Mining business-themes in the source language using the Latent Dirichlet distribution

ABSTRACT

One of the difficulties in maintaining a large software system is the absence of written business domain titles and the association between these domain titles and source code. Without such interactions, people with no prior knowledge of the application may find it difficult to understand the functionality of the application. The Latent Dirichlet Allocation (LDA), a mathematical model, has emerged as a popular method of finding titles in large corpus text. But its effectiveness in extracting business domain titles from source code has not been tested yet. This paper examines the LDA in the context of understanding major software systems and suggests a personalized approach in terms of the LDA for extracting domain titles from source code. This method has been used in many open source sources and related systems. Preliminary results indicate that the LDA is able to identify some of the domain titles and is a satisfactory starting point for further refinement of titles.

Categories and Definitions of Articles

Keywords: Distribution, storage, and upgrades - redesigning, reversing engineering, and remodeling;

General terms

Latent Dirichlet Offering

Keywords

Maintenance, system understanding, LDA

1. INTRODUCTION

Large legacy software programs are always in a state of disarray with bad or non-existent documents. Adding new features and fixing bugs to such a system is very flawed and time consuming from the original authors of

Permission to make digital or hard copies of all or part of the work for personal or classroom use is provided free of charge unless copies are made or distributed for profit or commercial gain and that copies contain this notice and full documentation on the front page. Alternative copying, republishing, sending to servers or redistributing lists, requires prior authorization and / or funding.

the system is usually no longer available. In addition, code-base people do not understand the purpose of the operation of various programs (tasks, files, classes, data structure etc.) and the roles they play in fulfilling the various services offered by the system.

When the software is small, one can understand its functionality by browsing the source code manually. For large systems, physicians often rely on system analytics techniques such as telephone graphs, data control and flow and cutting [2]. The reason is not difficult to understand. Structure analysis techniques apply to structural information based on a set of system source code. Composition data is at a very low level of granularity - such as files, tasks, data structure, dependence on flexible usage, activity calls and so on. This information does not disclose any operational intent. In a large system, this information becomes too much for the maintainer to get any purpose of working on it. Moreover, in a system with millions of lines of code, the memory requirement for capturing structural data and performing various analyzes is usually bottled.

An important step in understanding the purpose of a program's performance (or targeted performance structure) is to identify the business topics that are present in the system, in which high-level components (or modules) are included. For example, think of a large banking system that deals with customers, bank accounts, credit cards, interest rates, and so on. An inexperienced saver of the app will find it difficult to add a new calculation concept of interest. However, if it is possible to remove business topics such as "customer", "interest" from source code and establish a link between "interest" and various system programs, it can be of great help to the maintainer to find related functions, files, classes and data structures and make the necessary changes in the calculation of interest. This can also make the novice developer more productive in system maintenance, especially when the software is in line with millions of lines of code with small text.

A sensible way to identify these topics is to obtain valuable information by extracting and analyzing various "key words" in the source code text [13]. It is not uncommon for early coders to leave references in the description of system objects in the form of keywords in files, functions, data type names and so on. For example, it is very common to use keywords such as "proxy", "http" when http-proxy is enabled. Similarly, with a banking application one would certainly like to use a keyword such as “interest” in the performance of transactions, classes and data structures related to interest calculations.

Considering that meaningful keywords exist in system objects, is it possible to combine these keywords into logical categories where each sound group of keywords can be defined as “Subject”? For example, is it possible to gather all the proxy-related keywords into one set and identify them as the "proxy" and "verification" keywords in the other keywords that make up the "authenticity" title? If so, one can establish a connection between the "proxy" title and the various program elements (files, tasks or data structures) associated with the "proxy".

This paper addresses the above problem and suggests how to help people according to the Latent Dirichlet Allocation (LDA) [7] for identifying topics in the source code.

This paper addresses the above problem and suggests how to help people according to the Latent Dirichlet Allocation (LDA) [7] for identifying topics in the source code. The LDA is well-known in the field of textual classification and the identification of topics from literary documents. To the best of our knowledge, This is the first attempt to use LDA in the context of source code analysis.

This paper is structured as follows: In the following section we provide a brief review of the relevant documentation of our work and the information required after the LDA. Section 3 discusses the role of LDA in extracting domain titles from source code and provides an interpretation of the LDA model in the source code context. A detailed description of extracting domain titles from source code using LDA is presented in section 4. We have used LDA in open source programs and related programs. Section 5 presents the results obtained. The pros and cons of the LDA-based approach and other interesting developments made during our evaluation are described in Section 6. Finally, Section 7 discusses future research indicators and concludes the paper.

