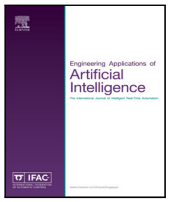
Engineering Applications of Artificial Intelligence 95 (2020) 103878

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



A novel methodology to classify test cases using natural language processing and imbalanced learning

Sahar Tahvili [a,b,*], Leo Hatvani [a,**], Enislay Ramentol [c,**], Rita Pimentel [d,g], Wasif Afzal [a], Francisco Herrera [e,f]

[a] School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
[b] Global Artificial Intelligence Accelerator (GAIA), Ericsson AB, Sweden
[c] Department of Financial Mathematics, Fraunhofer Institute for Industrial Mathematics, Germany
[d] RISE SICS, Västerås AB, Sweden
[e] Department of Computer Science and AI, University of Granada, Spain
[f] Faculty of Computing and Information Technology - North Jeddah, King Abdulaziz University, Saudi Arabia
[g] Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Norway

ARTICLE INFO

Keywords:
Software testing
Artificial intelligence
Imbalanced classification
Natural language processing
Optimization
IFROWANN
Doc2Vec

ABSTRACT

Detecting the dependency between integration test cases plays a vital role in the area of software test optimization. Classifying test cases into two main classes – dependent and independent – can be employed for several test optimization purposes such as parallel test execution, test automation, test case selection and prioritization, and test suite reduction. This task can be seen as an imbalanced classification problem due to the test cases' distribution. Often the number of dependent and independent test cases is uneven, which is related to the testing level, testing environment and complexity of the system under test. In this study, we propose a novel methodology that consists of two main steps. Firstly, by using natural language processing we analyze the test cases' specifications and turn them into a numeric vector. Secondly, by using the obtained data vectors, we classify each test case into a dependent or an independent class. We carry out a supervised learning approach using different methods for handling imbalanced datasets. The feasibility and possible generalization of the proposed methodology is evaluated in two industrial projects at Bombardier Transportation, Sweden, which indicates promising results.

1. Introduction

Software testing is an important and effort-intensive activity in the software development life cycle (SDLC), and thus test optimization plays a vital role in the testing domain. According to reports from both academia and industry, the process of software testing can take of up to 50% of the total development cost (Alégroth et al., 2016). The testing cost can be decreased by using test automation, test case selection and prioritization and test suite minimization (Nardo et al., 2015). The research results indicate that the application of an artificial intelligence (AI) technique has real potential for making a positive impact on software testing. Moreover, the increase in availability of AI technologies provides opportunities to improve the existing software testing processes (Chen et al., 1995).

Nowadays, the process of testing can be performed manually, semi- or fully automated. In a manual testing procedure, a set of test specifications needs to be created for every system under test (SUT). The manual testing process is still a popular approach especially in the safety critical systems, where assurance arguments ultimately depend on human judgment (Chechik et al., 2019). Since both manual test creation and manual test execution are time and resource consuming processes, it can be beneficial to apply AI technologies, such as machine learning and deep learning, for analyzing test cases.

Reports from different industries show that the dependencies between the integration of test cases have a direct impact on the test execution results, where the dependent test cases can fail after each other if they are ranked in the wrong order (Parsa et al., 2016; Arlt et al., 2015). Nonetheless, classifying test cases into dependent and independent classes is a challenging task because of the following reasons:

1. It requires capturing and analyzing various testing artifacts such as requirement specifications, test records, system architecture, signal information, among others.

* Corresponding author at: Global Artificial Intelligence Accelerator (GAIA), Ericsson AB, Sweden.
** Corresponding authors.
E-mail addresses: sahar.tahvili@mdh.se (S. Tahvili), leo.hatvani@mdh.se (L. Hatvani), enislay.ramentol@itwm.fraunhofer.de (E. Ramentol), rita.pimentel@ntnu.no (R. Pimentel), wasif.afzal@mdh.se (W. Afzal), herrera@decsai.ugr.es (F. Herrera).

https://doi.org/10.1016/j.engappai.2020.103878
Received 22 October 2019; Received in revised form 20 June 2020; Accepted 4 August 2020
Available online 14 August 2020

2. The problem suffers from imbalanced class distribution, which is still a challenging problem in data mining (Lopez et al., 2013).

Since the main goal of the dependency detection between test cases is test efficiency and cost minimization, the new approaches should be accurate, applicable in different contexts and should not require extra efforts. Employing AI techniques on manual test cases can provide useful input, which can be utilized for test optimization purposes.

In this study, we aim to split manual integration test cases into two main classes: dependent and independent. To tackle our problem, we propose a two-step methodology.

(I) The first step is to obtain a numeric dataset format. In this regard, we use neural networks model that turns the manual descriptions of the test cases into numeric vectors of descriptive features. Afterwards, each of the test cases is assigned to a decision attribute ("dependent" or "independent"). As stated earlier, in many cases, the number of dependent and independent test cases is unevenly distributed, which means that we are facing a class imbalance problem.

(II) The second step is to carry out a supervised learning classification for imbalanced datasets. The imbalance classification has become a top issue within the machine learning community.

There are many examples of real-world applications that study this problem, such as medical applications (Mazurowskia et al., 2008; Gao et al., 2016), image recognition (Kubat et al., 1998; Buda et al., 2018), risk management (Huang et al., 2006) or anomaly detection (Khreich et al., 2010). Indeed, a different variety of solutions for class imbalance problems have been proposed in recent years. They can be divided into four main groups: data level, design of specific classification algorithms, cost sensible and ensembles. Although the wide range of solutions currently available are state of the art, finding the most appropriate solution for a specific problem can lead to a fairly extensive experimental study. For our study, we select a set of well-known methods from the four-groups mentioned above.

The proposed methodology is evaluated on industrial case-studies at Bombardier transportation (BT) in Sweden. Two on-going testing projects – projects A and B – are selected, where all designed information for these two projects is extracted from BT's database. Both projects are designed for testing trains, where a separate team is behind each. Although they both test a train product, the test cases which are designed for testing project A should not be utilized for project B.

The organization of the paper is laid out as follows: Section 2 provides a background of the initial problem. The dependency detection in the software testing domain and an overview of the usage of AI technologies in this area are presented in Section 3. Section 4 describes the structure of the proposed methodology. Two industrial case studies are shown in Section 5. The obtained results are depicted in Section 6. Finally, Section 7 concludes the paper and clarifies some points of future directions of this study.

2. Preliminaries

This section provides the preliminaries for the present study. We explain the need of separating dependent and independent test cases. Moreover, some relevant theoretical elements are also described to achieve a better understanding of our proposal. Later in this section, we provide some background information about natural language processing (NLP) and the utilized method in this paper, Doc2Vec. Finally, the class imbalance problem and its most relevant solutions are presented.

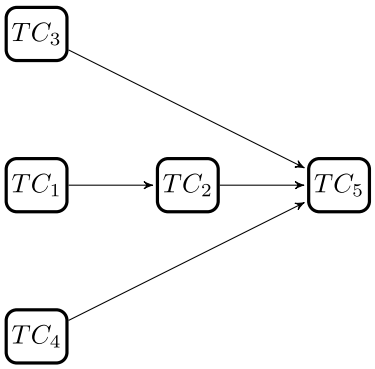


Fig. 1. Example of an embedded digraph of dependencies for five test cases.

2.1. Dependency detection

Dependency between integration test cases can be considered as a critical criterion for test optimization in any form of test selection, prioritization, scheduling (Yoo and Harman, 2007; Elbaum et al., 2002; Tahvili et al., 2016a) and test automation. There are several kinds of dependencies between test cases e.g. abstract dependency, causal dependency or temporal dependency. In our previous work (Tahvili et al., 2019) we defined a new type of dependency called functional dependency, which usually occurs between integration test cases, and has a direct effect on the test execution results. Given that test case TC_2 is functionally dependent on test case TC_1, if TC_1 fails during the testing process, TC_2 will fail as well. This kind of dependency has been observed in several cases in different domains (Arlt et al., 2015; Tahvili et al., 2016b). Therefore, detecting the dependencies between test cases is important, since it can be utilized later for ranking them for execution based on their relationship with other test cases. Usually in this approach, test cases are connected and following each other in succession as a directed graph (chain). Thus, independent test cases (e.g. TC_1 in Fig. 1) should be executed first and then all other dependent test cases. To clarify the explanation, let us consider a dummy example,[1] with five test cases: TC_1, TC_2, TC_3, TC_4 and TC_5. Let us assume that we can describe the following embedded directed graph of dependencies:

As we can see in Fig. 1, TC_1, TC_3 and TC_4 are independent test cases, TC_2 depends on TC_1. TC_5 directly depends on TC_2, TC_3 and TC_4, where we call them as the precedents, and indirectly depends on TC_1. The presented test cases in Fig. 1 can be executed in a different order such as first all independent test cases (TC_1, TC_3 and TC_4) are ranked first for execution, or TC_1 and TC_2 are scheduled for execution first and later two independent test cases TC_3 and TC_4. Detecting this grid of dependencies is required to analyze several layers of the testing process such as requirement specifications and software modules. However, the mentioned required information is not available in all testing projects and sometimes the process of capturing them takes more time than any random test execution. On the other hand, knowing the dependency between test cases can be utilized for other test optimization purposes such as parallel test execution and test automation, where independent test cases can be considered as good candidates for automation in a semi-automated testing procedure.

