Identifying Similar Test Cases That Are Specified in Natural Language

Ritik Gandhi
rgandhi1@ualberta.ca
University of Alberta

Ranit Ganguly
ranit@ualberta.ca
University of Alberta

Amey Joshi
ameyanan@ualberta.ca
University of Alberta

Abstract

In today's rapidly advancing software industry, we experience exciting technological growth every year in an extensive range of fields such as innovative AI-powered development, cloud and edge computing, machine learning, progressive and lightweight web and mobile applications etc. However, it is not quite relevant to the Software Testing industry. Most companies still rely on the outdated manual testing process despite the availability of Automation testing procedures. It may work for small teams testing the software using a limited number of test cases. Although, with the increase in team size and the number of test cases over time, the cost, effort and time needed to manually validate and review the test cases increase tenfold. It often leads to recurrent and unclear test cases in the test suite, delivered by different employees who often work across teams. These test cases are represented in Natural Language. Also, the redundant test cases can impact the manual testing process by testing the same feature multiple times and can reduce the possibility of writing multiple methods to test the same feature when automating tests in the future. Hence, in this project, we propose to address the problem of similar test cases using an unsupervised learning approach. Additionally, after removing redundancy in the test suite, we intend to identify key features in the software to be tested based on the description of the test cases in the suite, and group multiple test cases into a software feature. This feature can be directly assigned to a Quality Assurance engineer for testing.

Related Work

In this section, we discuss prior work that applies Natural Language Processing techniques to Software Testing. Viggiato et al. [29] use an unsupervised approach to identify similar test cases using a combination of text embedding, text similarity and clustering techniques. The approach supports the creation and maintenance of a high-quality, more consistent and more standardized test suite. The automated framework as used in the paper provides feedback to improve the test cases designed to test the Prodigy math game which is an online web-based educational math game with millions of users around the world. Embedding techniques such as Word2Vec, BERT, Sentence-BERT, Universal Sentence Encoder and TF-IDF have been used. The two similarity metrics used in this paper are Word Mover's Distance and cosine similarity. Hierarchical Agglomerative Clustering and K-Means are used for evaluation. Following a modular approach, they start with test case pre-processing wherein they split the test cases into test steps, followed by test step clustering which involves computation of text embedding and applying clustering techniques to cluster test steps. The last stage involves grouping similar test cases by identifying test step clusters belonging to each test case. The approach upon evaluation achieved an F-score of 86.13% in the best case.

Pass et al. [30] proposed an automated framework which provides feedback on improving a newly added manual test case based on recommendations. The framework is used after a new test case is specified so that feedback can be provided to improve the test case description. The project is divided into 3 main components namely (1) Data preparation component in which existing test cases are pre-processed to generate new test cases. (2) Analysis Component wherein recommendations are provided such as improving the terminologies to a newly added test case based on the existing test cases via language modelling, through Frequent itemset and Association Rule mining, recommendations are given on potentially missing test cases and recommendations of similar test cases are given through Word Mover's Distance and Cosine Similarity. (3) Report Generation Component has the purpose to aggregate the outputs of each used analysis module and presenting results to QA engineers for testing. Word2Vec is used for embedding, Word Mover's Distance for computing between test step embeddings and K-means algorithm for clustering. An F-score of approximately 83% was in the best case.

Li et al. [18] automated test automation on WeChat, which is a large industrial mobile application. The key insight highlighted in the paper is that semantically similar test steps can be implemented by the same test API method. The dataset used has a simple grammatical structure, with synonyms being frequent. The authors have devised a new tool called Clustep which receives test cases and in turn generates step clusters. These step clusters generate the test API methods through which an executable test script is generated. They tend to achieve high clustering accuracy, with a lesser number of clusters and editable clustering results so that wrong clusters can be fixed easily. Starting with preprocessing, they perform word embedding training, followed by distance measurement and clustering and finally post refining the k-means clustering results. The proposed approach has a few drawbacks such as inappropriate handling of the main executor, bad word embedding for a few words, and ignorance of the difference between a few important words.

Goals and Objectives

Understanding if we can effectively identify similar test steps automatically allows us to reduce redundancy as much as possible and group similar test cases together for a more innovative approach. Also, a method will be utilized to find test cases that are related based just on their natural language descriptions. Since neither labelled data nor human supervision is required, we emphasize that the method is unsupervised. More specifically, to put the finishing touches on a technique for finding related test cases with natural language specifications, we will use text embedding, text similarity, and clustering approaches to group related results that make up test cases, and then we evaluate test cases based on how similar they are to one another in terms of the steps that are in the same cluster. We hope to aid developers and the quality assurance team by enhancing the effectiveness and productivity of testing by removing similar test cases that were previously a part of the test suite. Finally, as redundant test cases can affect manual testing by testing the same feature more than once, and as they can reduce the possibility of writing multiple methods to test the same feature when automating tests in the future, our method should significantly increase productivity and decrease downstream manual work in a context with large industry.

