Mining Developer Expertise from Bug Tracking Systems using the Author-topic Model

Daniel Atzberger*, Jonathan Schneider*, Willy Scheibel[a],
Daniel Limberger[b], Matthias Trapp[c] and Jürgen Döllner
Hasso Plattner Institute, Digital Engineering Faculty, University of Potsdam, Germany

Keywords: Bug Triaging, Topic Models, Latent Dirichlet Allocation, Author-topic Model.

Abstract: During the software development process, software defects, so-called bugs, are captured in a semi-structured manner in a bug tracking system using textual components and categorical features. It is the task of the triage owner to assign open bugs to developers with the required skills and expertise. This task, known as *bug triaging*, requires in-depth knowledge about a developer's skills. Various machine learning techniques have been proposed to automate this task, most of these approaches apply topic models, especially Latent Dirichlet Allocation, for mining the textual components of bug reports. However, none of the proposed approaches explicitly models a developer's expertise. In most cases, these algorithms are treated as a black box, as they allow no explanation about their recommendation. In this work, we show how the Author-Topic Model, a variant of Latent Dirichlet Allocation, can be used to capture a developer's expertise in the latent topics of a corpus of bug reports from the model itself. Furthermore, we present three novel bug triaging techniques based on the Author-Topic Model. We compare our approach against a baseline model, that is based on Latent Dirichlet Allocation, on a dataset of 18 269 bug reports from the Mozilla Firefox project collected between July 1999 to June 2016. The results show that the Author-Topic Model can outperform the baseline approach in terms of the Mean Reciprocal Rank.

1 INTRODUCTION

Today's software projects are characterized by a large number of developers involved and an ever increasing structural and behavioral complexity. This is the main cause for the emergence of software misbehavior, called bugs. When a bug is detected, it is described by a bug report within an issue tracking or bug tracking system in a semi-structured manner. Besides textual components, e.g., a title and a more detailed description, bug reports use categorical attributes, e.g., the product, the affected component, the version, and the priority and severity, to specify details. Additionally, files can be attached and solutions can be discussed. Based on this rich source of information, it is the task of the bug triage owner to assign the bug report to a developer to fix it. This process is known as bug triaging. In order to ensure

high productivity, it is necessary that the developer to solve the bug has the required skills and expertise (Sommerville, 2016). In most cases, the decisions made by the triage owner are based on subjective perceptions (Zou et al., 2020), that in turn often lead to wrong assignments, which might result in a delay in the development process.

Automated bug triaging techniques try to overcome this issue by mining the information contained in a bug tracking system (Sajedi-Badashian and Stroulia, 2020; Zou et al., 2020; Zhang et al., 2015). Most of these approaches analyze closed bug reports with known bug resolvers to train a machine learning algorithm that can then in turn be used to rank developers for a new bug. A large class of those techniques models the textual components of bug reports using a probabilistic topic model, mostly Latent Dirichlet Allocation (LDA) (Blei et al., 2003). The goal of LDA is to extract semantic clusters in the vocabulary, given as distributions over this vocabulary, by examining patterns of co-occurring words within bug reports. The semantic structure of bug reports can then in turn be

[a] https://orcid.org/0000-0002-7885-9857
[b] https://orcid.org/0000-0002-9111-4809
[c] https://orcid.org/0000-0003-3861-5759
*Both authors contributed equally to this work.

107

described as multinomial distributions over the topics. The results of the LDA model are then used as input for the machine learning model. Though it has shown great advances in the last years, LDA is not able to describe a developer's expertise directly from the available data. As such, these models provide no explanation to the user, which is a limiting factor for their applicability in a real-world setting (Aktas and Yilmaz, 2020; Zou et al., 2020).

In this work, we apply the Author-Topic Model (ATM) on a corpus of closed bug reports with known resolvers, to model a developer's expertise in the latent topics (Rosen-Zvi et al., 2004). Based on this description of a developer, we modify an existing bug triaging algorithm, named Developer Recommendation based on Topic Models (DRETOM), based on LDA, proposed by Xie et al. (2012). This results in three alternative methods for automated bug triaging, namely

1. Developer Recommendation based on the Author-Topic Model (DRATOM),

2. Developer Recommendation based on the Author-Topic Model and Bayes Formula (DRATOMBayes): a further iteration of DRATOM, which makes use of the Bayes formula for conditional probabilities,

3. Developer Recommendation based on Author Similarity (DRASIM): By applying the ATM on past bug reports, developers and bug reports can be described in a joint feature space. This reduces the bug triaging task to a Nearest Neighbor (NN) search.

We further evaluate the effectiveness of the ATM and the influence of its hyperparameters for the bug triaging task. For this purpose, we use a publicly available dataset of 18 269 bug reports from the Mozilla Firefox project, collected between July 1999 and June 2016. The results show that our modification of DRETOM is able to outperform the baseline approach in terms of the Mean Reciprocal Rank (MRR).

The remainder of this work is structured as follows: Section 2 presents existing approaches for bug triaging, which apply LDA for modeling the textual components of bug reports. In Section 3 we give a short overview of LDA and its variant ATM. We detail our three novel bug triaging algorithms, based on the ATM, in section 4. We describe the experimental setup in section 5 and discuss its results in section 6. Section 7 concludes this paper and outlines directions for future work.

2 RELATED WORK

Various machine learning and information retrieval approaches for bug triaging have been proposed (Sajedi-Badashian and Stroulia, 2020). We focus our presentation of the related work on approaches that apply probabilistic topic models on the textual components of bug reports. We therefore disregard work that utilizes data from version control systems (Matter et al., 2009; Kagdi et al., 2012; Shokripour et al., 2013; Khatun and Sakib, 2016; Hu et al., 2014) or question-and-answer platforms (Sajedi-Badashian et al., 2015). Here, we want to clarify that we always refer to bug reports, even though issue reports or change requests are also common terms used in the literature. Precisely, we refer to the problem defined as follows: "Given a new bug report, identify a ranked list of developers whose expertise qualifies them to fix the bug" (Sajedi-Badashian and Stroulia, 2020).

Xie et al. were the first to apply LDA for the bug triaging task (Xie et al., 2012). Their approach DRETOM models a developer's interest and expertise in topics from a corpus of historical bugs for recommending suitable contributors for an open bug. As our approach builds upon the work of Xie et al., we will present a detailed description of it in Section 4.

