Predicting Good Configurations for GitHub and Stack Overflow Topic Models

Christoph Treude and Markus Wagner
School of Computer Science
University of Adelaide
{christoph.treude|markus.wagner}@adelaide.edu.au

***Abstract*—Software repositories contain large amounts of textual data, ranging from source code comments and issue descriptions to questions, answers, and comments on Stack Overflow. To make sense of this textual data, topic modelling is frequently used as a text-mining tool for the discovery of hidden semantic structures in text bodies. Latent Dirichlet allocation (LDA) is a commonly used topic model that aims to explain the structure of a corpus by grouping texts. LDA requires multiple parameters to work well, and there are only rough and sometimes conflicting guidelines available on how these parameters should be set. In this paper, we contribute (i) a broad study of parameters to arrive at good local optima for GitHub and Stack Overflow text corpora, (ii) an a-posteriori characterisation of text corpora related to eight programming languages, and (iii) an analysis of corpus feature importance via per-corpus LDA configuration. We find that (1) popular rules of thumb for topic modelling parameter configuration are not applicable to the corpora used in our experiments, (2) corpora sampled from GitHub and Stack Overflow have different characteristics and require different configurations to achieve good model fit, and (3) we can predict good configurations for unseen corpora reliably. These findings support researchers and practitioners in efficiently determining suitable configurations for topic modelling when analysing textual data contained in software repositories.**

***Index Terms*—Topic modelling, corpus features, algorithm portfolio.**

I. Introduction

Enabled by technology, humans produce more text than ever before, and the productivity in many domains depends on how quickly and effectively this textual content can be consumed. In the software development domain, more than 8 million registered users have contributed more than 38 million posts on the question-and-answer forum Stack Overflow since its inception in 2008 [1], and 67 million repositories have been created on the social developer site GitHub which was founded in the same year [2]. The productivity of developers depends to a large extent on how effectively they can make sense of this plethora of information.

The text processing community has invented many techniques to process large amounts of textual data, e.g., through topic modelling [3]. Topic modelling is a probabilistic technique to summarise large corpora of text documents by automatically discovering the semantic themes, or topics, hidden within the data. To make use of topic modelling, a number of parameters have to be set.

Agrawal et al. [4] provide a recent overview of literature on topic modelling in software engineering. In the 24 articles they highlight, 23 of 24 mention instability in a commonly used technique to create topic models, i.e., with respect to the starting conditions and parameter choices. Despite this, all use default parameters, and only three of them perform tuning of some sort—all three use some form of a genetic algorithm.

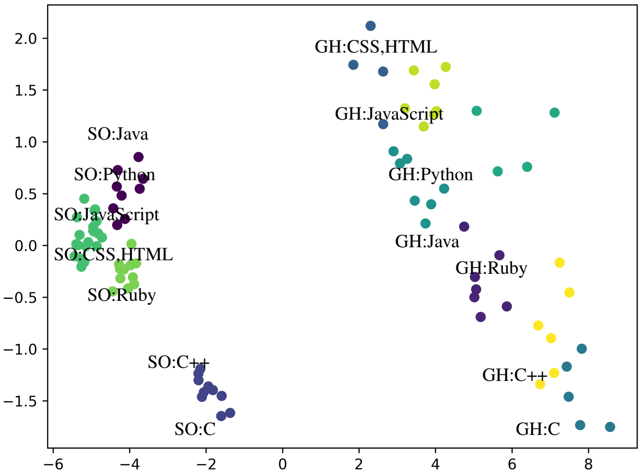


Fig. 1: Clustered corpora in 2d. The colour encodes the cluster assigned to each corpus. GH/SO refer to GitHub and Stack Overflow. The axes do not have any particular meaning in projections like these.

Even researchers who apply optimisation to their topic modelling efforts do not "learn" higher-level insights from their tuning, and there is very limited scientific evidence on the extent to which tuning depends on features of the corpora under analysis. For example, is the tuning that is needed for data from Stack Overflow different to the tuning needed for GitHub data? Does textual content related to some programming languages require different parameter settings compared to the textual content which discusses other programming languages? In this paper, we employ techniques from Data-Driven Software Engineering (DSE) [5] and Data Mining Algorithms Using/Used-by Optimizers (DUO) [6] on 40 corpora sampled from GitHub and 40 corpora sampled from Stack Overflow to investigate the impact of per-corpus configuration on topic modelling. We ask two research questions:

RQ1 What are the optimal topic modelling configurations for textual corpora from GitHub and Stack Overflow?

RQ2 Can we automatically select good configurations for unseen corpora based on their features alone?

We find that (1) popular rules of thumb for topic modelling parameter configuration are not applicable to textual corpora mined from software repositories, (2) corpora sampled from GitHub and Stack Overflow have different characteristics and require different configurations to achieve good model fit, and (3) we can predict good configurations for unseen corpora reliably based on corpus features. Figure 1 shows the corpora used in our experiments clustered in 2d based on their features after applying principal component analysis. The figure illustrates that textual corpora related to different programming languages and corpora taken from different sources (GitHub and Stack Overflow) can indeed be distinguished based on their features. Even across sources, the language-specific characteristics of the documents persist and corpora belonging to similar programming languages are close to each other. Moreover, the programming languages are in the vicinity of their spiritual ancestors and successors (e.g., C and C++).[1] We use this finding as a starting point for ad hoc per-corpus configuration of topic modelling of textual corpora mined from software repositories. Our predictions outperform the baseline by 4% and are less than 1% away from the virtual best solver.

These findings provide insight into the impact of corpus features on topic modelling in software engineering. They inform future work about efficient ways of determining suitable configurations for topic modelling, ultimately making it easier and more reliable for developers and researchers to understand the large amounts of textual data they are confronted with.

This article is structured as follows. First, we provide an introduction to topic modelling in Section II. Then, we describe in Section III our data collection, and we provide a first statistical characterisation of the data. In Section IV, we report on our tuning on individual corpora. Section V provides insights gained from our per-corpus configuration and from the per-corpus parameter selection. Section VI identifies threats which may affect the validity of our findings, before we discuss related work in Section VII. Finally, we conclude with a summary and by outlining future work.

II. Topic Modelling

Topic modelling is an information retrieval technique which automatically finds the overarching topics in a given text corpus, without the need for tags, training data, or predefined taxonomies [7]. Topic modelling makes use of word frequencies and co-occurrence of words in the documents in a corpus to build a model of related words [3]. Topic modelling has been applied to a wide range of artefacts in software engineering research, e.g., to understand the topics that mobile developers are talking about [8], to prioritise test cases [9], and to detect duplicate bug reports [10].

The technique most commonly used to create topic models is Latent Dirichlet Allocation (LDA), a three-level hierarchical Bayesian model, in which each item of a collection is modelled as a finite mixture over an underlying set of topics [3]. A document's topic distribution is randomly sampled from a Dirichlet distribution with hyperparameter α, and each topic's word distribution is randomly sampled from a Dirichlet distribution with hyperparameter β. α represents document-topic density—with a higher α, documents contain more topics—while β represents topic-word density—with a higher β, topics contain most of the words in the corpus [11]. In addition, the number of topics—usually denoted as k—is another parameter needed to create a topic model using LDA. While many studies use the default settings for these parameters ($\alpha = 1.0$, $\beta = 0.01$, $k = 100$; other sources suggest $\alpha = 50/k$ and $\beta = 0.1$ [12]), in recent years, researchers have found that the defaults do not lead to the best model fit and have investigated the use of optimisation to determine good parameter values (e.g., [4]). To measure model fit, researchers have employed *perplexity*, the geometric mean of the inverse marginal probability of each word in a held-out set of documents [13], which we also use in this work. Low perplexity means the language model correctly guesses unseen words in test data.