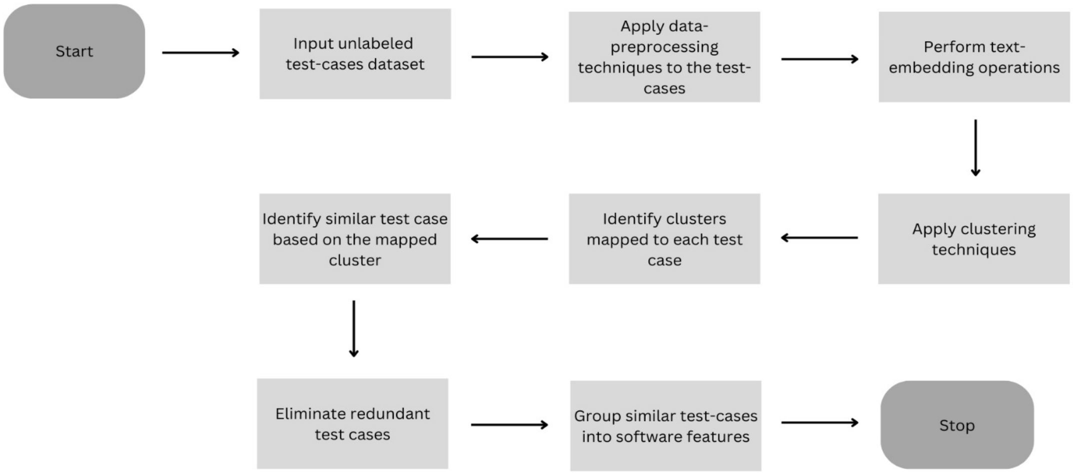

Start
Input unlabeled test-cases dataset
Apply data-preprocessing techniques to the test-cases
Perform text-embedding operations
Apply clustering techniques
Identify clusters mapped to each test case
Identify similar test case based on the mapped cluster
Eliminate redundant test cases
Group similar test-cases into software features
Stop

Figure 1: Flowchart depicting the key aspects of the project

Literature Review

Review of Test Automation and Prioritization

Thummalapenta et al. [27] propose an approach to infer a sequence of action-target-data tuples from manually written test cases in Natural Language. Each tuple consists of an action, a target user interface and a data value. A manual test case is converted into a non-deterministic program that encodes all alternative interpretations of the test steps. However, the paper fails to detect similarities between test steps and generates DOM UI test cases only for web applications and not for generalized test steps. By categorizing test artifacts into test entities and test morphisms, it combines test techniques to produce test sets that sufficiently cover mutant test situations, Zhu et al. [33] automation presents an automated tool for metamorphic testing. The authors describe a sentence generator powered by NLP that may be utilized by well-known automated testing tools and frameworks like Selenium as test input for conversational AI bots.

Ranking test cases is critical when test execution is time-consuming because the goal is to find the error as quickly as possible. The current case prioritizing system bases its decisions on code coverage and necessitates access to the source code for black box manual testing. The difficulty in Test-Case prioritization is that they are written in everyday language [10]. Black-box meta-data, test case history, and descriptions of the test cases in natural language are used for prioritization. SVM ranking is used to assess the quality of two subject systems. A binary linear classifier that assigns new instances to one category or the other is created by the SVM training method. Utilizing the APFD metric, the technique is evaluated on the test systems for early failure identification [15]. The effectiveness of prioritization strategies is improved by the requirements-based grouping methodology. Lines of code settled square depth, and cyclomatic complexity is measures of program complexity that are the main factors to determine the priority [14].

Review of Word Embedding techniques

The word2vec algorithm by Mikolov et al. is used for computing continuous vector representations of words from very large data sets and demonstrates that the algorithm performs well in word similarity tests. In this technique, billions of words are learned from large data sets, each of which contains millions of words. The output is a vector representation with multiple degrees of similarity [20]. As part of research entitled 'Word Embeddings for the Software Engineering Domain,' a word2vec model was trained on textual data from Stack Overflow posts to capture semantic information targeting information retrieval tasks in software engineering aiming to develop a general-purpose lexical model to represent software engineering knowledge. As a result, discussion topics, duplicate questions and semantically linked questions can be identified, code snippet usability and social aspects can be assessed [7].

BERT uses a multi-layer bidirectional Transformer encoder as its model architecture. Larger BERT models lead to strict accuracy improvements when trained with different numbers of layers, hidden units, and attention heads. A larger model size will result in continued improvements on large-scale tasks, as well as large improvements on very small-scale tasks provided that the model is sufficiently pre-trained [6]. One research application is 'BERT-Based Sentiment Analysis: A Software Engineering Perspective', which uses BERT-based models to analyze sentences from GitHub comments, Jira comments, and Stack Overflow posts to improve software engineering tools. The study proposes a robust method of classifying SE-specific datasets [3].

