The Journal of Systems and Software 113 (2016) 296–308

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project

Meng Yan[b], Ying Fu[b], Xiaohong Zhang[a,b,c,*], Dan Yang[b], Ling Xu[b], Jeffrey D. Kymer[b]

[a] Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing 400044, PR China
[b] School of Software Engineering, Chongqing University, Huxi Town, Shapingba, Chongqing 401331, PR China
[c] State Key laboratory of Coal Mine Disaster Dynamics and Control, Chongqing 400044, PR China

ARTICLE INFO

Article history:
Received 14 November 2014
Revised 11 November 2015
Accepted 7 December 2015
Available online 22 December 2015

Keywords:
Software change classification
Multi-category change
Discriminative topic model

ABSTRACT

Accurate classification of software changes as corrective, adaptive and perfective can enhance software decision making activities. However, a major challenge which remains is how to automatically classify multi-category changes. This paper presents a discriminative Probability Latent Semantic Analysis (DPLSA) model with a novel initialization method which initializes the word distributions for different topics using labeled samples. This method creates a one-to-one correspondence between the discovered topics and the change categories. As a result, the discriminative semantic representation of the software change messages whose largest topic entry directly corresponds to the category label of the change message which is directly used to perform single-category and multi-category change classification. In the evaluation on five open source projects, the experimental results show that the proposed approach achieves a more accurate performance than the four baseline methods. Especially with the multi-category classification task which improves the recall rate. Moreover, the different projects share the same vocabulary and the estimated model so that DPLSA is well applicable to cross-project software change message analysis.

1. Introduction

To aid further software analysis, it is necessary to classify software change as corrective, adaptive, or perfective. The proportion of each category provides a valuable window into the software development practices. Project managers need to be well informed to enhance their decision making process. For example, if 90% of the changes in a project are corrective, then it may mean that now is the time to intensify the quality assurance work like code reviews and unit tests. It has been applied to many important software engineering activities, such as software maintenance (Mockus and Votta, 2000) and defect prediction (Kim et al., 2008). Various change cues have been used for classifying software changes, for example, change author (Hindle et al., 2009a), change file (Alali et al., 2008), change size (Hattori and Lanza, 2008) and change messages (Hassan, 2008). In particular, change messages are attractive for software change classification because it does not require retrieving and then analyzing the source code of the change. Moreover, retrieving only the message is significantly less expensive, and allows for efficient browsing and analysis of the changes and their constituent revisions. These characteristics are useful to anyone who needs to quickly categorize or filter out irrelevant revisions (Hindle et al., 2009a). However, to the best of our knowledge, there has been little work classifying a multi-category change; but, there have been researches that have found this to be a realistic activity (Fu et al., 2015; Mauczka et al., 2012). In this work, we aim to accurately understand the category distribution by classifying both single-category and multi-category changes.

Researchers have proposed a variety of approaches for retrieving keywords in change messages to classifying software changes (Hassan, 2008; Mauczka et al., 2012; Mockus and Votta, 2000). Despite the great success achieved, there are some unsolved issues remaining in this research, such as the ambiguity coming from subjective interpretations of the relationship between relevant words and categories of changes. A similar work has demonstrated success in automatic software change classification by using semi-supervised Latent Dirichlet Allocation (LDA) (Fu et al., 2015). We noticed that both Mauczka et al. (2012) and Fu et al. (2015) found that single-category changes are not necessarily realistic. A major challenge which remains is how to automatically classify multi-category changes. To address this challenge, we focus on automatically classifying software changes by developing a novel discriminative Probability Latent Semantic Analysis, referred to DPLSA. The main difference from Fu et al. (2015) is that the three topics in this work have a one-to-one correspondence to

* Corresponding author. Tel.: +86 15923238399.
E-mail address: xhongz@cqu.edu.cn (X. Zhang).

http://dx.doi.org/10.1016/j.jss.2015.12.019
0164-1212/

corrective, adaptive and perfective software change categories, such that the change message categorization comes down to finding the single maximum entry (single-category) or multi maximum entries (multi-category) in the topic-document distributions, and we provide a method of cross-project classification without the need of re-learning. In particular, we motivated our investigation with three research questions:

RQ1 *What is a better way to evaluate the relationship between relevant words and the categories of software changes?* A change message often is a short description written by developers and the VCS does not enforce how to write a change message. Consequently, change messages are non-structured free format text. There are many salient words relevant to categories in change messages, such as "fix", "create" and "correct". The relationship between the relevant words and the categories is the key issue in the classification step. Mauczka et al. (2012) assigned weights to the salient words which is a subjective interpretation. We wish to perform a cross-project training using labeled messages to automatically determine a probabilistic relationship.

RQ2 *How well do the discovered topics correspond to software changes with multi-category?* A change message indicates a particular maintenance task, such as fixing a defect or adding a new feature, despite the fact that there exists a few change messages, which indicate multiple purposes as Mauczka et al. (2012) and Fu et al. (2015) presented in their validation step. We wish to create a one-to-one correspondence between discovered topics and categories by using the discriminative topic model. After that, the discovered topics can be directly used to perform the classification task including single-category and multi-category.

RQ3 *What is an accurate way to automatically obtain the distribution of software changes?* A project manager would be interested in knowing the distribution of categories of software changes. We wish to quantify a more accurate distribution by classifying both single-category and multi-category changes.

We address our research questions by proposing a topic modeling method. It is inspired by the recent success of topic modeling in mining software repositories (Grant et al., 2012; Hindle et al., 2011; Hindle et al., 2009b; Pollock et al., 2013; Thomas, 2012). Topic models, such as Probability Latent Semantic Analysis (Hofmann, 2001), Latent Dirichlet Allocation (Blei et al., 2003), Correlated Topic Models (Lafferty and Blei, 2006) and their variants and extensions, have been applied to various software engineering research questions, such as software evolution and software defect prediction (Chen et al., 2012; Gethers and Poshyvanyk, 2010; Grant et al., 2012). Despite the great success achieved, there are some unsolved, important issues that still remain in this line of research. First, in the original topic models, some words which are fully connected to different topics are noisy and irrelevant for model construction. Disconnecting the irrelevant words is helpful for generating a sparse representation over different topics of a document (Chien and Chang, 2014). In fact, the sparsity of the topic-document distribution (i.e. with a small number of dominant entries and most zero or close to zero entries) is helpful for directly performing the classification task. Second, a critical issue in understanding the latent topics uncovered from software repositories is how many topics should be sought (Grant et al., 2013). There is not a one-to-one correspondence between topics and category labels in the traditional models. The topic-document distributions are only used to decide which topics are important for a particular document and cannot determine which category a particular document belongs to. This is also the limitation in solving the multi-category problem.

The process of the proposed DPLSA is divided into three phases as illustrated in Fig. 1. We select single category change messages as our training datasets and use the semantically salient words derived from the work of Mauczka et al. (2012) to form the vocabulary as illustrated in Fig. 1(a). Moreover, the training messages from the same category are employed to initialize the category-conditional probability of a specific word conditioned on the corresponding topic. Hence, semantically salient words are forced to connect to the topic partially with a dominated probability. Such that, it creates a one-to-one correspondence between topics and categories. Due to the special initialization approach, the sparsity is achieved for the corresponding words to the corresponding topics (Chien and Chang, 2014). Finally, the topic representation of a test sample is sparse and its maximum entry directly determines the category to which the test sample belongs because the topic is the same as the category. When multiple topic entries of a change message reach the same maximum, the change message is regarded as a multi-purpose one.

In our experiments, change messages of five open source projects are extracted by using the CVSAnalY (Robles et al., 2004) tool. The change message is normalized by WordNet (Miller, 1995) and Gate (Cunningham et al., 2002). The five different projects in the experiment shared the same vocabulary and the estimated model, and moreover the sparse probabilistic representation of software change messages were directly used to assign software changes into Swanson's maintenance categories (Swanson, 1976) by finding the maximum topic entry. The proposed approach is proved capable of classifying changes well through manual validation performed by professional developers. Especially, the multi-category change classification task that improves the recall rate. In summary, the contributions of this paper can be summarized as follows:

- We explore the discovered word-topic distributions learned from labeled change messages and find they provide an ordered probabilistic relationship between relevant words and the categories of software changes. As a result, this overcomes the ambiguity coming from manually subjective weights.
- We explore the discovered topic-document distributions and find a one-to-one correspondence between these discovered topics and change categories. The maximum topic entry directly determines the category to which a change belongs. If multiple topic entries reach the same maximum, this indicates a change is a multi-category one.
- We evaluate our approach on five projects and compare the performance with four baselines. The results indicate that our performing multi-category classification improves the classification performance. As a result, this work provides a more accurate distribution of each category in a project. Besides, we provide a method of cross-project software change classification without the need for re-learning. The different projects share the same vocabulary and the estimated model.

The structure of this paper is as follows. In Section 2 we present the related work of our research, including previous software change classification methods, software change classification rules, topic modeling in mining software repositories (MSR), and PLSA. We describe our research preparation, models and techniques in Section 3. In Sections 4 and 5, we provide the experiment design, results and validation. Then at last in Sections 6 and 7, we discuss the potential threats to our findings and draw a conclusion.

