Comparative analysis of text mining and clustering techniques for assessing functional dependency between manual test cases

Citation for the original published paper (version of record):
Tahvili, S., Hatvani, L., Felderer, M. et al (2025). Comparative analysis of text mining and clustering techniques for assessing functional
dependency between manual test cases. Software Quality Journal, 33(2).
http://dx.doi.org/10.1007/s11219-025-09722-7

Software Quality Journal (2025) 33:24
https://doi.org/10.1007/s11219-025-09722-7

RESEARCH

Comparative analysis of text mining and clustering techniques for assessing functional dependency between manual test cases

Sahar Tahvili[1,2] · Leo Hatvani[2] · Michael Felderer[3,4] · Francisco Gomes de Oliveira Neto[5,7] · Wasif Afzal[2] · Robert Feldt[5,6]

Accepted: 23 April 2025

Abstract
Text mining techniques, particularly those leveraging machine learning for natural language processing, have gained significant attention for qualitative data analysis in software testing. However, their complexity and lack of transparency can pose challenges, especially in safety-critical domains where simpler, interpretable solutions are often preferred unless accuracy is heavily compromised. This study investigates the trade-offs between complexity, effort, accuracy, and utility in text mining and clustering techniques, focusing on their application for detecting functional dependencies among manual integration test cases in safety-critical systems. Using empirical data from an industrial testing project at ALSTOM Sweden, we evaluate various string distance methods, NCD compressors, and machine learning approaches. The results highlight the impact of preprocessing techniques, such as tokenization, and intrinsic factors, such as text length, on algorithm performance. Findings demonstrate how text mining and clustering can be optimized for safety-critical contexts, offering actionable insights for researchers and practitioners aiming to balance simplicity and effectiveness in their testing workflows.

Keywords Artificial intelligence · Clustering · Natural language processing · Text mining · Software testing

✉ Sahar Tahvili
sahar.tahvili@mdu.se; sahar.tahvili@ericsson.com

1 Compute Platforms Engineering Unit, Ericsson AB, Stockholm, Sweden

2 Department of Industrial AI Systems, Mälardalen University, Västerås, Sweden

3 Institute of Software Technology, German Aerospace Center, Cologne, Germany

4 Department of Mathematics and Computer Science, University of Cologne, Cologne, Germany

5 Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden

6 Department of Computer Science and Engineering, Blekinge Institute of Technology, Karlskrona, Sweden

7 Department of Computer Science and Engineering, University of Gothenburg, Gothenburg, Sweden

Published online: 17 May 2025

1 Introduction

Software testing is a crucial activity for ensuring the quality of software-based systems. However, it is labor-intensive, particularly at the system level, where natural language requirements are manually transformed into test cases executed by human testers (Tahvili & Hatvani, 2022; Felderer et al., 2023).

Various Natural Language Processing (NLP) approaches have been proposed to automate the conversion of requirements into test cases. A systematic review of 67 papers on NLP-assisted software testing identified 38 tools (Garousi et al., 2020), but only a small fraction of them (11%) were available for download, and many papers (45%) provided only a superficial exploration of NLP aspects. While machine learning (ML) and artificial intelligence (AI) have entered software engineering, they can only support but not fully replace conventional testing, which relies heavily on human judgment. AI is used cautiously in testing, with the potential to combine AI-generated recommendations with human verification. However, the complexity of certain AI solutions may impede their adoption in the industry (Lönnfalt et al., 2024). Therefore, achieving the right balance between complexity, accuracy, effort, and utility is essential to bridge the gap between research advancements and industrial applications. This study focuses exclusively on test cases written in structured natural language templates, which are commonly used in industrial software testing. It does not address test cases formatted as tuples of values, structured data models, or executable test programs. Our objective is to analyze dependencies and clustering techniques applied to these natural language test specifications, ensuring that the proposed methods are directly applicable to real-world test case management in software development.

Basic algorithms typically involve straightforward, interpretable techniques, such as keyword matching or rule-based approaches, which rely on predefined patterns and explicit rules. In contrast, advanced algorithms often leverage sophisticated techniques, such as machine learning models, deep learning architectures, or ensemble methods, which can capture complex patterns and relationships in data but may be harder to interpret and implement. Building on the premise that advanced algorithms might provide better performance in dependency detection compared to basic ones, this study explores the following research question:

*RQ*1 *Do more complex text mining algorithms outperform basic algorithms in detecting dependencies between test cases?*

Additionally, to investigate factors that may influence the effectiveness of text mining algorithms, we explore the following sub-research questions:

- *RQ*1.1 *What is the impact of text tokenization on the performance of text mining algorithms for dependency detection?*
- *RQ*1.2 *How does the length of the text influence the effectiveness of text mining algorithms in detecting dependencies?*

The sub-research questions will be addressed in this paper by conducting an industrial case study. The motivation for studying dependency detection arises from industrial practices, where test engineers commonly design and write test cases. Functionally dependent test cases are frequently created to test either the same function or different aspects of the same function. As a result, these test cases often share semantic similarities or exhibit similar specifications, making the identification of dependencies essential. Neglecting test case dependencies can lead to redundant failures, increased manual effort, and inefficiencies (Tahvili et al., 2016b).

Early detection helps reduce redundancy and streamline the testing process (Ansari et al., 2016). In industrial settings, clustering test cases enables efficient selection, prioritization, and scheduling (Tahvili, 2018), supports test automation, and aids in test suite reduction by identifying and removing redundant cases (Tahvili & Hatvani, 2022; Felderer et al., 2023). This study applies various text analysis algorithms-categorized as distance-based, compression-based, and neural network-based-in an industrial case study at Alstom in Sweden. The goal is to find the optimal trade-off between complexity, accuracy, and effort to reduce time, cost, and resources while maintaining high testing quality.

2 Background & related work

Understanding the similarities and dependencies between test cases is critical for improving test execution efficiency, reducing costs, and optimizing resources (Arlt et al., 2015; Feldt et al., 2016b; Tahvili et al., 2016b, 2019b, 2016a; Landin et al., 2020a). These insights enable more effective test case selection, prioritization, scheduling, and parallel execution. In industrial settings, large test suites often include functionally dependent test cases, designed to test the same function or its different aspects. Clustering and classifying such test cases based on their dependencies can mitigate redundant test failures and improve testing processes (Landin et al., 2020b). This study aims to advance test optimization by clustering test cases based on dependencies, improving upon earlier approaches (Tahvili et al., 2019a), utilizing ground truth data for supervised learning, and applying novel text analysis methodologies. To evaluate and compare text analysis techniques with varying complexity levels, the study employs distance-based, compression-based, and neural network-based methods. Simpler algorithms use distance matrices for similarity detection, while more complex methods involve lexical-semantic analysis or neural network models. Finding the right balance between complexity, accuracy, and effort is challenging, particularly for integration test cases with substantial interaction between software modules. The algorithms convert text specifications into distance metrics or vectors, which are subsequently used for clustering. Visualization of test case similarities helps identify redundant tests, streamline test repositories, and inform decisions to optimize cost, time, and resources (Neto et al., 2018; Tahvili et al., 2019a). Natural language processing has become a cornerstone in software testing, addressing challenges across multiple dimensions (Garousi et al., 2020; Roy et al., 2014; Chen et al., 2016). For example, Fischbach et al. (2020) utilized dependency parsing to generate test cases from acceptance criteria, while Tahvili et al. (2020) applied latent semantic analysis to classify integration test cases as dependent or independent. Beniwal et al. (2021) explored opinion mining for user acceptance testing, and Sutar et al. (2020) selected regression test cases based on semantic similarity measures. Similarly, Malik et al. (2020) automated test oracle generation using regex patterns and string distance functions, while Lin et al. (2019) transferred test cases between apps based on Word2Vec similarity. Greiler et al. (2012) proposed mining test connections between high- and low-level tests using shared word counts. Other works explored test case prioritization using topic models (Thomas et al., 2014), diversity measurement (Shi et al., 2016; Feldt et al., 2016b), and clustering unit test cases using normalized compression distance (Feldt et al., 2008a).

Table 1 categorizes the input, characteristics, and output of the text analysis algorithms used in this study. These algorithms generate either distance metrics or vectors, often requiring

Table 1 Categories and key characteristics of text analysis algorithms

Algorithm Category Input Output	Main Characteristics
Distance-based Tokenized/Non-tokenized text Distance matrix	Provides deterministic predictions based on token or q-gram counting, making results explainable and interpretable. Typically very fast in execution.
Compression-based Tokenized/Non-tokenized text Distance matrix	Produces deterministic predictions but is harder to interpret as similarities are not based on simple token counts. Typically fast but less explainable to humans.
Neural network-based Non-tokenized text High-dimensional vectors	Stochastic due to random seeds, meaning performance depends on training data and hyperparameter selection. Models are complex and opaque, reducing predictability and explainability.

post-processing to derive metrics for clustering. Additionally, the impact of test specification size on algorithm performance is analyzed by segmenting longer text strings into smaller fragments, enabling sensitivity assessment. By advancing clustering and dependency detection techniques, this study contributes actionable insights for optimizing software testing and balancing the trade-offs between complexity, accuracy, and resource allocation in industrial contexts.

An example of a manual test case is provided in Table 2, where any type of test specification from the table can serve as input to the proposed solution. This table illustrates the raw data that forms the foundation for our approach, highlighting the diverse formats and structures of test specifications. These specifications, whether they are detailed procedural steps, high-level requirements, or descriptive scenarios, are processed and analyzed by the proposed solution to detect functional dependencies and optimize clustering. By leveraging this input data, the solution demonstrates its applicability across various types of test specifications, ensuring its relevance to industrial testing practices.

3 The proposed solution

This section introduces a framework for categorizing manual integration test specifications into clusters, primarily aimed at supporting various test optimization objectives. However, this study specifically examines dependency detection on manual integration test cases. Rather than presenting the methodology itself as a standalone contribution, we use it as a structured process to apply and evaluate various text mining and clustering techniques for dependency detection. The core contributions of this work lie in the defined research questions and their empirical investigation. The proposed approach consists of two mandatory steps and three optional steps, as shown in Fig. 1, where the optional steps are represented by dashed boxes. The input comprises manual test cases written in natural text, typically derived from requirements specifications by testers and test engineers. The specifics of each step are outlined as follows:

Table 2 Example of a test case specification designed at Alstom

Attribute	**Details**
Test case name:	Auxiliary Compressor Control
Date:	2020-03-20
Test case ID:	3EST001845-2032 - RCM (v.1)
Test level(s):	Sw/Hw Integration
Comments:	None
Test Configuration	**Details**
TCMS baseline	TCMS 1.2.3.0
Test rig	VCS Release 1.16.5
VCS Platform	VCS Platform 3.24.0
Requirements	**Details**
Requirement 1	SRS-BHH-Line Voltage 1707
Requirement 2	SRS-BHH-Speed 2051
Tester ID	BR-1211
Initial State	**Details**
Initial State	No active cab
Test Steps	**Action and Expected Reaction**
Step 1	Lock and set Auxiliary reservoir pressure < 5.5 bar.
	Reaction: Signal Command auxiliary compressor.
Step 2	Activate cab A2 lock and set signal braking mode from ATP to 109.
	Reaction: Signal braking mode to IDU is set to 109.
Step 3	Lock and set Auxiliary reservoir pressure > 5.5 bar.
	Reaction: Signal Auxiliary compressor is running to IDU is set to FALSE.
Step 4	Wait 20 seconds.
	Reaction: None.
Step 5	Reset dynamic brake in the train for 5 seconds.
	Reaction: IDU in $B1$ car is set to On.
Step 6	Set Auxiliary reservoir pressure < 5.5 bar.
	Reaction: Signal Auxiliary compressor is running to IDU is set to FALSE.
Step 7	Clean up.
	Reaction: None.

Input: the input to the proposed solution in Fig. 1 is a set of manual test cases written using structured natural language templates, an example of which is presented in Table 2.

