WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Using Machine Learning to Categorise Flaws in Cloud-Based Software

Navender Singh[1], Dr. Rupali Ahuja[2]

[1,2]Maharshi Dayanand University, Rohtak

ABSTRACT

The fundamental issue in software development lies in identifying security-oriented vulnerabilities among reported faults, since the current failure rate in delivering reliable results for customer and product datasets is inadequate. The efficacy of the bug prediction model is compromised when bug reports are misclassified. The issue of misclassification significantly undermines the system's dependability. In order to tackle this matter, developers and testers are required to allocate a significant amount of time and effort into doing meticulous manual evaluations of defect reports. In this research, the authors propose an innovative integrated approach that combines machine learning and natural language processing (NLP). The objectives of addressing these issues involve the implementation of multi-class supervised classification and the utilisation of supervised classifiers for bug priority. Once the data was acquired, it was next subjected to exploratory data analysis, preprocessing, and vectorization. The techniques of TF-IDF and word2vec are employed for the purpose of extracting and selecting features from a given bag of words. Once the dataset has undergone a comprehensive transformation, the subsequent step involves the development of machine learning models. This study proposes, develops, and evaluates four classifiers: multinomial Naive Bayes, a decision tree, logistic regression, and a random forest. Upon conducting a meticulous optimisation of the models' hyper-parameters, it was determined that the random forest algorithm exhibited superior performance. This was evidenced by a test accuracy of 91.73% and a training accuracy of 100%. The dataset, initially created for the purpose of justified classification, was rebalanced using the Synthetic Minority Over-sampling Technique (SMOTE) approach. The results of the classification tests comparing models trained on balanced and unbalanced datasets demonstrated the improved performance of the balanced dataset.

Keywords: bugs; cloud clomputing; Machine Learning; NLP; Classification

1. Introduction

The concept of "software as a service" (SaaS) pertains to the paradigm in which clients are offered pre-built software applications that are hosted and accessible to them over the Internet. One of the main categories of cloud computing is Software as a Service (SaaS), alongside Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). Software as a Service (SaaS) involves the use of a singular application instance generated by the service provider, which is thereafter shared among several users. The determination of fees associated with utilising a service is sometimes established through a contractual agreement known as a Service Level Agreement (SLA) between the service provider and the service recipient. Service level agreements (SLAs) can be established either on a per-use basis or on a recurring basis, such as monthly or annually. There exists a diverse range of Software as a Service (SaaS) applications, including but not limited to email, banking, and human resources (HR) systems [1].

In Software as a Service (SaaS) arrangements, subscribers are granted unimpeded access to the programmes they have enrolled in. The service providers are responsible for the development, deployment, and ongoing management of the applications that end users depend on for their productivity. Web browsers serve as the principal means of accessing software as a service. The responsibility of software setup and maintenance is with the programme provider, therefore relieving customers of the need to attend to these tasks. In order to avail themselves of supplementary functionalities, several programmes require users to install supplementary plugins [2].

The usage of Software-as-a-Service (SaaS) applications hosted in cloud environments is experiencing a continuous growth in its user base. The occurrence of application flaws tends to increase with higher levels of traffic, resulting in a worse user experience. Maintaining the currency of these initiatives poses a challenging task that might potentially entail substantial investments of time and financial resources. It is necessary to examine the difficulties occuring inside these systems in order for users to classify and prioritise these deficiencies, facilitating the process of development and maintenance. The mending procedure may be expedited by the development team in accordance with these established priorities. The process of prioritising and categorising challenges is a labor-intensive manual endeavour. Consequently, the automation of bug classification and prioritising is necessary [3].

The increasing importance and acceptance of cloud computing has led to a demand for the categorisation of flaws in cloud computing applications using machine learning techniques. Cloud computing has become a need for enterprises and organisations of all kinds because to its numerous benefits, including scalability, flexibility, and reduced initial expenses. Nevertheless, with the increasing development of programmes designed for utilisation in cloud computing, a corresponding rise in the identification of vulnerabilities inside these settings has been seen. Cloud computing systems are susceptible to software defects that can lead to substantial periods of inoperability, dramatically impact system efficiency, and give rise to diminished productivity,

reduced customer satisfaction, and potentially even financial setbacks. The significance of developing effective methodologies for identifying and rectifying vulnerabilities in cloud computing software is increasingly evident. Machine learning algorithms have demonstrated significant potential in addressing this problem, since they possess the capability to process large amounts of data and detect patterns indicative of errors. The utilisation of machine learning techniques in bug classification for cloud computing systems is driven by the need to improve the efficiency, accuracy, and effectiveness of problem detection and resolution processes. This approach has the potential to assist enterprises in identifying the source of problems, prioritising the resolution of those faults, and mitigating any subsequent downtime or adverse effects on their cloud infrastructure. The utilisation of machine learning techniques for bug classification has the potential to enhance the overall efficiency and reliability of cloud computing systems. This, in turn, may alleviate the workload of developers and system administrators, enabling them to allocate their time and resources towards other critical responsibilities. In summary, the application of machine learning techniques to the task of classifying difficulties in cloud computing applications represents a significant area of scholarly enquiry. The integration of this technology has promise for significantly improving the security, reliability, and efficiency of cloud computing systems, hence presenting favourable prospects for organisations across several sectors.

1.1. Problem Statement

This research paper introduces an algorithmic approach to classifying and prioritising software failures in SaaS services. The occurrence of suboptimal user experiences can be attributed to software flaws or application issues, while the task of managing the increasing array of Software-as-a-Service (SaaS) applications is becoming progressively challenging. The objective of this study is to establish a systematic methodology for categorising and prioritising error reports and software faults in software as a service (SaaS) applications.

1.2. Research Objective

The objectives of our investigation are as follows:

- To develop a machine learning system capable of accurately detecting and classifying anomalies in cloud-based web services.
- In order to establish a hierarchy of concerns according to their pre-existing categorisation.
- In order to get optimal results and enhance precision in the classification and prioritisation of bugs,

1.3. Significance of Work

The presence of vulnerabilities in cloud computing applications can potentially compromise security, dependability, and performance. Furthermore, the occurrence of mistakes has the potential to result in unintended periods of system unavailability for users, since they may lead to the loss or corruption of data. Furthermore, it is worth noting that malevolent entities have the potential to exploit weaknesses inside a system in order to gain unauthorised access and exfiltrate sensitive data. In order to ensure the quality and reliability of their cloud computing applications, developers are required to thoroughly test and debug them to identify and eliminate any potential bugs or errors prior to their public release.

The introductory section of the research paper titled "Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques" provides a thorough overview of the utilisation of machine learning techniques for the purpose of classifying and categorising bugs in cloud computing applications. This section includes a detailed description of the problem statements as well as an explanation of the significance of the research work. The essay starts by providing a comprehensive introduction to cloud computing, followed by a discourse on the significance of identifying and rectifying software vulnerabilities. Subsequently, the discussion proceeds to include several machine learning techniques employed for bug categorisation, including decision trees, support vector machines (SVMs), and neural networks (NNs). Section 2 delves into the existing body of research that has examined resource allocation, configuration, and security vulnerabilities in cloud computing systems. In the third section, an examination is conducted on the complete strategy and its implementation through the use of many classifiers. The paper concludes by presenting a comprehensive methodology for bug categorisation through the utilisation of machine learning techniques. The methodology entails the collection and analysis of data, the identification and extraction of relevant characteristics, and the creation and assessment of models. In summary, this article serves as a valuable resource for anyone seeking to expand their knowledge on leveraging machine learning techniques to optimise the efficacy and functionality of cloud computing applications through the identification and resolution of flaws.

2. Literature Review

In order to ensure the accuracy and longevity of software systems, the process of Verification and Validation (V&V) include testing as an essential component. The objective of software development budgeting is to establish an effective way for predicting software flaws by utilising machine learning techniques based on softer computing. These methods aid in defect prediction, optimisation, and the acquisition of efficient features. Concurrently, the process of software analysis necessitates a significant allocation of time, money, technological infrastructure, and expertise. As the complexity of safety-critical software systems increases, there is a corresponding increase in the resources, both in terms of time and financial investment, required to ensure their reliability. Hence, it is imperative for enterprises to embrace a more analytical approach when dealing with high-stakes situations [4].

The major aim of our project is to devise an automated system that can identify issues in Software-as-a-Service (SaaS) applications throughout their distribution phase. Furthermore, the system will be designed to classify these problems and give them appropriate levels of priority. Extensive study has been undertaken about the categorisation of software bugs through the analysis of bug reports. Scholars have systematically classified these flaws and allocated priority levels to them. Furthermore, a plethora of ways has been developed to extract issues from software code and subsequently prioritise them based on their severity. Consequently, a comprehensive examination of the classification and ranking of software defects has been presented in the form of a literature study [5].

According to the findings presented in reference [6], the use of machine learning has been proposed as a potential means to automate some processes within software development, such as bug prioritising. This application of machine learning has the potential to significantly reduce the time burden on developers. The application has the capability to analyse historical data from bug reports in order to identify patterns and make predictions regarding the significance of emerging issues. This functionality serves to optimise and enhance the overall process. The process of manually detecting errors may be both time-consuming and prone to mistakes, mostly due to the increasing complexity of software systems. In safety-critical and mission-critical domains, it is imperative for the software system to possess a high level of quality and exhibit a minimal occurrence of errors. The utilisation of machine learning technologies for the automated detection of problematic components has the potential to contribute to the achievement of this objective. Additionally, this might potentially result in reduced costs and time requirements for software development and maintenance. [7].

