Generalized Probabilistic Language Model for Natural-Language Software Text Mining

FNU Vivek
CSC 695 MS Thesis Research, Spring'19,
Computer Science, North Carolina State University
Raleigh, NC
vvivek@ncsu.edu

ABSTRACT

Understanding software change messages that are in natural-language and unstructured text is useful in defect prediction. It is also a challenge to aptly classify them as bug-fixing commits or not. Traditionally, it has been done using a keyword-based model. Probabilistic models like Latent Dirichlet Allocation(LDA) have also been shown to be useful in mining software repository texts like this. We seek to create a novel language model that understands software change messages and can also automatically classify them. Moreover, this model should be generic to be applied to individual projects and work on small amount of data as well. This model has two advantages over existing methods: First, the number of topics is set to number of classes, so the latent topics of LDA are comprehensible. Second, the model is generic and localized to a project and does not depend on predefined keywords, so it can be applied to any type of software text expressed in natural-language. In this work we present an Latent Dirichlet Allocation(LDA) based approach to classify change messages as bug-fixing commits or not. We extract top N words from K topics. We use the probabilities of these N words and topic probabilities of K topics to train a Random Forest classifier. We verify the generality of this model by applying it on two different types of software texts: stackoverflow questions and technical debt datasets. On commit message dataset, our model has 82.2% recall averaged across projects and 60.6% F1 score. The baseline model has 76.2% recall and 26.5% F1 score. On stackoverflow dataset recall is 89% and F1 score is 79.2%. On tech debt dataset our model has 76.2% recall and 63.7% F1 score, compared to 77.33% and 73.7% for the baseline model, respectively. Our model outperforms the baseline model in terms of F1 score and has comparable recall for change message classification. Our model also performs well on stackoverflow dataset and has comparable performance on technical debt dataset. This model generalizes well on different types on natural-language software texts.

1 INTRODUCTION

Almost every software project uses version control system. Not only does version control system maintain the software, it also helps maintain history of a particular piece of code. It can be well established with a particular change, who did the change, for what purpose, at what time, what files that change corresponds to and why that change. *Why that change* is very important as this natural language message can give a wealth of information on the nature of the change introduced.

Classifying a change message as a bug-fixing commit or not requires understanding of the natural-language texts in the commit message. It is a challenge to understand natural-language software texts because they do not follow the natural convention of natural language, particularly related to grammar. Traditionally change message classification has been done using a generic keyword dictionary [16] [8]. We have also seen that probabilistic methods have shown some success too, particularly, Latent Dirichlet Allocation (LDA) [9] [5] and probabilistic Latent Semantic Analysis[11].

Our motivation is to create a language model that is generic and can be applied to any version control project. Therefore, we do not rely on dictionary based model, rather be inspired by fully automated probabilistic model. In this project, we explore automated machine learning methods to predict if a change message corresponds to a bug fix. Most of the git projects are managed by only a certain group of individuals, hence, they tend to have a characterizing way of expressing commit messages. However, as all of them are software projects, certain software engineering domain specific words are frequently used. These general keywords are more prominent to identify a bug-fixing commit, but not exhaustive. Due to this localized nature of software management, we need to explore a model that learns these words specific to a project. In a nutshell, we will explore a generalized "local" automated language model for discovering bug-fixing messages.

Mauckza et al. [15] proposed a variation of keyword based approach to automated classification change messages. This work has been based on Mockus et al. [16] natural language "modification request" words. Keyword based models have been a benchmark for change message classification. In this project, we aim to question if a generic keyword dictionary is suitable for all projects. Or if there exists some localized natural language reference of equal importance with respect to global software engineering specific keywords. This leads to our first and second research questions

- **RQ1:** Should we specify generic keywords for all software projects for change message classification?
- **RQ2:** Does a general automated model trained on specific project perform better than generic dictionary based classifiers?

We have seen that probabilistic approaches have been used earlier for change message classification. However, it is difficult to explain the latent features that are extracted like latent topics. Comprehensibility of the topics is a major concern. We aim to build features for the classifier that has comprehensible features.

- **RQ3:** How many latent topics to keep in LDA? Are these topics comprehensible?

Our motivation is to build a generic model that can do change message classification specific to a project. However, we also want to keep this model generic enough that it can be used to do any

natural-language software text mining. So, we aim to verify the model performance on stackoverflow questions and technical debt datasets.

- **RQ4:** How does the model generalize to different natural-language software texts?

Although, we focus mostly on change message classification but our core aim is to build a probabilistic language model that can be used for any type of natural-language software text mining. We establish that by answering this research question.

2 BACKGROUND AND RELATED WORK

The research field of change classification has been evolving over many decades. The first classification system of software changes was put forward by Swanson in 1976 [17]. Swanson defined that a change can be classified in one of the following 3 categories:

- Corrective: Changes that are fix for prior failures.
- Adaptive: Changes in environment.
- Perfective: Changes to improve efficiency.

Based on this foundation, corrective software changes have been studied for the purpose of software maintenance [16] and defect prediction[13]. Mockus and Votta [16] established that to understand the motive of introducing a change in the software can be estimated from the associated message. Natural language processing comes into play when understanding these change messages.

Kim et al [13] mentions that a *bug-introducing* change can be identified through the analysis of a bug-fix. These bug fix changes are in turn identified by mining change messages. He proposes that it can be done through two ways:

- Keyword search model aimed at searching for well identified keywords such as "Fixed" or "Bug". This is a reasonable assumption and is also seen from Hindle et al. to be successful [9]. This sort of injection of domain knowledge into the classification of change message does prove helpful. Therefore, we use a similar model as our baseline.
- Searching for references for bug reports such as their IDs. It is very difficult to track as Issue tracking system is often used for Feature tracking too.It seems that injecting more domain knowledge may help us identify bug-fixing commits more precisely. However, using issue tracking system IDs does not resonate well with our main idea.

These methods of identifying bug-fixing commits are costly. It is not easy to get messages labeled for each project individually. A generic keyword based model does well for identifying the bug-fix commits, but it is not local to the project. Tracking issue IDs is local to a project but is not a scalable model. We learn from this limitation and try to come up with an algorithms that is both scalable, and can identify both general and local keywords corresponding to a bug-fix commit message.

Initial models were built on keyword basis [16] [8] [7] and Naive Bayes classifiers [3]. In a way, it can be said that these classifiers were either look up or classifiers needed help from experts for understanding classification. Antonio et al [3] proposed a method based on decision trees, naive bayes and logistic regression models that does change message classification. However, Support Vector Machine (SVM) remains a popular choice for this classification [10] [13]. Hindle et al. [9] employed the Weka Machine learning version of SVM. Random Forest have also been shown to be used successfully for classification task in software text mining [19]. We find Random Forest to be most robust to any type of data because of its ensemble nature. Hence, we use Random Forest in our experiments.

