Improving Problem Identification via Automated Log Clustering using Dimensionality Reduction

Carl Martin Rosenberg
Simula Research Laboratory
Oslo, Norway
cmr@simula.no

Leon Moonen
Simula Research Laboratory
Oslo, Norway
leon.moonen@computer.org

ABSTRACT

Background: Continuous engineering practices, such as continuous integration and continuous deployment, see increased adoption in modern software development. A frequently reported challenge for adopting these practices is the need to make sense of the large amounts of data that they generate.
Goal: We consider the problem of automatically grouping logs of runs that failed for the same underlying reasons, so that they can be treated more effectively, and investigate the following questions: (1) Does an approach developed to identify problems in *system logs* generalize to identifying problems in *continuous deployment logs*? (2) How does *dimensionality reduction* affect the quality of automated log clustering? (3) How does the criterion used for merging clusters in the clustering algorithm affect clustering quality?
Method: We replicate and extend earlier work on clustering system log files to assess its generalization to continuous deployment logs. We consider the optional inclusion of one of these dimensionality reduction techniques: Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), and Non-negative Matrix Factorization (NMF). Moreover, we consider three alternative cluster merge criteria (Single Linkage, Average Linkage, and Weighted Linkage), in addition to the Complete Linkage criterion used in earlier work. We empirically evaluate the 16 resulting configurations on continuous deployment logs provided by our industrial collaborator.
Results: Our study shows that (1) identifying problems in continuous deployment logs via clustering is feasible, (2) including NMF significantly improves *overall* accuracy and robustness, and (3) Complete Linkage performs best of all merge criteria analyzed.
Conclusions: We conclude that problem identification via automated log clustering is improved by including dimensionality reduction, as it decreases the pipeline's sensitivity to parameter choice, thereby increasing its robustness for handling different inputs.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; **Software defect analysis**;

KEYWORDS

Continuous engineering, failure diagnosis, log analysis, log mining.

ACM Reference Format:
Carl Martin Rosenberg and Leon Moonen. 2018. Improving Problem Identification via Automated Log Clustering using Dimensionality Reduction. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '18), October 11–12, 2018, Oulu, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3239235.3239248

1 INTRODUCTION

Continuous Engineering (CE) practices such as Continuous Integration (CI), Continuous Delivery (CDy), Continuous Deployment (CDt), and Continuous Release (CR), are increasingly adopted to meet the demand for incremental software development with rapid feedback. Each of these practices can be characterized by their focus on creating short and automated cycles to give developers early feedback on potential issues and reduce risk by taking repeated incremental steps. These cycles are triggered in response to new code being committed to the Version Control System, via periodic scripts, or simply when a developer demands it.

CI aims at automatically building and (unit) testing software changes multiple times a day, CDy extends it with automated acceptance testing and quality checking to ensure that a product is ready for deployment, CDt adds automatic deployment to production-like hardware and deployment testing to CDy, and finally, CR further adds automatically releasing the new software to the customers.[1] When there is no need to distinguish, we will refer to them as CE.

In a traditional setup without CE, a developer or operator would manually build, test or deploy the system, observe the results and immediately react to them. With CE, on the other hand, it is possible to trigger a cycle that runs all necessary commands and tracks their outcomes in a logfile, switch context to a new task, and defer inspection of the resulting logfile to some later time. When the cycles are time consuming, this increases developer productivity. However, it also easily creates a situation where unprocessed results accumulate. This is not necessarily a problem, as long as the entire cycle is made to run correctly before the product is released. However, the vast amounts of results that can accumulate may interfere with this last goal.

Indeed, a frequently reported challenge for adopting CE practices concerns the systematic and integrated analysis of the wealth of data resulting from the automated build, test, and deployment processes [1–6]. A recent literature review identifies the lack of transparency and awareness regarding test and build results as one of the main threats to adopting CDy, together with the need

[1] There is a ongoing debate on the exact definitions and boundaries of these practices, but we will use the incremental CI $\subset$ CDy $\subset$ CDt $\subset$ CR definitions provided here.

for measures that improve coordination and collaboration on addressing such results [7]. As similar challenges were brought up by our industrial collaborator, we set out to investigate techniques to automatically support such coordination.

Goal: We consider the challenge of automatically grouping logs of CE runs that failed for the same underlying reasons. The idea is that instead of having to investigate all individual results, automated log clustering would allow for a more systematic, coordinated approach where one investigates a representative for a group of failed runs, addresses the issues that caused failure, and can reasonably expect that the issues of logs that failed for corresponding reasons have been accounted for (which will be checked by following runs). To this end, we replicate and extend earlier work by Lin et al. [8] that proposes an automatic clustering pipeline for system log sequences, so that an system operator diagnosing an issue only needs to investigate a few representative sequences.

Contributions: The main contributions of this work can be summarized as follows: (1) We conduct a replication study to investigate to what extent an approach developed to identify problems in system logs generalizes to the context of identifying problems in CDt logs. (2) We extend the original study in two ways: (2a) by investigating how the optional inclusion of one of three dimensionality reduction techniques affects the quality of automated log clustering, and (2b) by investigating three additional alternative criteria for merging clusters in the Hierarchical Agglomerative Clustering algorithm that it uses. (3) We empirically investigate the impact of the 16 resulting configurations on the quality of clustering CDt logs provided by our industrial collaborator. (4) We analyze and discuss how dimensionality reduction helps to lower the pipeline's sensitivity to parameter choices, thereby increasing its robustness for handling different inputs.

Overview: The remainder of this paper is organized as follows: Section 2 discusses Lin's approach, and Section 3 presents the variations that we investigate. Section 4 describes the setup of our empirical investigation, whose results are presented and discussed in Section 5. Section 6 presents related work, and Section 7 provides some concluding remarks.

2 BACKGROUND

The baseline for our work is LogCluster by Lin et al. [8], a technique developed for analyzing system logs from online services. This section presents the main parts of the LogCluster pipeline, and the next section describes our adaptions to it.

Conceptually, LogCluster groups logs into clusters, selects a representative log for each cluster, and presents these to the developer. To perform the clustering, the logs must first be preprocessed and represented in a manner that is amenable to clustering. In LogCluster, this comprises a *log abstraction* step that represents each log as a sequence of *events*, and a *log vectorization* step that uses *inverse document frequency* and *event contrast* to give higher priority to rare or specific events. We now present these steps in more detail.[2]

Log abstraction: This step removes runtime-specific information that is assumed to create artificial differences between logs. For example, most logs report timestamps that add such differences. Abstraction represents all timestamps using the same token, so the algorithm can focus on more salient aspects of the logs instead. Lin et al. [8] use the automatic abstraction mechanism developed by Fu et al. [9]. In our work, we use a log abstraction tool provided by our industrial collaborator, Cisco Systems Norway.

Next, each log is represented as a sequence of *event identifiers*. This assumes that the log file is a series of event reports delineated in a predictable manner, for example a newline followed by a date, time and timezone designator. After abstraction of runtime-specific information, many of these event descriptions will be identical. Thus, by giving each of these identical event descriptions the same unique identifier, logs can be compactly represented as sequence of event identifiers, which benefits the next steps of the pipeline.