Based on the results of this work, the use of ensemble-based approaches in bug triage emerges as an interesting area of investigation [8]. This approach holds potential for augmenting the efficacy and efficiency of bug management procedures. The superiority of ensemble classifiers over conventional machine learning algorithms in accurately allocating bug reports to the most suitable engineer is a promising development. The suggestion is that the utilisation of ensemble methodologies in the identification of the most appropriate programmer for resolving each bug report has the potential to enhance the effectiveness and accuracy of bug triaging.

The authors proposed a system for ranking bug reports that is based on emotions, as mentioned in reference [9]. The researchers employed machine learning classifiers to collect data from many online sources for the purpose of training the model. The retrieved feature vector from the problem report was utilised to generate a forecast about the priority. The bug report undergoes preprocessing utilising Natural Language Processing techniques. The bug report description was analysed using preprocessing techniques to identify emotional words, which were subsequently assigned numerical values. The model was evaluated using the Eclipse open-source projects. The F1 score demonstrates that the mechanism of the research exhibits a superior performance compared to its predecessor, with an improvement of over 6%.

Reducing error rates manually in contemporary software systems has become increasingly challenging, mostly because to the widespread presence of elements such as extensive storage capacity, high-speed internet connectivity, and the proliferation of Internet of Things (IoT) devices. Researchers have proposed a collaborative software defect computation prototype as a means to comprehend the class disparity issue in real-world software datasets. In the study conducted by the authors [10], an ensemble classifier was developed, including several oversampling techniques to address the issue of sparse samples in the flawed dataset. The findings indicate that the collaborative oversampling strategy outperforms traditional classification approaches in accurately identifying broken gears, while also significantly reducing the rate of false unfavourable identifications. The problem description process plays a key role as it serves as the means to identify the appropriate individual or entity capable of resolving the issue at hand. The process of problem correction is greatly delayed due to incorrect distribution.

The convergence of Blockchain and ERP systems is an exciting concept. The integration of blockchain technology into Accounting Information Systems (AIS) has significant promise in enhancing the dependability and precision of financial reporting [11]. The incorporation of blockchain technology into the enterprise resource planning (ERP) system enhances the security and integrity of data vaults, hence instilling confidence in the accuracy and reliability of the stored information. Industries like as banking, healthcare, and government, which place significant emphasis on data security and integrity, stand to derive substantial advantages from this technology. The potential widespread adoption of this technology may encounter scalability challenges due to the inevitable slowdown of the system caused by the expansion of the Blockchain. Nevertheless, the issue at hand might potentially be mitigated by the implementation of technological advancements and strategies such as sharding. The observation of the recommended approach's actual application and its impact on the field of AIS is anticipated to be intriguing.

In conventional software projects, which are characterised by their complexity, steady expansion, and compositional nature, the presence of defects poses a significant challenge to ensuring reliability and efficiency. In the present investigation, a model was constructed and three managed machine learning techniques (namely, Logistic Regression, Naive Bayes, and Decision Trees) were utilised to predict the occurrence of software issues based on past data. A novel approach was employed to develop an improved predictive model by leveraging collective classifiers such as Random Forest (RF). This involved applying advanced classifier techniques and generating replicas to boost the accuracy and reliability of the predictions. The accuracy of the model in predicting future software issues under both baseline and high-stress settings was verified using K-Fold cross-validation [12].

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used tool in the domains of text processing and information retrieval. Term frequency-inverse document frequency (TF-IDF) is a numerical metric that quantifies the frequency of a certain term inside a text or a corpus of documents. The essential principle of TF-IDF is that the importance of a word is enhanced when it occurs often inside a specific document, but diminished when it is found commonly across several texts.

The product of the term frequency (TF) and the inverse document frequency (IDF) results in the TF-IDF value. The frequency of a word in a document can be quantified as the logarithm of the ratio between the total number of documents and the number of documents that include the specific phrase. This measure is sometimes referred to as the inverse document frequency. TF-IDF is a widely employed technique in several information retrieval applications,

including search engines, document categorisation, and text clustering. The proposed approach provides a clear and direct means of describing the content of a document and effectively comparing it to the content of other documents. This approach enhances the significance of phrases that are exclusive to a certain text while diminishing the importance of words that are often found across several articles.

In certain instances, individuals involved in software development, testing, and consumption may erroneously categorise bug reports as enhancement requests, and conversely, leading to a laborious and challenging procedure for submitting and resolving software issues. The potential efficacy of this approach might be significantly augmented with the use of automated classification of these reports utilising machine learning techniques. This study aims to examine the process of classifying bug reports from three open-source projects through the utilisation of diverse machine learning techniques. These techniques include Naive Bayes, linear discriminant analysis, k-nearest neighbours, support vector machine (SVM) with various kernels, decision trees, and random forests. Based on our conducted studies, it has been shown that random forests exhibit superior performance. However, it is worth noting that support vector machines (SVM) outfitted with certain kernels also demonstrate encouraging outcomes. Metrics such as F-measure, average accuracy, and weighted average F-measure are employed to evaluate the results, yielding valuable insights into the potential of machine learning in automating the classification of software issues [13].

The utilisation of machine learning techniques in bug tracking to automate problem classification has the potential to alleviate the workload of developers and testers, as well as expedite the bug fixing process. The aforementioned capability possesses the potential to expedite the software testing procedure and yield cost savings. The use of machine learning has the potential to enhance the efficiency and effectiveness of the bug-tracking process, while also fostering improved teamwork, flexibility, and intelligent decision-making. In summary, the integration of machine learning techniques into cloud-based bug tracking systems has the potential to reduce the resources allocated to software testing, while simultaneously enhancing the overall quality of the programme. The issue at hand is particularly prevalent in the domain of embedded software design [14]. However, the field of machine learning presents a potential remedy by facilitating the identification and resolution of problems in a more streamlined manner.

3. Solution Design and Implementation

The subsequent response presents the design solution for the topic of "Classification of Bugs in Cloud Computing Applications using Machine Learning Techniques." The initial phase involves gathering data pertaining to the dysfunctions of cloud-based applications. Bug reports, tracking systems, and online discussion forums are among the several sources through which individuals might obtain such information. Data cleaning include the removal of superfluous information, handling of missing values, and reformatting of the data to render it suitable for analysis purposes. In this procedure, the data that has been purified is transformed in a manner that enables machine learning algorithms to interpret and effectively employ the fundamental attributes. This involves extracting information such as the bug's priority, description, and severity. The subsequent step is determining the most optimal machine learning technique for the purpose of categorising bugs. There are several approaches that may be employed to achieve this objective. One such approach involves doing a comprehensive review of existing literature to identify commonly utilised algorithms for bug categorisation. Another approach is assessing the effectiveness of various algorithms on a standardised dataset. After selecting an algorithm, it is trained using the changed data in order to establish a classification scheme for the flaws. The efficacy of the trained model is subsequently evaluated utilising a separate dataset. The evaluation of the model's performance in this study will involve the use of many measures, including but not limited to accuracy, precision, and recall. Finally, the model is deployed in a cloud-based production environment, enabling it to accurately classify defects in real-time as they manifest. However, it is evident from the visual representation in Figure 1 that the aforementioned idea is only a broad conceptual framework. Consequently, the practical execution of this concept may necessitate the inclusion of supplementary stages or the adoption of an alternative approach.

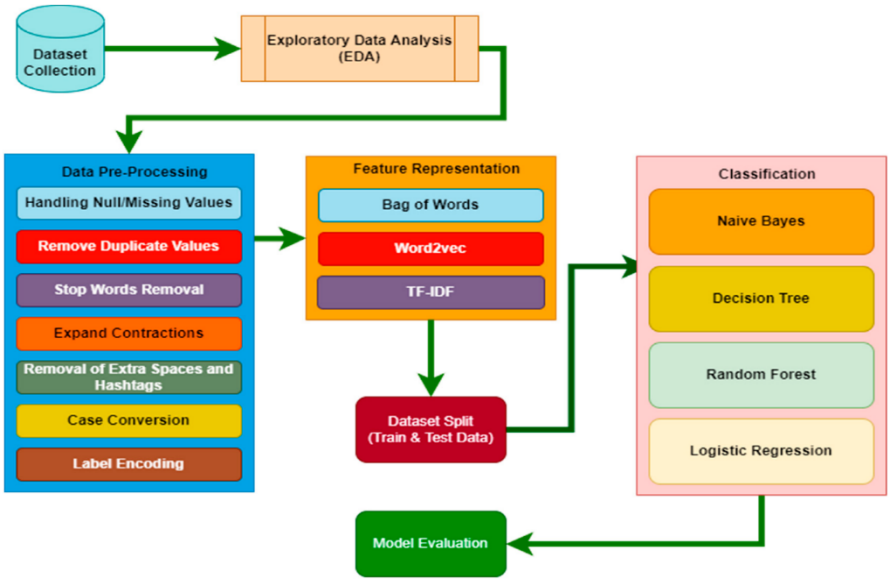


Figure 1. Proposed System Model.