Step 1: Data Preprocessing: this step, while optional, can significantly enhance the performance of the proposed solution by removing redundant or irrelevant information from the test specifications. Whether this step is necessary depends on the quality of the input data. For example, if test cases contain inconsistent formatting, excessive noise, or unnecessary details, data preprocessing ensures cleaner textual input. However, if the test specifications are already in a structured format, this step may be skipped to preserve original textual char-

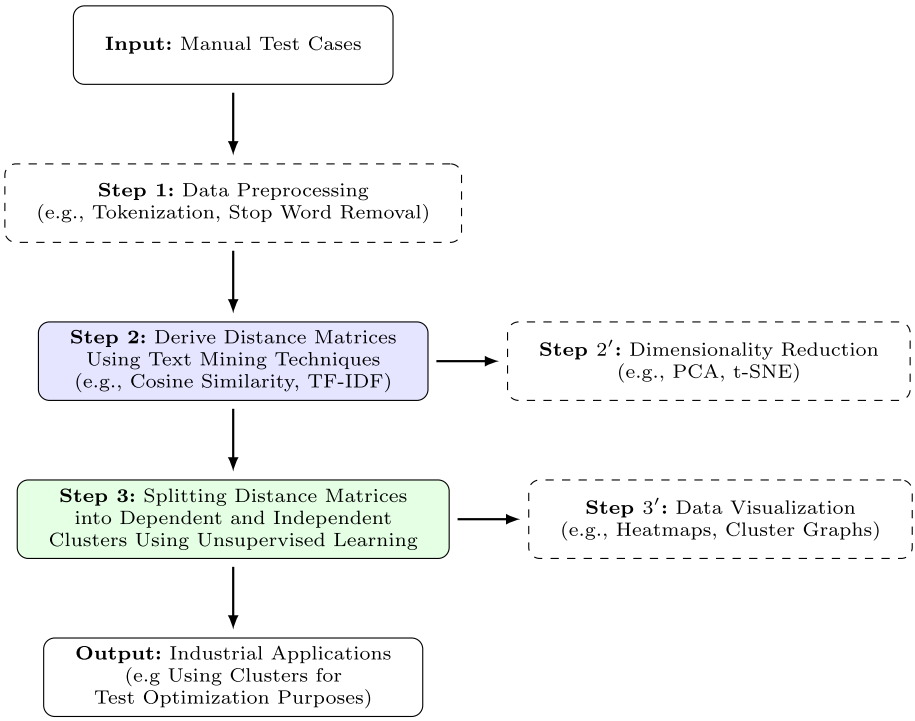

Fig. 1 Overview of the input, process, and output of the proposed solution

acteristics. The decision to apply preprocessing is data-driven, ensuring minimal impact on the overall approach (Table 3).

Step 2: Derive Distance Matrices Using Text Mining Techniques: manual test cases, being in textual form, can benefit from various text mining methods for dependency detection. In this study, the methods presented in Tables 4, 5, and 6 have been utilized. To ensure general applicability, a range of text mining techniques has been considered, ranging from traditional edit distances to machine learning-based embedding techniques. This selection allows for a systematic evaluation of how different approaches impact dependency detection.

Step 2′: Dimensionality Reduction: depending on the chosen text mining method in Step 2, we may obtain a large set of high-dimensional data points. Utilizing dimensionality reduction techniques like principal component analysis (PCA) or t-distributed stochastic neighbor embedding (t-SNE) can enhance the efficiency of handling these vectors. However, for other text mining techniques suggested (e.g., edit distance), this step may not be necessary.

Step 3: Splitting Distance Matrices into Clusters: in this step, all test cases are grouped into clusters, which are then categorized as either dependent or independent using the clustering algorithms listed in Table 7. Rather than applying all clustering algorithms to all distance metrics in a brute-force manner, this step adopts a systematic evaluation to identify the most effective combinations for dependency detection. Each clustering approach is carefully selected based on its compatibility with the specific text representation, ensuring meaningful and interpretable results. The effectiveness of these combinations is demonstrated through experimental validation (see Sections 8.3 and 8.4).

Step 3′: Data Visualization: this step aids in presenting an easily understandable overview of the clustered test cases, highlighting valuable information, trends, and outliers for testers. Early visualization of results is crucial for optimizing the testing process. We propose employing data visualization techniques compatible with the suggested clustering algorithms. However, this step is also optional.

Output: Industrial Applications: the methodology illustrated in Fig. 1 serves as a structured framework for applying text mining techniques to support dependency detection. However, this work does not claim the methodology itself as a core contribution. Instead, the main contribution lies in the experimental evaluation of different text mining and clustering techniques for their effectiveness in identifying test case dependencies. While the methodology has been applied here specifically for detecting dependencies among manual integration test cases, it can be extended to support a range of test optimization objectives:

i. Test case selection, prioritization, and scheduling: selecting clusters of test cases for execution and ranking them according to their dependencies and similarities.
ii. Test suite minimization: removing redundant test cases from each cluster by analyzing their relationships with other cases, thereby streamlining the test suite.
iii. Parallel test execution: scheduling test cases for parallel execution based on their clustering. For instance, clusters of similar or dependent test cases can be executed sequentially in one test station, while other clusters are executed in parallel on different stations (Landin et al., 2020b; Tahvili, 2018; Tahvili et al., 2018b).
iv. Test data visualization: providing insights into test suite structure and dependencies through visual representations of clusters, aiding in identifying inefficiencies or gaps in test coverage (Tahvili & Hatvani, 2022).
v. Customizable test workflows: enabling teams to design tailored workflows based on clustered dependencies, such as targeted retesting or automated fault isolation (Makki et al., 2018).

4 Text mining techniques for information extraction

As stated earlier, the primary objective of this study is to evaluate the effectiveness of three categories of text mining algorithms *distance-based*, *compression-based*, and *neural network-based* in detecting dependencies among manual integration test cases within safety-critical systems. Using a ground truth of functional dependencies (Tahvili et al., 2018a, 2019b), an industrial case study at Alstom evaluates selected algorithms from each category. These algorithms process key textual information from test artifacts, including input, output, test sequences, and execution logs, to effectively identify and cluster dependent test cases.

4.1 Data preprocessing

Data preprocessing is a critical step in data mining and significantly enhances the performance of supervised machine learning algorithms. Industrial-scale datasets often contain challenges such as missing values, out-of-range data, and inconsistencies, which, if not properly handled, can lead to inaccurate or unreliable results. Common preprocessing techniques include data cleaning, integration, transformation, and reduction (Kotsiantis et al., 2006). The selection of appropriate techniques depends on factors such as data quality, size, format, and specific application requirements. In software testing, manual test specifications are often stored in diverse formats, including rich text documents and spreadsheets. These formats may introduce

Table 3 The tokenized version of the test case presented in Table 2.

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 seconds reset dynamic brake in the train for 5 seconds 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

issues such as misspellings, ambiguities, and inconsistencies, requiring careful preprocessing to ensure accuracy and consistency in subsequent analyses. One essential preprocessing step is **tokenization**, which plays a fundamental role in structuring textual data for analysis.

Definition 1 Tokens are meaningful segments of a text, which can be individual words or larger units like word sequences, paragraphs, sentences, or lines, used for text analysis (Manning et al., 2008).

Tokenization involves splitting text into meaningful units called tokens, often including steps like removing punctuation and converting text to lowercase (Welbers et al., 2017). It plays a crucial role in modern natural language processing, enabling models to analyze word sequences, understand context, and interpret textual meaning (Mohan, 2015). However, the impact of tokenization on text clustering and similarity computations varies depending on the dataset and methodology used. While Fig. 2 provides an illustrative example, further analysis across multiple datasets is necessary to fully understand its broader implications.

As shown in Fig. 2, the original and tokenized text share 67.9% of their content, with 32.1% comprising minor changes in capitalization and word spacing. However, this example is not meant to generalize tokenization effects across all cases. The influence of tokenization on clustering and similarity measures depends on various factors, including text structure,

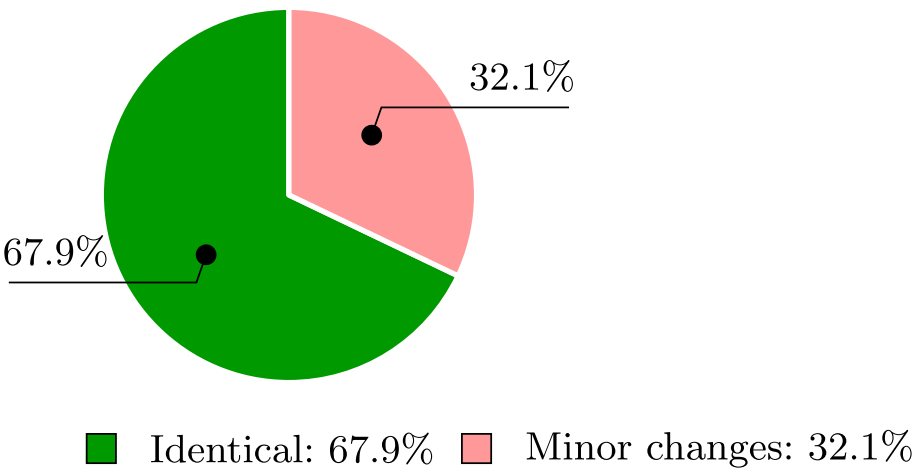

Fig. 2 Comparison of the original and tokenized versions of the test case from Tables 2 and 3, using the Levenshtein distance

preprocessing choices, and algorithmic sensitivity. Future work should systematically examine these aspects using diverse datasets to derive more generalizable conclusions.

4.2 String distances

This study explores various distance functions commonly used for approximate string matching, such as Levenshtein, Jaccard, and Hamming. These functions assess the similarity between text pairs within test artifacts. Different research efforts (de Oliveira Neto et al., 2018, 2016; Feldt et al., 2008b; Noor & Hemmati, 2015; Ledru et al., 2012) analyze the trade-offs among these functions in tasks like test selection prioritization and test suite minimization. The choice of the most suitable function depends on the specific context. For example, functions such as Levenshtein often excel in detecting subtle faults due to their

Table 4 Details of the string distance algorithms used in this study

Algorithm	Category & Definition	Characteristics
Cosine	Token-based. Measures the cosine of the angle between two non-zero feature vectors.	Commonly used for document similarity based on subject matter (Singhal, 2001). Less effective in low-dimensional spaces (Ye, 2015).
Jaccard	Token-based. Computes similarity based on the intersection between two sets of q-grams.	Efficient for large texts when combined with other techniques (Miranda et al., 2018). Sensitivity varies with the choice of q (Navarro et al., 2005).
Jaro	Edit-based. Measures similarity based on common characters and their transpositions, using a prefix scale p (here, $p = 0$).	Well-suited for shorter strings and names (Cohen et al., 2003). Enhances string matching when combined with TF-IDF (Cohen et al., 2003).
Levenshtein	Edit-based. Computes the minimum number of insertions, deletions, or substitutions to transform one string into another.	Effective for detecting similarities in text artifacts (de Oliveira Neto et al., 2018). Computationally expensive for long strings.
Overlap Coefficient	String-matching. Measures the overlap between two sets, similar to Jaccard similarity.	Commonly used in distribution comparisons; has favorable mathematical properties. Less frequent in string distance applications.
q-gram	Token-based. Computes similarity by summing the absolute differences between substrings of length q.	Faster than edit distance methods; runs in linear time (Navarro et al., 2005). Highly sensitive to the choice of q (Sidorov et al., 2013).
Ratcliff-Obershelp	String-matching. Measures similarity based on matching character sequences relative to total string length (Black, 2004).	Provides a confidence percentage to indicate similarity.
Sorensen-Dice Coefficient	String-matching. Measures spatial overlap between segmentations A and B (Prescott et al., 2009).	Sensitive in heterogeneous datasets; reduces the impact of outliers (Zou et al., 2004). Counts true positives only once in calculations (Dalirsefat et al., 2009).

sensitivity to small changes in string patterns (de Oliveira Neto et al., 2018), whereas Jaccard or token-based measures may be more effective in reducing the number of redundant tests by focusing on structural or semantic overlaps between test artifacts (Noor & Hemmati, 2015). While fault detection benefits from functions sensitive to small textual variations, test reduction relies more on measures that emphasize broader similarities or dissimilarities, such as token overlap. Understanding the advantages and limitations of each function is crucial for practitioners to make informed decisions that align with specific testing goals. To this end, string distance measures can be categorized into two groups: edit distance, which quantifies minimal operations to transform one string into another, and token-based distance, which counts common tokens between strings. Edit distance measures assess dissimilarity based on the minimum number of operations required for transformation (Ledru et al., 2012), making them suitable for detecting subtle variations. In contrast, token-based distances focus on structural similarity, often prioritizing efficiency in clustering and test minimization tasks.

