# Latent Dirichlet Allocation: Extracting Topics from Software Engineering Data

Joshua Charles Campbell, Abram Hindle, and Eleni Stroulia
{joshua2,hindle1,stroulia}@ualberta.ca

July 16, 2014

**Abstract**

Topic analysis is a powerful tool that extracts "topics" from document collections. Unlike manual tagging, which is effort intensive and requires expertise in the documents' subject matter, topic analysis (in its simplest form) is an automated process. Relying on the assumption that the document collection refers to a small number of topics, it extracts bags of words attributable to these topics. These topics can be used to support document retrieval or to relate documents to each other through their associated topics. Given the variety and amount of textual information included in software repositories, in issue reports, commit and source-code comments, and other forms of documentation, this method has found many applications in the software-engineering field of mining software repositories.

This chapter provides an overview of the theory underlying LDA (Latent Dirichlet Allocation), the most popular topic-analysis method today. Next it illustrates, with a brief tutorial introduction, how to employ LDA on a textual data set. Third, it reviews the software-engineering literature for uses of LDA for analyzing textual software-development assets, in order to support developers' activities. Finally, we discuss the interpretability of the automatically extracted topics, and their correlation with tags provided by subject-matter experts.

## 1 Introduction

Whether they consist of code, bug/issue reports, mailing-list messages, requirements specifications, or documentation, software repositories include text documents. The simplest approach to analyzing textual document is using the vector-space model (VSM), which views documents (and queries) as frequency vectors of words. For example "the" occured once, "my" occured twice, "bagel" occured 0 times, and so on. Effectively, VSM views terms as dimensions in a high-dimensional space, so that each document is represented by a point in that space based on the frequency of terms it includes. This model suffers from two major shortcomings: (a) it makes the consideration of all works impractical and

(b) it assumes that all words are independent. In response to these two problematic assumptions, methods for extracting topic models, i.e., thematic topics corresponding to related bags of words, were developed. The most well known topic-model methods are Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA). LSI employs singular value decomposition to describe relationships between terms and concepts as a matrix.

LDA has recently emerged as the method of choice for working with large collections of text documents. There is a wealth of publications reporting its applications in a variety of text-analysis tasks in general and software engineering in particular. LDA can be used to summarize, cluster, link, and preprocess large collections of data because it produces a weighted list of topics for every document in a collection dependent on the properties of the whole. These lists can then be compared, counted, clustered, paired, or fed into more advanced algorithms. Furthermore, each topic is comprised of a weighted list of words which can be used in summaries.

This chapter provides an overview of LDA and its relevance to analyzing textual software-engineering data. First, in Section 2, we discuss the mathematical model underlying LDA. In Section 3, we present a tutorial on how to use state-of-the-art software tools to generate an LDA model of a software-engineering corpus. In Section 4, we discuss some typical pitfalls encountered when using LDA. In Section 5, we review the mining-software repositories literature for example applications of LDA. Finally, in Section 6, we conclude with a summary of the important points one must be aware of when considering using this method.

## 2 How LDA Works

The input of LDA is a collection of documents and a few parameters. The output is a model consisting of weights which can be normalized to probabilities. These probabilities come in two types: (a) the probability that a specific document generates a specific topic at a position, and (b) the probability that a specific topic generates a specific word from the collection vocabulary. After step (a), the document contains a list of topics (often repeated), which become words after step (b).

LDA is a generative model. This means that it works with the probability of observing a particular dataset given some assumptions about how that dataset was created.

At its core is the assumption that a document is generated by a small number of "topics." An LDA "topic" is a probability distribution, assigning each possible word a probability. Topics are considered hypothetical and unobservable, which is to say that they don't actually exist in documents. Instead, we must infer the topic characteristics from a collection of documents. Each document is assumed to be generated by a few of these topics. So, every word in every document is assumed to be attributable to one of the document's topics.