Table 1: Commits dataset : class labels actual counts

Project	label_code: 0	label_code: 1	% positive
abinit	4024	572	12.44
libmesh	6462	651	9.15
mdanalysis	2802	379	11.91

Mauczka et al. [15] proposed a variation of keyword based approach to automated classification change messages. This model can be seen as a "Generalized evolving dictionary". The dictionary evolves as the more and more commits messages are studied. This is very specific to a project and has good performance, however the model is still rule based and stringent. Hindle et al. [10] discusses Latent Dirichlet Allocation (LDA). Use of probabilistic model helps the model discover more features and properties of natural-language text. A keyword based model is too strict, even when the keyword is developed specific to the project as modeled by Mauczka et al. [15]. Hindle et al proposes to discover change message classification topics over a window of time. LDA based method is used to analyze this unstructured data. This is an attempt to provide comprehension to latent topics of LDA. LDA based text mining proves to be useful in increasing comprehension of the model as opposed to only TF (term frequency) and TFIDF (term frequency inverse document frequency) models [2].

Another tool that has been studied and shown to have success is pLSA (probabilistic Latent Sematic Analysis) [11]. However, as we are not trying to find semantic information, rather looking for binary classification of change message, we choose LDA for our study. Studies have shown the success of LDA models for defect prediction [18] [6] [2] [5]. Success of these works have motivated us to explore topic modeling, specifically LDA, for our studies. Our model is therefore based on LDA and Random Forest taking inspiration from these works.

3 RESEARCH METHODOLOGY

We process our raw textual data. After pre-processing of the texts to convert them to appropriate vectors to feed to LDA. We extract top N (=100) words based on words in each topic and compute K(=2) probabilities of topics for each commit message. These N + K features are fed to a Random Forest classifier for binary classification. As we use both LDA and RF in sequence, we call our model LDA-RF.

3.1 Data

The data under study here is collected through mining of open source software projects from github.com. Three projects were selected for this study. These are popular github projects in the field of computation. They have a large number of commits, as will be explained later. The commit messages were labelled during a mechanical turks study. The mechanical turks were graduate students from the Computer Science department at North Carolina

State University, Raleigh, NC, USA. The details of the project is shown in Table 1.

Each of the commit messages has been labelled as 0 or 1. Here, label 1 corresponds to a message being tagged as bug-fixing commit or not. The value counts of each project is as shown in Figure-**??**. We see that each project has a large number of non-buggy commits compared to bug-fix commits. This is expected. Otherwise, a software project is poorly managed if there are a lot of bug-fix commits. Also, we can see from Table-1, that on an average, we have buggy commits to be at least 10 percent of the total commits.

3.2 Latent Dirichlet Allocation

Specific to text based corpus, we can say that Latent Dirichlet Allocation (LDA) is an unsupervised clustering algorithm aimed at clustering the words in the corpus to specific *topics*. Since, it is probabilistic in nature, it means that certain words can reside in different topics. According to Blei et al. [4] the joint distribution of LDA can be described as

$$P(w, z|\alpha, \beta) = P(w|z, \beta) * P(z|\alpha)$$

Here, z is the topic and w is the word. And, α and β are model parameters. P(z|α) is the probability of topic z appearing in the document d. P(w|z, β) describes the probability of word w appearing in a particular topic z. We will make use of both P(z|α) and P(w|z, β). We use python scikit-learn implementation of LDA. There are certain hyper parameters to that model. *Number of topics* is the first and foremost hyper parameter. We need to decide how many topics prior to the training process. In our work, we classify change messages in two distinct values 0 or 1 indicating non-buggy or buggy commit message. Hence, our natural option for selecting *'k'* i.e. number of topics is 2. This ensures that the topic could correspond to these labels. This decision greatly enhances our ability to interpret these topics. This is great addition to comprehensibility of our model. This decision is inspired by a similar work of Ying et al on change message classification where they choose number of topics corresponding to number of classes [5]. Another hyper parameter is total number of words to consider before sending to LDA for training. This is called *maximum number of features*, i.e. the number of dimensions. This sets the maximum number of words to be fed to LDA. We see that lower number of maximum features is much better [14]. For this experiment we default to the value of 1000 for this purpose. Another hyper parameter that we change in our experiment is *number of iterations* for LDA. We set it to 20. Essentially, LDA is a generative probabilistic model. LDA has two matrices. First, describes the probability of selecting a particular word when sampling a topic. Second, describes the probability of selecting a topic when sampling a particular document. Now to build word versus topic matrix, a sample is drawn from dirichlet distribution for each topic using β as input. For second a composite is defined as a document and a sample is drawn from dirichlet distribution using α as input. Then the actual composites are defined basis these two matrix for each document. LDA does expectation maximization in each iteration. We set *maximum number of features* as 10 for our study. The LDA iteration method is Gibbs sampling. The other hyper parameters namely β, and that we choose are taken from literature study [5] or set to default.

LDA is a common technique in text mining used for dimensionality reduction [1]. TF and TF-IDF are common feature generation methods applied to text mining. The order of number of features is 1000s or more. However, with LDA we get very small number of dimensions. As we keep only 2 topics, this is a reduced dimension from 25 as used in vectorization. The output of LDA is very comprehensible [1] [2]. We agree with their work and validate LDA and comprehensibility it offers in this paper.

3.3 Text Pre-processing

Even before we feed data to LDA, we need to process the natural language expressed in change messages. Some examples of change messages are as follows:

- Quickfix: paral_kgb=1 shoulkd be ignored for DFPT (was OK in previous versions)
- Fix bug that was preventing the kxc to be stored.
- multibinit: add initialization for the fit and add restartxf -3,0.0.
- Fixup receive_packed_range call.

These messages are expressed in a natural language, not necessarily following the English grammar well. Most of the the times it contains only few words. We want to vectorize these words to feed into our algorithm. First of all we extract important words from this message. Using the python NLTK module, the words are first tokenized using WordPunct Tokenizer. Then all the tokens which are either numerical values of symbols or punctuations are removed. To remove redundant tokens, all words that are English stopwords like "is", "are", "has", etc are removed. One caution is taken that even if there remains any token of length 2 character or less it is removed. In addition, all tokens are stripped of white spaces and symbols like "-" and "_". Finally, the tokens are lemmatized. Lemmatization brings the natural language English word to its root form. For instance, "fixes" is reduced to "fix".

