

# Reducing Regression Test Suites using the Word2Vec Natural Language Processing Tool

Bahareh Afshinpour<sup>a</sup>, Roland Groz<sup>a</sup>, Massih-Reza Amini<sup>a</sup>, Yves Ledru<sup>a</sup> and Catherine Oriat<sup>a</sup>

<sup>a</sup>Université Grenoble Alpes, Laboratoire d'Informatique de Grenoble (LIG), Bâtiment IMAG, F-38058 Grenoble Cedex 9, France

## Abstract

Continuous integration in software development requires to run the tests on a regular basis to ensure that the code does not regress. So that the execution time of the regression test suite remains reasonable its size must be reduced while preserving its fault detection capability. From test execution logs, we extract from each test a trace, which is a sequence of events. We then consider each event as a word, and apply natural language processing methods, here the Word2Vec tool, to detect similarities in sentences, and partition them into clusters. We thus can reduce the regression test suite by removing redundant tests. We present the approach on a small case study, and we use mutation based testing to assess the effectiveness of the reduction.

## Keywords

software testing, test suite reduction, software logs, regression testing, mutation testing

## 1. Introduction

Software testing is one of the most time-consuming and highly priced steps in software development specifically for large systems. It accounts for more than 52 percent of total software development budget and its fault detection coverage is directly connected with the quality assurance of the product [1]. Therefore, any effort in optimizing test process can save time and budget and increases the product quality. These, in turn, will ease software testing of large systems, shorten the time-to-market gap and increase the profit.

During the software testing process, the product may go through regression tests which include several recurring test-and-debug steps to ensure that the software still performs after modification the same functionality as originally. A large number of test traces can be gathered either from automated testing or from user traces. Test Suite Reduction (TSR) helps to reduce and purify the test cases by removing redundant and ineffective tests [2]. An automated TSR helps to reduce the human interaction in error case debugging, since the process leaves fewer test cases to be investigated by humans. During the last decade, several automated TSR methods have been proposed and recently Machine Learning (ML) has extended its realm to software testing and TSR.


---

*Joint Proceedings of SEED & NLPaSE co-located with APSEC 2020, 01-Dec, 2020, Singapore*

✉ bahareh.afshinpour@univ-grenoble-alpes.fr (B. Afshinpour); roland.groz@univ-grenoble-alpes.fr (R. Groz); massih-reza.amini@univ-grenoble-alpes.fr (M. Amini); Yves.Ledru@imag.fr (Y. Ledru); Catherine.Oriat@imag.fr (C. Oriat)



© 2020 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

In this paper, we propose an approach to TSR based on identifying similarities between tests with techniques from Natural Language Processing that can identify similarities between sentences. More specifically, we use Word2Vec [3], associated to other ML approaches (dimensionality reduction, clustering) in order to reduce a test suite.

Despite many of TSR methods which needs to run the test traces to decide on keeping or removing them, the proposed method regards the traces like natural language sentences, by separating and removing similar and redundant events. Therefore, it does not need to run the traces which eliminates the need to access to the software-under-test and makes the entire test reduction processes faster.

We test them on a Supermarket Scanner (Scanette) case study. In order to assess the fault coverage of a reduced test suite, we use mutation-based testing which has become mature and is being applied more and more both in research and industry [4]. Based on this approach, the source code of the Scanette software is artificially mutated. For that software, we had a source of handmade mutations that were known to provide a good assessment of fault coverage. Each mutation injects a bug into the Scanette software. For that case study, we also had 3 reference test logs of varying length, one of them from a (handwritten) functional test suite, and the others collected from random testing. We apply our method to assess its results on that case study.

## 2. Related Work

Recently, some similarity based approaches have been proposed for TSR, which generally try to find similar test cases and remove redundancy [5]. Also, applying clustering-based approaches for TSR has received a deal of attention [6, 7]. Reichstaller *et al.* [8] used two clustering techniques, *Affinity Propagation* and *Dissimilarity-based Sparse Subset Selection* to reduce test suite in a mutation-based scenario. Felbinger *et al.* [9] employed decision trees to build a model from the the test elements and remove those that does not change the model. In addition, classic machine learning approaches (*e.g.* Random Forest, GBT, and SVM) are employed for this purpose [10].