3.1. Dataset Description

Data science is heavily dependent on the acquisition and use of information, since it is considered a valuable asset. When the data possesses a high level of quality, it instills trust in the ability to effectively train a dependable model. Similarly, in this case, the initial stage was the accumulation of the dataset. The task of bug classification and prediction is widely recognised as one of the most challenging endeavours in contemporary times. This study utilises the sole dataset that is publically accessible, mostly because to its rarity and limited availability. In the year 2020, a dataset was made available on the Kaggle platform [15]. Table 1 presents an enumeration of the attributes pertaining to the dataset.

Table 1. Dataset Description.

SR#	Attribute	Description	DataType
1	Unnamed	The column has unique ID's against each record.	Integer
2	Title	The column contains all text of error as a record.	Object
3	Type	Label column.	Integer

Based on its nomenclature, it may be inferred that this software application is a web-based bug tracking system specifically designed for Python programming language. The data gathering encompasses five main kinds of errors, namely improvement, security, compilation error, wasteful use of resources, bad performance, and crashes. The classification challenge in this area is intriguing, as the titles frequently contain code-related terms such as SyntaxError or ImportError, as opposed to plain language [16]. As seen in Table 2, the dataset encompasses six main categories of problems.

Table 2. Types of bug.

Title	Type
Doc strings omitted from AST	Performance
Upload failed (400): Digests do not match on .tar.gz ending with x0d binary code	Resource usage
ConfigParser writes a superfluous final bank line	Performance
csv.reader() to support QUOTE_ALL	Crash
IDLE: make smart indent after comments line consistent	Performance
xml.etree.Elementinclude does not include nested xincludes	Crash
Add Py_BREAKPOINT and sys._breakpoint hooks	Crash
documentation of ZipFile file name encoding	Performance
Allow 'continue' in 'finally' clause	Crash
Move unwinding od stack for "pseudo exceptions" from interpreter to compile	Crash
Improve regular expression HOWTO	Crash
Windows python cannot handle an early PATH entry containing "…" and python.exe	Enhancement
tkinter after_cancel does not behave correctly when called with id=None	Performance
PEP 1: Allow provisional status for PEPs	Crash
os.chdir(), os.getcwd() may crash on windows in presence of races	Enhancement
tk busy command	Crash
os.chdir() may leak memory on windows	Compiler error

Data visualisation techniques are commonly employed in the context of exploratory data analysis (EDA) to facilitate the examination and synthesis of data sets. The proficiency to ascertain the appropriate methods for manipulating data sources in order to obtain the desired outcomes enhances the efficacy of data scientists in uncovering patterns, detecting anomalies, conducting hypothesis testing, and validating assumptions.

Exploratory Data Analysis (EDA) facilitates a deeper understanding of the variables and their interrelationships within a given dataset. This article delves into the exploration of additional insights that may be derived from the data, extending beyond the confines of the required testing of formal models or hypotheses. Furthermore, it is worth noting that the evaluation of the credibility of prospective statistical methodologies for data analysis can be facilitated [17]. The practise of data discovery continues to employ exploratory data analysis (EDA) techniques, originally formulated by the renowned American mathematician John Tukey throughout the 1970s.

The dataset has dimensions of (5300,3), indicating that it encompasses 5300 instances and a total of 3 attributes, including the label. Given the presence of three columns in the dataset, it can be inferred that these columns are denoted as Unnamed: 0, title, and type. The label/class, which encompasses the complete text of any code errors, is positioned afterwards to the Unnamed property, which serves as the unique identifier index column.

The utilisation of a bar chart facilitates the visual representation of the many types of mistakes present within the dataset. Figure 2 illustrates the presence of six separate categories of failures, wherein performance issues and crashes emerge as the prevailing types, while faults associated with resource utilisation exhibit the lowest occurrence rate. Balancing the dataset is not necessary in the context of multi-class classification. The evaluation of the efficacy of a machine learning model may be assessed using cross-validation, a statistical methodology. The utilisation of this technique assists in mitigating the issue of overfitting, a phenomenon characterised by a model achieving high performance on the training data but exhibiting poor generalisation capabilities when applied to new, unseen data.

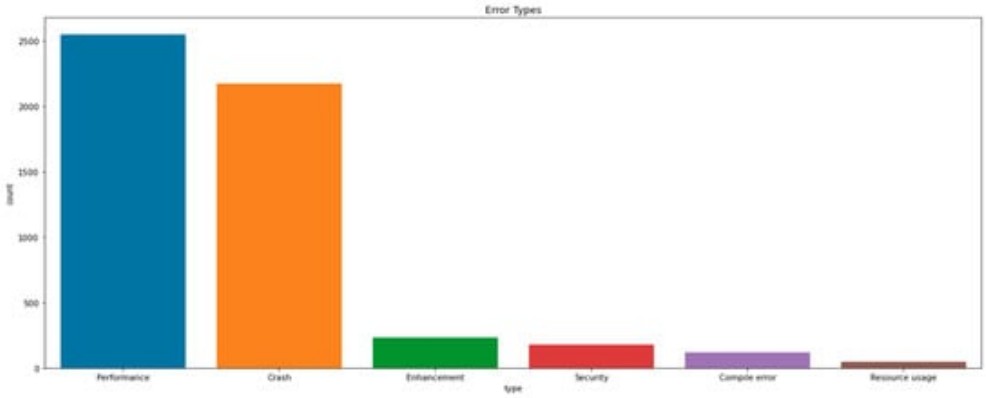


Figure 2. Graph showing "type" as an attribute bar.

The dataset is partitioned into several smaller subsets, commonly referred to as "folds," throughout the process of cross-validation. The model undergoes training on k-1 folds, followed by evaluation on the remaining fold. In each of the k iterations, every fold is used as the test set exactly once. Enhancing the model's predictive capability on unseen data is accomplished by consolidating the outcomes of several train and test divisions, commonly referred to as "folds," into a unified performance metric.

A unigram, sometimes referred to as a 1 g, is a linguistic unit consisting of a single word. In the context of uni-grams, it is postulated that each individual word has originated independently from the others. This implies that the significance of every term inside this particular context adheres to grammatical rules.

The phrases that are most commonly utilised are taken from the property known as "title uni-grams" and subsequently visualised using plotting. The scatter plot in Figure 3 depicts the distribution of the ten most often occuring phrases. Hence, the phrase "module" appears most frequently in the list of error texts, while "python" is the least frequently occuring keyword. The most often seen terms in Python-related git errors include module, windows, files, documentation, file, argument, function, functions, code, and python.

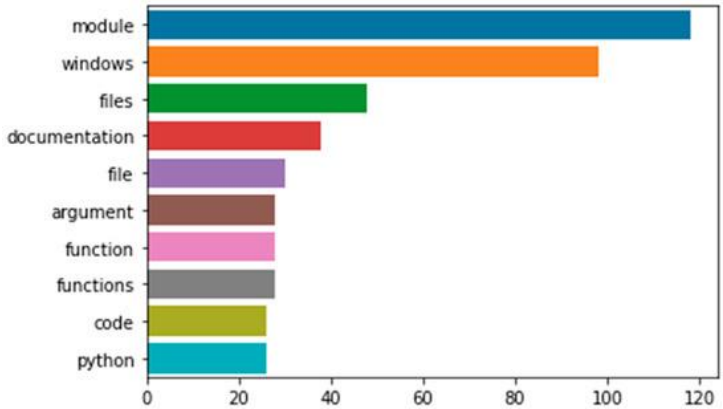


Figure 3. Unigrams that occur most often in the data collection.

The word cloud plot is a technique utilised for visualising the relative occurrence of keywords inside a certain text by adjusting the size of individual words in proportion to their frequency. A word cloud is then generated using the entirety of the gathered words.

Figure 4 presents a visual depiction of performance variation in the form of a word cloud. Frequently recurring terms such as "python," "IDLE," "return," "module," and "exception" prominently emerge, signifying their significance as fundamental notions within this context.

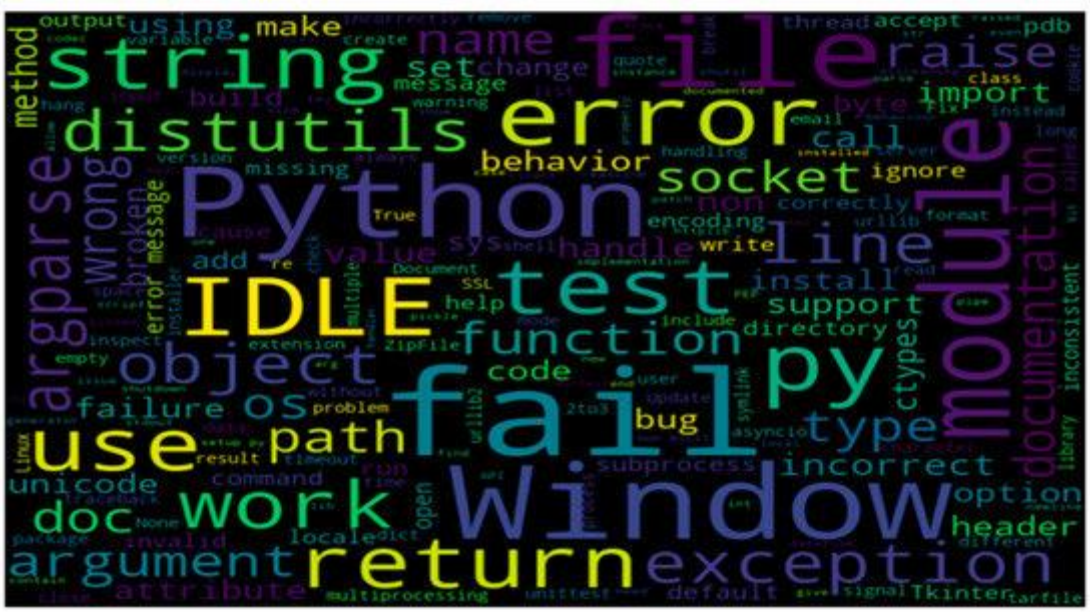


Figure 4. Definitions, synonyms, and antonyms for "performance error" in a Word Cloud format.

