What is Wrong with Topic Modeling? (and How to Fix it Using Search-based SE)

Amritanshu Agrawal
Computer Science, NCSU
Raleigh, North Carolina
aagrawa8@ncsu.edu

Tim Menzies
Computer Science, NCSU
Raleigh, North Carolina
tim.menzies@gmail.com

Wei Fu
Computer Science, NCSU
Raleigh, North Carolina
wfu@ncsu.edu

***Abstract*—Topic Modeling finds human-readable structures in large sets of unstructured SE data. A widely used topic modeler is Latent Dirichlet Allocation.**

When run on SE data, LDA suffers from "order effects" i.e. different topics be generated if the training data was shuffled into a different order. Such order effects introduce a systematic error for any study that uses topics to make conclusions.

This paper introduces LDADE, a Search-Based SE tool that tunes LDA's parameters using DE (Differential Evolution). LDADE has been tested on data from a programmer information exchange site (Stackoverflow), title and abstract text of thousands of SE papers, and software defect reports from NASA. Results were collected across different implementations of LDA (Python+Scikit-Learn, Scala+Spark); across different platforms (Linux, Macintosh) and for different kinds of LDAs (the traditional VEM method, or using Gibbs sampling). In all tests, the pattern was the same: LDADE's tunings dramatically reduces topic instability.

The implications of this study for other software analytics tasks is now an open and pressing issue. In how many domains can search-based SE dramatically improve software analytics?

***Keywords*—Topic modeling, Stability, LDA, tuning, differential evolution.**

I. Introduction

The current great challenge in software analytics is understanding unstructured data. As shown in Figure 1, most of the planet's 1600 Exabytes of data does not appear in structured sources (databases, etc) [**?**]. Rather it is *unstrucutred* data, often in free text, and found in word processing files, slide presentations, comments, etc etc.

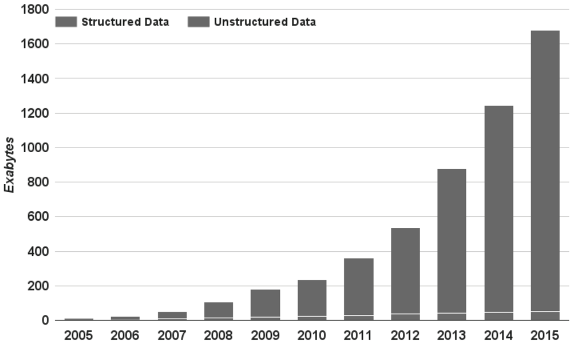


Figure 1: Data Growth 2005-2015. From [**?**].

Such unstructured data does not have a pre-defined data model and is typically text-heavy. Finding insights among unstructured text is difficult unless we can search, characterize, and classify the textual data in a meaningful way. One of the common techniques for finding related topics within unstructured text (an area called topic modeling) is Latent Dirichlet Allocation (LDA) [12]. Topic modeling is widely used in SE (see Table I) and many papers in recent years have reported topic modeling results at numerous SE venues (see Figure 2 and Table II).

Researchers can use topic models in one of two ways. Firstly, topics may be used as *feature selector* that finds useful inputs which are then used by, say, a classifier to characterize different kinds of software (e.g. buggy or not [**?**]). In this mode, no human need ever read the generated topics and instabilities in generated topics is less of an issue.

Secondly, researchers may present and reflect on the generated topics in order to offer some insight into the structure of the data. In this second mode, researchers present and discuss the specifics of the generated topics in order to defend particular conclusions.

This paper is about a systematic error relating to this second mode. Specifically, we show that the generated topics from Latent Dirichlet Allocation (LDA), a widely-used topic modeling algorithm in SE, are subject to "order effects". That is, if the order of the input examples is changed, the generated topics will also change– often dramatically so. Hence, any conclusion based on an order effect is unstable since that conclusion is the result of a (randomly selected) input ordering.

To fix this problem, we proposes LDADE: a combination of LDA and a search-based optimizer (differential evolution, or DE) [62]) that automatically tunes LDA's $< k, \alpha, \beta >$ parameters. This paper tests LDADE on:

- Data from a programmer information exchange site (Stackoverflow);
- Title and abstract text of 9291 SE papers;
- Software defect reports from NASA.

Using these datasets, we explore these research questions:

- **RQ1**: **Are the default settings of LDA incorrect?** We will show that using the default settings of LDA for SE data can lead to systematic errors since stability scores start to drop after $n = 5$ terms per topic.

Reference ID	Year	Citations	Venues	Mentions instability in LDA?	Talks about using-off-the shell parameters?	Does tuning?	Conclusion	Tasks / Use cases
[55]	2011	112	WCRE	Y	Y	N	Explored Configurations without any explanation.	Bug Localisation
[50]	2010	108	MSR	Y	Y	N	Explored Configurations without any explanation. Reported their results using multiple experiments.	Traceability Link recovery
[?]	2014	96	ESE	Y	Y	N	Explored Configurations without any explanation. Choosing right set of parameters is a difficult task.	StackOverflow Q&A data analysis
[51]	2013	75	ICSE	Y	Y	Y	Uses GA to tune parameters. They determine the near-optimal configuration for LDA in the context of only some important SE tasks.	Finding near-optimal configurations
[22]	2013	61	ICSE	Y	Y	N	Explored Configurations without any explanation.	Software Requirements Analysis
[?]	2011	45	MSR	Y	Y	N	They validated the topic labelling techniques using multiple experiments.	Software Artifacts Analysis
[30]	2014	44	RE	Y	Y	N	Explored Configurations without any explanation.	Requirements Engineering
[?]	2011	44	ICSE	Y	Y	N	Open issue to choose optimal parameters.	A review on LDA
[?]	2012	35	MSR	Y	Y	N	Choosing the optimal number of topics is difficult.	Software Defects Prediction
[67]	2014	35	SCP	Y	Y	N	Explored Configurations without any explanation.	Software Artifacts Analysis
[68]	2014	31	ESE	Y	Y	N	Choosing right set of parameters is a difficult task.	Software Testing
[?]	2009	29	MSR	Y	Y	N	Explored Configurations without any explanation and accepted to the fact their results were better because of the corpus they used.	Software History Comprehenion
[42]	2013	27	ESEC/FSE	Y	Y	Y	Explored Configurations using LDA-GA.	Traceability Link recovery
[?]	2013	20	MSR	Y	Y	N	In Future, they planned to use LDA-GA.	StackOverflow Q&A data analysis
[?]	2014	15	WebSci	Y	Y	N	Explored Configurations without any explanation.	Social Software Engineering
[?]	2012	13	ICSM	Y	Y	N	Explored Configurations without any explanation (Just with no. of topics).	Software Requirements Analysis
[?]	2013	13	SCP	Y	Y	N	Their work focused on optimizing LDA's topic count parameter.	Source Code Comprehension
[?]	2015	6	IST	Y	Y	N	Explored Configurations without any explanation. Choosing right set of parameters is a difficult task.	Software re-factoring
[24]	2016	5	CS Review	Y	Y	N	Explored Configurations without any explanation.	Bibiliometric and citation analysis
[?]	2014	5	ISSRE	N	Y	N	Explored Configurations without any explanation.	Bug Localisation
[49]	2015	3	JIS	Y	Y	N	They improvised LDA into ISLDA which gave stability across different runs.	Social Software Engineering
[63]	2015	2	IST	Y	Y	Y	Explored Configurations using LDA-GA.	Software Artifacts Analysis
[?]	2016	0	JSS	N	Y	N	Explored Configurations without any explanation. Choosing right set of parameters is a difficult task.	Software Defects Prediction

Table I: A sample of the recent literature on using topic modeling in SE. Note that some of these papers are widely-cited.