In LDA, words are discrete objects and have no particular order, each word is

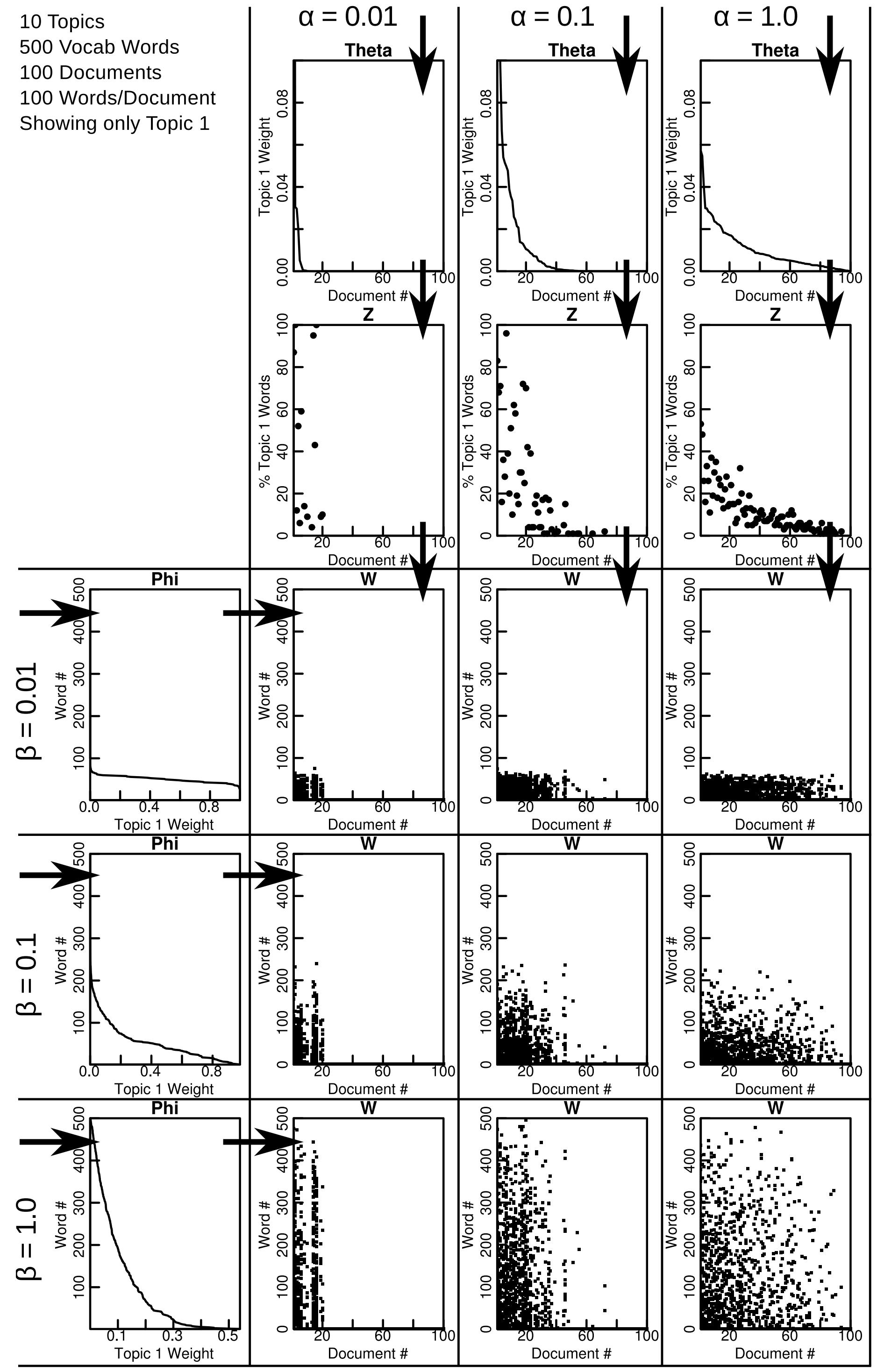

Figure 1: 9 Example LDA models produced by varying $\alpha$ and $\beta$. Arrows show the relationship between prior and posterior.

assigned an individual probability of being generated. Though, of course words do a have a particular order in a document, LDA does not consider their order. That is, the probability of a topic $k$ generating a word $v$ is a value $\phi_{k,v}$ [24]. The sum of these probabilities for a topic $k$ must be 1, which is to say, that a topic generates one word at time. However, most LDA software doesn't work with probabilities directly, but rather, weight vectors which require normalization. This is a difference in the software's internal representation of the distrubtion only, and can be converted to standard probabilities by dividing each word weight by the sum of all word weights.

$$\sum_{v} \phi_{k,v} = 1$$

Furthermore, the $\phi_{k,v}$ values are assumed to come from a random variable $\phi_k$, with a symmetric Dirichlet distribution (which is the origin of the name of the LDA method). The symmetric Dirichlet distribution has one parameter, $\beta$, which determines whether a topic is narrow, i.e., focuses on a few words, or broad, i.e., covers a bigger spread of words. If $\beta$ is 1, the probability of a topic generating a word often is the same as the probability of a topic generating a word rarely. If $\beta$ is less than 1, most words will be extremely unlikely while a few will make up the majority of words generated. In other words, larger values of $\beta$ lead to broad topics and smaller values of $\beta$ lead to narrow topics.

In summary, words are assumed to come from topics with the probability of a word coming from a specific topic coming from a Dirichlet distribution. Thus, if we know that $\beta$ is a very small positive integer, we know that a topic which would generate any word with equal probability is itself very unlikely to exist.

Another important concept involved in understanding how LDA works is that of "document." A document is also a probability distrubtion. Every possible topic has a probability from 0 to 1 of occuring, and the sum of these probabilities is 1. The probability that a document will generate a word from a specific topic $k$ is $\theta_{d,k}$. Again, the $\theta_{d,k}$ probabilities are real values, but the probabilities of $\theta_{d,k}$ taking on a particular value comes from a Dirichlet distribution of parameter $\alpha$. The topics that a document is observed to generate are a vector, $Z_d$. This vector is $N_d$ words long, representing every word in the document. If $\alpha$ is near 1, we expect to see documents with few topics and documents with many topics in equal proportion. If $\alpha$ is less than one, we expect most documents to only use a few topics. If $\alpha$ is greater than one, we expect most documents to use almost every topic.

To summarize, words come from topics. The probability of a word being generated by a specific topic comes from a symmetric Dirichlet distribution. The probability of a document containing a word from a specific topic is dictated by a different symmetric Dirichlet distribution. The words that a document is observed to generate are a vector, $W_d$, which is formed by observing the topic indicated by the entries in the $Z_d$ vector.

Figure 1 shows the words present in each document coming from topic 1 in 9 different LDA models of varying parameters. Additionally, it shows $\alpha$, $\beta$, $\theta$,

$\phi$, $Z$ and $W$. By following the arrows we can see how each prior generates each observed posterior. Figure 1 clearly shows the effects that $\alpha$ has on $\theta$, that $\theta$ has on $Z$, and that $Z$ has on $W$. As we increase $\alpha$, we observe that more documents contain words from topic 1. Additionally, it shows the effect that $\beta$ has on $\phi$ and that $\phi$ has on $W$. As we increase $\beta$, we end up with topic 1 including a larger variety of vocabulary words.

The LDA process consists of allocating and re-allocating weights (or probabilities if normalized) in $\theta$ and $\phi$ until the lower bound of the total probability of observing the input documents is maximized. Conceptually, this is accomplished by dividing up topics among words and by dividing up documents among topics. This iterative process can be implemented in many different ways.

## 2.1 Technical Details of LDA

The generative process LDA assumes is, given a corpus of $M$ documents, each of length $N_i$ [4]:

1. For every topic $k \in \{1, \ldots, K\}$:
   (a) Choose $\vec{\phi_k} \sim \mathrm{Dir}(\beta)$.
2. For every document $d \in \{1, \ldots, M\}$:
   (a) Choose $\vec{\theta_d} \sim \mathrm{Dir}(\alpha)$.
   (b) For every word $j \in 1, \ldots, N_d$ in document $d$:
      i. Choose a topic $z_{d,j} \sim \mathrm{Multinomial}\left(\vec{\theta_d}\right)$.
      ii. Choose a word $w_{d,j} \sim \mathrm{Multinomial}\left(\vec{\phi_{z_{d,j}}}\right)$.