1. DAY AND RELATED WORK

Researchers have long recognized the importance of language knowledge as identifying words and comments in the understanding of the system. For example, Biggerstaff et al. [5] suggested the provision of domain concepts as a way to understand the system. Tonella et al. [8] promote job qualifications and signatures with specific domain details. Anquetil et al. [3] suggests that the information obtained from the file name often serves the purpose of the source code specified in the file. Wilde et al. [24] also suggested the use of language knowledge to identify the purpose of the program. Since then, language data has been used in various system analyzes and archives such as tracking between external texts and source code [4, 16], embedded location [17, 21, 25], identifying high-level concepts [15] and so on.

More recently, language data has also been used to identify topics in source code and was later used for software collection [13] and software segmentation [11].

Kuhn et al. [13] use Latent Semantic Analysis (LSA)

[9] A method based on the identification of topics in the source code by skillfully integrating art software such as methods, files or packages based on indexes and comments. The way we work is different from that of Kuhn et al. in two ways. First, and most importantly, our interpretation of the "topic" is different from that of Kuhn. Kuhn translates software-based software technologies (such as methods, files, etc.) as titles while translating a set of doctrinal language words based on identifiable words and "title" comments. Another important difference lies in the method used for semantic collection. While Kuhn et al. we have adopted LSA to integrate a set of software components, our method of combining language words is based on the Latent Dirichlet Allocation.

Kawaguchi et al. [11] uses language information in source code to automatically identify categories and classify open source repositories. The set of related identifiers is classified as “category”. In our case, we refer to a set of identifiable words as "title"; therefore the title can certainly be considered a “category”. However, our method of integrating terminology-related terminology is different from the method proposed by Kawaguchi et al. [11]. Kawaguchi et al. first it uses LSA to find similar pairs between terms and then uses a merging algorithm to combine similar words. The LDA-based approach we have used reduces the need for two steps. Since LDA is actually a method of modeling topics, it not only finds similarities between words, and creates a collection of similar words to form a theme. Throughout this section, we provide a brief description of LDA and its use in extracting articles from text.

1.1 LDA

The Latent Dirichlet Allocation (LDA) [7] is a mathematical model, in particular the subject model, which was originally used in the field of representative textual analysis. The basic idea of ​​the LDA is that the document can be considered as a mixture of a limited number of titles and that every word that sounds familiar to the text can be combined with one of these titles. By providing a draft document, the LDA seeks to obtain the following:

• Identifies a set of topics

• Links vocabulary to topic

• Describes a specific combination of these topics in each text in the corpus.

LDA is used to extract titles from text. For example, Newman et al. [19] applied to the LDA for 400 articles such as "September 11 attack", "Harry Potter", "Basketball" and "Holidays" from a compilation of 330000 New York Times news articles representing the each news article as a compilation of these articles. The LDA was also requested to identify topics in various fields. For example, the LDA has been used to obtain scientific articles on paper photographs published in the national science school program [10]. McCallum et al. [18] suggested that LDA publish articles on social networks and use them in the 250,000 Enron email group. The difference in LDA was also used by Steyvers et al. [22] analyzed 160,000 themes from a computer-assisted "citeseer" collection Recently, Zheng et al. [6] used LDA to obtain various biological concepts in a protein-related organization

To be complete, we briefly introduce the LDA model. A complete and complete description of the LDA model can be found in [7]. The vocabulary for the LDA model is as follows:

Name Name is a basic unit defined as an object from a V-size vocabulary.

document Text is a sequence of N words denoted by d = (w1, · Saka, wN) where wn is the nth word respectively.

Corpus The corpus is a collection of M letters shown by D = {d1, · Saka ,, dM}.

The LDA adopts the production process of document documentation [7] as presented below.

1. Select N ∼ Poisson (ξ): Select the number of words

N

2. θ ∼ Dir (α): Select θ from the dirichlet distribution marked by the parameter by α.

3. They alone will do it

(a) Select the theme zn ∼ Multinomial (θ)

(b) Choose the word wn from p (wn | zn, β), which may be multinomial

In this model, the various distributions i.e., a set of titles, the distribution of titles in other texts and the possibility of the existence of words in each topic are generally not acceptable for direct discovery [7]. A variety of algorithms close to LDA are therefore considered. These algorithms attempt to increase the chances of a given corpus model. Several algorithms for incorporating the LDA model into text such as varies Bayes [7], expected distribution [14], and Gibbs sample [10] have been suggested.

