data, and demonstrate the results with the Eclipse 3.0 source code to answer the following data mining questions:

- What topics define the functionality of the Eclipse 3.0 code base?
- Which developers are most likely to contribute to a particular topic?
- Which developers are most similar based on their contributions across topics?

The remainder of the paper is dedicated to answering these questions. Section 2 describes the dataset used in our analysis. Section 3 summarizes the tools and methodology used in applying the AT model to the Eclipse source. Section 4 presents and discusses the results of our mining technique, which leads to conclusions and future work in section 5.

Fourth International Workshop on Mining Software Repositories (MSR'07)
0-7695-2950-X/07 $20.00 © 2007 IEEE

2 Input Data

To demonstrate our approach we consider the 3.0 version of Eclipse from the project archive [1]. While our tools are general enough to be applied to documentation, build files, and additional data distributed with the source, the focus of our efforts for this task is on Java source code alone. Though we have developed a substantial crawling infrastructure to be able to run topic modeling algorithms on arbitrarily large code repositories, its use was not required for the Eclipse mining challenge since all code was available in a single zip archive.

Exploratory data analysis indicates that author information is not readily available in the Eclipse 3.0 code base in the form of javadoc tags. To generate a list of authors for specific source files we leverage the Eclipse bug data available in [6]. Parsing the provided XML files for version 3.0 allows us to derive a list of contributing developers, as well as the source files modified by each developer.

After pruning files not associated with any author, the input dataset consists of 2,119 Java source files, comprising 700,000 lines of code, from a total of 59 developers.

3 Tools and Methodology

Applying AT models to software requires the development of several tools to facilitate the processing and modeling of source code. A key task of these tools is the conversion of source code into compatible representations for the AT algorithm, required as input parameters. Of these parameters, the most important are the word-document matrix and the author-document matrix. These matrices represent the occurrence of words in individual documents and the assignment of authors to individual documents, respectively.

To produce the word-document matrix for our input data we developed a comprehensive tokenization tool tuned to the Java programming language. This tokenizer includes language-specific heuristics that follow the commonly practiced naming conventions. For example, the Java class name "QuickSort" will generate the words "quick" and "sort." All punctuation is ignored.

As an important step in processing source files our tool removes commonly occurring stop words. For the purposes of our mining task we augment a standard list of stop words used for the English language (eg. and, the, but, etc) to include the names of all classes from the Java SDK (eg. ArrayList, HashMap, etc). This is done to specifically avoid extracting common topics relating to the Java collections framework, thus focusing the analysis on what the code is doing rather than how it is doing it.

Running the tokenizer on our input dataset yields a vocabulary of 15,391 distinct words. The result is then a 15,391 x 2,119 word-document matrix such that entry $[i,j]$ corresponds to the number of times word i occurs in document j.

The author-document matrix is produced directly from the Eclipse bug data XML. It is a 59 x 2,119 binary matrix where $[i,j]$=1 if author i contributed to document j, and 0 otherwise.

For convenience we relied on a popular Matlab implementation of the LDA-based AT algorithm from [2]. The algorithm was run on the input matrices with additional input parameters specifying that 100 topics (a number determined from experimentation) should be extracted from the code. The number of iterations, i, to run the algorithm was determined empirically by analyzing results for i ranging from 500 to several thousand. Final results were captured using 3,000 iterations, which we found to produce interpretable topics (ie. topics whose words could be associated with an intutive label such as testing or code completion) in a reasonable amount of time. Because the algorithm contains a stochastic component we also verified the results of multiple runs of i=3,000. In total, the process of parsing and topic modeling required about 14 hours to run to completion on a a single Sun SunFire X2200 M2 Server. This machine contained two dual-core AMD Opteron processors along with 8GB of RAM

As output the algorithm produces a document-topic matrix and author-topic matrix specifying the number of times a document or author was assigned to each of the 100 topics extracted from the code. The topics themselves are defined by representative words from the corpus.

4 Results

A representative subset of 7 topics extracted via Author-Topic modeling on the selected 2,119 source files is given in Table 1. Each topic is described by several words associated with the topic concept. To the right of each topic is a list of the most likely authors for each topic with their probabilities. The complete list of 100 topics with word and author distributions is available from: http://sourcerer.ics.uci.edu/msr2007/index.html.