The Levenshtein distance, commonly known as edit distance, finds extensive application in spell checkers, DNA sequencing, and comparing test inputs and sequences. Other variants of edit distance, like Jaro distance, prioritize matching strings from the beginning. Token-based distances, such as the Jaccard index and cosine distance, quantify the similarity based on similar substrings of length q between two strings. These metrics are utilized in diversity-based testing to identify diverse test subsets revealing various faults. However, token-based techniques can be computationally intensive due to the creation of q-grams and pairwise similarity comparisons. Alternative approaches like Shingling and Minhashing have been proposed to enhance efficiency by comparing entire test suites. Table 4 provides a summary, including examples and distinctions between each string distance measure. It's important to note that most string distance measures are limited to lexicographical similarity and may not capture semantic similarity unless complemented with NLP techniques. Additionally, these measures are suitable only for textual data, whereas numerical data may require other distance measures such as Euclidean or Manhattan distance.

4.3 Normalized Compression Distance (NCD)

NCD approximates the information distance based on the lengths of compressed entities x and y. This approach evaluates the similarity between entities by compressing x and y separately using a compression algorithm C and comparing their lengths to the compression of their concatenation (Cilibrasi & Vitányi, 2005).

$$NCD(x, y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \tag{1}$$

Moreover, NCD finds application across various domains, including software testing, where recent evidence suggests its effectiveness in black-box test prioritization (Miranda et al., 2018). However, its computational cost can render it impractical for large test suites. To address this, Feldt et al. (2016a) propose adapting NCD for multisets instead of pairwise comparison. Additionally, the choice of compressor C can significantly influence distance values, impacting parameters like throughput, compression, decompression speed, and memory usage. Different compressors have been compared across diverse domains, such as image processing and genome data sequencing. Table 5 provides a summary of various compres-

Table 5 Details of the lossless compressors C used in this study

Compressor	Definition	Characteristics
bzip2	Converts frequent sequences of characters into repeated letter strings.	Achieves high compression ratios but has slow performance due to multiple stacked compression techniques.
DEFLATE	A Lempel-Ziv algorithm combined with Huffman coding (Deutsch, 1996), compressing data using encoded literals and matching strings found in the preceding uncompressed data.	Effectively finds and encodes text redundancy. Available in many programming languages but has slower performance compared to the Lempel-Ziv family (e.g., LZ4).
gzip	Based on the DEFLATE algorithm, which combines LZ77 and Huffman coding.	Requires significant CPU resources for compression, making it a time-consuming process.
XZ	A Lempel-Ziv algorithm using a Markov chain-based prediction model (LZMA) to estimate probabilities for each bit.	Achieves higher compression ratios than gzip and bzip2 but has slower compression speeds.
Zlib	An abstraction of the DEFLATE algorithm used in gzip file compression.	Can compress and decompress data of any length. Requires parallel processing or CPU-level optimization for efficiency.
Zstd	Provides compression ratios comparable to DEFLATE.	Uses long-range search and deduplication similar to zip/gzip. Less flexible for small datasets.

sors used with NCD, highlighting their trade-offs. Compression rates and speeds are based on large text benchmarks as reported by Mahoney (2023), and the evaluation in this work builds upon these benchmarks with additional comparative analysis. Additionally, we assess NCD's effectiveness compared to other textual similarity measures, incorporating insights from both prior studies and our independent evaluations.

4.4 Machine learning approaches

To overcome the limitations of basic text similarity approaches, which struggle to capture word context (Wang & Kuo, 2020) and semantic meaning (Ledru et al., 2012), advanced techniques such as Doc2Vec and BERT have been developed (Devlin et al., 2019; Le & Mikolov, 2014). These methods transform textual content into numerical vector representations, enabling a more effective analysis of semantic and contextual relationships within the text. Doc2Vec, also known as Paragraph Vector, generates vector representations for entire documents by encoding both paragraph-level context and word-level information. This allows for a higher-level analysis of textual semantics, making it effective for detecting dependencies and similarities in textual data (Le & Mikolov, 2014). Although Doc2Vec is computationally demanding, it remains a practical choice for clustering test artifacts when appropriately tuned. BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019) is a deep neural network designed for language modeling and understanding complex semantics. By pre-training on large datasets and fine-tuning for specific tasks, BERT captures nuanced relationships between words and sentences. Due to its high computational cost, we use SBERT (Sentence BERT), an optimized variant that maintains high accuracy while

Table 6 Details of the Machine Learning approaches used for textual analysis

Algorithm	Description	Characteristics
Doc2Vec	Represents words and corresponding paragraphs as feature vectors to capture semantic relationships.	Provides a good balance between capturing sentence semantics and computational efficiency (Le & Mikolov, 2014). Performs well for test artifacts (Tahvili et al., 2019a) but is less efficient than simpler approaches (e.g., string distances) and requires parameter tuning (Le & Mikolov, 2014).
SBERT	A deep neural network trained for language inference and understanding tasks, capturing relationships at both word and sentence levels.	Achieves state-of-the-art performance on multiple benchmarks (Devlin et al., 2019). However, SBERT requires significantly more computational time than simpler approaches, limiting its efficiency for clustering tasks (Devlin et al., 2019; Reimers & Gurevych, 2019).

significantly reducing computational requirements (Reimers & Gurevych, 2019). This study evaluates the effectiveness of these advanced techniques in clustering test artifacts and detecting functional dependencies, comparing them with simpler methods such as string distances and generic approaches like NCD. The comparison highlights trade-offs between computational efficiency, semantic richness, and applicability to large-scale industrial datasets. Table 6 summarizes Doc2Vec and SBERT, outlining their differences in computational efficiency, semantic depth, and practical usability. The evaluation includes a comparative analysis of their performance in clustering and dependency detection tasks, providing insights into their suitability for various textual similarity measures.

5 Clustering analysis

This section elaborates on the clustering algorithms utilized in the study, along with the method applied for data visualization. Clustering algorithms are commonly employed to identify groups of similar objects in multivariate datasets (Xu & Wunsch, 2005). These algorithms often rely on similarity (or dissimilarity) measures between data points (Xu & Tian, 2015). While similarity is appropriate for qualitative data features, dissimilarity (distance) is preferred for quantitative data features to identify relationships between data points (Xu & Tian, 2015; Xu & Wunsch, 2005). The clustering algorithms used in this study are summarized in Table 7.

Graph-based: This algorithm does not require prior assumptions about the number, size, density, or shape of clusters, making it adaptable to various data structures (Blondel et al., 2008).

Density-based: This method defines clusters as contiguous regions of high point density, separated by areas of lower density (Sander, 2010).

Hierarchical-based: This approach constructs a cluster tree, where each group is linked to two or more subgroups, allowing for multi-level cluster representation (Cohen-addad et al., 2019).

Partition-based: This method divides a dataset into disjoint clusters, ensuring that each data point belongs to exactly one cluster (Mann & Chawla, 2020).

Table 7 Details of the clustering algorithms used in this study

Algorithm	Category & Description	Characteristics
Affinity	Partition-based clustering algorithm that identifies "exemplars" as representative members of clusters based on similarity metrics (Frey & Dueck, 2007).	Efficient for small datasets and directly provides exemplars as cluster centers. However, it can converge to suboptimal solutions, especially for large similarity matrices (Brusco et al., 2017).
Agglomerative	Hierarchical clustering technique where each observation starts in its own cluster, and pairs of clusters are iteratively merged up the hierarchy (Frigui & Krishnapuram, 1997).	Does not require a predefined number of clusters. However, it has high time complexity, making it less efficient for large datasets (Visalakshi et al., 2016).
DBSCAN	Density-based clustering method that groups closely packed points while marking low-density regions as outliers (Ester et al., 1996).	Effectively handles outliers and does not require specifying the number of clusters. However, it struggles with datasets of varying densities and high-dimensional data (Dang, 2015).
HDBSCAN	An extension of DBSCAN that extracts flat clustering based on cluster stability across multiple density levels (McInnes & Healy, 2017).	Suitable for high-dimensional data and datasets with varying densities. However, it is sensitive to hyperparameter choices and may not perform well with overlapping clusters (Guan et al., 2006).
Spectral	Graph-based clustering algorithm that utilizes the spectrum of a similarity matrix for dimensionality reduction before clustering (Kempe & McSherry, 2004).	Does not assume specific cluster shapes. However, its performance depends on the choice of initial centroids and similarity measures (Kannan et al., 2004).

These algorithms represent a diverse range of clustering techniques, each offering distinct advantages and trade-offs for grouping test cases based on textual similarity. Given the variety of clustering methods available, we selected those frequently used in NLP research and relevant to the objectives of this study.

6 Data visualization & dimensionality reduction

A large set of measured variables can significantly increase the computational load for data processing (Kaski & Peltonen, 2011). Additionally, data visualization is a crucial component of exploratory data analysis, often requiring dimensionality reduction to effectively interpret patterns (Kaski & Peltonen, 2011). Dimensionality reduction involves decreasing the number

Table 8 Details of dimensionality reduction and visualization techniques

Algorithm	Description	Characteristics
UMAP	Utilizes an exponential probability distribution in high-dimensional space rather than strictly relying on Euclidean distances.	Does not normalize probabilities in either high or low-dimensional spaces. Susceptible to small data changes and requires careful parameter tuning (Espadoto et al., 2020).
MDS	Reduces stress between optimally scaled data and distances by identifying a new configuration of points.	Uses fewer data points to represent the entire dataset. However, it introduces subjectivity, as modeling tabular data into a multidimensional scale requires decision-making (Bergener et al., 1976).

of input variables (the feature set dimension) in a dataset (Blum et al., 2013). Traditional techniques, such as eliminating low-variance columns and highly correlated features, help remove redundant information and improve computational efficiency (van der Maaten et al., 2008). Table 8 summarizes the advantages and limitations of different dimensionality reduction and data visualization techniques.

In this study, dimensionality reduction plays a key role in analyzing the effects of text tokenization (RQ1.1) and text length (RQ1.2) on text mining algorithms. Visualizing clustering structures helps us better understand how different textual representations influence dependency detection. Specifically, these techniques allow us to observe how various clustering algorithms handle tokenized versus non-tokenized text and how text length variations impact the organization of test cases. This enhances the validation of our research questions by providing an additional perspective on the clustering results. Given the large industrial dataset analyzed in this study, we employed two well-established techniques capable of both dimensionality reduction and data visualization: Manifold Approximation and Projection (UMAP) and Multidimensional Scaling (MDS). While the results from UMAP are presented later in this study, the MDS results can be found in the appendix[1].

7 Empirical evaluation

To assess the trade-off between performance and effort in text mining and clustering techniques, an industrial case study was conducted at Alstom in Sweden, following the guidelines of Runeson and Höst (2008). Alstom employs both manual and automated testing, with manual integration testing being the predominant approach. This study focused on integration test cases from the ongoing BR490 project, aiming to optimize testing processes by identifying functional dependencies. These dependencies were previously determined through an analysis of internal signal communications between software modules (Tahvili et al., 2018a). However, since such data is not always available, test cases often serve as the primary data source in testing projects. The text mining techniques described in Section 3 were applied to identify similarities and semantic relationships between test cases. Given that complex algorithms may require additional data and computational effort, selecting appropriate methods is crucial for optimization. The empirical evaluation specifically addresses RQ1.1 and RQ1.2, analyzing the impact of text tokenization and text length on the performance of text mining algorithms.

7.1 The ground truth

Functional dependencies between test cases are defined by the transfer of internal signals between software modules. If a signal flows from one module to another, the modules are considered dependent, and this dependency extends to their associated requirements and test cases. Figure 3 illustrates the traceability graph within the Train Control Management System (TCMS) platform, analyzed in the BR490 project. The dependency data in this study is extracted from system-level artifacts such as internal signal transfers, module interactions, and requirement traceability; however, such structured data is often unavailable in industrial settings and requires cross-department collaboration. To address this, we propose an alternative text analysis and NLP-based approach that leverages readily available test specifications.