The mentioned methods generally focus on the combination of the test elements and do not consider the order and vicinity of the elements in a test trace. In our study, we decided to adapt natural language processing methods, and specifically Word2Vec [3] in order to be able to effectively take into the account the combination and order of the events in test traces. For this purpose, the proposed method processes test traces as sentences in a natural language and builds a model based on the sequence of the words in each sentence.

## 3. The Software Under Test

A barcode scanner (nicknamed "Scanette" in French) is a device used for self-service checkout in supermarkets. The customers (shoppers) scan the barcodes of the items which they aim to buy while putting them in their shopping basket. The shopping process starts when a customer (client) **unlocks** the Scanette device. Then the customer starts to **scan** the items and adds them

Index	Time	Session ID	Object	Action	Input	Output
51,	1585070116817,	client6,	scan12,	unlock,	[],	0
52,	1585070116819,	client0,	scan0,	scan,	[3270190022534],	0
53,	1585070116820,	client1,	cashier1,	CloseSession,	[],	0
54,	1585070116820,	client2,	cashier2,	add,	[3570590109324],	0
55,	1585070116824,	client5,	scan5,	scan,	[8718309259938],	0
56,	1585070116825,	client6,	scan12,	scan,	[3560070139675],	0
57,	1585070116837,	client0,	scan0,	scan,	[3560070048786],	0
58,	1585070117030,	client6,	scan12,	scan,	[7640164630021],	-2
59,	1585070117073,	client6,	scan12,	delete,	[7640164630021],	-2
60,	1585070116838,	client1,	cashier1,	pay,	[353.06],	0
61,	1585070116839,	client2,	cashier2,	CloseSession,	[],	0
62,	1585070116840,	client3,	cashier3,	add,	[3570590109324],	0
64,	1585070117687,	client6,	scan12,	transmission,	[caisse6],	0
65,	1585070117687,	client6,	scan12,	abandon,	[],	?
66,	1585070117701,	client6,	cashier4,	OpenSession,	[],	0
67,	1585070116855,	client0,	scan0,	transmission,	[cashier0],	0
68,	1585070116855,	client0,	scan0,	abandon,	[],	?
69,	1585070117716,	client6,	cashier4,	add,	[7640164630021],	0
70,	1585070117731,	client6,	cashier4,	CloseSession,	[],	0
71,	1585070117747,	client6,	cashier4,	Pay,	[260],	9.11

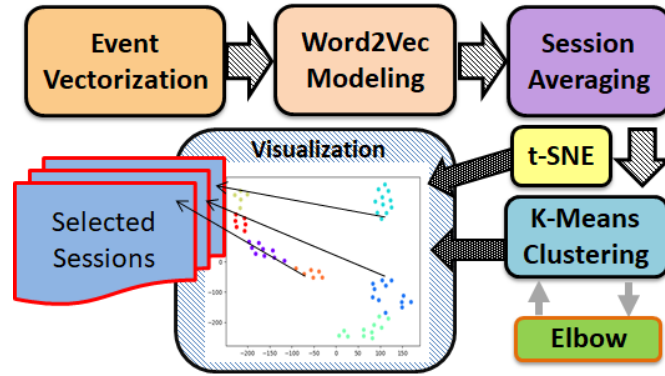
**Figure 1:** Test traces of the Scanette case study

to his/her basket. Later customers may decide to **delete** the items and put them back in their shelves. Among the scanned items, there may be barcodes with unknown prices. In this case, the scanner adds them to the basket and they will be processed later by the cashier, before the payment at checkout. The customer finally refers to the checkout machine for payment. From time to time, the cashier may perform a “control check” by re-scanning the items in the basket. The checkout system then **transmits** the items list for payment. In case that unknown barcodes exist in the list, the cashier controls and resolves them. The cashier has the ability to **add** or **delete** the items in the list. At the final step, the customer **abandons** the scanner by placing it on the scanner board and finalizes his purchase by **paying** the bill.