Above processing of words to a list of tokens gives us the corpus that we want the vectorizer to fit on. We use TF-IDF vectorizer. TF-IDF puts higher weight on word which are very frequent in a document and rarer across documents. This is to ensure that the words selected for training LDA are discriminatory of the classes. Maximum number of features is defined here as the length of our feature space for our experiments. We define it as 1000.

3.4 Random Forest Classifier

Random forest (RF) is an ensemble approach that is specifically designed for decision tree classifier. Simply defined as collection of decision trees that are used for consensus on the decision. Each decision tree in RF is built on randomly selected subset of features. We control the depth and number of trees in the forest. For our case, we use python scikit-learn implementation of RF and we have set number of estimators to be 100 and max_depth of 100. Since, we have top 100 words and 2 topic probabilities, summing to 102 features, we believe this is a reasonable choice of hyperparameter. However, we can very well tune these parameter by optimization algorithms or grid search. The main benefits of RF are: 1) Feature importance is defined automatically, it removes the bias of predefined keywords and onus of curating important keywords for

our case; 2) As RF is an ensemble of many different decision tree classifiers, it avoids overfitting.

4 EVALUATION STRATEGY

Since this is a binary classifier, evaluating our results on the basis of confusion matrix seems apt. We will evaluate the performance of the model by precision, recall and F1 score. The definitions of these metrics are below:

- Precision: Defined as number of true labels as a fraction of total truly predicted labels, it says, number of true positives of all the positive predicted labels. Basically it indicates, of all positives that was predicted how many of them were actually positive. For precision score, the higher the better.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

- Recall: Defined as number of predicted true labels as a fraction of total true labels, it says, number of true positives of all the positive labels. Basically it means, of all the true labels how many did the model correctly recall. For recall score, the higher the better.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

- F1-measure: F1 measure is the harmonic mean of precision and recall. For both precision and recall to be high, we need to have high F1 score.

$$\text{F1} = \frac{2 * \text{Precision} * \text{Precision}}{\text{Precision} + \text{Recall}}$$

Since, F1 is harmonic mean of precision and recall, we need not mention precision explicitly as it is implied. So, in some cases we have mentioned recall and F1 score only. We also report F2 score in some cases. F2 score is derived from general form of F1 score. F2 score puts higher weight on recall, i.e. cost of making False Negative is higher.

5 EVALUATION OF BASELINE MODELS

5.1 Models

Table 2: Mockus et al. [16] dictionary

Category	Associated words
Corrective	fix, bug, problem, incorrect, correct, error, fixup, fail
Adaptive	new, change, patch, add, modify, update
Perfective	style, move, removal, cleanup, unneeded, rework

As discusses in earlier sections we have some keyword dictionary lookup models and some probabilistic models that have shown to perform well for change message classification. We discuss 4 models proposed in literature. We argue for one of them to be contestant for baseline for our project.

Table 3: Mauczka et al. [15] dictionary

Category	Associated words
Corrective	wrong, except, cause, null, warn, correct, valid, fail, bug, dump, opps, error, invalid, failure, incorrect, bugfix, fix, problem, bad, miss
Adaptive	context, introduce, future, appropriate, information, add, compatibility, simple, structure, configuration, available, require, init, function, internal, security, current, config, provide, easier, switch, default, faster, text, release, install, user, feature, old, useful, method, change, trunk, protocol, patch, version, replace, necessary, new, create
Perfective	clean, include, whitespace, dead, consistent, prototype, static, style, remove, definition, header, documentation, variable, inefficient, useless, cleanup, move, unused, declaration
Blacklist	cvs2svn, cvs, svn

5.1.1 Fixed Dictionary. Mockus et al. [16] studied and created classification based on word frequency analysis and keyword clustering. Mockus et al. derives inspiration from Swanson's classifications [17]. They have created a generic keyword based dictionary that corresponds to Swanson's classes [17]. It has been envisioned that there are 3 primary types of maintenance: fault fixes for keywords such as *fix, problem, incorrect, correct*, new code development for keywords *add, new, modify, update* and code improvement for keywords *cleanup, unneeded, remove, rework*. These keywords help is clustering of similar words based on some heuristics, followed by a simple classification algorithm. The keywords used in our implementation of the baseline is shown in Table-2. A developer survey has been used as validation method. Kappa coefficient is 0.5 indication moderate agreement. A log linear model was employed ti further this validation that indicated strong agreement between automatic classification and developer's classification. A claim is made that this method can be used for different projects as only basic information from version control system is used. This is a very simple algorithm and very comprehensible. It can be easily reproduced. So, it is a good candidate as baseline for our project.

5.1.2 Generalized Evolving Dictionary. Mauczka et al. [15] proposed a variation of keyword based approach to automated classification change messages. This work has been based on Mockus et al. [16] natural language "modification request" words. Weights for certain keywords were also defined. Since, the weight was subject to researchers tuning of their algorithms, it may have suffered a bias. A first set of 3 keywords is for each category is selected based on prior work [7] [16]. This keywords dictionary is extended by new words basis a frequency analysis algorithm until 80% of the project is classified. This dictionary was localized to a project starting from an initial globally accepted seed. A modified version of the original dictionary is shown in Table-3. However, they go on to do validation over cross-projects to establish a general keyword dictionary. Mauczka et al. [15] made an extension of Swanson's original classification by adding a *Blacklist* category. This dictionary can be seen as an extension of exemplary work of Mockus et

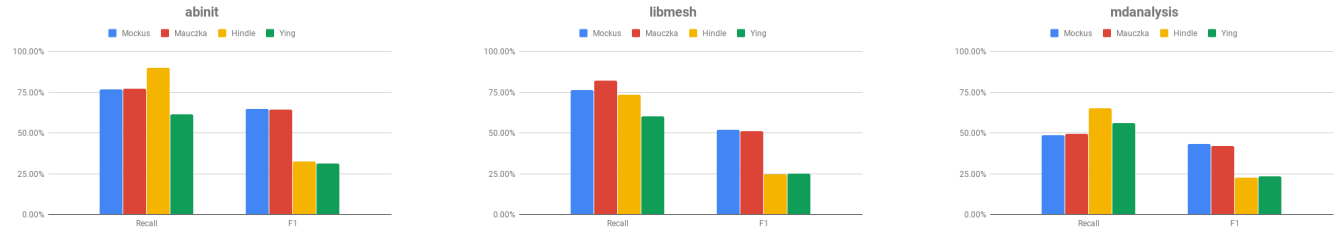

Figure 1: Evaluation of the baseline models on different data-sets

al. [16]. Moreover, it has been validated across different projects. Hence, this model can also be a viable candidate for baseline.