Another word embedding technique is TF-IDF which computes the importance of a word to a document by combining the word frequency in the document and the word frequency across all other documents. Numeric vectors for each document are built using the word importance values. Zero is assigned to the words that are not present in the document. TF-IDF gives importance to uncommon words rather than treating all words as equal in the case of a binary bag of words model. In this case, [13], test steps are considered as documents. Albitar et al. [2] propose a new measure called SemTFIDF for finding semantic similarity between concepts representing the compared text documents pair-to-pair. Semantics is involved in Supervised Text classification involved during indexing, training and prediction. The authors also conclude that semantic similarities are more adequate than classical similarities like Cosine similarities in comparing texts.

Review of Sentence Embedding techniques

Le et al. [16] presented doc2vec as a simple extension to word2vec to embed word sequences in a document. The doc2vec is a method for learning fixed-length feature representations from variable-length text pieces, including sentences, paragraphs, and documents. In this model, word vectors are trained to predict words in paragraphs. Word vectors are shared among paragraphs, but paragraph vectors are unique [1]. An application of doc2vec is 'Bug Prediction Using Source Code Embedding' where the source code is represented by traversing the abstract syntax tree, then multiple doc2Vec models are trained using different parameters. The trained models are used to generate fixed-length vectors derived from the source code. For bug prediction, the sequences generated from tokens are treated as documents, and a fixed-size vector is assigned to each class.

BERT has achieved state-of-the-art performance on supervised tasks such as semantic textual similarity. However, because of its construction, it is unsuitable for unsupervised tasks such as clustering. Sentence-BERT (SBERT) is a modification of the BERT network. SBERT searches for semantically similar sentences and clusters them using a siamese network architecture and similarity measures. As a result, SBERT can be used for both semantic similarity searches and clustering. Lastly, SBERT is computationally efficient and is suitable for modeling tasks that are computationally infeasible with BERT [24]. The paper 'An Unsupervised Sentence Embedding Method by mutual Information Maximization' uses an unsupervised sentence embedding model with a light-weight feature extractor on top of BERT for sentence encoding and trains it with the objective that maximizes the mutual information (MI) between the global sentence embedding and all its local contexts embeddings [32].

Universal Sentence Encoder proposed by Cer et al. [4] is used for NLP tasks by encoding sentences into embedding vectors. The model maps sentences to a fixed-length vector representation which encodes the meaning of the sentences. Naive techniques to get sentence embeddings have challenges which involve loss of information and no proper ordering. The Universal sentence encoder summarizes any given sentence into a 512-dimensional sentence embedding. Upon multiple iterations, it captures the most informative features and discards noise. In the first step, the sentences are tokenized using the Penn Treebank (PTB) tokenizer. Two architectures for the encoder have been proposed based on the differences in accuracy and inference speed.

The first variant is a transformer encoder [28] which consists of 6 stacked transformer layers, with each layer having a self-attention module followed by a feed-forward network. Word order is taken into consideration for each word representation and a 512-dimensional vector is generated as output sentence embedding. A drawback of this architecture is high memory usage due to complex architecture. The second variant is based on Deep Averaging Network (DAN) [12] proposed by Iyyer et al. The embeddings for word and bi-grams are passed through a 4-layer feed-forward deep DNN and a 512-dimensional sentence embedding is generated as the output. The embeddings for the same are learned during training. Compared to [28] it has a slightly reduced accuracy but the inference time is efficient and compute time is of linear complexity.

Review of Similarity Techniques

Text similarity measures are used in information retrieval, text classification, document clustering, topic detection, tracking, question generation, answering, essay scoring, short answer scoring, machine translation, text summarization and others [9]. Fry et al. use a metric that multiplies the frequencies of terms in two documents by the idf weight of those terms. The aggregate sum over all words' values then serves as the similarity measure for those two documents [8]. Additional three different approaches to text similarities are String-based, Corpus-based and Knowledge-based similarities. String similarity operates on string sequences and character composition. Latent Semantic Analysis (LSA) uses a mathematical technique called singular value decomposition (SVD) to reduce the number of columns while preserving the similarity structure among rows. Knowledge-Based Similarity is based on information derived from semantic networks [9]. Another Text-Similarity method is Distance Weighted Cosine Similarity Measure. Cosine is calculated as a dot-product of two normalized vectors and is overly biased by the features of higher values [17].