For clustering and classifying test cases based on their dependencies, we can employ just their test specifications. Previously (Tahvili et al., 2019), we divided test cases from one industrial project into several clusters based on their dependencies. However, in the current study, we are aiming to classify all test cases into two groups: dependent and independent as mirrored in Fig. 2. The main reasons behind this

[1] It is dummy in the sense that the number of test cases is very small compared with real industry cases.

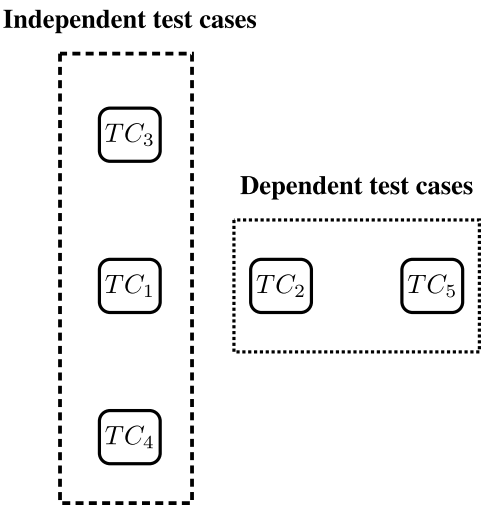


Fig. 2. Classifying test cases into two main classes: dependent and independent.

approach can be summarized as 1 - improving the obtained results from the clustering approach in Tahvili et al. (2019), 2- employing the ground truth for labeling data, and thereby applying a supervised learning approach, and 3- applying a different methodology for classifying test cases into dependent and independent.

2.2. On the use of natural language processing

Table 1 shows an example of a manual test case for a safety critical system which consists of test case ID, test case description, expected test result, test steps, corresponding requirements, etc. In this example, two requirements (`SRS-BHH-LineVolt1707` and `SRS-BHH-Speed2051`) are assigned to the test case.

As mentioned in Section 1 in a manual testing procedure, all testing artifacts (e.g. requirements and test cases) are written by testers. Therefore, employing natural language processing techniques might provide highly useful information, which can be utilized for test optimization purposes.

Examples include automatic test creation from the textual requirements and automatic testing oracle generation for unusual behaviors from Javadoc comments (Goffi et al., 2016), as well as automatic test generation from bug reports (Fazzini et al., 2018). Heretofore, some NLP techniques are employed by us for: extracting test case execution time (Tahvili et al., 2018a), scheduling test cases for execution based on their semantic similarity (Tahvili et al., 2018c), detecting the functional dependency between test cases through analyzing requirement specifications (Tahvili et al., 2018b) and also test case specifications (Tahvili et al., 2019). However, not all required test artifacts for dependency detection are accessible in all testing processes at industries. A deep understanding of test specifications can help us to detect the dependencies between test cases in a more promising way.

2.2.1. Neural networks in natural language processing

Although employing classical machine learning approaches (e.g. Naive Bayes, Support Vector Machine) received a great deal of attention for text analysis, neural networks models are shown to perform better for more complex tasks such as sentiment analysis and translation (Majumder et al., 2017).

Neural embedding models usually refer to methods for embedding words or documents into a vector space via utilizing neural networks (Maslova and Potapov, 2017). The neural networks take over adjacent word embeddings, where the information contained in adjacent words is learned effectively. Therefore, they show better performance compared to other text analysis methods on classification

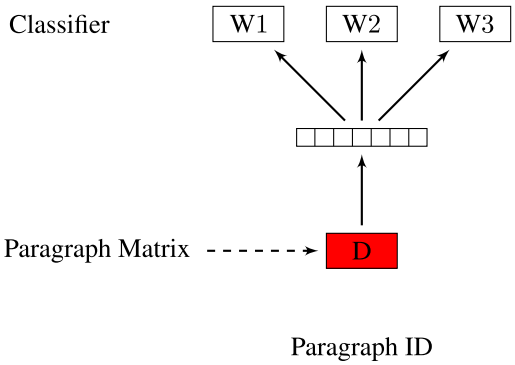


Fig. 3. The structure of the PV-DBOW model.

tasks e.g. sentiment analysis, spam detection, and also topic categorization (Zhang et al., 2015). Word2Vec, Doc2Vec, and FastText (Joulin et al., 2016) can be mentioned as some examples of the neural networks models for word and document embedding.

Doc2Vec was first proposed by Le and Mikolov (2014) in 2014. It has excellent scalability and has filled in gaps of previous approaches where the semantics of the words were ignored. Another of its advantages is that it learns from unlabeled data, meaning that it is applicable for tasks that do not have enough labeled data, which is often the case in industrial applications. The approach is also independent of the language and it can be applied to most languages with minimal modification. The goal of the algorithm is to create a fixed-length numeric representation of a document, regardless of its length. Indeed, for each document, Doc2Vec provides an n-dimensional vector, in which each dimension can be interpreted as a feature. Doc2Vec represents a document with a non-fixed length into a vector and concatenates each word of that document. There are several variants of the Doc2Vec method. In this paper, we use paragraph vectors with a distributed bag of words model (PV-DBOW) which uses a single layer neural networks to predict whether a word is contained in a given document.

Fig. 3 shows the basic idea behind the PV-DBOW model. We need to consider that the PV-DBOW model ignores the context words in the input, but it forces the model to predict words randomly sampled from the paragraph in the output and a Paragraph ID (Witt and Seifert, 2017). Moreover, the Paragraph Matrix in Fig. 3 is the matrix where each column represents the vector of a paragraph. Matrix D has the embeddings for "seen" paragraphs for words. For "unseen" paragraphs, the model is again ran by gradient descent to derive a document vector (Le and Mikolov, 2014; Witt and Seifert, 2017). For instance, given a sentence including three words "[W1 W2 W3]" (see Fig. 3), two words are sampled from it.

After training these neural networks, we can extract the trained values to form feature vectors. These vectors are intended to represent the concepts of the documents. Each element of the vector creates an abstraction for several terms from the corpus. Given a large enough corpus, the distance between the vectors corresponding to semantically similar documents is lower than the ones that are not. Doc2Vec is based around learning vector representations of words using neural networks. It is trained using stochastic gradient descendent, where the gradient is obtained via back-propagation. Therefore, each time we apply the algorithm to a group of documents, we get a different set of vectors. Nevertheless, the relative distance of semantically related documents is statistically indistinguishable from run to run (Rekabsaz et al., 2017). This means that for each run, it is expected that the vectors from the two different classes keep a higher distance, while the vectors from the same class have a lower distance, which is the key point for the classification step. Taking this into account, in order to keep the vectors comparable, we run Doc2Vec at the same time for both training and testing documents. Previously (Tahvili et al., 2019), we showed the

Table 1
A test case specification designed at Bombardier Transportation.

Test case name:	Auxiliary Compressor Control	**Date:**	2020-03-20
Test case ID	**Test level (s)**	**Test Result**	**Comments**
3EST001845-2032 - RCM (v.1)	Sw/Hw Integration		
Test configuration			
TCMS baseline: TCMS 1.2.3.0			
Test rig: VCS Release 1.16.5			
VCS Platform 3.24.0			
Requirement(s)			
SRS-BHH-Line Voltage 1707			
SRS-BHH-Speed 2051			
Tester ID			
BR−1211			
Initial State			
No active cab			
Step	**Action**	**Reaction**	**Pass/Fail**
1	Lock and set Auxiliary reservoir pressure < 5.5 bar	Signal Command auxiliary compressor	
2	Activate cab $A2$ lock and set signal braking mode from ATP to 109	Signal braking mode to IDU is set to 109	
3	Lock and set Auxiliary reservoir pressure > 5.5 bar	Signal Auxiliary compressor is running to IDU is set to FALSE	
4	Wait 20 s		
5	Reset dynamic brake in the train for 5 s	IDU in $B1$ car as On	
6	Set Auxiliary reservoir pressure < 5.5 bar	Signal Auxiliary compressor is running to IDU is set to FALSE	
7	Clean up		

relationship between test case semantics similarity and their functional dependency using Doc2Vec and a clustering algorithm, with a F-score of 0.75.

2.3. On the use of imbalance learning

As emphasized earlier, the dependency problem suffers from an imbalanced dataset. The class imbalance problem appears when one concept (the minority class) to be classified is significantly less represented than the other (the majority class). The imbalanced ratio (IR) is a well-known measure in the imbalanced domain. It is a ratio between the number of samples in the majority class and the number of samples in the minority class (see Eq. (1)).

$$IR = \frac{Majority_{examples}}{Minority_{examples}}. \quad (1)$$

Some researchers consider that a dataset suffers from an imbalanced problem if the IR is higher than 3. We can compute the IR using Eq. (1).

To find a more appropriate solution for imbalanced proportion of classes, we use imbalanced learning, which has recently become a top research topic in the machine learning community (Fernández et al., 2018). The classification task can become very complicated when concepts are not equally represented in the data sample (Batista et al., 2004; Bi and Zhang, 2018; Galar et al., 2013; Jensen and Cornelis, 2011; Khan et al., 2018; Ramentol et al., 2015).

The main problem with imbalanced data is that traditional classifiers do not take into account class distribution. Instead, they operate with global metrics, and as a consequence, the less represented class, known as minority or positive, tends to be poorly classified while the majority, or negative, tends to be very well classified. However, the concept of greatest interest is usually in the minority class.