Log Vectorization: The logs must be represented in a way that enables reasoning about the differences between them. A common representation strategy for text documents is the *bag-of-words* model [10]. In this model, every document is represented as a vector, the length of which is determined by the number of unique words in the corpus to be represented, as each cell in the vector represents a specific word. For a document, each cell in the vector tracks the number of times its corresponding word occurs in that document. LogCluster employs a similar strategy, but uses the event identifiers that were uncovered in the log abstraction step instead of words.

Event frequency weighting: When employing a bag-of-words-style representation strategy, it is common to transform the obtained vectors with an *inverse document frequency scheme* [10]. The goal is to give higher weight to words (or in LogCluster's case, events) that are rare in the documents to be clustered, and proportionally lower weight to frequently occurring words. In LogCluster [8], the event frequency weight $w_f(e)$ of an event e is given by: $w_f(e) = S(\log \frac{N}{n_e})$ where N is the number of logs to be clustered, n_e is the number of logs where the event e appears, and S is the Sigmoid function $1/(1 + \exp[-x])$ which normalizes the vector by ensuring that all values are between 0 and 1.

Contrast-based event weighting: Given separate sets of "interesting" logs (in our case, logs from failing CE runs) and "uninteresting" logs (i.e. the passing CE runs), Lin et al. [8] propose to give a lower weight to those events which occur in both the interesting and uninteresting sets. They refer to this scheme as *contrast-based event weighing*, and it works as follows: First, determine the set of events ΔS that only occur in the interesting set of logs, and then use ΔS to assign a *contrast weight* $w_{con}(e)$ to each event such that $w_{con}(e) = 1$ if $e \in \Delta S$, and $w_{con}(e) = 0$ otherwise.

The final weight of each event is determined by combining contrast weight with the inverse document frequency weight. LogCluster assigns equal weight to the inverse document frequency weight and the contrast-based weight, so that the final event weight $w(e)$ becomes $w(e) = 0.5 * w_{con}(e) + 0.5 * w_f(e)$.

Clustering and selecting representatives: LogCluster employs the *cosine distance* dissimilarity metric and Hierarchical Agglomerative Clustering (HAC) to cluster the logs. The clustering stops when the calculated merge distance between two candidate cluster exceeds a given threshold θ, which LogCluster sets to 0.5. Finally, a representative for each cluster is chosen by finding the log with the smallest average cosine distance to the other logs in the cluster.

[2] Note that we do not adopt the LogCluster scheme [8] wholesale. In particular, the use of a database of previously seen Log Sequences to save computational costs and increase performance is an orthogonal extension that is beyond the scope of this paper.

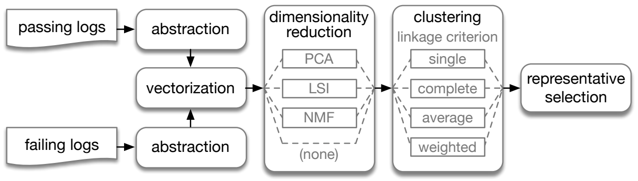

Figure 1: Overview of the variations in the clustering pipeline investigated in this paper.

3 VARIATIONS ON LOGCLUSTER

This section motivates and discusses the variations on the LogCluster pipeline (Section 2) that we investigate in our work. A high level overview of the variations is presented in Figure 1.

3.1 Dimensionality reduction

Recall that if there are n unique events in a collection of logs, each log will be represented by an n-dimensional vector. For diverse collections, n can quickly grow enormous. Since the clustering algorithm reasons about distances between vectors in an n-dimensional space, the algorithm is vulnerable to Bellman's *curse of dimensionality*: As the number of dimensions n grows, it becomes increasingly difficult to properly discriminate between points [11, 12].

A strategy to combat the curse of dimensionality is to apply *dimensionality reduction*, a transformation of the data that exploits correlations among dimensions to create a reduced space that capture most of the variance in the data [11]. Note that dimensionality reduction can improve the ability to discriminate between points, but there is also a risk of removing too much information.

We investigate the impact of applying one of the following dimensionality reduction techniques: Principal Component Analysis (PCA), Latent Semantic Indexing (LSI) or Non-Negative Matrix Factorization (NMF). While PCA is heavily used as a dimensionality reduction technique in all domains, LSI and NMF have shown particular promise in text applications. LSI has been shown to be particularly effective on text collections plagued by *polysemy* (when words have multiple meanings) and *synonymy* (when several words have the same meaning) [10]. NMF has been shown to be especially effective when the data is *sparse* and *non-negative*, which is the case for textual data in the bag-of-words model [11]. All entries in a bag-of-words vector will be either zero or some positive value as they track word frequencies. Since a document typically only contains a fraction of the words occurring throughout the collection of documents, the vectors tend to be sparse, i.e. most cells will be 0.

Estimating the number of components: To use these dimensionality reduction techniques, one must typically specify a number of components k such that the data gets reduced to a k-dimensional space, and one of the outcomes is an estimate of the *explained variance* with respect to the original data. For PCA, each component has a known contribution to the overall explained variance (evar), and components are typically ordered by decreasing contribution. This allows us to determine how many components are needed to reach a desired explained variance, by first performing an unconstrained PCA, and then iteratively accumulating components (and evar contributions) until the desired value is exceeded.

We conjecture that this PCA-based computation also provides an appropriate estimate for the components needed by LSI and NMF to achieve the same explained variance. We aim at an explained variance of 80%, a value commonly used in the literature [13], use the PCA and LSI implementations from scikit-learn [14], and the NMF implementation provided by Nimfa [15].

3.2 Alternative merge criteria in HAC

Another point of variation that we investigate is the criterion that is used to merge clusters in the clustering step of LogCluster. The Complete Linkage merge criterion that is used in LogCluster is known to be sensitive to outliers, which could prevent creation of the most intuitive clusters [10, Sec. 17.2]. For this reason, we investigate the impact of using one of the following alternative merge criteria: Single Linkage, Average Linkage, and Weighted Linkage. We give a short overview of the intuitions behind the various merge criteria, and refer to Müllner [16] for details:

Single Linkage merges the two distinct clusters for which the *smallest* distance between a pair of elements from either cluster is the *global minimum* for all pairs of clusters.

Complete linkage merges the two distinct clusters for which the *maximum distance* between a pair of elements from either cluster is the *global minimum* for all pairs of clusters.

Average Linkage (UPGMA) merges the two distinct clusters for which the *average distance* between *all pairs of elements* from either cluster is the *global minimum* for all pairs of clusters.

Weighted Linkage (WPGMA) merges a previously merged cluster $A \cup B$ and merge candidate C for which the *average distance* between sub-cluster pairs (A, C) and (B, C) is the *global minimum*.

Our experiments use the HAC implementations from SciPy [17], which implement Müllner's algorithm and merge criteria [16].