Thus, probability of a topic $k$ generating a word $v$ at a position $j$ in a document $d$ is $p\left(w_{d,j} = v \mid \alpha, \beta, K\right)$:

$$\int_{\Theta} \sum_{k=1}^{K} \int_{\Phi} p\left(w_{d,j} = v \mid \vec{\phi_k}\right) p\left(z_{d,j} = k \mid \vec{\theta_d}\right) p\left(\vec{\phi_k} \mid \beta\right) p\left(\vec{\theta_d} \mid \alpha\right) d\vec{\phi_k} d\vec{\theta_d},$$

integrating over all possible probability vectors of length $K$ ($\Theta$) and of length $V$ ($\Phi$). The goal of LDA software is to maximize the probability

$$p\left(\theta, Z \mid W, \alpha, \beta, K\right)$$

by choosing $\theta$ and $Z$ given a corpus $W$ and parameters $\alpha$ and $\beta$. Unfortunately, this problem is intractable [4], so the $\theta$ and $\phi$ that maximize the above probability are estimated by LDA software. The exact techniqe employed to estimate the maximum varies between different pieces of software.

## 3 LDA Tutorial

In this tutorial, we illustrate the LDA method in the context of analyzing textual data extracted from the issue-tracking system of a popular project.

1. The first task involves acquiring the issue-tracker data and representing it in a convenient format.
2. Then we analyze the text of the input data, namely we format the text as word counts, where words are represented as integer IDs.
3. Once we have filtered and transformed the text, we apply LDA software to produce a topic-document matrix and a topic-word matrix.
4. We then summarize the top words from the topic-word matrix to produce topic-word summaries and store the topic-document matrix.
5. Finally, we analyze the document matrix and the topics. The objective of this analysis step is to (a) examine the latent topics discovered; (b) plot the topic relevance over time; and (c) cluster the issues (i.e., input documents) according to their associated topics.

### 3.1 Materials

This tutorial will use source code that the authors have developed to run LDA on issue tracker issues. To access the source code of the tutorial visit the following URL `http://bitbucket.org/abram/lda-chapter-tutorial/` and git clone that project, or download a zipfile of the tutorial data and source code from `http://webdocs.cs.ualberta.ca/~hindle1/2014/lda-chapter-tutorial.zip`. The data directory contains the issue tracker data for the `bootstrap` project. While the important source code files are `lda_from_json.py` which depends on `lda.py`. We use `lda_from_json.py` to apply the LDA algorithm on issue tracker issues.

### 3.2 Acquiring Software Engineering Data

The data source for this tutorial will be the issues and comments of the Bootstrap [1] issue tracker. Bootstrap is a popular JavaScript-based website front-end framework that allows preprocessing, templating and dynamic-content management of web pages. Bootstrap is a very active project, and its developer community is regularly reporting issues regarding web browser compatibility and developer support. As of March 2014, Bootstrap had 13182 issues in its issue tracker.

Our first task is to acquire the issue-tracker data for Bootstrap. To achieve this result we have written a Github issue tracker extractor that relies on the Github API and the Ruby Octokit library. Our program `github_issues_to_json.rb` (included in the chapter tutorial repository) uses the Github API to

[1] http://getbootstrap.com/

download the issues and comments from the Github issue tracker. One must first sign up to Github as a registered user and provide the `GHUSERNAME` and `GHPASSWORD` in the `config.json` file in the root of the chapter repository. One can also specify `GHUSER` (target Github user) and `GHPROJECT` (target Github user's project to mirror) in the `config.json` or as an environment variable. `github_issues_to_json.rb` downloads issue tracker data and every page of issues and issue comments. It saves this data to a JSON file, resembling the original format obtained from the Github API. The JSON file created, `large.json`, contains both issues and comments, stored as a list of JSON objects (issues), each of which contains a list of comments. Mirroring Bootstrap takes a couple of minutes due to the thousands of issues and thousands of comments within Bootstrap's issue tracker.

Once we have downloaded the Bootstrap issues (and comments) into `large.json` we need to load and prepare that data for LDA. Most LDA programs will require that documents are preprocessed.

## 3.3 Preprocessing

In this section we will cover preprocessing the data for LDA. Generally those who use LDA apply the following proprocessing steps:

- Loading text
- Mapping text into final textual representation
- Lexical analysis of the text
- Optionally removing stop words
- Optionally stemming
- Building a vocabulary
- Optionally removing uncommon or very common words
- Mapping each text document into a word-bag

### 3.3.1 Loading Text

Loading the text is usually a matter of parsing a data file or querying a database where the text is stored. In this case, it is a JSON file containing Github issue tracker API call results.

### 3.3.2 Mapping Text

The next step is to transform the text into a final textual representation. This will be the textual representation of the documents. Some text is structured and thus must be processed. Perhaps section headers and other markup needs to be removed. If the input text is raw HTML perhaps one needs to strip HTML from the text before use. For the issue tracker data we could include author names in the comments and in the issue description. This might allow for author-oriented topics, but might also confuse future analysis when we notice there was no direct mention of any of the authors. In this tutorial we have chosen to concatenate the title and the full description of the issue report, so that topics will have access to both fields. Many uses of LDA in software analysis includes sourc code in the document texts. Using source code requires lexing, parsing, filtering and often renaming values. When feeding source code to LDA, some users do not want comments, some do not want identifiers, some do not want keywords, and some want only indentifiers. Thus, the task of converting documents to a textual representation is non-trivial, especially if documents are marked up.

### 3.3.3 Lexical Analysis

The next step is lexical analysis of the texts. We need to split the words or tokens out of the text in order to eventually count them. Once the texts have been loaded and processed we need to split the texts into tokens. With source code this requires lexical analysis, where one extracts tokens from source code in a similar fasion to how compilers perform lexical analysis before parsing. With natural language text one wants to separate words and punctuation where appropriate. For example, some words, such as initialisms, contain periods but most of the time a period indicates the end of a sentence and is not a part of the word. With texts about source code, it might be useful to have some tokens start with a period, for instance if you are analyzing CSS (cascading style sheets) or texts with CSS snippets, where a period prefix indicates a CSS class.