This phenomenon has attracted a lot of attention resulting in a set of good solutions. These solutions can be divided into four main groups:

I. **Data level solutions** (Ramentol et al., 2012; Batista et al., 2004; Han et al., 2005; Chawla et al., 2002): This group of solutions consists of modifying the data distribution, via removing examples from the majority class, known as undersampling, or by creating examples from the minority class, known as oversampling. Moreover, any combinations of undersampling and oversampling, is known as hybrid solutions.
II. **Cost sensitive solutions** (Ting, 2002; Zadrozny et al., 2003; Zhou and Liu, 2010): This group of solutions consists of the use of solutions at data level, algorithm level or both at the same time, in order to minimize higher cost errors.
III. **Ensemble solutions** (Galar et al., 2012, 2013): Ensemble solutions in imbalanced domains are mostly a combination of an ensemble with one of the techniques above. The most common ensemble are the solutions which use a data level approach or a cost sensitive solution.
IV. **Algorithm level solutions** (Cieslak et al., 2012; Ramentol et al., 2015): This group of solutions consists of modifying the learning algorithm in order to deal directly with the imbalance data. These solutions could be more versatile since it does not modify the data distribution. For example, modifying the cost per class or adjusting probability estimation in the leaves of the decision trees.

In this work we compare the performance of ten proposed imbalance approaches in the state of the art. Next, a short description of each of them is given.

1. *Synthetic Minority Oversampling Technique (SMOTE)* (Chawla et al., 2002): This oversampling technique creates synthetic examples through performing an interpolation of one minority example and its nearest neighbor.

2. *Synthetic Minority Oversampling Technique with Edited Nearest Neighbor (SMOTE-ENN)* (Batista et al., 2004): This hybrid solution first creates synthetic examples using SMOTE, and then removes the examples which are misclassified by their three closest neighbors. ENN is also able to remove examples from both classes.
3. *SMOTE-Borderline 1* (Han et al., 2005): This method presents a modified version of the original *SMOTE*. Instead of creating synthetic examples, *SMOTE-Borderline 1* utilizes all the minority examples and only oversamples those minorities which are considered as "Borderline". The method classifies every minority example into three categories: **noise, danger** and **safe**, attending to the number of nearest neighbors in the majority class. Finally, in the SMOTE phase, only the examples in **danger** will be used for generating new ones.
4. *SMOTE-Borderline 2* (Han et al., 2005): This method is very similar to the *SMOTE-Borderline 1* and they only differ from each other by the generating strategy for the synthetic examples. *SMOTE-Borderline 2* uses **danger** examples for generating synthetic examples and minorities nearest neighbor. This method also finds the nearest neighbor in both classes. Later, its nearest majority neighbor will be multiplied by a random number between 0 and 0.5, and therefore, the newly generated examples are closer to the minority class.
5. *Safelevel* (Bunkhumpornpat et al., 2009): This method defines a safe level for every minority example, which creates synthetic examples closer to them.
6. *Spider* (Stefanowski and Wilk, 2008): This method removes majority examples that can affect the correct classification of the minority examples. Later, it creates synthetic examples using the minority examples which are "overwhelmed" by objects surrounding the majority classes.
7. *Cost Sensitive–C4.5* (Ting, 2002): This method builds some decision trees that try to minimize the number of high cost errors and, as a consequence, leads to the minimization of the total misclassification costs in most cases.
8. *SVM-Cost Sensitive* (Vapnik, 2013): This method biases traditional Support Vector Machine classifier (SVM) in a way that will push the boundary away from the positive instances using different error costs for the positive and negative classes.
9. *EUSBOOST* (Galar et al., 2012): This algorithm belongs to the ensemble group. It combines boosting with the use of Evolutionary Undersampling (EUS), where each chromosome has a binary coding that represents the presence or non-presence of the example in the dataset. The fitness function takes into account the imbalance distribution favoring the minority class.
10. *IFROWANN* (Ramentol et al., 2015): This algorithm is a powerful classifier specifically designed for imbalanced data. It is based on the Fuzzy Rough Nearest Neighbor (FRNN) classifier (Jensen and Cornelis, 2011). It computes the sum of the memberships of each example to the fuzzy-rough lower and upper approximation of each class, which is then assigned to the class with the higher sum. IFROWANN introduces the use of Ordered Weighted Average (OWA) operators to determine the positive and negative regions, in order to consider the differences between the numbers of examples in classes. The authors proposed the use of 6 different weighting strategies combined with 3 alternatives for defining the fuzzy: Minimum, Average and Łukasiewicz.

3. Related work

In this section, we discuss the related work from two dimensions of interest, beginning with some of the proposed solutions for solving the dependency problem in the software testing domain and then presenting a non-exhaustive overview of the use of NLP approaches in software testing.

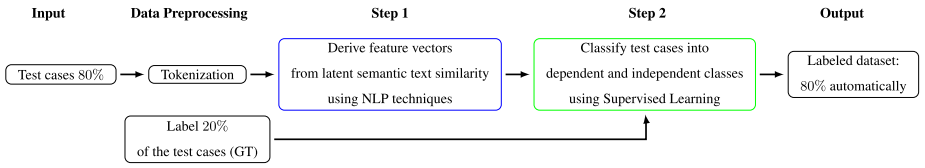
3.1. Dependency identification in software testing

To avoid unproductive testing of unaffected components in a modified program, Bates and Horwitz (1993) identify components through program dependence graphs and slicing with test adequacy data. A new diagram type, a dependency chart, is introduced by Ryser and Glinz (2000) to manage dependencies and interrelations between scenarios of system testing. Haidry and Miller (2013) use a directed graph to identify functionally dependent test cases. An approach to automatically detect redundant test cases based on logical dependency between structured requirements is given by Arlt et al. (2015). In our earlier work (Tahvili et al., 2019), we divided test cases into dependent and independent classes using semantic text similarity, clustering and random undersampling. However, random undersampling causes missing some data points, where some important information about the decision boundary between the minority and majority class (independent test cases in Tahvili et al., 2019) might be eliminated. On the other hand, the application of text similarity techniques in software testing is already a growing research topic, where researchers have applied it for test selection (Unterkalmsteiner et al., 2016), test prioritization (Thomas et al., 2014a) and prioritization of test automation (Tahvili et al., 2019).

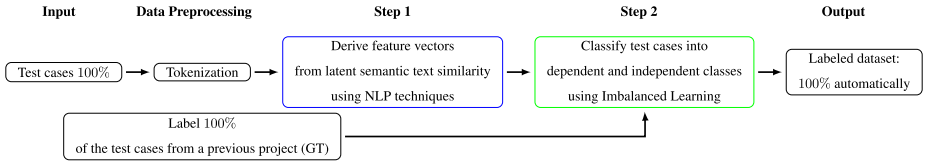
3.2. Natural language processing in software testing

Chen et al. (2015) wrote a survey paper on the use of topic models in software engineering. Topic models discover structure within an unstructured collection of written text through statistical properties of word frequencies. Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA) are two examples of topic models. A topic is a collection of words that co-occur frequently and are often semantically related. Organization of unstructured text into topics helps to index, searching and clustering of information. In software testing, Islam et al. (2012) used LSI to link test cases and software requirements. This was employed to prioritize test cases using a multi-objective optimization approach. Thomas et al. (2014b) applied LDA in test suites to give higher priority to test cases that test different functionalities of the software under test. Chen et al. (2017) adopted LDA to find topics in source code files and test cases. Based on topic defect density and how well-tested a topic is, less-tested and defect-prone topics are then suggested as candidates for allocating more testing resources. Lin et al. (2017) proposed a NLP-based approach for improving the effectiveness of crawling-based web application testing. They applied three transformations sequentially to feature vectors: Bag-of-words, tf-idf and LSI. The evaluation of their approach on input topic identification showed improved accuracy. Preeti et al. (2017) used NLP to extract combinatorial test design model elements from use case specification using Alchemy API and NLTK with Python scripts. They combined the results with combinatorial test design model elements extraction from use case diagrams using a rules-based approach. The parameters and values are then presented as a list of suggestions to the test designer. Nakagawa and Tsuchiya (2015) proposed a linguistic approach to extract constraints from requirements document for combinatorial test design. They identify constraints based on the distance between the words in the requirements document using a coupling metric. Zalmanovici et al. (2016) provide a clustering approach where a test suite is automatically clustered into disjoint clusters. These clusters can then be exploited in different ways: treating each cluster as a functional part, suggesting redundant tests in a cluster, and creating input for combinatorial testing by extracting a model per cluster. Test clustering is achieved using Levenshstein distance and DBScan is used as the clustering algorithm. Clustering of test cases has also been experimented by Yoo et al. (2009) using hamming distance, Leon and Podgurski (2003) using Euclidean distance, Arafeen and Do (2013) using tf-idf for requirements based testing.

In this work, we propose a novel methodology for analyzing the test case specifications and then dividing test cases into dependent



(a) Having only 20% of the data labeled.



(b) Having 100% of labeled data from any previous related project (unseen data).

Fig. 4. The required input, steps and expected output of the proposed methodology in this study.

and independent classes, using a supervised learning approach. The most important difference between exiting solutions and our proposed methodology is the required input. Most of the proposed approaches require several testing artifacts e.g. requirement specifications and architecture of the software product. The proposed methodology in this work just needs the test case specifications, which are always available in all testing projects.