4 EXPERIMENTAL DESIGN

We conduct an empirical study to assess whether logs of CE runs that failed for the same underlying reasons can be automatically grouping using automated log clustering, and what improvements can be achieved in terms of accuracy and robustness. Specifically, we set out to answer the following research questions:

RQ1 Can the application of LogCluster [8] be generalized to identifying problems in *continuous deployment log files*?

RQ2 To what extent does applying dimensionality reduction impact the results of automated log clustering?

RQ3 To what extent does the merge criterion in HAC impact the results of automated log clustering?

Next, we detail the experimental design to answer these questions.

4.1 Datasets

We use a dataset provided by Cisco Systems Norway consisting of CDt logs and associated meta-data for 18 different comprehensive integration tests. The logs are loosely structured execution outputs that capture the process of building, deploying and testing various scenarios, not unlike the output of running a make command. Table 1 summarizes the main features of the data.

The dataset is constructed so that every failing run is associated with one (and only one) known issue. Thus, we have a *ground truth* for how the logs should be clustered together so that each cluster only contains logs concerning the same issue. This allows us to evaluate the accuracy of proposed clusterings. The ground truth is

Table 1: Overview of datasets used in this investigation.

dataset	failing	passing	dimensions	avg. dims. a/red.
1	45	4189	164	9.50
2	102	4182	192	8.00
3	103	1315	850	15.00
4	8	4277	70	1.00
5	41	297	118	5.50
6	23	2421	352	6.50
7	50	1591	71	2.50
8	78	3213	605	2.00
9	52	461	177	8.00
10	342	3152	201	17.83
11	15	3816	23	2.00
12	74	2811	133	9.00
13	106	572	181	8.00
14	60	613	162	9.50
15	140	3247	204	15.50
16	132	3238	213	16.50
17	36	530	201	7.50
18	29	344	171	6.00

The number of dimensions indicates the number of unique events in the dataset. The rightmost column shows the average number of dimensions after dimension reduction with PCA.

derived from handcrafted regular expressions developed by Cisco Systems Norway to identify whether a log concerns a known issue.

4.2 Accounting for Parameter Sensitivity

Recall from Section 2 how the contrast-based event weighting in LogCluster [8] assigns equal weight to the inverse document frequency weight and the contrast-based weight. We submit that it is unlikely that an equal split between the contrast weight and the inverse document frequency weight is suitable for all scenarios and inputs. To include a wider range of splits in our study, we introduce the contrast parameter γ such that the event weight $w(e)$ is given by: $w(e) = \gamma * w_{con}(e) + (1 - \gamma) * w_f(e)$. Observe that the original LogCluster approach is equivalent to $\gamma = 0.5$.

A second parameter to consider is the threshold θ used by the clustering algorithm to decide when to stop merging clusters. LogCluster sets this to 0.5. Also for this parameter we argue that it is unlikely that a single choice is suitable for all scenarios and inputs.

Low sensitivity to parameter choices implies the pipeline has increased robustness for handling different inputs, a desirable characteristic for an unsupervised approach such as automated log clustering [11]. To assess the sensitivity of different choices of dimensionality reduction and merge criteria to changes in parameters θ and γ, we execute each of the 16 configurations discussed in Section 3 on each of the 18 datasets with 21 choices of γ from 0 up to and including 1 in increments of 0.05, and 17 different choices of θ from 0.1 up to and including 0.9 in increments of 0.05. Thus, we execute each configuration (a choice of dimensionality reduction and merge criterion) a total of 357 times, each run representing a different choice of the γ and θ parameters on a specific dataset.

4.3 Quality Measures

We evaluate the performance of all runs with the various configurations using the following quality measures:

Adjusted Mutual Information: Lin et al. [8] use *Normalized Mutual Information* (NMI) [10] to benchmark LogCluster. However, NMI has a systematic bias in favor of clustering algorithms that group data into many small clusters, as those are more likely to have many agreements solely due to chance [18]. For this reason, we use *Adjusted Mutual Information* (AMI), which corrects for this bias [18]. It is used to compare two ways of partitioning the input, in our case the ground truth and the clustering result. An AMI score can maximally be 1, indicating a perfect correspondence between the proposed clustering and the ground truth. Conversely, an AMI near 0 indicates that the proposed clustering performs as one would expect from a solution based on random guessing.[3]

Effort Reduction: We measure how much less effort the operator has to exert after clustering by computing an *effort reduction* measure $ER = 1 - (\textit{distinct proposed clusters}/n)$ for each dataset, where n is the number of logs to be clustered. We also record the effort reduction a perfect solution would achieve by computing $IER = 1 - (\textit{distinct ground-truth clusters}/n)$.

Homogeneity and Completeness: These represent two competing quality concerns [19]. Homogeneity measures the extent to which members of a proposed cluster come from the same ground-truth class. Completeness, on the other hand, measures to what extent all members of a given ground-truth class are put in the same cluster. Perfect Homogeneity can trivially be achieved by putting each data-point in an individual cluster, but this solution will score low on Completeness. Conversely, perfect Completeness can be achieved by putting every data-point in a single cluster, but such a solution would obtain a very low Homogeneity score. Thus, in order to achieve high accuracy a clustering algorithm must score high on both Homogeneity and Completeness.

These four quality metrics gives us a nuanced view of a configuration's performance. If accuracy is low, we can determine whether it lacks Homogeneity, Completeness or both, and we can assess whether it has a bias towards either Homogeneity and Completeness. We expect Completeness to be heavily correlated with effort reduction, as both favor large clusters. Homogeneity, furthermore, serves as a risk indicator: A high Homogeneity score indicates that each log in a cluster can serve as a good representative for the issues in its cluster (i.e., few false positives or mis-clustered logs).

4.4 Statistical Procedures

To compare the configurations, we first run a Friedman test [20], as recommended by Demšar [21]. We let each combination of dimensionality reduction technique and merge criterion represent a distinct *treatment*, and every distinct combination of dataset, γ and θ represent a *block*, as illustrated in Table 2.

The Friedman test checks the null hypothesis that the treatments are equally effective against the alternative that at least one pair of treatments differ. With twenty-one choices of contrast γ, seventeen choices of threshold value θ, 18 different datasets and 16 competing configurations, our scheme has 16 different treatments and $18 \times 21 \times 17 = 6426$ blocks. If the Friedman test rejects the null hypothesis of equal effectiveness at significance level $\alpha = 0.05$, we proceed with a post-hoc analysis consisting of a paired Wilcoxon signed-rank

[3] A caveat for inspecting our results is that an AMI score of 0.5 does not correspond to random guessing, in contrast to many other accuracy measures (e.g., the F1 score).