Venue	Full Name	Count
ICSE	International Conference on Software Engineering	4
CSMR-WCRE / SANER	International Conference on Software Maintenance, Reengineering, and Reverse Engineering / International Conference on Software Analysis,Evolution, and Reengineering	3
ICSM / ICSME	International Conference on Software Maintenance / International Conference on Software Maintenance and Evolution	3
ICPC	International Conference on Program Comprehension	3
ASE	International Conference on Automated Software Engineering	3
ISSRE	International Symposium on Software Reliability Engineering	2
MSR	International Working Conference on Mining Software Repositories	8
OOPSLA	International Conference on Object-Oriented Programming, Systems, Languages, and Applications	1
FSE/ESEC	International Symposium on the Foundations of Software Engineering / European Software Engineering Conference	1
TSE	IEEE Transaction on Software Engineering	1
IST	Information and Software Technology	3
SCP	Science of Computer Programming	2
ESE	Empirical Software Engineering	4

Table II: SE Venues that publish on Topic Modeling.

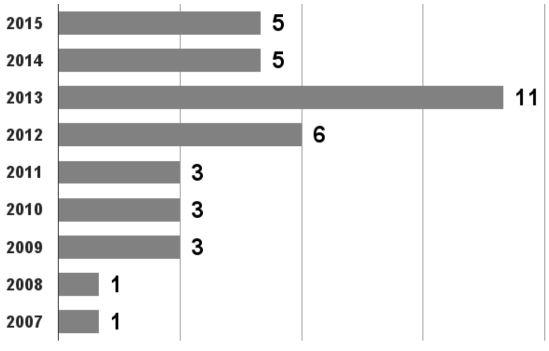


Figure 2: Topic modeling in SE: recent papers since 2007.

- **RQ2**: **Does LDADE improve the stability scores?** In our work, we will show dramatic improvement in the stability scores with the parameters found automatically by DE. We would strongly recommend tuning for future LDA studies.
- **RQ3**: **Do different data sets need different configurations to make LDA stable?** We will show that DE finds different "best" parameter settings for different data sets. Hence reusing tunings suggested by other researchers from other data sets. Instead, use automatic tuning methods to find the best tuning parameters for the current data set.
- **RQ4**: **Is tuning easy?** We show that, measured in the terms of the internal search space of the optimizer, tuning LDA is much simpler than standard optimization methods.
- **RQ5**: **Is tuning slow who are making conclusions out of these topics?** Tuning with DE makes LDA around five times slower- which is not an alarming speed reduction given modern cloud computing environments. Tuning LDA is not slow for those who are making conclusions out of these topics.
- **RQ6**: **Should topic modeling be used "off-the-shelf" with their default tunings?** Based on these findings, our answer to this question is an emphatic "no". We can see little reason to use "off-the-shelf" LDA by those who are making conclusions out of these topics.

Before proceeding, we offer three caveats. Firstly, having mitigated for unstable topics in LDA, the reader may be asking "ok, but what useful conclusions can you draw from this new kind of stabler LDA?". In a companion paper submitted to ICSE'17 [**?**] we apply LDADE to 9291 SE papers to find trends in topics between 1993 to 2013. That paper reports (a) what topics that are becoming more/less prominent; (b) what topics are most frequently accepted to what conferences; (c) which conferences mostly overlap with other conferences;

(d) what topics most distinguish different conferences. Finally, that study can assist in teaching via (e) a list of the 21 most cited papers within the seven top topics of SE.

Secondly, as witnessed by the central columns of Table I, many prior papers have commented that the results of a topic modeling analysis can be effected by tuning the control parameters of LDA. Yet a repeated pattern in the literature is that, despite these stated concerns, researchers rarely take the next step to find ways to better control tunings. To the best of our knowledge, apart from this paper, there are no reports in the SE literature of simple reproducible automatic methods that can resolve the problem of unstable topics.

Thirdly, while the specifics of this paper are about taming instability in topic modeling, we also think that this work makes a more general point. This is the second time we have applied DE to automatically adjust the control parameters of an algorithm used in software analytics. Previously [?], we showed that (a) tuning software defect predictors can lead to large improvements of the performance of those predictors; and that (b) those tunings are different in different data sets; so (c) it is important to apply automatic tuning as part of the analysis of any new data set. Note that those old conclusions are the same as for this paper. These two results suggests that it is time to revisit past results to check:

- In the past, just how poorly have we been using software analytics in the past?
- And how much better can we improve future results using methods taken from search-based software engineering?

II. Related Work

A. About Tuning: Important and Ignored

The impact of tuning are well understood in the theoretical machine learning literature [?]. When we tune a data miner, what we are really doing is changing how a learner applies its heuristics. This means tuned data miners use different heuristics, which means they ignore different possible models, which means they return different models; i.e. *how* we learn changes *what* we learn.

Yet issues relating to tuning are poorly addressed in the software analytics literature. Wei et al. [?] surveyed hundreds of recent SE papers in the area of software defect prediction from static code attributes. They found that most SE authors do not take steps to explore tunings (rare exception: [?]). For example, Elish et al. [?] compared support vector machines to other data miners for the purposes of defect prediction. That paper tested different "off-the-shelf" data miners on the same data set, without adjusting the parameters of each individual learner. Yet Wei et al. showed that (a) finding useful tunings for software defect prediction is remarkably easy using DE; (b) different data sets require different tunings; hence, for software defect prediction, (c) tuning is essential for all new data sets.

Accordingly, we are now engaged in an on-going research project that explores tuning for software analytics.

- Find some data mining widely used in SE;
- Check the conclusions of that data miner under different parameter tunings;
- See if those different tunings significantly change the results of that learner;
- Look for ways to better manage the exploration of those tunings.

Topic: String Manipulation	Topic: Function	Topic: OO Programming	Topic: UI Development	Topic: File Operation
string	function	class	control	file
charact	paramet	method	view	directori
encod	pass	object	event	path
format	return	call	button	folder
convert	argument	interfac	click	creat

Figure 3: Example (from [50]) of generating topics from Stackoverflow. For each topic, we show just the five most heavily weighted words.

This paper explores these four steps in the context of topic modeling.

B. About Topic Modeling

Latent Dirichlet Allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. It learns the various distributions (the set of topics, their associated word probabilities, the topic of each word, and the particular topic mixture of each document). What makes topic modeling interesting is that these algorithms scale to very large text corpuses. For example, in this paper, we apply to whole of Stackoverflow, as well as to two other large text corpuses in SE.

Figure 3 illustrates topic generation from Stackoverflow. To find these words, LDA explores two probability distributions:

- $\alpha = P(k|d)$; probability of topic k in document d.
- $\beta = P(w|k)$; probability of word w in topic k;

Initially, α, β may be set randomly as follows: each word in a document was generated by first randomly picking a topic (from the document's distribution of topics) and then randomly picking a word (from the topic's distribution of words). Successive iterations of the algorithm count the implications of prior sampling which, in turn, incrementally updates α, β.

Apart from α, β, the other parameters that define LDA are:

- k = number of topics.

C. About Order Effects

This paper uses tuning to fix "order effects" in topic modeling. Langley [?] defines such effects as follows:

> *A learner L exhibits an order effect on a training set T if there exist two or more orders of T for which L produces different knowledge structures.*

Many learners exhibit order effects: e.g. certain incremental clustering algorithms generate different clusters, depending on the order with which they explore the data [?]. Hence, some algorithms survey the space of possible models across numerous random divisions of the data (e.g. Random Forests [?]).

From the description offered above in §II-B, we can see (a) how topic modeling might be susceptible to order effects and (b) how such order effects might be tamed:

- In the above description, k, α, β are initialized at random then updated via an incremental re-sampling process. Such incremental updates are prone to order effects.
- One technique to reduce the effect of different data orderings is to initialize, k, α, β to some large value. The trick when applying this technique is that different data sets will requiring different initializations; i.e. the tuning process will have to be repeated for each new data set.