2. Related work

In this section, we discuss related literature from several aspects: previous software change classification methods, software

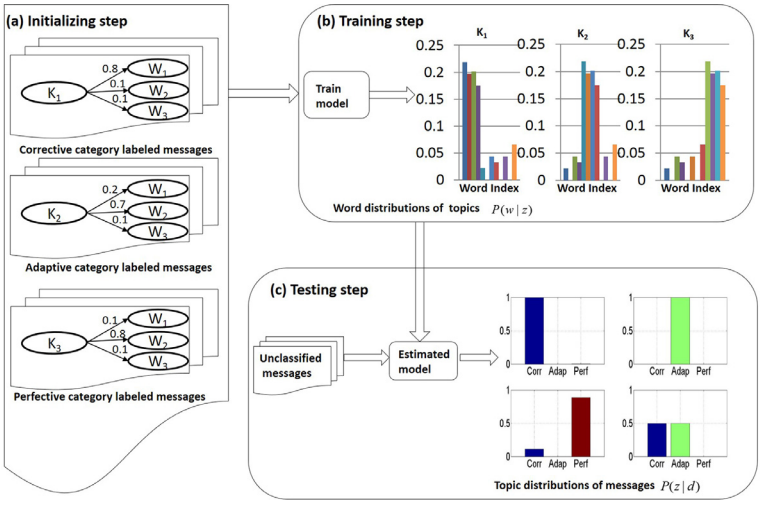

Fig. 1. The software change classification process using DPLSA. (a) Three categories of the training sample messages from the same category are used to initialize the category-conditional probability of a specific word conditioned on the corresponding topic, where K1, K2, K3 indicates corrective, adaptive and perfective, and W1, W2, W3 indicates three group words derived from the work of Mauczka et al. (2012), respectively. (b) The sparse word distributions of each topic are estimated by the standard EM algorithm in PLSA. (c) The topic distributions of each test message are computed by fixing the estimated words distribution in PLSA. The maximum topic entry determines to which category this message belongs. When multiple topic entries reach the same maximum, this message is classified as a multi-category one.

change classification rules, topic modeling in mining software repositories (MSR), and Probabilistic Latent Semantic Analysis.

2.1. Previous software change classification methods

A variety of approaches have been presented to classify software change. In 2000, Mockus and Votta presented the first textual approach for classifying software changes (Mockus and Votta, 2000). Their approach was based on the keywords in the textual message of the changes. For example, if keywords such as “fix”, “correct”, “error” or “fail” were presented in a change message, the change was classified as corrective. The validation surveys showed that 61% of the time, their automatic classification results were in agreement with developer opinions. Hassan (2008) extended Mockus’s approach. A case study using change messages from several open source projects showed that his approach produced results similar to manual classifications performed by professional developers. Hattori and Lanza (2008) investigated the relationship between the change size and the change category. They found that the majority of tiny commits were not related to development activities. Corrective actions are the ones that generate more tiny commits. Also, Hindle et al. (2009a) provided another classification model. They took the terms distribution, author, module, and file type as features. It was shown that the author’s identity may be significant for identifying the purpose of a change.

Along with the progress of methodology, an Eclipse plug-in named Subcat which also used the features of the word distribution to classify the change messages was developed by Mauczka et al. (2012). Subcat can automatically assess if a change to the software was due to a bug fix or refactoring based on a set of keywords in the change messages. In addition, Subcat further introduced a weighting of keywords and rule sets for ambiguous, yet strongly indicative words. It is noted that every weight of keywords in the Subcat was decided using a biased manual method. Our work differs from Mauczka’s work in that the relationship in our model is not a simply weight value like 1 or 2, it is presented as a probabilistic value for each topic, which can deal with ambiguity.

In the classification algorithms mentioned above, the classification result is a definite 0 or 1 result. However, sometimes, even the developer himself cannot definitely classify his commit message into Swanson’s maintenance categories (Swanson, 1976). Hence, a probability weight and result would be more insightful (Fu et al., 2015). Fu et al. (2015) provided a topic model based method which measured the distance between probability distributions to perform change classification by using semi-supervised Latent Dirichlet Allocation (LDA). Both Fu et al. (2015) and Mauczka et al. (2012) have presented that there exists some software changes, which are conducted for multiple purposes. For example, the content of *1817-th* change message in Bugzilla is “Bug 155343: header template interface comment correction; extra parameter renamed to header_html.Note; the patch on the bug didn’t apply cleanly to branch anymore; fixed manually.2xr=bbaetz.” This indicated the change was conducted for correcting bug 155343 and doing some refactoring work at the same time. Although Fu et al. (2015), Mauczka et al. (2012), Hindle et al. (2008) and Mauczka et al. (2015) have addressed the multi-category changes, one issue is that they identify the multi-category changes in their manually labeling step rather than classifying them in an automatic way. How to classify the multi-category changes in an automatic way is rarely addressed. The difference between their works and this work is that this work performs both single-category and multi-category change classification in an automatic way.

2.2. Software change classification rules

A software change is an activity which is concerned with maintaining a software system. The research area of the identification and classification of such software changes has been evolving for decades. The earliest works about the classification rules is “The Dimensions of Maintenance” (Swanson, 1976) by Swanson. He defines a maintenance task as a change that is classified into one of the following three categories:

- Corrective: These changes are made to fix processing failures, performance failures or implementation failures. For example a bug fix or the correction of a typing error.
- Adaptive: These changes focus on the changes in the data environment or processing environment. For example the introduction of a new function.
- Perfective: These changes are made to improve processing efficiency, enhance the performance or increase the maintainability.

For the development of the automated classification for software change, Swanson's original definition of maintenance tasks is used and slightly extended. A modified categorization is provided by Hassan (2008). In Hassan's work, a change record is classified as one of the following three categories.

- Bug Fixing change (BF): These changes are done to fix a bug.
- General Maintenance change (GM): These changes are mainly about book keeping changes and do not reflect the functions and features.
- Feature Introduction changes (FI): These changes focus on adding or enhancing features.

Hindle et al. extended Swanson's categorization into corrective, adaptive and perfective changes with two additional changes (Hindle et al., 2009a).

- Feature addition: New requirements.
- Non-functional: Legal Source Control System management and code clean-up.

Our method is based on the technique of topic modeling. The classification categories are required specific and independent enough. Hassan's categories are more understandable, but the general maintenance (GM) activity is not specific enough. While Hindle's extension work is more practical, but there are some overlap between categories, such as feature addition and adaptive change (Swanson, 1976).

Our approach relies on two sources of information which carry out the classification, namely the change message and semantically salient words of the three categories. The change message is attached to every software change by developers and encapsulates the category of the modification. The semantically salient words were made by adopting the work of Mauczka et al. (2012). In addition, their final keywords dictionary was validated with cross-project analysis. Since we are dependent on the final keywords dictionary by Mauczka et al. (2012), we adopted Swanson's (Swanson, 1976) original definition of maintenance tasks which has been widely used among mining software repositories researches. Moreover, a fourth category which indicates the message cannot be classified in this method, namely Not Sure (NS) was added. It was also adopted and defined by Hassan.

2.3. Topic modeling in mining software repositories

Mining Software Repositories (MSR) is a technique that focuses on analyzing and understanding these data repositories, which are related to the software development lifecycle. The idea of extracting higher-level concepts, aspects, or topics from software repositories has been approached in recent years. Thomas (2012) provided a long technical-report on topic modeling in mining software repositories. He surveyed the past decade of 71 highly-relevant software engineering documents and collected dozens of attributes on each to explore how topic models have helped researchers in their efforts of mining software repositories. In his work, several challenges for bringing topic models into the domain of software engineering were addressed: understanding the peculiarities of the data in this domain; choosing the right parameters (e.g., number of topics); and making results of topic models easier to interpret.

Software projects produce several types of data repositories during their lifecycles, such as mailing-lists, bug reports, source code, and change messages. Topic models have been applied on such data repositories to guide diverse software engineering activities. On mailing-lists and bug reports mining, Asuncion et al. (2010) proposed an automated technique that recovers traceability links between mailing-lists and diverse artifacts with Latent Dirichlet Allocation. Ahsan et al. (2009) presented a technique behind an automatic bug triage system, which was based on the categorization of bug reports. They reduced the dimensionality of the obtained term-to-document matrix by applying feature selection and latent semantic indexing methods. On source code mining, Savage et al. (2010) provided a Topic XP tool which can support developers during software maintenance tasks by extracting and analyzing unstructured information in source code identifier names and comments using Latent Dirichlet Allocation. Gethers et al. (2011) provided another topic tool in source code namely CodeTopics. It is an Eclipse plug-in that in addition to showing the similarity between source code and related high-level artifacts (HLAs) also highlights to what extent the code under development covers topics described in HLAs using Latent Dirichlet Allocation. It can help developers to identify functionality that are not implemented yet or newcomers to comprehend source code artifacts by showing them the topics. Considering software change messages, Hindle et al. (2009b) introduced a windowed topic analysis approach which can extract a set of topics by analyzing change messages. They demonstrated its utility compared to global topic analysis. By using a defined time-window of, for example, one month, their approach can track which topics come and go over time. Another approach presented by Hindle et al. (2011) is to label change messages with non-functional requirements by supervised Latent Dirichlet Allocation. They introduced the concept of supervised topic extraction, using a non-functional requirement (NFR) taxonomy as prior knowledge which is based on the ISO quality model Commission (2001) for their labels.

A common issue in most of the above works is that they adopted the original topic model approach. The original goal of the topic model was not for inference or classification, but rather representation and compression of signals. It is hard to define what a topic is or interpret each topic (Ramage et al., 2009). In this work, in order to build a discriminative model, we brought the peculiarities of the domain data into the training step to make each topic corresponds to each category which is useful for decision-making tasks such as classification.

2.4. Probabilistic latent semantic analysis

The Probabilistic Latent Semantic Analysis provides a probabilistic formulation to model documents in a text collection. It assumes that the words are generated from a mixture of latent topics which are decomposed from a document. The PLSA ignores the orders of words occurring in a document. We briefly outline the principle of the PLSA and our work in this subsection. More details of PLSA can be found in Hofmann (2001).