Review of Clustering techniques

Walting et al. [31] propose a test case synthesis method wherein after removing redundant test steps, all test steps are arranged into a new set of test cases. A path-finding algorithm is used to find an optimized test step execution for each test case. Along with logic operators, the natural language descriptions of test cases are converted into a representation form of parameters. However, all test steps are supposed to have a formal description of precondition, action and postcondition. Chetouane et al. [5] focus on test suite minimization, i.e finding a subset of redundant test cases from the actual test suite. An approach using K-means clustering along with binary search is proposed. Binary search is used for looking for the exact number of clusters that allows the test suite to not deviate from the mutation score obtained in the initial test suite. However, a potential drawback is that this approach must have source codes of the test cases. Pei et al. [21] use a technique called Dynamic Random Testing (DRT) strategy which guides the test case selection by using testing results and in turn enhances the fault detection effectiveness. The vectorized test cases are further classified into subdomains through clustering methods. Clustering techniques such as K-means clustering, hierarchical clustering and K-medoids were used. The distance matrix is then calculated which identifies the similarities based on the classification results. The evaluation was done using Java programs and it was concluded that fault detection effectiveness was achieved with a low computational cost.

Using the deterministic annealing (DA) approach to clustering, this paper discusses an optimization problem in clustering. DA aims at finding the global minimum of an energy function, instead of getting greedily attracted to a nearby local minimum. It is a useful approach to clustering and related optimization problems that are derived from fundamental principles and are completely independent of the initial configuration used [26].To determine centroid distributions, two connected problems are solved: identifying appropriate forms for cluster membership and centroid distributions and maximizing the degree of a good fit between clusters and clusters. A two-stage iterative process, similar to the EM method, is used to determine the membership probabilities by maximizing the relative entropy between the centroid distributions of objects and clusters. In the presence of a critical value, the original cluster splits into two new clusters. We then search for the lowest free energy (local) minimum for a single leaf cluster split. The procedure is then repeated until the desired number of clusters is achieved [22].

This paragraph highlights document clustering techniques using an unsupervised approach based on language modeling and wordnet-based similarity measures. Agglomerative Clustering algorithm to determine the set of flat clusters for each different threshold defined by the joining points for the polish language dataset [19]. Different types of clustering methods can also be categorized between hierarchical and partitioning methods. Hierarchical algorithms cannot scale well and have huge I/O costs. Partitioning methods move instances between clusters, typically by iterative optimization. To achieve global optimality, an exhaustive enumeration process of all possible partitions is required. As here an application of the K-means algorithm is that it partitions the data into K clusters, represented by their centers or means. Khan et al. [25] found that K -means was the most efficient method in terms of execution time and that GA obtained the best solution faster than TS and SA. There are several methods to improve the performance of k-means. In order to improve the probability of a small cluster getting a seed in k-means initializing, we extend the value of k. In the Hierarchical K-means clustering algorithm, we use the random initialization method to choose k starting centers, assign points into k clusters, get feature values of mean for each cluster, perform hierarchical clustering along with k-means adjusting iteration, and perform top-n nearest clusters merging [23]. Finally the important five measures for text document clustering: are Euclidean distance, cosine similarity, Jaccard coefficient, Pearson correlation coefficient and averaged Kullback-Leibler divergence. The averaged Kullback-Leibler divergence showed the highest effectiveness in clustering text. The study found that except for the Euclidean distance measure, the other measures have comparable effectiveness for the partitional text document clustering task [11].

Dataset

Li et al. [18] studied natural language test cases of a large industrial app (WeChat). Viggiato et al.[29] studied natural language test cases of Prodigy Game which is an educational math game company. Their dataset consisted of 3,233 test cases since the company provided the data.Since we did not have a publicly available dataset related to software testing cases we decided to create our own dataset based on which we could test whether our method works or not. We decided to manually write our own test cases along with test steps based on different software testing parameters. Based on our research we tried to incorporate different test cases related to databases, OTP Validation, Profile Creation, Injection Attacks and many more and finally created 127 test cases and 369 test cases. To test our methodology we duplicated certain test cases by changing the tense of the sentence, adding similar words to those test steps to check whether the embedding techniques perform well or not, however kept the interpretation the same so that we could check whether we are able to remove the redundant test cases or not.

Type	**Key**	**Case_Name**	**Step_ID**	**Steps**
OTP Validation	TC1	Check if User is able to edit the mobile number	1	Enter the phone number
OTP Validation	TC1	Check if User is able to edit the mobile number	2	Send OTP
OTP Validation	TC1	Check if User is able to edit the mobile number	3	Click edit button
OTP Validation	TC2	Verify if ten digit number is accepted	1	Enter ten digit phone number
OTP Validation	TC2	Verify if ten digit number is accepted	2	Send OTP button should become active
OTP Validation	TC3	Verify if OTP is received on user number	1	Enter the phone number
OTP Validation	TC3	Verify if OTP is received on user number	2	Send OTP
OTP Validation	TC3	Verify if OTP is received on user number	3	Check inbox of the mobile phone