The most prevalent errors occur in the usage of cloud and storage resources. The visualisation of the error in resource utilisation was represented in the form of a word cloud, as seen in Figure 5. The often seen phrases within this particular situation encompass "attack," "security," "injection," and "header."

Figure 5. Error-ridden "Resource utilisation" word cloud.

The utilisation of visual representations known as "word clouds" serves to emphasise the prevalence of specific phrases within a given text, wherein the size and prominence of these keywords are increased in proportion to their frequency of occurrence, while less common words are comparatively diminished in size and visibility. The use of patterns and themes in textual data can aid in the identification of frequently occuring terms associated with each category of mistakes. The word clouds may be used to infer the frequency of words in each category of mistakes. In the event that the often appearing terms in the word cloud of spelling mistakes mostly consist of different forms of regularly misspelt words, it might suggest the necessity of implementing spell-checking software or providing further training to effectively address this specific type of error. Likewise, in the event if the predominant terms observed in the word cloud representing grammatical errors consist of conjunctions or prepositions, it is plausible to infer that students might benefit from additional exercises pertaining to these particular word classes and their appropriate application.

3.2. Preparing Datasets

The preprocessing of data mining involves the conversion of unprocessed data into a format that is suitable for computer processing, hence enabling the extraction of meaningful information. The findings of this study illustrate the constraints, disparities, and frequent inaccuracies associated with real-world data. The figure presented below illustrates the preprocessing methodologies employed in the current investigation.

The two primary stages of dataset preparation encompass the collection of raw data and the first preprocessing and labelling of the data. The assessment of the fundamental aspects of a dataset includes the examination of its dimensions, namely the number of rows and columns, as well as the review of its structural organisation to guarantee its readability and suitability for analysis. The null or missing values are examined during the post-primary analysis phase in order to identify any instances of data gaps. In order to effectively address this issue and mitigate any additional loss of words, the removal of duplicate values is implemented. The third step in the process is rectifying typographical problems, grammatical inaccuracies, and formatting discrepancies, such as the inclusion of hashtags and mentions. Figure 11 illustrates the successful implementation of the case standard and label encoding techniques. The dataset is structured as (5300,3), denoting the presence of 5300 instances and 3 features. Both objects in the collection have a single integer attribute called "title" with 5299 unique values and a frequency of 2. The present problem pertains to multi-class classification since it involves the categorisation of data instances into one of six distinct classes. These classes are determined by the values assigned to the "type" attribute, which encompasses the dataset under consideration.

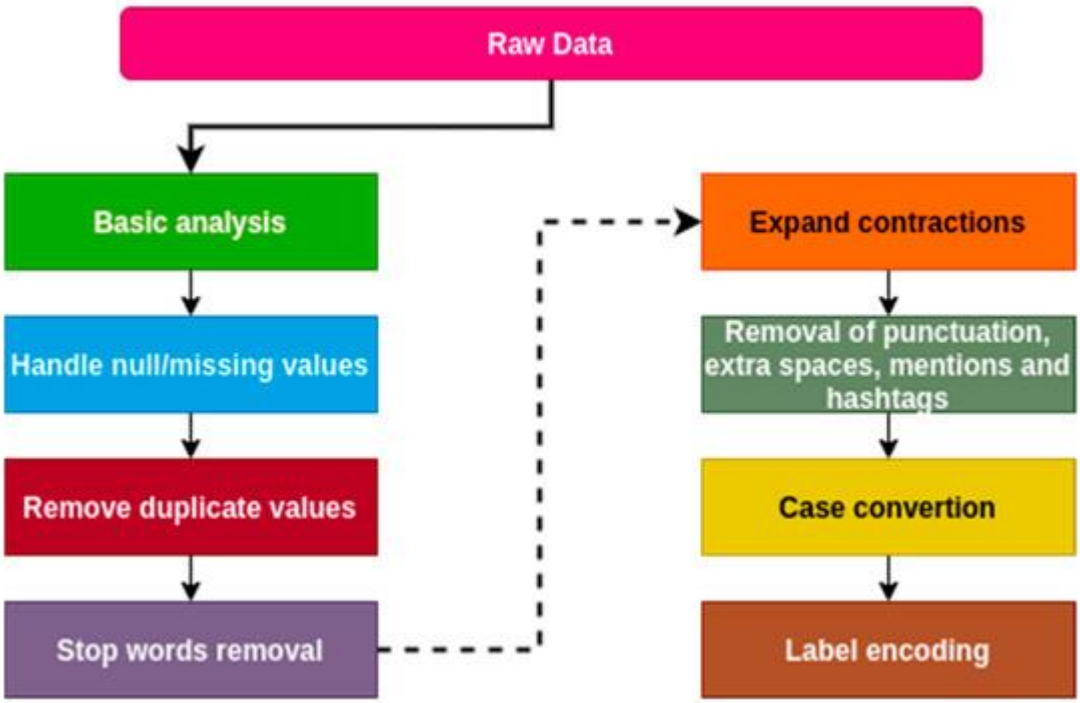


Figure 11. Preparing Dataset.

One of the challenges that requires meticulous handling in datasets is the presence of missing or null values. Alternatively, the integrity of the model may be compromised, leading to the generation of unreliable predictions. Fortunately, the dataset being utilised does not include any null or missing values.

Machine learning algorithms encounter significant challenges when confronted with duplicate data. Overfitting might manifest in some instances as a consequence of the model's inclination to excessively learn and train on the identical dataset. To mitigate this issue, it is imperative to either eliminate or preprocess duplicate variables.

A single instance of a duplicate value was eliminated from the dataset, resulting in a final configuration of (5299,2). Due to its very poor correlation with the labels, the variable labelled as "Unnamed: 0" was subsequently excluded from the analysis.

This marks the start of the most intricate preparatory measures. Eliminating stop words from the textual material is a crucial initial stage. Given that stop words lack semantic relevance, the inclusion of these terms in a dataset renders it ineffective for the purpose of text categorisation. In the process of indexing material for the purpose of searching and retrieval, a search engine has the capability to exclude occurrences of a certain phrase referred to as a "stop word." The terms "a," "an," and "in" are considered as stop words. Typically, such types of information are commonly overlooked due to their necessitation of more store capacity within the database and subsequent augmentation of processing durations. Another advantage of eliminating stopwords is the reduction in dataset size. The Python package NLTK (Natural Language Toolkit) facilitates the elimination of stop words. The process of stop-word filtering is achieved by utilising the Natural Language Toolkit (NLTK). The inclusion of English stopword lists is justified by the linguistic characteristics of the dataset [18].

Writing that includes superfluous features such as punctuation marks, excessive white space, hashtags, and mentions can be considered as writing that lacks cleanliness or refinement. Furthermore, the inclusion of numerical values is of utmost importance in this context, as typographical errors can have significant consequences for every individual digit or numeric representation present within the written content. This is the rationale behind abstaining from rounding numbers or truncating digits.

The process of case folding plays a vital role in addressing the issue of bias and facilitating text comprehension for robots. The process of case folding, as seen in Figure 12, involves converting all text to lowercase utilising the Python package [19].

type	clean_text
Performance	doc strings omitted ast
Resource usage	upload failed 400 digests match tar gz ending x0d binary code
Performance	configparser writes superfluous final blank line
Performance	applepersistenceignorestate warning macos
Performance	implement pep 529 os getcwdb windows

Figure 12. Dataset after case conversion.

Following a proficient cleaning process, the subsequent stage involves the transformation of the data. The dataset has a column labelled "type" that consists of categorical data. The LabelEncoding package is utilised for the purpose of converting data into a numerical representation. The dataset labels were prepared according to the pattern depicted in Figure 13.

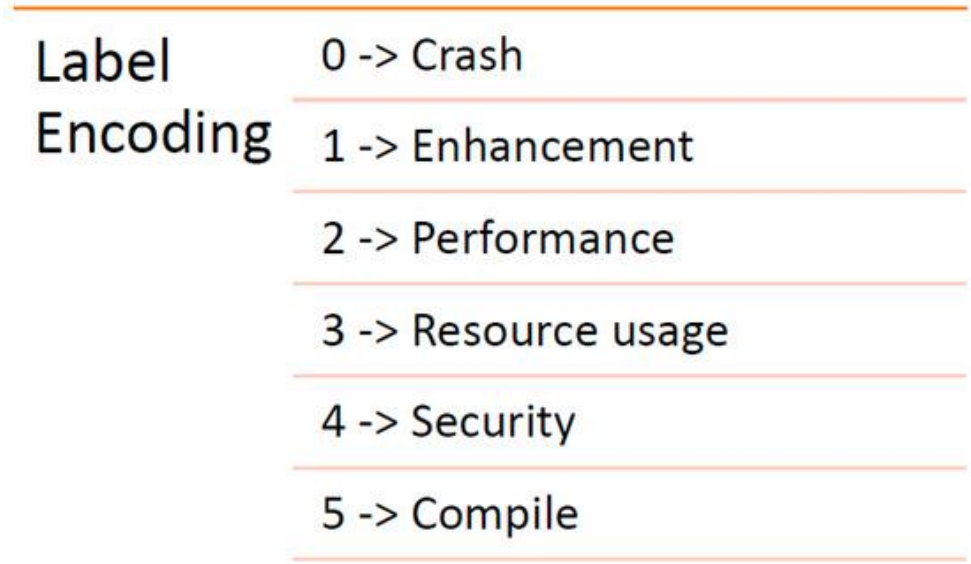


Figure 13. Coding of Labels.