2. USE OF LDA TO RECEIVE ACT

Given that LDA has been successfully applied to a large corpus of textual information (as discussed in section 2.1), it is interesting to examine i) how it works in the context of the source code ii) how this process works in identifying business topics large software program. To incorporate LDA into source code, we consider the software program as a collection of source code files and the software program is associated with a collection of business domain concepts (or topics). For example, the Apache web server uses http-proxy compatibility, authentication, server, caching and more. Similarly, a data server like Postgresql uses storage-related functionality. In addition, there is a great deal of interaction between these topics such as authentication, storage management and source code files that use these topics. Therefore the source code file can be considered as a combination of these domain titles.

Incorporating LDA into source code now leads to software mapping of source software for the LDA model, which is described in Table 1. Given this map,

|  |  |
| --- | --- |
| LDA Model | Source Code Entities |
| word | We define domain specific keywords extracted from names of program elements such as functions, files, data structures and comments to be the vocabulary set with cardinality *V* . A word *w* is an item from this vocabulary. |
| document | A source code file becomes a document in LDA parlance. For our purpose, we represent a document  **f***d* = (*w*1*,w*2*,...,wN*)  to be a sequence of *N* domain specific keywords. |
| corpus | The software system  S = {**f**1*,***f**2*,...,***f***M*}  having *M* source code files forms the corpus. |

LDA source code boards

Name We define certain domain keywords derived from the names of system objects such as functions, files, data structure and comments to become words set by cardinal V. The word w is an object from this vocabulary.

document The source code file becomes a document with LDA terminology. For our purpose, we represent the document

fd = (w1, w2, ..., wN)

to be a sequence of key N domain names.

corpus Software program

S = {f1, f2, ..., fM}

having source code files M form corpus.

The LDA feature for obtaining corpus code is not difficult. Given a software program containing a collection of source code files, domain-related names are extracted from each source code file. Using this, the source code file-word matrix is ​​created where source code files form rows, domain names form columns and each cell represents a weighty occurrence of the word, representing the column, in the source code file representing the line. This source code matrix file name is provided as an input to LDA. The result of LDA is a collection of topics and distribution of these topics in each source code file. A title is a set of domain names and the significance of each word in the title represented as

fraction number.

3. USE/IMPLEMENTATION

We used a static code analysis tool. Figure 1 shows the part of this tool that deals specifically with title delivery, identification of the subject area [5, 24], visualizing the distribution of the title and analyzing the situation according to the domain titles.

The key input for LDA-based title rendering is matrixwordword wd [w, fd] = η where η is the value that indicates the value of the word w in fd file. We will soon explain how to make η according to the number and place of w in fd. Our current implementation uses the Gibbs sampling method [10] using the markov chain monte carlo method to convert to a target

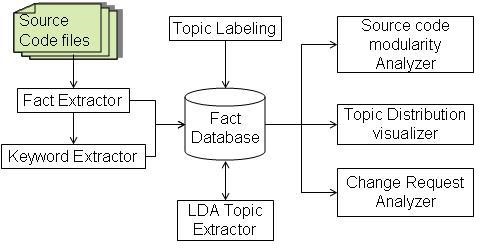


Figure 1: Drawing Tool Block

repeated distribution. A detailed description of this method does not exist in this category.

Input parameters.

Input parameters.

Our tool, based on the LDA process, takes two parameters α and β (as described in paragraph 2.1.1) and creates a distribution of word-over-word and kwezi-word-distribution over text. In addition to the parameters α and β the tool also requires a multiplication value. Remember that the LDA defines the topic as the allocation of opportunities in all policies in the corpus. Define the user limit used to identify the most important words in each topic.

1.1 Keyword deletion

In order to build an LDA word document matrix, it is very important to find domain-related words in the source code. The methods used by the LDA in text-based texts look at each word in the text for this purpose. However, unlike plain text document the source code file is an edited text and it is never wrong to assume that each word in the file will be related to the domain. First, a large percentage of words in source code files form the syntax of a programming language, such as, if, category, return, time and so on. In addition, domain keywords are often included among the identifiers such as subtitles and identifiers that need to be separated separately in order to exclude appropriate sub-keywords. Given this idea we suggest the following steps to extract keywords from source code files:

1. True Release.

2. Separation of identification and analysis by key words.

3. Enter keywords in their common communities.

Filtering keywords to eliminate keywords that do not reflect any business idea.