Table 2
The tokenized version of the presented test case in Table 1.

no active cab lock and set auxiliary reservoir pressure 5.5 bar signal command auxiliary compressor activate cab a2 lock and set signal braking mode from atp to 109 signal braking mode to idu is set to 109 lock and set auxiliary reservoir pressure 5.5 bar signal auxiliary compressor is running to idu is set to false wait 20 s reset dynamic brake in the train for 5 s idu in b1 car as on set auxiliary reservoir pressure 5.5 bar signal auxiliary compressor is running to idu is set to false clean up

4. The methodology

This section provides a methodology for solving the initial problem. The main goal of this paper is to split manual integration test cases into dependent and independent classes. The proposed methodology in this study can mainly be divided into two main steps:

I. **Text analysis:** Since the manual test cases are written in text, employing NLP technologies for latent semantic analysis can provide some clues for dependency detection. In this step, all designed test cases need to be parsed and converted to another format (e.g. vectors) for further syntactic analysis.
II. **Classification:** In this step, all test cases need to be classified as dependent or independent. If the number of dependent and independent test cases is very different, then this step may suffer from an imbalanced dataset, and therefore a special algorithm which can handle it should be applied. Otherwise, other classification algorithms can be utilized in this step.

Since the data preprocessing might have a significant impact on the generalization performance of supervised machine learning algorithms (Kotsiantis et al., 2006), thus the input data need to be preprocessed in an early stage of the proposed solution in this study. Data preprocessing can be performed in the form of data cleaning, data integration, transformation, and reduction. However, based on the data type, size, and quality, one or more of the mentioned data preprocessing methods need to be applied to the raw data. Since the input data in this study is manual integration test cases written in a natural text, determinately, we applied tokenization on the original test cases to process the raw data. Table 2 shows the tokenized version of the presented test case in Table 1.

In Section 6.1 Experimental Setup, we provide more details regarding the utilized data preprocessing in the current study.

Excluding the text from each test case, a ground truth (GT) for the training set is required as we are using a classification algorithm to address the problem (supervised learning). As can be observed in Fig. 4, there are two possible scenarios where our methodology can be used: having a least 20% of the data already labeled or having a previous similar project with 100% of the examples labeled. Given that we intend to apply this methodology in a real industrial context, the division between training and test sets needs to be carefully chosen. The common choices are 80%–20%, 70%–30% or 50%–50% at most. However, it is not realistic to ask the experts to label 80%, 70% or even 50% of the test cases. This would be a time-consuming process leading them to reject our methodology. Having this in mind, we propose that the company takes the responsibility of labeling 20% of the data, which will be used to train the model. It is important that the 20% of the test cases chosen to be labeled represents the complete dataset. In other words, the distribution between dependent and independent test cases in the sample and in the whole dataset should be similar.

To decide the training size as 20%, we did experimentation with different possibilities: 5%, 10%, 15%, 20%, 25% and so on. The results start to increase in 20%. That is why we recommend 20% as the minimum percent of labeled data when using our methodology. Due to page restriction, this study is not included in the paper.

Next, we describe our methodology for both mentioned steps in detail. Fig. 4 represents a generic overview of the methodology proposed in this study for classifying dependent and independent manual integration test cases.

As we can see in Figs. 4(a) and 4(b), the discussed steps are mapped with step 1 and step 2. Moreover, the raw data need to be processed before we apply each of the mentioned steps. As stated earlier in this section, there are several ways to process raw data for supervised learning purposes. However, in the described case in this study, since the input is test cases that are written in a natural text, thus all test cases need to be tokenized in advance. As can be seen in Fig. 4(a), for the first scenario, just 80% of all test cases are used (since 20% of the data is labeled before), wherein Fig. 4(b), 100% of test cases are employed. We need to consider that, regardless of the utilized data portion (80% or 100%), all raw data (manual integration test cases in the present study) need to be preprocessed (tokenized in this case) as a part of the proposed methodology in this study.

- **Input:** Our methodology has been designed to operate from two types of entries:

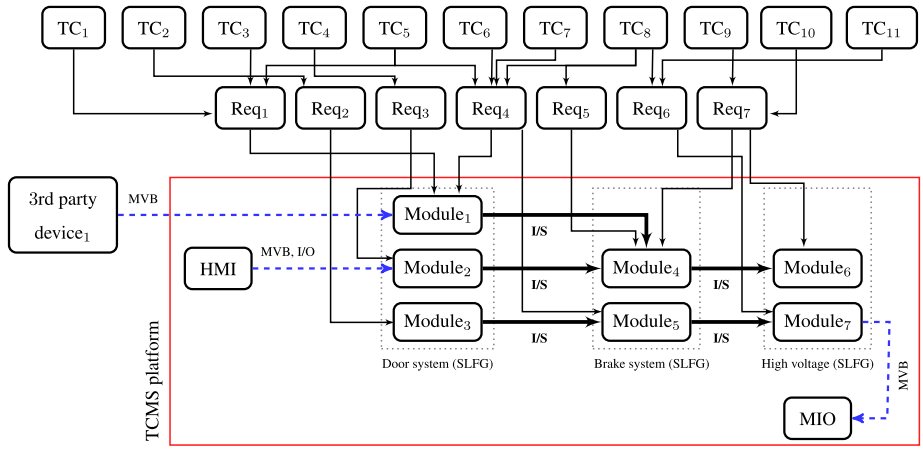


Fig. 5. The input–output signals, requirements, and test cases in the testing platform (TCMS).

(a) Having at least 20% of the data labeled (from one project). As shown in Fig. 4(a).
(b) Having previous similar project with 100% of labeled examples (using unseen data), which is mirrored in Fig. 4(b).

- **Data Preprocessing:** As the natural language test cases are commonly stored in various rich text formats and even spreadsheets, the text is extracted from them. The text is then preprocessed using standard tokenization methods to remove the majority of punctuation and convert all the text to lowercase.
- **Step 1:** Latent semantic analysis technique then reads the text documents and provides a set of vectors with n-dimensional features, where each vector represents a text document. This step can be performed by using various implementations of neural embedding models for text analysis such as Gensim (Rehurek and Sojka, 2010), Paragraph vectors (see Section 2.2.1), and fastText (Joulin et al., 2016).
- **Step 2:** Applying supervised learning on the imbalanced data provided from step 1, each vector is classified into one of the two classes. This step can be performed by using various implementations of the mentioned imbalance approaches in Section 2.3.
- **Output:** For each test case to be classified, a class label is automatically assigned.

At the end of the procedure all test cases from the initial dataset are assigned to either independent or dependent class.

5. Industrial Case studies

This section analyzes the feasibility of the proposed methodology, which has been done through studying two on-going testing projects *A* and *B* at Bombardier Transportation (BT) in Sweden. All designed requirements and test case specifications for the two projects are extracted from DOORS[2] database at BT.

5.1. Case studies

The units of analysis in these case studies are requirements and test specifications at the integration testing level for projects *A* and *B*, where for both projects, there exists the ground truth. Table 3 represents some additional information about the projects. As we can see, the number of independent test cases is different for the two projects, which emphasizes the need for an automated approach for dependency detection.

[2] Rational Dynamic Object Oriented Requirements System, which is a requirement management tool.

Table 3
The units of analysis in the industrial case study.

Project	Requirement	Test case	Independent test case	Dependent test case	IR
A	3201	1748	328	1420	4.33
B	3401	2028	1511	517	2.92

Moreover, there is a considerable imbalanced ratio between independent and dependent test cases in both projects *A* and *B*. According to Table 3 the number of independent test cases for project *A* is equal to 328 whereas the number of dependent test cases for this project is equal to 1420. On the other hand, the number of independent test cases for project *B* is equal to 1511, whereas the dependent test cases is equal to 517 for this project. The inserted information in Table 3 indicates that the dependency detection problem suffers from an imbalanced classification problem.

5.2. The ground truth

As stated earlier, the dependencies between requirements and test cases are detected by analyzing the signal communications between software modules (Tahvili et al., 2018b). In this regard, the structure of the Train Control Management System (TCMS) which is the testing platform at BT is analyzed by us. Fig. 5 shows a part of the TCMS platform, where the traceability graph between software modules, requirements, and test cases is visible.

As we can be seen in Fig. 5 two test cases are functionally dependent on each other if the output *internal signal* from the corresponding software module enters as an input *internal signal* to another corresponding software module. Indeed, if there exists any shared *internal signal* between two software modules (transmitting and receiving the same *internal signal*), then these two modules are depending on each other. Thereby, there is a dependency between all requirements and test cases that are assigned to test those software modules. For instance, TC_7 is functionally dependent on TC_2 because an internal signal from **Module**$_3$ enters to **Module**$_5$. The proposed approach in Tahvili et al. (2018b) was applied on both projects *A* and *B*, and the results are utilized in this paper as the ground truth, where we know the dependencies between test cases. Since the provided information in Fig. 5 is not available for all testing projects, we need to employ other methodologies for detecting the dependencies between test cases. However, the test cases are usually available and detecting the dependencies through analyzing them might be applicable in different industries.