5.1.3 Latent Dirichlet Allocation with Manual Annotation of Topics. Hindle et al. [10] discusses Latent Dirichlet Allocation (LDA). The proposal says discovering change message classification topics over a window of time. LDA is a mixture model based method used to analyze unstructured data. Hindle et al. used certain parameters to conclude topic meaning. Top 10 words were seen among all 20 topics. Then they were manually assigned a topic name. If some topic did not give proper meaning it was discarded. The main objective is to create a trend in topics for a software project. For this purpose, similarity between two topics needs to be computed. A simple algorithm of 8 matches among top 10 words implied similar topic. There are many "magic" parameters used in this study. Nevertheless, as the generation of topic from commit message is an automated process. The final classification of a topic is based on human intervention. Hence, it is kind of semi-supervised classification based on LDA with software domain knowledge injection. Such a model is viable candidate for our baseline.

5.1.4 Semi-supervised Latent Dirichlet Allocation. Ying et al. [5] proposes an automated discovery of change message features using LDA. The algorithm includes a semi-supervised LDA method for change message classification. It is called semi-supervised as topic generation and data transformation through LDA is largely depended on *signifier documents*. A set of documents is built as signifier documents corresponding to Mauczka et al. [15] extended dictionary that is validated across project. The exact dictionary used is shown in Table-3. In a way, Ying et al. method is initialized using Mockus et al. [16] "keywords" and is based on Swanson's original change message classification [17]. Key difference of this algorithm is two folds:

- Injection of software engineering domain knowledge to generate topics and words distribution.
- Classification algorithm is similarity between topic distribution of signifier documents and target documents.

Validation is done through developer survey and accuracy is reported. Nearly, 85% of the messages were classified with nearly 70% accuracy. The algorithm is claimed to work on cross project analysis of software change message classification because of developer agreement. This algorithm is a viable candidate as our baseline. It's dependence on both software domain knowledge and automated change message classification with features specific to a project makes it a very strong candidate.

5.2 Comparing Models

Table 4: Evaluation metrics of baseline models

	Models	Recall	F1
abinit	Mockus et al.	76.60%	64.76%
	Mauczka et al.	77.30%	64.41%
	Hindle et al.	90.05%	32.35%
	Ying et al.	61.26%	31.19%
libmesh	Mockus et al.	76.14%	51.77%
	Mauczka et al.	81.99%	51.19%
	Hindle et al.	73.52%	24.54%
	Ying et al.	60.07%	25.16%
mdanalysis	Mockus et al.	48.68%	43.06%
	Mauczka et al.	49.48%	42.13%
	Hindle et al.	65.22%	22.72%
	Ying et al.	55.92%	23.42%

For the purpose of establishing a baseline, a version of each of the above is implemented. We sample the data with replacement 20 times and report the results as shown in Figure-1 and in Table 4.

The model for Mockus et al. [16] is implemented as a look-up dictionary as in Table-2. If any keyword is present first in the change message tokens, it is classified according to the category. Similar implementation is for Mauczka et al. [15]. The dictionary look-up is mentioned in Table-3. This is not the actual algorithm implemented in the paper. However, this is based on the final dictionary proposed for cross project change message analysis. The algorithm proposed by Ying et al. [5] is implemented exactly as mentioned. The signifier documents used are as in Table-3. Taking inspiration for using LDA as language model as in Hindle et al. [10]. A similar model is implemented. Instead of manual annotation of a topic classification, we generated only 2 topics and use *Corrective* keywords from Table-2 to annotate one of the topics as "corrective" topic. After a topic is annotated as buggy based on analysis of top 10 words with the dictionary, we classify the change message according to the probability of that topic. We also implement a model using LDA as language model and Random Forest as classifier on transformed features which are topic distributions. The difference between this LDA implementation and Ying et al. [5], is that Ying et al. model is semi-supervised and is classified basis similarity to topic distributions in signifier documents using cosine similarity.

We compare all the three projects based on the implementation of the baseline models as described above. Although, Mockus et al. [16] and Mauczka et al. [15] are not automated models but are based on dictionary lookup, they tend to perform better than the other models for the given dataset. One other thing to note is that all the models based on "corrective keywords" is better. That is to say, precision is low for all these models as they have been specifically designed for bug-fix indicating words. If we say that we require an automated language model rather a dictionary based model, Hindle et al. [10] model is the most acceptable baseline. Otherwise, our baseline is Mauczka et al. [15] keyword based model. So, we will evaluate our model, LDA-RF, against both these baselines.

6 EVALUATION OF LDA-RF

The text-processed data is split into 80% training and 20% testing. Using the TfidfVectorizer of python's scikit-learn, the training set of each project is vectorized and top 1000 features are sent to LDA for training. LDA is used to select features (words) that matter the most in this classification. We select top 100 words from both LDA topics. We also take the topic probabilities of each commit message. This algorithm is described below.

First, after text processing and train/test split, we go through LDA transformation to get both probabilities of topics and probabilities of feature words. Let's call the two topic probabilities as *Topic_0* and *Topic_1*. Table-6 describes the top 10 words found in each topic according to the probabilities of occurrence in the topic. It is quite evident that *Topic_1* corresponds to a topic that describes bug fix words.

After LDA transformation, we go through the following algorithm:

(1) Compute the probability of each word in the topic and store in a `word_topic_map`
(2) Go through each document.
    - Iterate over all the initial tokens present.
    - If a token matches the featured words, add the probability (from `word_topic_map`) of it being in a topic.
    - If the word is in both topics, sum the probabilities, use the maximum probability.
    - Along with these 100 columns, either 0 or otherwise computed in previous step, append the topic probabilities for the document as last two columns i.e. total 102 features. Create columns for these features.
(3) Split train data into 80% and 20% for training and validation.
(4) Feed these 102 columns to RF(max_depth=100, n_estimators=100) for training.

All the above steps including initial train/test split, training LDA and training RF is done 20 times to remove sampling bias. Finally average evaluation metrics are collected as shown in Table 5. Performance of LDA-RF model with corresponding baseline model i.e. Hindle et al is show in Fig. 2. Table 6 shows the top words corresponding to the topics generated by LDA.

7 VERIFY GENERALITY OF LDA-RF

To establish generality and robustness of our model we evaluate the performance of LDA-RF model on two different datasets. These datasets are natural-language software texts. The datasets have been generously provided by Real-world Artifical Intelligence for Software Engineering at department of Computer Science at NC State. First dataset is the built by extracting data from StackOverflow questions and second one pertains to technical debt i.e. code comments indicating tech debts in public open source projects.