Historically, label encoding has been commonly employed as the final stage in the preprocessing pipeline of machine learning. However, given that the information provided is in textual form, an additional step is required, namely the utilisation of feature vector representation. This work has examined two conventional techniques, namely Bag-of-words and TF-IDF, together with a state-of-the-art approach known as word2vec, for the purpose of word embedding.

The weight characteristic is determined by the frequency of individual words in relation to the total number of phrases. The formula for it may be expressed as equation (1).

Equation (1) is the term frequency-inverse document frequency (TF-IDF) formula, where N_(t,d) denotes the frequency of term t in document d, N_d represents the total number of terms in document d, and TF_(

The TF-IDF method has demonstrated a history of successful use across several contexts. Furthermore, the models have undergone testing using TF-IDF vectorization on the training dataset.

In other words, the term frequency-inverse document frequency (tf-idf) formula may be expressed as tf-idf(t,d,D) = tf(t,d).Let tf (t,d) be defined as the logarithm of one plus the frequency of term t in document d, denoted as log(1 + freq (t,d)) (3).

The indefinite disjoint function (t,D), also known as idf (t,D), is defined as the logarithm of the ratio between the total number of documents in a corpus (N) and the count of documents (d) in which a specific term (t) appears. (4)

The TF-IDF vectorization function was imported using the sklearn package, and subsequently applied to the training data for vectorization. The dataset was transformed into a vectorized form, resulting in a shape of (5299,5940). This indicates that 5299 records were utilised to create 5940 vectors, each representing a distinct phrase depending on its frequency.

In the year 2013, Google developed Word2vec, a word embedding technique that use a shallow neural network. There are two models that may be utilised, namely the Continuous Bag of Word model and the Skip gramme model (20).

When the value of min_count is set to 5, only words that consist of a minimum of five characters are taken into consideration.

We will exclusively evaluate sentences that possess a minimum length of 50 characters. The training process of the model will include the use of a workforce consisting of four individuals (workers = 4).

Given that word2vec is the default model, it will be employed for the training of the continuous bag of words.

In order to prevent the development of biassed models, it is imperative to exclude datasets that are deemed unfair. Regrettably, the selected dataset exhibits this limitation despite the inclusion of several classes. In order to get data parity, the researchers employed the Synthetic Minority Over-sampling Technique (SMOTE) in combination with a random oversampling approach. The acronym SMOTE represents the Synthetic Minority OverSampling Technique. The Synthetic Minority Over-sampling Technique (SMOTE) is a widely used approach in data analysis and machine learning for addressing the issue of imbalanced datasets. It entails generating synthetic samples for the underrepresented group in order to balance the class distribution. The utilisation of this particular method seems to be advantageous in mitigating the potential issue of overfitting that may develop when employing a randomly selected high sample size. By directing attention towards the feature space and employing interpolation techniques with neighbouring positive examples, it is possible to produce novel instances.

Following the use of the Synthetic Minority Over-sampling Technique (SMOTE), the dataset achieved a state of equilibrium.

The dataset has been appropriately normalised for use in a classification machine learning technique, as seen in Figure 14 [21].

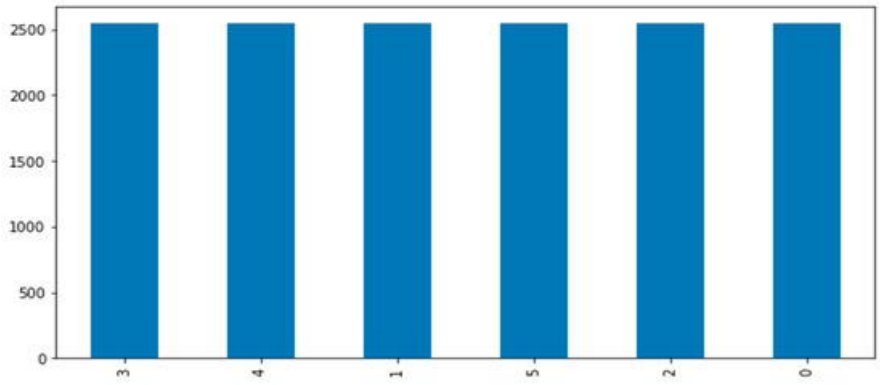


Figure 14. Rational Dataset.

4. Result and Analysis

This study employs four classifiers for classification using various vectorization strategies and experimental designs:

- Naive Bayes
- Decision-making tree
- The Random Forest Method
- Inference from Logs
- Seven experiments, in addition to the classifier comparison, are detailed below.
- Bag-of-words, TF-IDF, and Wfigord2vec, three vectorization methods.
- Tuning of parameters (for BOW and TF-IDF) by Randomised SearchCV.

- Accuracy, Precision, Recall, and F1-score comparisons among four different models.

The dataset was partitioned into two subsets, with the majority of the data allocated for training and the remaining fraction designated for testing, following an 80/20 split. The records are picked in a random manner, with an 80/20 split, with the random state set to 42. Classification models are commonly assessed based on four key criteria. It is imperative to optimise the parameters and features of the model in order to improve precision, recall, and F1 score, hence maximising accuracy within the given dataset. In the event if the precision, recall, and F1 score exhibit suboptimal values, it is important to recognise that a high accuracy metric may not always reflect robust performance. Hence, it is important to take into account all four indicators while assessing the effectiveness of the model. [22].

The major goal of our study is to optimise the accuracy of the dataset presented, while also ensuring consistency across the four input parameters [23]. The formulas for the four metrics, namely Accuracy, Precision, F1, and Recall, are presented as follows: Precision is calculated by dividing the true positives (TP) by the sum of true positives, false positives (FP), and false negatives (FN) (5). Similarly, recall is determined by dividing the true positives (TP) by the sum of true positives and false negatives (FN) (6).

The calculation of a perfect F1 score is derived as 2 multiplied by the difference between Precision and Recall, divided by the sum of Precision and Recall. (8)

Naive Bayes approaches have been developed based on Bayes' theorem, assuming conditional independence between pairs of attributes given the class variable's numerical value [24].

The Multinomial Naive classifier is imported from the sklearn package and utilised in conjunction with all the mentioned experiments, as this pertains to a multi-class classification problem. Table 3 presents a comprehensive summary of the experimental results obtained.

Table 3. Results of Naive Bayes.

Sr#	Dataset	BOW	TF-IDF	Word2vec	PT—BOW	PT—TFIDF
1	Imbalance	Train: 86.43 Test: 66.66	Train: 81.42 Test: 65.28	Train: 98.47 Test: 34.16	Train: 64.98 Test: 65.13	Train: 66.19 Test: 66.19
2	Balance	Train: 91.66 Test: 83.54	Train: 91.89 Test: 84.11	Train: 92.11 Test: 41.29	Train: 93.34 Test: 87.01	Train: 97.14 Test: 88.18

Based on the findings shown in Table 3, it can be observed that Naive Bayes exhibited superior performance compared to BOW when hyper-parameters were optimised. The training phase achieved a level of accuracy of 97.14%, whilst the testing phase yielded an accuracy rate of 88.18%.

The Naive Bayes model with Bag-of-Words (BOW) adjusted parameters, as seen in Figure 15, was utilised for the classification task. The corresponding classification report is presented in Figure 16.