In this work, we set out to investigate to what extent the optimal parameter settings for topic modelling depend on characteristics of the corpora being modelled. All our experiments were conducted with the LDA implementation Mallet, version 2.0.8.[2]

III. GitHub and Stack Overflow Corpora

We now describe how we collected the documents used in our research. We define the features that we use to describe them, and we characterise them based on these features.

A. Data Collection

To cover different sources and different content in our corpora, we sampled textual content related to eight programming languages from GitHub and Stack Overflow. We selected the set of languages most popular across both sources: C, C++, CSS, HTML, Java, JavaScript, Python, and Ruby. As Stack Overflow has separate tags for HTML and HTML5 while GitHub does not distinguish between them, we considered both tags. Similarly, Stack Overflow distinguishes Ruby and Ruby-on-Rails, while GitHub does not.

For each programming language, we collected 5,000 documents which we stored as five corpora of 1,000 documents each to be able to generalise beyond a single corpus. Our sampling and pre-processing methodology for both sources is described in the following.

Stack Overflow sampling. We downloaded the most recent 5,000 threads for each of the eight programming languages through the Stack Overflow API. Each thread forms one document (title + body + optional answers, separated by a single space).

Stack Overflow pre-processing. We removed line breaks (`\n` and `\r`), code blocks (content surrounded by `<pre><code>`), and all HTML tags from the documents.

[1] See Section III-C for details.

[2] http://mallet.cs.umass.edu/download.php, last accessed on 24 December 2018.

In addition, we replaced the HTML symbols `&quot;` `&amp;` `&gt;` and `&lt;` with their corresponding character, and we replaced strings indicating special characters (e.g., `&#39;`) with double quotes. We also replaced sequences of whitespace with a single space.

GitHub sampling. We randomly sampled `README.md` files of GitHub repositories that used at least one of the eight programming languages, using a script which repeatedly picks a random project ID between 0 and 120,000,000 (all GitHub repositories had an ID smaller than 120,000,000 at the time of our data collection). If the randomly chosen GitHub repository used at least one of the eight programming languages, we determined whether it contained a README file (cf. [14]) in the default location (https://github.com/⟨user⟩/⟨project⟩/blob/master/README.md). If this README file contained at least 100 characters and no non-ASCII characters, we included its content as a document in our corpora.

GitHub pre-processing. Similar to the Stack Overflow pre-processing, we removed line breaks, code blocks (content surrounded by at least 3 backticks), all HTML tags, single backticks, vertical and horizontal lines (often used to create tables), and comments (content surrounded by `<!-- ... -->`). We also removed characters denoting sections headers (# at the beginning of a line), characters that indicate formatting (*, _), links (while keeping the link text), and badges (links preceded by an exclamation mark). In addition, we replaced the HTML symbols `&quot;` `&amp;` `&gt;` and `&lt;` with their corresponding character, and we replaced strings indicating special characters (e.g., `&#39;`) with double quotes. We also replaced sequences of whitespace with a single space.

B. Features of the Corpora

We are not aware of any related work that performs per-corpus configuration of topic modelling and uses the features of a corpus to predict good parameter settings for a particular corpus. As mentioned before, Agrawal et al. [4] found that only a small minority of the applications of topic modelling to software engineering data apply any kind of optimisation, and even the authors who apply optimisations do not "learn" higher-level insights from their experiments. While they all conclude that parameter tuning is important, it is unclear to what extent the tuning depends on corpus features. To enable such exploration, we calculated the 24 corpus features listed in Table I (each feature is calculated twice, once with and once without taking into account stopwords[3] to account for potential differences between feature values with and without stopwords, e.g., affecting the number of unique words).

We computed the number of characters in each entire corpus as well as the number of characters separately for each document in a corpus. To aggregate the number of characters per document to corpus level, we created separate features for their median and their standard deviation. This allowed us to capture typical document length in a corpus as well as diversity of the corpus in terms of document length. Similarly, we calculated the number of words and the number of unique words for each corpus and for each document.

While these features capture the basic characteristics of a document and corpus in terms of length, they do not capture the nature of the corpus. To capture this, we relied on the concept of *entropy*. As described by Koutrika et al. [15], "the basic intuition behind the entropy is that the higher a document's entropy is, the more topics the document covers hence the more general it is". To calculate entropy, we used Shannon's definition [16]:

$$-\sum_i p_i \log p_i \qquad (1)$$

where p_i is the probability of word number i appearing in the stream of words in a document. We calculated the entropy for each corpus and each document, considering the textual content with and without stopwords separately. Note that the runtime for calculating these values is at least $\Omega(n)$ since the frequency of each word has to be calculated separately.

C. Descriptive Statistics

While we have defined many corpus features, it is unclear how correlated these are, and whether the same relationships hold for GitHub README files and Stack Overflow discussions. Figure 2 shows the correlations based on Pearson product-moment correlation coefficients between the 24 features and clustered with Wards hierarchical clustering approach.[4] As expected, the entropy-based features are correlated, as are those based on medians and standard deviations—this becomes particularly clear when we consider the relationships across all corpora (Figure 2c).

There are, however, differences between the two sources GitHub and Stack Overflow. For example, the stdevDocumentEntropy across the GitHub corpora is less correlated with the other features than among the Stack Overflow corpora. A reason for this could be that the README files from GitHub are different in structure from Stack Overflow threads. Also, the median-based feature values of the GitHub corpora are less correlated with the other features than in the Stack Overflow case. We conjecture this is because the README files vary more in length than in the Stack Overflow case, where thread lengths are more consistent.

Next, we will investigate differences between the programming languages. As we have 24 features and eight programming languages across two sources, we will limit ourselves to a few interesting cases here.

In Figure 3, we start with a few easy-to-compute characteristics. For example, we see in the first row that GitHub documents are about twice as long as Stack Overflow discussions (see corpusWords). The distribution in the union shows this as well, with the left and the right humps (largely) coming from the two different sources. The trend remains the same if we remove stop words (see the second row). This already shows

[3] We used the "Long Stopword List" from https://www.ranks.nl/stopwords, last accessed on 24 December 2018.

[4] Implementation provided by asapy [17], https://github.com/mlindauer/asapy, last accessed on 24 December 2018.