D. WorkArounds for LDA Instability Problem

In order to understand how other researchers have explored LDA instability, in April 2016, we searched scholar.google.com for the conjunction of "lda" and "topics" or "stable" or "unstable" or "coherence". Since 2012, there are 189 such papers, 57 of which are related to software engineering results. Of those papers:

- 28 mention instability in LDA.
- Of those 28, despite mentioning stability problems, 10 papers used LDA's "off-the-shelf" parameters;
- The other papers used some combination of manual adjustment or some under-explained limited exploration of tunings based on "engineering judgment" (i.e. some settings guided by the insights of the researchers).
- Only 3 of the authors acknowledge that tuning might have a large impact on the results.

Apart from tuning, there are several other workarounds explored in the literature in order to handle LDA instability. Overall, there was little systematic exploration of tuning and LDA in the SE literature. Instead, researchers relied on other methods that are less suited to automatic reproduction of prior results.

In the literature, researchers [**?**], [**?**], [30] manually accessed the topics and then used for further experiments. Some made use of Amazon Mechanical Turk to create gold-standard coherence judgements [**?**]. All these solutions are related to results stability rather than model stability. Note that this workaround takes extensive manual effort and time.

Another approach to taming LDA instability is to incorporate user knowledge into the corpus. For example, SC-LDA [**?**] can handle different kinds of knowledge such as word correlation, document correlation, document label and so on. Using such user knowledge, while certainly valuable, is somewhat subjective. Hence, for reasons of reproducability, we prefer fully automated methods.

Some researchers [42], [51], [63] used Genetic Algorithm (GA) to tune the parameters– but for different purposes than LDADE. Annibale et al. [51] used GAs to increase the precision of predictions made by a classifier; i.e. they are exploring *supervised classification* while the task explored in this paper is *unsupervised clustering*.

Finally, other researchers explore some limited manual parameter tuning for LDA (e.g. experiment with one parameter: cluster size) [22], [71] achieved higher stability by just increasing the number of cluster size. Note that the automatic tuning methods explored by this paper can explore multiple parameters. Further, our analysis is repeatable.

	Size	
	Before	**After**
Data set	**Preprocessing**	**Preprocessing**
PitsA	1.2 MB	292 KB
PitsB	704 KB	188 KB
PitsC	143 KB	37 KB
PitsD	107 KB	26 KB
PitsE	650 KB	216 KB
PitsF	549 KB	217 KB
Ciemap	8.6 MB	3.7 MB
StackOverflow	7 GB	527 MB

Table III: Statistics on our datasets. PitsA, PitsB, etc refer to the issues from six different NASA projects.

III. Methods

This section describes our evaluation methods for measuring instability as well as the optimization methods used to reduce that instability.

A. Data Sets

To answer our research questions, and to enable reproducibility of our results, we use three open source datasets summarized in Table III and described below.

PITS is a text mining data set generated from NASA software project and issue tracking system (PITS) reports [**?**], [**?**]. This text discusses bugs and changes found in big reports and review patches. Such issues are used to manage quality assurance, to support communication between developers. Topic modeling in PITS can be used to identify the top topics which can identify each severity separately. The dataset can be downloaded from the PROMISE repository [**?**]. Note that, this data comes from six different NASA projects, which we label as PitsA, PitsB, etc.

Stackoverflow is the flagship site of the Stack Exchange Network which features questions and answers on a wide range of topics in computer programming. Topic modeling on Stackoverflow is useful for finding patterns in programmer knowledge. This data can be downloaded from various sources[1].

Citemap contains titles and abstracts of 9291 papers from a database of 11 senior software engineering conferences from 1993-2013. This data was obtained in the form of an SQL dump from the work of Vasilescu et al. [74]. This dataset is available on-line[2].

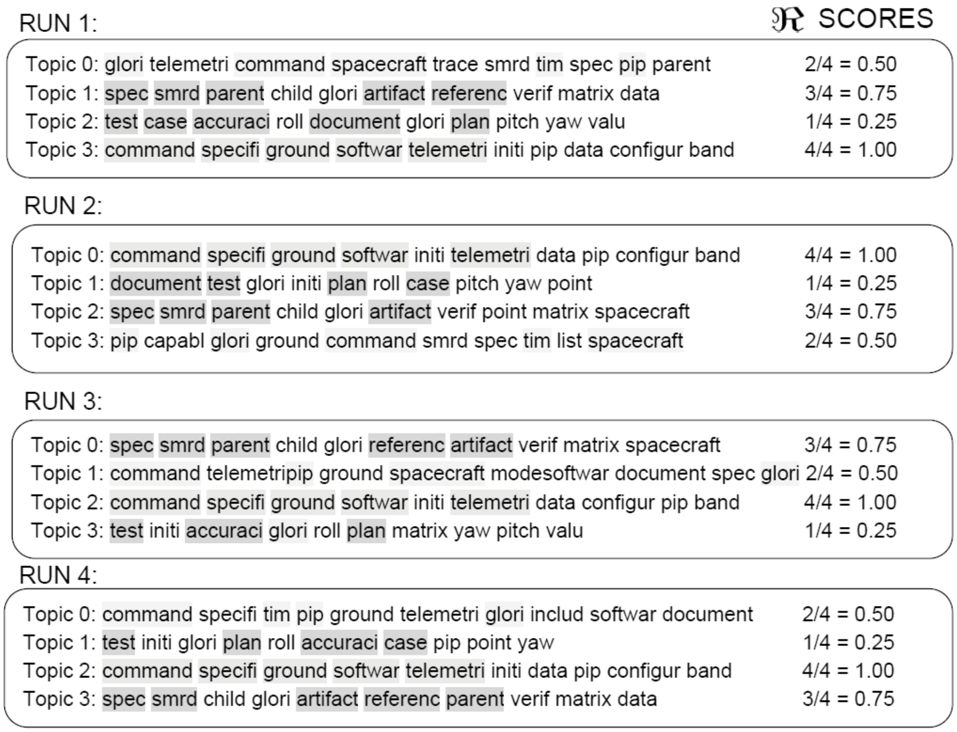
For this study, all datasets were preprocessed using the usual text mining filters [**?**]:

- Stop words removal using NLTK toolkit[3] [**?**] : ignore very common short words such as "and" or "the".
- Porter's stemming filter [**?**]: delete uniformative word endings; e.g. after stemming, all the following words

[1] https://archive.org/details/stackexchange

[2] https://github.com/ai-se/citemap/blob/master/citemap.csv

[3] http://www.nltk.org/book/ch02.html



RUN 1: ℜ SCORES

Topic 0: glori telemetri command spacecraft trace smrd tim spec pip parent 2/4 = 0.50
Topic 1: spec smrd parent child glori artifact referenc verif matrix data 3/4 = 0.75
Topic 2: test case accuraci roll document glori plan pitch yaw valu 1/4 = 0.25
Topic 3: command specifi ground softwar telemetri initi pip data configur band 4/4 = 1.00

RUN 2:

Topic 0: command specifi ground softwar initi telemetri data pip configur band 4/4 = 1.00
Topic 1: document test glori initi plan roll case pitch yaw point 1/4 = 0.25
Topic 2: spec smrd parent child glori artifact verif point matrix spacecraft 3/4 = 0.75
Topic 3: pip capabl glori ground command smrd spec tim list spacecraft 2/4 = 0.50

RUN 3:

Topic 0: spec smrd parent child glori referenc artifact verif matrix spacecraft 3/4 = 0.75
Topic 1: command telemetripip ground spacecraft modesoftwar document spec glori 2/4 = 0.50
Topic 2: command specifi ground softwar initi telemetri data configur pip band 4/4 = 1.00
Topic 3: test initi accuraci glori roll plan matrix yaw pitch valu 1/4 = 0.25

RUN 4:

Topic 0: command specifi tim pip ground telemetri glori includ softwar document 2/4 = 0.50
Topic 1: test initi glori plan roll accuraci case pip point yaw 1/4 = 0.25
Topic 2: command specifi ground softwar telemetri initi data pip configur band 4/4 = 1.00
Topic 3: spec smrd child glori artifact referenc parent verif matrix data 3/4 = 0.75

Figure 4: Example of topics overlap off size $n = 5$ across multiple runs.

would be rewritten to "connect": "connection", "connections", "connective", "connected", "connecting".