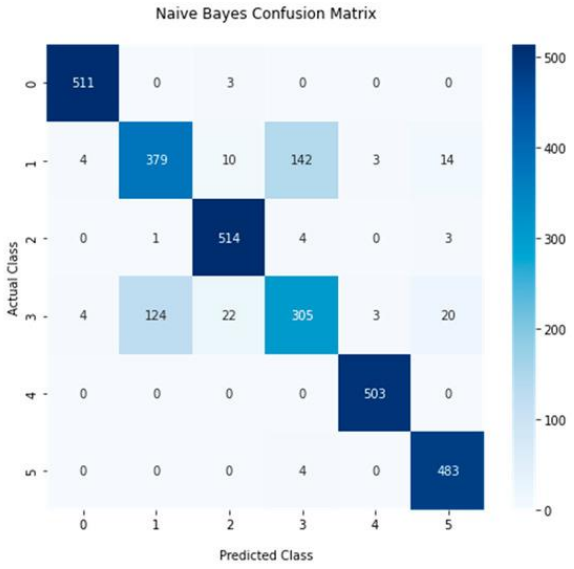
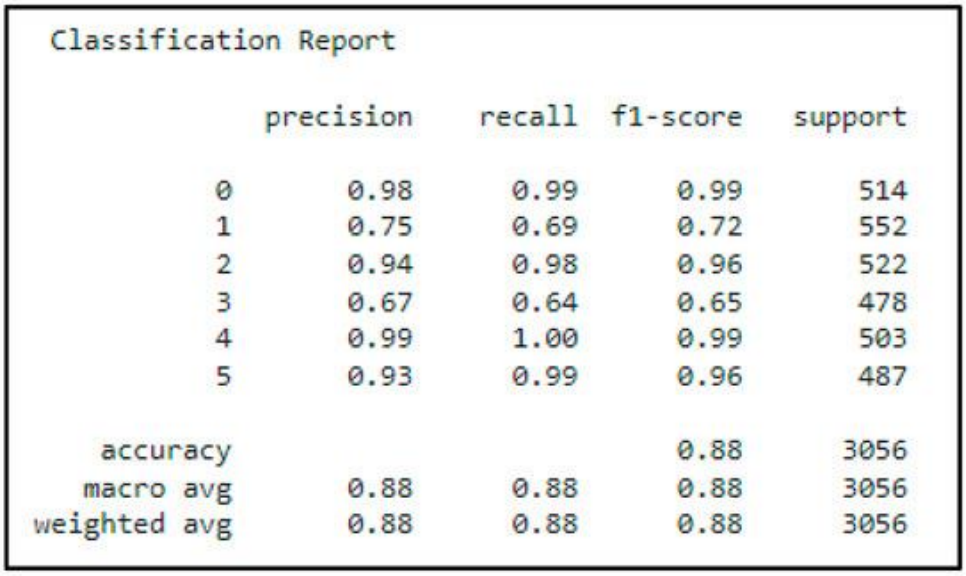


Figure 15. Naive Bayes's confusion matrix.



Classification Report

	precision	recall	f1-score	support
0	0.98	0.99	0.99	514
1	0.75	0.69	0.72	552
2	0.94	0.98	0.96	522
3	0.67	0.64	0.65	478
4	0.99	1.00	0.99	503
5	0.93	0.99	0.96	487
accuracy			0.88	3056
macro avg	0.88	0.88	0.88	3056
weighted avg	0.88	0.88	0.88	3056

Figure 16. Report on Naive Bayes Classification.

Class 4 and 5 exhibit flawless recall rates of 100%, while the remaining classes have accuracy levels ranging from 73% to 95% (refer to Figure 16). The absence of overfitting and underfitting is seen, indicating that the trained model may be effectively generalised to novel data without compromising its accuracy, as evidenced by the F1 score. Each class exerted a similar amount of effort in terms of training [25].

Decision tree analysis is a very effective approach in predictive modelling, including a diverse array of applications. The mathematical methodology employed in the construction of decision trees frequently involves the identification of techniques for dividing a dataset into subsets using various criteria. The present work involved the use of a decision tree classifier obtained from the sklearn trees package. A model was subsequently constructed, whereby the absence of a training dataset or labels was deliberately maintained. The results of all requested tests are presented in Table 4.

Table 4. Results of the decision tree.

Sr#	Dataset	BOW	TF-IDF	Word2vec	PT—BOW	PT—TFIDF
1	Imbalance	Train: 100 Test: 62.89	Train: 100 Test: 61.88	Train: 99.81 Test: 43.11	Train: 99.52 Test: 64.02	Train: 100 Test: 65
2	Balance	Train: 100 Test: 86.14	Train: 99.98 Test: 85.56	Train: 99.28 Test: 43.12	Train: 100 Test: 88.45	Train: 100 Test: 87.79

The selected hyper-parameters for the decision tree classifier were as follows: a maximum depth of 54, a minimum number of samples per leaf of 4, a minimum number of samples required to divide an internal node of 95, and the criterion used for splitting nodes was the Gini impurity.

The results shown in Table 4 indicate that the performance of the decision tree classifier remained rather consistent when employing both the Bag-of-Words (BOW) and Term Frequency-Inverse Document Frequency (TF-IDF) approaches, in comparison to the use of the default settings. The TFIDF method yields a test accuracy score of 87.59 percent and a training accuracy score of 99.89 percent, which closely trails the accuracy produced by the Bag-of-Words (BOW) approach.

The confusion matrices for the decision trees are presented in Figure 17, while the corresponding classification reports can be found in Figure 18.

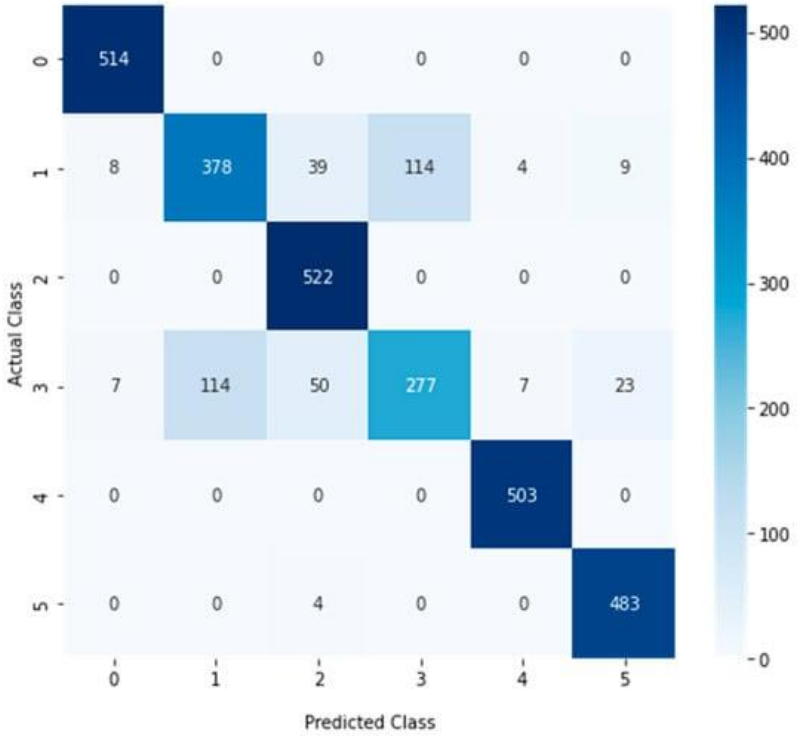


Figure 17. Decision tree confusion matrix.

Decision Tree Classification Report

	precision	recall	f1-score	support
0	0.97	1.00	0.99	514
1	0.77	0.68	0.72	552
2	0.85	1.00	0.92	522
3	0.71	0.58	0.64	478
4	0.98	1.00	0.99	503
5	0.94	0.99	0.96	487
accuracy			0.88	3056
macro avg	0.87	0.88	0.87	3056
weighted avg	0.87	0.88	0.87	3056

Figure 18. Decision Tree Classification Report.

The F1 score suggests that the generalised trained model exhibits neither overfitting nor underfitting. Based on the categorisation reports provided, the observed accuracy spans from 71% to 98%. Notably, Class 4 exhibits a recall rate of 100%, indicating an exceptional performance in correctly identifying instances belonging to this class. The influence on the classroom was nearly comparable across all groups [26].

Random forest classifiers belong to a larger category of ensemble-based learning techniques. These tools exhibit rapid learning and utilisation capabilities, possess broad applicability, and are characterised by straightforward implementation procedures. The fundamental principle behind the random forest methodology involves the creation of many decision trees, which are considered to be relatively straightforward, during the training phase. Subsequently, the median, or average, of these decision trees is employed to facilitate the process of making classification judgements [27]. The voting strategy serves as a means to alleviate the issue of decision trees tending to overfit the training data. During the training phase, random forests employ the general bagging technique on every tree inside the ensemble. The technique of Bagging involves iteratively selecting a subset of the training data and fitting trees to this subset. The random forest classifier used in this work is imported from the scikit-learn ensemble package. The model was generated using the training dataset and corresponding labels. The results of all the studies that were included in the analysis are reported in Table 5.

Table 5. Results of random forest.

Sr#	Dataset	BOW	TF-IDF	Word2vec	PT—BOW	PT—TFIDF
1	Imbalance	Train: 100 Test: 66.54	Train: 100 Test: 66.03	Train: 99.81 Test: 57.91	Train: 100 Test: 64.84	Train: 100 Test: 65.53
2	Balance	Train: 100 Test: 89.52	Train: 99.24 Test: 88.76	Train: 100 Test: 58.12	Train: 100 Test: 90.86	Train: 100 Test: 91.73

The hyper-parameters used for the random forest model were as follows: the criteria for splitting nodes was set to "entropy", the maximum depth of the tree was set to 79, the minimum number of samples required to be at a leaf node was set to 1, and the minimum number of samples required to divide an internal node was set to 79.

Based on the findings shown in Table 5, it can be concluded that the random forest classifier outperforms the TF-IDF classifier within the context of the fine-tuned hyper-parameter model. Despite the exam's performance accuracy being just 91.73%, it is noteworthy that all training objectives were successfully achieved. The confusion matrix of the random forest classifier is depicted in Figure 19, while Figure 20 presents the classification results of the model.