TABLE I: Features of Corpora. Features include the number of characters, words, and unique words as well as entropy, calculated separately for entire corpora and single documents.

	scope		
	corpus	document (agg. via median)	document (agg. via std dev)
# characters	*with stopwords:* corpusCharacters *without stopwords:* corpusCharactersNoStopwords	*with stopwords:* medianDocumentCharacters *without stopwords:* medianDocumentCharactersNoStopwords	*with stopwords:* stdevDocumentCharacters *without stopwords:* stdevDocumentCharactersNoStopwords
# words	*with stopwords:* corpusWords *without stopwords:* corpusWordsNoStopwords	*with stopwords:* medianDocumentWords *without stopwords:* medianDocumentWordsNoStopwords	*with stopwords:* stdevDocumentWords *without stopwords:* stdevDocumentWordsNoStopwords
# unique words	*with stopwords:* corpusUniqueWords *without stopwords:* corpusUniqueWordsNoStopwords	*with stopwords:* medianDocumentUniqueWords *without stopwords:* medianDocumentUniqueWordsNoStopwords	*with stopwords:* stdevDocumentUniqueWords *without stopwords:* stdevDocumentUniqueWordsNoStopwords
entropy	*with stopwords:* corpusEntropy *without stopwords:* corpusEntropyNoStopwords	*with stopwords:* medianDocumentEntropy *without stopwords:* medianDocumentEntropyNoStopwords	*with stopwords:* stdevDocumentEntropy *without stopwords:* stdevDocumentEntropyNoStopwords

that we could tell the two sources apart with good accuracy by just considering either one of these easy-to-compute features. Despite this, the reliable classification of a single document does not appear to be as straightforward based on just the number of unique words that are not stop words: we can see in the third row that the two distributions effectively merged.

Looking at entropy, which is significantly more time-consuming to compute, we can see the very same characteristics (see bottom two rows in Figure 3). As seen before in Figure 2, entropy and word counts are correlated, but not as strongly with each other than some of the other measures.

Interestingly, GitHub documents contain fewer stop words (about 40%) than Stack Overflow documents (almost 50%). This seems to show the difference of the more technical descriptions present in the former in contrast to the sometimes more general discussion in the latter, which is reflected in the higher entropy of GitHub content compared to Stack Overflow content.

Next, we briefly investigate the heavy (right) tail of the Stack Overflow characteristics. It turns out that this is caused by the C and C++ corpora. These are about 20-30% longer than the next shorter ones on Ruby, Python, Java, with the shortest documents being on HTML, JavaScript and CSS. Roughly the same order holds for the entropy measures on the Stack Overflow data.

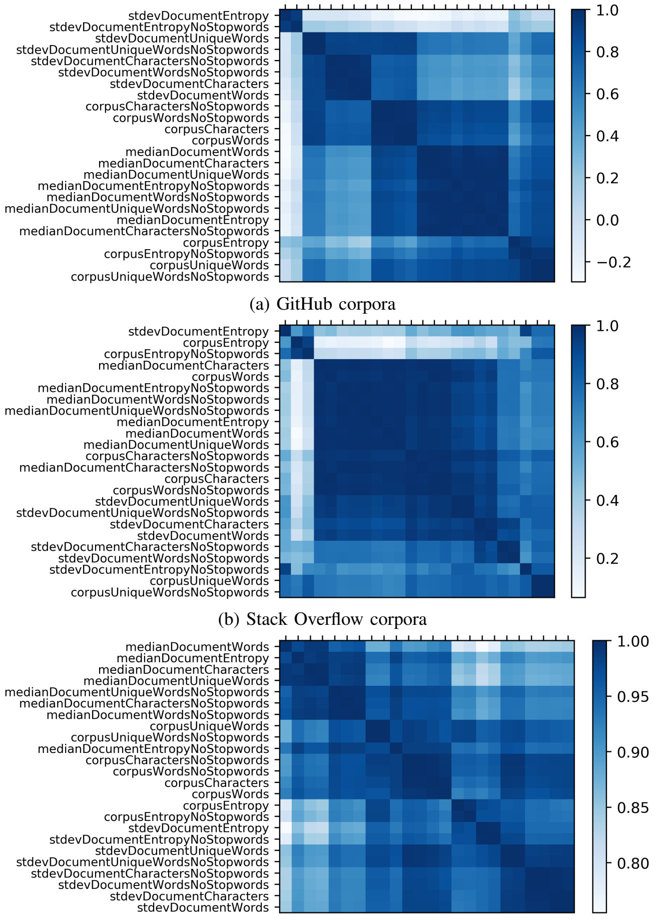
In the entropy characteristics of GitHub corpora, we note a bi-modal distribution. This time, Python joins C and C++ on the right-hand side, with all 15 corpora having a corpusEntropyNoStopwords value between 12.20 and 12.30. The closest is then a Java corpus with a value of 12.06. We speculate that software written in languages such as Python, Java, C, and C++ tends to be more complex than software written in HTML or CSS, which is reflected in the number of topics covered in the corresponding GitHub and Stack Overflow corpora measured in terms of entropy.

Lastly, we cluster the corpora in the feature space using a k-means approach. As pre-processing, we use standard scaling and a principal component analysis to two dimensions. To guess the number of clusters, we use the silhouette score on the range of 2 to 12 in the number of clusters. It turns out the individual languages per source can be told apart using this clustering almost perfectly (Figure 4), and the two sources GitHub and Stack Overflow can be distinguished perfectly—we see this as a good starting point for ad hoc per-corpus configuration of topic modelling. Even across sources, the language-specific characteristics of the documents persist and similar languages are near each other (see Figure 1). Moreover, the programming languages are in the vicinity of their spiritual ancestors and successors.

IV. Per-Corpus Offline Tuning

Many optimisation methods can be used to tune LDA parameters. As mentioned before, three works identified in a recent literature review [4] performed tuning, in particular, using genetic algorithms.

LDA is sensitive to the starting seed, and this noise can pose a challenge to many optimisation algorithms as the optimiser gets somewhat misleading feedback. Luckily, in recent years, many automated parameter optimisation methods have been developed and published as software packages. General purpose approaches include ParamILS [18], SMAC [19], GGA [20], and the iterated f-race procedure called irace [21]. The aim of these is to allow a wide range of parameters to be efficiently tested in a systematic way. For example,



(a) GitHub corpora

(b) Stack Overflow corpora

(c) Union

Fig. 2: Correlations of features for each of the sources and for the union of GitHub and Stack Overflow corpora. Darker fields correspond to a larger correlation between the features. X-labels are omitted as they follow the order (optimised to co-locate correlated features) of the y-labels.

irace's procedure begins with a large set of possible parameter configurations, and tests these on a succession of examples. As soon as there is sufficiently strong statistical evidence that a particular parameter setting is sub-optimal, then it is removed from consideration (the particular statistical test used in the f-race is the Friedman test). In practice, a large number of parameter settings will typically be eliminated after just a few iterations, making this an efficient process.

To answer our first research question *What are the optimal topic modelling configurations for textual corpora from GitHub and Stack Overflow?*, we use irace 2.3 [21].[5] We give irace a budget of 10,000 LDA runs, and we allow irace to conduct restarts if convergence is noticed. Each LDA run has a computation budget of 1,000 iterations, which is based on preliminary experiments to provide very good results almost independent of the CPU time budget. The LDA performance is measured in the perplexity (see Section II). In the final testing phase, the best configurations per corpus (as determined by irace) are run 101 times to achieve stable average performance values with a standard error of the mean of 10%. In our following analyses, we consider the median of these 101 runs.