- Tf-idf feature selection: focus on the 5% of words that occur frequently, but only in small numbers of documents. If a word occours w times and is found in d documents and there are W, D total number of words and documents respectively, then tf-idf is scored as follows:

$$tfidf(w, d) = \frac{w}{W} * \log \frac{D}{d}$$

Table III shows the sizes of our data before and after preprocessing. These datasets are of different sizes and so are processed using different tools:

- PITS and Citemap is small enough to process on a single (four core) deskop machine using Scikit-Learn [?] and Python.
- StackOverflow is so large (7GB) that its processing requires extra hardware support. This study used a Spark and Mllib on a cluster of 45 nodes to reduce the runtime.

B. Similarity Scoring

To evaluate topics coherence in LDA, there is a direct approach, by asking people about topics, and an indirect approach by evaluating *pointwise mutual information (PMI)* [?], [?] between the topic words. We could not use any of these criteria, as it requires experts to have domain knowledge. *Perplexity* is the inverse of the geometric mean per-word likelihood. The smaller the perplexity, the better (less uniform) is the LDA model. The usual trend is that as the value of perplexity drops, the number of topics should grow [?]. Researchers caution that the value of perplexity doesn't remain constant with different topic size and with dictionary sizes [?], [?]. A lot depend on the code implementation of perplexity and the type of datasets used. Since, we are using different implementations of LDA across different platforms on various datasets, we are not using perplexity as evaluation measure.

Another approach researchers have used is *Jaccard Similarity* [?], [22]. We define our measure similar to it. For this work, we assess topic model stability via the *median number overlaps of size n words* ($size\_of\_topic$), which we denote $\Re_n$.

For this measurement, we first determine the maximum size of topics we will study. For that purpose, we will study the $n = 9$ which we call but the standard in the LDA literature [?], [51] says that top 10 terms per topic are ideal. We did not select $n = 10$ just because our results showed that the $\Re_9$ is negligible.

Next, for $1 \leq n \leq 9$, we will calculate the median size of the overlap, computed as follows:

- Let one *run* of our rig shuffle the order of the training data, then builds topic models using the data;
- m runs of our rig executes m copies of one run, each time using a different random number seed,
- We say topics are stable, when there are x occurrences of n terms appearing in all the topics seen in the m runs.

For example, consider the topics shown in Figure 4. These are generated via four *runs* of our system. For $n = 5$, we note that Topic 0 of run1 scores $\frac{2}{4} = 0.5$ since it shares 5 words with topics in only two out of four runs. Repeating that calculation for the other run1 topics shows that:

- Topic 1 of run1 scores $\frac{3}{4} = 0.75$;
- Topic 2 or run1 scores $\frac{1}{4} = 0.25$;
- Topic 3 of run1 scores $\frac{4}{4} = 1$.

From this information, we can calculate $\Re_5$ (the *median number overlaps of size* $n = 5$ *words*) as:

$$median(0.5, 0.75, 0.25, 1) = 0.625$$

Figure 5 shows the $\Re_n$ scores of Figure 4 for $1 \le n \le 9$. From this figure, we can see LDA topic instability since any report of the contents of a topic that uses more than three words per topic would be unreliable.

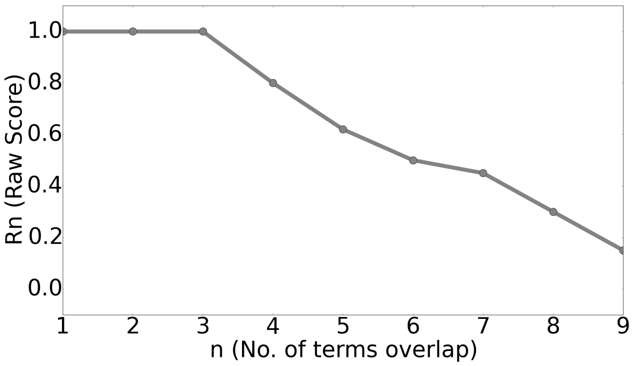


Figure 5: $\Re_n$ scores of Figure 4 for $1 \le n \le 9$

For the following analysis, we distinguish between the ***Raw score*** and the ***Delta score***:

- The two ***Raw scores*** are the $\Re_n$ median similarity scores seen *before* and *after* tuning LDA;
- The ***Delta score*** is the difference between the two ***Raw scores***.

The pseudocode for these calculations is shown in Algorithm 1 with the default set of parameters. In the following description, superscript numbers denote lines in the pseudocode. The data ordering is shuffled every time LDA is run^5. Data is in the form of term frequency scores of each word per document. Shuffling is done in order to induce maximum variance among the ordering of data with different runs of LDA. Topics^6 are a list of list which contains topics from all the different runs. A stability score is evaluated on every 10 runs (Fixed), and this process is continued 10 (Fixed) times. At the end, the median score is selected as the untuned raw score ($\Re_n$) $^{3-11}$. Hence, the runtimes comes from $10*10$ evaluations of untuned experiment.

C. Tuning Topic Modeling with LDADE

LDADE is a combination of topic modeling (with LDA) and an optimizer (differential evolution, or DE) that adjusts the parameters of LDA in order to optimize (i.e. maximize) similarity scores.

We choose to use DE after a literature search on search-based SE methods. The literature mentions many optimizers: simulated annealing [?], [?]; various genetic algorithms [?] augmented by techniques such as differential evolution [62], tabu search and scatter search [?], [?], [?], [?]; particle swarm optimization [?]; numerous decomposition approaches that use heuristics to decompose the total space into small problems, then apply a response surface methods [?], [?]. Of these, the simplest ones are simulated annealing (SA) and differential evolution (DE) and our reading of the current literature is that there are more advocates for differential evolution than SA. We use DE here, since it was successful in our study also.

Algorithm 1 Pseudocode for untuned LDA with Default Parameters

```
Input: $Data$, $n$, $k$, $\alpha$, $\beta$ (Defaults)
Output: $Raw\_Score$
 1: function LDASCORE( $n$, $Data$)
 2:     $Score \leftarrow \emptyset$
 3:     for $j = 0$ to 10 do
 4:         for $i = 0$ to 10 do
 5:             $data \leftarrow$ shuffle($Data$)
 6:             $Topics$.add(lda($k, \alpha, \beta$ ))
 7:         end for
 8:         $Score$.add($Overlap$(Topics, $n$, $l$[0]))
 9:     end for
10:     $Raw\_Score \leftarrow$ median(Score)
11:     return $Raw\_Score$
12: end function
```

LDADE adjusts the parameters of Table IV. Most of these parameters were explained above. As to the rest:

- VEM is the deterministic *variational EM* method that computes α, β via expectation maximization [45].
- Gibbs sampling [?], [29] is a Markov Chain Monte Carlo algorithm, which is an approximate stochastic process for computing and updating α, β. Topic modeling researchers in SE have argued that Gibbs leads to stabler models [?], [?] (a claim which we test, below).

We manually adjust these inference techniques according to different implementations across different platforms.