Examining the topic column of the table it is clear that various functions of the Eclipse framework are represented. For example, topic 1 clearly corresponds to unit testing, topic 2 to debugging, topic 4 to building projects, and topic 6 to automated code completion. Remaining topics range from package browsing to compiler options. The presence of some words that do not seem to "fit" the overall topic category is both a result of the text corpus itself and the mathematical machinery behind the author-topic algorithm. For example, "swt" is assigned to the testing topic, likely due to co-occurrence within source files. Some topics are also more specific than others, and do not lend themselves as easily to high-level lables such as "testing" or "debug-

0-7695-2950-X/07 $20.00

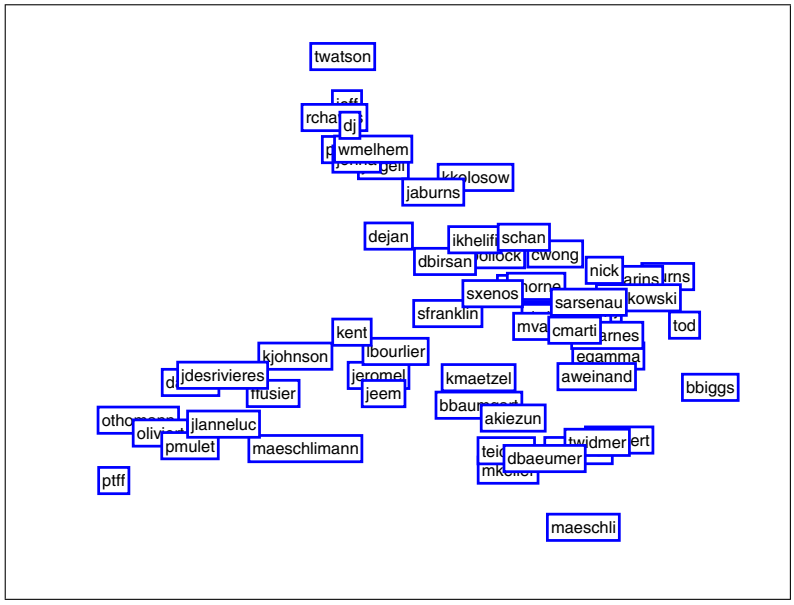

Figure 1. Authors Clustered by KL Divergence

ging". Specific topics are ultimately best interpreted by domain experts rather than those only casually familiar with the contents of the corpus being analyzed.

Examining the author assignments (and probabilities) for the various topics provides a simple means by which to discover developer contributions and infer their competencies. It should come as no surprise that the most probable developer assigned to the JUnit framework topic is "egamma", or Erich Gamma. In this case, there is a 97% chance that any source file in our dataset assigned to this topic will have him as a contributor. Based on this rather high probability, we can also infer that he is likely to have extensive knowledge of this topic. This is of course a particularly attractive example because Erich Gamma is widely known for (among other things) being a founder of the JUnit project, a fact which lends credibility to the ability of the topic modeling algorithm to assign developers to reasonable topics.

One can interpret the remaining author-topic assignments along similar lines. For example, developer "daudel" is assigned to the topic corresponding to automatic code completion with probability .99. Referring back to the Eclipse bug data it is clear that the overwhelming majority of bug fixes for the codeassist framework were made by this developer. One can infer that this is likely to be an area of expertise of the developer.

The observation that most topics are dominated by a single developer stems from the fact that only a limited amount of author information can be extracted from the bug data. Source files that were not included in bug fixes could not be associated with developers, and some Eclipse components had a single developer responsible for all bug fixes. Nevertheless, topics 5 and 7 in Table 1 are examples of concepts to which several developers are assigned with non-trivial probability. Since authors assigned to the same topic with similar probabilities are likely to have made similar contributions, using topic assignments as a basis for developer comparison is a natural next step.

To calculate developer similarity we convert the author-topic matrix to a similarity matrix capturing the pairwise "distance" between authors. Several methods for computing distance were considered, including standard metrics such as Euclidean and cosine distance. Best results were ultimately found with symmetrized Kullback-Leibler (KL) divergence based on reconstructing the probability distributions over topics for each author. Using visualization tools from [2] as a template, multidimensional scaling (MDS) was employed to visualize author similarities, resulting in Figure 1. The boxes represent individual developers, and are arranged such that developers with similar topic distributions are nearest one another.