True Release.

True output is the process of analyzing source code documents and extracting interesting meta data such as files, tasks, performance dependencies, data structure etc. We used the source sailor [1] to extract the facts from the source code.

Identity separation.

Logical classification of sub-words is important because unlike natural language text where each word is independent and found in a dictionary, source code identification words are usually not single words but logical sequence of letters and acronyms separated by a particular letter or word meeting. For example, a function called "add auth details" to httpd-2.0.53 source code creates three audible pieces of characters "add", "auth" and "info" determined by "".

It is the word we use for the letters that sound “keyword”. Each identifier is divided into groups of keywords. In order to make this distinction it is important to know how to identify words. There are many programs such as underscore, hyphen, or capitalization (camel) as in "getLoanDetails". We have implemented a program to separate adjectives in Perl.

It stops.

Often the keywords of the source code are used in the singular and plural. For example "loans" and "loans". In our analysis we do not need to take them as two separate keywords. So we combine all keywords by adding them to their default roots. Also, It is a common practice [23, 12] to invent words in order to improve the results of analysis. We used Porter’s stemming algorithm [20] to insert all the words into common roots.

Sorting.

Not all keywords are subject to domain names. For example keywords such as "find" and "set" are very common and a stop list is started to filter such words.

1.2 Keyword map

After finding a collection of different program names provided by the software, we now calculate the wd matrix. For this purpose, we enter the nearest number of the word w fd fd as follows:

1. Defines heuristic weight

λ: {lt} → ℵ

which provides the total user-defined value in the “locationtype” lt. "Location type" lt defines a portion of the source code file such as file name, username, valid parameter names, comments, data structure name and so on when keyword is found. The weight given to the name derived from the function name will differ from the name obtained from the name of the data structure.

2. The wd [w, fd] value of the word w from the fd file is calculated from the estimated total frequency of each event type in the source code fd file. That is, wd [w, fd] = Xλ (lti) × ν (w, fd, lti)

lti

where ν (w, fd, lti) defines the frequency of the word w in the local area lti of the source file fd.

To show wd value calculations consider the following code snippets from the OrderDetails.java file

the OrderDetails community section uses Java.io.Serializable {

Private String Order; String for private user; Private String OrderDate; private floatValue building; String Private Order Status;

GetOrderStatus () public String {

return (order Status);

} ...

...

} As mentioned in section 4.1, identifiers are categorized to determine the domain name that makes sense and the importance of counting each word. One such name extracted from the caption code above is "Order" which is found in comments and names of the identification type such as class name, adjective name and path name. These different types of word sources form our types of places lt. Usually, in something

targeted program, classes that represent domain objects and their names are more likely to produce important domain names in that category. Therefore, λ (class) is usually given a higher value by the domain specialist than λ (attribute). Let us assume that in this particular case λ (paragraph) is equal to 2, λ (attribute) is equal to 1 and λ (method) is equal to 1. The essence of the word "Order" in the caption code above as calculated according to the formula given above it is 7 p.m.

wd [Order, OrderDetails.Java] = 2 + 1 + 1 ∗ 4 + 1 ∗ 1 = 7

Similarly, weighted events are calculated by other words such as “details”, “user” and “status”.

3.1 Title label

The LDA could not adequately find a comprehensible human label on a specific topic. In many cases, the words that can be labeled are simply abbreviations of business ideas or acronyms. As a result it becomes difficult to create a logical title label automatically. In the current version of the tool, the identified topics are labeled manually.

4. CASE STUDIES

We have tested the way we work on many open source processes and related programs. Throughout this section we discuss the results obtained using some of the topics as examples.

Apache title domain

We have downloaded Apache 30 articles. Due to the brevity we only mention two titles, namely "SSL" and "Logging". Table 1 (a) lists the top keywords for “SSL” and the chances associated with the occurrence of a random keyword appearing in the heading “SSL”.

Our tool can extract not only domain titles, but also infrastructure-level articles and cutting-edge articles. For example, "login" is a topic that cuts files and modules. Our tool, based on LDA, is able to put together all the keywords related to the login as shown in Table 1 (b) which lists the top keywords in the "Login" and their corresponding values.