Based on the work of Xie et al. (2012), successive work focused on improving the quality of developer recommendations considering additional input features of a bug report (Xia et al., 2013; Naguib et al., 2013; Zhang et al., 2014a; Yang et al., 2014; Nguyen et al., 2014; Zhang et al., 2016; Xia et al., 2016; Zhang et al., 2014b). What all approaches have in common is that for each bug report, the title and description are used to determine the latent topics.

Given a bug report, Xia et al. extract a feature vector that captures the topics via LDA, the affected product, and components of the bug report (Xia et al., 2013). Using a newly defined distance metric, the algorithm recommends a set of developers who are assigned to a new bug report.

Naguib et al. proposed an approach leveraging LDA and the developer activities, such as reviewing, fixing, and assigning bug reports (Naguib et al., 2013). First, they use in addition to a bug report's title and description its system component to categorize bug reports into topics. Subsequently, they create an activity profile for each developer, so that it can be used alongside the topic associations to recommend a suitable developer.

Zhang et al. also utilized LDA to extract topics from historical bug reports (Zhang et al., 2014a). In addition, they analyzed the relationship between the

developers, i.e., assignees, commenters, and the bug reporter whose bug reports attracted the most number of comments.

Yang et al. proposed a new method by utilizing not only the natural language for LDA, but multiple features including product, component, severity, and priority (Yang et al., 2014). They extract the set of candidate developers who have contributed to the bug reports having the same combination of topic and features and rank them by the scores determined using their number of activities, i.e., a developer's number of commits, comments, attachments, and bug assignments.

Zhang et al. use structural information from the submitter-bug-commenter heterogeneous network to improve the quality of developer recommendations (Zhang et al., 2014b). Through this network, submitters and commenters obtain the topic information propagated from bugs, which adjust the topic distributions obtained by the LDA model.

Nguyen et al. used the inferred topics from LDA along with the bug fixer and the severity levels of historical bug reports as an input of a log-normal regression model for predicting defect resolution time (Nguyen et al., 2014). Based on each developer's mean time to repair and dependent on the most prominent topic of the open bug report, they recommend the developer that is likely the fastest in resolving the issue.

In a later work, Zhang et al. proposed a new similarity measure to find related historical bug reports based on its title, description, product, and component (Zhang et al., 2016). Their approach uses 18 hyperparameters that need to be tuned to weight the individual features, of which only the number of topics is a hyperparameter of the underlying LDA model. The other hyperparameters of LDA remain on the default settings of the respective LDA implementation. For the actual developer recommendation, that also incorporates a developer's number of comments, fixed and reopened bugs, there are two more hyperparameters to be tuned.

Xia et al. proposed the Multi-feature Topic Model (MTM), a method that extends the basic topic modeling algorithm LDA (Xia et al., 2016). The MTM includes a bug report's feature combination, e.g., its product and component, as an additionally observed variable, and models the topic distributions for each feature combination. Their approach recommends the developers based on the affinity score towards a topic and the feature combination.

A detailed overview on how topic models are used for software engineering tasks in general is presented by Chen et al. (2016).

Table 1: Three extracted topics from Mozilla Firefox. For each topic, the ten most probable terms are displayed.

Topic #1	Topic #2	Topic #3
private	preferences	bar
browsing	pref	url
mode	options	location
clear	dialog	autocomplete
cookies	default	text
history	set	address
window	prefs	results
data	option	type
cookie	preference	enter
cache	change	result

3 LATENT DIRICHLET ALLOCATION

In this section we give a short overview on LDA required to understand the rest of our work. Topic models are a widely used technique for mining the semantic structure of a collection of documents, e.g., for summarization or classification tasks (Aggarwal and Zhai, 2012). The by far most popular topic model for mining software repositories is LDA proposed by Blei et al. (Chen et al., 2016; Sun et al., 2016). Given a corpus $C = \{d_1, \ldots, d_m\}$, where each element $d_1, \ldots, d_m$ denotes a single document stored as a Bag-of-Words (BOW), LDA extracts latent topics $\varphi_1, \ldots, \varphi_K$, where the number of topics K is a hyperparameter of the model (Blei et al., 2003). The topics $\varphi_1, \ldots, \varphi_K$ are given as multinomial distributions over the vocabulary $\mathcal{V}$ of the corpus C. In most cases the most probable words are a strong indicator for the "concept" underlying a topic. Table 1 shows an example for three topics with their ten most probable words that were extracted from the Mozilla Firefox corpus, which we will describe in detail in Section 5.1. LDA further results in descriptions $\theta_1, \ldots, \theta_m$ of the documents $d_1, \ldots, d_m$ as multinomial distributions over the set of topics. The vectors $\theta_1, \ldots, \theta_m$ therefore capture the semantic structure of the documents and allow a formal comparison using a dissimilarity measure, e.g., the cosine-similarity or the Jensen-Shannon distance. The generative process underlying LDA is given by

1. For each document d in the corpus $\mathcal{D}$ choose a distribution over topics $\theta \sim \text{Dirichlet}(\alpha)$

2. For each word w in d

   (a) Choose a topic $z \sim \text{Multinomial}(\theta)$

   (b) Choose the word w according to the probability $p(w|z, \beta)$

The hyperparameter $\alpha = (\alpha_1, \ldots, \alpha_K)$, where $0 < \alpha_i$ for all $1 \leq i \leq K$, is called the Dirichlet prior for the
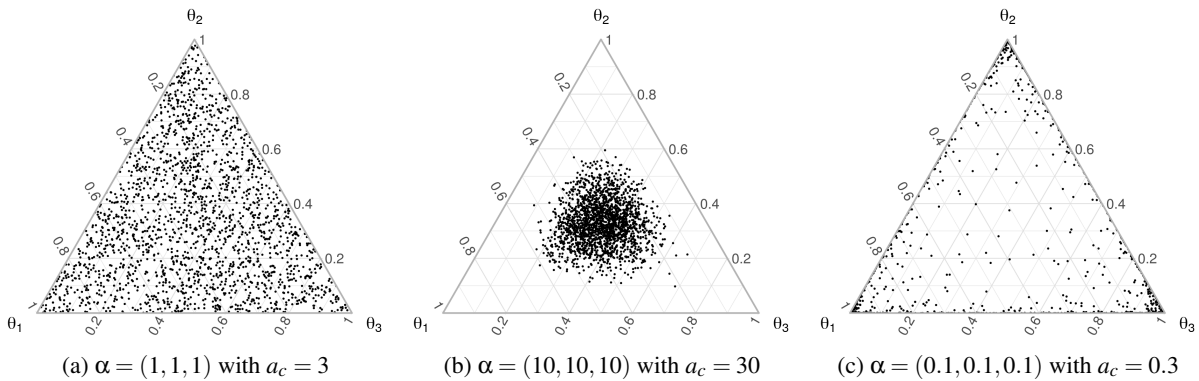
Figure 1: 2,000 randomly sampled topic distributions θ using varying concentration parameters a_c for symmetrical Dirichlet prior α. We illustrate the distributions for $K = 3$ topics in the three-dimensional topic space $\theta = (\theta_1, \theta_2, \theta_3)$. The base measure $m = (1/3, 1/3, 1/3)$ is uniform for every symmetrical Dirichlet distribution.

document-topic distribution. It is often written as the product $\alpha = a_c \cdot m$ of its concentration parameter $a_c \in \mathbb{R}$ and its base measure $m = (m_1, \ldots, m_K)$, which is a vector whose components sum up to one. In the case of $m_1 = \cdots = m_K = 1/K$, the Dirichlet prior α is said to be symmetric. Figure 1 illustrates the effect of the chosen Dirichlet prior α.