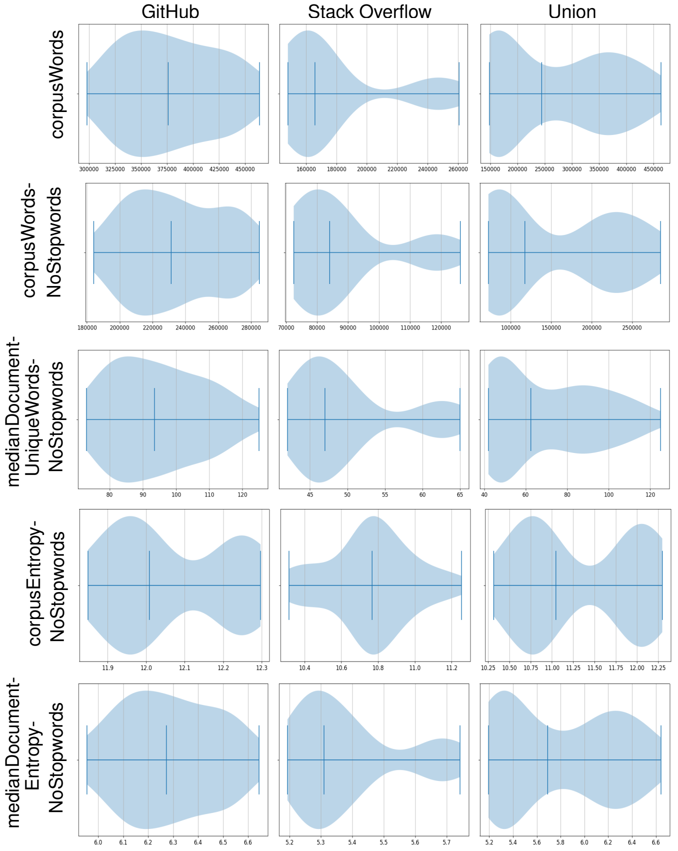


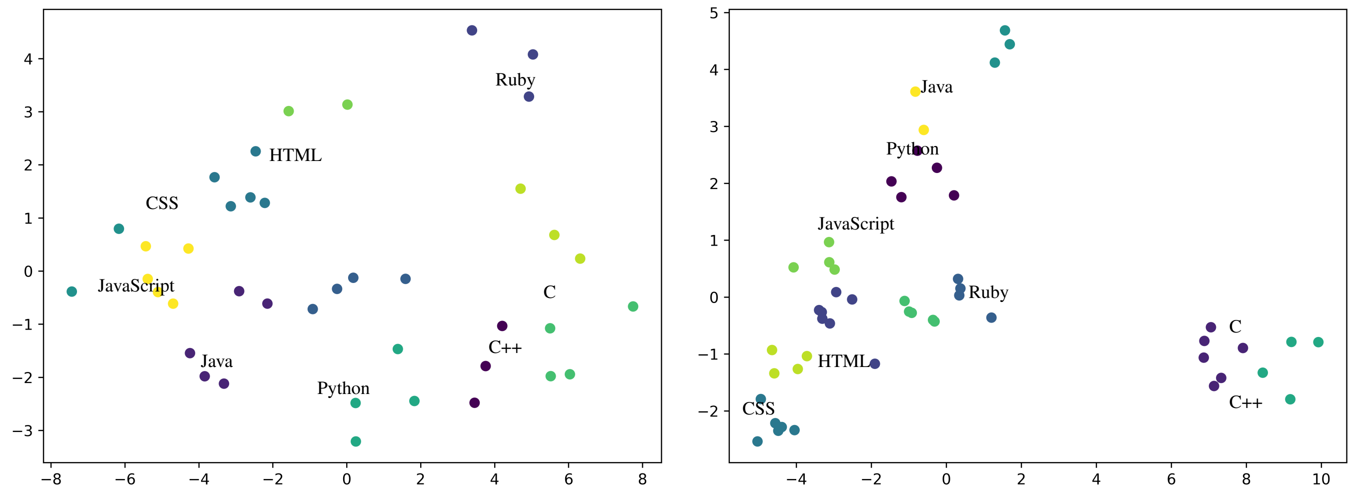
Fig. 3: Characteristics of the corpora. Top: three features based on word counts; bottom: two features based on entropy. These violin plot are an alternative to box plots, and they indicate with thickness how common values are.

Our parameter ranges are wider than what has been considered in the literature (e.g., [12]), and are informed by our preliminary experiments: number of topics $k \in [3, 1000]$, $\alpha \in [0.001, 200.0]$, $\beta \in [0.001, 200.0]$. As an initial configuration that irace can consider we provide it with $k = 100$, $\alpha = 1.0$, and $\beta = 0.01$, which are Mallet's default values.

This set of experiments is performed on a compute node with Intel(R) Xeon(R) E7-4870 CPUs with 1 TB RAM. Determining a well-performing configuration for each corpus takes 30-36 hours on a compute node with 80 cores with 80x-parallelisation. The total computation time required by the per-corpus optimisations is about 30 CPU years.

As an example, we show in Figure 5 the final output of irace when optimising the parameters for one of the five corpora related to C and taken from GitHub `CGitHub-1`. For comparison, the seeded default configuration achieves a median perplexity of 342.1. The configuration evolved to one with a large number of topics, and a very large β value. We observe that the perplexity values are very close to each to

[5] The irace Package, http://iridia.ulb.ac.be/irace, last accessed on 24 December 2018.



(a) GitHub corpora

(b) Stack Overflow corpora

Fig. 4: Clustered corpora in 2d. The colour encodes the cluster assigned to each corpus. The dimensionality reductions for both sources were conducted independent of each other, thus resulting in arrangements different from the one in Figure 1.

```
# Testing configurations:            (the first number is the configuration ID)
     topics  alpha   beta
288     628 10.474 55.835
847     550  4.958 68.056
1408    562  3.745 23.961
1558    556  3.884 21.293
1575    496 11.660 39.552
# Testing of elite configurations:   (medians of 101 independent runs)
     288   847  1408 1558  1575
   236.7 236.3 235.9  235 234.2
```

Fig. 5: irace results for the corpus `CGitHub-1`. The upper block lists the configurations returned by irace.

each other (at about 234 to 237, or 31% below Mallet's default performance) even though the configurations vary.

We show the results in Table II. It turns out that the corpora from both sources and from the eight programming languages require different parameter settings in order to achieve good perplexity values—and thus good and useful "topics". While the α values are at least (almost always) in the same order of magnitude as the seeded default configuration ($k = 100$, $\alpha = 1.0$, $\beta = 0.01$), the β values deviate significantly from it, as does the number of topics, confirming recent findings by Agrawal et al. [4].

For example, the numbers of topics addressed in the GitHub corpora is significantly higher (based on the tuned and averaged configurations for good perplexity values) than in the Stack Overflow corpora. This might be due to the nature of the README files of different software projects in contrast to potentially a more limited scope of discussions on Stack Overflow. Also, the Stack Overflow corpora appear to vary a bit more (standard deviation is 22% of the mean) than the GitHub corpora (16%).

When it comes to the different programming languages, we observe that the number of topics in Python / C / C++ is highest for the GitHub corpora, which appears to be highly correlated with the outstanding values of corpusEntropyNoStopwords of these corpora observed in Section III-C. Similarly, the corpora with lowest entropy (i.e., CSS / HTML / JavaScript) appear to require the smallest number of topics for good perplexity values.