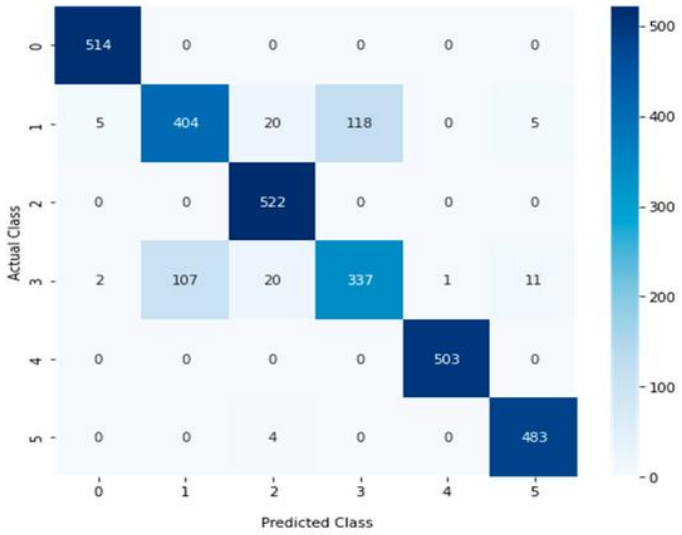


Figure 19. Random Forest Confuse Matrix.

Classification report

	precision	recall	f1-score	support
0	0.99	1.00	0.99	758
1	0.80	0.68	0.74	780
2	0.97	1.00	0.99	792
3	0.73	0.77	0.75	764
4	1.00	1.00	1.00	776
5	0.96	1.00	0.98	719
accuracy			0.91	4589
macro avg	0.91	0.91	0.91	4589
weighted avg	0.91	0.91	0.91	4589

Figure 20. Random Forest Classification report.

Based on the categorisation data provided, it can be observed that all classes exhibit accuracy levels ranging from 73% to 100%. In addition, it is noteworthy that both classes 4 and 5 have achieved a recall rate of 100%, indicating exceptional performance. The F1 score suggests that the generalised trained model exhibits neither overfitting nor underfitting. All the kids exhibited a comparable level of effort in their classroom tasks.

Logistic regression is an illustrative instance of supervised learning. In order to determine or predict the likelihood of a binary event, it is necessary to do a probability calculation. The utilisation of machine learning techniques, specifically logistic regression [26], has the potential to assist in the identification of COVID-19 infection in individuals. The logistic regression utilised in this work was imported using the sklearn linear_model package. The model was constructed without utilising the training dataset or its corresponding labels. The outcomes of all the specified experiments are consolidated and shown in Table 6.

Table 6. Results of logistic regression.

Sr#	Dataset	BOW	TF-IDF	Word2vec	PT—BOW	PT—TFIDF
1	Imbalance	Train: 95.44 Test: 66.28	Train: 84.33 Test: 65.66	Train: 48.47 Test: 46.60	Train: 66.66 Test: 65.78	Train: 67.14 Test: 66.16
2	Balance	Train: 90.52 Test: 87.52	Train: 92.24 Test: 86.22	Train: 40.12 Test: 42.15	Train: 93.37 Test: 88.27	Train: 90.03 Test: 85.55

The hyper-parameters employed for logistic regression were C = 10 and solver = "newton-cg".

The table provides convincing evidence of the ineffectiveness of logistic regression. The test accuracy achieved the greatest value of 88.27%, while the training accuracy reached its peak at 93.37%.

Figure 21 presents the confusion matrix of the logistic regression model, while Figure 22 displays the classification reports for the aforementioned model.

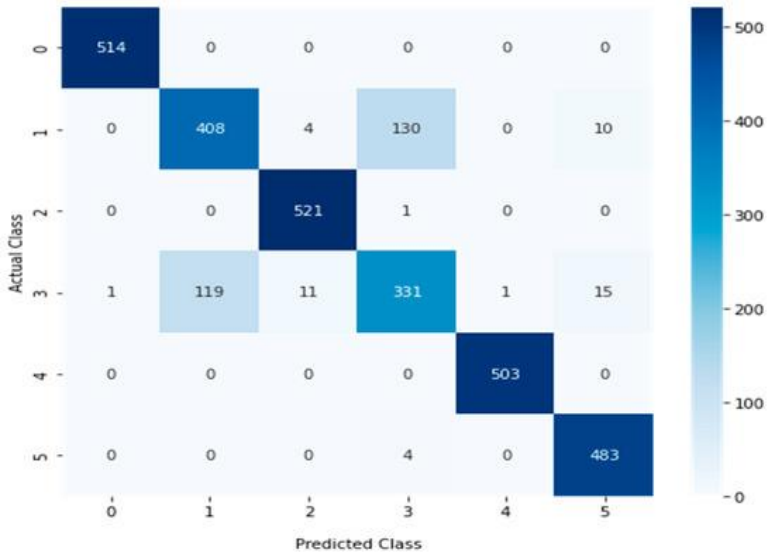


Figure 21. Logistics Regression Confusion matrix.

Classification report

	precision	recall	f1-score	support
0	0.97	0.98	0.98	758
1	0.75	0.71	0.73	780
2	0.92	0.98	0.95	792
3	0.73	0.63	0.68	764
4	0.99	1.00	0.99	776
5	0.91	1.00	0.95	719
accuracy			0.88	4589
macro avg	0.88	0.88	0.88	4589
weighted avg	0.88	0.88	0.88	4589

Figure 22. Logistic Regression Classification report.

Based on the provided classification report, it can be observed that all classes exhibit precision values ranging from 73% to 99.3%. Additionally, classes 4 and 5 have a recall rate of 100%, indicating exceptional performance across all evaluated categories. The F1 score suggests that the generalised trained model exhibits neither overfitting nor underfitting. All the kids exhibited a comparable level of effort in their classroom tasks.

In this research, errors are prioritised based on their frequency, and the insights obtained from the dataset are shown in Figure 23.

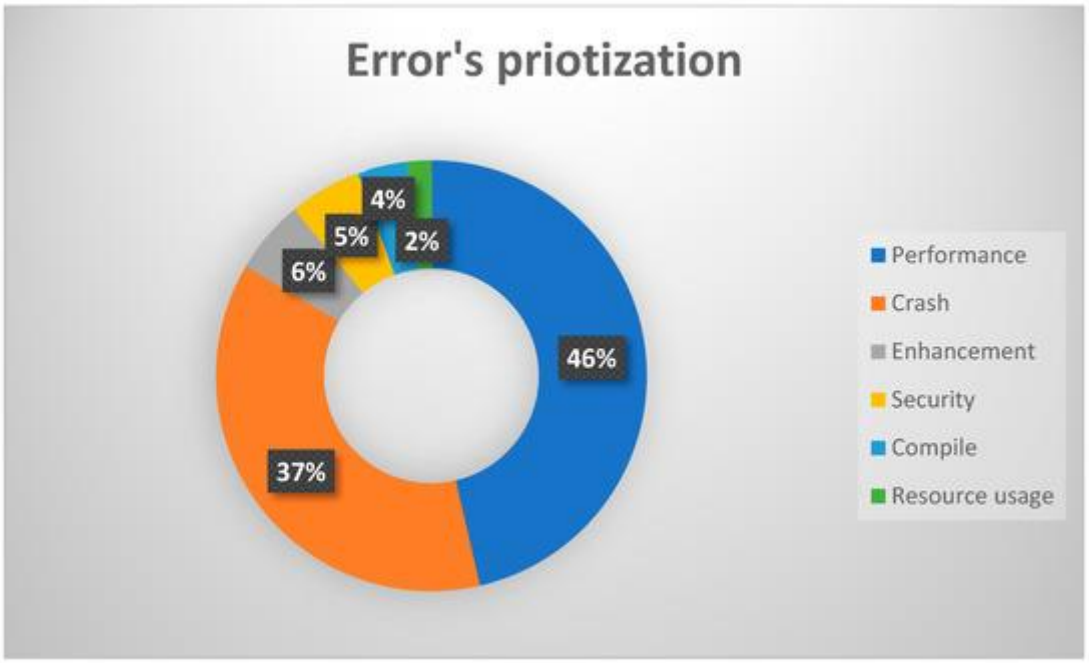


Figure 23. SaaS application bug prioritisation.

The study revealed that instances of crash mistakes were less frequent in comparison to performance issues. The frequency of improperly utilised resources was at its minimum. This commonly occurs when the user is confronted with a substantial dataset and engaging in computationally demanding operations.

5. Comparative Analysis

The research consistently yielded favourable results when employing datasets that were well balanced and optimising the hyper-parameters of the model. The construction of the four classifiers (Naive Bayes, decision tree, random forest, and logistic regression) involves the utilisation of both balanced and unbalanced datasets.

The aforementioned hyperparameters facilitated the random forest classifier in achieving its highest levels of training and test accuracy. The accuracy is visually depicted in Figure 24.

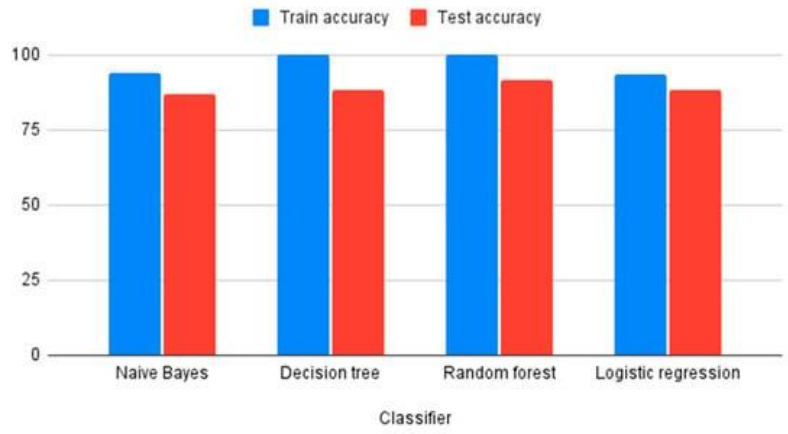


Figure 24. Comparison between training and testing accuracy as a scatter plot.