Table 1: Sample of the dataset

Implementation

Test-case similarity using Clustering

The data obtained as the input [Refer: Table 1] contains multiple columns which include *Type* which corresponds to a feature in which this test case exists, the *Key*, which is a unique identifier for each test case, the *Case_Name*, which refers to the title of the test-case. Now, a test case involves multiple steps, so the *Step_ID* column refers to the identifier for each of the test steps in a test case. Lastly, column *Steps* refers to a description for verifying the test case written in natural language. We perform pre-processing on this data to generate a result as shown in Figure-3. In order to achieve this, the following steps are performed for the columns *Case_Name* and *Steps.* Initially, we get the number of unique words in the data set for both columns followed by the frequency for each word in order to find the words which have the highest and lowest frequencies. Next data cleaning is performed where we

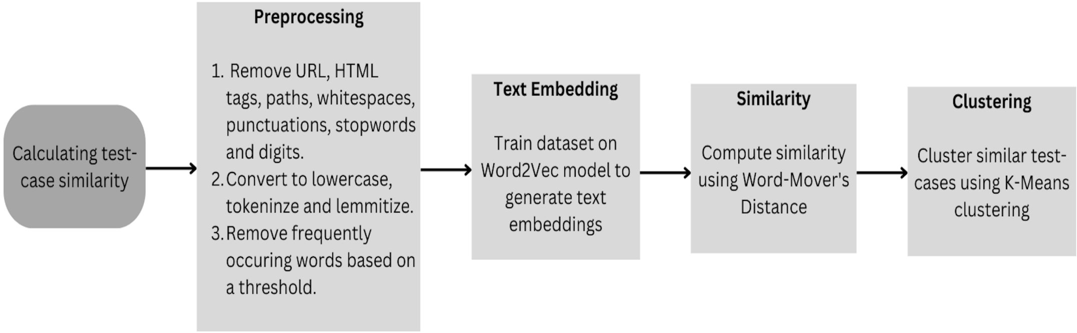
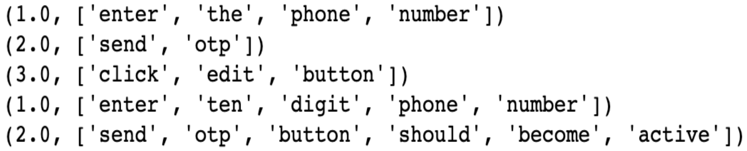

Calculating test-case similarity
Preprocessing
1. Remove URL, HTML tags, paths, whitespaces, punctuations, stopwords and digits.
2. Convert to lowercase, tokeninze and lemmitize.
3. Remove frequently occuring words based on a threshold.
Text Embedding
Train dataset on Word2Vec model to generate text embeddings
Similarity
Compute similarity using Word-Mover's Distance
Clustering
Cluster similar test-cases using K-Means clustering

Figure 2: Block diagram explaining the steps involved in clustering similar test cases

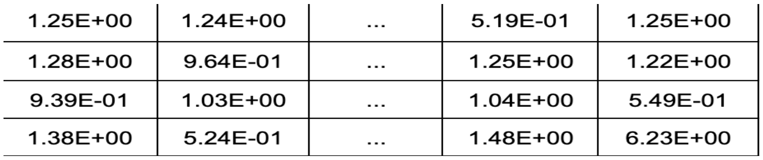
initially clean the URLs, paths, HTML tags, and punctuation, and remove extra spaces and stop-words. It is followed by converting the strings into lowercase followed by removing digits and words which have digits. Lastly, we tokenize, lemmatized and remove words which are repeated more than a certain number of times.



```
(1.0, ['enter', 'the', 'phone', 'number'])
(2.0, ['send', 'otp'])
(3.0, ['click', 'edit', 'button'])
(1.0, ['enter', 'ten', 'digit', 'phone', 'number'])
(2.0, ['send', 'otp', 'button', 'should', 'become', 'active'])
```

Figure 3: Tuple after preprocessing

The next step involves performing training on a pre-trained Word2Vec model using [7] having a dimensionality of 200, a context of 2, a downsampling of 0.001 and using Continious-Bag-of-Words (CBOW). The pre-trained model is then updated using the new vocabulary from our training corpus. Next, we build tuples as shown in Figure-2 with the *(Step_ID, Step)* which is used to retrieve the step ID in the end after the clustering and get only test steps for clustering. The second step involves creating and initializing a distance matrix with zeros and assigning the number of test steps to rows and columns. Then, we compute the distances between the test-step rows and columns using the Word-Movers Distance and save the distance matrix as shown in Figure-4. The average word vector of a test step is then calculated.