As highlighted earlier, the presented results in Tahvili et al. (2018b) (Fig. 5 in the current study) are utilized here as the ground truth for labeling the vectors provided by Doc2Vec. In the example presented in Fig. 5 we have a dataset of test cases and we want our model to identify

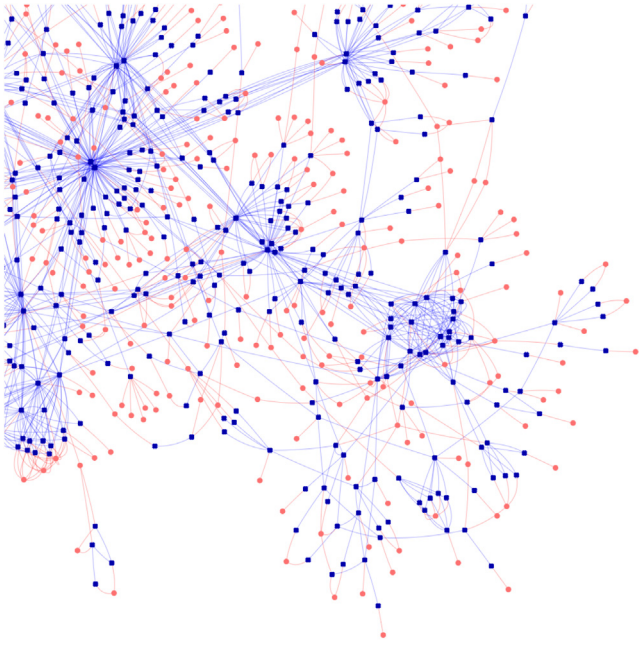


Fig. 6. Partial dependency graph for the requirements and test cases for the project A (the ground truth). The blue squares represent requirements and the red circles represent test cases. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

dependent test cases. Since we only have two classes, we need to use a binary classifier. In other words, we need to decide whether if a test case is dependent (class 1) or not dependent (class 0) from any of the other test cases. For instance, in the above-mentioned example, TC_7 is labeled as a dependent (class 1).

In the upcoming section, the feasibility of the proposed methodology is evaluated by comparing the obtained results in this study with the ground truth. From the ground truth (also presented in Table 3) we know that our dataset in Project A has 1748 instances, from which 1420 are independent (majority class) and only 328 are dependent (minority class). As we explained before, this means that the datset suffers from a class imbalance problem, with an imbalance ratio of 4.3 using Eq. (1). Indeed, there are more than four independent test cases for each dependent test case (Tahvili et al., 2019). Using the signal information in Fig. 5 helped us to visualize the dependency relationships between test cases (via requirements), which has been partially mirrored in Fig. 6. This figure highlights that detecting the dependencies between test cases (red nodes) is very challenging, therefore applying an automated approach can help testers and test managers to rank test cases for execution in a more efficient way.

6. Experimental evaluation

In this section we evaluate the performance of our methodology using two projects from Bombardier Transportation, Sweden. The evaluation is performed in two parts: the first part will contain an overall evaluation of the proposed methodology through applying a semi-supervised classification method (Isaac et al., 2015). In the second part we evaluate the proposed methodology using unseen data.

6.1. Experimental setup

The test cases were obtained as a set of Microsoft Excel documents from which we removed formatting and extraneous information. First we extract only the initial state and test steps (actions and reactions, as shown in Table 1) as pure text. These are then tokenized by converting all letters to lowercase and replacing any punctuation other than `()`,`.!?\'` with spaces. For the example, for the test case in Table 1, this produces output presented in Table 2. To associate feature vectors with documents, we used Doc2Vec (Mikolov et al., 2013; Ilenic, 2017) method, which trains neural networks using the content of the document and extracts the values from a layer of neurons as feature vectors. We generated 128 real-value features for each document over 100 training epochs. Increasing the number of features or training epochs, in our case, did not yield more accurate results.

The rest of the parameters were as follows: the batch was set to 32, 2 noise words were used to sample from the noise distribution, and the learning rate was set to 0.001.

In this experiment we use 5-fold cross validation, applying a semi-supervised classification method. The semi-supervised classification applies to problems where the number of labeled examples (the training sample) is less than the unlabeled examples (the test sample). We attempted to show that although only 20% of the ground truth is available, it is still possible to classify the remaining test cases. To classify the feature vectors and consequently the test cases, we used KEEL[3] (Triguero et al., 2017). KEEL is a software tool developed in Java[4] that provides the implementation of the most relevant state of the art data mining algorithms. This tool has a module called "Imbalanced learning". All the algorithms used in our comparative analysis are implemented in this module. The data partitions used in our study have been made using the KEEL module "Data Management". In this way we guarantee a uniform distribution of examples by classes for each partition (Triguero et al., 2017). Any interested reader can easily reproduce our proposal.

The original data distribution in project A is: 328 examples in the minority class and 1420 examples in the majority. When the data is split for the cross validation, we get the following data distribution: 65 examples in the minority class and 384 examples in the majority class (with an imbalance ratio of 4.3).

We also provide all experimental setup, data description and parameters and lastly, we present and discuss the obtained results.

6.2. Parameters

For our experimental study we select ten (from the 4 main groups of solutions) of the most representative algorithms for imbalanced domain. All the selected algorithms have shown good performance in previous studies.

As we mentioned before our experiments can be perfectly reproducible by any interested reader. In this regard, Table 4 presents the parameters which have been used in this study. For each method, the parameters shown in Table 4 are the ones recommended by the authors, and thus the ones used in our experimental study. Regarding to IFROWANN classifier, the authors propose 18 configurations of the algorithm as a result of combining 3 t-norms and 6 weighting strategies. In this study we use an average fuzzy relation, as the authors suggest. Moreover, we apply three weighted strategies, namely w_1, w_2 and w_3, given that those are the ones recommended by the authors when the imbalance ratio is low.

Usually the companies have several projects with the same characteristics, especially in regard to the distribution between dependent and independent test cases. Due to the fact that we know the labels of all test cases in this specific case, we proceed to the experimental study using cross validation. We use a 5-fold cross validation, and as we mentioned before, only 20% of the data is trained which is then tested on the remaining 80%.

Table 5 shows the results of the 3 IFROWANN configurations for each of the 5 partitions as well as the average. As we can see, the best IFROWANN configuration is obtained for $av - w_1$, which will be used for the performance comparison with other algorithms. Moreover, the

[3] Knowledge and Extraction based on Evolutionary Learning.
[4] Available at: https://sci2s.ugr.es/keel/index.php.

Table 4
Parameters recommended by the state of the art methods.

Method	Parameters and values
SMOTE	Number of Neighbors = 5, Type of SMOTE = both, Balancing = Yes, Quantity of generated examples = 1 Distance Function = HVDM, Type of Interpolation = standard
S-ENN	Number of Neighbors ENN = 3, Number of Neighbors SMOTE = 5 Type of SMOTE = both, Balancing = Yes Quantity of generated examples = 1, Distance Function (SMOTE) = HVDM Distance Function (ENN) = Euclidean
S-BL1	Number of Neighbors for SMOTE = 5 Number of Neighbors for considering a instance BORDER = 3 Type of Borderline SMOTE = 1, Type of SMOTE = Both Balancing = Yes, Quantity of generated examples = 1 Distance Function = HVDM, Type of Interpolation = standard Alpha = 0.5, Mu = 0.5
S-BL2	Number of Neighbors for SMOTE = 5 Number of Neighbors for considering a instance BORDER = 3 Type of Borderline SMOTE = 2, Type of SMOTE = Both Balancing = Yes, Quantity of generated examples = 1 Distance Function = HVDM, Type of Interpolation = standard Alpha = 0.5, Mu = 0.5
Safelevel	Number of Neighbors for SMOTE = 5, Type of SMOTE = Both Balancing = Yes, Quantity of generated examples = 1 Distance Function = HVDM, Type of Interpolation = standard Alpha = 0.5, Mu = 0.5
SPIDER	Number of Neighbors = 3, Distance Function = HVDM, Preprocessing Option = WEAK
C4.5	pruned = TRUE, confidence = 0.25, instancesPerLeaf = 2
CS-C4.5	pruned = TRUE, confidence = 0.25, instancesPerLeaf = 2 minimumExpectedCost = TRUE
CS-SVM	Kernel Type = polynomial, C = 100.0, eps = 0.001
EUSBOOST	pruned = TRUE, confidence = 0.25, instancesPerLeaf = 2 Number of Classifiers = 10, Algorithm = ERUSBOOST Train Method = NORESAMPLING, Quantity of balancing SMOTE = 50 IS Method = HammingEUB_M_GM
IFROWANN	Weighting strategy = w_1, w_2, w_3 Fuzzy Relation = average

Table 5
AUC results of the three IFROWANN configurations.

Method	1tst	2tst	3tst	4tst	5tst	Average
$av - w_1$	0.8508	0.8315	0.8381	0.8402	0.8057	0.8333
$av - w_2$	0.8463	0.8210	0.7990	0.8346	0.7948	0.8191
$av - w_3$	0.4201	0.5284	0.5081	0.5180	0.5419	0.5033

worst result is obtained with $av - w_3$. From this experimental study we can conclude that despite the fact that $av - w_3$ is highly recommended by IFROWANN authors for low imbalance problems, when the training set is much smaller than the test set , $av - w_3$ does not have the better performance.

As we explained before, standard classification methods tend to be biased by the class most represented. This is due to the optimization of global metrics, such as error or accuracy, which do not take into account the distribution of the instances by classes. As a result of this, notably precision is achieved in the class that is represented most, while the instances concerning the lesser represented class tend to be poorly classified. In all the utilized methods, we use the parameters which are recommended by the authors.

In imbalanced learning the area under the Receiver Operating Characteristic (ROC) curve, known as AUC (area under curve) is a very common metric for showing the diagnostic ability of a binary classifier. Indeed, the ROC curve plots the rate of correctly classified minority instances against the rate of incorrectly classified majority instances (Huang and Ling, 2005) and the AUC represents the trade-off between them (Fawcett, 2006).