It can be inferred that the random forest classifier has greater performance in terms of both training and test accuracy when compared to other classifiers. In comparison to alternative classifiers, it exhibited strong performance in terms of recall, accuracy, and F1 score.

As cloud computing systems undergo modifications or the introduction of new capabilities, the constraints associated with the classification of issues using this approach may also undergo alterations. Over the course of time, it is possible for models to see a decrease in accuracy and reliability. In order to maintain accuracy, it is imperative to periodically retrain models using up-to-date data.

6. Conclusions

The task of classifying errors in cloud computing systems using machine learning techniques is a significant undertaking that requires both technical expertise and a deep understanding of the topic. The present study demonstrates the potential of employing machine learning methodologies for the effective identification and classification of vulnerabilities in cloud-based software. Machine learning algorithms have the capability to offer highly precise real-time bug predictions through the analysis of extensive data generated by cloud computing systems. This approach has the potential to assist organisations in mitigating costly periods of inactivity and ensuring the uninterrupted operation of their cloud computing infrastructure by expediting problem identification and resolution. Hence, the application of machine learning techniques for the classification of bugs in cloud computing applications is a very promising domain for future research and advancement. This field holds the potential to significantly transform the detection and resolution of issues inside cloud environments. Future study in this particular sector may explore other avenues. Enhance the categorisation of bugs in cloud applications by leveraging insights gained from similar classification assignments through the utilisation of transfer learning techniques. In cloud computing applications, the utilisation of domain adaption methods can enhance bug classification performance when there exists a disparity between the distributions of training and test data.

7. References:

[1] Kim, J. Deep Learning vs. Machine Learning vs. AI: An InDepth Guide, readspeaker.ai, 3 May 2021. Available online: https://www.readspeaker.ai/blog/deep-learning-vs-machine-learning/ (accessed on 16 July 2022).

[2] Thota, M.K.; Shajin, F.H.; Rajesh, P. Survey on software defect prediction techniques. Int. J. Appl. Sci. Eng. 2020, 17, 331–344. [Google Scholar] [CrossRef]

[3] Iqbal, S.; Naseem, R.; Jan, S.; Alshmrany, S.; Yasar, M.; Ali, A. Determining Bug Prioritization Using Feature Reduction and Clustering With Classification. IEEE Access 2020, 8, 215661–215678. [Google Scholar] [CrossRef]

[4] Umer, Q.; Liu, H.; Sultan, Y. Emotion Based Automated Priority Prediction for Bug Reports. IEEE Access 2018, 6, 35743–35752. [Google Scholar] [CrossRef]

[5] Harer, J.A.; Kim, L.Y.; Russell, R.L.; Ozdemir, O.; Kosta, L.R.; Rangamani, A.; Hamilton, L.H.; Centeno, G.I.; Key, J.R.; Ellingwood, P.M.; et al. Automated software vulnerability detection with machine learning. arXiv 2018, arXiv:1803.04497. [Google Scholar]

[6] Waqar, A. Software Bug Prioritization in Beta Testing Using Machine Learning Techniques. J. Comput. Soc. 2020, 1, 24–34. [Google Scholar]

[7] Huda, S.; Liu, K.; Abdelrazek, M.; Ibrahim, A.; Alyahya, S.; Al-Dossari, H.; Ahmad, S. An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction. IEEE Access 2018, 6, 24184–24195. [Google Scholar] [CrossRef]

[8] Goyal, A.; Sardana, N. Empirical Analysis of Ensemble Machine Learning Techniques for Bug Triaging. In Proceedings of the 2019 Twelfth International Conference on Contemporary Computing (IC3), Noida, India, 8–10 August 2019; pp. 1–6. [Google Scholar] [CrossRef]

[9] Gupta, A.; Sharma, S.; Goyal, S.; Rashid, M. Novel XGBoost Tuned Machine Learning Model for Software Bug Prediction. In Proceedings of the 2020 International Conference on Intelligent Engineering and Management (ICIEM), London, UK, 17–19 June 2020; pp. 376–380. [Google Scholar] [CrossRef]

[10] Ahmed, H.A.; Bawany, N.Z.; Shamsi, J.A. CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms. IEEE Access 2021, 9, 50496–50512. [Google Scholar] [CrossRef]

[11] Sarwar, M.I.; Iqbal, M.W.; Alyas, T.; Namoun, A.; Alrehaili, A.; Tufail, A.; Tabassum, N. Data Vaults for Blockchain-Empowered Accounting Information Systems. IEEE Access 2021, 9, 117306–117324. [Google Scholar] [CrossRef]

[12] Leotta, M.; Olianas, D.; Ricca, F. A large experimentation to analyze the effects of implementation bugs in machine learning algorithms. Future Gener. Comp. Syst. 2022, 133, 184–200. [Google Scholar] [CrossRef]

[13] Hai, T.; Zhou, J.; Li, N.; Jain, S.K.; Agrawal, S.; Dhaou, I.B. Cloud-based bug tracking software defects analysis using deep learning. J. Cloud. Comp. 2022, 11. [Google Scholar] [CrossRef]

[14] Pandey, N.; Sanyal, D.K.; Hudait, A.; Sen, A. Automated classification of software issue reports using machine learning techniques: An empirical study. Innov. Syst. Softw. Eng. 2017, 13, 279–297. [Google Scholar] [CrossRef]

[15] Tabassum, N.; Alyas, T.; Hamid, M.; Saleem, M.; Malik, S. Hyper-Convergence Storage Framework for EcoCloud Correlates. Comput. Mater. Contin. 2022, 70, 1573–1584. [Google Scholar] [CrossRef]

[16] Catolino, G.; Palomba, F.; Zaidman, A.; Ferrucci, F. Not all bugs are the same: Understanding, characterizing, and classifying bug types. J. Syst. Softw. 2019, 152, 165–181. [Google Scholar] [CrossRef]

[17] Kukkar, A.; Mohana, R. A Supervised Bug Report Classification with Incorporate and Textual field Knowledge. Procedia Comput. Sci. 2018, 132, 352–361. [Google Scholar] [CrossRef]

[18] Shuraym, Z. An efficient classification of secure and non-secure bug report material using machine learning method for cyber security. Mater. Today Proc. 2021, 37, 2507–2512. [Google Scholar] [CrossRef]

[19] Kukkar, A.; Mohana, R.; Nayyar, A.; Kim, J.; Kang, B.-G.; Chilamkurti, N. A Novel Deep-Learning-Based Bug Severity Classification Technique Using Convolutional Neural Networks and Random Forest with Boosting. Sensors 2019, 19, 2964. [Google Scholar] [CrossRef] [PubMed][Green Version]

[20] Dam, H.K.; Pham, T.; Ng, S.W.; Tran, T.; Grundy, J.; Ghose, A.; Kim, C.J. Lessons learned from using a deep tree-based model for software defect prediction in practice. In Proceedings of the IEEE International Working Conference on Mining Software Repositories, Montreal, QC, Canada, 26–27 May 2019; pp. 46–57. [Google Scholar] [CrossRef]

[21] Bani-Salameh, H.; Sallam, M.; Al Shboul, B. A deep-learning-based bug priority prediction using RNN-LSTM neural networks. E-Inform. Softw. Eng. J. 2021, 15, 29–45. [Google Scholar] [CrossRef]

[22] Ramay, W.Y.; Umer, Q.; Yin, X.C.; Zhu, C.; Illahi, I. Deep Neural Network-Based Severity Prediction of Bug Reports. IEEE Access 2019, 7, 46846–46857. [Google Scholar] [CrossRef]

[23] Polat, H.; Polat, O.; Cetin, A. Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models. Sustainability 2020, 12, 1035. [Google Scholar] [CrossRef][Green Version]

[24] Umer, Q.; Liu, H.; Illahi, I. CNN-Based Automatic Prioritization of Bug Reports. IEEE Trans. Reliab. 2020, 69, 1341–1354. [Google Scholar] [CrossRef]

[25] Ni, Z.; Li, B.; Sun, X.; Chen, T.; Tang, B.; Shi, X. Analyzing bug fix for automatic bug cause classification. J. Syst. Softw. 2020, 163, 110538. [Google Scholar] [CrossRef]

[26] Aung, T.W.W.; Wan, Y.; Huo, H.; Sui, Y. Multi-triage: A multi-task learning framework for bug triage. J. Syst. Softw. 2022, 184, 111133. [Google Scholar] [CrossRef]

[27] Hirsch, T. Using textual bug reports to predict the fault category of software bugs. Array 2022, 15, 100189. [Google Scholar] [CrossRef]

[28] Wu, H. A spatial–temporal graph neural network framework for automated software bug triaging. Knowl. Based Syst. 2022, 241, 108308. [Google Scholar] [CrossRef]