Other interesting observations are that the β values vary more among the Stack Overflow corpora. The α values are mostly comparable across the two sources.

Summary: Popular rules of thumb for topic modelling parameter configuration are not applicable to textual corpora from GitHub and Stack Overflow. These corpora have different characteristics and require different configurations to achieve good model fit.

V. Per-Corpus Configuration

An alternative to the tuning of algorithms is that of selecting an algorithm from a portfolio or determining an algorithm configuration, when an instance is given. This typically involves the training of machine learning models on performance data of algorithms in combination with instances given as feature data. In software engineering, this has been recently used as an

TABLE II: Results of tuning (number of topics k, α, β) for eight programming languages from two sources, with the goal of minimising perplexity.

		k		α		β		perplexity	
source	langauge	mean	stdev	mean	stdev	mean	stdev	mean	stdev
GitHub	C	521.2	73.7	3.94	4.35	68.4	35.8	236.5	6.5
	C++	577.4	173.6	1.75	1.20	61.7	32.9	228.4	5.2
	CSS	455.4	34.1	1.52	0.82	36.7	16.0	236.7	7.8
	HTML	439.2	37.0	0.93	0.09	45.4	17.6	236.6	8.6
	Java	480.2	76.0	1.81	0.89	44.6	37.1	226.0	3.1
	JavaScript	484.0	19.9	1.59	0.57	23.4	18.2	238.1	2.7
	Python	529.0	43.7	1.51	0.27	32.9	14.9	257.4	10.9
	Ruby	505.4	28.0	2.41	1.49	89.1	37.0	213.9	6.0
	all	499.0	81.0	1.93	1.80	50.3	32.4	234.2	13.3
Stack Overflow	C	377.0	34.3	0.95	0.35	51.8	55.1	202.9	4.5
	C++	337.6	29.6	3.33	3.30	97.4	61.8	199.3	3.0
	CSS	196.2	24.2	1.01	0.96	18.1	15.3	184.1	2.7
	HTML	244.4	18.1	2.45	2.33	76.4	69.5	196.7	5.9
	Java	349.8	49.1	0.85	0.46	10.0	8.2	223.9	2.5
	JavaScript	252.8	34.5	4.24	3.66	50.9	44.0	213.6	2.0
	Python	295.8	47.3	1.10	0.18	67.6	78.6	229.0	4.0
	Ruby	269.3	33.1	2.11	2.72	64.0	52.4	215.9	7.3
	all	283.7	61.9	2.06	2.37	57.6	57.4	207.8	14.2
all		379.4	128.7	2.00	2.12	54.4	47.8	219.5	19.1

approach for the Software Project Scheduling Problem [22], [23]. The field of per-instance configuration has received much attention recently, and we refer the interested reader to a recent updated survey article [24]. The idea of algorithm selection is that given an instance, an algorithm selector selects a well-performing algorithm from a (often small) set of algorithms, the so-called portfolio.

To answer our second research question *Can we automatically select good configurations for unseen corpora based on their features alone?*, we study whether we can apply algorithm selection to LDA configuration to improve its performance further than with parameter tuning only. We take from each language and each source the tuned configuration of each first corpus (sorted alphabetically), and we consider our default configuration, resulting in a total of 17 configurations named gh.C, ... so.C, ... and default. As common in the area of algorithm portfolios, we treat these different configurations as different algorithms and try to predict which configuration should be used for a new given instance—"new" are now all corpora from both sources. Effectively, this will let us test which tuned corpus-configuration performs well on others. A similar approach was used by Wagner et al. to investigate the importance of instance features in the context of per-instance configuration of solvers for the minimum vertex cover problem [25], for the traveling salesperson problem [26], and for the traveling thief problem [27].

As algorithm selection is often implemented using machine learning [28], [29], we need two preparation steps: (i) instance features that characterise instances numerically, (ii) performance data of each algorithm on each instance. We have already characterised our corpora in Section III-B, so we only need to run each of the 17 configurations on all corpora.

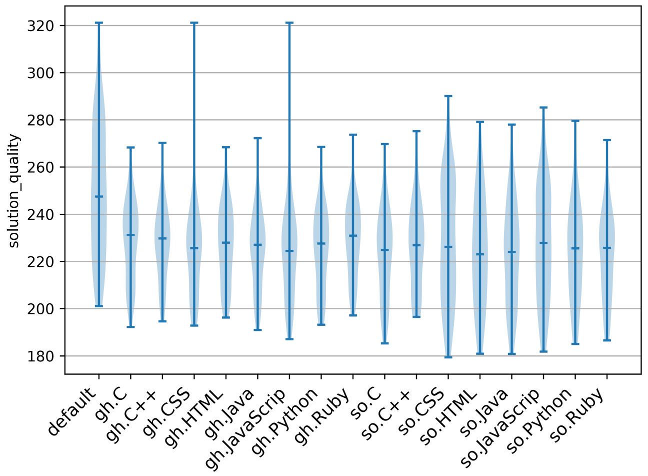


Fig. 6: Results of 17 given configurations across all corpora.

Figure 6 provides an overview of the performance of the 17 configurations when run across all corpora.[6] As we can see, a per-corpus configuration is necessary to achieve the lowest perplexity values in topic modelling (Figure 6). Many configuration corpora can be optimised (within 5%) with a large number of configurations (Figure 7, red), however, a particular cluster of Stack Overflow corpora requires specialised configurations.

The average perplexity of the 17 configurations is 227.3. The single best configuration across all data is so.Java (tuned on one of the five Stack Overflow Java corpora) with an

[6] gh.CSS and gh.JavaScript crashed on two corpora: we assigned as a result the maximum value observed (321.2).

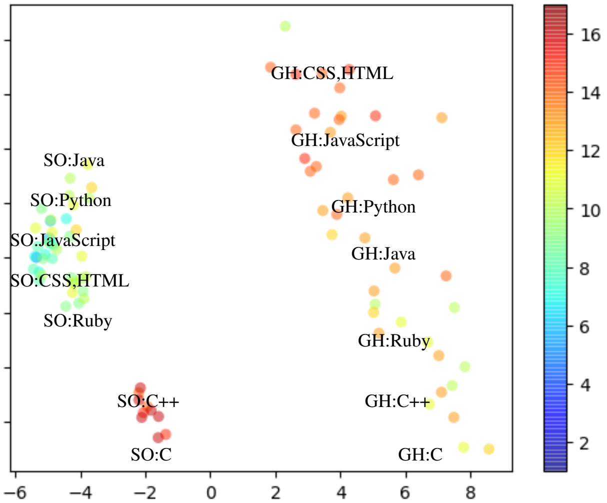


Fig. 7: Results of per-corpus configuration. Hardness: projected into 2d feature space (see Section III-C), the colour encodes the number of configurations that perform within 5% of the best performance. The arrangement of instances is identical to that in Figure 1.

average perplexity value of 222.9; the default configuration achieves an average of 250.3 (+12%).

Based on all the data we have, we can simulate the so-called virtual best solver, which would pick for each corpus the best out of the 17 configurations. This virtual best solver has an average perplexity of 217.9, which is 2% better than so.Java and 14% better than the default configuration.