### 3.3.4 Stop Word Removal

Often words appear in texts that are not helpful to topic analysis. Such words are called stop words. It is common in Natural Language Processing (NLP) and Information Retrieval (IR) systems to filter out stop words before executing a query or building a model. Stop words are words that are not relevant to the desired analysis. Whether a word is considered a stop word or not is dependant on the analysis, but there are some sets of common stop words available. Some users of NLP and LDA tools view terms such as “the”, “at”, and “a” as unnecessary, where as other researchers, depending on the context, might view the definitives and prepositions as important. We have included `stop_words`, a text file that contains various words that we do not wish to include in topics in this tutorial. For each word extracted from the document we remove those found within our stop word list.

### 3.3.5 Stemming

Since words in languages like English have multiple forms and tenses it is common practice to *stem* words. Stemming is the process of reducing words to their original root. Stemming is optional and often used to reduce vocabulary sizes. For instance, given the words act, acting, acted and acts, the stem for all 4 words will be act. Thus if a sentence contains any of the words upon stemming it will resolve to the same stem. Unfortunately, sometimes stemming reduces the semantic meaning of a term. For example, "acted" is in the past tense, but this information will be lost if stemmed. Stemming is not always necessary.

Stemming software is readily available. NLTK [2] comes with an implementation of the Porter stemmer and Snowball stemmers. One caveat with stemmers is they often produce word roots that are not words or conflict with other words. Sometimes this leads to unreadable output from LDA unless one keeps the original documents and their original words.

#### 3.3.6 Common and Uncommon Word Removal

Since LDA is often used to find topics it is common practice to filter out exceptionally common words and infrequent words. Words that appear in only 1 document are often viewed as unimportant, because they won't form a topic with multiple documents. Unfortunately if very infrequent words are left in, some documents which do not contain the word will be associated with that word via the topics that include that word. The common words are often skipped because they muddle topic summaries and make interpretation more difficult.

Once the documents are preprocessed and prepared via lexing, filtering, and stemming we can start indexing them for use as documents within a LDA implementation.

## 3.4 Applying LDA

In this tutorial, we use Vowpal Wabbit by Langford et al. [13]. Vowpal Wabbit accepts a sparse document word matrix format where one line represents a document and each element of the line is an integer word joined by a colon to its count within that document. We provide `lda.py`, a program to convert text to Vowpal Wabbit's input format, and parse its output format. One difficulty encountered using LDA libraries and programs is that often you have to maintain your own vocabulary or dictionary.

We provide our parameters to Vowpal Wabbit: $\alpha$ set to 0.01, $\beta$ set to 0.01 (called $\rho$ in vowpal wabbit), and $K$, the number of topics. 0.01 is a common default for $\alpha$ and $\beta$ in many pieces of LDA software. These parameters should be chosen based on the desired breadth of documents and topics, respectively. If documents that only discuss a few topics and never mention all others are desired, $\alpha$ should be set small, to around $1/K$. With this setting, almost all documents will almost never mention more than a few topics. Inversely, if documents that discus all topics but focus on some more than others are desired, $\alpha$ should be set closer to 1. With this setting, almost all documents will discuss almost every topic but not in equal proportions. Setting $\beta$ is similar to setting $\alpha$ except that it controls the variety words each topic will mention.

[2]NLTK: http://www.nltk.org/howto/stem.html

We must also calculate and provide the size of the vocabulary as $\lceil log_2(|words|)\rceil$. We choose 20 for the number of topics for the sake reading and interpreting the topics. The number of topics depends on the intent behind the analysis. If one wants to use LDA for dimensionality reduction perhaps keeping the number of topics low is important. If one wants to cluster documents using LDA a larger number of topics might be warranted. Conceptual coupling might be best served with many topics over fewer topics.

## 3.5 Vowpal Wabbit

Vowpal Wabbit reads the input documents and parameters and outputs a document topic matrix and a topic word matrix. `predictions-00.txt` where `00` is the number of topics, is a file containing the document topic matrix. Each document is on one line, and each row is the document topic weight. If multiple passes are used, the last $M$ lines of `predictions-00.txt`, where $M$ is the number of documents, are the final predictions for the document topic matrix. The first token is the word ID and the remaining $K$ tokens are the allocation for each topic (topics are columns).

## 3.6 Tutorial LDA Output

Our program `lda.py` produces `summary.json` a JSON summary of the top topic words for each topic extracted, ranked by weight. 2 other JSON files are created, `document_topic_matrix.json` and `document_topic_map.json`. The first file (matrix) contains the documents and weights represented by JSON lists. The second file (map) contains the documents by their ID mapped to a list of weights. `document_topic_map.json` contains both the original ID and the document weight where as the matrix uses indices as IDs. `lda-topics.json` is also produced and it lists the weights of words associated with each topic, as lists of lists. `lids.json` is a list of document IDs in the order presented to Vowpal Wabbit and the order used in the `document_topic_matrix.json` file. `dicts.json` maps words to their integer IDs.

### 3.6.1 Semantic Topic Analysis

Since the topics are extracted, let's go take a look! In Table 1 we see a depiction of 20 topics extracted from the Bootstrap project issue tracker. The words shown are the top 10 ranking words from each of the topics, the most heavily allocated words in the topic.

Each topic is assigned a number by LDA software, however the order in which it assigns numbers is arbitrary and has no meaning. If you ran LDA again with different seeds or different timing (depending on implementation) you would probably get different topics or similar topics but in different orders. Nonetheless, we can see in Table 1 that many of these topics are definitely related to the Bootstrap project. Topic summaries such as these are often your first canary in the coal mine: they give you some idea of the health of your

LDA output. If they are full of random tokens and numbers one might consider stripping out such tokens from the analysis. If we look to Topic 20 we see a set of terms: *license org mit apache copyright xl cc spec gpl holder.* MIT, Apache, GPL and CC are all copyright licenses and all of these licenses have terms and require attribution. Perhaps documents related to Topic 20 are related to licensing? How do we verify if Topic 20 is about licensing or not?

Using the document topic matrix we can look at the documents that are ranked high for topic 20. Thus we can load the CSV file, `document_topic_map.csv` or JSON file, `document_topic_map.json`, with our favorite spreadsheet program, R, or Python, and sort by descending order on the `T20` (topic 20) column. Right at the top is issue 2054. Browsing `large.json` or by visiting issue 2054 on Github [3] we can see that the subject of the issue is "Migrate to MIT License". The next issues relate to licensing for image assets (#3942), javascript minification (unrelated, but still weighted heavily toward topic 20) (#3057), phantomJS error (#10811) and two licensing issues (#6342 and #966). The LDA python program also produces a `document_topic_map_norm.csv` that has normalized the topic weights, reviewing the top weighted documents from the normalized CSV film reveals different issues, but 4 of the 6 top issues are still licensing relevant (#11785, #216, #855, and #10693 are licensing related but #9987, and #12366 are not).