Parameters	**Defaults**	**Tuning Range**	**Description**
k	10	[10,100]	Number of topics or cluster size
α	None	[0,1]	Prior of document topic distribution. This is called alpha
β	None	[0,1]	Prior of topic word distribution. This is called beta
inferences	VEM	VEM Gibbs Sampling	Sampling methods

Table IV: List of parameters tuned by this paper

Algorithm 2 shows LDADE's version of DE. DE evolves a *NewGeneration* of candidates from a current Population. Each candidate solution in the Population is a pair of (Tunings, Scores). Tunings are selected from Table IV and Scores come similarly from Alogrithm 1^{3-11}. Note that there is not any outer loop^3: in Algorithm 2, LDA is only run as 1 rig$^{11\&12}$. Here, the runtimes comes from $iter * np * 10$ evaluations of tuned experiment.

The main loop of DE^9 runs over the *Population*, replacing old items with new Candidates (if new candidate is better). DE generates *new Candidates* via extrapolating^{23} between current solutions in the frontier. Three solutions a, b, c are selected at random. For each tuning parameter i, at some probability *cr*, we replace the old tuning x_i with y_i. For booleans, we use $y_i = x_i$ (see line 31). For numerics, $y_i = a_i + f \times (b_i - c_i)$ where f is a parameter controlling crossover. The trim function^{33} limits the new value to the legal range min..max of that parameter.

Algorithm 2 Pseudocode for DE with a constant number of iterations

```
Input: np= 10, f= 0.7, cr= 0.3, iter= 3, Goal ∈ Finding maximum score
Output: $Raw\_Score, final\_generation$
function DE($np, f, cr, iter, Goal$)
    $Cur\_Gen \leftarrow \emptyset$
    $Population$ ←InitializePopulation(np)
    for $i = 0$ to $np - 1$ do
        $Cur\_Gen$.add($Population$[i],ldascore($Population$[i],$n$,$Data$)
    end for
    for $i = 0$ to $iter$ do
        $NewGeneration \leftarrow \emptyset$
        for $j = 0$ to $np - 1$ do
            $S_i$ ←Extrapolate($Population$[j],Population,cr,f,np)
            if ldascore($S_i$) $\geq$ $Cur\_Gen$[j][1] then
                $NewGeneration$.add($S_i$,ldascore($S_i$,$n$, $Data$))
            else
                $NewGeneration$.add($Cur\_Gen$[j])
            end if
            $Cur\_Gen \leftarrow NewGeneration$
        end for
    end for
    $Raw\_Score$ ← GetBestSolution($Cur\_Gen$)
    $final\_generation \leftarrow Cur\_Gen$
    return $Raw\_Score, final\_generation$
end function
function EXTRAPOLATE($old, pop, cr, f, np$)
    $a, b, c \leftarrow threeOthers$(pop, old)
    $newf \leftarrow \emptyset$
    for $i = 0$ to $np - 1$ do
        if $cr \leq$ random() then
            $newf$.add($old[i]$)
        else
            if typeof($old$[i])== bool then then
                $newf$.add(not $old[i]$)
            else
                $newf$.add(trim(i,($a$[i]+$f$∗($b$[i] − $c$[i]))))
            end if
        end if
    end for
    return $newf$
end function
```

The loop invariant of DE is that, after the zero-th iteration[7], the *Population* contains examples that are better than at least one other candidate. As the looping progresses, the *Population* is full of increasingly more valuable solutions which, in turn, also improves the candidates, which are Extrapolated from the Population. Hence, Vesterstrom and Thomsen [**?**] found DE to be competitive with particle swarm optimization and other GAs.

Note that DEs have been applied before for parameter tuning (e.g. see [**?**], [**?**], [**?**], [42], [51], [63]) but this is the first time they have been applied to tune LDA to increase stability.

IV. EXPERIMENTATION

In this section, any result from the smaller data sets (Pits and Citemap) come from Python implementation based on Scikit-Learn running on a single 4-core machine. Also, any results from the larger data (Stackoverflow) comes from a Scala implementation based on Mllib running on a 45 node Spark system (8 cores per node).

Note that, for the most part, we defer the Stackoverflow results to the §V *Threats to Validity* where we will show that (a) topic modeling instability exists across multiple platforms and implementation languages; (b) tuning can improve stabilities for different platforms and implements for LDA.

A. *RQ1: Are the default settings of LDA correct?*

This first research question checks the core premise of this article– that changes in the order of training data dramatically effects the topics learned via LDA. Note that if this is *not true*, then there would be no value-added to this paper.

Figure 6 plots n vs $\Re_n$ for untuned LDA. Note that the stability collapses the most after $n = 5$ words. This means that any report of LDA topics that uses more than five words per topic will be changed, just by changing the order of the inputs. This is a significant result since the standard advice in the LDA papers [**?**], [51] is to report the top 10 words per topic. As shown in Figure 7a, it would be rare that any such 10 word topic would be found across multiple runs.

Result 1
Using the default settings of LDA for SE data can lead to systematic errors due to topic modeling instability.

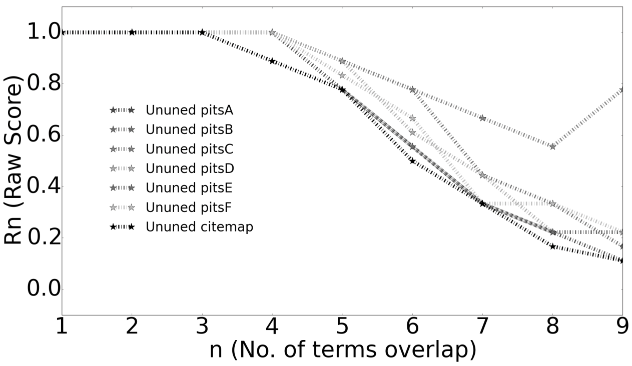


Figure 6: *Before* tuning: uses LDA's default parameters

B. *RQ2: Does tuning improve the stability scores of LDA?*

Figure 7a and Figure 7b shows the stability improvement generated by tuning. Tuning never has any negative effect (reduces stability) and often has a large positive effect– particular after 5 terms overlap. The largest improvement we was in PitsD dataset which for up to 8 terms overlap was 100% (i.e. was always found in all runs). Overall, after reporting topics of up to 7 words, in the majority case (66%), those topics can be found in models generated using different input orderings. Accordingly, our answer to **RQ2** is:

Result 2
Based on Figure 7, we strongly recommend tuning for future LDA studies.

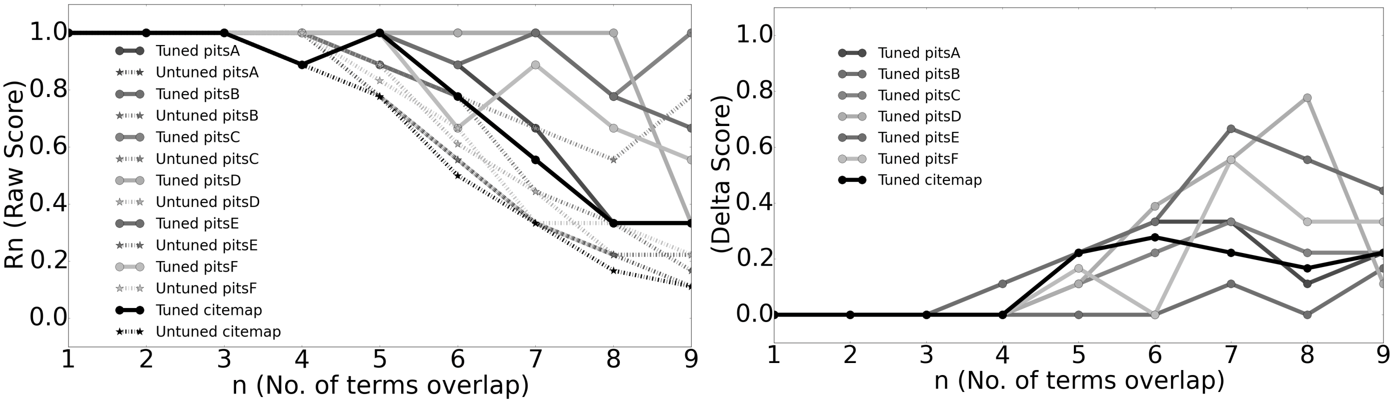


Figure 7a: *After* tuning: uses parameters learned by DE.