[1] Additional figures in the appendix can be downloaded from https://github.com/leohatvani/Comparative-Analysis-Functional-Dependency

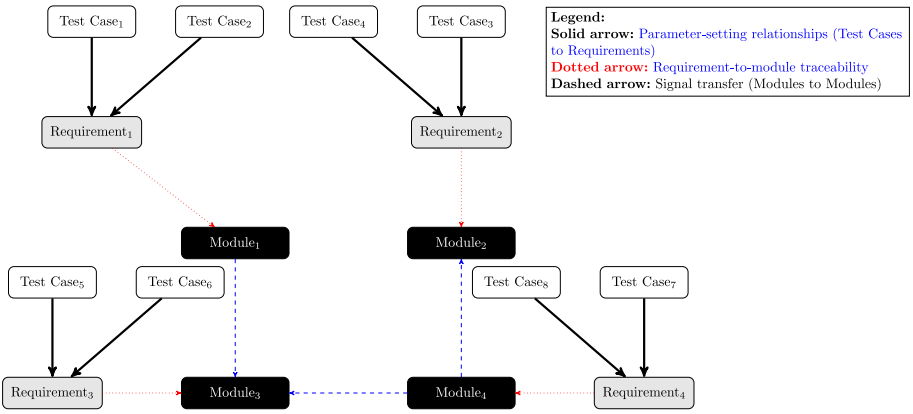

Fig. 3 Traceability and functional dependencies among modules, requirements, and test cases, with arrows indicating signal transfer, requirement-to-module traceability, and parameter-setting relationships

While companies with structured dependency data (as in our ground truth) may not need additional methods, our approach offers a practical solution for handling incomplete or missing information, helping test teams cluster test cases effectively and streamline prioritization and execution.

Dashed blue arrows in Fig. 3 represent signal transfers between modules, highlighting internal communications and dependencies. Dotted red arrows depict requirement-to-module traceability, representing the linkage between high-level requirements and software components that implement them. Solid black arrows indicate parameter-setting relationships, where a test case influences a requirement by setting its parameters or conditions.

It is important to clarify that dependencies between modules are **directional** (i.e., asymmetric). If Module_1 sends signals to Module_3, then Module_3 **depends on** Module_1 because it receives an internal signal transfer. However, this dependency is **one-way**, meaning that Module_1 is **not necessarily dependent** on Module_3 in return. Since Module_3 depends on Module_1, all requirements associated with Module_3 also **depend on** the requirements associated with Module_1. In Fig. 3, Requirement_3 depends on Requirement_1 because its implementation is influenced by signals originating from Module_1. Consequently, all test cases verifying Requirement_3 (i.e., Test Case_5 and Test Case_6) are dependent on those verifying Requirement_1 (i.e., Test Case_1 and Test Case_2). Using this information as a ground truth can help the testing team gain a better overview of the testing process. By identifying functional dependencies, testers can rank requirements for verification and prioritize test cases for execution. Dependencies significantly impact test execution, particularly when redundant or dependent test cases are executed sequentially. If a dependent test case fails, others relying on it may also fail (Tahvili et al., 2018a). Detecting these dependencies is crucial for optimizing execution order and reducing redundancy. Test cases are often structured as a directed graph, where independent cases should ideally precede dependent ones to enhance testing efficiency. The relationships in Fig. 3 are based on ground truth established in prior research (Tahvili et al., 2018a), offering a clear visualization of dependencies within the TCMS platform. This structured approach aids in effective test prioritization, ensuring that

critical requirements and test cases receive the necessary focus. Further details and figures related to the ground truth utilized in this study are available in the appendix.[2]

7.2 Unit of analysis and procedure

The units of analysis in this case study are 1, 748 manually designed integration test cases for a safety-critical train control subsystem at Alstom. The study follows these steps:

- The $BR490$ project is selected as the case study.
- A total of 1, 748 test specifications are extracted from Alstom's database.
- Each test case is stored in a separate comma-separated values (.CSV) file.
- Irrelevant details, such as tester names, dates, and timestamps, are removed.
- The .CSV files serve as input to the text mining techniques described in Section 3.
- Outputs from the text mining techniques are clustered and visualized using the algorithms and methods outlined in Sections 5 and 6.

7.3 Case study report

The initial results of the proposed approach in this study were obtained by following the steps outlined in Fig. 1. Test specifications designed for integration testing in the $BR490$ project at Alstom served as input for detecting dependencies between test cases. After preprocessing the data, applying text mining techniques, and performing clustering, a set of clusters containing test cases was generated. Notably, the number of clusters varied depending on the clustering algorithm employed.

Table 9 summarizes these results, highlighting that even when using the same clustering technique, variations in the number of clusters arise due to differences in the distance matrices used as input. The clustering outcomes vary significantly depending on the algorithm and text-mining technique employed.

Since ML-based approaches require numerical vector representations, tokenization was applied only to these techniques, while other approaches used standard text representations. This distinction is necessary because traditional similarity measures, such as Levenshtein distance and q-gram, operate directly on raw text and do not rely on tokenization. Given this methodological difference, Table 9 should be interpreted with caution: comparisons within the same category (ML-based or non-ML-based) are valid, while direct comparisons between the two categories may be influenced by preprocessing differences. For example, DBSCAN consistently produces a single cluster for all test cases except when using Levenshtein distance and q-gram. In contrast, HDBSCAN consistently generates a large number of clusters, regardless of the text mining technique. Agglomerative and Spectral clustering algorithms consistently produce five and eight clusters, respectively, across all text mining techniques. However, the number of clusters obtained using the Affinity algorithm varies. Unlike DBSCAN, Affinity groups all test cases into one cluster when applied with Levenshtein distance and q-gram. In summary, the minimum number of clusters is one, where all 1, 748 test cases are grouped into a single cluster, while the maximum number of clusters is 311, achieved by HDBSCAN when using XZ for text mining. These variations highlight the impact of text preprocessing choices on clustering results. While ML-based techniques generate embeddings that capture contextual meaning, traditional similarity measures rely solely on surface-level textual differences. Consequently, different distance metrics produce

[2] https://github.com/leohatvani/Comparative-Analysis-Functional-Dependency

Table 9 Number of clusters obtained using different distance metrics across various clustering algorithms in the case study

Clustering Algorithm	String Distance								Norm. Comp. Distance						ML	
	Cosine	Jaccard	Jaro	Levenshtein	Overlap	$q-$gram	Ratcliff	Sorensen	bzip2	Deflate	gzip	XZ	Zlib	Zstd	Doc2Vec	SBERT
Affinity	10	3	6	1	11	1	7	3	7	5	3	2	4	11	127	10
Agglomerative	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
DBSCAN	1	1	1	3	1	2	1	1	1	1	1	1	1	1	1	1
HDBSCAN	235	257	167	238	275	239	200	257	234	254	256	311	258	234	217	226
Spectral	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

distinct cluster structures. To enhance the understanding and analysis of these results, the UMAP model is employed for dimensionality reduction and visualization. The visual representations of selected results are provided in the Appendix (Hatvani & Tahvili, 2024).

8 Performance evaluation

This section evaluates string distance methods, NCD compressors, and machine learning approaches against the ground truth, addressing RQ1.1 on the impact of text tokenization on clustering performance. In this section, we assess the performance of the text mining and clustering techniques detailed in Table 9 against the established ground truth (GT) outlined in Fig. 3. The evaluation aims to compare the accuracy of detecting functionally dependent and independent test cases across different techniques. The number of resulting clusters varies depending on the combination of text mining technique and clustering algorithm. To measure performance, we employ the F1-Score, which considers both precision and recall. Precision denotes the ratio of correctly identified dependencies to the total detected dependencies, while recall represents the ratio of correctly identified dependencies to the total existing dependencies. The F1-Score provides a balance between precision and recall, considering false positives and false negatives, making it suitable for cases with uneven class distribution. This metric is preferable when the costs of false positives and false negatives differ significantly.

8.1 Experimental setup

Since the distribution of dependent and independent test cases is not equal in this problem (there are more dependent test cases compared to the independent test cases), the initial problem is an imbalance problem. The imbalance ratio (IR) is a well-known measure in the imbalanced domain (Tahvili et al., 2020). It is a ratio between the number of samples in the majority class and the number of samples in the minority class (see (2)):

$$IR = \frac{Majorityexamples}{Minorityexamples} \tag{2}$$

Some researchers consider that a dataset suffers from an imbalance problem if the IR is higher than 3. Employing (2), the imbalance ratio is equal to 4.33 for the $BR490$ project. Therefore, some performance metrics, such as accuracy, cannot be applied here. To address the imbalance problem, we applied random under-sampling by removing random entries from the majority class until we reached a one-to-one ratio between observed classes. This step ensures balanced classes for subsequent analysis. The experimental setup then proceeds to compare the performance of applied text mining techniques with different clustering algorithms. Tables 10, 11, 12, 13 and 14 respectively present the results, where the highest F1-Score values for each method are highlighted.

8.2 Parameters

Implementation of these approaches requires several different parameters to be set. To increase reproducibility, we are focused on employing the default parameters wherever possible. However, several choices were implemented and they are listed here. To apply the SBERT model, we have used the pre-trained model `bert-base-nli-mean-tokens`

(Reimers & Gurevych, 2019) and kept all of the other default settings at their default values. For the Doc2Vec implementation (Ilenic, 2017), we have used 2 noise words, vector dimensions of 100, batch size of 32, learning rate of 0.001, and 100 epochs. With these parameters, we trained our model on the same preprocessed dataset of manual test used throughout this study that then resulted in the document vectors. In our experiments, we observed that altering the default parameters rarely resulted in significantly improved results, whereas a significant deviation from these parameters often led to much worse outcomes. This observation highlights the sensitivity of parameter selection to the dataset being analyzed. While we relied on the default parameters provided by the MultiDistances.jl[3] package, we acknowledge that fine-tuning parameters tailored to specific datasets could enhance clustering performance. However, determining optimal parameters for generic use cases requires the analysis of a more diverse set of datasets. This remains an area for further exploration in future work. While a full replication package is not provided for this study, we have ensured that all essential elements required for experiment replication are open-source and publicly available. Specifically, the dataset used in this work can be accessed, and all software components utilized, such as the *MultiDistances.jl* package for Julia and the *scikit-learn* package for Python, can be freely obtained from their corresponding repositories. Additionally, all methodology steps, parameter settings, and implementation details are described to allow researchers to reproduce the experiments independently. Although variations in execution environments and parameter fine-tuning may lead to slightly different results, the overall approach remains fully replicable using the provided information.

8.3 Performance comparison of string distance methods against the ground truth

The combination of various string distance algorithms with different clustering methods results in the segmentation of test cases into multiple clusters. To ensure comparability with the ground truth, clusters are converted into dependencies between individual test cases.