The parameter $\beta = (\beta_1, \ldots, \beta_N)$, where $0 < \beta_i$ for $1 \leq i \leq N$ and N denotes the size of the vocabulary $\mathcal{V}$, i.e., the set of words in the corpus $\mathcal{C}$, is the Dirichlet prior for the topic-term distribution.

As inference for LDA is intractable, it is required to apply approximation techniques (Blei et al., 2003). Popular examples include Collapsed Gibbs Sampling (CGS) (Griffiths and Steyvers, 2004), Variational Bayes (VB) (Blei et al., 2003), and its online version (OVB) (Hoffman et al., 2010).

Many variants of LDA have been developed taking into account additional meta information about the documents of a corpus, e.g., Labeled LDA which makes use of tags attached to the documents, thus resulting in topics with a defined label (Ramage et al., 2009). The ATM is another such variant, which takes information about authorship into account. The ATM is based on a generative process, similar to that of LDA, which assumes that each document is written in a collaborative manner by its authors, who are observed as additional information (Rosen-Zvi et al., 2004). In addition to the document-topic distributions, the ATM learns representations of the authors as distributions over the topics. LDA can be seen as a special case of the ATM, where each file has one unique author. To the best of our knowledge, the ATM was only used for a software engineering task in the work of Linstead et al. for mining developer activities on the Eclipse source code (Linstead et al., 2007; Linstead et al., 2009). When applied to historical bug reports, each developer who has contributed to bug

fixing can be described as a distribution over topics.

4 MINING BUG REPORTS WITH AUTHOR-TOPIC-MODELS

The approach DRETOM follows three steps. First, an LDA model is trained on a corpus of historical bug reports, then associations between developers and topics are constructed using a heuristic, and finally authors are recommended for incoming bug reports according to some ranking scheme. In this section, we present three approaches to how the ATM can be utilized for the bug triaging task by taking its capability to relate developers and topics into account. It has to be noted that in the context of bug triaging, we denote the resolver of the bug report as the author, who should not be confused with the reporter of a bug. As our first two approaches are modifications of DRETOM, we start with a description of the approach by Xie et al. (2012). Our first approach (DRATOM) is derived from DRETOM by simply replacing its LDA core with the ATM. The second modification (DRATOMBayes) utilizes the relation between developers and topics learned from the ATM, thus making the heuristic and trade-off parameter θ from DRETOM redundant. Our third approach makes full use of the ATM by modeling developers as well as bug reports in a joint feature space, thus reducing the bug triaging task to a NN search.

4.1 Preprocessing

In the remainder of this work, we view each bug report as a document consisting of its title and a more detailed description. Discussions below the bug report are not taken into account, as they are not available during the entire lifecycle of a bug, especially

not during the first assignment. When applying topic models, it is common to undertake several preprocessing steps, in order to remove noise in the vocabulary (Chen et al., 2016). It further reduces the size of the vocabulary thus accelerating the computation time. Our preprocessing pipeline follows the best-practice recommendations systematically studied by Schofield et al. (Schofield et al., 2017b; Schofield et al., 2017a; Schofield et al., 2017c):

1. Removal of URL, hex code, stack trace information, timestamps, line numbers, tokens starting with numerics, tokens consisting of only one character and punctuation,

2. Lower casing the entire document,

3. Removal of all words from the English *NLTK Stopwords Corpus*[1]

4. We remove words with a frequency less than 5, we further remove words that occur in more than 20% across all bug reports.

In particular, in contrast to Xie et al. (2012) we do not apply stemming, as words with the same morphological roots are often placed in the same topic, thus making stemming redundant and might damage models (Schofield and Mimno, 2016). After preprocessing, all bug reports are stored as BOW.

4.2 DRETOM

In the first step of DRETOM, an LDA model is trained on a corpus of historical bug reports and each of these bug reports is assigned to the topic of its maximum probability. Additionally, all collaborators for the historical bug reports are extracted from the bug tracking system.

The suitability of a developer d to solve a bug b is given by the conditional probability $P(d|b)$. This probability can be computed as the sum

$$P(d|b) = \sum_z P(d|z) \cdot P(z|b), \quad (1)$$

where $P(d|z)$ denotes the skill level of the developer d in the respective topic z, and $P(z|b)$ is the probability of the topic in the given bug report. The skill of developer d in topic z comprises two parts, his interest $P(d \rightarrow z)$ and his expertise $P(z \rightarrow d)$, which are balanced by a trade-off parameter $\theta \in [0, 1]$ according to

$$P(d|z) = \theta P(d \rightarrow z) + (1 - \theta)P(z \rightarrow d). \quad (2)$$

The interest of a developer in a topic is computed as

$$P(d \rightarrow z) = \frac{N_{d,z}}{N_d}, \quad (3)$$

where $N_{d,z}$ is the number of bug reports that belong to topic z and were resolved with the participation of developer d, N_d is the number of bugs, where the developer has contributed.

The expertise component is computed as

$$P(z \rightarrow d) = \frac{N_{d,z}}{N_z}, \quad (4)$$

where N_z is the number of bug reports that are associated with topic z.

For an incoming bug report, its topic distribution is inferred using the trained LDA model and the developers are ranked according to their conditional probabilities from Equation (1).

4.3 Modifying DRETOM with the ATM

By replacing the LDA model in DRETOM with the ATM, we end up with a first modification, named DRATOM, i.e., the probability $P(d|z)$ is computed as in the DRETOM approach, whereas the probability distribution of a bug over topics is now derived from the ATM.