Notations: Here, we use the simple notations described by Lu et al. (2011), and in this paper the two terms for message and document are identical. The PLSA assumes that all the documents in the collection D fit into a finite set of K topics and each topic z is associated with a multinomial word distribution $P(w|z)$ over the vocabulary V. A document d is represented as a bag of words. We use $P(z|d)$ for the distribution over topics z in a particular document and $P(w|z)$ for the probability distribution over words given topic z. To simplify the notations, let $\phi_w^{(j)} = P(w|z = j)$ refer to the multinomial distribution over words for topic$j$ and $\theta_j^{(d)} = P(z = j|d)$ refer to the multinomial distribution over topics for document d.

For PLSA, a document d is regarded as a sample of the following mixture model.

$$P(w|d) = \sum_{j=1}^{K} \phi_w^{(j)} \theta_j^{(d)} \tag{1}$$

The Expectation-Maximization (EM) algorithm (Dempster et al., 1977) is used for estimating the word-topic distributions $\hat{\phi}$ and

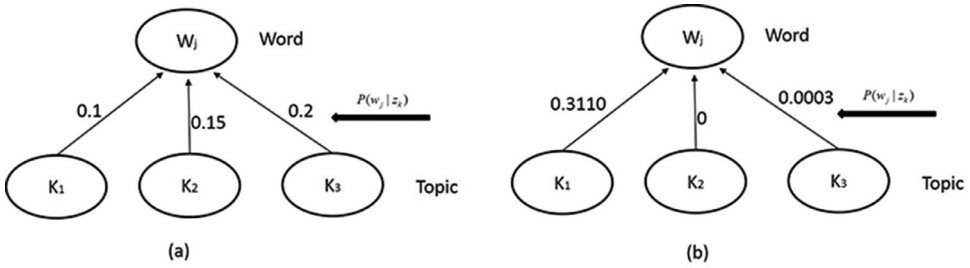

Fig. 2. (a) Illustration for the original PLSA in case that a word w_j is assigned using three topics k_1, k_2 and k_3. All three topics are connected to word w_j. (b) Illustration for the partially-connected network in the training step. Only one topic is connected to the word w_j with a dominated probability.

topic-document distributions $\hat{\theta}$ by maximizing the likelihood that the collection D is generated by this model:

$$\log P(D|\phi,\theta) = \sum_{d\in D}\sum_{w\in V}\left\{c(w,d)\log\sum_{j=1}^{K}\phi_w^{(j)}\theta_j^{(d)}\right\} \tag{2}$$

where $c(w, d)$ is the number of times word w occurs in document d. In the E-step, the hidden variable z is estimated based on the model parameter at the previous iteration:

$$P(z_{d,w}=j) = \frac{\phi_w^{(j)}\theta_j^{(d)}}{\sum_{j'=1}^{K}\phi_w^{(j')}\theta_{j'}^{(d)}} \tag{3}$$

Then, in the M-step, the parameters $\theta_j^{(d)}$ and $\theta_j^{(d)}$, respectively, are updated as:

$$\theta_j^{(d)} = \frac{\sum_{w\in V}c(w,d)P(z_{d,w}=j)}{\sum_{j'}\sum_{w\in V}c(w,d)P(z_{d,w}=j')} \tag{4}$$

$$\phi_w^{(j)} = \frac{\sum_{d\in D}c(w,d)P(z_{d,w}=j)}{\sum_{w'\in V}\sum_{d\in C}c(w',d)P(z_{d,w'}=j)}. \tag{5}$$

While PLSA is useful for analyzing message data, it is necessary to note several issues of PLSA. First, the number of topics needs to be pre-defined by the user. The performance is sensitive to the number of topics used (Lu et al., 2011). Second, PLSA ignores the features of training samples for the classification problem, such as the structure of the training samples from a single category, relationship between the samples from a single category and the word-topic distributions, relationship between topics and categories. Third, PLSA randomly initializes the word-topic distributions $\phi_w^{(j)}$ so that the words in a document are fully connected to different topics as shown in Fig. 2(a). It is difficult to obtain the very sparse distributions $\hat{\phi}_w^{(j)}$ and $\hat{\theta}_j^{(d)}$ in PLSA because the fully-connected network between words and topics is generally affected by some noisy, irrelevant or redundant features and topics (Chien and Chang, 2014). It is a worth noting that the sparsity of $\hat{\theta}_j^{(d)}$ is necessary to directly perform classification using $\theta_j^{(d_{new})}$. Furthermore, $\theta_j^{(d_{new})}$ only indicates which topics are relevant for d_{new} not to which category d_{new} belongs. That is, $\theta_j^{(d_{new})}$ cannot be directly used for performing the classification task.

In the following three sections, we address these questions by choosing the salient words in the software change domain to form the vocabulary and designing a word-topic distribution initialization method which sufficiently captures the features of the software change messages. Thus our findings offer insights about how to obtain the topic-document distribution with discriminative decision power and perform both single-category and multi-category classification task directly using $\theta_j^{(d_{new})}$ of DPLSA.

3. Research methodology

Two steps are necessary to conduct a DPLSA model of a software change repository: (1) the change message extracting and preprocessing step and (2) the topic modeling step. These steps are detailed below and illustrated in Fig. 3.

3.1. Data extracting and preprocessing

In this paper, change messages were extracted by CvsAnalY Robles et al. (2004) and preprocessed by WordNet Miller (1995) and GATE Cunningham et al. (2002). The preprocessing step consists of sentence splitting, term splitting, stop words filtering, and stemming. The stop words which cannot provide the category information were filtered, such as prepositions and pronouns.

3.2. Topic modeling

Two steps are necessary in our topic modeling step: (1) construct vocabulary and (2) build DPLSA model.

3.2.1. Step 1: constructing vocabulary

The first step of topic modeling is building the vocabulary. Each word in a message is viewed as a feature which is represented by a fixed set of topic mixtures. The mixture weights are used to build a coordinate vector of a word in semantic or topic space. In unsupervised topic modeling, the vocabulary consists of the top N words ordered by descending frequency in the whole corpus. However, some words like “be” occur in most messages, it does not provide any information about the category of the change. Therefore, constructing a salient words based vocabulary for topic modeling is more insightful.

Researchers like Mockus and Votta (2000) presented a set of salient keywords which can represent the maintenance type. However their words were narrow terms which were more suitable for commercial projects. Mauczka et al. (2012) provided a change classification dictionary which was validated by cross-project analysis. To facilitate cross-project analysis, the semantically salient words in Table 1 are analyzed and selected derived from the dictionary of Mauczka et al. (2012). The differences between this work and Mauczka et al. (2012) are as follows: First, the salient words in this work are utilized to form the cross-project vocabulary without using the category. While each word in their work has a category label which is directly used to perform the classification task. Second, the salient word in this work is assigned a probabilistic relationship to three categories automatically after training. While each word in their work is assigned a weight (either 1 or 2) correlated with a category relying on an empirical way.

3.2.2. Step 2: building DPLSA

In the training phase of DPLSA, the messages in the training set were examined and labeled as corrective, adaptive and perfective.

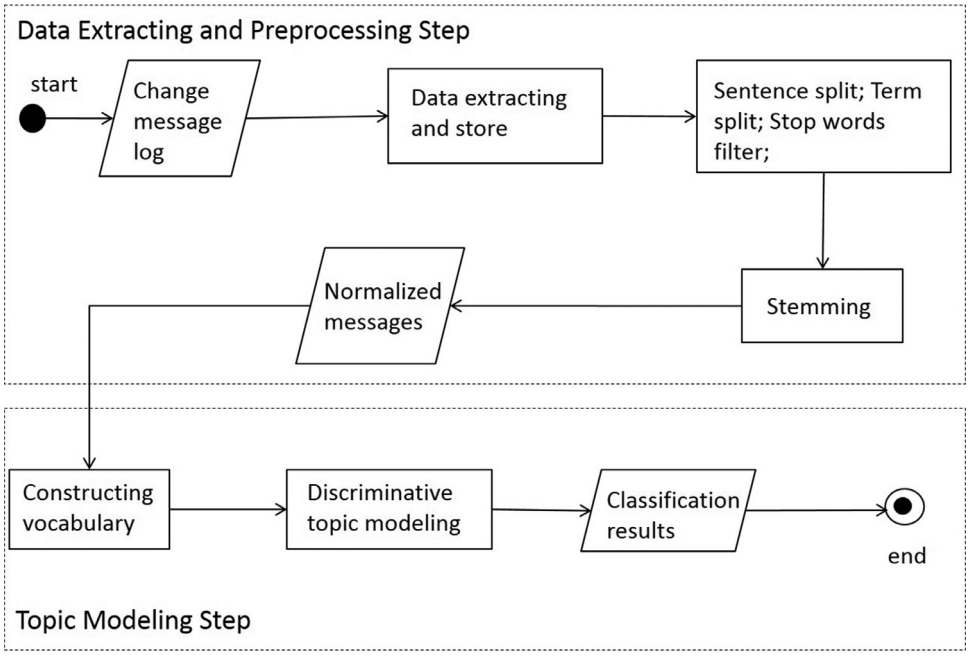

Fig. 3. Research methodology process view.

Table 1
Semantically salient keywords in the cross-project vocabulary.