Figure 7b: *Delta = After - Before.*

Figure 7: **RQ1, RQ2** stability results over ten repeated runs. In these figures, *larger* numbers are *better*.

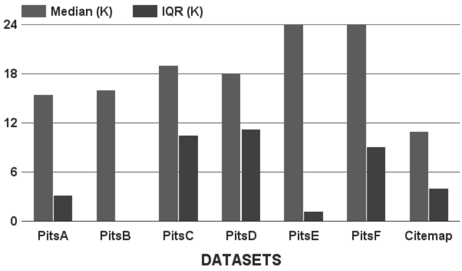


Figure 8: Datasets vs Parameter (k) variation

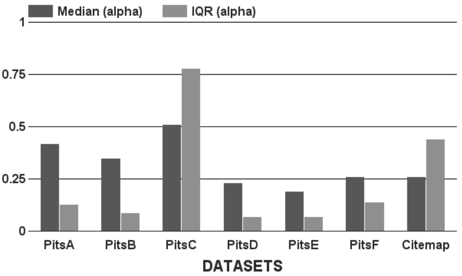


Figure 9 Datasets vs Parameter (α) variation

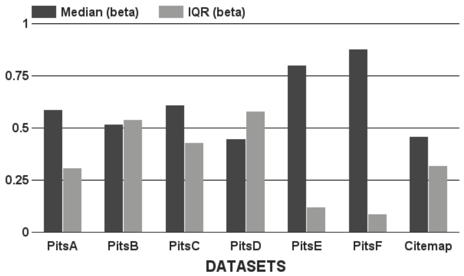


Figure 10 Datasets vs Parameter (β) variation

C. *RQ3: Do different data sets need different configurations to make LDA stable?*

Figures 8, 9, and 10 show the results of tuning. On display in each set of vertical bars are the median values generated across 10 tunings. Also, shown are the inter-quartile range (IQR) of those tunings (the IQR is the 75th-25th percentile values and is a non-parametric measure of variation around the median value). Note that in Figure 8, IQR=0 for PitsB dataset where tuning always converged on the same final value.

These figures show how tuning selects the different ranges of parameters. Some of the above numbers are far from the standard values; e.g. Garousi et al. [24] recommend using $k = 67$ topics yet in our data sets, best results were seen using $k \leq 24$. Clearly:

Result 3
Do not reuse tunings suggested by other researchers from other data sets. Instead, always retune for all new data.

D. *RQ4: Is tuning easy?*

The DE literature recommends using a population size np that is ten times larger than the number of parameters being optimized [62]. For example, when tuning k, α, β, the DE literature is recommending $np = 30$. Figure 11 explores $np = 30$ vs the $np = 10$ we use in Algorithm 2 (as well as some other variants of DE's F and CR parameters). That figure shows results jsut for Citemap and, for space reasons, results relating to other data sets are shown at https://goo.gl/HQNASF. After reviewing the results from all the datasets, we can say that there isn't much of an improvement by using different F, CR, and Population size. So our all other experiments used $F = 0.7$, $CR = 0.3$ and $np = 10$. Also:

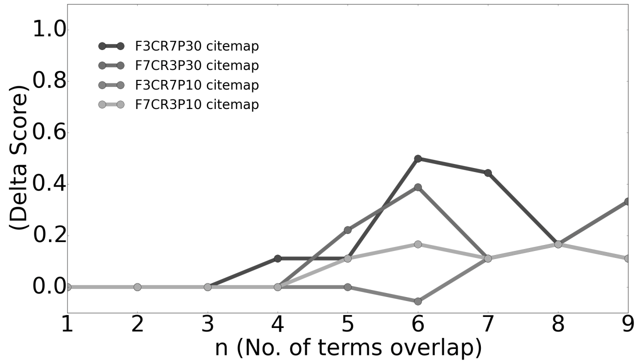


Figure 11: Terms vs Delta Improvement using Different settings of DE

Result 4
Finding stable parameters for topic models is easier than standard optimization tasks.

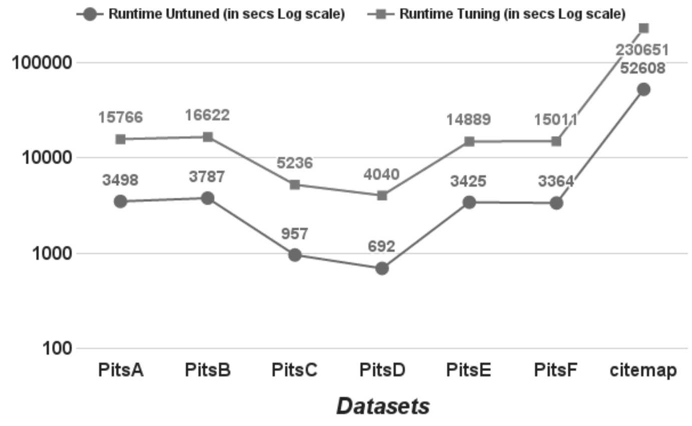


Figure 12 VEM: Datasets vs Runtimes

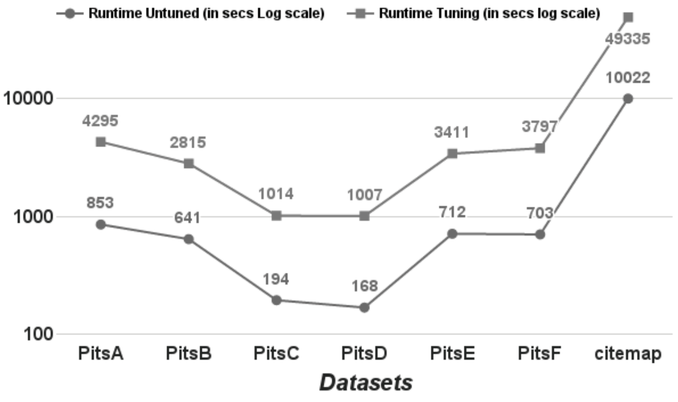


Figure 13: Gibbs: Datasets vs Runtimes

*E. **RQ5: Is tuning slow who are making conclusions out of these topics?***

Search-based SE methods can be very slow. Harman et al. once needed 15 years of CPU time to find and verify the tunings required for software clone detectors [**?**]. Sayyad et al. routinely used 10^6 evaluations (or more) of their models in order to extract products from highly constrained product lines [**?**]. Hence, before recommending any search-based method, it is wise to consider the runtime cost of that recommendation.

To understand our timing results, recall that untuned, tuned LDA use Algorithm 1, Algorithm 2 respectively. In general, the number of evaluations taken by tuning is 3 times more than untuned experiments.

Figures 12 and 13 are shown in blue, red the runtimes required to run LDA untuned, tuned (respectively). The longer runtimes (in red) include the times required for DE to find the tunings. Overall, tuning slows down LDA by a factor of up to seven, but often less than five. The answer is in the relative times required for LDA and DE: the latter is much slower than the former which means variances in the performance of LDA dominate these timing results.

Regardless of whether or not the slow down is a factor of five or seven, the main result here is that the overall slowdown is not dramatic. Given the added benefit of tuning (stable topics), we conclude that:

Result 5
Tuning LDA is not slow for those who are making conclusions out of these topics.