(a) A title labeled as SSL (b) A title labeled Login

Possibility of keyword Possibility of keyword

ssl 0.373722 log 0.141733

expr application 0.042501 .036017

Init 0.033207 mod 0.0311

0.026447 config engine 0.029871

var 0.022222 name 0.023725

ctx 0.023067 articles 0.021266

ptemp 0.017153 autoindex 0.020037

mctx format 0.013773 0.017578

looking at 0.012083 cmd 0.01512

modssl 0.011238 header 0.013891

ca 0.009548 add 0.012661

Table 2: Sample titles extracted from the Apache source code

4.2 Background Petstore Title

In order to investigate the effect of naming the results of the title release we have examined Petstore, a J2EE system used by Sun Microsystems. If J2EE is a reliable use, it follows good java naming conventions and a large number of pointers have logical names.

(a) An title labeled as Con- (b) An title labeled Adtact Information dress Information

Possibility of keyword

details 0.418520

contact 0.295719

email 0.050116

address 0,040159

family 0.040159

given 0.036840

phone 0.026884

by 0.000332

Possibility of keyword

address 0.398992

road 0.105818

city ​​0.055428

code 0.055428

country 0.055428

zip 0.055428

name1 0.050847

status 0.046267

name2 0.046267

end 0.005039

add 0.009548

Table 3: Sample titles extracted from the petstore source code

As shown in Table 2 (a) we are able to successfully combine all words related to "contact details". However, the most important thing in this example is that the keywords "info", "contact" are logical and indicate a possible title of the title. For example if we combine these two keywords in the "information contact" it can be considered as a valid label for the "contact information" title.

Similarly, even if there is a "address details" title, shown in Table 2 (b), the combination of the key words "address" and "street" can be used to name the "address details". It can be seen in the sample articles released that a good naming convention produces more logical words thus simplifying the process of labeling titles.

4.3 Decision of Synonymy and Polysemy

One of the most important factors in extracting relevant headings and collections of related keywords by combining them together is the ability of the algorithm used to solve different keywords with the same meaning. We have seen that our tool is capable of resolving synonyms satisfactorily to the best of our ability since the LDA models the titles in the file and the words in the title using multiple opportunity distribution. For example, consider the topic marked “transaction” on PostgreSQL shown in Table 5.3. The LDA found that “performance” and “xact” are the same and grouped together in the same collection as shown below.

Possibility of keyword

made at 0.149284

space name 090856

make 0.035349

xact 0.035349

visible 0.029506

the current 0.029506

abortion 0.026585

words 0.026585

command 0.023663

start 0.020742

method 0.017821

Table 4: Transactions and Xact Synonymy resolution made by LDA

Most interestingly, our tool was able to solve the same polysemy words with a different meaning to the source code. Polyseme can appear in many domain topics depending on the context. The reason for our polyseme detection tool is not difficult to understand. Note that the LDA is modeling the topic as a policy allocation; it is therefore perfectly permissible for a word to appear in two headings with different values ​​of existence. In addition, the LDA attempts to include the subject of a given term information with word context information, e.g. For example, in the Linux-kernel source code we noticed that the word "volume" is used in the context of audio control and in the context of file systems. The LDA is able to distinguish between these different uses of the term and collect the same word on different topics.

# DISCUSSION

In this section we discuss various factors that impact the results obtained. Subsequently we will discuss benefits and limitations of our approach.

## Effect of number of Topics

Our approach for topic extraction accepts the number of topics to be extracted as an input from the user. We have observed that varying the number of topics has a significant impact on polysemy resolution. For instance, consider the example of polysemy resolution of the keyword“volume” in Linux-kernel, discussed in subsection 5.3. We have conducted our experiment on Linux-kernel source code twice. In both the times we have kept all the parameters, namely *α*, *β* the number of iterations and the cut-off threshold Ψ same except for the number of topics. In the first experiment the number of topics *T* was set to 50 and in the second experiment *T* was set to 60. In both these experiments, of the total topics extracted two topics were“sound”related topics and one topic for“file systems”. Table 6.1 lists the probabilities of keyword“volume”in“sound”and“file systems”topic for both the experiments.