While machine learning-based methods inherently require tokenized data due to their reliance on vectorized text representations, some string distance methods may also benefit from tokenization. Therefore, in this study, tokenization was selectively applied to certain string distance methods based on their mathematical formulation and sensitivity to raw text structure. In Table 10, we report the performance of string distance methods on tokenized data, ensuring their compatibility with ML-based approaches. In contrast, Table 11 presents results for string distance methods on non-tokenized data, maintaining a separate analysis to avoid mixing heterogeneous preprocessing strategies. This separation ensures that each method is evaluated under conditions most suitable for its intended usage. The highest F1-Score, as presented in Table 10 for tokenized data, is achieved with the Sorensen-Dice algorithm and the Affinity propagation clustering method (F1=0.633). Conversely, for non-tokenized data, presented in Table 11, the highest F1-Score is attained with the combination of Jaro distance and the Agglomerative clustering algorithm (F1=0.652). These observations suggest that tokenized data has a noticeable impact on some distance measures. Sorensen-Dice and Jaccard demonstrate strong performance with the Affinity clustering algorithm in the tokenized data setting, whereas Jaro distance excels for non-tokenized data, particularly when handling shorter strings. The lowest F1-Score is observed with the combination of cosine distance and Jaro distance with the spectral clustering algorithm, recording values of 0.240 and 0.379 for tokenized data and non-tokenized data, respectively. Sorensen-Dice demonstrates sensitivity in heterogeneous data, potentially contributing to its superior performance. However, both

[3] https://github.com/robertfeldt/MultiDistances.jl

Table 10 Results of string distance methods on tokenized data against the ground truth

Distance Measure	Clustering Algorithm	Precision	Recall	Accuracy	F1-Score
Cosine	Affinity	0.504	0.820	0.568	0.624
Cosine	Agglomerative	0.419	1	0.421	0.590
Cosine	DBSCAN	0.423	1	0.423	0.595
Cosine	HDBSCAN	0.643	0.576	0.680	0.608
Cosine	Spectral	0.424	0.167	0.542	0.240
Jaccard	Affinity	0.470	0.957	0.529	0.630
Jaccard	Agglomerative	0.427	0.996	0.433	0.597
Jaccard	DBSCAN	0.412	1	0.412	0.583
Jaccard	HDBSCAN	0.667	0.538	0.699	0.596
Jaccard	Spectral	0.385	0.760	0.395	0.511
Jaro	Affinity	0.527	0.480	0.601	0.502
Jaro	Agglomerative	0.500	0.787	0.585	0.611
Jaro	DBSCAN	0.417	1	0.417	0.588
Jaro	HDBSCAN	0.470	0.526	0.534	0.496
Jaro	Spectral	0.507	0.302	0.583	0.379
Levenshtein Distance	Affinity	0.422	1	0.422	0.593
Levenshtein Distance	Agglomerative	0.425	1	0.428	0.597
Levenshtein Distance	DBSCAN	0.414	1	0.421	0.586
Levenshtein Distance	HDBSCAN	0.488	0.489	0.576	0.489
Levenshtein Distance	Spectral	0.438	0.892	0.494	0.587
Overlap Coefficient	Affinity	0.503	0.462	0.578	0.482
Overlap Coefficient	Agglomerative	0.438	1	0.445	0.609
Overlap Coefficient	DBSCAN	0.430	1	0.430	0.601
Overlap Coefficient	HDBSCAN	0.562	0.393	0.618	0.462
Overlap Coefficient	Spectral	0.643	0.391	0.654	0.486
q-gram	Affinity	0.407	1	0.407	0.578
q-gram	Agglomerative	0.422	0.992	0.422	0.592
q-gram	DBSCAN	0.427	1	0.435	0.599
q-gram	HDBSCAN	0.503	0.473	0.575	0.488
q-gram	Spectral	0.446	0.218	0.561	0.293
Ratcliff-Obershelp	Affinity	0.486	0.456	0.561	0.470
Ratcliff-Obershelp	Agglomerative	0.437	0.998	0.442	0.608
Ratcliff-Obershelp	DBSCAN	0.433	1	0.433	0.604
Ratcliff-Obershelp	HDBSCAN	0.588	0.576	0.629	0.582
Ratcliff-Obershelp	Spectral	0.586	0.385	0.613	0.465
Sorensen-Dice Coefficient	Affinity	0.473	0.957	0.526	0.633
Sorensen-Dice Coefficient	Agglomerative	0.410	1	0.413	0.582
Sorensen-Dice Coefficient	DBSCAN	0.417	1	0.417	0.589
Sorensen-Dice Coefficient	HDBSCAN	0.677	0.538	0.696	0.600
Sorensen-Dice Coefficient	Spectral	0.395	0.716	0.410	0.509

Table 11 Results of string distance methods on non-tokenized data against the ground truth

Distance Measure	Clustering Algorithm	Precision	Recall	Accuracy	F1-Score
Cosine	Affinity	0.449	0.896	0.506	0.598
Cosine	Agglomerative	0.422	1	0.422	0.593
Cosine	DBSCAN	0.438	1	0.438	0.609
Cosine	HDBSCAN	0.647	0.578	0.686	0.610
Cosine	Spectral	0.361	0.480	0.409	0.412
Jaccard	Affinity	0.455	0.979	0.510	0.621
Jaccard	Agglomerative	0.417	1	0.419	0.588
Jaccard	DBSCAN	0.415	1	0.415	0.587
Jaccard	HDBSCAN	0.663	0.576	0.697	0.616
Jaccard	Spectral	0.372	0.797	0.373	0.508
Jaro	Affinity	0.627	0.557	0.677	0.590
Jaro	Agglomerative	0.5	0.938	0.579	0.652
Jaro	DBSCAN	0.405	1	0.405	0.577
Jaro	HDBSCAN	0.411	0.461	0.522	0.451
Jaro	Spectral	0.456	0.259	0.550	0.331
Levenshtein Distance	Affinity	0.411	1	0.411	0.583
Levenshtein Distance	Agglomerative	0.427	1	0.428	0.598
Levenshtein Distance	DBSCAN	0.436	1	0.444	0.607
Levenshtein Distance	HDBSCAN	0.515	0.534	0.587	0.524
Levenshtein Distance	Spectral	0.437	0.892	0.463	0.587
Overlap Coefficient	Affinity	0.555	0.584	0.625	0.569
Overlap Coefficient	Agglomerative	0.424	1	0.429	0.596
Overlap Coefficient	DBSCAN	0.423	1	0.423	0.595
Overlap Coefficient	HDBSCAN	0.532	0.480	0.601	0.505
Overlap Coefficient	Spectral	0.637	0.385	0.659	0.480
q-gram	Affinity	0.419	1	0.419	0.591
q-gram	Agglomerative	0.429	0.992	0.428	0.599
q-gram	DBSCAN	0.422	1	0.422	0.594
q-gram	HDBSCAN	0.55	0.557	0.615	0.553
q-gram	Spectral	0.425	0.920	0.438	0.581
Ratcliff-Obershelp	Affinity	0.442	0.407	0.528	0.424
Ratcliff-Obershelp	Agglomerative	0.415	1	0.423	0.586
Ratcliff-Obershelp	DBSCAN	0.411	1	0.411	0.583
Ratcliff-Obershelp	HDBSCAN	0.579	0.576	0.652	0.578
Ratcliff-Obershelp	Spectral	0.579	0.415	0.636	0.483
Sorensen-Dice Coefficient	Affinity	0.488	0.979	0.546	0.651
Sorensen-Dice Coefficient	Agglomerative	0.433	1	0.436	0.604
Sorensen-Dice Coefficient	DBSCAN	0.436	1	0.436	0.607
Sorensen-Dice Coefficient	HDBSCAN	0.671	0.576	0.699	0.620
Sorensen-Dice Coefficient	Spectral	0.398	0.789	0.404	0.529

Sorensen-Dice and Jaccard perform well with the Affinity clustering algorithm, suggesting a favorable combination. Additionally, Jaro distance exhibits high performance for non-tokenized data (0.65), particularly suitable for shorter strings.

8.4 Performance comparison of the NCD compressor methods against the ground truth

The results obtained by combining the presented NCD compressor with clustering algorithms are summarized in Tables 12 and 13.

HDBSCAN demonstrates superior performance compared to other clustering algorithms, achieving F1-Scores of 0.654 and 0.620 for the tokenized and non-tokenized versions of

Table 12 Results of NCD compressors on tokenized data against the ground truth

Compressor	Clustering Algorithm	Precision	Recall	Accuracy	F1-Score
bzip2	Affinity	0.533	0.467	0.607	0.498
bzip2	Agglomerative	0.422	0.973	0.436	0.588
bzip2	DBSCAN	0.430	1	0.430	0.602
bzip2	HDBSCAN	0.699	0.614	0.717	0.654
bzip2	Spectral	0.653	0.418	0.670	0.510
Deflate	Affinity	0.479	0.605	0.564	0.535
Deflate	Agglomerative	0.414	1	0.422	0.586
Deflate	DBSCAN	0.419	1	0.419	0.591
Deflate	HDBSCAN	0.652	0.572	0.682	0.609
Deflate	Spectral	0.619	0.296	0.614	0.400
Gzip	Affinity	0.436	0.760	0.489	0.554
Gzip	Agglomerative	0.413	0.974	0.441	0.597
Gzip	DBSCAN	0.414	1	0.414	0.585
Gzip	HDBSCAN	0.636	0.570	0.674	0.601
Gzip	Spectral	0.601	0.301	0.621	0.401
Xz	Affinity	0.398	0.822	0.400	0.536
Xz	Agglomerative	0.408	0.985	0.408	0.578
Xz	DBSCAN	0.434	1	0.434	0.605
Xz	HDBSCAN	0.678	0.527	0.700	0.593
Xz	Spectral	0.593	0.305	0.614	0.403
Zlib	Affinity	0.462	0.681	0.530	0.551
Zlib	Agglomerative	0.424	0.976	0.435	0.592
Zlib	DBSCAN	0.419	1	0.419	0.590
Zlib	HDBSCAN	0.626	0.572	0.673	0.598
Zlib	Spectral	0.600	0.297	0.620	0.398
Zstd	Affinity	0.475	0.538	0.548	0.505
Zstd	Agglomerative	0.439	1	0.441	0.610
Zstd	DBSCAN	0.406	1	0.406	0.577
Zstd	HDBSCAN	0.628	0.613	0.679	0.620
Zstd	Spectral	0.594	0.372	0.633	0.458

Table 13 Results of NCD compressors on non-tokenized data against ground truth

Compressor	Clustering Algorithm	Precision	Recall	Accuracy	F1-Score
bzip2	Affinity	0.486	0.405	0.573	0.442
bzip2	Agglomerative	0.428	0.923	0.431	0.585
bzip2	DBSCAN	0.415	1	0.415	0.586
bzip2	HDBSCAN	0.637	0.575	0.691	0.605
bzip2	Spectral	0.556	0.320	0.606	0.406
Deflate	Affinity	0.508	0.630	0.587	0.563
Deflate	Agglomerative	0.431	1	0.439	0.603
Deflate	DBSCAN	0.450	1	0.450	0.620
Deflate	HDBSCAN	0.650	0.578	0.686	0.612
Deflate	Spectral	0.538	0.223	0.581	0.315
Gzip	Affinity	0.452	0.717	0.522	0.555
Gzip	Agglomerative	0.405	1	0.409	0.576
Gzip	DBSCAN	0.410	1	0.410	0.582
Gzip	HDBSCAN	0.628	0.606	0.687	0.617
Gzip	Spectral	0.563	0.225	0.602	0.321
Xz	Affinity	0.375	0.778	0.378	0.506
Xz	Agglomerative	0.441	0.908	0.466	0.594
Xz	DBSCAN	0.424	1	0.424	0.596
Xz	HDBSCAN	0.665	0.519	0.693	0.583
Xz	Spectral	0.529	0.288	0.584	0.373
Zlib	Affinity	0.485	0.657	0.652	0.558
Zlib	Agglomerative	0.432	0.976	0.447	0.599
Zlib	DBSCAN	0.414	1	0.414	0.585
Zlib	HDBSCAN	0.659	0.614	0.708	0.636
Zlib	Spectral	0.552	0.232	0.606	0.327
Zstd	Affinity	0.522	0.386	0.604	0.444
Zstd	Agglomerative	0.420	1	0.422	0.592
Zstd	DBSCAN	0.437	1	0.437	0.608
Zstd	HDBSCAN	0.678	0.638	0.718	0.657
Zstd	Spectral	0.618	0.305	0.628	0.409

test cases, respectively, when using bzip2 and Zstd compressors. The term "superior performance" in this context refers to HDBSCAN's ability to form clusters that align closely with the functional dependencies and relationships in the test cases, as measured by F1-Scores. This capability is particularly evident in its handling of tokenized test cases, where it identifies clusters with a balance of precision and recall. It is important to note that smaller, highly specific clusters, which HDBSCAN can produce, may offer advantages in pinpointing subtle dependencies between test cases. However, managing a large number of small clusters can increase the complexity of test case prioritization and scheduling. Conversely, larger clusters may simplify handling but risk including dissimilar test cases, reducing cluster utility. For example, in scenarios with hundreds of elements in a cluster, the diversity within the cluster might complicate deriving actionable insights. Our findings suggest that HDBSCAN's hierarchical clustering approach strikes a balance, producing clusters that are neither excessively

granular nor overly broad. This balance enables efficient handling of test cases while preserving meaningful relationships. Furthermore, the analysis of Tables 12 and 13 suggests that high compression may not effectively detect similarities in short texts. For longer texts, the Xz compressor may be more suitable, while for the analyzed test cases, the bzip2 compressor and Gzip compression yield satisfactory results.