Table 2: Overview of our experimental design. Each configuration is used as *treatment* (a row in the table), and every distinct choice of γ, θ and data-set as a *block* (a column in the table). The earlier work by Lin et al. [8] is indicated as ■, new data as △.

configurations		results (AMI)											
dimension. reduction technique	HAC merge criterion	1	…	18	…	1	…	18	…	1	…	18	dataset
		0.00	…	0.05	…	0.5	…	0.5	…	0.95	…	1.0	γ
		0.10	…	0.15	…	0.5	…	0.5	…	0.95	…	1.0	θ
none	complete	△	△	△	△	■	■	■	△	△	△	△	
	single	△	△	△	△	△	△	△	△	△	△	△	
	average	△	△	△	△	△	△	△	△	△	△	△	
	weighted	△	△	△	△	△	△	△	△	△	△	△	
PCA	…	△	△	△	△	△	△	△	△	△	△	△	
LSI	…	△	△	△	△	△	△	△	△	△	△	△	
NMF	…	△	△	△	△	△	△	△	△	△	△	△	

test [22] on each pair of treatments, as recommended by Benavoli et al. [23]. Thus, when comparing treatment a against treatment b, the paired Wilcoxon test will for every block (i.e. choice of data-set, γ and θ) pair the measurement made for a and the measurement made for b on that block. We apply the the Pratt correction [24] to the Wilcoxon signed-rank tests to handle ties, and control the family-wise error rate resulting from multiple comparisons with Holm’s procedure [25]. We measure the effect size of each pairwise comparison in terms of Vargha-Delaney A_{12} and A_{21} [26].

We use the `stats.wilcoxon` and `stats.friedmanchisquare` procedures from SciPy [17] to implement our statistical tests.

5 RESULTS AND DISCUSSION

This section first addresses RQ1 by assessing the performance of LogCluster on our dataset with respect to the quality measures discussed earlier (Section 4.3). Next, we present the results of the simultaneous empirical evaluation of all 16 configurations that arise from the variations for both dimensionality reduction and merge criterion. Finally, we use these results to answer RQ2 and RQ3.

Table 3: Results for LogCluster’s exact configuration

Dataset	AMI	H	C	ER	IER
1	0.383	0.775	0.537	0.733	0.844
2	0.673	0.853	0.747	0.853	0.873
3	0.201	0.676	0.251	0.883	0.951
4	1.000	1.000	1.000	0.875	0.875
5	0.085	0.171	0.135	0.927	0.927
6	0.211	0.327	1.000	0.870	0.609
7	0.676	0.705	1.000	0.920	0.900
8	0.341	0.722	0.494	0.808	0.859
9	0.623	0.708	0.815	0.846	0.827
10	0.569	0.645	0.677	0.936	0.912
11	1.000	1.000	1.000	0.933	0.933
12	0.453	0.734	0.574	0.797	0.892
13	0.200	0.539	0.317	0.887	0.887
14	0.263	0.562	0.445	0.817	0.817
15	0.734	0.855	0.808	0.836	0.857
16	0.565	0.734	0.722	0.803	0.826
17	0.628	0.780	0.796	0.722	0.722
18	0.281	0.607	0.506	0.759	0.724
Median	0.509	0.715	0.700	0.850	0.866
Mean	0.494	0.689	0.657	0.845	0.846

5.1 RQ1: Generalization of LogCluster to CE

The performance of LogCluster is reported in Table 3. We observe a median AMI of 0.509 and a wide spread: LogCluster achieves a perfect AMI score on datasets 4 and 11, but scores below 0.22 on datasets 3, 5, 6 and 13. The perfect scores are obtained on small datasets (see Table 1). On datasets 3, 8 and 13, LogCluster achieves relatively high Homogeneity, but very low Completeness. Conversely, dataset 6 yields perfect Completeness but rather low Homogeneity (0.327). Performance on dataset 5 is poor on all accounts.

For over two-thirds of the data, the obtained effort reduction (ER) is slightly below the ideal effort reduction (IER). Exceptions are datasets 6, 7, 9, 10 and 18 where ER exceeds IER, suggesting fewer clusters were created than required by the ground truth.

Trade-offs: Recall that Homogeneity can be seen as a risk measure, reflecting the probability that a cluster element accurately represents all the issues of its cluster. Likewise, Completeness can serve as proxy for effort reduction, as Completeness rewards creating large clusters. Depending on preferences, a user could choose to adopt LogCluster in scenarios where the overall AMI score is low, but either the Homogeneity or Completeness score is high. For dataset 3 for example, a risk-averse user could elect to use LogCluster at a Homogeneity score of 0.676, and accept that the effort reduction achieved is 88% rather than 95%. In this case one basically trades off dealing with a few more clusters for the assurance that they do not contain false positives.

Answer to RQ1: Based on our study using data from our industrial partner, our answer to RQ1 is that the LogCluster approach generalizes to problem identification in continuous deployment logs. Moreover, a user can make a trade-off between risk and effort reduction, for example by choosing to deal with a few more clusters for the assurance that they ate less likely to contain false positives.

5.2 Analysis of LogCluster and its Variations

Our blocked design yields 6426 results for each of the 16 configurations. Figure 2 shows boxplots of the AMI for each configuration. The Friedman test reported a chi square statistic of 19593.623 at a p-value lower than $2.2e - 16$. We thus reject the null hypothesis that the configurations are equally performant at significance level $\alpha = 0.05$, and proceed with the post-hoc analysis.

Our post-hoc analysis consists of $\binom{16}{2} = 120$ paired Wilcoxon Signed-Rank tests: One test for each combination of configurations.

Table 4: Summary of post-hoc comparisons (A v. B) for which the null hypothesis could not be rejected.

A		B					
dimr	crit	dimr	crit	T	VDA_{AB}	VDA_{BA}	p
NMF	Average	PCA	Complete	10026913	0.5411	0.4589	0.0450
NMF	Weighted	PCA	Weighted	10295318	0.5570	0.4430	0.8419

All comparisons were found to be significant after the Holm correction, except for the two comparisons shown in Table 4. Due to the presence of insignificant comparisons we cannot impose a total order on the competing configurations. Table 5 reports the variants by their Rank-Sum groups, including multiple alternatives in the same group when the Wilcoxon Signed-Rank test found no significant differences. We make the following observations from Table 5, which are corroborated by the boxplots in Figure 2:

(1) NMF with Complete Linkage scores significantly better on both mean and median AMI than all other considered configurations.
(2) LSI is the worst performing dimensionality reduction technique. While LSI with complete linkage scores better than *no* dimensionality reduction with weighted or average linkage, this is likely due to complete linkage rather than LSI, as *no* dimensionality reduction with complete linkage performs better.
(3) For any choice of dimensionality reduction technique, Complete linkage is the best performing cluster merge criterion.
(4) Single linkage is the worst performing cluster merge criterion, regardless of dimensionality reduction technique.
(5) Average and Weighted linkage have very similar performance characteristics, with the strongest difference obtained when used in combination with NMF.
(6) For all merge criteria besides single linkage, PCA and NMF obtain a better result than *no* dimensionality reduction and complete linkage, which is LogCluster's configuration [8].