An advantage of the ATM against a heuristic method for deriving relations between developers and topics lies in its interpretability, as each developer can be modeled as a distribution over the topics. This sort of explainability is particularly interesting for its use in a real-world setting (Zou et al., 2020; Aktas and Yilmaz, 2020).

Xie et al. have shown that the quality of the generated developer recommendations is sensitive to its parameter θ. We circumvent this problem by taking the description of a developer as a distribution over topics, learned from the ATM, into account. From the known probability $P(z|d)$ for an arbitrary topic z and developer d, we derive from the Bayes' formula:

$$P(d|z) = \frac{P(z|d) \cdot P(d)}{P(z)}. \quad (5)$$

The probability for a developer can be computed as

$$P(d) = \frac{N_d}{N_{total}}, \quad (6)$$

where N_{total} denotes the number of all bug reports in the training corpus. We approximate the marginal probability $P(z)$ using the Dirichlet prior α. Written α as the product between its normalization base measure m and its concentration parameter a_c, we conclude

$$m = (m_1, \dots, m_K) = (P(z_1), \dots, P(z_K)). \quad (7)$$

We approximate the probability $P(z)$ via its respective Dirichlet parameter. We further denote this bug triaging algorithm as DRATOMBayes.

---

[1]http://www.nltk.org/nltk_data/

4.4 DRASIM

Our third bug triaging algorithm adopts the idea presented by Linstead et al., who applied the ATM on source code files, for the case of bug reports (Linstead et al., 2007). As the ATM models documents, i.e., bug reports, as well as developers as distributions over common topics, both categories are embedded in a joint feature space and can therefore be compared. The affinity of a developer d to solve a given bug b can thus be computed via $D(b,d)$, where D denotes a chosen similarity measure, e.g., the Jensen-Shannon distance, the cosine-similarity, or the Manhattan distance. The ranking of developers reduces to a NN search.

5 EXPERIMENTAL SETUP

To guarantee the reproducibility of our study, we present a detailed description of our experimental setup. We start with a description of the data set used in our study and a justification for the chosen preprocessing techniques. We further provide details on the longitudinal evaluation scheme as well as on the chosen topic modeling implementation.

5.1 Dataset and Preprocessing

In our work, we use a data set presented by Mani et al., which contains 18269 bug reports from the Mozilla Firefox project, collected between July 1999 to June 2016, with 169 involved developers (Mani et al., 2019). We train our model on bug reports with an assigned developer and a status marked as *verified fixed*, *resolved fixed*, or *closed fixed* and consider each bug reports' final assignee as the actual bug resolver. We then preprocess the documents as described in section 4.1. Since our proposed approaches use the ATM instead of LDA, we need enough bug reports in the training data for each developer to learn their expertise with regard to certain topics directly from the data. Mani et al. showed that a threshold greater than 20 did not improve the recommendation accuracy on the same dataset (Mani et al., 2019). We make sure there exist at least 20 bug reports per developer in each training set. The characteristics of the Mozilla Firefox dataset used for evaluation are summarized in Table 2.

5.2 Evaluation Scheme

In every large software project, the developers keep changing over time, hence chronological splitting en-

Table 2: Summary of the Mozilla Firefox project used for evaluation.

Property	Mozilla Firefox
Period considered	July 1999 – June 2016
# Bugs	18,269
# Bug resolvers	169
# Terms	22,209
# Terms w/o stop words	18,590
# Terms w/o extremes	7,195

sures that the bug reports used for training and testing have a high overlap of bug resolvers and we do not suffer from information leakage. All the fixed bug reports are sorted in chronological order and split into eleven sets. In the l-th run, $1 \leq l \leq 10$, the first l sets are used for training the model and the $(l+1)$-th subset is used for testing. This evaluation scheme is called longitudinal evaluation and is known to increase internal validity (Bhattacharya and Neamtiu, 2010; Jonsson et al., 2016; Mani et al., 2019; Tamrawi et al., 2011; Xia et al., 2016). Different evaluation metrics have been used in related work. Among the most widely used metrics are Top-k accuracy, Precision@k and Recall@k, often used with $k \in \{1,3,5,10\}$, as used by Xie et al. Those set-based metrics have in common that they do not consider the rank as long as the real assignee is among the top k recommended developers. However, a good developer recommendation approach should place the probably most suitable developer for a bug report at the top of the list. This is because the bug triager usually checks the higher ranks in the list and may not proceed to the last one. Therefore, errors in the higher ranks are worse than errors in the lower ranks. The MRR and Mean Average Precision (MAP), are more affected by a hit in the first ranks compared to the following ranks (Sajedi-Badashian and Stroulia, 2020). In a sense, they penalize the mistakes in the first ranks more than in subsequent ranks. While the MRR stops at the first real assignee, this penalization continues for MAP, for every incorrect guess until the last real assignee in the list (Sajedi-Badashian and Stroulia, 2020). In our case, MAP and MRR are the same, since there is only one real assignee for each bug report. In our experiments we consider the MRR@10, which is given by

$$MRR@10 = \frac{1}{\#bug\,reports} \sum_{i=1}^{\#bug\,reports} (RR@10)_i \quad (8)$$

$$(RR@10)_i = \begin{cases} \frac{1}{rank_i}, & \text{if } rank_i \leq 10 \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

where RR denotes the reciprocal rank and $rank_i$ is the rank position of the real assignee for the ith bug report.

Table 3: Features of used LDA and ATM implementations of *Gensim* and *MALLET*.

	LdaModel	LdaMulticore	AuthorTopicModel	TopicTrainer
project	*Gensim*	*Gensim*	*Gensim*	*MALLET*
varying K	✓	✓	✓	✓
symmetrical α/β	✓	✓	✓	✓
optimized symmetrical α/β	✗	✗	✗	✓
optimized asymmetrical α	✓	✗	✓	✓
optimized asymmetrical β	✓	✗	✓	✗[4]
parallelism	✓[5]	✓	✗	✓

5.3 Topic Modeling Implementations

In our experiments, we use two widely used and actively maintained libraries for topic modeling. Table 3, summarizes the features of the different topic modeling libraries.

1. *Gensim* (Rehůřek and Sojka, 2010) offers an LDA implementation based on the original implementation by Blei et al. (Blei et al., 2003), that we refer to as Variational Bayes (VB), as well as its online version Online Variational Bayes (OVB) introduced by Hofman et al. (Hoffman et al., 2010). Further features of *Gensim* include the automatic hyperparameter estimation of the Dirichlet priors α and β, as well as a parallel version. For the ATM we always refer to *Gensim*.