Bug cause error failure fix bugfix miss null warn wrong bad correct incorrect problem valid invalid fail bad dump except add new create feature function appropriate available change compatibility configuration text current default future information method necessary old patch protocol provide release replace require security simple structure switch context trunk useful user version install introduce faster init clean cleanup consistent declaration definition documentation move template prototype remove static style unused variable whitespace header rename include dead inefficient useless

This labeling step was manually analyzed by professional developers according to Swanson's category definition. Set $D^{(j)}$ $(j = 1, 2, 3)$ denotes the collections of corrective, adaptive and perfective, respectively. Obviously, $D^{(j)}$ is a subset of D.

Our objective is to classify a new message to a specify category. We can treat each topic in PLSA as one category. The maximum entry of the topic document distribution $\theta_j^{(d_{new})}$ determines the category of a new change message. Thus, we simply specify the number of topics equal to the number of categories.

In the initialization, given sufficient training samples from the jth category $D^{(j)}$ the word-topic distribution $\phi_w^{(j)}$ is initialized as Formula (6) shows. This differs from the random initialization in PLSA.

$$\phi_w^{(j)} = c(w, D^{(j)})/c(D^{(j)}) \tag{6}$$

$$c(w, D^{(j)}) = \sum_{d \in D^{(j)}} c(w, d), \quad c(D^{(j)}) = \sum_{v \in V} \sum_{d \in D^{(j)}} c(v, d) \tag{7}$$

where $c(w, D^{(j)})$ is the number of times word w occurs in the collection $D^{(j)}$, $c(D^{(j)})$ is the number of times all words in the vocabulary V occur in the collection $D^{(j)}$.

As we know, the subspace models are flexible enough to capture much of the variation in real datasets. Similar to the work of Wright et al. (2009), DPLSA also exploits the structure of the training samples from a single category and assumes that the training samples from a single category lie in one subspace. This implies that $\phi_w^{(j)}$ lies in the same subspace spanned by the training samples from the jth category. In addition, this reflects the structure of the training samples from the jth category and discovers which words are informative for this category.

Initialization is a crucial stage in the framework of the EM algorithm as it determines the subsequent convergence of the algorithm. In the initialization, we treat $\phi_w^{(j)}$ in Formula (6) as the initialized base for each subspace. It indicates a one-to-one correspondence between topics and categories (i.e., the first, second and third topic corresponds to corrective, adaptive and perfective, respectively). Since EM finds a local optimum, after the EM algorithm converges, the estimated $\phi_w^{(j)}$ is the local optimum which is close to the initialized $\phi_w^{(j)}$. It serves as the final base for each subspace which reflects the most informative words for this category. In this manner, we treat the first, second and third topic in the estimated $\phi_w^{(j)}$ as corrective, adaptive and perfective, respectively.

From Formulas (6) and (7), DPLSA initializes $\phi_w^{(j)}$ using the training samples from a single category such that word w uniquely connects with a topic or category j and the redundant features are pruned for $\phi_w^{(j)}$. In this manner, DPLSA reduces redundant connections between topic or category j. The estimated word distribution $P(w|z)$ over the vocabulary V turns out to be a sparse matrix containing several components with near zero values and $\hat{\phi}_w^{(j)}$ effectively reflects which words are informative for topic$j$. Given a test message d_{new} and fixing the estimated value $\hat{P}(w|z)$ from DPLSA, we obtain the topic document distribution $\theta_j^{(d_{new})}$ of the test message d_{new} by the EM algorithm. The topic coverage distribution for each message serves as an alternative discriminative semantic representation of the message, which is potentially better than the original word-based representation in supporting classification of software change messages because topics directly correspond to categories. The topic-document distributions $\theta_j^{(d_{new})}$ indicate this particular document d_{new} belongs to which category. In other words, $\theta_j^{(d_{new})}$ possesses classification decision power. Thus,

Table 2
Experiment data list.

Projects	Application type	Start date	Collecting date	Devs	Changes
Bugzilla	Project management	Aug 1998	Mar 2012	62	27,539
Wireshark	Packet analyzer	Sep 1998	May 2012	43	40,551
Boost	Prog. library	July 2000	May 2012	294	42,208
Firebird	RDBMS	May 2001	May 2012	43	48,622
Python	Interpreter	Aug 1998	May 2012	216	45,032

the maximum entry of the topic document distribution $\theta_j^{(d_{new})}$ determines the category of a new change message, that is:

$$C^* = \arg\max_j(\theta_j^{(d_{new})}), j \in \{\text{corrective, adaptive, perfective}\} \quad (8)$$

As mentioned in Section 1, a few of the change messages may indicate multiple purposes. Multiple purposes indicate a software change has multiple categories. For example, a software change is conducted not only to fix a bug but also to clean up. Then we should label it as "corrective" and "perfective". Consider the existence of multiple categories, in formula (8), we allow C* be a set of categories when there are more than one topic entries that reach the same maximum in the distribution of $\theta_j^{(d_{new})}$. Moreover, if the distribution of $\theta_j^{(d_{new})}$ is a zero vector, in other words, the content of this message cannot be classified by this method, the change message would be classified as "Not Sure".

In summary, the main differences between DPLSA and PLSA are as follows: First, DPLSA is a supervised method, the training samples in DPLSA are labeled. While PLSA is an unsupervised method. Second, DPLSA initializes the word-topic distribution $\phi_w^{(j)}$ as Formulas (6) and (7) shows. As a result, there is a one-to-one correspondence between topics and categories. While PLSA uses a random initialization method. Third, the initialization method in DPLSA determines that the number of topics is equal to the number of categories. While in PLSA, there is no agreed-upon standard method for choosing the value for the number of topics.

The main differences between DPLSA and the sLDA method in Fu et al. (2015) are as follows: First, in the training phase, half of the training samples were labeled as signifier documents in sLDA, and it initialized the word-topic distribution $\phi_w^{(j)}$ in a random way. While in DPLSA, all the training messages are labeled and we improved the initialization step as Formulas (6) and (7) by using the labels. Second, we classify a new message by directly finding the maximum of the topic-document distribution, since we treat the topics as categories. The topic-document distribution in their work did not possess a one-to-one correspondence between topics and categories. Thus, they classified a new message by measuring the distance between topic-document distributions of the new change and the category signifier change.

4. Experiment

4.1. Data sets

Five open source projects in Table 2 were chosen to build, test, and verify this method. The following criteria were used to select the projects:

- ***Public accessible.*** The candidate open source projects were mature projects and had public accessible source control repositories.
- ***Number of commits and developers.*** For this work, only projects with near 30,000 commits were considered. In order to validate if the topic modeling understands natural languages and the expression diversity of different projects or developers, only projects with at least 40 developers were considered.
- ***Repository type.*** The preprocessing step in this work was conducted by CvsAnalY tool. Only projects with CVS or subversion repositories were supported.
- ***Previous support.*** Most of the candidate projects were used by previous researchers. For example, Wireshark, Boost, Firebird and Python listed in Table 2 were also analyzed by Mauczka et al. (2012).

Table 3
Training data set and test data set.

Projects	Labeled set Corrective	Adaptive	Perfective	Test set
Bugzilla	100	100	100	7661
Wireshark	100	100	100	32,916
Boost	100	100	100	25,002
Firebird	100	100	100	7377
Python	100	100	100	34,766

In our experiment, we consider the distinct messages whereas messages with less than five words are not used in our classification so we filter them out from the experiment data. The details of the training set and the test set are in Table 3. In Firebird, the test set appears a little exceptional compared with the original changes in Table 2. The percentage of short messages is surprisingly high. After checking the data, we find that there are 31,698 changes with the same message of "increment build number". The reason why there are too many changes with the same message is that the message was not carried out by humans and the messages do not actually include any source code modifications. For example commits generated by the "cvs2svn" repository-converter or commits that just "tag" a version. In the work Mauczka et al. (2012), if words like "cvs2svn" occurred, these changes were classified in the "Blacklist" type. However, our approach stressed a probability correlation. We referred to Swanson's original definition of maintenance tasks without the "Blacklist" category.

To facilitate cross-project analysis, 1500 messages (100 per category per project) in the training set were manually examined and labeled as corrective, adaptive and perfective according to Swanson's classification definition by professional developers. The details of the labeled set and test set are as Table 3 shows. Since the objective of the training set is to provide information about which words are informative for each category, the single-category messages are informative enough to provide such information. Thus, it is rational that the training set in this work only includes the single-category messages.