Effect sizes: For each pairwise comparison, we measure the effect size in terms of $VDA = max(A_{12}, A_{21})$. Recall that 118 of the 120 comparisons obtained statistically significant results, with the two remaining comparisons shown in Table 4. The majority of the 118 comparisons indicate small or small-to-medium effects. Concretely, 84 of the 118 significant comparisons yielded small effect sizes at or below 0.64, while the remaining 34 comparisons exceeded this

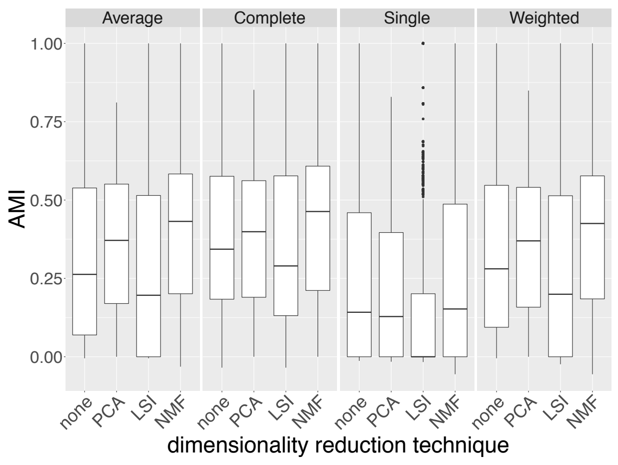

Figure 2: Boxplots of AMI scores for each configuration.

Table 5: Overall performance of the 16 configurations. LogCluster's baseline configuration is indicated in bold.

Groups*	Dim. Red.	HAC	Median AMI	Mean AMI
a	NMF	Complete	0.4634	0.4488
b	NMF	Average	0.4319	0.4287
b	PCA	Complete	0.3990	0.3692
c	NMF	Weighted	0.4252	0.4157
c	PCA	Weighted	0.3698	0.3452
d	PCA	Average	0.3714	0.3537
e	**none**	**Complete**	**0.3432**	**0.3909**
f	LSI	Complete	0.2898	0.3718
g	none	Weighted	0.2806	0.3544
h	none	Average	0.2629	0.3426
i	LSI	Weighted	0.1995	0.3029
j	LSI	Average	0.1963	0.3024
k	NMF	Single	0.1525	0.2719
l	none	Single	0.1420	0.2745
m	PCA	Single	0.1284	0.2037
n	LSI	Single	0.0000	0.1831

* Two configurations belong to the same group if the Wilcoxon Signed-Rank Test could not establish a significant difference between them.

threshold. Unsurprisingly, the large effects sizes occur when one of the six poorest performers (as indicated by Table 5) is compared to one of the ten best performers. When comparing the best performer (NMF + Complete) against the poorest performer (LSI + Single), we measure a VDA score of 0.796 (a large effect). We present the effect sizes of the more performant configurations with respect to the LogCluster baseline (no dimensionality reduction with complete linkage) in Table 6. The improvements over the baseline are statistically significant but modest. The largest effect (a VDA effect size of 0.5687) is obtained when using NMF with complete linkage.

Parameter sensitivity: To analyze the sensitivity of the configurations against variations in the choice of parameters θ and γ, we draw heatmaps of the quality metrics AMI, Completeness and Homogeneity in Figure 3. We show data for the pipelines using PCA, NMF and without dimensionality reduction (all with Complete linkage), and the differences of respectively PCA and NMF with respect to the baseline (no dimensionality reduction). The performance of the original LogCluster approach is indicated by the black box. Higher values are indicated by darker tones. For the difference heatmaps, the positive values (in red) indicated where PCA resp. NMF outperform the baseline, the negative values (in blue) show where the baseline outperforms PCA resp. NMF.

All heatmaps, except those for PCA (3b, 3g and 3l), show that variations in the merge threshold θ have more impact than variations in the relative contrast γ. When not using dimensionality reduction, θ must be in a particular range (roughly $[0.25 - 0.6]$) to

Table 6: Effect sizes (VDA) with respect to the baseline.

Dim. Red.	HAC	T	VDA v. baseline	p
NMF	Complete	6390965	0.5687	< 0.00001
NMF	Average	6917540	0.5448	< 0.00001
NMF	Weighted	7786880	0.5289	< 0.00001
PCA	Weighted	8714866	0.5263	< 0.00001
PCA	Average	8264239	0.5164	< 0.00001
PCA	Complete	8350632	0.5029	< 0.00001