Lastly, let us look into the actual configuration selection. Using the approach of SATZilla'11 [30] as implemented in AutoFolio [31], we train a cost-sensitive random forest for each pair of configurations, which then predicts for each pair of configurations the one that will perform better. The overall model then proposes the best-performing configuration. In our case, we use this approach to pick one of the 17 configurations given an instance that is described by its features. The trained model's predictions achieve an average perplexity of 219.6: this is a 4% improvement over the average of the 17 tuned configurations, and it is less than 1% away from the virtual best solver.

We are interested in the importance of features in the model—not only to learn about the domain, but also as the calculation of instance features forms an important step in the application of algorithm portfolios. The measure we use is the Gini importance [32] across all cost-sensitive random forests models, that can predict for a pair of solvers which one will perform better [30]. Figure 8 reveals that there is not a single feature, but a large set of features which together describe a corpus. It is therefore hardly possible to manually come up with good "rules of thumb" to choose the appropriate configuration depending on the corpus features—even though many of the features are correlated (see Section III-C).

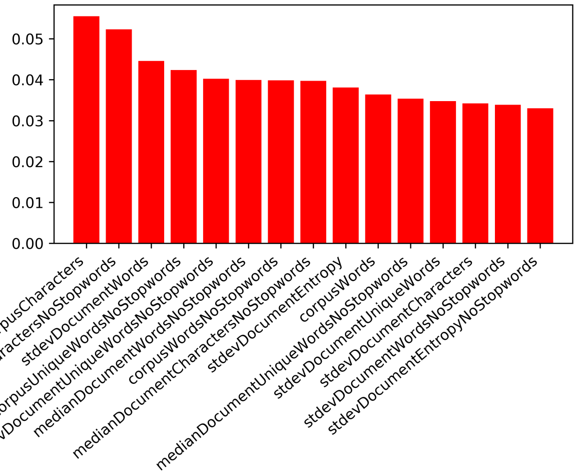


Fig. 8: Gini importance, features from Table I.

Interestingly, the expensive-to-compute entropy-based features are of little importance in the random forests (1x 9th, 1x 15th). This is good for future per-corpus configuration, as the others can be computed very quickly.

Summary: We can predict good configurations for unseen corpora reliably. Our predictions outperform the default configuration by 14%, the best tuned single configuration by 4%, and they are less than 1% away from the virtual best solver.

VI. Threats to Validity

As with all empirical studies, there are a number of threats that may impair the validity of our results.

Threats to construct validity concern the suitability of our evaluation metrics. Following many other works, we have used perplexity, the geometric mean of the inverse marginal probability of each word in a held-out set of documents [13], to measure the fit of our topic models. Perplexity is not the only metric which can be used to evaluate topic models, and a study by Chang et al. [33] found that surprisingly, perplexity and human judgement are often not correlated. Future work will have to investigate the prediction of good configurations for textual software engineering corpora using other metrics, such as conciseness or coherence. The optimal may differ depending on the objective of the topic model, e.g., whether topics are shown to end users or whether they are used as input for another machine learning algorithm. In addition, selecting different corpus features might have led to different results. We selected easy-to-compute features as well as entropy as a starting point—studying the effect of other features is part of our future work.

Threats to external validity affect the generalisability of our findings. We cannot claim that our findings generalise beyond the particular corpora which we have considered in this work. In particular, our work may not generalise beyond GitHub

README files and Stack Overflow threads, and also not beyond the particular programming languages we considered in this work. In addition, the amount of data we were able to consider in this work is necessarily limited. Choosing different documents might have resulted in different findings.

Threats to internal validity relate to errors in implementation and experiments. We have double-checked our implementation and experiments and fixed errors which we found. Still, there could be additional errors which we did not notice.

VII. Related Work

We summarise related work on the application of topic modelling to software artefacts, organised by the kind of data that topic modelling was applied to. We refer readers to Agrawal et al. [4] for an overview of the extent to which parameter tuning has been employed by software engineering researchers when creating topic models. To the best of our knowledge, we are the first to explore whether good configurations for topic models can be predicted based on corpus features.

A. Topic modelling of source code and its history

In one of the first efforts to apply topic modelling to software data, Linstead et al. [34] modelled Eclipse source code via author-topic models with the goal of mining developer competencies. They found that their topic models were useful for developer similarity analysis. Nguyen et al. [35] also applied topic modelling to source code, but for the purpose of defect prediction. The goal of their work was to measure concerns in source code, and then use these concerns as input for defect prediction. They concluded that their topic-based metrics had a high correlation with number of bugs.

With the goal of automatically mining and visualising API usage examples, Moritz et al. [36] introduced an approach called ExPort. They found that ExPort could successfully recommend complex API usage examples based on the use of topic modelling. The goal of work by Wang and Liu [37] was to establish a project overview and to bring search capability to software engineers. This work also applied topic modelling to source code, and resulted in an approach which can support program comprehension for Java software engineers.

Thomas et al. [9] focused their work on a subset of source code—test cases. The goal of their work was static test case prioritisation using topic models, and it resulted in a static black-box test case prioritisation technique which outperformed state-of-the-art techniques.

Applying topic modelling to source code history, Chen et al. [38]'s goal was to study the effect of conceptual concerns on code quality. They found that some topics were indeed more defect-prone than others. Hindle et al. [39], [40] looked at commit-log messages, aiming to automatically label the topics identified by topic modelling. They presented an approach which could produce appropriate, context-sensitive labels to support cross-project analysis of software maintenance activities. Finally, Corley et al. [41] applied topic modelling to change sets with the goal of improving existing feature location approaches, and found that their work resulted in good performance.

B. Topic modelling of bug reports and development issues

Software engineering researchers have also applied topic modelling to bug reports and development issues, to answer a wide variety of research questions. In one of the first studies in this area, Linstead and Baldi [42] found substantial promise in applying statistical text mining algorithms, such as topic modelling, for estimating bug report quality. To enable this kind of analysis, they defined an information-theoretic measure of the coherence of bug reports.

The goal of Nguyen et al. [43]'s application of topic modelling to bug reports was the detection of duplicates. They employed a combination of information retrieval and topic modelling, and found that their approach outperformed state-of-the-art approaches. In a similar research effort, Klein et al. [10]'s work also aimed at automated bug report deduplication, resulting in a significant improvement over previous work. As part of this work, the authors introduced a metric which measures the first shared topic between two topic-document distributions. Nguyen et al. [44] applied topic modelling to a set of defect records from IBM, with the goal of inferring developer expertise through defect analysis. The authors found that defect resolution time is strongly influenced by the developer and his/her expertise in a defect's topic.

Not all reports entered in a bug tracking system are necessarily bugs. Pingclasai et al. [45] developed an approach based on topic modelling which can distinguish bug reports from other requests. The authors found that their approach was able to achieve a good performance. Zibran [46] also found topic modelling to be a promising approach for bug report classification. His work explored the automated classification of bug reports into a predefined set of categories.

Naguib et al. [47] applied topic modelling to bug reports in order to automatically issue recommendations as to who a bug report should be assigned to. Their work was based on activity profiles and resulted in a good average hit ratio.

In an effort to automatically determine the emotional state of a project and thus improve emotional awareness in a software development team, Guzman and Bruegge [48] applied topic modelling to textual content from mailing lists and Confluence artefacts. They found that their proposed emotion summaries had a high correlation with the emotional state of a project.