Table 5: Evaluation metrics of baseline models

project	accuracy	precision	recall	f1	f2
abinit	88.33%	54.40%	79.88%	64.65%	72.98%
libmesh	89.75%	47.24%	85.54%	60.79%	73.52%
mdanalysis	85.37%	43.42%	81.42%	56.49%	69.12%

Table 6: LDA-RF Topics - Top Words

project	topic 0	topic 1
abinit	merge, branch, develop, abinit, trunk, remote, gitlab, initialize, abirules, routine	fix, error, file, update, add, test, compilation error, minor fix, correction, problem
libmesh	merge, request, pull, libmesh, mesh, update citation, refactor, pull request	fixed, warning, bugfix, patch, bug, change, enable, removed, disable, update citation, refactor
mdanalysis	merge, doc, mdanalysis, update, pull, request, version, change, release, issue	fix, test, issue, develop, added, updated, removed, code, mdanalysis, error

7.1 Datasets

7.1.1 Stack Overflow dataset. StackOverflow dataset is collected from stackoverflow.com. The data contains questions and tags associated with them. We specifically focus on 10 classes as shown in Table 7 created by analyzing associated tags. This table shows the amount of data (number of stackoverflow questions) we have in each category. While using this data we will mark one class as 1 and rest as 0 and sample desired amount of rows from this modified dataset.

7.1.2 Technical Debt Dataset. Technical Debt dataset is extracted from comments in the code of 10 open source projects as shown in Table 8. Each of the comments have been associated with one of the classes namely: defect, design, documentation, implementation, test or without any classification. For the purposes of testing the generality of our model, we classify each of the technical details as 1 and the rows without classification as 0.

7.2 Evaluation

Our LDA based model has claims to create a generic and local language model for a specific project. This translated into the fact that our model when modeled as a binary classifier, how efficient

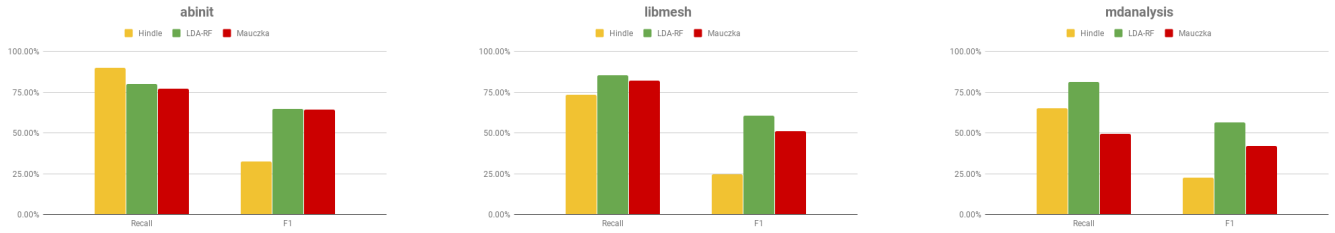

Figure 2: Evaluation of LDA-RF with baseline

Table 7: StackOverflow.com dataset

project	Counts
academia	8081
cs	9270
diy	15287
expressionengine	9469
judaism	14393
photo	12733
rpg	11208
scifi	19546
ux	13424
webmasters	17911
total	131322

is it to capture the features of the natural language. We used the same metrics we have used to evaluate our language model on github commits dataset i.e F1 and Recall. However, we also report accuracy, precision and F2 score here. These are only for references and verifying correctness of the model.

7.2.1 StackOverflow. We have 10 classes in this dataset as shown in Table 7. For creating the dataset we follow following steps for one class at a time:

- Label all data for a class as 1 and others as 0, and shuffle the data.
- Split the data randomly in 80:20 for train and test, respectively.
- From the training data create a random dataset of 2:1 label counts i.e. one-third of data should be 1s and two-thirds as 0s and drop other data points. Shuffle the training data.
- Split training data in training and validation 80:20.
- Train on training data, validate on validation.
- Test on testing data and record metrics
- Repeat above steps 20 times and report average metrics.

Recall and F1 for StackOverflow dataset is as shown in Fig. 3. We see that on average we have nearly 89% recall and 79.2% F1 score. We find that average precision is less that brings down the F1 score. Depending on the use case we can assign relative importance to recall or not. Therefore, we have included precision and F2 score in Table 3. In particular, it can be seen that this model consistently has better recall than precision which was evident from github commits dataset too. We can see from the Table 10 that the classes with high recall and F1 produce much coherent topics (Topic 1 corresponds to topic related to the category). This model is generic enough to learn features of natural language in texts like stackoverflow, which is loosely SE based texts.

7.2.2 Tech Debt. Tech Debt data is merely comments from the code-base of the project. We use the same metrics as above i.e. F1 and recall to gauge the performance of our model. The evaluation graph is as shown in Fig. 4. We see that recall is consistently high for all the projects except for 4 classes. F1 score is greater than 50% in all classes. Table 12 shows the words associated each of the topic. In 'jEdit-4.2' project we see a lot of overlap between these words i.e. technical terms. We can attribute this fact for this project to have one of the lower recall and F1 scores. This fact is also evident from the 'emf-2.4.1' and 'apache-ant-1.7.0' project. Also, it is worthy to note that the average recall is 76.2% and F1 is 66.2%. Average recall as established in the baseline model Huang et al [12] is 77.33% and average F1 is 73.7%. Our model does not outperform Huang et al on average. However, in 2 of the projects namely 'columba-1.4-src' and 'jruby-1.4.0' performs better than the baseline model. Although, both our approach and Huang et al. [12] approach is a generic approach that generalizes to any project. Huang et al. [12] use a Naive Bayes classifier on top of words transformed to a Vector Space Model. Our model uses a probabilistic framework by using LDA built on top of TF-IDF vector and the features from LDA are used to train a Random Forest classifier.