While one could conceive of other ways in which to define developer similarity, a topic based approach has its advantages. Because topics are based on actual source files modified by the developer, the very nature of the comparison takes into account skill areas and experience. This information is useful when considering how to form a development team, or choosing suitable programmers to perform code updates. Indeed, a similar concept was used in [4] to identify candidates for bug fixes based on bug report descriptions. The technique presented here furthers this work by taking actual code into account, as well as leveraging the added flexibility of modeling mixtures of topics.

0-7695-2950-X/07 $20.00

Topic	Author Probabilities
junit	egamma 0.97065
run	wmelhem 0.01057
listener	darin 0.00373
item	krbarnes 0.00144
suite	kkolosow 0.00129
swt	maeschli 0.00129
runner	twidmer 0.00114
target	jaburns 0.96894
source	darin 0.02101
debug	lbourlier 0.00168
breakpoint	darins 0.00113
location	jburns 0.00106
step	aweinand 0.00086
core	kkolosow 0.00038
ast	maeschli 0.99161
button	mkeller 0.00097
cplist	othomann 0.00055
entries	tmaeder 0.00055
astnode	teicher 0.00046
iclasspath	jeromel 0.00038
key	cmarti 0.00038
nls-1	darins 0.99572
ant	dmegert 0.00044
manager	nick 0.00044
listener	kkolosow 0.00036
classpath	maeschli 0.00031
property	kjohnson 0.00023
build	darin 0.00015
type	kjohnson 0.59508
length	jlanneluc 0.32046
names	darin 0.02286
match	johna 0.00932
methods	pmulet 0.00918
enclosing	lbourlier 0.00854
table	aweinand 0.00783
token	daudel 0.99014
completion	teicher 0.00308
current	jlanneluc 0.00155
identifier	twatson 0.00084
unicode	dmegert 0.00046
assist	lptff 0.00046
top	prapicau 0.00025
fragment	dmegert 0.44173
parent	maeschli 0.23942
children	bbaumgart 0.22126
ipackage	egamma 0.02897
delta	mkeller 0.02868
ijava	tmaeder 0.01953
root	darin 0.00167

Table 1. Seven Representative Topics and Most Likely Authors with Probabilities

5 Conclusions

This paper addressed the effectiveness of the probabilistic Author-Topic model for mining developer contributions and similarities from the Eclipse 3.0 source code. Results indicate that the algorithm produces reasonable and interpretable automated topics and author-topic assignments. Additionally, topics provide a basis for comparing the similarity of developers based on their contributions, which is of potential use in bug fix assignments and staffing.

We are currently expanding the scope of our tools by considering a much larger repository of open source code consisting of over 6,000 Java projects from SourceForge and Apache. To facilitate this process, we have built crawlers to automatically download and depackage projects from the Internet. Our parser was also augmented to extract several thousand author names directly from code. Preliminary results indicate that our methods scale well with repository size, and a substantial increase in the diversity of both topics and author contributions has been observed.

In the future we intend to explore the applicability of topic models to aspect-oriented programming (AOP), as many topics extracted by our tools correspond to common crosscutting concerns. To this end, we envision topic distributions providing an automated mechanism for computing scattering and tangling for Internet-scale code repositories.

References

[1] Eclipse Archive. *http://archive.eclipse.org/eclipse/downloads*.

[2] Steyvers, Mark. Topic Modeling Toolbox. *http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm*.

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

[4] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1767–1772, New York, NY, USA, 2006. ACM Press.

[5] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *UAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494, Arlington, Virginia, United States, 2004. AUAI Press.

[6] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller. If your bug database could talk.... In *Proceedings of the 5th International Symposium on Empirical Software Engineering, Volume II: Short Papers and Posters*, pages 18–20, September 2006.

[7] M. Steyvers, P. Smyth, M. Rosen-Zvi, and T. Griffiths. Probabilistic author-topic models for information discovery. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315, New York, NY, USA, 2004. ACM Press.

0-7695-2950-X/07 $20.00 © 2007 IEEE