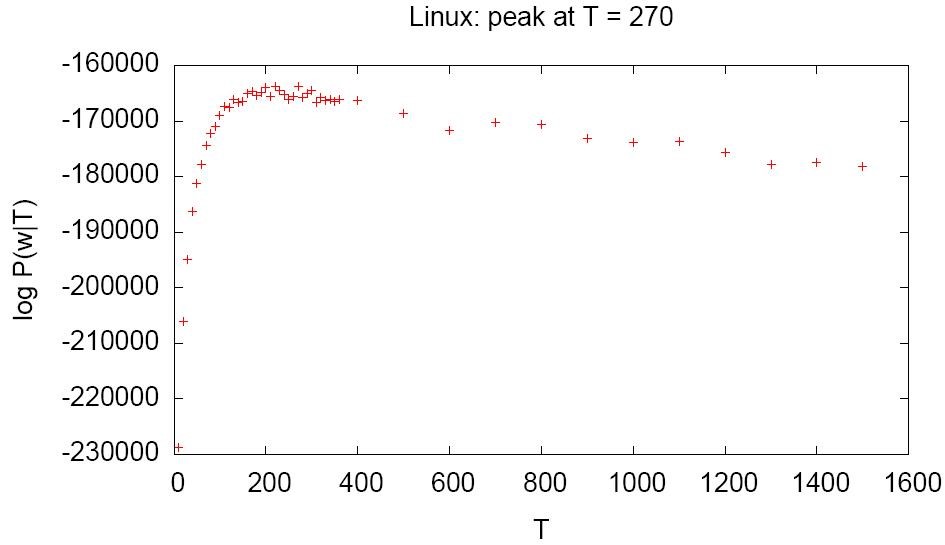
|  |  |  |
| --- | --- | --- |
| Topic type | ’Volume’ probability for Experiment1 with T=50 | ’Volume’ probability for Experiment2 with T=60 |
| Sound topic 1 | 0.024 | 0.032 |
| Sound topic 2 | 0.009 | 0.009 |
| file systems topic | *<* 0.0002 | 0.004 |

**Table 5: Effect of number of topics on polysemy resolution in Linux-kernel**

In our experiments we have used a value of the threshold Ψ to be 0*.*001 for determining whether a keyword belongs to a topic or not. If the probability of a keyword associated with a topic is less than 0*.*001 then we do not consider that keyword as indicator of that topic. In view of this, it can be observed from the table 6.1 that in experiment 1 the keyword “volume” has a probability of less than 0*.*0002 for topic “file systems”. Hence “volume” is associated with only the two sound related topics and not with the“file systems” topic. However, in experiment 2 when the number of topics was increased to 60, the probability of “volume” for topic “file systems” is 0*.*004. This probability is greater than our threshold 0.001 and hence “volume” is also considered as an indicator for “file systems” topic apart from the sound related topics. The polysemy in the keyword “volume” is revealed only in the second experiment with 60 topics.

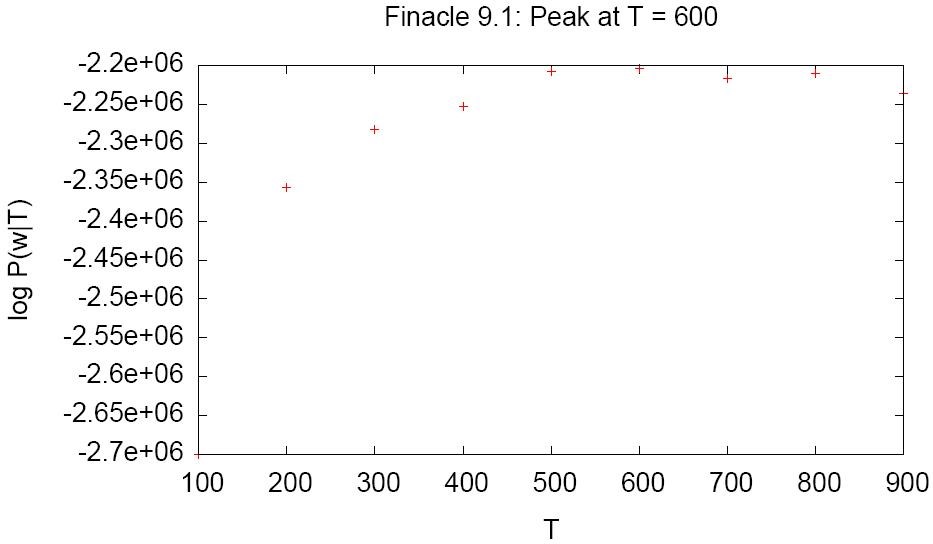
## Discovering optimal number of Topics

The problem of identifying optimal number of topics is not specific to source code alone. Topic extraction from text documents faces a very similar problem. Griffiths and Steyvers recommend trying different numbers of topics T and suggest using the *maximum likelihood* method on *P*(*w*|*T*) [10]. Applying this technique to extract topics from Linux suggests that the optimal number of topics in the case of Linux is 270 as shown in figure 2.