2. *MALLET* implements the simple parallel threaded implementation proposed by Newman et al. (Newman et al., 2009) with the SparseLDA Gibbs scheme and data structure introduced by Yao et al. (2009). Based on the work of Wallach (2008), *MALLET* implements several strategies for hyperparameter optimizations of Dirichlet priors.

6 RESULTS

In this section, we give details on the conducted experiments for evaluating our three bug triaging techniques based on the ATM. We compare our approaches with DRETOM, the approach proposed by Xie et al. (2012). Though more advanced bug triaging algorithms exist, we choose the DRETOM approach proposed by Xie et al. (2012) for comparison. This is mainly because novel bug triaging algorithms try to improve their models by taking additional information into account, rather than improving the underlying topic model and its respective hyperparameters (Xia et al., 2013; Naguib et al., 2013; Zhang et al., 2014a; Yang et al., 2014; Nguyen et al., 2014; Zhang et al., 2016; Xia et al., 2016; Zhang et al., 2014b).

On this exemplary approach, we want to discuss the following research questions:

RQ1. *Are the approaches based on the ATM able to outperform DRETOM?*

RQ2. *How far do the choice of hyperparameter, i.e., the number of topics and the Dirichlet prior, affect the approaches for bug triaging tasks?*

6.1 Naive Replication of DRETOM

As DRETOM mainly depends on LDA, we first investigate the effect of the chosen LDA implementation on the results. We choose *Gensim* as representative LDA implementation using VB and *MALLET* using Collapsed Gibbs Sampling (CGS). We used the proposed parameters as specified by Xie et al. (2012): a symmetrical Dirichlet prior both for the document-topic distribution $\alpha = 0.01$, the topic-term distribution $\beta = 0.01$, and the number of topics to $K = 20$. For the *MALLET* implementation we set the number of iterations to 100. We keep the remaining parameters set to their respective default values defined by the respective LDA implementation.

We report the mean of ten cross-validation runs of the MRR@10 on the Mozilla Firefox project in Figure 2.

The DRETOM approach using *MALLET* is superior to the other three implementations, with a relative improvement of at least 47%. On average, the MRR@10 for DRETOM using *MALLET* is 13.1%. For all other implementations using VB inference this number is significantly lower, 8.9% for *Gensim*, and slightly worse for the parallelized LDA implementation of *Gensim* (8.3%) Our next observation is that the best hyperparameter setting for θ varies between the different cross-validation splits between 0.0 and 0.3

---

[4]Asymmetrical β priors are not beneficial (Wallach et al., 2009).

[5]When passing `distributed=True` as a parameter, the implementation makes use of a cluster of machines via pyro4.
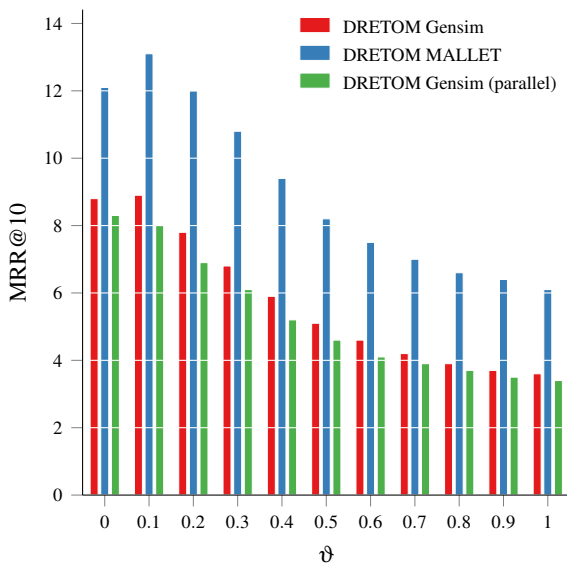
Figure 2: Impact of underlying inference techniques using default parameters for LDA implementation on MRR@10 obtained on the Mozilla Firefox project. The mean over a 10-fold cross-validation is reported.
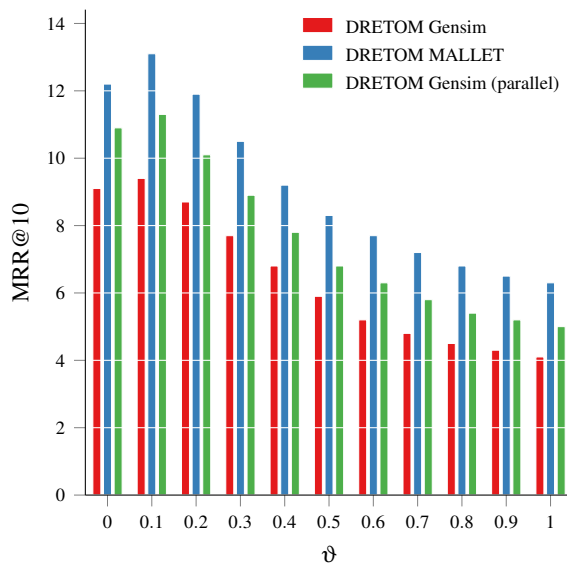

Figure 3: Impact of underlying inference technique using modified number of iteration steps for LDA implementation on MRR@10 obtained on the Mozilla Firefox project. The mean over a 10-fold cross-validation is reported.

according to the MRR@10. Xie et al. report that the average precision and recall has its peak at $\theta = 0.6$ for the Mozilla Firefox project which favors the developer's interest slightly more compared to a developer's expertise (Xie et al., 2012). Surprisingly, regardless of the LDA implementation DRETOM never reaches an optimal value of θ greater than 0.3 in the first experiment of our reproduction study with respect to the MRR@10. Across all cross-validation splits the MRR@10 peaks at $\theta = 0.0$ in 50% of the cases, i.e., the recommendation is completely based on a developer's expertise.

Since all LDA implementations using VB perform significantly worse than the Gibbs implementation used in *MALLET*, we devote our attention to the question of whether the differences with respect to the MRR@10 are due to insufficient convergence of the models trained with OVB. Particularly decisive are the training parameters i_{VB} and i_{CGS} to make sure the models have enough iterations to converge. We increase the number of iterations to $i_{VB} = 400$ in the case of *Gensim*. Following the original implementation, the number of Gibbs iterations in our first experiment was relatively low with $i_{CGS} = 100$, compared to other tool's default values. We therefore increase the value to $i_{CGS} = 1000$. *Gensim*'s default value for the maximum number of iterations for updating the document-topic distribution within one E-step is $i_{VB} = 50$. The impact of the modified learning parameters on the MRR@10 is presented in Figure 3.