*F. **RQ6: Should data miners be used "off-the-shelf" with their default tunings?***

Figure 7 shows that there is much benefit in tuning. Figures 8, 9, and 10 show that the range of "best" tunings is very dataset-specific. Hence, for a new dataset, the off-the-shelf tunings may often fall far from the useful range. Figures 12 and 13 show that tuning is definitely slower than otherwise, but the overall cost is not prohibative. Hence:

Result 6
Overall, we can see little reason to use "off-the-shelf" LDA by those who are making conclusions out of these topics.

V. Threats to Validity

The usual software analytics threats to validity hold for this paper:

- Our conclusions assume that the goal of topic modeling is to produce stable topics that humans will browse and reflect on. As mentioned in the introduction, there are class of papers that do *not* do that. Rather, they use LDA as part of some internal process where the topics are intermediaries used to generate some other goal [**?**]. As we said at the start of this paper, our results do not effect that kind of paper.
- The conclusions of this paper are based on a finite number of data sets and it is possible that other data might invalidate our conclusions. As with all analytics papers, any researcher can do is to make their conclusions and materials public, then encourage other researchers to repeat/ refute/ improve their conclusions. For example, the footnotes of this paper contain all the urls where other researchers can download our materials and explore the conclusions of this paper.

As to more specific threats to validity, one issue might be that our conclusions on "LDA" are really quirks of a specific implementation. To check this, it is insightful to compare our results with:

- The Pits and Citemap results, executed in Scikit-Learn and Python running on a desktop machine.
- The StackOverflow data set executed in Scala using Mllib running on a Spark cluster.

Another useful comparison is to change the internal of the LDA:

- Sometimes use VEM sampling;
- Sometimes use Gibbs sampling;

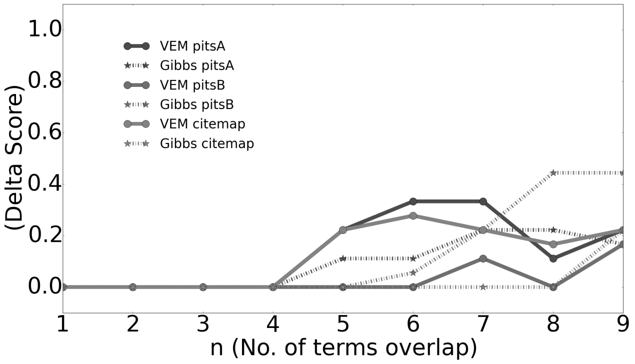


Figure 14: GIBBS vs VEM

Figure 14 compares the VEM vs Gibbs sampling. and Figure 15 shows tuning results for StackOverflow, Citemap, and PitsA using Scala/Spark cluster (for results on other data sets, see https://goo.gl/faYAcg and https://goo.gl/UVaql1) When compared with the Python/desktop results of Figure 7 we see the same patterns:

- Tuning never makes stability worse.
- Sometimes, it dramatically improves it (in particular, see the Citemap results of Figure 15).

That said, there are some deltas between VEM and Gibbs where it seems tuning is more important for VEM than Gibbs (evidence: the improvements seen after tuning are largest for the VEM results of Figure 14 and at https://goo.gl/faYAcg).

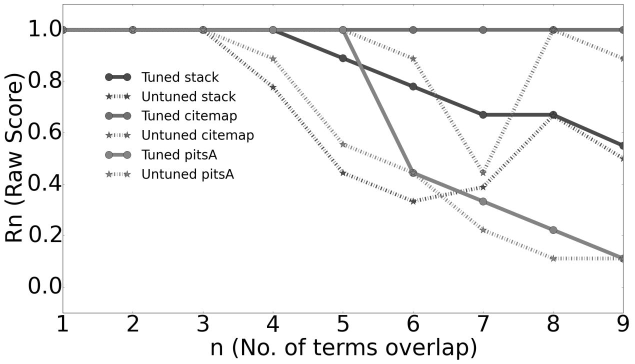


Figure 15: Spark Results

Another threat to validity of this work is that it is a quirk of the control parameters used within our DE optimizer. We have some evidence that this is not the case. Figure 11 and https://goo.gl/HQNASF explored a range of DE tunings and found little difference across that range. Also, Table V explores another choice within DE – how many evaluations to execute before terminating DEs. All the results in this paper use an evaluation budget of 30 evaluations. Table V compares results across different numbers of evaluations. While clearly, the more evaluations the better, there is little improvement after the 30 evaluations used in this paper.

Datasets\Evaluations	**10**	**20**	**30**	**50**
PitsA	0.9	0.9	1.0	1.0
PitsB	0.9	0.9	0.9	1.0
PitsC	0.9	1.0	1.0	1.0
PitsD	0.9	1.0	1.0	1.0
PitsE	0.9	0.9	1.0	1.0
PitsF	0.9	0.9	0.9	0.9
Citemap	0.67	0.67	0.77	0.77
StackOverflow	0.6	0.7	0.8	0.8

Table V: Evaluations vs Stability Scores

VI. Conclusion and Future Work

Based on the above, we offer a specific and general conclusion. Most specifically, we recommend

- Any study that shows the topics learned from LDA, then uses that display to make a particular conclusion, needs to first tune LDA. We say this since the topics learned from untuned LDA are unstable, i.e. different input orderings will lead to different conclusions. However, after tuning, stability can be greatly increased.
- Unlike the advise of Stacy et al. [**?**], LDA topics should not be reported as the top ten words. Due to order effects, such a report can be highly incorrect. Our results show that up to eight words can be reliably reported, but only after tuning for stability using tools like LDADE.
- Do not download someone else's pre-tuned LDA since, as shown here, the best LDA tunings vary from data set to data set.

Our experience is that this recommendation is not an arduous demand since tuning adds less than a factor of ten to the total run times of an LDA study.

More generally, we comment that the field of software analytics needs to make far more use of search-based software engineering in order to tune their learners. In other work, we have shown that tuning significantly helps defect prediction [**?**] (a result that has also been reported by other reseachers [51]). In this work, we have shown that tuning significantly helps topic modeling by mitigating a systematic error in LDA (order effects that lead to unstable topics). The implications of this study for other software analytics tasks is now an open and pressing issue. In how many domains can search-based SE dramatically improve software analytics?

References

[1] A. Agrawal, T. Menzies, and W. Fu. What's wrong with topic modeling? (and how to fix it using search-based se). Submitted to ICSE'17. Available from http://tiny.cc/LDADE.

[2] A. Al-Zubidy and J. C. Carver. Review of systematic literature review tools. *University Of Alabama Technical Report*, 2014.

[3] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Learning operational requirements from goal models. In *Proceedings of the 31st international conference on software engineering*, pages 265–275. IEEE Computer Society, 2009.

[4] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[5] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the 31st International Conference on Software Engineering*, pages 298–308. IEEE Computer Society, 2009.

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

[7] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, 2004.

[8] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 368–377. IEEE, 1998.

[9] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 125–134. IEEE, 2010.

[10] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced?: bias in bug-fix datasets. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 121–130. ACM, 2009.

[11] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35. ACM, 2008.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[13] K.-Y. Cai and D. Card. An analysis of research topics in software engineering–2006. *Journal of Systems and Software*, 81(6):1051–1058, 2008.

[14] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 335–344. ACM, 2010.

[15] N. Coulter, I. Monarch, and S. Konda. Software engineering as seen through its research literature: A study in co-word analysis. *Journal of the American Society for Information Science*, 49(13):1206–1223, 1998.

[16] D. Dig, J. Marrero, and M. D. Ernst. Refactoring sequential java code for concurrency via concurrent libraries. In *Proceedings of the 31st International Conference on Software Engineering*, pages 397–407. IEEE Computer Society, 2009.

[17] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *Proceedings of the 31st International Conference on Software Engineering*, pages 111–121. IEEE Computer Society, 2009.