Table 6 represents the AUC values of the selected approaches from state of the art algorithms using the presented data in Table 3 from project *A* at BT. As highlighted in Table 6, IFROWANN shows an outstanding result (AUC = 0.83) compared to all other approaches. Moreover, CS-SVM has the highest AUC among the selected classification methods. In conclusion, the presented results in Table 6 can help us to put the rest of the classification methods into the same level ($50 \leq$ AUC ≤ 60). In the next subsection, the performance of all presented methods in Table 6 is studied on another testing project at BT, used as unseen data.

6.3. Model evaluation using unseen data

In order to assess the generalizability of the proposed methodology in this study, the data of another testing project at Bombardier transportation is collected as unseen data. In fact, the ability to perform well

Table 6
AUC results for the state of the art classification methods.

Group	Method		1tst	2tst	3tst	4tst	5tst	Mean
Data	SMOTE	+ C4.5	0.61953	0.60851	0.66168	0.55398	0.62882	0.61451
	S-ENN		0.66179	0.61991	0.63855	0.63285	0.70081	0.65078
	S-BL1		0.64660	0.68675	0.54710	0.66509	0.61316	0.63174
	S-BL2		0.69814	0.60875	0.61689	0.54328	0.56731	0.60669
	Safelevel		0.57741	0.64583	0.60798	0.71593	0.68415	0.64626
	Spider		0.66050	0.65026	0.64453	0.62969	0.66747	0.65049
CS	CS-C4.5		0.60920	0.60048	0.55228	0.59171	0.58708	0.58815
	CS-SVM		0.73263	0.74061	0.73140	0.68286	0.7091	0.71932
Ensemble	EUSBOOST		0.67065	0.67348	0.68476	0.667478	0.68014	0.67530
Algorithm	IFROWANN av-w_1		0.85083	0.83148	0.83812	0.84022	0.80567	0.83327

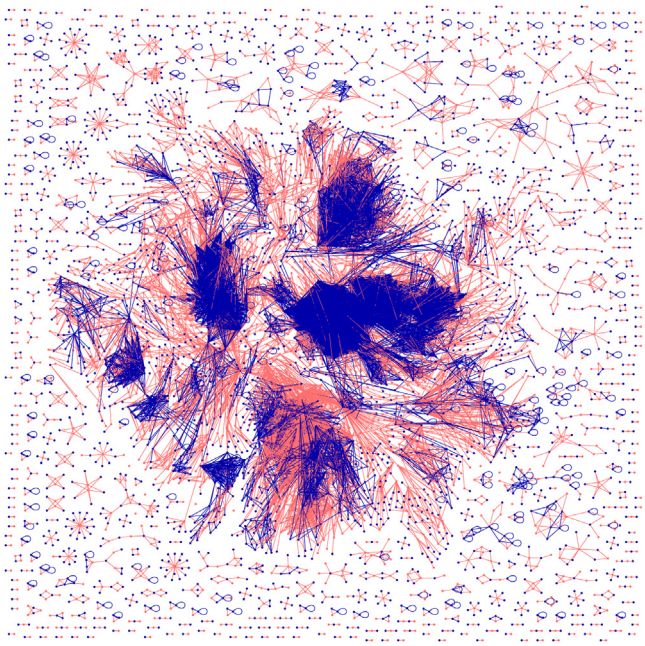


Fig. 7. Partial dependency graph for the requirements and test cases for project *B*. The blue nodes represent requirements and the red nodes test cases. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

on unseen data can provide some clues for the model generalization, which is the desirable characteristic of any demanded model (Wohlin et al., 2000).

To obtain this target, all designed requirements and test cases for project *B* are utilized. The size of the captured data for the project *B* is already presented in Table 3. The dependencies between requirements and thereby test cases for the project *B* are detected through using our tool sOrTES.[5] Fig. 7 shows the dependency between requirements (blue nodes) and thereby test cases (red nodes) for the project *B*. In other words, the ground truth for the dependencies between test cases exists for the performance evaluation of the proposed methodology. It should be kept in consideration that the testing team which works on the project *B* is completely different from the project *A*. Thus, there is no common test case between these two projects. Moreover, the way that the testers wrote the test cases is slightly different. Nonetheless all test cases are designed to test trains, which means that both have the same number of sub-level function groups (SLFG). The number of test cases as well as the ratio of dependent and independent test cases are completely different (see Table 3).

In order to evaluate the proposed dependency detection methodology on a set of unseen data (project *B*), the following steps are conducted:

1. A number of 3401 requirement specifications and 202 test specifications are extracted from the DOORS database.
2. The ground truth for the dependency between requirements and thereby test cases is detected via using sOrTES (Tahvili et al., 2019).
3. All test case specifications are inserted to the Doc2Vec.
4. The provided vectors by Doc2Vec are used as an input to the imbalanced classification method in order to train the algorithm.
5. The test cases from project *A* are used as a target for dependency detection using the imbalanced classification method.
6. The provided dependent and independent classes by imbalance learning are then compared with the ground truth for the project *A*.
7. The steps 4 to 6 are repeated for the various imbalanced classification algorithms and then the results are compared (see Table 7).

The AUC values using unseen data are summarized in Table 7, which indicates that IFROWANN performance on unseen data is good (AUC = 0.76) and is still superior than all other methods. Thus, the proposed methodology is likely to accommodate other projects within the same domain. From this experimental study we infer that when the training set is sufficiently large — implying that the domain of the variables are well covered $av - w_1$ and $av - w_3$ show the best results. Nevertheless, $av - w_1$ proved to be the most stable configuration in both experiments. Furthermore, $av - w_2$ presents the highest difference in performance. This is due to the definition of the weighting vectors, since for $av - w_2$ the weight values of the majority class decrease more rapidly than in the other configurations.[6]

Additionally, Fig. 8 represents the harmonic relationship between the false positive and true positive rates using the proposed methodology in this study. Each point represents the rates of false positive and true positive for a specific threshold. The closer the curve is to the top-left corner, the better the algorithm's performance. Indeed, Fig. 8 shows the probability of dividing test cases into two classes, dependent and independent test cases, using unseen data.

6.4. Threats to validity

Finally, we discuss the validity threats, the research limitations and challenges in conducting the present study.

- **Construct validity** addresses if the study measures what we intend it to measure (Robson and McCartan, 2011). The major

[5] Stochastic Optimizing TEstcase Scheduling, see Tahvili et al. (2019).

[6] More details can be found in the original paper (Ramentol et al., 2015).

Table 7
AUC results of the state of the art classification methods, where the model is trained on the project *B* and tested on the project *A*.

Method		AUC
Data Level	SMOTE + C4.5	0.518107
	S_ENN + C4.5	0.466959
	S_SB1 + C4.5	0.505793
	S_TL + C4.5	0.467895
	Safelevel + C4.5	0.509415
	Spider + C4.5	0.511009
Cost Sensitive	CS-C4.5	0.529903
	CS-SVM	0.605371
Ensemble	EUSBOOST	0.557895
Algorithm	IFROWANN av-w1	0.700140
	IFROWANN av-w2	0.245611
	IFROWANN av-w3	0.764342

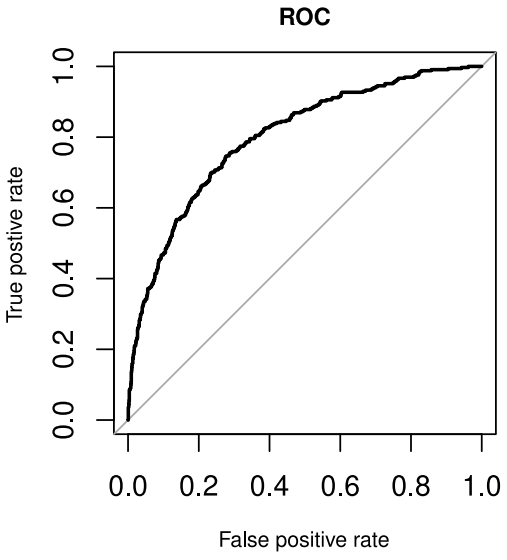


Fig. 8. ROC curve analysis for classifying test cases into dependent and independent.

construct validity threat in the present study is the way that the vectors are obtained from the Doc2Vec tool. Writing test cases in a more formal way might directly affect the performance of Doc2Vec. Since test cases are written by the testers some parameters such as language skill, domain knowledge and experience influence the quality of the test cases. This issue is even obvious in the presented results. Our analysis indicates that the test cases from project *A* are created with higher quality when compared to project *B*.