Layman et al. [49] applied topic modelling to NASA space system problem reports, with the goal of extracting trends in testing and operational failures. They were able to identify common issues during different phases of a project. They also reported that the process of selecting the topic modelling parameters lacks definitive guidance and that defining semantically-meaningful topic labels requires non-trivial effort and domain expertise.

Focusing on security issues posted in GitHub repositories, Zahedi et al. [50] applied topic modelling to identify and understand common security issues. They found that the majority of security issues reported in GitHub issues was related to identity management and cryptography.

C. Topic modelling of Stack Overflow content

Linares-Vásquez et al. [51] conducted an exploratory analysis of mobile development issues, with the goal of extracting hot topics from Stack Overflow questions related to mobile development. They found that most questions included topics related to general concerns and compatibility issues. In a similar more recent effort, Rosen and Shihab [8] set out to identify what mobile developers are asking about on Stack Overflow. They identified various frequently discussed topics, such as app distribution, mobile APIs, and data management.

Looking beyond the scope of mobile development, Barua et al. [7] contributed an analysis of topics and trends on Stack Overflow. They found that topics of interest ranged widely from jobs to version control systems and C# syntax. Zou et al. [52] applied topic modelling to Stack Overflow data with a similar goal, i.e., to understand developer needs. Among other findings, they reported that the most frequent topics were related to usability and reliability.

Allamanis and Sutton [53]'s goal was the identification of programming concepts which are most confusing, based on an analysis of Stack Overflow questions by topic, type, and code. Based on their work, they were able to associate programming concepts and identifiers with particular types of questions. Aiming at the identification of API usage obstacles, Wang and Godfrey [54] studied questions posted by iOS and Android developers on Stack Overflow. Their topic modelling analysis revealed several iOS and Android API classes which appeared to be particularly likely to challenge developers.

Campbell et al. [55] applied topic modelling to content from Stack Overflow as well as project documentation, with the goal of identifying topics inadequately covered by project documentation. They were able to successfully detect such deficient documentation using topic analysis. As part of the development of a recommender system, Wang et al. [56] set out to recommend Stack Overflow posts to users which are likely to concern API design-related issues. Their topic modelling approach was able to achieve high accuracy.

D. Topic modelling of other software artefacts

Source code, bug reports, and Stack Overflow are not the only sources which researchers have applied topic modelling to. Other sources include usage logs, user feedback, service descriptions, and research papers. We briefly highlight related papers in this subsection.

Bajracharya and Lopes [57], [58]'s goal was to understand what users search for. To achieve this, they mined search topics from the usage log of the code search engine Koders. They concluded that code search engines provide only a subset of the various information needs of users.

Aiming at the extraction of new or changed requirements for new versions of a software product, Galvis Carreño and Winbladh [59] applied topic modelling to user feedback captured in user comments. Their automatically extracted topics matched the ones that were manually extracted.

Nabli et al. [60] applied topic modelling to cloud service descriptions with the goal of making it more efficient to discover relevant cloud services. They were able to improve the effectiveness of existing approaches.

In one of the first papers to report the application of topic modelling to software engineering data, Asuncion et al. [61] applied topic modelling to a variety of heterogeneous software artefacts, with the goal of improving traceability. They implemented several tools based on their work, and concluded that topic modelling indeed enhances software traceability.

Finally, Sharma et al. [62] applied topic modelling to abstracts of research papers published in the Requirements Engineering (RE) conference series. Their work resulted in the identification of the structure and composition of requirements engineering research.

VIII. Conclusions and Future Work

Topic modelling is an automated technique to make sense of large amounts of textual data. To understand the impact of parameter tuning on the application of topic modelling to software development corpora, we employed techniques from Data-Driven Software Engineering [5] to 40 corpora sampled from GitHub and 40 corpora sampled from Stack Overflow, each consisting of 1,000 documents. We found that (1) popular rules of thumb for topic modelling parameter configuration are not applicable to the corpora used in our experiments, (2) corpora sampled from GitHub and Stack Overflow have different characteristics and require different configurations to achieve good model fit, and (3) we can predict good configurations for unseen corpora reliably.

These findings play an important role in efficiently determining suitable configurations for topic modelling. State-of-the-art approaches determine the best configuration separately for each corpus, while our work shows that corpus features can be used for the prediction of good configurations. Our work demonstrates that source and context (e.g., programming language) matter in the textual data extracted from software repositories. Corpora related to the same programming language naturally form clusters, and even content from related programming languages (e.g., C and C++) are part of the same clusters. This finding opens up interesting avenues for future work: after excluding source code, why is the textual content that software developers write about the same programming language still more similar than textual content written about another programming language? In addition to investigating this, in our future work, we will expand our exploration of the relationship between features and good configurations for topic modelling, using larger and more diverse corpora as well as additional features and a longitudinal approach [63]. We will also make our approach available to end users through tool support and conduct qualitative research to determine to what extent the discovered topics make sense to humans.

Acknowledgments. Our work was supported by the Australian Research Council projects DE180100153 and DE160100850. We acknowledge the support by the HPI Future SOC Lab, who granted us access to their computing resources.

References

[1] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts," in *Proc. of the Int'l. Conf. on Mining Software Repositories*, 2018, pp. 319–330.

[2] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *Proc. of the Conf. on Computer Supported Cooperative Work*, 2012, pp. 1277–1286.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[4] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? and how to fix it using search-based software engineering," *Information and Software Technology*, vol. 98, pp. 74–88, 2018.

[5] V. Nair, A. Agrawal, J. Chen, W. Fu, G. Mathew, T. Menzies, L. Minku, M. Wagner, and Z. Yu, "Data-driven search-based software engineering," in *Proc. of the Int'l. Conf. on Mining Software Repositories*, 2018, pp. 341–352.

[6] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu, "Better software analytics via "duo": Data mining algorithms using/used-by optimizers," *CoRR*, vol. abs/1812.01550, 2018.

[7] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[8] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[9] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Software Engineering*, vol. 19, no. 1, pp. 182–212, 2014.

[10] N. Klein, C. S. Corley, and N. A. Kraft, "New features for duplicate bug detection," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2014, pp. 324–327.

[11] P. Luangaram and W. Wongwachara, "More Than Words: A Textual Analysis of Monetary Policy Communication," Puey Ungphakorn Institute for Economic Research, PIER Discussion Papers 54, Feb. 2017.

[12] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proc. of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[13] M. Hoffman, F. R. Bach, and D. M. Blei, "Online learning for latent dirichlet allocation," in *Advances in neural information processing systems*, 2010, pp. 856–864.

[14] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of GitHub README files," *Empirical Software Engineering*, 2019.

[15] G. Koutrika, L. Liu, and S. Simske, "Generating reading orders over document collections," in *Proc. of the Int'l. Conf. on Data Engineering*, 2015, pp. 507–518.

[16] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[17] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Frechétte, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, "Aslib: A benchmark library for algorithm selection," *Artificial Intelligence Journal*, vol. 237, pp. 41–58, 2016.

[18] F. Hutter, H. H. Hoos, and T. Stützle, "Automatic algorithm configuration based on local search," in *Proc. of the National Conf. on Artificial Intelligence*, 2007, pp. 1152–1157.