[18] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001.

[19] J. M. Fernandes. Authorship trends in software engineering. *Scientometrics*, 101(1):257–271, 2014.

[20] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 23–32. IEEE, 2003.

[21] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198. IEEE, 1998.

[22] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 582–591. IEEE Press, 2013.

[23] V. Garousi. A bibliometric analysis of the turkish software engineering research community. *Scientometrics*, 105(1):23–49, 2015.

[24] V. Garousi and M. V. Mäntylä. Citations, research topics and active countries in software engineering: A bibliometrics study. *Computer Science Review*, 19:56–77, 2016.

[25] V. Garousi and T. Varma. A bibliometric assessment of canadian software engineering scholars and institutions (1996-2006). *Computer and Information Science*, 3(2):19, 2010.

[26] R. L. Glass and T. Y. Chen. An assessment of systems and software engineering scholars and institutions (1999–2003). *Journal of Systems and Software*, 76(1):91–97, 2005.

[27] R. L. Glass, I. Vessey, and V. Ramesh. Research in software engineering: an analysis of the literature. *Information and Software technology*, 44(8):491–506, 2002.

[28] C. Gould, Z. Su, and P. Devanbu. Static checking of dynamically generated queries in database applications. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 645–654. IEEE, 2004.

[29] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[30] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 153–162. IEEE, 2014.

[31] S. Hangal and M. S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the 24th international conference on Software engineering*, pages 291–301. ACM, 2002.

[32] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.

[33] H. Hemmati, Z. Fang, and M. V. Mantyla. Prioritizing manual test cases in traditional and rapid release environments. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pages 1–10. IEEE, 2015.

[34] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 837–847. IEEE, 2012.

[35] A. Hoonlor, B. K. Szymanski, and M. J. Zaki. Trends in computer science research. *Communications of the ACM*, 56(10):74–83, 2013.

[36] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*, pages 467–477. ACM, 2002.

[37] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst. Automatic creation of sql injection and cross-site scripting attacks. In *2009 IEEE 31st International Conference on Software Engineering*, pages 199–209. IEEE, 2009.

[38] M. Kim and D. Notkin. Discovering and representing systematic code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 309–319. IEEE Computer Society, 2009.

[39] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering - A systematic literature review, 2009.

[40] J. Krinke. Identifying similar code with program dependence graphs. In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, pages 301–309. IEEE, 2001.

[41] J. Liu, D. Batory, and C. Lengauer. Feature oriented refactoring of legacy applications. In *Proceedings of the 28th international conference on Software engineering*, pages 112–121. ACM, 2006.

[42] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving trace accuracy through data-driven configuration and composition of tracing features. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 378–388. ACM, 2013.

[43] C. Marshall and P. Brereton. Tools to support systematic literature reviews in software engineering: A mapping study. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 296–299. IEEE, 2013.

[44] A. K. Massey, J. Eisenstein, A. I. Antón, and P. P. Swire. Automated text mining for requirements analysis of policy documents. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 4–13. IEEE, 2013.

[45] T. Minka and J. Lafferty. Expectation-propagation for the generative aspect model. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 352–359. Morgan Kaufmann Publishers Inc., 2002.

[46] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th international conference on software engineering*, pages 503–512. ACM, 2002.

[47] E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, 38(1):5–18, 2012.

[48] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[49] S. I. Nikolenko, S. Koltcov, and O. Koltsova. Topic modelling for qualitative studies. *Journal of Information Science*, page 0165551515617393, 2015.

[50] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 68–71. IEEE, 2010.

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

[52] M. Ponweiser. Latent dirichlet allocation in r. *Institute for Statistics and Mathematics*, 2012.

[53] K. Prete, N. Rachatasumrit, N. Sudan, and M. Kim. Template-based reconstruction of complex refactorings. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[54] D. C. Rajapakse and S. Jarzabek. Using server pages to unify clones in web applications: A trade-off analysis. In *29th International Conference on Software Engineering (ICSE'07)*, pages 116–126. IEEE, 2007.

[55] S. Rao and A. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 43–52. ACM, 2011.

[56] A. Restificar and S. Ananiadou. Inferring appropriate eligibility criteria in clinical trial protocols without labeled data. In *Proceedings of the ACM sixth international workshop on Data and text mining in biomedical informatics*, pages 21–28. ACM, 2012.

[57] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 222–231. ACM, 2014.

[58] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold. Lightweight fault-localization using multiple coverage types. In *Proceedings of the 31st International Conference on Software Engineering*, pages 56–66. IEEE Computer Society, 2009.

[59] K. Sen, D. Marinov, and G. Agha. Cute: a concolic unit testing engine for c. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 263–272. ACM, 2005.

[60] Simple h-index estimation. http://shine.icomp.ufam.edu.br/index.php. Accessed: February 2013.

[61] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *ACM sigsoft software engineering notes*, volume 30, pages 1–5. ACM, 2005.

[62] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[63] X. Sun, B. Li, H. Leung, B. Li, and Y. Li. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology*, 66:1–12, 2015.

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

[65] T. Systä, M. Harsu, and K. Koskimies. Inbreeding in software engineering conferences, 2012.

[66] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[67] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Studying software evolution using topic models. *Science of Computer Programming*, 80:457–479, 2014.

[68] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein. Static test case prioritization using topic models. *Empirical Software Engineering*, 19(1):182–212, 2014.

[69] S. Thummalapenta and T. Xie. Mining exception-handling rules as sequence association rules. In *Proceedings of the 31st International Conference on Software Engineering*, pages 496–506. IEEE Computer Society, 2009.

[70] S. Thummalapenta, T. Xie, N. Tillmann, J. De Halleux, and W. Schulte. Mseqgen: Object-oriented unit-test generation via mining source code. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 193–202. ACM, 2009.

[71] K. Tian, M. Revelle, and D. Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 163–166. IEEE, 2009.

[72] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *Proceedings of the 31st International Conference on Software Engineering*, pages 12–22. IEEE Computer Society, 2009.

[73] G. Tsafnat, P. Glasziou, M. K. Choong, A. Dunn, F. Galgani, and E. Coiera. Systematic review automation technologies. *Systematic reviews*, 3(1):1, 2014.

[74] B. Vasilescu, A. Serebrenik, and T. Mens. A historical dataset of software engineering conferences. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 373–376. IEEE Press, 2013.

[75] B. Vasilescu, A. Serebrenik, T. Mens, M. G. van den Brand, and E. Pek. How healthy are software engineering conferences? *Science of Computer Programming*, 89:251–272, 2014.

[76] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[77] G. Wassermann and Z. Su. Static detection of cross-site scripting vulnerabilities. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 171–180. IEEE, 2008.

[78] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering*, pages 364–374. IEEE Computer Society, 2009.

[79] C. Wohlin. An analysis of the most cited articles in software engineering journals-2000. *Information and Software Technology*, 49(1):2–11, 2007.

[80] T. Wolf, A. Schroter, D. Damian, and T. Nguyen. Predicting build failures using social network analysis on developer communication. In *Proceedings of the 31st International Conference on Software Engineering*, pages 1–11. IEEE Computer Society, 2009.

[81] G. Yang and B. Lee. Predicting bug severity by utilizing topic model and bug report meta-field. *KIISE Transactions on Computing Practices*, 21(9):616–621, 2015.

[82] F. Yu, M. Alkhalaf, and T. Bultan. Patching vulnerabilities with sanitization synthesis. In *Proceedings of the 33rd International Conference on software engineering*, pages 251–260. ACM, 2011.

[83] Y. Zhang, M. Harman, Y. Jia, and F. Sarro. Inferring test models from kate's bug reports using multi-objective search. In *Search-Based Software Engineering*, pages 301–307. Springer, 2015.