4.2. Exploring word-topic distributions (RQ1)

A probability relationship between relevant words and categories automatically generated by learning from labeled messages can overcome the ambiguity coming from manually subjective weights in the previous method. To explore the probability relationship learned from the labeled training set in Table 3, Fig. 4 shows the word-topic distributions estimated by DPLSA. Since DPLSA possesses a one-to-one correspondence between topics and categories, we use the corrective, adaptive and perfective

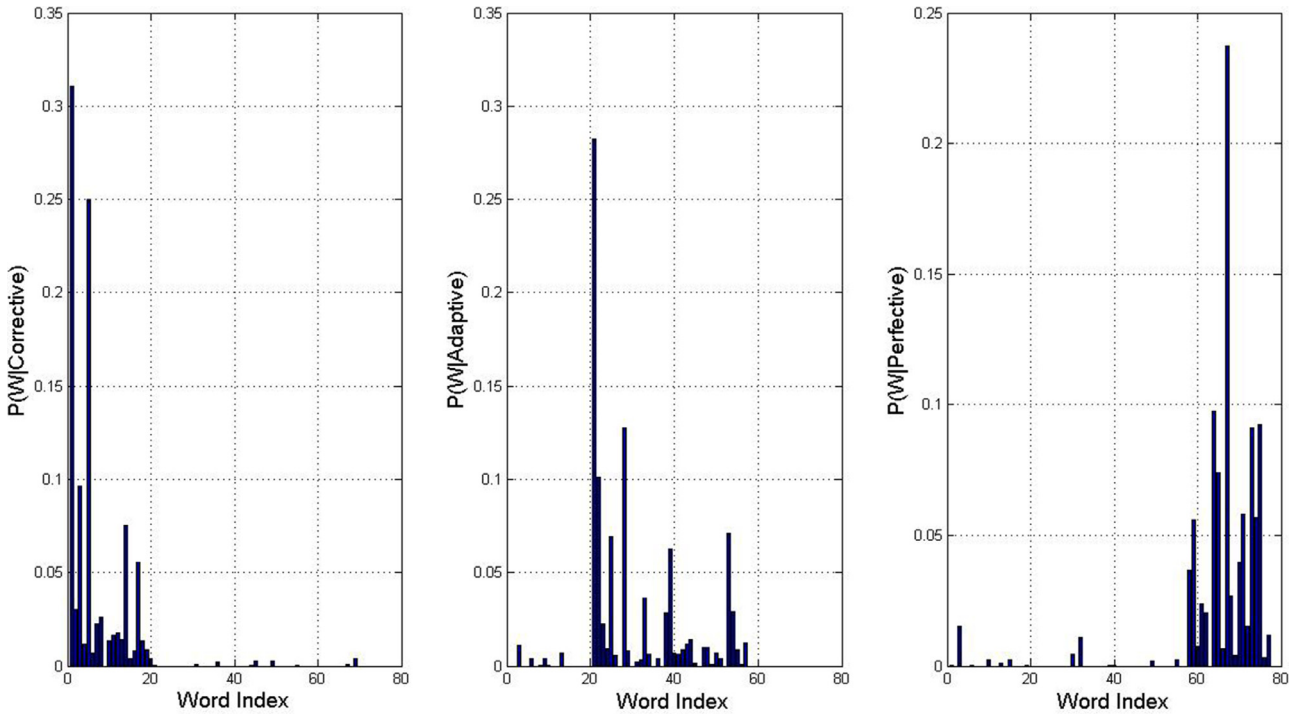

Fig. 4. Word distributions over three topics $P(w|z)$ in DPLSA, which are corrective, adaptive and perfective, respectively.

Table 4
A list of top 10 topic words and the probability generated by DPLSA.

Corrective	Adaptive	Perfective
bug(0.3110)	add(0.2827)	remove(0.2374)
fix(0.2501)	change(0.1277)	move(0.0977)
error(0.0966)	new(0.1012)	include(0.0924)
problem(0.0754)	version(0.0713)	header(0.0913)
fail(0.0556)	function(0.0693)	template(0.0741)
cause(0.0302)	patch(0.0629)	variable(0.0580)
null(0.0262)	default(0.0365)	rename(0.0569)
miss(0.0222)	install(0.0292)	cleanup(0.0558)
correct(0.0175)	old(0.0283)	unused(0.0397)
bad(0.0167)	create(0.0228)	clean(0.0365)

(the sequence is the same as the initialization step) to represent the three topics. It is seen that the estimated word distributions over the vocabulary V in Fig. 4 turns out to be a sparse distribution, i.e., three groups of informative words connect to different topics with a single dominate probability. And the sparse word-topic distributions $\hat{\phi}_w^{(j)}$ contain many entries with near zero values effectively reflecting which words are informative for each topic j.

To explore the informative words for each topic, Table 4 lists the Top 10 topic words for each topic in descending order by their relevant probability. The probability value determines which word is more relevant for the topic which is different from the previous subjective weighted value. In Table 4, the words within different topics are very distinct from each other. It is obvious that the words within a topic are closely related and those words crossing topics are significantly apart from each other. Hence, we can see that the 1st topic relates to the corrective category, the 2nd topic relates to the adaptive category, while the 3rd topic relates to the perfective category, such that a one-to-one correspondence between topics and categories is created.

In conclusion, the word-topic distributions generated by DPLSA are sparse and discriminative. This characterizes a probability relationship between relevant words and categories, and overcomes the ambiguity caused by the manually assigning method. As a result, a one-to-one correspondence between topics and categories is created. It possesses the decision-making power in classification.

4.3. Exploring topic-document distributions (RQ2)

To illustrate the correspondence between discovered topics and software change categories in the test samples, Fig. 5 displays 400 single-category messages per category randomly selected from Bugzilla by using DPLSA. The horizontal axis denotes three topics that correspond to three categories and the vertical axis denotes the topic-document distributions. Each line presents a test sample. Fig. 6 displays 330 multi-category messages which is comprised of 300 two-category messages (Fig. 6a–c) and 30 three-category messages (Fig. 6d) randomly selected from Firebird. For single-category messages, it is seen in Fig. 5 that there is only one dominate topic in the topic-document distributions. For multi-category messages, it is seen in Fig. 6 that there are more than one dominate topic in the topic-document distributions. A topic-document distribution serves as an alternative discriminative semantic representation of a change message whose largest entry directly corresponds to the category label. The results show that the semantic representation of change messages is sparse and discriminative. It directly corresponded to software changes with both single-category and multi-category.

In conclusion, there is a one-to-one correspondence between the discovered topics and change categories. The maximum topic entry solely determines the category in a single-category change. If multiple topic entries reach the same maximum, this indicates it is a multi-category change.

4.4. Exploring the category distributions (RQ3)

To illustrate the distribution of software changes, the classification results of our approach are as Table 5 shows. The fifth column presents the multi-category changes, including four category combinations. Table 6 lists the four detail category combinations of the

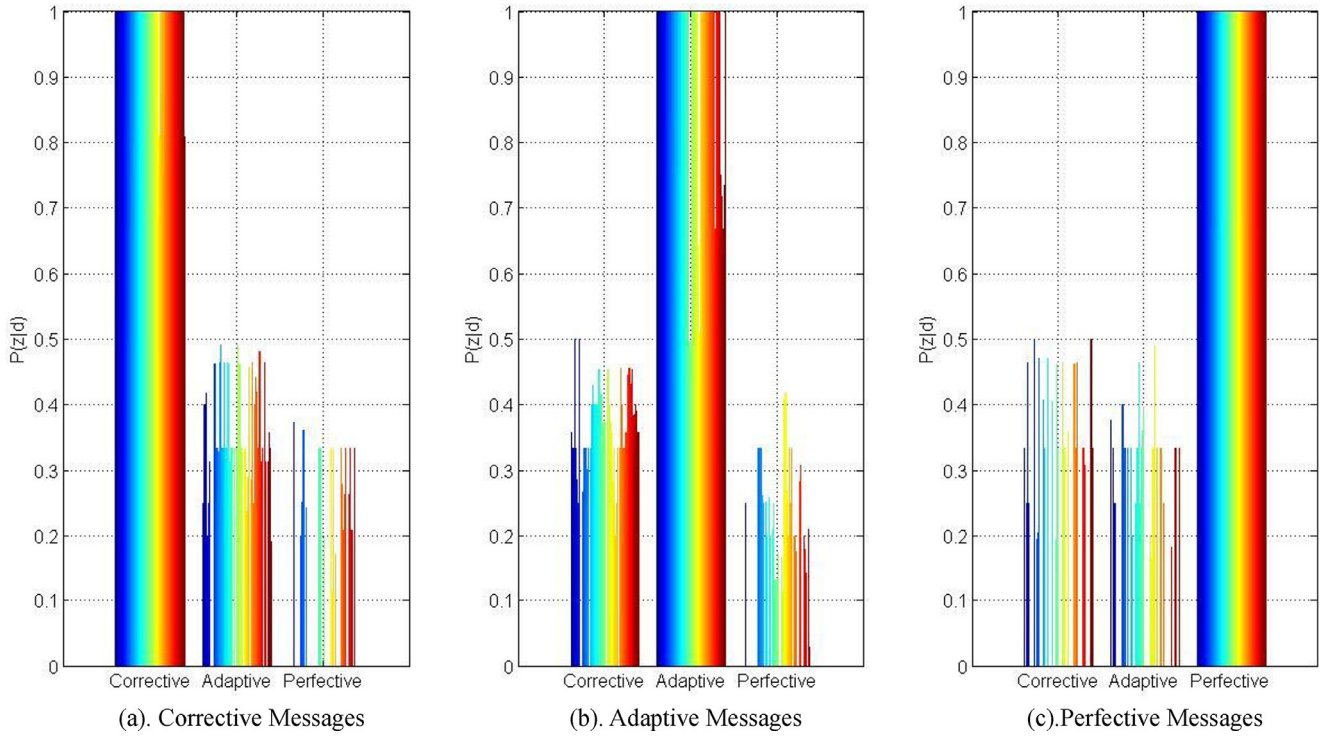

Fig. 5. The topic-document distributions of 400 single-category messages per category randomly selected from Bugzilla by using DPLSA.

multi-category. Table 7 shows the final proportion of three categories in five projects. Take the proportion of corrective category for the example, its proportion is computed by:

$$Proportion_{corrective} = \frac{Single_{corrective} + \frac{1}{2}(\text{corr\&adap} + \text{corr\&perf}) + \frac{1}{3}\text{corr\&perf\&perf}}{All\,Changes - Not\,Sure\,Changes} \quad (9)$$

In Formula (9), $Single_{corrective}$ represents the number of changes in the second column in Table 5, 'corr&adap' represents the number of changes in the second column in Table 6 while 'corr&adap&perf' represents the number of changes in the fifth column in Table 6. We make a few observations.