- **Internal validity** addresses the conclusions of the study (Runeson and Höst, 2008). In order to reduce the threats to internal validity, a diverse number of state of the art algorithms are applied and the obtained results are compared.
- **External validity** refers to the generalization of the findings (Wohlin et al., 2000). The proposed methodology in this study has been applied to only two industrial testing projects in the railway domain, however, it should be applicable to other similar contexts in other testing domains. Since the classification steps are designed for solving imbalanced datasets, the proposed methodology can also be applied for solving other imbalanced problems as well. Moreover, the context information has been provided in Section 6 to enable comparison with other proposed approaches in the testing domain.
- **Conclusion validity** addresses the factors which can affect the observations and may lead to an inaccurate conclusion (Cozby and Rawn, 2012). Utilizing the human's judgment, decision and supervision might decrease this threat. Since the proposed methodology in this study is designed and developed for an industrial usage at Bombardier, a close collaboration and dialogue with the subject mater experts (SME) is established in order to ensure the industry's requisite and needs. Moreover, the detected results (independent and dependent test cases) are presented in a few project meetings at BT.
- **Reliability** addresses the repeatability and consistency of the study (Runeson and Höst, 2008). The text analysis step of this study is very sensitive and relies on the way that test cases are designed, which is usually a common problem faced by many researchers in the area of natural language processing. In consultation with the SMEs at BT, a set of wrongly spelled words were found in the test case specifications. Thus, the data in the DOORS database sometimes contains ambiguity, uncertainty and spelling issues, which have a direct impact on the performance of the Doc2Vec tool. Doc2Vec searches for semantic similarity for providing the vectors. In the case that no semantic similarity is found between two test cases, they will end up as independent test cases later on. This issue can directly impact the accuracy of the proposed solution. In addition, most of the language parsing techniques have some performance issues when a large set of data needs to be processed. There are demerits in the available tools for natural language processing.

7. Concluding remarks

In this paper, we present a novel methodology for splitting manual test cases into dependent and independent classes. The proposed methodology uses neural networks model for natural language processing and imbalanced learning for the classification task. Our proposal has been validated in two real industrial case studies in Bombardier Transportation, Sweden.

We make the following contributions.

I. **From a machine learning point of view:**

- A novel two-steps methodology is proposed.
- Doc2Vec proves to be a good tool when transforming the manual test cases into feature vectors.
- IFROWANN performs well when splitting dependent and independent test cases as an imbalance learning algorithm.

II. **From a software testing point of view:**

- It is possible to classify, with an acceptable accuracy, the test cases of a project as dependent or not from only knowing 20% of the ground truth.
- By having the ground truth of a previous project, it is possible to classify all the test cases of an unseen project with a high accuracy.
- The membership degree to the minority class provided by IFROWANN (called $Pert_{min}$) can be used to define an execution order.

We highlight that the proposed methodology is not limited to software testing. It can be applied in other types of problems where a set of text documents should be split into two main classes with a high probability of class imbalance data such as use Liu et al. (2009).

The main future direction of this study is tool making. Previously, we designed, implemented and developed two tools: ESPRET[7] (Tahvili

[7] EStimation and PRediction of Execution Time.

et al., 2018a) and sOrTES (Tahvili et al., 2019) for execution time prediction and test scheduling, respectively. Both tools are utilized at BT today, which in turn help us to modify our tools. Having an automated tool for dependency detection is demanded by our industrial partner and can be used for different purposes such as test automation, test selection and prioritization. Furthermore, the dependencies between each and every test case need to be detected as well. Hitherto, all test cases are divided into two main classes. The independent test cases can be executed without order or parallel with other test cases, but the execution order of the dependent class of test cases needs to be detected in the future.

Another future direction of this work is to reduce the training set even more (e.g. 15% or 10% instead of 20%) and also applying different architecture in the word embeddings such as the proposed adaptation of the Skip-gram model by Ye et al. (2016). Moreover, other text mining algorithms such as string-matching, token-based, edit-based algorithms (e.g. Levenshtein distance that previously employed for similarity detection between test cases Landin et al., 2020) can be considered as a replacement for the text analysis step of this work.

The goal is to minimize the human efforts and judgment for data labeling. Furthermore, in the limit we have planned to apply some unsupervised learning approaches such as clustering. However, the feasibility of the proposed methodology needs to be analyzed on different industrial testing projects.

CRediT authorship contribution statement

Sahar Tahvili: Conceptualization, Methodology, Contact with the industrial partner, Data collection. **Leo Hatvani:** Conceptualization, Methodology, Implementation (text analysis), Visualization. **Enislay Ramentol:** Conceptualization, Methodology, Implementation (classification). **Rita Pimentel:** Conceptualization, Methodology. **Wasif Afzal:** State of the art. **Francisco Herrera:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by VINNOVA through TESTOMAT project, the European Union's Horizon 2020 research and innovation program under grant agreement No 871319 and ERCIM "Alain Bensoussan" Fellowship Programme. The authors are thankful to Dr. Ola Sellin at Bombardier Transportation AB in Västerås, Sweden for his valuable comments and suggestions. Francisco Herrera would like to thank the Spanish Government for its funding support (SMART-DaSCI project, TIN2017-89517-P).

References

Alégroth, E., Feldt, R., Kolström, P., 2016. Maintenance of automated test suites in industry: An empirical study on visual GUI testing. Inf. Softw. Technol. 73, 66–80.

Arafeen, M.J., Do, H., 2013. Test case prioritization using requirements-based clustering. In: Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. IEEE Computer Society, USA.

Arlt, S., Morciniec, T., Podelski, A., Wagner, S., 2015. If A fails, can B still succeed? inferring dependencies between test results in automotive system testing. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST, pp. 1–10.

Bates, S., Horwitz, S., 1993. Incremental program testing using program dependence graphs. In: Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL'93, ACM.

Batista, G., Prati, R., Monard, M., 2004. A study of the behaviour of several methods for balancing machine learning training data. SIGKDD Explor. 6 (1), 20–29.

Bi, J., Zhang, C., 2018. An empirical comparison on state-of-the-art multi-class imbalance learning algorithms and a new diversified ensemble learning scheme. Knowl.-Based Syst. 158, 81–93.

Buda, M., Maki, A., Mazurowski, M., 2018. A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw. 106, 249–259.

Bunkhumpornpat, C., Sinapiromsaran, K., Lursinsap, C., 2009. Safe-Level-SMOTE: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Vol. 3644, pp. 475–482.

Chawla, N., Bowyer, K., Hall, L., Kegelmeyer, W., 2002. SMOTE: Synthetic minority over-sampling technique. J. Artif. Intell. Res. 16, 321–357.

Chechik, M., Salay, R., Viger, T., Kokaly, S., Rahimi, M., 2019. Software assurance in an uncertain world. In: Hähnle, R., van der Aalst, W. (Eds.), Fundamental Approaches to Software Engineering. Springer International Publishing, pp. 3–21.

Chen, I., Bastani, F., Tsao, T., 1995. On the reliability of AI planning software in real-time applications. IEEE Trans. Knowl. Data Eng. 7 (1), 4–13.

Chen, T., Thomas, S., Hassan, A., 2015. A survey on the use of topic models when mining software repositories. Empir. Softw. Eng. 21 (5), 1843–1919.

Chen, T., Thomas, S.W., Hemmati, H., Nagappan, M., Hassan, A.E., 2017. An empirical study on the effect of testing on code quality using topic models: A Case study on software development systems. IEEE Trans. Reliab. 66 (3), 806–824.

Cieslak, D.A., Hoens, T.R., Chawla, N.V., Kegelmeyer, W.P., 2012. Hellinger distance decision trees are robust and skew-insensitive. Data Min. Knowl. Discov. 24 (136–158).

Cozby, P., Rawn, C., 2012. Methods in Behavioural Research. McGraw-Hill Ryerson.

Elbaum, S., Malishevsky, A.G., Rothermel, G., 2002. Test case prioritization: a family of empirical studies. IEEE Trans. Softw. Eng. 28 (2), 159–182.

Fawcett, T., 2006. An introduction to ROC analysis. Pattern Recognit. Lett. 27, 861–874.

Fazzini, M., Prammer, M., d'Amorim, M., Orso, A., 2018. Automatically translating bug reports into test Cases for mobile apps. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2018, ACM, pp. 141–152.

Fernández, A., García, S., Galar, M., Prati, R., Krawczyk, B., Herrera, F., 2018. Learning from Imbalanced Data Sets. Springer.

Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F., 2012. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. IEEE Trans. Syst. Man Cybern. C 42 (4), 463–484.

Galar, M., Fernández, A., Barrenechea, E., Herrera, F., 2013. EUSboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. Pattern Recognit. 46, 3460–3471.

Gao, X., Chen, Z., Tang, S., Zhang, Y., Li, J., 2016. Adaptive weighted imbalance learning with application to abnormal activity recognition. Neurocomputing 173, 1927–1935.

Goffi, A., Gorla, A., Ernst, M.D., Pezzè, M., 2016. Automatic generation of oracles for exceptional behaviors. In: Proceedings of the 25th International Symposium on Software Testing and Analysis. ISSTA 2016, ACM, pp. 213–224.

Haidry, S., Miller, T., 2013. Using dependency structures for prioritization of functional test suites. IEEE Trans. Softw. Eng. 39 (2), 258–275.

Han, H., Wang, W., Mao, B., 2005. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. Springer-Verlag, pp. 878–887.

Huang, Y., Hung, C., Jiau, H., 2006. Evaluation of neural networks and data mining methods on a credit assessment task for class imbalance problem. Nonlinear Anal. RWA 7 (4), 720–747.

Huang, J., Ling, C., 2005. Using AUC and accuracy in evaluating learning algorithms. IEEE Trans. Knowl. Data Eng. 17 (3), 299–310.

Ilenic, N., 2017. A PyTorch implementation of Paragraph Vectors (doc2vec). GitHub repository, GitHub, https://github.com/inejc/paragraph-vectors.

Isaac, T., Salvador, G., Francisco, H., 2015. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowl. Inf. Syst. 42, 245–284.

Islam, M., Marchetto, A., Susi, A., Scanniello, G., 2012. A multi-objective technique to prioritize test Cases based on latent semantic indexing. In: Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering. IEEE Computer Society, USA.