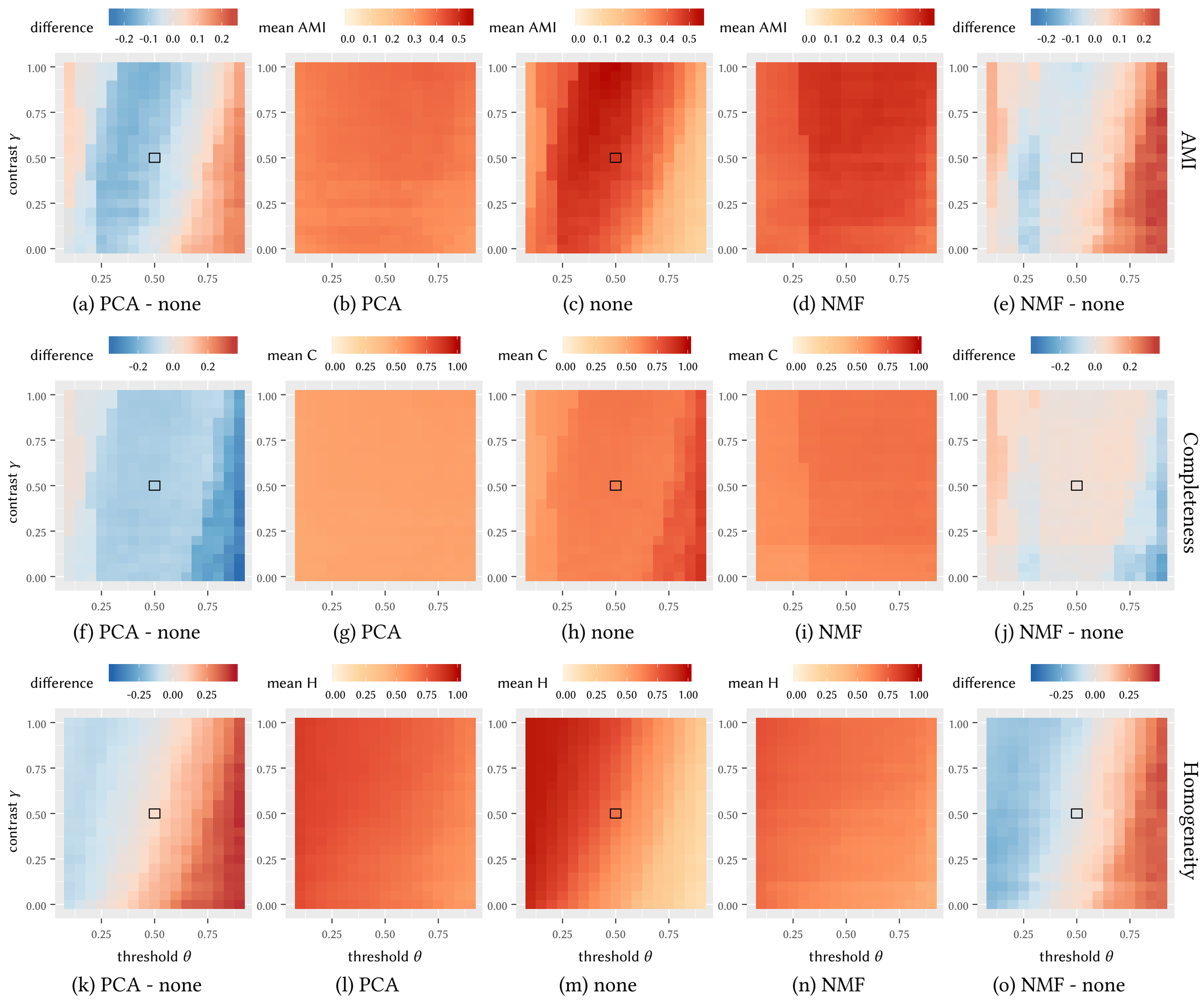

Figure 3: Heatmaps of parameter sensitivity in terms of quality metrics for clustering pipelines using PCA, NMF and without dimensionality reduction (all with Complete linkage) and their differences. LogCluster is indicated by the black box.

achieve an adequate AMI score. The plots also shows that without dimensionality reduction, *increasing* γ is almost always beneficial: The score will either increase or stay the same. A few exceptions can be found for the very lowest values of θ (see especially the AMI scores in 3c). This effect is less pronounced for PCA and NMF.

Moreover, the heatmaps show how Homogeneity and Completeness are competing concerns. This is especially pronounced in the heatmaps without dimensionality reduction (3h and 3m), where a high θ increases Completeness and decreases Homogeneity, and vice versa. We also see the impact of dimensionality reduction on the trade-off between Homogeneity and Completeness: No dimensionality reduction has a relatively stable performance for Completeness, while Homogeneity varies drastically, especially as a function of θ. PCA performs smoothly on both measures, but scores significantly better on Homogeneity than Completeness. NMF, on the other hand, achieves a consistent and high score on both measures.

Overall, both PCA and NMF show a more consistent performance across variations in γ and θ. PCA scores worse than no dimensionality reduction on Completeness (3f), but very closely on Homogeneity. Overall, no dimensionality reduction outperforms PCA in terms of mean AMI (shown by 3a, and confirmed in Table 5). Nevertheless, PCA scores a higher *median* AMI than no dimensionality reduction when using Complete Linkage, and the Wilcoxon Signed-rank test established the two variants as significantly different. NMF clearly beats no dimensionality reduction on all measures, and PCA on most (corroborated by Table 5).

Analysis of LSI performance: As shown in Table 5, LSI performs markedly worse than PCA and NMF. To investigate, we checked if the method for estimating the number of components (see Section 3.1) could be to blame. We record what is the actual explained variance whenever dimensionality reduction is applied. Figure 4

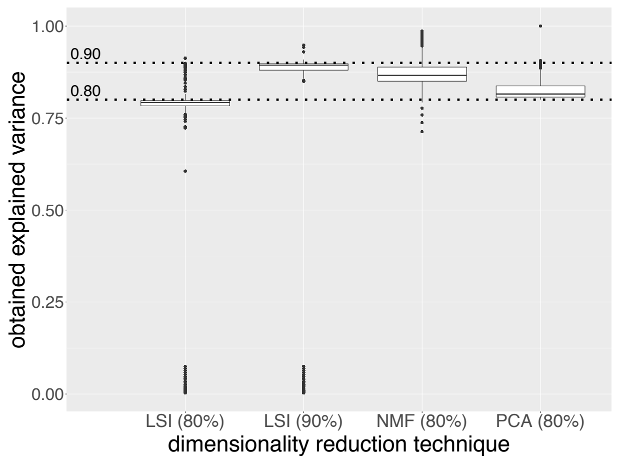

Figure 4: Explained variance after dimensionality reduction. The percentages in labels show desired explained variance.

shows a boxplots of these. We see that a requested explained variance of 80% always results in an actual explained variance of 80% or higher for NMF and PCA. For LSI, however, the requested explained variance systematically is below the requested percentage.

We ran an additional experiment for LSI where we requested a 90% explained variance. As seen in Figure 4, this increased the actual explained variance, but again did not bring it above the requested threshold. This indicates that our PCA-based predictor of the number of components is not appropriate for LSI, which is somewhat unexpected, given that PCA and LSI are both based on Singular Value Decomposition. We leave the search for a better predictor to future work.

As was to be expected, increasing the requested explained variance to 90% did increase LSI's median AMI scores to 0.31 for Complete Linkage, 0.23 for Weighted Linkage, 0.22 for Average linkage, and 0.029 for Single Linkage. However, these still put LSI below the baseline in Table 5 and (far) below PCA and NMF with corresponding merge criteria. As such, our conclusions remain unchanged.

Another potential explanation for LSI's poor performance is that it is geared more towards natural language corpora, which easily have bag-of-words having 20 000 dimensions, and suffer from polysemy and synonymy [10]. Our largest bag-of-events has 850 dimensions (Table 1), and it is unlikely that polysemy and synonymy play a role (unless there is an issue with log abstraction). LSI may just fit better with problems whose dimensionality is at least an order of magnitude greater than what we observe.

5.3 RQ2: impact of dimensionality reduction

Overall, the impact of applying dimensionality reduction in the form of either PCA or NMF is increased robustness: While the baseline scenario requires a sensible choice of θ for adequate performance, especially NMF gives a much wider range of performant parameter choices, as seen in Figure 3c and d.

However, the choice of dimensionality reduction strategy is crucial. As discussed in the previous section, LSI performs worse than the baseline for all merge criteria. While both PCA and NMF provide highly stable performance, the statistical analysis (Table 5) clearly establishes NMF as the superior choice (confirmed by Figure 3).

Answer to RQ2: Our study suggests that using either NMF or PCA yields more performant solutions that are more robust against changes in γ and θ. NMF is clearly superior to PCA for the dataset considered in this study, while LSI performs worse than the baseline.

5.4 RQ3: impact of merge criteria

The results presented in Table 5 and Figure 2 show that Complete Linkage is the best performer for every choice of dimensionality reduction, while Single Linkage is unanimously the poorest performer. Weighted and Average Linkage show roughly similar performance.

The strong performance of Complete Linkage may indicate that our dataset has few outliers, as discussed in Section 3.2. The poor performance of Single Linkage may indicate that a *chaining effect*, a known drawback of Single Linkage [10, Sec. 17.2], may be adversely affecting the results for that criterion. Our current study cannot conclusively establish whether this is indeed the case.

Answer to RQ3: Our study suggests that using Complete Linkage yields the most performant solutions, which is consistent with the choice for this merge criterion in the work of Lin et al. [8] that was the starting point of our investigation.