- In five projects, the average percentage of multi-category messages is near 9%. It indicates that the multi-category messages are an existed common phenomenon which was ignored in most previous software change classifications.
- In Bugzilla and Firebird, the percentage of corrective changes is higher than both adaptive and perfective changes as Table 7 shows. In Boost, Wireshark and Python, the percentage of adaptive changes accounts for more than both corrective and perfective changes.
- Due to the diversity of developer's expression habits, there are still some messages which cannot be classified by this method. After checking the data, this is because of irregular writing or because the content contains nothing about the purpose related to the three categories. The number of Not Sure changes is acceptable, less than 22%. Therefore, we conclude that our approach identifies most of the categories of software changes.

The distribution difference of three categories can be seen from Tables 5 and 7. We believe that Table 7 provides a more accurate distribution by classifying both single-category and multi-category changes after the validation in Section 5.

5. Validation

To evaluate our classification results, we did a survey with a small number of professional software developers which were accessible to us and we could easily interview them to explain their replies when necessary. Five developers were surveyed by our questionnaires, each with 80 messages which were selected randomly including both single-category and multi-category messages from each project. Too many surveyed messages for a participant is a big burden to finish an accurate survey (Hassan, 2008). We determined the test set size as a trade-off value in an empirical way which is similar to the works in Table 8. At the time of our study, these developers worked in different software domains, such as security, databases, cloud computing, and software project management. All the participants had used version control systems for most of their software career.

5.1. Conducting the validation

We took precision, recall and *F*-measure (Sebastiani, 2002) to evaluate the classification performance. The *F*-measure value considers both the precision (p) and the recall (r) of the classification to compute the value, where the p is the number of correct results divided by the number of all returned results and the r is the number of correct results divided by the number of results which should have been returned. The precision and recall are calculated by:

$$\text{Precision}(H,V) = \frac{1}{|V|}\sum_{i=1}^{|V|}\frac{|Y_i \cap Z_i|}{|Z_i|} \quad (10)$$

$$\text{Recall}(H,V) = \frac{1}{|V|}\sum_{i=1}^{|V|}\frac{|Y_i \cap Z_i|}{|Y_i|} \quad (11)$$

where V is our validation set, H indicates our classifier, $Z_i = H(x_i)$ indicates the prediction label set of message x_i, Y_i indicates the label set by participants. $Y_i \subseteq L$,

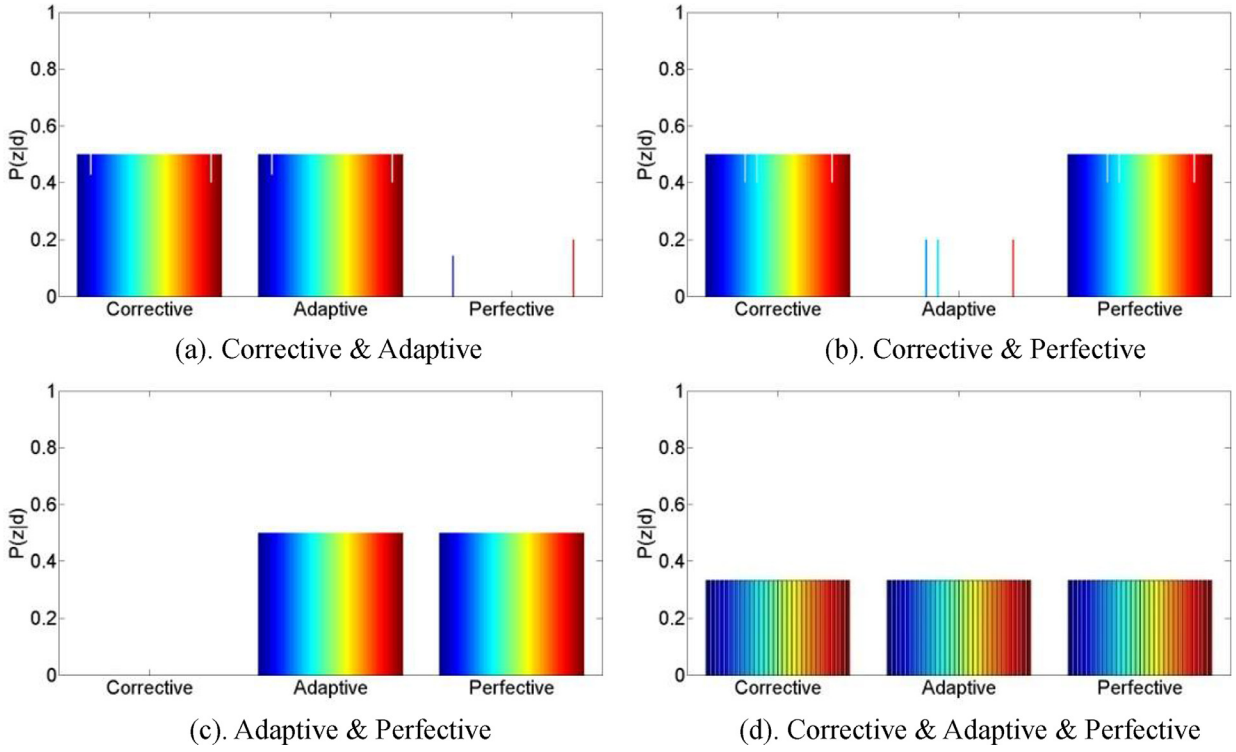

Fig. 6. The topic-document distributions of 330 multi-category messages including 300 two-categories messages (100 in (a), 100 in (b), 100 in (c)) and 30 three-categories messages (d) in Firebird by using DPLSA.

Table 5
Automatic classification results.

Projects	Corrective	Adaptive	Perfective	Multi-category	Not Sure	All
Bugzilla	5718(74.64%)	717(9.36%)	433(5.62%)	555(7.24%)	238(3.11%)	7661
Wireshark	8107(24.63%)	9181(27.89%)	4641(14.10%)	2924(8.88%)	8063(24.50%)	32,916
Boost	6410(25.64%)	7710(30.84%)	2520(10.08%)	2793(11.17%)	5569(22.27%)	25,002
Firebird	2682(36.36%)	1560(21.15%)	791(10.72%)	643(8.72%)	1701(23.06%)	7377
Python	9019(25.94%)	12044(34.64%)	3257(9.37%)	2880(8.28%)	7566(21.76%)	34,766

Table 6
Detail categories of the multi-category in Table 5.

Projects	Corr&Adap	Corr&Perf	Adap&Perf	Corr&Adap&Perf	All multi-category
Bugzilla	501(90.27%)	15(2.70%)	31(5.59%)	8(1.44%)	555
Wireshark	1178(40.29%)	552(18.88%)	968(33.11%)	226(7.73%)	2924
Boost	1181(42.28%)	481(17.22%)	881(31.54%)	250(8.95%)	2793
Firebird	289(44.95%)	126(19.60%)	198(30.79%)	30(4.67%)	643
Python	1358(47.15%)	326(11.32%)	1045(36.28%)	151(5.24%)	2880

$L = \{'\text{corrective}', '\text{adaptive}', '\text{perfective}'\}$. In the manual validation, a message can be assigned to both single-category and multi-category by the participant. The *F*-measure is interpreted as a weighted average of the precision and recall and the higher score indicates a better classification performance.

$$F\text{-}measure = \frac{2^{*}(\text{Precision}^{*}\text{Recall})}{\text{Precision} + \text{Recall}} \tag{12}$$

The performance of this method is as Table 9 shows. The average *F*-measure of five projects is 0.7618. The robust performance across five projects indicates that our method is valid for cross-project classifying of software changes.

5.2. Comparing with published methods

To make our results comparable with previously published methods, we choose the same preprocessed datasets in this work to compare with four methods. The first one is a lightweight method presented by Hattori and Lanza (2008) which is a typical keywords retrieving based method. It classified each software change according to the first keywords found in the message. The words remaining in the message are non-stop words and it retains the original order. If a software change is conducted for multiple

Table 7
The final proportion of three categories in five projects.

Projects	Corrective	Adaptive	Perfective
Bugzilla	80.54%	13.28%	6.18%
Wireshark	36.40%	41.56%	22.03%
Boost	37.69%	45.41%	16.90%
Firebird	51.08%	31.96%	16.97%
Python	36.44%	48.88%	14.68%

Table 8
The size of evaluation size in similar papers.

Study Study	Number of participants	Number of changes per participant
(Mockus and Votta, 2000)	5	30
(Hassan, 2008)	6	18
(Mauczka et al., 2012)	5	21
(Fu et al., 2015)	5	60
Our work	5	80

Table 9
Validation results.

Projects	Precision	Recall	*F*-measure
Bugzilla	0.8812	0.7479	0.8091
Wireshark	0.6813	0.7813	0.7278
Boost	0.7813	0.7125	0.7453
Firebird	0.8042	0.8000	0.8021
Python	0.7313	0.7188	0.7249

purposes, e.g., a developer can do some refactoring work while fixing a bug, the method classifies the change according to the first keyword found. In this work, we call their method the "First key" method. Specially, the keywords in the First key method must have a category label. When implementing the method, we adopt the same keywords in this work and keep the category label provided by Mauczka et al. (2012). The second one is the work of Fu et al. (2015) which is based on semi-supervised LDA (sLDA). We implement it as the description in Fu et al. (2015), i.e., we keep the number of topics K to be 3, the hyper-parameters α to be 50/K, β to be 0.01, and iterations to be 100. The third one is a common classification method: the Naïve Bayes method. We adopt the term frequencies as features and implement the multinomial Naïve Bayes method. The above three baselines only provide single-category classifications. The fourth one produces both single-category and multi-category classifications, namely Labeled LDA (L-LDA) (Ramage et al., 2009), which also creates a one-to-one correspondence between topics and categories. We adopt the toolbox provided by Zeng (2012) which includes the L-LDA implementation. We apply the four methods to automatically classify the same validation dataset and calculate the performance with the survey. We take the precision, recall and F-measure value to make the comparison. Fig. 7 shows the performance comparison between Naïve Bayes, "First key", sLDA, L-LDA and our method.