For *MALLET*, the increase of the number of Gibbs

iterations from 100 to 1000 does not affect the average MRR@10 across the ten cross-validation splits which remains unchanged. This confirms the assumption that the LDA model trained by CGS has already converged previously and a lower number of iterations is indeed sufficient on the Mozilla Firefox dataset. In contrast, all OVB based implementations benefit from the changed parametrization, but still do not match the MRR@10 obtained using *MALLET*.

6.2 Learning Asymmetrical Dirichlet Priors

We will now focus on the shared hyperparameters between LDA and the ATM to compare DRETOM with our proposed algorithms, starting with the Dirichlet priors. All of the approaches presented in Section 2 use the default values of the Dirichlet priors, specified in the respective implementation, none of them therefore investigates the effect of an asymmetrical Dirichlet prior. However, Wallach et al. (2009) found that an asymmetrical Dirichlet prior over the document-topic distributions has substantial advantages over a symmetrical prior. In contrast, an asymmetrical Dirichlet prior over the topic-term distributions provides no known benefit. Learning an asymmetrical Dirichlet α prior from the data while keeping the β prior symmetrical increases the robustness of topic models to variations in the number of topics and to the highly skewed word frequency distributions common in natural language (Wallach et al., 2009). We therefore only con-
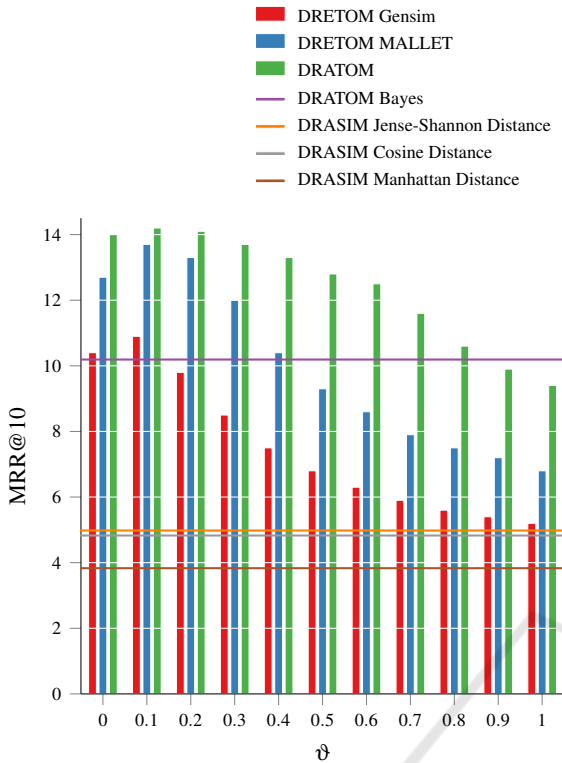
Figure 4: Impact of an optimized asymetric α prior on the MRR@10 obtained on the Mozilla Firefox project. DRATOM, DRATOMBayes and DRASIM use the same trained Author-Topic Model (ATM). The mean over a 10-fold cross-validation is reported.

sider the case of an asymmtrical prior α. Concretely, we learn an asymmetrical prior from the data using *Gensim*'s and *MALLET*'s built-in hyperparameter optimization.

Figure 4 shows the effects of learning an asymmetrical Dirichlet prior α for the task of developer recommendation measured as MRR@10.

The DRETOM using *Gensim* approach does not benefit significantly from the use of automatic hyperparameter optimization compared to the symmetrical α prior used in the previous experiments. In contrast, the DRETOM approach using *MALLET*'s hyperparameter optimization benefits noticeably more. The MRR@10 improves from 13.1 to 13.7 across all ten folds. The MRR@10 improves by over 30% on average when we use DRATOM (14.2) instead of DRETOM with the algorithmically comparable implementation of *Gensim* (10.9). This indicates that choosing a more appropriate topic model can significantly improve the quality of developer recommendations. Surprisingly, even with the VB-based implementation of the ATM, we manage to outperform the previously superior CGS-based LDA implementation of *MALLET* (13.7). We further observe that the opti-

mal value for the common hyperparameter θ changes abruptly at various times in the project. This is particularly pronounced for DRATOM using ATM, though similar behavior can be observed for DRETOM with both the *Gensim* and *MALLET* implementations. It is all the more astonishing that our DRATOMBayes approach (13.3) performs over 20% better than the comparable DRETOM implementation using *Gensim* and its optimal value θ=0.1 (10.9), although our approach neither knows nor needs to know the trade-off hyperparameter. Our novel approach DRASIM seems inferior and practically useless compared to all other methods independent of the distance metric used.

6.3 The Influence of the Number of Topics

The last remaining hyperparameter that needs to be considered is the number of latent topics *K*. Xie et al. applied a heuristic to determine the number of topics to *K* = 20 in their corpus (Xie et al., 2012). If we would adapt their method, we would use *K* = 30 as the number of topics. However determining the ideal number of topics for a topic model is still an open question, and should therefore be investigated in our experiments (Hasan et al., 2021).

Table 4 shows how increasing the number of topics affects the choice of the optimal value for θ for DRETOM and DRATOM as well as our other proposed approaches with regard to the MRR@10.

It becomes apparent that all compared approaches benefit from a larger number of topics but to different degrees. As in all previous experiments, the CGS-based LDA implementation of *MALLET* consistently outperformed the VB-based implementation of *Gensim* independently of the number of topics and θ by margins of up to 35%. The optimal value for DRETOM's hyperparameter is not constant, even when averaged across all cross-validation folds, and increases at different rates depending on the LDA implementations to θ = 0.2 at *K* = 60. This underlines the sensitivity of DRETOM from an optimal choice of θ which also showed this deficiency in all previous experiments. Even more affected by this problem is DRATOM, the same developer recommendation approach using ATM. However, the latter outperforms the DRETOM approach, regardless of the used LDA implementation, the number of topics, and even the value for θ by margins of up to 40%. If we compare DRATOMBayes with the similar VB implementation of DRETOM using *Gensim*, the superiority in the use of ATM over LDA becomes evident also in this case. On the other hand, if we compare DRATOMBayes with the fundamentally superior *MALLET* implemen-

Table 4: Impact of a varying number of topics K on MRR@10 obtained on the Mozilla Firefox project with 10-fold cross validation. DRATOM(ϑ), DRATOM and DRASIM use the same trained Author-Topic model (ATM). The mean over the cross-validation and standard deviation are reported. The best hyperparameter setting for ϑ and the best distance metric for DRASIM are highlighted. The table only shows the results for $\theta \leq 0.4$, as in not any case best results where achieved with a greater value for θ.