The Scanette system has a Java implementation for development and testing and a Web-based graphical simulator for illustration purpose. The web-based version emulates costumers’ shopping and self-service check-out in a supermarket by a randomized trace generator derived from a Finite-State Machine. The trace logs of the Scanette system contain interleaved actions from different customers who are shopping concurrently. Each customer has a unique session ID which distinguishes his/her traces from another customer. Figure 1 shows a snippet of a trace log with different actions from different sessions. Each event in the test (trace) has an index and time-stamp which are stored chronologically. The Session ID determines to which user (or session) the event belongs. In this figure, we can find activities of *Client6*, interleaved with other clients’ actions, which starts by an ‘unlock’ and ends by a ‘pay’ action. The triplet of action-input-output obviously shows which action is called, what is given as its input and what is obtained as the output of the action. Some actions may also include a parameter, such as the actual value of the barcode of an item that is scanned.

To artificially inject faults, the source code of the Scanette software is mutated with 49 *mutants*, all made by a modification on the source code by hand.

We needed a few test suites to be used as the test bench for the proposed method. Hence, we have created three test suites with different number of traces: 1026, 100043 and 200035. We will call them as 1026-event, 100043-event and 200035-event names, respectively. They include shopping steps of different number of clients (sessions). They were created as random test suites



**Figure 2:** Flow chart of the proposed session reduction approach

by a generator of events that simulates the behaviour of customers and cashiers. The goal of the proposed TSR methods is to reduce the number of traces needed to kill the same mutants as the original test suite can kill. In the rest of this paper, *session* and *client* are equivalent.

## 4. The Proposed Method

Here we explain the steps for the proposed TSR method. The input of the proposed method is a test suite comprised of several clients' sessions, each of which contains a trace of client's actions. The goal is to remove redundant and ineffective sessions so as to have a considerably lower number of sessions which have the same effect as the original test suite.

We pursue these steps, which will be discussed in turn: session vectorization, model creation by the Word2Vec method [3], session averaging, sessions clustering and finally, session selection which is depicted in Figure 2.

### 4.1. Overview of Event and Session Vectorizations

We perform abstraction and vectorization at several levels. In an initial step, we cut a trace that records interleaved sessions of different independent customers into a set of sessions, each one for a unique customer. Then, each event of a trace is converted to a triplet that captures the relevant features of the event. As a session is a sequence of events, it becomes a sequence of triplets. In a second step, we associate a vector to each triplet using the Word2Vec vectorization process, as described in section 4.3. The dimension of those vectors is the number  $n$  of different triplets that appear in the trace. Each session of length  $L$  is therefore associated to a sequence of  $L$  vectors of size  $n$ . Finally, we associate a single vector of size  $n$  to a session by using session averaging as described in section 4.4. But as  $n$  would be too high a dimension in most cases, we first reduce the dimensionality using the t-SNE method before proceeding to clustering.

### 4.2. Event abstraction

In this section, we describe the association of triplets to events. We decided to differentiate among similar events which have different inputs and outputs because every combination of

action-input-output may show different behavior of the system. For example, a ‘scan’ action with an error output code should be treated differently from the same action with the success output code. For this purpose, each action-input-output was coded as a triplet vector like  $[a, p, o]$ , in which ‘ $a$ ’ is the index of the action from set  $A$  which itself includes  $n$  possible actions denoted by  $A_1$  to  $A_n$ . Likewise, ‘ $p$ ’ is the index of the input parameter from set  $P$ , containing all possible input parameters from  $P_1$  to  $P_m$  and finally ‘ $o$ ’ is the index of the output parameter in set  $O$  including a list of all possible outputs, from  $O_1$  to  $O_k$ .

$$\begin{aligned}
 A &= \{A_1, A_2, A_3, \dots, A_n\}, \\
 &A_1 : unlock, A_2 : scan, A_3 : add, \dots \\
 P &= \{P_1, P_2, \dots, P_m\}, \\
 &P_1 : barcode, P_2 : null[], P_3 : floatnumber, \dots \\
 O &= \{O_1, O_2, \dots, O_k\} \\
 &O_1 : 0, O_2 : -2, \dots
 \end{aligned}$$