8.5 Performance comparison of the machine learning methods against the ground truth

As previously mentioned, the tokenization process is inherent in the machine learning algorithms used for text analysis, resulting in the presented results in Table 14 consisting solely of the tokenized version of the test cases.

Doc2Vec emerges as the best-performing algorithm, despite its relatively simple approach to semantics compared to SBERT. As described in Table 6, Doc2Vec performs well in capturing sentence-level semantics. However, Doc2Vec's exceptional performance is only observed when paired with Agglomerative clustering; for all other clustering algorithms, Doc2Vec exhibits a lower F1-Score compared to the top string distances and compressors. The strong performance of Doc2Vec with Agglomerative clustering could be attributed to the hierarchical nature of the algorithm. Agglomerative clustering progressively merges clusters based on similarity, which aligns well with the textual semantic representations generated by Doc2Vec. These representations are relatively simple yet effective for identifying hierarchical relationships, particularly in structured test data. In contrast, clustering algorithms like DBSCAN or HDBSCAN are density-based and may be less effective with Doc2Vec's output due to its tendency to produce evenly distributed embeddings rather than dense clusters. On the other hand, SBERT, which leverages transformer-based embeddings, performs consistently well across multiple clustering algorithms. Its ability to capture fine-grained semantic relationships makes it more adaptable to a variety of clustering techniques. However, its performance does not surpass that of Doc2Vec when paired with Agglomerative clustering, likely due to the hierarchical clustering algorithm's ability to exploit Doc2Vec's simpler embedding structure effectively. The implications of these findings are significant for test optimization. While Doc2Vec coupled with Agglomerative clustering provides an

Table 14 Results obtained using machine learning algorithms against the ground truth

NLP Algorithm	Clustering Algorithm	Precision	Recall	Accuracy	F1-Score
Doc2Vec	Affinity	0.853	0.101	0.610	0.181
Doc2Vec	Agglomerative	0.580	0.973	0.693	0.727
Doc2Vec	DBSCAN	0.413	1	0.413	0.585
Doc2Vec	HDBSCAN	0.574	0.586	0.646	0.580
Doc2Vec	Spectral	0.425	0.126	0.574	0.195
SBERT	Affinity	0.690	0.554	0.712	0.615
SBERT	Agglomerative	0.429	1	0.439	0.600
SBERT	DBSCAN	0.412	1	0.412	0.584
SBERT	HDBSCAN	0.564	0.548	0.621	0.556
SBERT	Spectral	0.339	0.088	0.548	0.140

efficient and effective solution for capturing hierarchical dependencies, SBERT's versatility across clustering algorithms makes it a robust choice for scenarios requiring flexibility in clustering approaches. This suggests that the choice of machine learning algorithm and clustering technique should consider the specific characteristics of the test data and the desired outcomes. Across Tables 10, 11, 12, 13 and 14, the F1-Score results are consistently close to each other, often varying by only 0.1 F1-Score. It's worth noting that to achieve an additional 0.1 F1-Score, models would need to predict an additional 100 true positives. Figure 4 illustrates the relationship between the obtained F1-Score and true positives.

8.6 Explicit answer to RQ1.1

The results in Tables 10, 11, 12, 13, and 14 show that text tokenization significantly enhances algorithm performance. For example, Jaccard with HDBSCAN achieves an F1-Score of 0.630 with tokenized data, compared to 0.510 without. Similarly, Zstd with HDBSCAN improves from 0.605 to 0.654, and Doc2Vec with Agglomerative clustering reaches 0.727 for tokenized data. Tokenization boosts recall, precision, and F1-Scores, underscoring its value as a preprocessing step for detecting functional dependencies.

9 Impact analysis

This section examines the impact of text length on the proposed solution's performance, addressing RQ1.2 by evaluating how text length variations influence clustering accuracy and dependency detection. The evaluation is conducted against the ground truth (GT) to assess the reliability of the results.

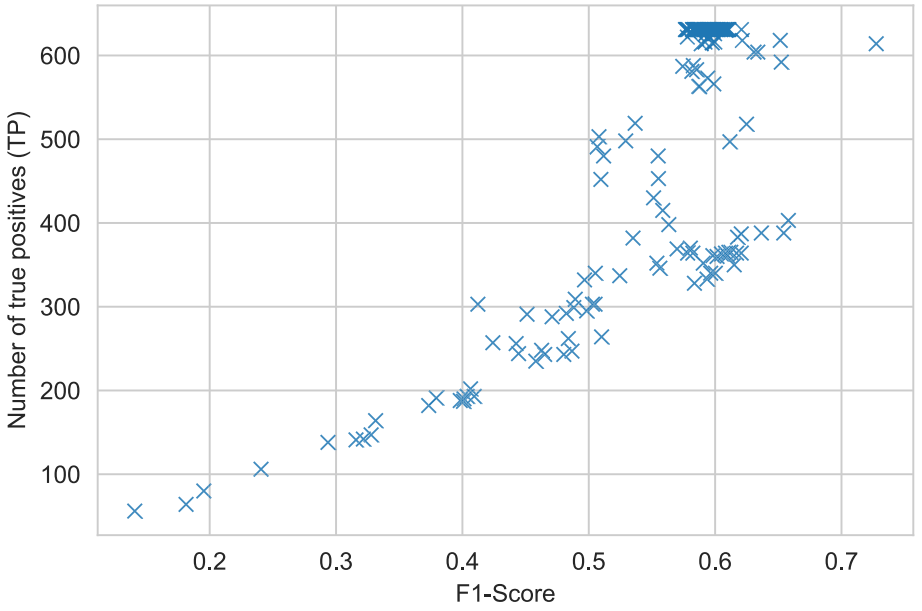

Fig. 4 A scatter plot to compare the F1-Score and the number of true positives

Table 15 Descriptive statistics of the utilized tokenized data

Statistic	Value
Mean	1605.36
Standard Error	39.11
Median	1168
Mode	332
Standard Deviation	1629.43
Sample Variance	2, 655, 048.38
Kurtosis	37.47
Skewness	4.02
Range	26, 618
Minimum	75
Maximum	26, 693
Count	1, 736
Confidence Level (95%)	76.70
Quartiles	
First Quartile ($Q1$)	[75, 607)
Second Quartile ($Q2$) (Median)	[607, 1168)
Third Quartile ($Q3$)	[1168, 1949.75)
Fourth Quartile ($Q4$)	[1949.75, 26, 693]

9.1 Impact of text length on performance

The impact of text length on algorithm performance was analyzed by dividing the dataset into quartiles based on descriptive statistics (Table 15), with text length considered in terms of document length, measured as the number of tokens.

The quartiles ranged from a minimum of 75 tokens in the first quartile to a maximum of 26, 693 tokens in the fourth quartile. This approach allowed for a systematic evaluation of the F1-Scores for both dependent and independent test cases. The second research question examines the relationship between text length and the performance of text mining and clus-

Table 16 Comparison of text length and performance results for dependent test cases

Tokenized	Method	1st quartile	F1-Score	2nd quartile	F1-Score
	String Distances	SoDi / Agglo	0.004	SoDi / Agglo	0.002
✔	String Distances	Jaccard / Agglo	0.002	Jaccard / HDBSCAN	0.002
	NCD Compression	XZ / HDBSCAN	0.003	XZ / HDBSCAN	0.003
✔	NCD Compression	XZ / HDBSCAN	0.003	XZ / HDBSCAN	0.003
✔	Machine Learning	Doc2Vec / Affinity	0.008	Doc2Vec / Agglo	0.025
Tokenized	Method	3rd quartile	F1-Score	4th quartile	F1-Score
	String Distances	Jaccard / HDBSCAN	0.005	RaOb / Agglo	0.038
✔	String Distances	Cosine / HDBSCAN	0.005	RaOb / Agglo	0.051
	NCD Compression	Bzip2 / Agglo	0.006	Bzip2 / Agglo	0.059
✔	NCD Compression	XZ / HDBSCAN	0.006	Gzip / Agglo	0.059
✔	Machine Learning	Doc2Vec / Agglo	0.034	Doc2Vec / Agglo	0.106

Table 17 Comparison of text length and performance results for independent test cases

Tokenized	Method	1st quartile	F1-Score	2nd quartile	F1-Score
	String Distances	Levensthein / DBSCAN	0.943	Levensthein / DBSCAN	0.852
✔	String Distances	Levensthein / DBSCAN	0.912	Levensthein / DBSCAN	0.856
	NCD Compression	Zstd / DBSCAN	0.831	Zstd / DBSCAN	0.834
✔	NCD Compression	Deflate / DBSCAN	0.817	Zstd / DBSCAN	0.854
✔	Machine Learning	Doc2Vec / DBSCAN	0.878	Doc2Vec / DBSCAN	0.827
Tokenized	Method	3rd quartile	F1-Score	4th quartile	F1-Score
	String Distances	Levensthein / DBSCAN	0.789	Levensthein / DBSCAN	0.682
✔	String Distances	Levensthein / DBSCAN	0.793	Levensthein / DBSCAN	0.676
	NCD Compression	Zstd / DBSCAN	0.751	Zlib / DBSCAN	0.694
✔	NCD Compression	Zstd / DBSCAN	0.756	Deflate / DBSCAN	0.694
✔	Machine Learning	Doc2Vec / DBSCAN	0.723	Doc2Vec / DBSCAN	0.675

tering algorithms. The data used in the study are summarized in Table 15, where test cases are divided into quartiles based on test specification length, ensuring that the analysis considers both tokenized and non-tokenized representations.

Table 16 presents the results for dependent test cases. The combination of the Doc2Vec algorithm with Affinity and Agglomerative clustering achieved the highest F1-Scores, particularly in the fourth quartile, indicating that longer texts provide richer semantic content, which enhances the detection of dependencies. This trend highlights the positive correlation between text length and performance for dependent test cases.

Table 17 shows that shorter text lengths (first and second quartiles) yield higher F1-Scores for independent test cases. Levenshtein distance with DBSCAN consistently performs best, highlighting the advantage of reduced noise in shorter texts for distinguishing independent cases. To improve readability and avoid redundancy, string distance approaches have been grouped under a single category labeled "String Distances" in Tables 16 and 17. Although minor variations exist between different string distance measures, these differences were found to be negligible in terms of their impact on the overall results. By merging them into a single category, we ensure that the key findings remain clear while maintaining a concise presentation of the data.

9.2 Explicit answer to RQ1.2

The results demonstrate that text length significantly influences the performance of text mining algorithms, with its impact differing between dependent and independent test cases. For dependent test cases, longer texts (4th quartile) consistently achieve higher F1-Scores, particularly for machine learning methods like Doc2Vec combined with Agglomerative clustering. The richer semantic content of longer texts provides more contextual information, improving the detection of dependencies. In contrast, shorter texts (1st and 2nd quartiles) perform better for independent test cases, yielding higher F1-Scores. For example, Levenshtein distance with DBSCAN consistently outperforms other methods for shorter texts, as reduced noise and complexity allow the algorithms to better distinguish between independent cases. These findings highlight the need to tailor algorithm selection based on text length. Longer texts should be leveraged for dependent cases to enhance dependency detection, while shorter

texts are better suited for independent cases to improve cluster separation and reduce redundancy in computation. This dual trend provides valuable guidance for optimizing algorithmic performance based on the characteristics of the textual data.