Model	Parameter	K=20	K=30	K=40	K=50	K=60
DRETOM *Gensim*	ϑ=0.0	10.4 ± 2.8	11.1 ± 2.7	11.6 ± 2.7	11.9 ± 3.2	11.9 ± 2.6
	ϑ=0.1	**11.0 ± 3.0**	**11.6 ± 2.9**	**12.0 ± 2.8**	**12.3 ± 3.4**	12.2 ± 2.8
	ϑ=0.2	10.1 ± 2.7	11.3 ± 2.9	11.8 ± 2.9	12.1 ± 3.5	**12.3 ± 2.9**
	ϑ=0.3	8.7 ± 2.1	10.3 ± 2.9	11.0 ± 2.7	11.5 ± 3.4	11.7 ± 2.8
	ϑ=0.4	7.7 ± 1.6	9.3 ± 2.7	10.0 ± 2.4	10.8 ± 3.1	10.9 ± 2.7
DRETOM *MALLET*	ϑ=0.0	12.7 ± 3.3	13.4 ± 3.1	14.3 ± 3.6	15.0 ± 3.9	15.8 ± 3.7
	ϑ=0.1	**13.7 ± 3.4**	**14.2 ± 3.1**	15.1 ± 3.8	15.8 ± 4.1	16.6 ± 3.9
	ϑ=0.2	13.2 ± 3.1	14.1 ± 2.7	**15.1 ± 3.7**	**15.8 ± 3.9**	**16.7 ± 3.9**
	ϑ=0.3	12.0 ± 2.7	13.1 ± 2.2	14.3 ± 3.4	15.0 ± 3.3	16.4 ± 3.6
	ϑ=0.4	10.3 ± 2.3	11.9 ± 2.0	13.0 ± 2.9	14.0 ± 2.5	15.6 ± 3.1
DRATOM(ϑ)	ϑ=0.0	14.0 ± 2.7	15.5 ± 3.8	15.8 ± 3.8	15.5 ± 3.8	16.0 ± 3.3
	ϑ=0.1	**14.2 ± 2.6**	15.8 ± 3.8	16.5 ± 3.8	16.1 ± 3.9	16.6 ± 3.4
	ϑ=0.2	14.0 ± 2.5	**15.9 ± 3.8**	16.9 ± 3.9	16.5 ± 3.9	17.1 ± 3.4
	ϑ=0.3	13.7 ± 2.4	15.8 ± 3.7	**16.9 ± 3.7**	16.7 ± 3.8	17.4 ± 3.3
	ϑ=0.4	13.2 ± 2.4	15.6 ± 3.6	16.8 ± 3.6	**16.7 ± 3.6**	**17.6 ± 3.2**
DRATOM	n/a	**13.3 ± 2.3**	**14.9 ± 2.9**	**16.0 ± 4.2**	**15.8 ± 4.2**	**16.1 ± 3.7**
DRASIM	Jensen-Shannon distance (D_{JS})	**6.7 ± 2.0**	**8.6 ± 1.5**	10.8 ± 2.6	**12.0 ± 2.4**	**12.7 ± 2.1**
	Cosine distance (D_{Cos})	6.6 ± 2.0	8.6 ± 1.6	**10.9 ± 2.7**	12.0 ± 2.3	12.6 ± 2.0
	Manhattan distance (D_{Man})	5.1 ± 1.9	6.9 ± 1.7	8.5 ± 2.3	10.1 ± 2.5	10.8 ± 2.3

tation of DRETOM, using the ATM instead of LDA, we are able to match the accuracy of the developer recommendations, even exceeding them in the case of 30 or 40 topics. Though DRASIM achieves worse results than all other approaches with regard to the MRR@10, it benefits most from an increasing number of topics and the quality of the developer recommendations improves by almost 90%.

6.4 Threats to Validity

The internal threat to validity mainly lies in the implementation of the studied baseline approach DRETOM. Our approach based on the ATM used the implementation provided by *Gensim*, which uses VB and its online version OVB. A direct comparison between the *MALLET*-based DRETOM and ours is therefore not possible. However, our first experiment indicates the prevalence of CGS over VB and its online version. Therefore, we think that an implementation of the ATM based on CGS can increase the competitiveness of our findings.

The main threat for generalization is that we reported our observations only from the conducted experiments on the Mozilla Firefox project. The dataset used for evaluation contains only the final developer that is assigned to a resolved bug report and we consider this developer as the ground truth. However, this could become problematic as bug resolving is often a team effort and multiple developers with different roles are involved. Furthermore, the ground truth data originates from the current process of manually bug triaging issues to developers and assumes that this assignment is optimal.

7 CONCLUSIONS AND FUTURE WORK

In this work, we applied the ATM, a variant of LDA taking authorship into account, for modeling a developer's expertise based on his activities stored in a bug tracking system. By examining patterns of co-occurring words, the ATM is able to capture semantic clusters in the vocabulary, so-called topics, which often describe concepts in the bug reports. Describing a developer as a distribution over these topics leads to interpretable developer profiles, thus motivating its use for the bug triaging task. Based on the ATM, we further proposed three novel bug triaging algorithms DRATOM, DRATOMBayes, and DRASIM, of which the first two originate from an approach proposed by Xie et al. (2012). We conducted experiments on a large dataset from the Mozilla Firefox project to evaluate the effect of our approaches. The results show that

1. The chosen topic modeling library can have a

large impact on the bug triaging results.

2. The approaches based on the ATM are able to outperform the *DRETOM* approach

3. The hyperparameters α and K can have a significant impact on the developer recommendation results.

One direction for future work would be the modification of our approaches DRATOM, DRATOMBayes and DRASIM by taking additional categorical features of bug reports into account. There are also topic models that were conceptualized for short documents, that bugs are associated with. Further experiments should be conducted, where our approaches are then compared to more modern approaches. Furthermore, datasets with more than one assigned developer for each bug report would be interesting to observe with regards to a different definition of the ground truth.

ACKNOWLEDGEMENTS

REFERENCES

Aggarwal, C. C. and Zhai, C. (2012). *Mining text data.* Springer Science & Business Media.

Aktas, E. and Yilmaz, C. (2020). Automated issue assignment: results and insights from an industrial case. *Empirical Software Engineering*, 25(5):3544–3589.

Bhattacharya, P. and Neamtiu, I. (2010). Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE.

Blei, D., Ng, A., and Jordan, M. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Chen, T.-H., Thomas, S. W., and Hassan, A. E. (2016). A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21(5):1843–1919.

Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. volume 101, pages 5228–5235.