Jensen, R., Cornelis, C., 2011. Fuzzy rough nearest neighbour classification and prediction. Theoret. Comput. Sci. 412 (42), 5871–5884.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., Mikolov, T., 2016. FastText.zip: Compressing text classification models.

Khan, S., Bennamoun, M., Sohel, F., Togneri, R., 2018. Cost-sensitive learning of deep feature representations from imbalanced data. IEEE Trans. Neural Netw. Learn. Syst. 29 (8), 3573–3587.

Khreich, W., Granger, E., Miri, A., Sabourin, R., 2010. Iterative boolean combination of classifiers in the ROC space: An application to anomaly detection with HMMs. Pattern Recognit. 43, 2732–2752.

Kotsiantis, S., Kanellopoulos, D., Pintelas, P., 2006. Data preprocessing for supervised learning. Int. J. Comput. Sci. 1, 111–117.

Kubat, M., Holte, R., Matwin, S., 1998. Machine learning for the oil spills in satellite radar images. Mach. Learn. 30, 195–215.

Landin, C., Tahvili, S., Haggren, H., Längkvist, M., Muhammad, A., Loutfi, A., 2020. Cluster-based parallel testing using semantic analysis. In: The Second IEEE International Conference on Artificial Intelligence Testing.

Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Vol. 32. ICML'14, JMLR, pp. II–1188–II–1196.

Leon, D., Podgurski, A., 2003. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In: 14th International Symposium on Software Reliability Engineering.

Lin, J., Wang, F., Chu, P., 2017. Using semantic similarity in crawling-based web application testing. In: 2017 IEEE International Conference on Software Testing, Verification and Validation, ICST.

Liu, Y., Loh, H., Sun, A., 2009. Imbalanced text classification: A term weighting approach. Expert Syst. Appl. 36 (1), 690–701.

Lopez, V., Fernandez, A., Garcia, S., Palade, V., Herrera, F., 2013. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. Inform. Sci. 250, 113–141.

Majumder, N., Poria, S., Gelbukh, A., Cambria, E., 2017. Deep learning-based document modeling for personality detection from text. IEEE Intell. Syst. 32 (2), 74–79.

Maslova, N., Potapov, V., 2017. Neural network doc2vec in automated sentiment analysis for short informal texts. In: Karpov, A., Potapova, R., Mporas, I. (Eds.), Speech and Computer. Springer International Publishing, Cham, pp. 546–554.

Mazurowskia, M., Habasa, P.A., Zuradaa, J., Lob, J., Bakerb, J., Tourassib, G., 2008. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. Neural Netw. 21, 427–436.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space, CoRR. pp. 1–12.

Nakagawa, H., Tsuchiya, T., 2015. Towards automatic constraints elicitation in pair-wise testing based on a linguistic approach: elicitation support using coupling strength. In: 2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing.

Nardo, D., Alshahwan, N., Briand, L., Labiche, Y., 2015. Coverage-based regression test Case selection, minimization and prioritization: A Case study on an industrial system. Softw. Test. Verif. Reliab. 25 (4), 371–396.

Parsa, M., Ashraf, A., Truscan, D., Porres, I., 2016. On optimization of test parallelization with constraint. In: 1st Workshop on Continuous Software Engineering Co-Located with Software Engineering, pp. 164–171.

Preeti, S., Milind, B., Narayan, M.S., Rangarajan, K., 2017. Building combinatorial test input model from use case artefacts. In: 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW.

Ramentol, E., Caballero, Y., Bello, R., Herrera, F., 2012. SMOTE-RSB$_*$: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory. Int. J. Knowl. Inf. Syst. 33, 245–265.

Ramentol, E., Vluymans, S., Verbiest, N., Caballero, Y., Bello, R., Cornelis, C., Herrera, F., 2015. IFROWANN: Imbalanced fuzzy-rough ordered weighted average nearest neighbor classification. IEEE Trans. Fuzzy Syst. 23 (5), 1622–1637.

Rehurek, R., Sojka, P., 2010. Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. ELRA, pp. 45–50.

Rekabsaz, N., Lupu, M., Hanbury, A., 2017. Exploration of a threshold for similarity based on uncertainty in word embedding. In: Jose, J.M., Hauff, C., Altıngovde, I.S., Song, D., Albakour, D., Watt, S., Tait, J. (Eds.), Advances in Information Retrieval. Springer International Publishing, Cham, pp. 396–409.

Robson, C., McCartan, K., 2011. Real World Research : A Resource for Users of Social Research Methods in Applied Settings, third ed. Wiley-Blackwell, Chichester, West Sussex ; Hoboken, N.J.

Runeson, P., Höst, M., 2008. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131.

Ryser, J., Glinz, M., 2000. Using dependency charts to improve scenario-based testing - Management of inter-scenario relationships: Depicting and managing dependencies between scenarios. In: Proceedings of the 17th International Conference on Testing Computer Software, TCS'00.

Stefanowski, J., Wilk, S., 2008. Selective pre-processing of imbalanced data for improving classification performance. In: Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery, DaWaK08, Turin, pp. 283–292.

Tahvili, S., Afzal, W., Saadatmand, M., Bohlin, M., Ameerjan, S.H., 2018a. ESPRET: A tool for execution time estimation of manual test Cases. J. Syst. Softw. 135, 1–43.

Tahvili, S., Ahlberg, M., Fornander, E., Afzal, W., Saadatmand, M., Bohlin, M., Sarabi, M., 2018b. Functional dependency detection for integration test cases. In: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C, pp. 207–214.

Tahvili, S., Bohlin, M., Saadatmand, M., Larsson, S., Afzal, W., Sundmark, D., 2016a. Cost-benefit analysis of using dependency knowledge at integration testing. In: Product-Focused Software Process Improvement. Springer International Publishing, pp. 268–284.

Tahvili, S., Hatvani, L., Felderer, M., Afzal, W., Bohlin, M., 2019. Automated functional dependency detection between test cases using doc2vec and clustering. In: 2019 IEEE International Conference on Artificial Intelligence Testing, AITest, pp. 19–26.

Tahvili, S., Hatvani, L., Felderer, M., Afzal, W., Saadatmand, M., Bohlin, M., 2018c. Cluster-based test scheduling strategies using semantic relationships between test specifications. In: Proceedings of the 5th International Workshop on Requirements Engineering and Testing. RET '18, ACM, pp. 1–4.

Tahvili, S., Pimentel, R., Afzal, W., Ahlberg, M., Fornander, E., Bohlin, M., 2019. SOrTES: A supportive tool for stochastic scheduling of manual integration test Cases. IEEE Access 1–19.

Tahvili, S., Saadatmand, M., Larsson, S., Afzal, W., Bohlin, M., Sundmark, D., 2016b. Dynamic integration test selection based on test case dependencies. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops, ICSTW, pp. 277–286.

Thomas, S., Hemmati, H., Hassan, A., Blostein, D., 2014a. Static test Case prioritization using topic models. Empir. Softw. Eng. 19 (1), 182–212.

Thomas, S., Hemmati, H., Hassan, A., Blostein, D., 2014b. Static test Case prioritization using topic models. Empir. Softw. Eng. 19 (1), 182–212.

Ting, K., 2002. An instance-weighting method to induce cost-sensitive trees. IEEE Trans. Knowl. Data Eng. 14 (3), 659–665.

Triguero, I., González, S., Moyano, J.M., García, S., Alcalá-Fdez, J., Luengo, J., Fernández, A., del Jesús, M.J., Sánchez, L., Herrera, F., 2017. KEEL 3.0: An open source software for multi-stage analysis in data mining. Int. J. Comput. Intell. Syst. 10 (1), 1238–1249.

Unterkalmsteiner, M., Gorschek, T., Feldt, R., Lavesson, N., 2016. Large-scale information retrieval in software engineering - an experience report from industrial application. Empir. Softw. Eng. 21 (6), 2324–2365.

Vapnik, V., 2013. The Nature of Statistical Learning Theory. Springer.

Witt, N., Seifert, C., 2017. Understanding the influence of hyperparameters on text embeddings for text classification tasks. In: Kamps, J., Tsakonas, G., Manolopoulos, Y., Iliadis, L., Karydis, I. (Eds.), Research and Advanced Technology for Digital Libraries. Springer International Publishing, Cham, pp. 193–204.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers.

Ye, X., Shen, H., Ma, X., Bunescu, R., Liu, C., 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In: 2016 IEEE/ACM 38th International Conference on Software Engineering, ICSE, pp. 404–415.

Yoo, S., Harman, M., 2007. Pareto efficient multi-objective test Case selection. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis. ISSTA '07, ACM, pp. 140–150.

Yoo, S., Harman, M., Tonella, P., Susi, A., 2009. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis. Association for Computing Machinery, New York, NY, USA.

Zadrozny, B., Langford, J., Abe, N., 2003. Cost-sensitive learning by cost-proportionate example weighting. In: Proceedings of the 3rd IEEE International Conference on Data Mining, ICDM 03, pp. 435–442.

Zalmanovici, M., Raz, O., Tzoref-Brill, R., 2016. Cluster-based test suite functional analysis. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Association for Computing Machinery, New York, NY, USA.

Zhang, X., Zhao, J., LeCun, Y., 2015. Character-level convolutional networks for text classification. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Vol. 1. NIPS'15, MIT Press, Cambridge, MA, USA, pp. 649–657.

Zhou, Z., Liu, X., 2010. On multi-class cost-sensitive learning. Comput. Intell. 26 (3), 232–257.