10 Threats to validity

The threats to the validity, limitations, and challenges faced in conducting the present study are discussed in this section. Given that our study is an experiment in a controlled setting and based on already collected data the internal threats to validity could more readily be controlled. We see few threats to our use of the proposed solution, or in collecting and interpreting their output. The main threat to our study is concerning the external validity. Since this is an industrial case study and uses data from a single company we cannot claim what results would be for other companies and projects. While our study would be even stronger if it also included multiple projects and, for example, open-source data could provide additional context we leave this for future work due to reasons of brevity. We note though that experiments on open-source data are more common, overall, in the community and that we also need industrial cases as a complement. Construct validity is focused on whether a study measures what it intended to measure (Tahvili et al., 2019b). The key construct validity threat in this study is concerning the ground truth: can we trust the functional dependencies provided by the engineers? In general, we can not since it and the test specifications themselves might suffer from ambiguity and inconsistencies. We thus acknowledge the threat but also highlight that this will always be the case even if a fully manual analysis of dependencies would be conducted. However, we tried to mitigate this threat by analyzing the traceability graph so we could augment the data derived from the clustering of test cases (such as the presented ground truth in this study). Conclusion validity is concerned with whether the treatments we used, here different text mining and clustering algorithms and tools, are related to and, ultimately, causing the outcomes that we identify and evaluate them on. In particular, if any claimed differences are statistically significant and can be trusted. We provide a more detailed analysis in the following to further study the sensitivity of our results. Furthermore, a sensitivity analysis using the Mantel[4] model is also provided in the appendix (Hatvani & Tahvili, 2024).

11 Discussion & future work

Knowing the dependencies between test cases at an early stage of a testing process can help testers and test managers schedule test cases for execution in a more efficient way. The obtained clusters in this study have been actively used at Alstom to address critical challenges in test scheduling and optimization. For example, the clusters were employed to prioritize test case execution, with independent clusters being executed first to minimize downstream failures caused by dependent test cases. This approach has helped the company streamline its test execution process, reducing delays caused by unanticipated failures in the dependent test cases. The clusters were also utilized to reduce redundancy in the test suite by identifying similar test cases within each cluster. In practice, testers at Alstom reviewed clustered test cases and eliminated those deemed redundant, which reduced the overall size of the test

[4] Mantel test is a non-parametric statistical method that computes the correlation between two distance matrices (Guillot & Rousset, 2011).

suite without compromising coverage. This action alone contributed to a significant reduction in test effort while maintaining quality. The relationship between similar and dependent test cases is central to the clustering process. Clusters of dependent test cases helped the company identify critical functional relationships between modules, ensuring that interdependencies were addressed early in the testing process. These clusters have been integrated into Alstom's test planning workflow to inform decisions on resource allocation and scheduling. For example, dependent test cases clustered together are now executed sequentially in the same test environment to avoid unnecessary context-switching and to ensure dependencies are resolved systematically. Furthermore, the actionable insights provided by the clustering approach have informed discussions across teams, fostering better communication between development and testing departments. By sharing cluster-based dependency and similarity insights, teams have been able to align their efforts more effectively, leading to improved collaboration and reduced ambiguity in test planning. These practical applications demonstrate the value of the proposed workflow beyond theoretical benefits. While the results are specific to the case study at Alstom, the workflow is designed to be adaptable to other industrial contexts with similar challenges. For example, the use of natural-language test specifications and clustering-based dependency detection can be generalized to production environments where formalized test cases are not readily available. The comparison of clustering methods and text mining techniques in this study provides actionable insights for practitioners. For instance, selecting the appropriate clustering algorithm (e.g., DBSCAN for independent test cases or HDBSCAN for non-clusterable cases) allows practitioners to tailor the approach to their specific needs. By integrating these findings into existing test processes, organizations can achieve tangible improvements in test efficiency and effectiveness. These examples demonstrate the practical value of the proposed approach and its applicability beyond the specific case study. Future work will involve additional case studies to explore these impacts further and integrate the approach into real-time test processes. This will help bridge the gap between post-mortem analysis and proactive dependency detection, ultimately improving the effectiveness and efficiency of testing in practice.

11.1 Guidelines for researchers and practitioners in software testing

Drawing on the empirical findings of this study, we propose a set of evidence-based guidelines aimed at supporting researchers and practitioners in effectively detecting dependencies and clustering test cases within software testing processes. These recommendations are derived from observed trends across various text mining and clustering techniques and are intended to inform the design and implementation of test optimization strategies in industrial settings:

(i) *Choice of Text Mining Technique:* For scenarios requiring deeper semantic analysis, advanced methods such as Doc2Vec or SBERT are recommended. In contrast, simpler approaches (e.g., string distance algorithms) may suffice for well-structured or shorter test specifications, offering a trade-off between computational cost and interpretability.
(ii) *Tokenization Strategy:* Tokenization should be carefully aligned with the structure of the test cases. For highly structured test descriptions, minimal tokenization is advised to retain key phrases, whereas unstructured inputs may benefit from comprehensive preprocessing to enhance clarity and consistency.
(iii) *Consideration of Text Length:* The effectiveness of dependency detection improves with longer test cases, as they tend to provide richer contextual information. Consequently, clustering algorithms should be calibrated to accommodate the increased complexity introduced by longer textual inputs.

(iv) *Clustering Algorithm Suitability:* Algorithms such as Agglomerative and HDBSCAN have shown strong performance in identifying functional dependencies among test cases. However, their effectiveness is contingent upon the nature of the text representation employed, necessitating careful pairing with the appropriate similarity measures.
(v) *Reproducibility and Adaptability:* To ensure transparency and facilitate replication, it is recommended to adopt open-source libraries such as `MultiDistances.jl` and `scikit-learn`. These tools support the application of the proposed methodology across varying industrial contexts and enable further customization by the testing community.

12 Concluding remarks

In this study, we analyzed the trade-off between performance and effort when employing text mining and clustering techniques for grouping manual integration test cases based on their functional dependencies, as identified through their test specifications.

We evaluated and applied the proposed approach using various text mining and clustering techniques in a real industrial case study at Alstom, Sweden. While our solution has been demonstrated in the domain of software testing, it is not limited to this context. The approach is generalizable to other problems where text documents must be clustered based on shared characteristics or dependencies. In summary, we have made the following contributions:

(i) Applied multiple text mining and similarity estimation techniques, with different levels of complexity, to a real, industrial case,
(ii) Proposed the use of clustering algorithms of similarity estimates to understand dependent and independent test cases,
(iii) Evaluated the performance of the different text processing pipelines (similarity estimation followed by clustering) against the ground truth based on F1-Scores,
(iv) Evaluated the impact of data preprocessing (with or without tokenization), and
(v) Analyzed how text length affects the performance of the different techniques.

While the Doc2Vec approach combined with agglomerative clustering achieved the highest F1-Score overall, the supposedly more advanced SBERT-based methodology fell short. Doc2Vec's performance was heavily influenced by the choice of clustering algorithm, performing poorly without agglomerative clustering. SBERT and simpler compression-based methods showed less sensitivity to clustering choice but still exhibited considerable variation. Surprisingly, even basic string distance measures like Jaro and Sorensen-Dice approached or surpassed the performance of SBERT. Our experiments also revealed that algorithm performance is affected by the length of test cases, with Doc2Vec declining as test cases grew longer, while compression-based methods excelled in identifying independence among lengthy cases. Despite the promise of AI/ML in software engineering, their adoption in testing practices remains limited, potentially due to perceived complexity and resource requirements. Our study emphasizes the importance of considering simpler algorithms alongside complex neural network-based approaches, as their utility may not always justify their complexity. The optimal choice of technique depends on available information and specific goals, and simpler algorithms can offer speed and ease of use for engineers without specialized exper-

tise. Researchers should explore a variety of approaches rather than assuming that the latest AI/ML advancements will universally benefit software engineering.

Author Contributions Sahar Tahvili: Conceptualization, Methodology, Contact with the industrial partner, Data collection. Leo Hatvani: Conceptualization, Methodology, Implementation, Visualization. Michael Felderer: State of the art and Conceptualization. Francisco Gomes: State of the art. Wasif Afzal:State of the art. Robert Feldt: State of the art and Conceptualization.

Funding Open access funding provided by Mälardalen University. No funding was obtained for this study.

Data Availability The data used in this study was collected from Alstom and is subject to confidentiality agreements, therefore, it cannot be shared publicly. However, representative samples of the utilized data, such as excerpts from test specifications specifically released by Alstom, have been included in the manuscript to illustrate the methodology. Additional materials, including the findings and supplementary resources from this study, are available at: https://github.com/leohatvani/Comparative-Analysis-Functional-Dependency.

Declarations

Competing Interests The authors declare no competing interests.

References

Ansari, A., Khan, A., Khan, A., & Mukadam, K. (2016). Optimized regression test using test case prioritization. *Procedia Computer Science, 79*, 152–160.

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 *The IEEE 8th international conference on software testing, verification and validation*

Beniwal, R., Jain, M., & Gupta, Y. (2021). Opinion mining to aid user acceptance testing for open beta versions. In P. Bansal, M. Tushir, V. E. Balas, & R. Srivastava (Eds.), *Proceedings of international conference on artificial intelligence and applications*. Singapore: Springer

Bergener, M., Escher, H., & Linden, K. (1976). multidimensional diagnostics in pharmacopsychiatry-results of therapy with desmethyl-loxapine (author's transl). *Arzneimittel-Forschung, 26*(2), 290–299.

Black, P. E. (2004). Ratcliff/obershelp pattern recognition. Dictionary of algorithms and data structures [online]

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment, 2008*(10), 10008.

Blum, M. G. B., Nunes, M. A., Prangle, D., & Sisson, S. A. (2013). A comparative review of dimension reduction methods in approximate Bayesian computation. *Statistical Science, 28*(2), 189–208.

Brusco, M., Steinley, D., Stevens, J., & Cradit, D. (2017). Affinity propagation: An exemplar-based tool for clustering in psychological research. *British Journal of Mathematical and Statistical Psychology,72*.

Chen, T.-H., Thomas, S. W., & Hassan, A. E. (2016). A survey on the use of topic models when mining software repositories. *Empirical Software Engineering, 21*(5), 1843–1919.

Cilibrasi, R., & Vitányi, P. M. (2005). Clustering by compression. *IEEE Transactions on Information theory, 51*(4), 1523–1545.

Cohen, W. W., Ravikumar, P., & Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of the 2003 international conference on information integration on the web*. IIWEB'03, pp. 73–78. AAAI Press, Acapulco, Mexico

Cohen-addad, V., Kanade, V., Mallmann-trenn, F., & Mathieu, C. (2019). Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM,66*(4).

Dalirsefat, S. B., Silva Meyer, A., & Mirhoseini, S. Z. (2009). Comparison of similarity coefficients used for cluster analysis with amplified fragment length polymorphism markers in the silkworm, Bombyx mori. *Journal of Insect Science,9*(1).

Dang, S. (2015). Performance evaluation of clustering algorithm using different datasets. *IJARCSMS, 3*, 167–173.

Deutsch, P. (1996). *RFC1951: Deflate compressed data format specification version 1.3*. RFC Editor, USA

Devlin, J., Chang, M. -W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (Long and Short Papers)* (pp. 4171–4186).

Espadoto, M., Hirata, N. S. T., & Telea, A. C. (2020). Deep learning multidimensional projections. *Information Visualization, 19*(3), 247–269.

Ester, M., Kriegel, H. -P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the second international conference on knowledge discovery and data mining*. KDD'96, pp. 226–231. Portland, Oregon: AAAI Press.

Felderer, M., Enoiu, E. P., & Tahvili, S. (2023). Artificial intelligence techniques in system testing. In J. R. Romero, & F. C. Inmaculada Medina-Bulo (Eds.), *Optimising the software development process with artificial intelligence*. Singapore: Springer

Feldt, R., Poulding, S., Clark, D., & Yoo, S. (2016a). Test set diameter: Quantifying the diversity of sets of test cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)* (pp. 223–233)

Feldt, R., Poulding, S., Clark, D., & Yoo, S. (2016b). Test set diameter: Quantifying the diversity of sets of test cases. In *2016 IEEE International conference on software testing, verification and validation*

Feldt, R., Torkar, R., Gorschek, T., & Afzal, W. (2008a). Searching for cognitively diverse tests: Towards universal test diversity metrics. In *2008 IEEE international conference on software testing verification and validation workshop* (pp. 178–186). IEEE

Feldt, R., Torkar, R., Gorschek, T., & Afzal, W. (2008b). Searching for cognitively diverse tests: Towards universal test diversity metrics. In *2008 IEEE international conference on software testing verification and validation workshop* (pp. 178–186).