5.5 Threats to Validity

We identified the following threats that could affect the construct, internal, and external validity of our experimental results.

Threats to Construct Validity: The ground truth used for evaluation was established using regular expressions handcrafted by our industrial collaborator to identify whether a log concerns a known issue. One threat to validity is that these patterns do not cover all issues encountered in the full dataset (which also motivates our research). This is mitigated by creating the ground truth from the subset of data that is fully covered. Another threat is that we can not guarantee or check the absence of matching errors in these patterns. This is, to some extent, mitigated by the fact that they have been used to satisfaction by our partner. A final threat to construct validity is that a hard clustering approach like LogCluster or its variants cannot distinguish between multiple issues in a log file. At best, a new cluster is formed of all logs sharing the same set of issues. This is mitigated by limiting the dataset to logs that match with only one known issue.

Threats to Internal Validity: We implemented and thoroughly tested all algorithms and statistical procedures used in this paper in Python, with the help of widely used libraries such as NumPy and SciPy. However, we can not guarantee the absence of implementation errors which may have affected our evaluation.

Threats to External Validity: We evaluated the applicability and efficacy of the LogCluster algorithm and its variations on logs that resulting from CDt activities at our industrial partner. These logs varied considerably in size and events covered, which should provide a good picture of the behavior that can be expected in various contexts. However, we are likely not to have captured all possible variation, and cannot rule out that different logging practices in other systems or organizations would lead to different results.

Moreover, as discussed in Section 4.1, we are left with a ground truth that is below half the size of the original dataset, which limits the strength of statements that can be made about generalizability.

6 RELATED WORK

We distinguish the following categories of related work:

Need for support to handle CE results: The literature on adopting continuous software engineering practices, such as CI, CDy, and CDt, frequently alludes to the challenges and needs concerning systematic and integrated analysis of the wealth of data resulting from the automated build, test, and deployment steps [1–6].

Hilton et al. [2] interviewed developers from various industries on how they debug test failures detected in CI, and report: *" our participants told us that they get the output logs and start their search there. These output logs can be quite large in size though, [which] can create quite a challenge when trying to find a specific failure."*

Shanin et al. [7] identify the lack of transparency and awareness regarding test and build results as one of the main threats to adopting CDy, together with the need for measures that improve coordination and collaboration on addressing such results.

These findings were echoed in discussions with our industrial collaborator Cisco Systems Norway, who developed an in-house tool to match CDt logs against hand-crafted regular expressions to identify recurring issues. The work described in this paper investigates to what extent this identification can be automatically supported, and what effort reductions can be achieved by clustering similar failing runs. It removes the need for hand-crafting regular expressions by utilizing IR-based clustering techniques and exploiting the contrast between events in passing and failing logs.

Analysis of CE results: Brandtner et al. propose SQA-Mashup [3], a quality awareness framework that integrates information from the entire CI-toolchain. They create a single service to monitor data that would otherwise be scattered over various locations. In subsequent work, the same authors propose SQA-Profiles [4], a rule-based mechanism for the dynamic composition of CI dashboards based on stakeholder activities in the environment.

Nilsson et al. propose CiVIT [27], a technique to visualize testing activities to support CI. They distinguish several types of testing and indicate the coverage for these types. In addition, the scope and periodicity of various testing efforts are used to plot an integrated overall view of the testing status of a system.

Ståhl and Bosch propose Cinders [28], an architecture framework designed specifically to meet the needs of CI and CDy environments. Based on an analysis of CI and CDy literature, they phrase twelve requirements that such an architectural framework should support. Cinders offers four separate viewpoints of the same underlying data model, with six optional layers of additional information which can be used to adjust the focus and level of detail within each of those viewpoints. It uses CiVIT [27] to visualize the testing status.

These approaches all focus on the visual analysis of integrated CI results in dashboards to address the transparency and awareness needs discussed above. The work that we present in this paper is complementary: We focus on integrating CI or CDy results into actionable clusters that can be treated more cost-effectively. Information derived from these clusters could in turn be incorporated in dashboards similar to the ones proposed by these authors.

Analysis of crash reports: Several authors have worked on analyzing the crash reports that modern software systems offer to send to their developers upon detecting a runtime failure.

Podgurski et al. [29] use a combination of supervised and unsupervised pattern classification to cluster software failure reports for prioritization and diagnosis. They first train a classifier to identify the features that distinguish crash reports from normal non-crashing behavior, and then use automated clustering on those features to classify the crash reports. The initial classifier is used as a form of domain-specific dimensionality reduction.

Kim et al. [30] use supervised machine learning to learn the features of top crash reports from previous releases and predict the top potential crashes before a new release is made.

Clustering of system logs: The work presented in this paper explores to what extent techniques developed for the clustering of system logs can be used in a CE context. Oliner et al. [31] give an overview of the advances and challenges in system log analysis.

In Section 2 we already presented an in-depth overview of the LogCluster approach by Lin et al [8] that forms the basis of our investigation. Shang et al. [32] propose an approach that uses the difference between small controlled runs and real-life data from a cloud environment to analyze Hadoop logs. This insight that the contrast between passing and failing runs can be exploited to improve clustering accuracy is also used in LogCluster.

He et al. [33] present an experience report on using system log analysis for anomaly detection. They describe six state-of-the-art log-based anomaly detection methods but do not draw conclusions on which method performs best. Their focus on anomaly detection is different from ours, as we aim to group logs that fail for the same underlying issue in order to handle those issues more efficiently.

Parameter Tuning: Recent research highlighted that the successful application of machine learning and data mining algorithms in concrete domains is highly impacted by the selection of their configuration parameters [34]. We are not aware of any other work in a software engineering context that shares our goal of investigating the impact of particular dimensionality reduction techniques or clustering parameters such as the linkage criterion on the quality of clustering software logs or crash reports.

7 CONCLUDING REMARKS

7.1 Contributions

Our study considers the challenge of automatically grouping logs of runs that failed for the same underlying reasons, so that they can be treated more effectively. We replicate and extend earlier work on clustering system log files [8] to assess its efficacy for the analysis of continuous deployment logs. We consider optional inclusion of one of three dimensionality reduction techniques: Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), and Non-negative Matrix Factorization (NMF). Moreover, we consider three alternative cluster merge criteria (Single Linkage, Average Linkage, and Weighted Linkage), in addition to the Complete Linkage criterion used in the earlier work. We empirically evaluate the 16 resulting configurations on continuous deployment logs provided by our industrial collaborator to answer the following three research questions:

(1) Can the application of LogCluster [8] be generalized to identifying problems in *continuous deployment log files*?
(2) To what extent does applying dimensionality reduction impact the results of automated log clustering?

(3) To what extent does the merge criterion in HAC impact the results of automated log clustering?

The study results allow us to answer these questions as follows:

(*ad* 1) The LogCluster approach generalizes to problem identification in continuous deployment logs.
(*ad* 2) Including an NMF-based dimension reduction step results in significantly better overall performance and robustness.
(*ad* 3) Complete Linkage performs best of all the merge criteria that were considered.