[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. of the Int'l. Conf. on Learning and Intelligent Optimization*, 2011, pp. 507–523.

[20] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Proc. of the Int'l. Conf. on Principles and Practice of Constraint Programming*, 2009, pp. 142–157.

[21] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proc. of the Genetic and Evolutionary Computation Conf.*, 2002, pp. 11–18.

[22] X.-N. Shen, L. L. Minku, N. Marturi, Y.-N. Guo, and Y. Han, "A q-learning-based memetic algorithm for multi-objective dynamic software project scheduling," *Information Sciences*, vol. 428, pp. 1–29, 2018.

[23] X. Wu, P. Consoli, L. Minku, G. Ochoa, and X. Yao, "An evolutionary hyper-heuristic for the software project scheduling problem," in *Proc. of the Parallel Problem Solving from Nature*, 2016, pp. 37–47.

[24] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," in *Data Mining and Constraint Programming*. Springer, 2016, pp. 149–190.

[25] M. Wagner, T. Friedrich, and M. Lindauer, "Improving local search in a minimum vertex cover solver for classes of networks," in *Proc. of the Congress on Evolutionary Computation*, 2017, pp. 1704–1711.

[26] S. Nallaperuma, M. Wagner, and F. Neumann, "Analyzing the effects of instance features and algorithm parameters for maxmin ant system and the traveling salesperson problem," *Frontiers in Robotics and AI*, vol. 2, p. 18, 2015.

[27] M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter, "A case study of algorithm selection for the traveling thief problem," *Journal of Heuristics*, vol. 24, no. 3, pp. 295–320, 2018.

[28] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys*, vol. 41, no. 1, pp. 6:1–6:25, 2009.

[29] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary Computation*, vol. 27, no. 1, pp. 3–45, 2019, pMID: 30475672.

[30] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, "Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming," in *Proc. of the RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the Int'l. Joint Conf. on Artificial Intelligence (IJCAI)*, 2011.

[31] M. Lindauer, H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *Artificial Intelligence Research*, vol. 53, pp. 745–778, 2015.

[32] L. Breimann, "Random forests," *Machine Learning Journal*, vol. 45, pp. 5–32, 2001.

[33] J. Chang, J. Boyd-Graber, S. Gerrish, C. Wang, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Proc. of the Int'l. Conf. on Neural Information Processing Systems*, 2009, pp. 288–296.

[34] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining eclipse developer contributions via author-topic models," in *Proc. of the Int'l. Workshop on Mining Software Repositories*, 2007, pp. 30–33.

[35] T. T. Nguyen, T. N. Nguyen, and T. M. Phuong, "Topic-based defect prediction (NIER Track)," in *Proc. of the Int'l. Conf. on Software Engineering*, 2011, pp. 932–935.

[36] E. Moritz, M. Linares-Vásquez, D. Poshyvanyk, M. Grechanik, C. McMillan, and M. Gethers, "Export: Detecting and visualizing api usages in large source code repositories," in *Proc. of the Int'l. Conf. on Automated Software Engineering*, 2013, pp. 646–651.

[37] T. Wang and Y. Liu, "Infusing topic modeling into interactive program comprehension: An empirical study," in *Annual Computer Software and Applications Conference*, vol. 2, 2017, pp. 260–261.

[38] T.-H. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan, "Explaining software defects using topic models," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2012, pp. 189–198.

[39] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos, "Automated topic naming to support cross-project analysis of software maintenance activities," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2011, pp. 163–172.

[40] ——, "Automated topic naming," *Empirical Software Engineering*, vol. 18, no. 6, pp. 1125–1155, 2013.

[41] C. S. Corley, K. Damevski, and N. A. Kraft, "Changeset-based topic modeling of software repositories," *IEEE Transactions on Software Engineering*, 2019.

[42] E. Linstead and P. Baldi, "Mining the coherence of gnome bug reports with statistical topic models," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2009, pp. 99–102.

[43] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proc. of the Int'l. Conf. on Automated Software Engineering*, 2012, pp. 70–79.

[44] T. T. Nguyen, T. N. Nguyen, E. Duesterwald, T. Klinger, and P. Santhanam, "Inferring developer expertise through defect analysis," in *Proc. of the Int'l. Conf. on Software Engineering*, 2012, pp. 1297–1300.

[45] N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *Proc. of the Asia-Pacific Software Engineering Conference - Volume 02*, 2013, pp. 13–18.

[46] M. F. Zibran, "On the effectiveness of labeled latent dirichlet allocation in automatic bug-report categorization," in *Proc. of the Int'l. Conf. on Software Engineering Companion*, 2016, pp. 713–715.

[47] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2013, pp. 22–30.

[48] E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proc. of the Joint Meeting on Foundations of Software Engineering*, 2013, pp. 671–674.

[49] L. Layman, A. P. Nikora, J. Meek, and T. Menzies, "Topic modeling of NASA space system problem reports: Research in practice," in *Proc. of the Int'l. Conf. on Mining Software Repositories*, 2016, pp. 303–314.

[50] M. Zahedi, M. A. Babar, and C. Treude, "An empirical study of security issues posted in open source projects," in *Proc. of the Hawaii Int'l. Conf. on System Sciences*, 2018, pp. 5504–5513.

[51] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk, "An exploratory analysis of mobile development issues using stack overflow," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2013, pp. 93–96.

[52] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang, "An empirical study on stack overflow using topic analysis," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2015, pp. 446–449.

[53] M. Allamanis and C. Sutton, "Why, when, and what: Analyzing stack overflow questions by topic, type, and code," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2013, pp. 53–56.

[54] W. Wang and M. W. Godfrey, "Detecting api usage obstacles: A study of ios and android developer questions," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2013, pp. 61–64.

[55] J. C. Campbell, C. Zhang, Z. Xu, A. Hindle, and J. Miller, "Deficient documentation detection: A methodology to locate deficient project documentation using topic analysis," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2013, pp. 57–60.

[56] W. Wang, H. Malik, and M. W. Godfrey, "Recommending posts concerning api issues in developer q&a sites," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2015, pp. 224–234.

[57] S. Bajracharya and C. Lopes, "Mining search topics from a code search engine usage log," in *Proc. of the Int'l. Working Conf. on Mining Software Repositories*, 2009, pp. 111–120.

[58] S. K. Bajracharya and C. V. Lopes, "Analyzing and mining a code search engine usage log," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 424–466, 2012.

[59] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proc. of the Int'l. Conf. on Software Engineering*, 2013, pp. 582–591.

[60] H. Nabli, R. B. Djemaa, and I. A. B. Amor, "Efficient cloud service discovery approach based on lda topic modeling," *Journal of Systems and Software*, vol. 146, pp. 233–248, 2018.

[61] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proc. of the Int'l. Conf. on Software Engineering - Volume 1*, 2010, pp. 95–104.

[62] R. Sharma, P. Aggarwal, and A. Sureka, "Insights from mining eleven years of scholarly paper publications in requirements engineering (re) series of conferences," *SIGSOFT Software Engineering Notes*, vol. 41, no. 2, pp. 1–6, 2016.

[63] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, 2018.