Fischbach, J., Vogelsang, A., Spies, D., Wehrle, A., Junker, M., & Freudenstein, D. (2020). SPECMATE: Automated creation of test cases from acceptance criteria. In *2020 IEEE 13th International conference on software testing, validation and verification*

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science, 315*(5814), 972–6.

Frigui, H., & Krishnapuram, R. (1997). Clustering by competitive agglomeration. *Pattern Recognition, 30*(7), 1109–1119.

Garousi, V., Bauer, S., & Felderer, M. (2020). Nlp-assisted software testing: A systematic mapping of the literature. *Information and Software Technology, 126*, Article 106321.

Greiler, M., Deursen, A., & Zaidman, A. (2012). Measuring test case similarity to support test suite understanding. In C. A. Furia & S. Nanz (Eds.), *Objects, models, components, patterns.* Berlin, Heidelberg: Springer.

Guan, J., Zhu, F., & Bian, F. (2006). *Scalable and visualization-oriented clustering for exploratory spatial analysis* (vol. 2).

Guillot, G., & Rousset, F. (2011). Dismantling mantel tests. *Methods in Ecology and Evolution,4*.

Hatvani, L., & Tahvili, S. (2024). *Comparative analysis of text mining and clustering techniques for assessing functional dependency between manual test cases*. GitHub

Ilenic, N. (2017). *A PyTorch implementation of Paragraph Vectors (doc2vec)*. GitHub

Kannan, R., Vempala, S., & Vetta, A. (2004). On clusterings: Good, bad and spectral. *Journal of the ACM, 51*(3), 497–515.

Kaski, S., & Peltonen, J. (2011). Dimensionality reduction for data visualization [applications corner]. *IEEE Signal Processing Magazine, 28*(2), 100–104.

Kempe, D., & McSherry, F. (2004). A decentralized algorithm for spectral analysis. In *36th Annual ACM symposium on theory of computing* (pp. 561–568)

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Data preprocessing for supervised learning. *International Journal of Computer Science, 1*, 111–117.

Landin, C., Hatvani, L., Tahvili, S., Haggren, H., Längkvist, M., Loutfi, A., & Håkansson, A. (2020a). Performance comparison of two deep learning algorithms in detecting similarities between manual integration test cases. In *The fifteenth international conference on software engineering advances*

Landin, C., Tahvili, S., Haggren, H., Muhammad, A., Längkvist, M., & Loutfi, A. (2020b). 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 *Proc. of the 31st int. conf. on machine learning*

Ledru, Y., Petrenko, A., Boroday, S., & Mandran, N. (2012). Prioritizing test cases with string distances. *Automated Software Engineering, 19*(1), 65–95.

Lin, J., Jabbarvand, R., & Malek, S. (2019). Test transfer across mobile apps through semantic mapping. In *2019 34th IEEE/ACM International conference on Automated Software Engineering (ASE)*.

Lönnfalt, A., Tu, V., Gay, G., Singh, A., & Tahvili, S. (2024). An intelligent test management system for optimizing decision making during software testing. *Journal of Systems and Software, 161*, 1–10.

Maaten, L., Postma, E. O., & Herik, J. (2008). *Dimensionality reduction: A comparative review*.

Mahoney, M. (2023). *Large text compression benchmark*. Webpage. http://www.mattmahoney.net/dc/text.html

Makki, M., Van Landuyt, D., Lagaisse, B., & Joosen, W. (2018). A comparative study of workflow customization strategies: Quality implications for multi-tenant saas. *Journal of Systems and Software, 144*, 423–438.

Malik, M. I., Sindhu, M. A., Khattak, A. S., Abbasi, R. A., & Saleem, K. (2020). Automating test oracles from restricted natural language agile requirements. *Expert Systems,* n/a(n/a), 12608

Mann, S. K., & Chawla, S. (2020). Clustering based algorithmic design for cab recommender system (crs). In S. Fong, N. Dey, & A. Joshi (Eds.), *ICT analysis and applications* (pp. 355–363). Singapore: Springer.

Manning, C. D., Schütze, H., & Raghavan, P. (2008). *Introduction to information retrieval*. Cambridge: Cambridge University Press.

McInnes, L., & Healy, J. (2017). Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* (pp. 33–42).

Miranda, B., Bertolino, A., & Sabetta, A. (2018). Fast approaches to scalable similarity-based test case prioritization. *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*

Miranda, B., Cruciani, E., Verdecchia, R., & Bertolino, A. (2018). FAST approaches to scalable similarity-based test case prioritization. In *Proceedings of the 40th international conference on software engineering*. ICSE '18, pp. 222–232. New York, NY, USA: ACM

Mohan, V. (2015). Preprocessing techniques for text mining - an overview.

Navarro, G., Sutinen, E., & Tarhio, J. (2005). Indexing text with approximate q-grams. *Journal of Discrete Algorithms,3*(2), 157–175. Combinatorial Pattern Matching (CPM) Special Issue

Neto, F. G. D. O., Feldt, R., Erlenhov, L., & Nunes, J. B. D. S. (2018). Visualizing test diversity to support test optimisation. In *2018 25th Asia-Pacific software engineering conference* (pp. 149–158). IEEE

Noor, T. B., & Hemmati, H. (2015). A similarity-based approach for test case prioritization using historical failure data. In *2015 IEEE 26th international symposium on software reliability engineering* (pp. 58–68).

Oliveira Neto, F. G., Torkar, R., & Machado, P. D. L. (2016). Full modification coverage through automatic similarity-based test case selection. *Information and Software Technology, 80*, 124–137.

Oliveira Neto, F. G., Ahmad, A., Leifler, O., Sandahl, K., & Enoiu, E. (2018). Improving continuous integration with similarity-based test case selection. In *Proceedings of the 13th international workshop on automation of software test* (pp. 39–45). New York, NY, USA: ACM

Prescott, J., Pennell, M., Best, T., Swanson, M., Haq, F., Jackson, R., & Gurcan, M. (2009). An automated method to segment the femur for osteoarthritis research. *Conference proceedings: ... annual international conference of the IEEE engineering in medicine and biology society*. IEEE Engineering in Medicine and Biology Society. Conference 2009, pp. 6364–7.

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 3973–3983)

Roy, C. K., Zibran, M. F., & Koschke, R. (2014). The vision of software clone management: Past, present, and future. In *2014 Software evolution week - IEEE conference on software maintenance, reengineering, and reverse engineering*

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*(2), 131.

Sander, J. (2010). In C. Sammut, & G. I. Webb (Eds.), *Density-based clustering* (pp. 270–273). Boston, MA: Springer

Shi, Q., Chen, Z., Fang, C., Feng, Y., & Xu, B. (2016). Measuring the diversity of a test set with distance entropy. *IEEE Transactions on Reliability, 65*(1), 19–27.

Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., & Chanona-Hernández, L. (2013). Syntactic dependency-based n-grams as classification features. In *Advances in computational intelligence* (pp. 1–11). Springer, Berlin Heidelberg

Singhal, A. (2001). Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin, 24*(4), 35–43.

Sutar, S., Kumar, R., Pai, S., & BR, S. (2020). Regression test cases selection using natural language processing. In *2020 International conference on intelligent engineering and management*

Tahvili, S. (2018). *Multi-criteria optimization of system integration testing*. PhD thesis, Mälardalen University

Tahvili, S., Ahlberg, M., Fornander, E., Afzal, W., Saadatmand, M., Bohlin, M., & Sarabi, M. (2018a). Functional dependency detection for integration test cases. In *Companion of the 18th IEEE int. conf. on software quality, reliability, and security*.

Tahvili, S., Bohlin, M., Saadatmand, M., Larsson, S., Afzal, W., & Sundmark, D. (2016a). Cost-benefit analysis of using dependency knowledge at integration testing. In *The 17th int. conf. on product-focused software process improvement*

Tahvili, S., Hatvani, L., Felderer, M., Afzal, W., & Bohlin, M. (2019a). Automated functional dependency detection between test cases using doc2vec and clustering. In *The first IEEE international conference on artificial intelligence testing*

Tahvili, S., Hatvani, L., Felderer, M., Afzal, W., Saadatmand, M., & Bohlin, M. (2018b). Cluster-based test scheduling strategies using semantic relationships between test specifications. In *5th Int. workshop on requirements engineering and testing*.

Tahvili, S., Saadatmand, M., Larsson, S., Afzal, W., Bohlin, M., & Sundmark, D. (2016b). Dynamic integration test selection based on test case dependencies. In *The 11th workshop on testing: Academia-industry collaboration, practice and research techniques*

Tahvili, S., & Hatvani, L. (2022). *Artificial intelligence methods for optimization of the software testing process with practical examples and exercises*. Amsterdam: Elsevier.

Tahvili, S., Hatvani, L., Ramentol, E., Pimentel, R., Afzal, W., & Herrera, F. (2020). A novel methodology to classify test cases using natural language processing and imbalanced learning. *Engineering Applications of Artificial Intelligence, 95*, 103878.

Tahvili, S., Pimentel, R., Afzal, W., Ahlberg, M., Fornander, E., & Bohlin, M. (2019b). sortes: A supportive tool for stochastic scheduling of manual integration test cases. *Journal of IEEE Access, 6*, 1–19.

Thomas, S., Hemmati, H., Hassan, A., & Blostein, D. (2014). Static test case prioritization using topic models. *Empirical Software Engineering, 19*(1), 182–212.

Visalakshi, R., Ponnusamy, R., & Manikandan, K. (2016). *Literature survey of data mining clustering algorithms* (vol. 1, pp. 310–313).

Wang, B., & Kuo, C.-C.J. (2020). Sbert-wk: A sentence embedding method by dissecting bert-based word models. *IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28*, 2146–2157.

Welbers, K., Atteveldt, W. V., & Benoit, K. (2017). Text analysis in r. *Communication Methods and Measures, 11*(4), 245–265.

Xu, D., & Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science,2*.

Xu, R., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks, 16*(3), 645–678.

Ye, J. (2015). Improved cosine similarity measures of simplified neutrosophic sets for medical diagnoses. *Artificial Intelligence in Medicine, 63*(3), 171–179.

Zou, K., Warfield, S., Bharatha, A., Tempany, C., Kaus, M., Haker, S., Wells, W., Jolesz, F., & Kikinis, R. (2004). Statistical validation of image segmentation quality based on a spatial overlap index. *Academic Radiology, 11*, 178–89.

Sahar Tahvili is an AI product development leader at Ericsson AB and an Adjunct Associate Professor of Industrial AI Systems at Mälardalen University. With a background in applied mathematics and computer science, her research focuses on artificial intelligence, advanced software testing, and decision support systems (DSS). She previously served as a senior researcher at the Research Institutes of Sweden (RISE). In 2022, Sahar Tahvili and Leo Hatvani co-authored a book titled "Artificial Intelligence Methods for Optimization of the Software Testing Process: With Practical Examples and Exercises." also named a Best New Intelligence Testing Book by BookAuthority.

Leo Hatvani PhLic, is a Lecturer in the Division of Computer Science and Software Engineering at Mälardalen University. His research covers embedded systems, formal verification, and machine learning, with additional interests in data visualization and human-computer interaction.

Michael Felderer is a Professor of Software Engineering in the Department of Computer Science at the University of Innsbruck, Austria, and a Guest Professor at the Blekinge Institute of Technology, Sweden.

Francisco Gomes de Oliveira Neto is a Senior Lecturer in Interaction Design and Software Engineering within the Department of Computer Science and Engineering at Chalmers University of Technology.

Wasif Afzal earned his M.Sc., Licentiate, and Ph.D. in Software Engineering from Blekinge Institute of Technology (BTH), Sweden, in 2007, 2009, and 2011, respectively. He was a postdoctoral researcher at BTH from September 2011 to March 2012, and also served as an Assistant Professor of Software Engineering at Bahria University in Islamabad, Pakistan, from 2011 to 2013.

Robert Feldt is a Professor of Software Engineering at Chalmers University of Technology in Gothenburg, where he is affiliated with the Interaction Design and Software Engineering division of the Department of Computer Science and Engineering. He also holds a part-time professorship at Mid Sweden University in Östersund. Since April 2017, he has served as co-Editor-in-Chief of the Empirical Software Engineering (EMSE) journal.