### 3.6.2 Visualization

Looking at the numbers and topics is not enough, usually we want to visually explore the data to tease out interesting information. One can use simple tools such as spreadsheets to make basic visualizations.

Common visualization tasks with LDA include:

- Plotting Document to Topic Association over Time
- Plotting the Document Topic Matrix
- Plotting the Document Word Matrix
- Plotting the association between two distinct kinds of documents within the same LDA run.

Given the CSV files, one can visualize the prevalence of topics over time. Figure 2 depicts a chart from LibreOffice's spreadsheet that shows the proportional topic weights of the first 128 issues over time against topics 15 to 20 from Table 1.

Based on the spreadsheet inspection the reader should notice that the document topic weights are somewhat noisy and hard to immediately interpret. For instance, it is hard to tell when a topic is popular and when it comes less popular. Alternatively one might ask is a topic constantly referenced over time or is it periodically popular? One method of gaining an overview is to bin or group

[3] Bootstrap issue 2054 `https://github.com/twbs/bootstrap/issues/2054`

| Topic # | Top 10 topic words |
|---|---|
| Topic 1 | *grey blazer cmd clipboard packagist webview kizer ytimg vi wrench* |
| Topic 2 | *lodash angular betelgeuse ree redirects codeload yamlish prototypejs deselect manufacturer* |
| Topic 3 | *color border background webkit image gradient white default rgba variables* |
| Topic 4 | *asp contrast andyl runat hyperlink consolidating negatory pygments teuthology ftbastler* |
| Topic 5 | *navbar class col css width table nav screen http span* |
| Topic 6 | *phantomjs enforcefocus jshintrc linting focusin network chcp phantom humans kevinknelson* |
| Topic 7 | *segmented typical dlabel signin blockquotes spotted hyphens tax jekyllrb hiccups* |
| Topic 8 | *modal input button form btn data http tooltip popover element* |
| Topic 9 | *dropdown issue chrome menu github https http firefox png browser* |
| Topic 10 | *zepto swipe floor chevy flipped threshold enhanced completeness identified cpu* |
| Topic 11 | *grid width row container columns fluid column min media responsive* |
| Topic 12 | *div class li href carousel ul data tab id tabs* |
| Topic 13 | *parent accordion heading gruntfile validator ad mapped errorclass validclass collapseone* |
| Topic 14 | *bootstrap github https css http js twitter docs pull don* |
| Topic 15 | *left rtl support direction location hash dir ltr languages offcanvas* |
| Topic 16 | *percentage el mistake smile spelling plnkr portuguese lokesh boew ascii* |
| Topic 17 | *font icon sm lg size xs md glyphicons icons glyphicon* |
| Topic 18 | *tgz cdn bootstrapcdn composer netdna libs yml host wamp cdnjs* |
| Topic 19 | *npm js npmjs lib http install bin error ruby node* |
| Topic 20 | *license org mit apache copyright xl cc spec gpl holder* |

Table 1: The top 10 ranked words of the 20 topics extracted from Bootstrap's issue tracker issues.

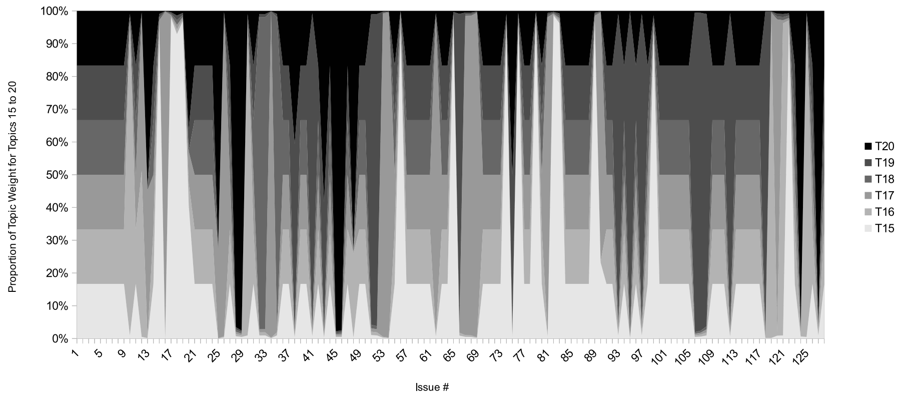

Figure 2: Example of using simple spreadsheet charting to visualize part of the document topic matrix of Bootstrap (topics 15 to 20 of the first 128 issues).

documents by their date (e.g., weekly, biweekly, monthly) and then plot the mean topic weight of 1 topic per time bin over time. This allows one to produce a visualization depicting peaks of topic relevance over time. Within the tutorial distribution we have included an R script called `plotter.R`, that produces a summary of the 20 topics extracted combined with the dates extracted from the issue tracker. This R script produces Figure 3, a plot of the average relevance of documents per two week period over time. This plot is very similar to the plots in Hindle et al. [11]. If one looks to the bottom right corner of the figure at the plot of Topic 20 one can see that Topic 20 peaks up from time to time, but is not constantly discussed. This matches our perception of the licensing discussions found within the issue tracker: they occur when licenses need to be clarified or change, but they do not change all the time. This kind of overview can be integrated into project dashboards to give managers an overview of issue tracker discussions over time.

Further directions for readers to explore is using different kinds of documents, such as documentation, commits, issues and source code, and then relying on LDA's document topic matrix to link these artifacts. We hope this tutorial has helped illustrate how LDA can be used to gain an overview of unstructured data within a repository and infer relationships between documents.

## 4 Potential Pitfalls and Hazards

A typical erroneous assumption frequently made by LDA users is that an LDA topic will represent a more traditional topic that humans write about such as sports, computers or Africa. It is important to remember that LDA topics may not have a valid interpretation. This problem was explored in Hindle et al. [11].