Therefore, we can encode all actions/inputs/outputs triplets. Note that we still abstract from the timestamp and from the parameters of the action (typically, the actual barcode of a shop item): the actual delay between actions is not relevant, only the ordering of events should be kept. Similarly, the exact choices of items picked by a customer are irrelevant for the logic of the application. We can now display each session as a sequence of triplet vectors. You can see three different clients’ sessions below. Each session has been printed in form of triplets:

```

client60: '['[0,0,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';
'[2,2,0]';[3,0,2]';[4,4,3]']
client40: '['[0,0,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';
'[1,1,0]';[2,2,0]';[3,0,2]';[4,4,3]']
client17: '['[0,0,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';[1,1,0]';
'[1,1,0]';[2,2,0]';[3,0,2]';[4,4,3]']

```

### 4.3. Word2Vec Model Construction

Word2Vec [3] is a Neural Networks based method for learning a representation of words appearing in the documents of a collection. The output of this approach is a vector representation of each word. In different studies, it has been shown that the distance between each pair of words translates their semantic relation. From word representation, a representation of each sentence in a given document of the collection can be induced by for example averaging the vector representations of all words that are present in the sentence. In our work, since we need to cluster/merge similar sessions into a fewer number of sessions which can trigger the same errors (or "kill the same number of mutants"), we chose Word2Vec to find semantically similar sessions by treating test traces as sentences. The Word2Vec clustering method may tell us that some sessions are equivalent or very close to each other, although their actions and inputs/outputs are different. For this purpose, first, we calculate a measure for each triplet and use Word2Vec method to learn its vector representation. It should be noted that we store each triplet as a string (e.g: '[1,0,1]') and we treat them like words in natural language processing. These are two 15-element vectors created by Word2Vec method and represent [1,1,0] and [3,0,2]

triplets.

[1, 1, 0] :

[ 1.2445878, 1.613417, -0.1642392, 3.0873055, -0.355896 ,  
1.0599929, -0.49392796, 1.0838877, -1.1861929, -0.2639794,  
-0.09810112, -0.9824149, 0.881457, -3.6238787, -1.1903458 ]

[3, 0, 2] :

[ 0.17494278, 0.15232983, -0.14811908, -1.5120562, -0.0818198,  
-0.35962805, -0.65130717, -0.18931173, 0.85284257, -0.23423576,  
0.8646087, 0.41952062, 0.5157884, 1.593384, 0.50375664]

#### 4.4. Session Averaging

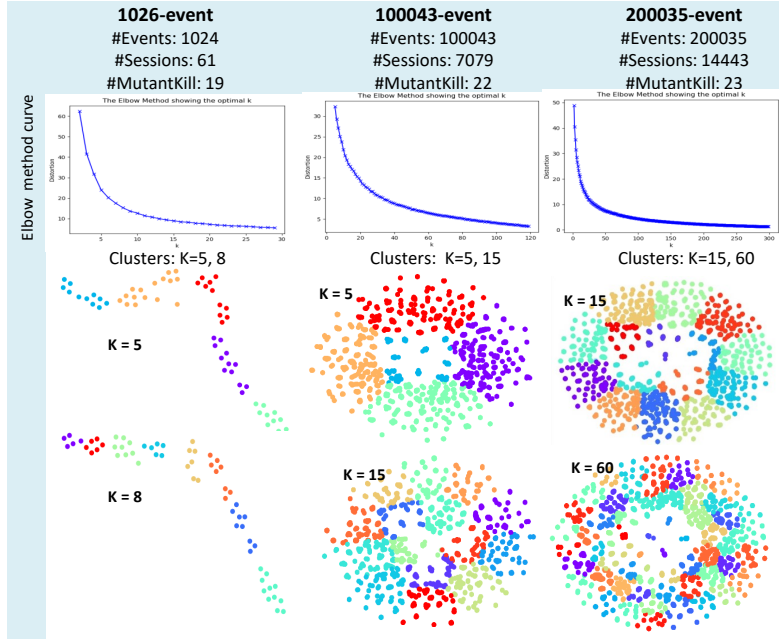
Having a vector representation for each of the sessions, in the next step, we need to have a measure for different sessions in order to be able to compare them and find their similarity. There are different methods to get the session vectors [11, 12, 13, 14]. In fact, a common method to achieve sentence representations is to average word representations. Vector averaging has been effective in some applications [15]. Averaging over word vectors in a sentence was shown to be an effective method for sentence representation. The authors in [16] studied three supervised NLP tasks and observed that, in two cases including sentence similarity, averaging could achieve better performance in comparison to LSTM. To this end, we compute an average of the Word2vec vectors in each session. It could be simply an element-wise average of  $n$ -element vectors that finally leaves us an  $n$ -element average of the words in a sentence. In the end, we have a single vector measure for each session. In our experiments, we used the element-wise arithmetic mean to compute the averaged measure for a session. Hence, each session is associated to a single vector of size  $n$ .