Table 10
Detailed comparison results.

Category	Classification *F*-measure				
	DPLSA	sLDA	First key	Naïve Bayes	L-LDA
Corrective	0.7832	0.7857	0.7052	0.6607	0.7612
Adaptive	0.7297	0.7841	0.7614	0.4886	0.7543
Perfective	0.7574	0.5761	0.5761	0.3370	0.7137
Corr&&Adap	0.7738	0.6512	0.5891	0.5581	0.6357
Corr&&Perf	0.6836	0.6250	0.6042	0.4375	0.6179
Adap&&Perf	0.7680	0.5000	0.4583	0.3542	0.6340
Corr&&Adap&&Perf	0.5000	0.5000	0.5000	0.5000	0.5000

The following conclusions are drawn from the validation results in Fig. 7. First, considering the average *F*-measure value across five projects, DPLSA improves the sLDA, First key, Naïve Bayes, and L-LDA methods by 8.12%, 15.29%, 50.63% and 4.69%, respectively. Specially, we believe that the high recall of DPLSA results from the power of classifying multi-category changes. It also answers the RQ3. Second, topic model based methods beat the First key and Naïve Bayes methods in average. We believe that it is the result of discovering hidden semantic relations between words. Third, the robust average *F*-measure value (0.7618) of the five projects also proves our topic model based approach is valid for cross-project analysis in software change classification.

Other observations. Considering the results across five projects, we list the classification *F*-measure values of each method over each category in Table 10 and the number of instances in each case in Table 11. There are some notations which need explanation in Table 11: R indicates "right" which means the method result is the same as the survey result. W indicates "wrong" which means the method result has no intersection with the survey

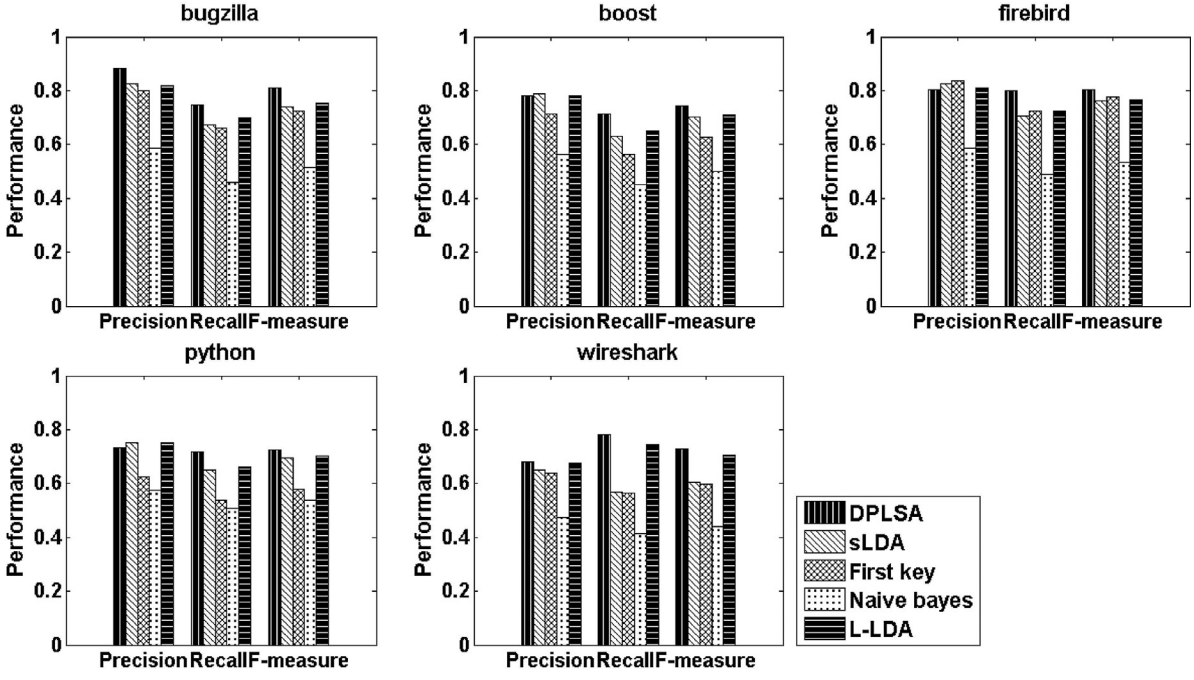

Fig. 7. Precision, Recall, *F*-measure comparison between Naïve Bayes, First key, sLDA, L-LDA and DPLSA.

Table 11
Number of instances in each category of different methods.

Category	Instances	DPLSA				sLDA			First key			Naïve Bayes			L-LDA				Errors in
	in survey	R	W	P	M	R	W	P	R	W	P	R	W	P	R	W	P	M	all methods
Corrective	112	80	21	11	11	88	24	0	79	33	0	74	38	0	79	24	9	9	2
Adaptive	88	55	20	13	13	69	19	0	67	21	0	43	45	0	63	20	5	5	0
Perfective	92	60	18	14	15	53	39	0	53	39	0	31	61	0	56	22	14	15	0
Corr&&Adap	43	15	2	26	15	0	1	42	0	5	38	0	7	36	0	2	41	0	0
Corr&&Perf	32	3	1	28	7	0	2	30	0	3	29	0	11	21	1	3	28	1	0
Adap&&Perf	32	8	1	23	16	0	8	24	0	10	22	0	15	17	0	4	28	5	0
Corr&&Adap&&Perf	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0

result. P indicates "partial right" which means the method result is not the same as the survey result, but it has intersection(s) with the survey result. M indicates "Multi-category" which means the number of multi-category instances in the method result. Obviously, the column M does not exist in a single-category method. The following observations are seen from Tables 10 and 11: (1) In DPLSA, sLDA, Naïve Bayes and L-LDA, the corrective is easiest to classify among all the cases and the corr&&adap is easier to classify among the multi-category cases. The First key method is more feasible to classify the adaptive among all the categories. (2) Considering the single-category case, other methods may be superior to DPLSA in some particular cases. For example, sLDA performs better than DPLSA in the corrective and the adaptive cases, whereas First key and L-LDA performs better that DPLSA in the adaptive case. However, it is seen that DPLSA is more feasible to classify multi-category commits, including the corr&&adap, corr&&perf and adap&&perf. (3) There are 108 multi-category instances in the survey, 77 multi-category instances in DPLSA and 35 multi-category instances in L-LDA. On the multi-category cases of the survey, there are 4 wrong classifications in DPLSA and 9 wrong classifications in L-LDA. (4) The "Errors in all methods" indicates the instances which are classified wrong in all five methods. There are only 2 commits which all the methods have trouble with. This reflects that different methods are feasible to handle different commits.

6. Limitation and threats to validity

Size of evaluation set. We determined the test set size as a trade-off value in an empirical way which is similar to the works in Table 8. Using a more systematic approach for choosing the size of the evaluation dataset is better. However, the size of surveyed messages needed cannot be controlled in a systematic approach. Too many surveyed messages are a big burden for participants to finish an accurate survey. Also, too few surveyed messages may not contain any multi-category messages. It may impact the diversity and suffer from bias in the test set. Therefore, we tried to mitigate this problem by setting a trade-off test size value.

Over-fitting problem in PLSA. PLSA is reported not very robust when applied to unseen documents, the problem of over-fitting may limit the performance (Lu et al., 2011). In this work, the vocabulary is constructed by a set of semantically salient words. And the original random initialization method in PLSA is replaced by our discriminative initialization method. All the unseen documents are assigned a probability distribution by sharing the same salient vocabulary and estimated model, which can alleviate the un-robust problem in unseen documents.

Baseline approaches. Several keywords retrieving based methods have been presented in software change classification, such as the works of Mockus and Votta (2000), Hassan (2008), and Mauczka et al. (2012). A comparison with all these methods should be used for completeness. However, the difference of the change category definitions or the inaccessibility of the tools relied on prevent them from being implemented for comparison. Therefore, we implemented a typical keywords retrieving based method First key (Hattori and Lanza, 2008) as a baseline which can represent the keywords retrieving method thinking in this line of research.

Unclassified change messages and vocabulary. During our experiment, we found that there were several change messages which cannot be categorized into any three categories. After checking the contents, we found a phenomenon that most of these messages were written irregularly or there was little information included. It is correlated with the cross-project vocabulary. If we yield a vocabulary by using an automatic way, more words will be added in the vocabulary which can classify more messages. However, many unsalient words (e.g. "be") are also included which may be noisy for topic modeling (Chien and Chang, 2014). Therefore, we captured a typical and salient words set in the vocabulary from a cross-project dictionary presented by Mauczka et al. (2012).

7. Conclusions and future work

In this paper, we investigated an artifact of software development, namely, the change messages attached to every change committed to a version control system. It presented a discriminative topic model technique supporting multi-category classification and cross-project. The results of a set of controlled experiments carried out to validate whether it can evaluate the probability relationship between relevant words and categories and provide a more accurate distribution of software changes. In summary, the conclusions are drawn as follows: first, we provide a simple initialization approach for topic models, which sufficiently considers the subspace structure of the training datasets and assigns the word-document distributions by using the training samples from the same category and the semantically salient words. It overcomes difficulties in determining an appropriate number of topics by making the topics in DPLSA correspondence with category labels of change messages. Second, we define a one-to-one correspondence between topics and categories. The estimated topic-document distributions of software change messages are discriminative and sparse, which allows it to perform single-category and multi-category change classification. Third, the experiments were conducted on 5 open source projects. The achieved results prove that the DPLSA approach provides an average *F*-measure of 0.7618 which improves the sLDA, First key, Naïve Bayes, and L-LDA methods by 8.12%, 15.29%, 50.63% and 4.69%, respectively. Besides, the different projects share the same vocabulary and estimated model. It also proves that the DPLSA is well applicable to cross-project analysis without the need for re-learning.