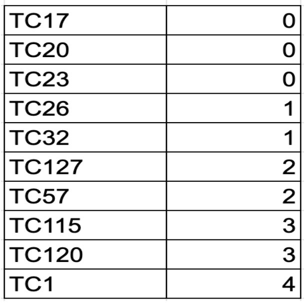
1.25E+00	1.24E+00	...	5.19E-01	1.25E+00
1.28E+00	9.64E-01	...	1.25E+00	1.22E+00
9.39E-01	1.03E+00	...	1.04E+00	5.49E-01
1.38E+00	5.24E-01	...	1.48E+00	6.23E+00

Figure 4: Similarity matrix using word mover's distance

Lastly, we perform K-means clustering on the distance matrix obtained from the previous step where we calculate the average sentence vector between the selected sentences. As a result, we obtain the file which shows the clusters with similar test steps. This is then provided as input for the ensembling approach.

Ensembling approach and computing similarity

Since we performed different embedding techniques we tend to obtain different clusters of test steps. To get better clarity for certain tasks such as classification and clustering we use an ensemble approach that uses majority voting similar to [29]. The clusters are generated by getting the set of all test steps in the data. We then perform an iteration through each of the test steps and perform a pairwise comparison with other test steps. The majority voting technique also assigns a threshold value and it is decided whether the particular test step belongs to the same cluster or to a different cluster. After this, we have the test steps that belong to the same cluster. After the ensemble approach, we try different techniques to identify similar test cases. The first technique that we tried uses Simple Overlap to compute groups of similar test cases. We use the identifiers of test step clusters to represent test cases. A simple overlap metric is used to determine the pairwise similarity, which indicates the proportion of overlap that test cases have. We also vary the threshold value to find out the optimal similarity threshold. The second technique that we tried uses a binary representation of test cases, wherein we generate a binary vector for each test case. A binary matrix is created where the row corresponds to the test case and the column corresponds to the test step. We finally compute the pairwise similarity of the test cases using the Jaccard index. If the Jaccard index is greater than a certain threshold value then we identify the corresponding test cases to be similar. Finally, the third technique used was representing the test steps as a numeric vector and the numeric vector corresponds to the number of test steps that the test case has in each cluster. Cosine similarity between the test steps was then computed between the test steps to identify whether they are similar or not. The weighted sum between the similarity score obtained with test step clusters is used to compute the final similarity score for test cases.



TC17	0
TC20	0
TC23	0
TC26	1
TC32	1
TC127	2
TC57	2
TC115	3
TC120	3
TC1	4

Figure 5: Output generated after computing the similarity

The test cases are then clustered based on the similarity score. Test cases that are similar or have similar test steps are assigned the same similarity score as shown in Figure 5. After computing the similarity, we then remove the redundant test steps and test cases based on the test steps which we computed using the ensemble approach. Finally, we remove the redundant test cases and test steps from the input file and generate a new output file with all the non-redundant test cases and test steps.

Results

Based on the different embedding and clustering techniques that we used, we were able to achieve considerable results and functionalities wherein we were able to identify similar test steps, identify similar test cases and remove the redundant test cases and steps as well. For the 127 test cases we had in the input file, we duplicated 5 of the test cases in a different manner and were able to successfully remove those redundant 5 test cases from the output file. Although the size of the input file is small and the number of test cases were less as well , we believe there might be certain edge cases where our implementation might not be able to remove all the redundant test cases. Based on the different word embedding and sentence embedding techniques that we adapted we are also trying different techniques such as fasttext and infersent and study the similarity metrics so that we are able to use them for our purpose as well.

Type	Key	Case_Name	Step_ID	Steps
Profile creation Validation	TC17	Verify if it allows to Click on ”Next”	1	Leave any of the parameter fields empty
Profile creation Validation	TC17	Verify if it allows to Click on ”Next”	2	Check for the appropriate error message from frontend
Profile creation Validation	TC18	Verify if ”Go back” is working	1	Press back button from Step two screen
Profile creation Validation	TC18	Verify if ”Go back” is working	2	Should return back to Step one Screen
Profile creation Validation	TC19	Verify if data is retained after user presses ”Go Back”	1	Press back button from Step two screen
Profile creation Validation	TC19	Verify if data is retained after user presses ”Go Back”	2	All user input data should be retained
Profile creation Validation	TC20	Verify if data is retained after user presses ”Go Back”	1	Leave any of the parameter fields empty
Profile creation Validation	TC20	Verify if data is retained after user presses ”Go Back”	2	Check for the appropriate error message from frontend

Table 2: A sample of the input file having redundant test steps

To be able to understand the removal of redundant test cases, an example has been shown. In Table 2 we have a common test type which is Profile creation Validation along with 4 different test cases and their corresponding test steps. Although TC19 and TC20 have the exact same test case description, they are not similar because they have different test steps and hence the purpose of those test cases becomes completely different. However TC17 and TC20 have the same test steps and functionality as highlighted despite having different test case descriptions. Table 3 shows the sample from the output file wherein TC20 along with other redundant test cases has been removed and the test team finally gets a new output file with all the new test cases and test steps without any redundancy which will save a lot of time and resources.