#### 4.5. Session Clustering & Selection

Traditional approaches to data analysis and visualization often fail in the high dimensional setting, and it is common to perform dimensionality reduction in order to make data analysis tractable and meaningful [17]. To this end, we applied the t-SNE algorithm [18] in order to reduce the initial dimension of the vector space to 2, as it has been proved that this method successfully maps well separated disjoint clusters from high dimensions to the real line so as to approximately preserve the clustering. Also, dimension reduction is shown to be effective in some clustering case studies [18]. We will show the effect of the dimension reduction on the results later in the next Section.

After dimension reduction, we apply the K-means algorithm for clustering the sessions in the reduced space of dimension 2 and employ the Elbow technique [19] to estimate the optimal number of clusters. Since t-SNE and K-means choose random initial points, we repeated each experiments two times and chose the best result.

For test selection, we decided to keep one representative from each cluster and see how many mutants they can kill. For this purpose, we select from each cluster, the client with the highest number of events. Experimentally, this client kills more mutants than shorter ones, as can be expected.



**Figure 3:** Elbow method curve and clustering visualization in finding optimal number of clusters

**Table 1**

Optimal number of clusters for the 1026-event Scanette case study

K	#Events	Sessions Numbers	Killed mutants	Killed Mutant IDs
2	36	[ 30, 24]	9	[7, 12, 17, 19, 20, 21, 25, 26, 42]
3	67	[ 30, 23, 35]	17	[0, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44]
4	81	[ 30, 6, 10, 24]	10	[7, 12, 17, 19, 20, 21, 25, 26, 42, 48]
5	104	[ 30, 6, 10, 24, 23]	18	[0, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]
6	117	[ 30, 28, 23, 6, 24, 10]	18	[0, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]
7	128	[ 6, 27, 23, 10, 24, 30, 28]	18	[0, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]
8	147	[ 27, 52, 23, 10, 24, 6, 30, 28]	18	[0, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]
9	171	[ 10, 28, 19, 52, 27, 24, 6, 30, 23]	18	[0, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]
10	194	[ 28, 22, 23, 26, 35, 24, 30, 10, 6, 33]	19	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]
11	214	[ 23, 52, 22, 35, 30, 24, 6, 27, 33, 28, 10]	19	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]

## 5. Results

Here we explain the test reduction results on the Scanette case study. We applied our approach on three different test suites introduced in section 3. W2V model creates a vocabulary for each of them. The number of W2V vocab for 1026-event test suite is 15: this corresponds to the number of different triplets. It has 1026 events from 61 shopping sessions. The entire test suite can kill 19 mutants and the goal was to reduce events while maintaining the same fault detection capability viz. killing the same number of mutants. The second test suite, 100043-event, has 100043 events from 7079 shopping sessions. The number of W2V vocab for this test suite is 18 vocab. It can kill 22 mutants. And the third test suite, 200035-event has 200035 events from 14443 shopping sessions that can kill 23 mutants. The number of W2V vocab for this test suite is 20 vocab. These numbers are summarised in Figure 3 on the first row.