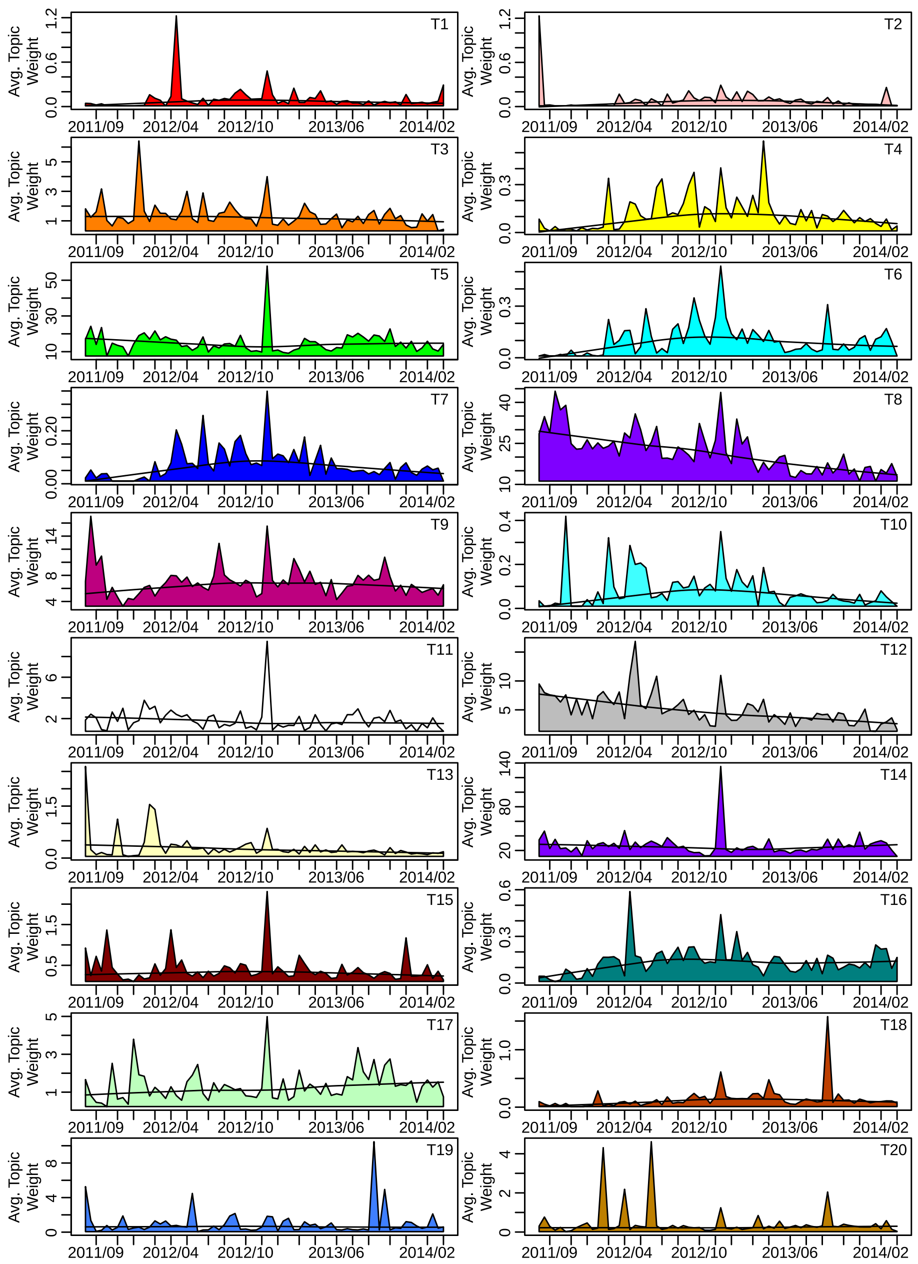

Figure 3: Average Topic Weight for Bootstrap Issues in 2 week bins. The topics are clearly described in Table 1

| Topic # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Topic #1 | 1 $\sim$ 1 | -0.22 $\sim$ 0.17 | -0.21 $\sim$ 0.18 | -0.22 $\sim$ 0.17 | -0.16 $\sim$ 0.24 | -0.11 $\sim$ 0.28 | -0.22 $\sim$ 0.17 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.17 | -0.22 $\sim$ 0.17 |
| Topic #2 | -0.22 $\sim$ 0.17 | 1 $\sim$ 1 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.23 $\sim$ 0.16 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.23 $\sim$ 0.16 | -0.24 $\sim$ 0.15 | -0.23 $\sim$ 0.16 |
| Topic #3 | -0.21 $\sim$ 0.18 | -0.22 $\sim$ 0.17 | 1 $\sim$ 1 | -0.22 $\sim$ 0.18 | -0.22 $\sim$ 0.18 | -0.21 $\sim$ 0.18 | -0.21 $\sim$ 0.18 | -0.21 $\sim$ 0.18 | -0.19 $\sim$ 0.21 | -0.21 $\sim$ 0.18 |
| Topic #4 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.22 $\sim$ 0.18 | 1 $\sim$ 1 | -0.23 $\sim$ 0.16 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.23 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.23 $\sim$ 0.16 |
| Topic #5 | -0.16 $\sim$ 0.24 | -0.23 $\sim$ 0.16 | -0.22 $\sim$ 0.18 | -0.23 $\sim$ 0.16 | 1 $\sim$ 1 | -0.21 $\sim$ 0.19 | -0.23 $\sim$ 0.16 | -0.23 $\sim$ 0.16 | -0.24 $\sim$ 0.16 | -0.11 $\sim$ 0.28 |
| Topic #6 | -0.11 $\sim$ 0.28 | -0.22 $\sim$ 0.17 | -0.21 $\sim$ 0.18 | -0.22 $\sim$ 0.17 | -0.21 $\sim$ 0.19 | 1 $\sim$ 1 | -0.2 $\sim$ 0.2 | -0.22 $\sim$ 0.18 | -0.22 $\sim$ 0.17 | -0.22 $\sim$ 0.18 |
| Topic #7 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.21 $\sim$ 0.18 | -0.23 $\sim$ 0.16 | -0.23 $\sim$ 0.16 | -0.2 $\sim$ 0.2 | 1 $\sim$ 1 | -0.21 $\sim$ 0.18 | -0.21 $\sim$ 0.18 | -0.23 $\sim$ 0.17 |
| Topic #8 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.21 $\sim$ 0.18 | -0.23 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.22 $\sim$ 0.18 | -0.21 $\sim$ 0.18 | 1 $\sim$ 1 | -0.23 $\sim$ 0.16 | -0.22 $\sim$ 0.17 |
| Topic #9 | -0.23 $\sim$ 0.17 | -0.24 $\sim$ 0.15 | -0.19 $\sim$ 0.21 | -0.23 $\sim$ 0.16 | -0.24 $\sim$ 0.16 | -0.22 $\sim$ 0.17 | -0.21 $\sim$ 0.18 | -0.23 $\sim$ 0.16 | 1 $\sim$ 1 | -0.05 $\sim$ 0.33 |
| Topic #10 | -0.22 $\sim$ 0.17 | -0.23 $\sim$ 0.16 | -0.21 $\sim$ 0.18 | -0.23 $\sim$ 0.16 | -0.11 $\sim$ 0.28 | -0.22 $\sim$ 0.18 | -0.23 $\sim$ 0.17 | -0.22 $\sim$ 0.17 | -0.05 $\sim$ 0.33 | 1 $\sim$ 1 |