Type	**Key**	**Case_Name**	**Step_ID**	**Steps**
Profile creation Validation	TC17	Verify if it allows to Click on ”Next”	1	Leave any of the parameter fields empty
Profile creation Validation	TC17	Verify if it allows to Click on ”Next”	2	Check for the appropriate error message from frontend
Profile creation Validation	TC18	Verify if ”Go back” is working	1	Press back button from Step two screen
Profile creation Validation	TC18	Verify if ”Go back” is working	2	Should return back to Step one Screen
Profile creation Validation	TC19	Verify if data is retained after user presses ”Go Back”	1	Press back button from Step two screen
Profile creation Validation	TC19	Verify if data is retained after user presses ”Go Back”	2	All user input data should be retained

Table 3: Output sample file generated after removing redundant test steps

Future Scope

The main intention for the future of this project is to overcome limitations that were faced during the process; some of which are identified and mentioned below.
1. Increase size of the dataset: One way to increase the size of our dataset is to collect more data through various methods. Here are a few ideas:

- Web scraping: we can use tools like Beautiful Soup or Selenium to scrape data from websites and add it to our dataset.

- Surveys and experiments: we can conduct surveys or experiments to collect data from participants.

- Crowdsourcing: we can use platforms like Amazon Mechanical Turk to gather data from a large number of people.

- Public datasets: There are many public datasets available online that we can use to supplement our own data. Some examples include the UCI Machine Learning Repository, Kaggle datasets, and data from government agencies.

  It's important to keep in mind that the quality of the data is also important, not just the quantity. Make sure to validate the data and ensure it is accurate and relevant to our research or project.

2. Try similarity metrics on fasttext and infersent: Using FastText for unsupervised learning we got incorrect similarity outputs as we carried out training the data with more datasets.As with fairly small size of dataset the quality cannot be ensured. Furthermore, having issues with the model after multiple trials we found it worth considering different model. Different approaches to text classification and similarity, and it may be that a different approach is more suitable for our specific problem. We were unable to use Infersent due to computing limitations, there may be other NLP models or approaches that we can use that are more suitable for our specific needs and resources. For example, we may be able to use a smaller or less resource-intensive model, or we could try using pre-trained word embeddings instead of a full NLP model.

3.Explore more different clustering techniques such as Gaussian mixture model: A probabilistic model called a Gaussian mixture model (GMM) posits that the underlying data was produced by combining a limited number of Gaussian distributions with unknowable parameters. Because it uses a soft clustering method, each data point is given a chance of becoming a member of each cluster rather than being categorically allocated to one. GMM has the ability to model more complex distributions than k-means, which presumes that the clusters are spherical and of equal size, is one advantage of GMMs. Clusters with varying sizes and forms can also be handled by GMMs. GMMs can, however, be sensitive to how the parameters are initialised, and selecting the right number of mixing components can be difficult.

4. Try and achieve a better F score once we have a proper dataset: To improve the F-score of a model, we can try the following strategies:

- Collect more and high-quality data: The model may be better able to generalise and generate more precise predictions with additional data. Similar to this, having high-quality data—data that is accurate and representative of the target population—can also enhance the performance of the model.

- Fine-tune the model's hyperparameters: Different hyperparameters control the behaviour of various models. The model's performance can be enhanced by tuning these hyperparameters. we can change the maximum depth of the tree, the minimum number of samples needed to divide a node, and other parameters in a decision tree model, for instance.

- Use a different model: We can experiment with employing a new model that might be more suited to the task if the one we are using isn't performing as expected. A non-linear model may produce superior results, for instance, if we are using a linear model and the data is significantly non-linear.

- Use feature engineering: The quality and relevance of the input features can have a significant impact on the model's performance. we can try different feature engineering techniques, such as feature selection, dimensionality reduction, or creating new features, to see if they improve the model's F-score.

- Use class weights or imbalance handling techniques: The model might be biased in favour of the class that is more prevalent if the classes in the dataset are unbalanced (i.e., there are noticeably more cases of one class than the other). In certain circumstances, class weighting or imbalance handling strategies can aid in raising the model's F-score.

Conclusion

In this research, we suggest an automated methodology for identifying and eliminating similar test cases by automatically analyzing and producing output. We go over various analysis modules that have so far been used with our framework. The modules are capable of recommending improvements to the recommendations of similar test cases that already exist in the test suite. Also, on average, our association rules can correctly recommend redundant test steps most of the time per test case. Finally, we can identify similar test cases with high performance using text embedding, text similarity, and clustering techniques. Our suggested framework automatically analyzes test cases written in plain language using a creative and effective method of fusing conventional and cutting-edge methodologies. The framework is capable of offering practical advice, which was the significant challenge presented to us, and taking into account the multiple recurrences of test cases in the software business (in particular, the Testing industry).