8 RESEARCH QUESTIONS - CONCLUSIONS

RQ1: *Should we specify generic keywords for all software projects for change message classification?*

It can be seen from Fig. 1 that keyword based models perform better than Hindle et al model in 'libmesh' and 'mdanalysis' project. These models do have a better F1 score than all other models. It is interesting to note that project specific evolving dictionary Mauczka et al. model performs better than generic dictionary. Hindle et al constantly has comparable or greater recall for all three projects. Moreover, Hindle et al model is automated and can be project specific. Both these conclusions points to the fact being project specific reduces a lot of false negatives. However, it also leads to a rise in false positive in case of Hindle. We tackle this problem by doing project level feature engineering by choosing LDA transformed features. We see that from Fig. 2 that LDA-RF outperform Hindle et al. F1 score is also higher indicating that false positive has also decreased. We can also see from Table 6 that it is not necessary

Table 8: Technical Debt Dataset

project	DEFECT	DESIGN	DOCUMENTATION	IMPLEMENTATION	TEST	WITHOUT_CLASSIFICATION
apache-ant-1.7.0	13	95	0	13	10	3967
apache-jmeter-2.10	22	316	3	21	12	7683
argouml	127	801	30	411	44	8039
columba-1.4-src	13	126	16	43	6	6264
emf-2.4.1	8	78	16	2	0	4286
hibernate-distribution-3.3.2.GA	52	355	1	64	0	2496
jEdit-4.2	43	196	0	14	3	10066
jfreechart-1.0.19	9	184	0	15	1	4199
jruby-1.4.0	161	343	2	110	6	4275
sql12	24	209	2	50	1	6929

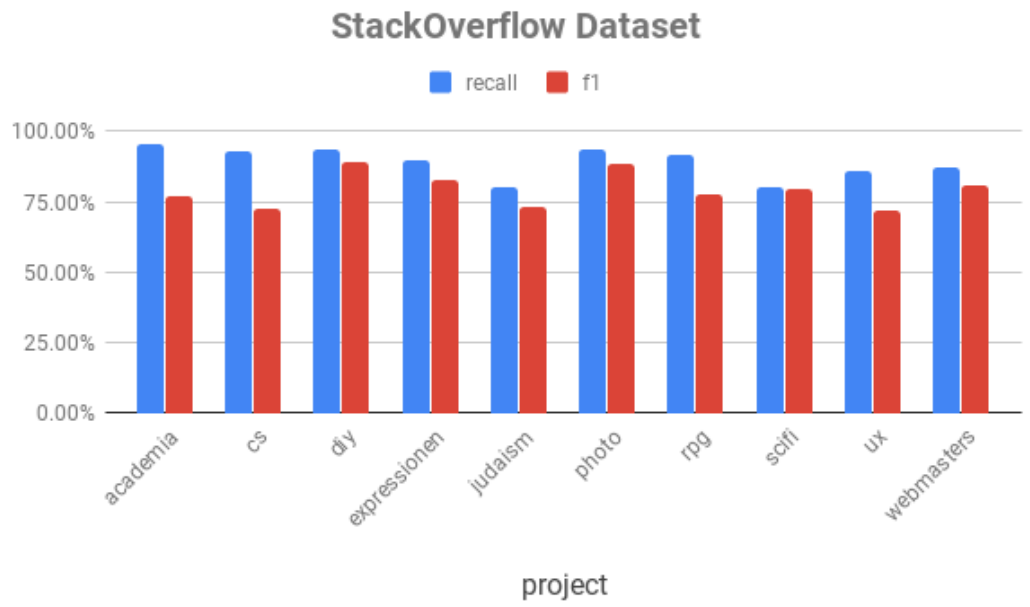

Figure 3: Performance on Stack Overflow dataset

that only those words have higher weight that are mentione in dicitonary models like in Table 2 or Table 3. For instance, 'compilation error' is a two word phrase that is characteristic of a bug-fixing commit message. The dictionary based model do not account for phrases. Even if they do, it will become unfeasible to determine common phrases for different projects. Hence, we should not specify keywords that do not localize to a project as they do not capture important information that is a characteristic of the project for change message classification.

RQ2: *Does a general automated model trained on specific project perform better than generic dictionary based classifiers?*

Yes. As it is evident from Fig. 2, LDA-RF outperforms Mauckza et al. [15] keywords based model in all projects. Moreover, LDA-RF has the best F1-score and recall score on averag across all projects. LDA-RF does this well because it captures local features of a project that are characteristic of the project. Only, in one project, 'abinit', Hindle et al. probabilistic and general model has better recall than LDA-RF. However, in reaching such high recall this model has lots of false positive. Both Hindle et al model and LDA-RF does better than keyword based model in terms of recall. LDA-RF has smart feature generation, therefore goes on to have better F1 score too.

RQ3: *How many latent topics to keep in LDA? Are these topics comprehensible?*

To answer for comprehensibility part, we need to understand that apart from *Keyword search* model, no other model offers comprehensibility other than LDA-RF. In fact, LDA-RF offers a better comprehensibility than *Keyword search* model. It is such because, LDA-RF identifies keywords that are local to the software project. Not only does LDA-RF capture localized keywords, but it also captures keywords that are similar to corrective category keywords

Table 9: StackOverflow Dataset Metrics

project	accuracy	precision	recall	f1	f2
academia	96.35%	64.30%	95.44%	76.71%	86.89%
cs	94.95%	59.38%	92.81%	72.27%	83.26%
diy	97.40%	85.56%	93.41%	89.31%	91.72%
expressionengine	97.27%	76.64%	89.88%	82.72%	86.87%
judaism	93.42%	67.01%	80.24%	73.01%	77.17%
photo	97.54%	83.32%	93.77%	88.23%	91.47%
rpg	95.53%	67.49%	91.62%	77.72%	85.50%
scifi	93.85%	79.36%	79.96%	79.65%	79.84%
ux	93.11%	61.60%	85.65%	71.65%	79.44%
webmasters	94.24%	74.92%	87.00%	80.50%	84.28%

Table 10: StackOverflow Topics - Top words

project	topic 0	topic 1
academia	use, page, site, like, user, would, need, way, set, one, websit, googl, look, imag	paper, research, one, student, would, work, univers, year, phd, question, time, know, ask, get, book
cs	use, page, site, user, like, would, com, work, imag, want, get, look, googl, websit, name	problem, one, time, would, algorithm, languag, number, question, find, know, say
diy	use, one, would, like, page, know, site, user, time, question, say, name, exampl, com, differ	water, wall, light, wire, hous, use, would, need, floor, instal, one, like, switch, replac
expressionengine	one, would, use, like, know, time, question, make, say, think, look, could	entri, page, site, field, channel, user, use, form, exp, file, categori, name
judaism	use, page, site, user, like, want, work, would, set, get, com, look, need	one, say, would, know, sourc, time, question, peopl, day, answer, make, torah, read
photo	page, use, site, user, name, com, file, one, would, websit, like, work, exampl, googl, question	camera, len, use, photo, would, one, like, get, light, take, canon, look, imag, pictur
rpg	use, page, site, user, name, would, like, work, imag, websit, com, googl, one, know, problem	would, charact, one, use, level, player, like, attack, spell, game, make, get, rule, power, book
scifi	use, page, user, site, would, like, need, work, want, problem, get, com, way, imag, websit	one, book, would, time, know, read, say, like, year, seem, stori, rememb, peopl, question, power, charact
ux	one, would, use, know, time, say, like, get, question, make, seem, read, take, look	user, page, use, site, design, button, websit, like, search, form, list, com, field, exampl
webmasters	one, would, use, time, like, know, question, say, make, get, seem, look, could, take, think	site, page, com, googl, websit, user, use, domain, link, http, search, file, content, imag, url