Table 2: Topic-topic correlation matrix. 95% confidence intervals of the correlation amount. From the same LDA model as Figure 1.

Thus, working with LDA-produced topics has some hazards: for example, even if LDA produces a recognizable sports topic it may be combined with other topics or there may be other sports topics.

Another important property of LDA topics is that they are independent. This means that a word can only come from a single topic. Even if LDA produced a recognizable "sports" topic and a recognizable "news" topic, their combination is assumed never to occur. "Sports news," may then appear as a third topic, independent from the first two. Or, it may be present in other topics whose focus is neither sports nor news. The independence of topics makes their comparison problematic. For example, it might be desirable to ask if two topics overlap in some way, but due to their independence they are not allowed to correlate: the output of LDA has topic-to-topic correlation values that are never significantly different from 0, as shown within the confidence intervals of Table 2.

Another pitfall is that some LDA software reports probabilities while others report word counts or other weights. While one can convert between probabilities and word counts, it is important to consider whether each document receives equal weight, or whether longer documents should receive more weight than shorter documents.

Since LDA models are found iteratively, it is important to ensure that they have had adequate time to converge before use. Otherwise, the model does not correspond to the input data. The time required for convergence depends on the number of topics, documents and vocabulary words. For example, Vowpal Wabbit with 100 topics and 20 000 documents, each pass takes a matter of seconds on modest hardware, but at least 2 passes are recommended. In order to choose the correct number of passes, the output should be examined and the number of passes increased until the output stops changing significantly.

Finally, it is necessary to remember that LDA assumes that word-topic probabilities and topic-document probabilities are Dirichlet distributions. Furthermore, many pieces of LDA software use symmetric Dirichlet distributions. This implies the assumption that the Dirichlet parameters are the same for every word ($\beta$) or topic ($\alpha$), respectively, and that these parameters are known beforehand. In most software this means that $\alpha$ and $\beta$ must be set carefully.

## 5 LDA uses in Software Analysis

The Latent Dirichlet Allocation (LDA) method was originally formulated by Blei [4] and it soon became quite popular within the software engineering community. LDA's popularity comes from the variety of its potential applications and uses.

LDA excels at feature reduction and can employed as a pre-processing step for other models, such as machine learning algorithms. LDA can also be used to augment the inputs to machine learning and clustering algorithms by producing additional features from documents. One example of this type of LDA usage is described in Wang et al. [23], where it is employed in a recommender system. Similarly, labelled LDA can be used to create vectors of independent features from arbitrary feature sets such as tags.

An important use of LDA is for linking software artifacts. There are many instances of such artifact-linking applications, such as measuring coupling between code modules [19] and matching code modules with natural-language requirement specifications [20] for traceability purposes. Asuncion et al. [2] applied LDA on textual documentation and source-code modules and used the topic document matrix to indicate traceability between the two. Thomas et al. [22] focused on the use of LDA on yet another traceability problem, linking email messages to source-code modules. Gethers et al. [8] investigated the effectiveness of LDA for traceability-link recovery. They combined IR techniques including Jenson/Shannon model, vector space model and the relational topic model using LDA together. They concluded that each technique had its positives and negatives yet, the integration of the methods together tended to produce the best results. Typically steeped in the information retrieval domain, Poshyvanyk et al. [21, 18, 16] have explored the use of IR techniques such as LSI [15] and LDA to recover software traceability links in source code and other documents. For a general literature survey related to traceability techniques (including LDA), the interested reader should refer to De Lucia [7].

Baldi et al. [3] labelled LDA extracted topics and compared them to aspects in software development. Baldi claims that some topics do map to aspects, this was somewhat corroborated by Hindle et al. [12].

Clustering is frequently used to compare and identify (dis)similar documents and code, or to quantify the overlap between two sets of documents. Clustering algorithms can potentially be applied to topic probability vectors produced by LDA. LDA has been used in a clustering context, for issue report querying and deduplication. Lukins et al. [14] applied LDA topic analysis to issue reports, leveraging LDA inference to infer if queries, topics, and issue reports were related to each other. Alipour et al. [1] leveraged LDA topics to add context to deduplicate issue reports and found that LDA topics added useful contextual information to issue/bug deduplication. Campbell et al. [5] used LDA to examine the coverage of popular project documentation by applying LDA to two collections of documents at once: user questions and project documentation. This was done by clustering and comparing LDA output data.

Often LDA is used to summarize the contents of large datasets. This is done by manually or automatically labelling the most popular topics produced by unlabeled LDA. Labeled LDA can be used to track specific features over time, for example to measure the fragmentation of a software ecosystem as in Han et al. [10].

Even though LDA topics are assumed to be implicit and not observable, there is substantial work on assessing the interpretability of those summaries by developers. Labelling software artifacts using LDA was investigated by De Lucia et al. [6]. By using multiple IR approaches such as LDA and LSI, they labelled and summarized source code and compared against human-generated labels. Hindle et al. [11] investigated if developers could interpret and label LDA topics. They report limited success with 50% being successfully labelled by the developers and that non-experts tend to do poorly at labelling topics on systems they have not dealt with.