Conclusion: We conclude that problem identification via automated log clustering is improved by including dimensionality reduction, as it decreases the pipeline's sensitivity to parameter choice, thereby increasing its robustness for handling different inputs, a desirable characteristic for an unsupervised approach such as automated log clustering [11].

7.2 Future Work

We see several directions in which this work can be extended. First and foremost, these techniques should be evaluated on a wider range of case studies. A challenge in that respect is establishing a good ground truth for the qualitative evaluation of results, as fully labelled sets of log files are rare. One option to address this challenge could be to programmatically synthesize labelled log files with known characteristics.

Another area for future work is in finding better predictors for the number of components used by LSI, and possibly NMF. Although our intuition to use the amount needed by PCA as predictor for NMF worked well, it underperformed for LSI, even though both PCA and LSI are based on singular value decomposition.

A final direction for future work is (support for) the in-depth analysis of the cluster merging behavior using dendrograms and merge distances. Such an analysis could help understand why some of the merge criteria (like Single Linkage) performed poorly, and why others (such as Complete Linkage) performed well, despite their known pitfalls.

ACKNOWLEDGMENTS

We thank Marius Liaaen and Thomas Nornes of Cisco Systems Norway for extensive discussions, help with obtaining and understanding the data set, and for developing the log abstraction mechanisms. This work is supported by the Research Council of Norway through the Certus SFI (#203461/030).

REFERENCES

[1] E. Laukkanen, J. Itkonen, and C. Lassenius. "Problems, causes and solutions when adopting continuous delivery - A systematic literature review." In: *IST* 82 (2017), pp. 55–79.

[2] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig. "Trade-offs in continuous integration: assurance, security, and flexibility." In: *Joint Meeting of the European Software Engineering Conference and the Symp. Foundations of Softw. Engineering.* ACM, 2017, pp. 197–207.

[3] M. Brandtner, E. Giger, and H. Gall. "SQA-Mashup: A mashup framework for continuous integration." In: *IST* 65 (2015), pp. 97–113.

[4] M. Brandtner, S. C. Muller, P. Leitner, and H. C. Gall. "SQA-Profiles: Rule-based activity profiles for Continuous Integration environments." In: *Int'l Conf. Softw. Analysis, Evolution, and Reengineering.* IEEE, 2015, pp. 301–310.

[5] A. Debbiche, M. Dienér, and R. Berntsson Svensson. "Challenges When Adopting Continuous Integration: A Case Study." In: *Lecture Notes in Computer Science (LNCS).* Vol. 8892. Springer, 2014, pp. 17–32.

[6] H. H. Olsson, H. Alahyari, and J. Bosch. "Climbing the Stairway to Heaven - A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software." In: *Euromicro Conf. Softw. Engineering and Advanced Applications.* IEEE, 2012, pp. 392–399.

[7] M. Shahin, M. Ali Babar, and L. Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices." In: *IEEE Access* 5 (2017), pp. 3909–3943.

[8] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen. "Log Clustering Based Problem Identification for Online Service Systems." In: *Int'l Conf. Software Engineering - Softw. Engineering in Practice.* ACM, 2016, pp. 102–111.

[9] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis." In: *Int'l Conf. Data Mining.* IEEE, 2009, pp. 149–158.

[10] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[11] C. C. Aggarwal. *Data Mining.* Springer, 2015.

[12] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. "What is the nearest neighbor in high dimensional spaces?" In: *26th Int'l Conf. Very Large Data Bases.* 2000, pp. 506–515.

[13] T. Korenius, J. Laurikkala, and M. Juhola. "On principal component analysis, cosine and Euclidean measures in information retrieval." In: *Information Sciences* 177.22 (2007), pp. 4893–4905.

[14] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *J. Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] M. Zitnik and B. Zupan. "Nimfa: A Python Library for Nonnegative Matrix Factorization." In: *J. Machine Learning Research* 13 (2012), pp. 849–853.

[16] D. Müllner. "Modern hierarchical, agglomerative clustering algorithms." ArXiv: 1109.2378 [stat.ML]. 2011.

[17] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python.* 2017.

[18] N. X. Vinh, J. Epps, and J. Bailey. "Information theoretic measures for clusterings comparison." In: *Int'l Conf. Machine Learning (ICML).* ACM, 2009, pp. 1073–1080.

[19] A. Rosenberg and J. Hirschberg. "V-measure: A conditional entropy-based external cluster evaluation measure." In: *Joint Conf. empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL).* Association for Computational Linguistics, 2007, pp. 410–420.

[20] M. Friedman. "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance." In: *J. American Statistical Association* 32.200 (1937), pp. 675–701.

[21] J. Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *J. Machine Learning Research* 7 (2006), pp. 1–30.

[22] F. Wilcoxon. "Individual Comparisons by Ranking Methods." In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83.

[23] A. Benavoli, G. Corani, and F. Mangili. "Should we really use post-hoc tests based on mean-ranks?" In: *J. Machine Learning Research* 17.5 (2016), pp. 1–10.

[24] J. W. Pratt. "Remarks on Zeros and Ties in the Wilcoxon Signed Rank Procedures." In: *J. American Statistical Association* 54.287 (1959), p. 655.

[25] S. Holm. "A Simple Sequentially Rejective Multiple Test Procedure." In: *Scandinavian J. Statistics* 6.2 (1979), pp. 65–70.

[26] A. Vargha and H. D. Delaney. "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong." In: *J. Educational and Behavioral Statistics* 25.2 (2000), pp. 101–132.

[27] A. Nilsson, J. Bosch, and C. Berger. "Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study." In: *Agile Processes in Softw. Engineering and Extreme Programming.* Springer, 2014, pp. 171–186.

[28] D. Ståhl and J. Bosch. "Cinders: The continuous integration and delivery architecture framework." In: *IST* 83 (2017), pp. 76–93.

[29] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, Jiayang Sun, and Bin Wang. "Automated support for classifying software failure reports." In: *Int'l Conf. Softw. Engineering.* Vol. 6. IEEE, 2003, pp. 465–475.

[30] D. Kim, X. Wang, S. Kim, A. Zeller, S. C. Cheung, and S. Park. "Which crashes should i fix first?: Predicting top crashes at an early stage to prioritize debugging efforts." In: *IEEE TSE* 37.3 (2011), pp. 430–447.

[31] A. Oliner, A. Ganapathi, and W. Xu. "Advances and challenges in log analysis." In: *Communications of the ACM* 55.2 (2012), pp. 55–61.

[32] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin. "Assisting developers of Big Data Analytics Applications when deploying on Hadoop clouds." In: *Int'l Conf. Softw. Engineering.* IEEE, 2013.

[33] S. He, J. Zhu, P. He, and M. R. Lyu. "Experience Report: System Log Analysis for Anomaly Detection." In: *Int'l Symp. Softw. Reliability Engineering.* IEEE, 2016, pp. 207–218.

[34] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook.* Springer, 2010, p. 1383.