used in *Keyword search* model. It can be seen in Table 6. Hindle et al. [10] used LDA but number of topics was arbitrary. Nonsensical topics were later weeded out by human intervention. However, LDA-RF sets number of topics equal to number of classes. The objective that LDA-RF has that each topic should provide with enough corresponding features in order to distinguish the two classes. When we look at Table 6, we can find that the two topics are distinctly different. Moreover, many of the top words match to the generic keywords given by Mauckza et al. [15] and Mockus et al. [16]. So, we can conclude keeping topics corresponding to classes helps in comprehensibility.

RQ4: *How does the model generalize to different natural-language software texts?*

We verified the generality of LDA-RF by testing on two different software texts. On the stackoverflow set we find that average recall across projects is 89% and F1 as 79%. Although, we do not have a baseline to compare against, we can still day that recall and F1 score is very high. Based on this score it can be claimed that LDA-RF generalizes to stackoverflow dataset. Comparing against the baseline on the tech debt dataset, LDA-RF has comparable score with the baseline in terms of recall. F1 score is not very far from baseline. We can actually explain why LDA-RF performs poorly in few projects. This is due to the fact that LDA-RF offers more comprehensibility. We see from the table 12 that the projects with lower F1 score have higher correlated top words in the topics. In other words, those projects have correlated topics. Other projects

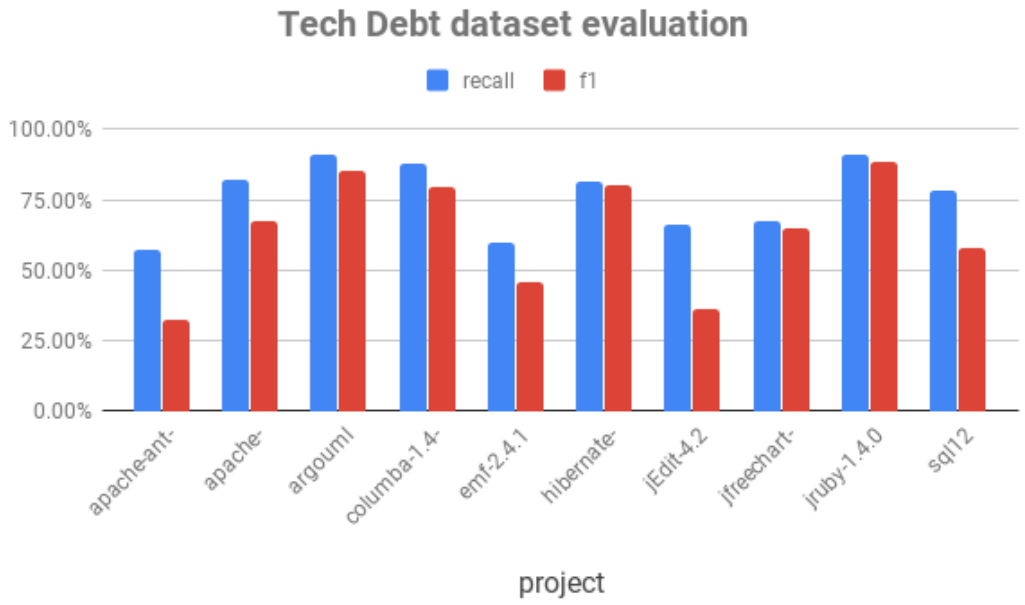

Figure 4: Performance on Technical Debt datasets

Table 11: Technical Debt Dataset Metrics

project	accuracy	precision	recall	f1	f2
apache-ant-1.7.0	92.02%	23.19%	56.89%	32.10%	42.66%
apache-jmeter-2.10	96.11%	57.33%	82.23%	67.13%	75.27%
argouml	95.39%	80.76%	90.95%	85.52%	88.69%
columba-1.4-src	98.61%	73.63%	87.83%	79.57%	84.16%
emf-2.4.1	96.50%	37.85%	59.79%	45.57%	52.76%
hibernate-distribution-3.3.2.GA	93.54%	78.73%	81.54%	79.98%	80.88%
jEdit-4.2	92.02%	26.33%	66.32%	36.36%	48.34%
jfreechart-1.0.19	96.61%	64.45%	67.11%	64.50%	65.78%
jruby-1.4.0	96.90%	85.93%	90.92%	88.30%	89.84%
sql12	95.30%	47.32%	78.38%	58.09%	68.29%

where LDA-RF generated distinct top words, i.e. LDA based feature generation is in tandem with the features required for classification, we have high recall and F1, ad higher than the baseline Huang et al. [12] model. Hence, LDA-RF generalizes to different kinds of natural-language software texts.

9 CONCLUSION

Our model is a generic model that generalizes on a variety of datasets. We see that our model which is an LDA-based classifier works with github commit messages for change message classification, also with stackoverflow dataset for topic classification and tech debt for comments classification. Though, we should note to achieve this performance on StackOverflow and tech debt dataset, we had to tune some hyperparameters associated with our model. We have kept number of features as 1000 and top N words from topics to be 100. In change message classification model, these values were 100 and 25 respectively. We find that a small amount of tuning is required to created project specific model. Another point to note is that, we used TFIDFVectorizer in these experiments, however we have noted that CountVectorizer gives similar performance. TFIDF tends to give more coherent and distinct topics. Our earlier analysis of using CountVectorizer to be the choice needs a thorough analysis.