In the future, we plan to further enhance the exploration of the results of our work. For example, why are some commits more difficult to classify and what common features do these commits possess? We plan to perform our work on more projects and conduct a field study by interviewing developers in both industrial and open source projects. This consists of including more projects and

participants, designing surveys, obtaining feedback statistics and interviewing participants.

Acknowledgments

The work described in this paper was partially supported by the National Natural Science Foundation of China (grant nos. 91118005, 61173131), Changjiang Scholars and Innovative Research Team in University (grant no. IRT1196), Chongqing Graduate Student Research Innovation Project (grant no. CYS14008), and the Fundamental Research Funds for the Central Universities (grant nos. CDJZR12098801 and CDJZR11095501).

References

Ahsan, S.N., Ferzund, J., Wotawa, F., 2009. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In: Proceedings of the 4th International Conference on Software Engineering Advances. ICSEA 2009, pp. 216–221.

Alali, A., Kagdi, H., Maletic, J.I., 2008. What's a typical commit? A characterization of open source software repositories. In: Proceedings of the 16th IEEE International Conference on Program Comprehension. ICPC 2008, pp. 182–191.

Asuncion, H.U., Asuncion, A.U., Taylor, R.N., 2010. Software traceability with topic modeling. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. ICSE 2010, pp. 95–104.

Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent Dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022.

Chen, T.H., Thomas, S.W., Nagappan, M., Hassan, A.E., 2012. Explaining software defects using topic models. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories. MSR 2012, pp. 189–198.

Chien, J.-T., Chang, Y.-L., 2014. Bayesian sparse topic model. J. Signal Process. Syst. 74, 375–389.

Commission, I.O.F.S.I.E., 2001. Software engineering–Product quality–Part 1: Quality model. ISO/IEC 9126, 2001.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., 2002. GATE: an architecture for development of robust HLT applications. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. ACL 2002, pp. 168–175.

Dempster, A.P., Laird, N.M., Rubin, D.B., 1977. Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc., Ser. B (Methodol.) 39, 1–38.

Fu, Y., Yan, M., Zhang, X., Xu, L., Yang, D., Kymer, J.D., 2015. Automated classification of software change messages by semi-supervised latent Dirichlet allocation. Inf. Software Technol. 57, 369–377.

Gethers, M., Poshyvanyk, D., 2010. Using relational topic models to capture coupling among classes in object-oriented software systems. In: Proceedings of the IEEE International Conference on Software Maintenance. ICSM 2010, pp. 1–10.

Gethers, M., Savage, T., Penta, M.D., Oliveto, R., Poshyvanyk, D., Lucia, A.D., 2011. CodeTopics: which topic am I coding now? In: Proceedings of the ACM/IEEE 33rd International Conference on Software Engineering. ICSE 2011, pp. 1034–1036.

Grant, S., Cordy, J.R., Skillicorn, D.B., 2012. Using topic models to support software maintenance. In: Proceedings of the 16th European Conference on Software Maintenance and Reengineering. CSMR 2012, pp. 403–408.

Grant, S., Cordy, J.R., Skillicorn, D.B., 2013. Using heuristics to estimate an appropriate number of latent topics in source code analysis. Sci. Comput. Program. 78, 1663–1678.

Hassan, A.E., 2008. Automated classification of change messages in open source projects. In: Proceedings of the 2008 ACM symposium on Applied computing, SAC 2008, pp. 837–841.

Hattori, L.P., Lanza, M., 2008. On the nature of commits. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering. ASE 2008, pp. 63–71.

Hindle, A., Ernst, N.A., Godfrey, M.W., Mylopoulos, J., 2011. Automated topic naming to support cross-project analysis of software maintenance activities. In: Proceedings of the 8th Working Conference on Mining Software Repositories. MSR 2011, pp. 163–172.

Hindle, A., German, D.M., Godfrey, M.W., Holt, R.C., 2009. Automatic classication of large changes into maintenance categories. In: Proceedings of the 17th IEEE International Conference on Program Comprehension. ICPC 2009, pp. 30–39.

Hindle, A., German, D.M., Holt, R., 2008. What do large commits tell us?: a taxonomical study of large commits. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. ACM, Leipzig, Germany, pp. 99–108.

Hindle, A., Godfrey, M.W., Holt, R.C., 2009b. What's hot and what's not: windowed developer topic analysis. In: Proceedings of the IEEE International Conference on Software Maintenance, ICSM 2009, pp. 339–348.

Hofmann, T., 2001. Unsupervised learning by probabilistic latent semantic analysis. Mach. Learn. 42, 177–196.

Kim, S., Whitehead, E.J., Zhang, Y., 2008. Classifying software changes: clean or buggy? IEEE Trans. Software Eng. 34, 181–196.

Lafferty, J.D., Blei, D.M., 2006. Correlated topic models. Adv. Neural Inf. Process. Syst. 18, 147–154.

Lu, Y., Mei, Q., Zhai, C., 2011. Investigating task performance of probabilistic topic models: an empirical study of PLSA and LDA. Inf. Retrieval 14, 178–203.

Mauczka, A., Brosch, F., Schanes, C., Grechenig, T., 2015. Dataset of developer-labeled commit messages. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), pp. 490–493.

Mauczka, A., Huber, M., Schanes, C., Schramm, W., Bernhart, M., Grechenig, T., 2012. Tracing your maintenance work–a cross-project validation of an automated classification dictionary for commit messages. In: Fundamental Approaches to Software Engineering, pp. 301–315.

Miller, G.A., 1995. WordNet: a lexical database for English. Commun. ACM 38, 39–41.

Mockus, A., Votta, L.G., 2000. Identifying reasons for software changes using historic databases. In: Proceedings of the International Conference on Software Maintenance. ICSM 2000, pp. 120–130.

Pollock, L., Vijay-Shanker, K., Hill, E., Sridhara, G., Shepherd, D., 2013. Natural language-based software analyses and tools for software maintenance. Software Engineering. Springer,, Berlin, Heidelberg, pp. 94–125.

Ramage, D., Hall, D., Nallapati, R., Manning, C.D., 2009. Labeled LDA: a supervised topic model for credit attribution in multi-labeled corpora. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, pp. 248–256.

Robles, G., Koch, S., Gonzalez-Barahona, J.M., 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In: Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems. RAMSS 2004, pp. 51–56.

Savage, T., Dit, B., Gethers, M., Poshyvanyk, D., 2010. Topic XP: exploring topics in source code using latent Dirichlet allocation. In: Proceedings of the IEEE International Conference on Software Maintenance. ICSM 2010, pp. 1–6.

Sebastiani, F., 2002. Machine learning in automated text categorization. ACM Comput. Surv. (CSUR) 34, 1–47.

Swanson, E.B., 1976. The dimensions of maintenance. In: Proceedings of the 2nd International Conference on Software Engineering. ICSE 1976, pp. 492–497.

Thomas, S.W., 2012. Mining software repositories with topic models. School of Computing, Queen's University Technical Report 2012-586.

Wright, J., Yang, A.Y., Ganesh, A., Sastry, S.S., Ma, Y., 2009. Robust face recognition via sparse representation. IEEE Trans. Pattern Anal. Mach. Intell. 31, 210–227.

Zeng, J., 2012. A topic modeling toolbox using belief propagation. J. Mach. Learn. Res. 13, 2233–2236.

Meng Yan was born in 1989 and received his B.S. in Chongqing University in 2011, and his M.S. degree in Software Engineering in 2013. He is currently a Ph.D. candidate of the School of Software Engineering, Chongqing University. His research interests include data mining of software engineering and topic modeling.

Ying Fu was born in 1991 and received her B.S. in Chongqing University in 2013. She is a M.S. candidate of the School of Software Engineering, Chongqing University now. Her research interests include data mining of software engineering and topic modeling.

Xiaohong Zhang received the Ph.D. degree in Computer Software and Theory from Chongqing University, PR China in 2006, where he also received the M.S. degree in Applied Mathematics. He is a professor in School of Software Engineering at Chongqing University. His current research interests include data mining of software engineering, topic modeling, image semantic analysis and video analysis.

Dan Yang received the Ph.D. degree from Chongqing University, PR China in 1997, where he also received the M.S. degree in 1985 and B.S. in 1982. He is a professor in School of Software Engineering at Chongqing University. His current research interests include data mining of software engineering, topic modeling, image semantic analysis and video analysis.

Ling Xu was born in 1975 and received her B.S. in Hefei University of Technology in 1998, and her M.S. degree in Software Engineering in 2004. She received her Ph.D. degree in Computer Science Technology from Chongqing University, PR China in 2009. Her research interests include data mining of software engineering, topic modeling, and image processing.

Jeffrey D. Kymer received a B.S. in Computer Science and a B.S. in Mathematics from Westfield State College in Westfield, Massachusetts, USA in 1986. He is a senior lecturer in School of Software Engineering at Chongqing University. He has worked on a range of software products from a top selling CD-Rom (the MPC Wizard) to one of the first multi-track audio boards. His current research interests include software engineering, computational linguistics, teaching, data mining, and alternate computer interfaces.