References

[1] Tamás Aladics, Judit Jász, and Rudolf Ferenc. Bug prediction using source code embedding based on doc2vec. In *International Conference on Computational Science and Its Applications*, pages 382–397. Springer, 2021.

[2] Shereen Albitar, Sébastien Fournier, and Bernard Espinasse. An effective tf/idf-based text-to-text semantic similarity measure for text classification. In *International Conference on Web Information Systems Engineering*, pages 105–114. Springer, 2014.

[3] Himanshu Batra, Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Bert-based sentiment analysis: A software engineering perspective. In *International Conference on Database and Expert Systems Applications*, pages 138–148. Springer, 2021.

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[5] Nour Chetouane, Franz Wotawa, Hermann Felbinger, and Mihai Nica. On using k-means clustering for test suite reduction. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 380–385. IEEE, 2020.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Vasiliki Efstathiou, Christos Chatzilenas, and Diomidis Spinellis. Word embeddings for the software engineering domain. In *Proceedings of the 15th international conference on mining software repositories*, pages 38–41, 2018.

[8] Zachary P Fry and Westley Weimer. Fault localization using textual similarities. *arXiv preprint arXiv:1211.2858*, 2012.

[9] Wael H Gomaa, Aly A Fahmy, et al. A survey of text similarity approaches. *international journal of Computer Applications*, 68(13):13–18, 2013.

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

[11] Anna Huang et al. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, volume 4, pages 9–56, 2008.

[12] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 1681–1691, 2015.

[13] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[14] Rayapureddy Kalyani, Padmanabhuni Sai Mounika, Ravipati Naveen, Gnaneswari Maridu, and Paruchuri Ramya. Test case prioritization using requirements clustering. *International Journal of Applied Engineering Research*, 13(15):11776–11780, 2018.

[15] Remo Lachmann, Sandro Schulze, Manuel Nieke, Christoph Seidl, and Ina Schaefer. System-level test case prioritization using machine learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 361–368. IEEE, 2016.

[16] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[17] Baoli Li and Liping Han. Distance weighted cosine similarity measure for text classification. In *International conference on intelligent data engineering and automated learning*, pages 611–618. Springer, 2013.

[18] Linyi Li, Zhenwen Li, Weijie Zhang, Jun Zhou, Pengcheng Wang, Jing Wu, Guanghua He, Xia Zeng, Yuetang Deng, and Tao Xie. Clustering test steps in natural language toward automating test automation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1285–1295, 2020.

[19] Michał Marcińczuk, Mateusz Gniewkowski, Tomasz Walkowiak, and Marcin Bedkowski. Text document clustering: Wordnet vs. tf-idf vs. word embeddings. In *Proceedings of the 11th Global Wordnet Conference*, pages 207–214, 2021.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[21] Hanyu Pei, Beibei Yin, Min Xie, and Kai-Yuan Cai. Dynamic random testing with test case clustering and distance-based parameter adjustment. *Information and Software Technology*, 131:106470, 2021.

[22] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of english words. *arXiv preprint cmp-lg/9408011*, 1994.

[23] Jianpeng Qi, Yanwei Yu, Lihong Wang, Jinglei Liu, and Yingjie Wang. An effective and efficient hierarchical k-means clustering algorithm. *International Journal of Distributed Sensor Networks*, 13(8):1550147717728627, 2017.

[24] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[25] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[26] Kenneth Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239, 1998.

[27] Suresh Thummalapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra. Automating test automation. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 881–891. IEEE, 2012.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[29] Markos Viggiato, Dale Paas, Chris Buzon, and Cor-Paul Bezemer. Identifying similar test cases that are specified in natural language. *IEEE Transactions on Software Engineering*, pages 1–1, 2022.

[30] Markos Viggiato, Dale Paas, Chris Buzon, and Cor-Paul Bezemer. Using natural language processing techniques to improve manual test case descriptions. In *International Conference on Software Engineering-Software Engineering in Practice (ICSE-SEIP) Track.(May 8, 2022)*, 2022.

[31] Benedikt Walter, Maximilian Schilling, Marco Piechotta, and Stephan Rudolph. Improving test execution efficiency through clustering and reordering of independent test steps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 363–373. IEEE, 2018.

[32] Yan Zhang, Ruidan He, Zuozhu Liu, Kwan Hui Lim, and Lidong Bing. An unsupervised sentence embedding method by mutual information maximization. *arXiv preprint arXiv:2009.12061*, 2020.

[33] Hong Zhu, Ian Bayley, Dongmei Liu, and Xiaoyu Zheng. Automation of datamorphic testing. In *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pages 64–72. IEEE, 2020.