Since, the performance in each of these datasets our model performs well within the range of the performance of baseline models and also outperforming in certain cases. It can be established that LDA-RF i.e. LDA based Random Forest classifier is very generic and robust to datasets. The property of this model to generalize

Table 12: Technical Debt Topic - Top words

project	topic 0	topic 1
apache-ant-1.7.0	file, ignore, attribute, property, set, directory, specified, create, need, name	checkstyle, ignore, visibilitymodifier, class, default, method, end, get, test
apache-jmeter-2.10	todo, used, set, test, file, check, default, use, name	nls, non nls, noop, panel, main, org, apache, see, apache jmeter, main panel
argouml	todo, class, end, set, needed, name, element, model, add, nothing	see, org, argouml, uml, object, java, lang, lang object,
columba-1.4-src	nls, non nls, message, folder, add, todo, create, get, new, author	columba, text, org, non javadoc, javadoc see, event, update, tooltip,
emf-2.4.1	value, byte, fill, object, non nls, nls, copied, non, interface, javadoc copied, javadoc	ignore, create, feature, nothing, command, class, add, file, new, content
hibernate-distribution-3.3.2.GA	todo, cache, check, add, column, alias, new, sql, object, use, one, node, key, note, method	collection, type, return, null, element, name, join, support, property, table, first, handle, select, array
jEdit-4.2	constructor, variable, type, line, case, instance, value, buffer, need, check, object, note, instance variable	method, class, member, private, private member, create, init, init method, find, package, workaround, inner class, inner
jfreechart-1.0.19	draw, argument, defer, defer argument, checking, argument checking, test, nothing, todo, paint	check, series, add, item, data, point, value, case, try, plot, independence, set, null, axis, fixme, check independence
jruby-1.4.0	line, fixme, switch, set, thread, constant, new, add, receiver, module, ignore, true, create, error, node, exception	method, value, args, class, block, scope, ruby, string, mri, array, name, variable, case, java, call
sql12	sql, error, todo, see, null, session, non, class, row, user, squirrel, oracle, nothing, javadoc, test, non javadoc	table, file, column, data, name, type, use, new, object, set, need, method, value, add, create, graph, text, expected, read

and localize to a project is effective in extracting useful features from natural language SE texts. The projects that generally have lower F1 and recall scores have more correlation between the two topics. For our case, as a binary classifier, we have kept number of topics to be 2. It is our conscious decision that the two topics should correspond to the two classes in order to learn features of different classes. However, in case of high correlation between the topics we need to figure out a way to incorporate more feature extraction technique. We can either let LDA learn for a larger number of iteration rather than 20 that is default for our model. We could also explore creating more topics and understanding topic perplexity to incorporate more granular features.

REFERENCES

[1] Amritanshu Agrawal, Wei Fu, and Tim Menzies. 2016. What is Wrong with Topic Modeling? (and How to Fix it Using Search-based SE). *CoRR* abs/1608.08176 (2016). arXiv:1608.08176 http://arxiv.org/abs/1608.08176

[2] Amritanshu Agrawal, Huy Tu, and Tim Menzies. 2018. Can You Explain That, Better? Comprehensible Text Analytics for SE Applications. *CoRR* abs/1804.10657 (2018). arXiv:1804.10657 http://arxiv.org/abs/1804.10657

[3] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON '08)*. ACM, New York, NY, USA, Article 23, 15 pages. https://doi.org/10.1145/1463788.1463819

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993–1022. http://dl.acm.org/citation.cfm?id=944919.944937

[5] Ying Fu, Meng Yan, Xiaohong Zhang, Ling Xu, Dan Yang, and Jeffrey D. Kymer. 2015. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology* 57 (2015), 369–377.

[6] Scott Grant, James R. Cordy, and David B. Skillicorn. 2012. Using Topic Models to Support Software Maintenance. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering (CSMR '12)*. IEEE Computer Society, Washington, DC, USA, 403–408. https://doi.org/10.1109/CSMR.2012.51

[7] Ahmed E. Hassan. 2008. Automated Classification of Change Messages in Open Source Projects. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08)*. ACM, New York, NY, USA, 837–841. https://doi.org/10.1145/1363686.1363876

[8] Lile P. Hattori and Michele Lanza. 2008. On the Nature of Commits. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. IEEE Press, Piscataway, NJ, USA, III–63–III–71. https://doi.org/10.1109/ASEW.2008.4686322

[9] Abram Hindle, Daniel M. German, and Ric Holt. 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 (MSR '08)*. ACM, New York, NY, USA, 99–108. https://doi.org/10.1145/1370750.1370773

[10] A. Hindle, M. W. Godfrey, and R. C. Holt. 2009. In *What's hot and what's not: Windowed developer topic analysis*. IEEE, 339–348.

[11] Thomas Hofmann. 2001. Unsupervised Learning by Probabilistic Latent Semantic Analysis. *Mach. Learn.* 42, 1-2 (Jan. 2001), 177–196. https://doi.org/10.1023/A:1007617005950

[12] Qiao Huang, Emad Shihab, Xin Xia, David Lo, and Shanping Li. 2018. Identifying Self-admitted Technical Debt in Open Source Projects Using Text Mining. *Empirical Softw. Engg.* 23, 1 (Feb. 2018), 418–451. https://doi.org/10.1007/s10664-017-9522-4

[13] Sunghun Kim, E. James Whitehead, Jr., and Yi Zhang. 2008. Classifying Software Changes: Clean or Buggy? *IEEE Trans. Softw. Eng.* 34, 2 (March 2008), 181–196. https://doi.org/10.1109/TSE.2007.70773

[14] George Mathew, Amritanshu Agrawal, and Tim Menzies. 2016. Trends in Topics at SE Conferences (1993-2013). *CoRR* abs/1608.08100 (2016). arXiv:1608.08100 http://arxiv.org/abs/1608.08100

[15] Andreas Mauczka, Markus Huber, Christian Schanes, Wolfgang Schramm, Mario Bernhart, and Thomas Grechenig. 2012. Tracing Your Maintenance Work — a Cross-project Validation of an Automated Classification Dictionary for Commit Messages. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering (FASE'12)*. Springer-Verlag, Berlin, Heidelberg, 301–315. https://doi.org/10.1007/978-3-642-28872-2_21

[16] Audris Mockus and Lawrence G. Votta. 2000. Identifying Reasons for Software Changes Using Historic Databases. In *Proceedings of the International Conference on Software Maintenance (ICSM'00) (ICSM '00)*. IEEE Computer Society, Washington, DC, USA, 120–. http://dl.acm.org/citation.cfm?id=850948.853410

[17] E. Burton Swanson. 1976. The Dimensions of Maintenance. In *Proceedings of the 2Nd International Conference on Software Engineering (ICSE '76)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 492–497. http://dl.acm.org/citation.cfm?id=800253.807723

[18] Stephen W. Thomas. 2011. Mining Software Repositories Using Topic Models. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, USA, 1138–1139. https://doi.org/10.1145/1985793.1986020

[19] Meng Yan, Xin Xia, Emad Shihab, David Lo, Jianwei Yin, and Xiaohu Yang. 2018. Automating change-level self-admitted technical debt determination.