The sessions were vectorized and their W2V models were constructed. After applying t-SNE



**Table 2**

Optimal number of clusters for the 100043-event and 20035-event Scanette case studies

100043-event			20035-event		
K	#Events	Killed mutants	K	#Events	Killed mutants
16	424	20	15	234	10
18	467	22	30	443	18
22	551	22	60	912	21
30	709	22	73	1111	23

**Table 3**

The effectiveness of using t-SNE method for the 200035-event test suite

<i>with t-SNE</i>				
# Events	K	Killed mutants	Mutant IDs	
234	15	10	[1, 7, 12, 17, 20, 21, 25, 26, 42, 48]	
443	30	18	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 37, 42, 43, 45, 48]	
912	60	21	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 43, 44, 45, 48]	
1111	70	22	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 38, 41, 42, 43, 44, 45, 48]	
<i>without t-SNE</i>				
# Events	K	Killed mutants	Mutant IDs	
236	15	9	[1, 7, 12, 17, 20, 21, 25, 26, 42]	
470	30	11	[1, 7, 12, 17, 19, 20, 21, 25, 26, 42, 48]	
901	60	19	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]	
998	70	19	[0, 1, 7, 9, 10, 12, 17, 19, 20, 21, 25, 26, 34, 35, 37, 41, 42, 44, 48]	

and averaging sessions, we employed Elbow method to find the optimal number of clusters. The Elbow method curve of each test suite is shown in Figure 3 in the second row. These figures also provide two samples of clustering for two different number of clusters ( $K$ ) around the proposed range by the Elbow method. We can observe that for the 1026-event test suite, the sessions are distributed in some distinct clusters. As the number of events and sessions increases, the projected model tends to make a circle with some singular points in the middle. Specifically for the 200035-event test suite for  $K = 60$ , in the middle of the circle, there were some sessions (points) that conveyed different clients' behavior which is to say that their series of action were rare in comparison to the other sessions around the circle.

To observe the effect of optimal number of clustering, we examined the K-means clustering from 2 to 10 clusters for the 1026-event test suite and from each cluster, the longest session was chosen. The selected sessions were executed again to see how many mutants they kill all together. Table 1 conveys these results. It can be seen that when  $K = 10$ , 10 sessions chosen



from 10 clusters can kill all 19 mutants (highlighted by green color) that the original 1026-event test suite can kill. In this case, 194 client actions are enough and the remaining 832 actions (1026-194) can be removed. By a simple ratio, the amount of reduction is %81. For  $K < 5$ , the number of killed mutants are unstable and less than 19. We have shown them by yellow and pink colors. This table also provides more details on the number of selected sessions and the number of killed mutants.

The same results for two other test suites, namely 100043-event and 200035-event are presented in Table 2. For the 100043-event test suite, only 467 events from 18 client sessions are enough to kill all 22 mutants. Therefore, the proposed TSR succeeded in removing more than 99% of the redundant traces. For the 200035-event test suite, 1111 events from 73 sessions are enough to have the same fault detection effect. Again the same success rate is achieved.

Finally, the effectiveness of using the t-SNE method to reduce dimensions can be observed in table 3 which shows the number of killed mutants with and without using t-SNE for the 200035-event test suite. With the same number of clusters ( $K$ ), in all cases, using t-SNE effectively improves the number of killed mutants. This comes from the fact that t-SNE preserves and normalizes the features of each dimension when it merges them together.

From the experiments, we can conclude that treating user actions like words in sentences and building a Word2Vec+t-SNE model from them can effectively preserve users' behavior and extract their features. Applying clustering method can group similar user sessions and in turn enables us to remove redundant sessions which was the TSR goal of our case study.

## 6. Conclusion and Future Work

This preliminary experiment on a simple case study shows that using a technique from NLP, namely Word2Vec, seems to provide a valuable tool for the analysis of similarity between tests in a software testing contexts. As we work on traces, it also shows a potential for reducing the information to analyze lengthy software logs.

### 6.1. Summary of findings on our case study

- Word2Vec can yield meaningful feature sets for software log events.
- Using an average of the vectors of the words in a sequence associates a relevant measure for clustering software sessions made of sequences of events.
- t-SNE improves clustering and the quality of clusters of tests.
- Quite significant test suite reduction from random test suites can be achieved by clustering with Word2Vec associated to t-SNE and the Elbow method, with no loss of fault detection capabilities.

### 6.2. Threats to validity

There are obvious limits to generalizing those preliminary results.

- We just address a single case study, that is not too complex.

- We consider only test traces from random testing. Random testing is indeed an important approach and often considered a touchstone in software testing, but we plan to consider also other sources of tests, such as carefully handcrafted tests, conformance tests, tests from DevOps approaches...
- More parameters of the method should be investigated, such as the selection of representatives from each cluster, the influence of the initial abstraction of events etc.