**Figure 2: Inferring optimum number of topics for Linux**

However, automatically inferring the number of topics by maximizing the likelihood is not without problems.



**Figure 3: log-likelihood graph for our proprietary system**

We applied our tool to extract topics for a very large proprietary business application having multi-million lines of C code. The number of topics predicted using the likelihood method was much larger than what the architects and domain experts of the proprietary system considered reasonable. As shown in figure 3, likelihood peaked at *T* = 600 suggesting that the optimal number of topics for our system was 600. However, the architects and domain experts felt that the resonable number of topics to be around 100.

## Effect of *α* and *β*

As discussed in section 2.1.1, the LDA model accepts two parameters *α* and *β*. *α* controls the division of documents into topics and *β* controls division of topics into words. Larger values of *β* yield coarser topics and larger values of *α* yields coarser distribution of document into topics. Hence the right value of *α* and *β* is needed to derive good quality topics and assignment of topics to documents.

Some of the implementations of LDA estimate these values on-the-fly while other implementations rely on the user to provide appropriate values. In our implementation the values of *α* and *β* needs to be provided by the user.

## Human Intervention

Even though LDA based topic extraction presented in this paper is automatic and unsupervised, we believe that human intervention is necessary in a number of aspects to achieve results of acceptable quality. In this subsection we point out areas of our method which would be helped by human intervention.

*Expert delineated Keyword Filtering:.*

Both keyword extraction and subsequent filtering has impact on the quality of the results obtained. The vocabulary of source code is much smaller than that of natural language text corpus and hence the effect of missing or incorrect terms is much stronger. Also, in the context of keyword extraction from identifiers in the program, we have observed that not all identifiers are equally good indicators of the business topics. For this purpose we have introduced the weighing scheme *λ* as described in Section 4.2 where an expert, for instance, can assign more weight to file names over say comments in the program. Human intervention can also improve the filtering of keywords by identifying infrastructure and domain specific stop words. For instance “EJB”, “get”, “set” are some of the common keywords which needs to be filtered out.

*Number of Topics:.*

As discussed in section 6.2, the log-likelihood method for estimating the number of topics is not always appropriate and in a number of cases the number of topics is better supplied by domain experts and architects of the system. During our experiments we have observed that one needs to try different number of topics and repeat the topic extraction process to get a set of topics of acceptable quality.

*Topic validation and labeling:.*

In our experience topic extraction has been an iterative process. Topics extracted initially are evaluated and based on the results keyword extraction and filtering heuristics are updated and the *α* and *β* parameters varied to extract better topics. It is difficult to automatically evaluate the quality of topics obtained. We needed a domain expert who can manually examine the cluster of terms and check if it truly represents a domain topic. Moreover, when a domain topic has been identified labeling has to be done manually.

# CONCLUSION AND FUTURE WORK

In this paper we have investigated the effectiveness of the LDA in the context of the comprehension program and suggested a LDA-based approach to subject-coding the source code. A review of several open sources and affiliate programs has shown that our tool is able to adequately extract some but not all domain titles. We also saw the need to include something personal in order to improve the quality of published articles.

Another disadvantage of the LDA is that it does not find correlations between published titles or identifying topics at a different level of granularity. As part of our future work we plan to investigate ways of extracting topics at different levels of granularity and to identify the different relationships between them. we intend to compare the alternative methods of publication with the LSA with the LDA-based approach presented here. Finally, We believe that the proposed LDA-based approach to the issuance of a business title is promising and ensures continuous investigation and validation.

# REFERENCES

1. Source navigator 5.4.1. http://sourcenav.sourceforge.net, 2003.
2. P. Anderson and M. Zarins. The codesurfer software understanding platform. In *IWPC*, pages 147–148. IEEE Computer Society, 2005.
3. N. Anquetil and T. C. Lethbridge. Recovering software architecture from the names of source files. *Journal of Software Maintenance: Research and Practice*, 11:201–221, 1999.
4. G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions in Software Engineering*, 28(10):970–983, 2002.

T. J. Biggerstaff, B. G. Mitbander, and D. Webster. Program understanding and the concept assignment problem.