Hasan, M., Rahman, A., Karim, M. R., Khan, M. S. I., and Islam, M. J. (2021). Normalized Approach to Find Optimal Number of Topics in Latent Dirichlet Allocation (LDA). In *Advances in Intelligent Systems and Computing*, volume 1309, pages 341–354. Springer Science+Business Media.

Hoffman, M., Bach, F., and Blei, D. (2010). Online learning for latent dirichlet allocation. *Advances in Neural Information Processing Systems*, 23:856–864.

Hu, H., Zhang, H., Xuan, J., and Sun, W. (2014). Effective bug triage based on historical bug-fix information. In *2014 25th International Symposium on Software Reliability Engineering*, ISSRE, pages 122–132. IEEE.

Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S., and Runeson, P. (2016). Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, 21(4):1533–1578.

Kagdi, H., Gethers, M., Poshyvanyk, D., and Hammad, M. (2012). Assigning change requests to software developers. *Journal of Software: Evolution and Process*, 24(1):3–33.

Khatun, A. and Sakib, K. (2016). A bug assignment technique based on bug fixing expertise and source commit recency of developers. In *2016 19th International Conference on Computer and Information Technology*, ICCIT, pages 592–597. IEEE.

Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., and Baldi, P. (2009). Sourcerer: Mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336.

Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P. (2007). Mining eclipse developer contributions via author-topic models. In *Proc. 4th International Workshop on Mining Software Repositories*, MSR, pages 1–4.

Mani, S., Sankaran, A., and Aralikatte, R. (2019). Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *2019 India Joint International Conference on Data Science and Management of Data*, pages 171–179. ACM.

Matter, D., Kuhn, A., and Nierstrasz, O. (2009). Assigning bug reports using a vocabulary-based expertise model of developers. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR, pages 131–140. IEEE.

Naguib, H., Narayan, N., Brügge, B., and Helal, D. (2013). Bug report assignee recommendation using activity profiles. In *2013 10th Working Conference on Mining Software Repositories*, MSR, pages 22–30. IEEE.

Newman, D., Asuncion, A., Smyth, P., and Welling, M. (2009). Distributed Algorithms for Topic Models. *Journal of Machine Learning Research*, 10:1801–1828.

Nguyen, T. T., Nguyen, A. T., and Nguyen, T. N. (2014). Topic-based, time-aware bug assignment. *ACM SIGSOFT Software Engineering Notes*, 39(1):1–4.

Ramage, D., Hall, D., Nallapati, R., and Manning, C. D. (2009). Labeled lda: A supervised topic model for

credit attribution in multi-labeled corpora. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, pages 248–256.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50. ELRA.

Rosen-Zvi, M., Griffiths, T., Steyvers, M., and Smyth, P. (2004). The author-topic model for authors and documents. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI, pages 487–494. AUAI Press.

Sajedi-Badashian, A., Hindle, A., and Stroulia, E. (2015). Crowdsourced bug triaging. ICSME, pages 506–510. IEEE.

Sajedi-Badashian, A. and Stroulia, E. (2020). Guidelines for evaluating bug-assignment research. *Journal of Software: Evolution and Process*, 32(9).

Schofield, A., Magnusson, M., and Mimno, D. (2017a). Pulling Out the Stops: Rethinking Stopword Removal for Topic Models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 432–436, Stroudsburg, PA, USA. Association for Computational Linguistics.

Schofield, A., Magnusson, M., Thompson, L., and Mimno, D. (2017b). Understanding Text Pre-Processing for Latent Dirichlet Allocation. *Proceedings of the 15th conference of the European chapter of the Association for Computational Linguistics*, 2:432–436.

Schofield, A. and Mimno, D. (2016). Comparing Apples to Apple: The Effects of Stemmers on Topic Models. *Transactions of the Association for Computational Linguistics*, 4:287–300.

Schofield, A., Thompson, L., and Mimno, D. (2017c). Quantifying the Effects of Text Duplication on Semantic Models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2737–2747. Association for Computational Linguistics.

Shokripour, R., Anvik, J., Kasirun, Z., and Zamani, S. (2013). Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation. In *2013 10th Working Conference on Mining Software Repositories*, MSR, pages 2–11. IEEE.

Sommerville, I. (2016). *Software Engineering*. Pearson Education.

Sun, X., Liu, X., Li, B., Duan, Y., Yang, H., and Hu, J. (2016). Exploring topic models in software engineering data analysis: A survey. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, SNPD, pages 357–362. IEEE.

Tamrawi, A., Nguyen, T. T., Al-Kofahi, J. M., and Nguyen, T. N. (2011). Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*, FSE/ESEC, pages 365–375.

Wallach, H. M. (2008). *Structured Topic Models for Language*. PhD thesis, Newnham College, University of Cambridge.

Wallach, H. M., Mimno, D., and McCallum, A. K. (2009). Rethinking LDA: Why Priors Matter. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS, pages 1973–1981.

Xia, X., Lo, D., Ding, Y., Al-Kofahi, J. M., Nguyen, T. N., and Wang, X. (2016). Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297.

Xia, X., Lo, D., Wang, X., and Zhou, B. (2013). Accurate developer recommendation for bug resolution. In *2013 20th Working Conference on Reverse Engineering*, WCRE, pages 72–81. IEEE.

Xie, X., Zhang, W., Yang, Y., and Wang, Q. (2012). Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, PROMISE, pages 19–28.

Yang, G., Zhang, T., and Lee, B. (2014). Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 97–106. IEEE.

Yao, L., Mimno, D., and McCallum, A. K. (2009). Efficient methods for topic model inference on streaming document collections. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 937–945.

Zhang, J., Wang, X. Y., Hao, D., Xie, B., Zhang, L., and Mei, H. (2015). A survey on bug-report analysis. *Science China Information Sciences*, 58(2):1–24.

Zhang, T., Chen, J., Yang, G., Lee, B., and Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software*, 117:166–184.

Zhang, T., Yang, G., Lee, B., and Lua, E. K. (2014a). A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In *2014 21st Asia-Pacific Software Engineering Conference*, pages 223–230. IEEE.

Zhang, W., Han, G., and Wang, Q. (2014b). Butter: An approach to bug triage with topic modeling and heterogeneous network analysis. In *2014 International Conference on Cloud Computing and Big Data*, pages 62–69. IEEE.

Zou, W., Lo, D., Chen, Z., Xia, X., Feng, Y., and Xu, B. (2020). How Practitioners Perceive Automated Bug Report Management Techniques. *IEEE Transactions on Software Engineering*, 46(8):836–862.