Finally, there has been some research on the appropriate choice of LDA hyper-parameters and parameters: $\alpha$, $\beta$, $K$ topics. Grant et al. [9] were concerned about $K$ where $K$ is the number of topics. While Panichella et al. [17] proposed LDA-GA, a genetic algorithm approach to searching for appropriate LDA hyper-parameters and parameters. LDA-GA needs an evaluation measure, thus Panichella et al. used software engineering specific tasks that allowed their GA to optimize against the cost effectiveness.

## 6 Threats to Validity

This section summarizes the threats to validity that practitioners may face when using LDA. Pitfalls have been identified throughout the chapter, especially in section 4, but some are summarized here.

### 6.1 Construct Validity

Construct validity relates to the ability of research to measure what it intended to measure. LDA topics are independent topics extracted from word distributions. This independence means that correlated or co-occuring concepts or ideas will not necessarily be given their own topic, and if they are the documents might be split between topics. One should be aware of the constraints and properties of LDA when trying to infer if LDA output shows an activity or not. LDA topics are not necessarily human ideas, concepts or topics. Comparisons between topics in terms of document association can be troublesome due to the independence assumption of topics.

### 6.2 Internal Validity

Internal Validity refers to how well conclusions can be made about casual effects and relationships. An important aspect of LDA is that topics are independent, thus if two ideas are being studied to see if one causes the other one has to guard against LDA's word allocation strategy. To show that an event caused a change in LDA output one should use a different data source and manual validation. LDA output changes given different $\alpha$ and $\beta$ parameters, and sometimes given a test one could tune these parameters to pass or fail this test. Thus one has to motivate the choice of $\alpha$ and $\beta$ parameters carefully.

### 6.3 External Validity

External validity is about generalization and how broadly findings can be made. LDA topics are relevant to the corporae provided thus their topics and words associated with the topics might not be generalizable. Alternatively LDA can be applied to numerous collections of documents and thus external validity can be addressed in some situations.

### 6.4 Reliability

Reliability is about how well one can repeat the findings of a study. With LDA the exact topics found will not be found again without sharing of initial parameters or seeds. Thus all LDA studies should report their parameters. Yet even if parameters are reported LDA implementations will return different results and the same implementation might produce different topics or different topic orderings each time it is run. Others might not be able to replicate the exact topics found or the ordering of the topics found.

## 7 Conclusions

Latent Dirichlet Allocation is a powerful tool for working with collections of structured, unstructured, and semi-structured text documents, of which there are plenty in software repositories. Our literature review has documented the abundance of LDA applications in software analysis, from document clustering for issue/bug deduplication, to linking for traceability between code, documentation, requirements and communications, to summarizing for association with software-lifecycle activities.

We have demonstrated a simple case of using LDA to explore the contents of an issue tracker repository and showed how the topics link back to documents. We also discussed how to visualize the output.

LDA however relies on a complex underlying probabilistic model and a number of assumptions. Therefore, even though off-the-shelf software is available to compute LDA models, the user of this software must be aware of potential pitfalls and caveats. This chapter has outlined the basics of the underlying conceptual model and discussed these pitfalls in order to enable the informed use of this powerful method.

## References

[1] A. Alipour, A. Hindle, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 183–192. IEEE Press, 2013.

[2] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 95–104, New York, NY, USA, 2010. ACM.

[3] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya. A theory of aspects as latent topics. In *Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA '08, pages 543–562, New York, NY, USA, 2008. ACM.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[5] J. C. Campbell, C. Zhang, Z. Xu, A. Hindle, and J. Miller. Deficient documentation detection: a methodology to locate deficient project documentation using topic analysis. In T. Zimmermann, M. D. Penta, and S. Kim, editors, *MSR*, pages 57–60. IEEE / ACM, 2013.

[6] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Using ir methods for labeling source code artifacts: Is it worthwhile? In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 193–202. IEEE, 2012.

[7] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information retrieval methods for automated traceability recovery. In *Software and Systems Traceability*, pages 71–98. Springer, 2012.

[8] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 133–142. IEEE, 2011.

[9] S. Grant and J. R. Cordy. Estimating the optimal number of latent concepts in source code analysis. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, SCAM '10, pages 65–74, Washington, DC, USA, 2010. IEEE Computer Society.

[10] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia. Understanding android fragmentation with topic analysis of vendor-specific bugs. In *WCRE*, pages 83–92. IEEE Computer Society, 2012.

[11] 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 *ICSM*, pages 243–252. IEEE Computer Society, 2012.

[12] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 163–172. ACM, 2011.

[13] J. Langford, L. Li, and A. Strehl. Vowpal wabbit, 2007.

[14] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Source code retrieval for bug localization using latent dirichlet allocation. In *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, WCRE '08, pages 155–164, Washington, DC, USA, 2008. IEEE Computer Society.

[15] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings of the 11th Working Conference on Reverse Engineering*, WCRE '04, pages 214–223, Washington, DC, USA, 2004. IEEE Computer Society.

[16] C. McMillan, D. Poshyvanyk, and M. Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '09, pages 41–48, Washington, DC, USA, 2009. IEEE Computer Society.

[17] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 522–531. IEEE Press, 2013.

[18] D. Poshyvanyk. *Using Information Retrieval to Support Software Maintenance Tasks.* PhD thesis, Wayne State University, Detroit, MI, USA, 2008.

[19] D. Poshyvanyk and A. Marcus. The conceptual coupling metrics for object-oriented systems. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, pages 469–478. IEEE, 2006.

[20] B. Ramesh. Factors influencing requirements traceability practice. *Commun. ACM*, 41(12):37–44, Dec. 1998.

[21] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk. Topicxp: Exploring topics in source code using latent dirichlet allocation. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–6, Washington, DC, USA, 2010. IEEE Computer Society.

[22] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Validating the use of topic models for software evolution. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, SCAM '10, pages 55–64, Washington, DC, USA, 2010. IEEE Computer Society.

[23] H. Wang and K. Wong. Recommendation-assisted personal web. In *Services (SERVICES), 203 IEEE Ninth World Congress on*, pages 136–140. IEEE, 2013.

[24] Wikipedia. Latent dirichlet allocation — wikipedia, the free encyclopedia, 2014. [Online; accessed 15-July-2014].