### 6.3. Future Work

The visualized results in Figure 3 motivates using spectral clustering instead of K-Means. While K-Means takes into account the distance and closeness, spectral clustering considers connectivity of the points. For the continuation of this work, we are going to apply these clustering approaches and compare it with K-Means.

Moreover, session averaging can be replaced by a more insightful method which can preserve the order of the events in a session. We plan to replace session averaging with Paragraph Vectors [12] to achieve a representative vector from each session. Therefore, regarding the TSR goal of this paper, we expect to see better results by considering the order of events.

We have also started investigating another direction for further reducing a test suite. Notice that we selected the longest test (session) from each cluster. However, it is often the case that not all events of such a test are necessary to achieve fault detection. There can be many redundant events that could be removed to keep the core fault triggering capabilities of the test. We are developing an analysis of the relation between events that can trigger a fault and the necessary enablers that precede them.

## Acknowledgments

This work was supported in part by the French National Research Agency: PHILAE project (N° ANR-18-CE25-0013). We are also grateful to our partners in that project for their feedback: U. of Bourgogne Franche-Comté (lab Femto-ST), Orange labs, Smartesting, and U. of Queensland (Mark Utting).

## References

- [1] X. Zhang, H. Shan, J. Qian, Resource-aware test suite optimization, in: 2009 Ninth International Conference on Quality Software, IEEE, 2009, pp. 341–346.
- [2] A. Shi, A. Gyori, M. Gligoric, A. Zaytsev, D. Marinov, Balancing trade-offs in test-suite reduction, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 246–256.
- [3] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781 (2013).
- [4] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, M. Harman, Mutation testing advances: an analysis and survey, in: Advances in Computers, volume 112, Elsevier, 2019, pp. 275–378.

- [5] E. Cruciani, B. Miranda, R. Verdecchia, A. Bertolino, Scalable approaches for test suite reduction, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 419–429.
- [6] C. Coviello, S. Romano, G. Scanniello, Poster: Cuter: Clustering-based test suite reduction, in: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), IEEE, 2018, pp. 306–307.
- [7] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, G. Antoniol, A. Corazza, Clustering support for inadequate test suite reduction, in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2018, pp. 95–105.
- [8] A. Reichstaller, B. Eberhardinger, H. Ponsar, A. Knapp, W. Reif, Test suite reduction for self-organizing systems: a mutation-based approach, in: Proceedings of the 13th International Workshop on Automation of Software Test, 2018, pp. 64–70.
- [9] H. Felbinger, F. Wotawa, M. Nica, Test-suite reduction does not necessarily require executing the program under test, in: 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, 2016, pp. 23–30.
- [10] M. R. Naeem, T. Lin, H. Naeem, H. Liu, A machine learning approach for classification of equivalent mutants, *Journal of Software: Evolution and Process* 32 (2020) e2238.
- [11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: *Advances in neural information processing systems*, 2013, pp. 3111–3119.
- [12] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: *International conference on machine learning*, 2014, pp. 1188–1196.
- [13] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, A. Bordes, Supervised learning of universal sentence representations from natural language inference data, *arXiv preprint arXiv:1705.02364* (2017).
- [14] S. Arora, Y. Liang, T. Ma, A simple but tough-to-beat baseline for sentence embeddings (2016).
- [15] J. Mitchell, M. Lapata, Composition in distributional models of semantics, *Cognitive science* 34 (2010) 1388–1429.
- [16] J. Wieting, M. Bansal, K. Gimpel, K. Livescu, Towards universal paraphrastic sentence embeddings, *arXiv preprint arXiv:1511.08198* (2015).
- [17] G. Linderman, S. Steinerberger, Clustering with t-sne, provably, *SIAM J. Math. Data Sci.* 1 (2019) 313–332.
- [18] L. v. d. Maaten, G. Hinton, Visualizing data using t-sne, *Journal of machine learning research* 9 (2008) 2579–2605.
- [19] R. L. Thorndike, Who belongs in the family, *Psychometrika* (1953) 267–276.