The Journal of Systems and Software 167 (2020) 110616

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection

Jinfu Chen[a], Patrick Kwaku Kudjo[c,*], Solomon Mensah[b], Selasie Aformaley Brown[c], George Akorfu[d]

[a] School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China
[b] Department of Computer Science, University of Ghana, Legon, Africa
[c] Department of Information Technology, University of Professional Studies, Accra-Ghana, Africa
[d] Department of Business Computing, Wisconsin International University College, Accra-Ghana, Africa

ARTICLE INFO

Article history:
Received 15 September 2019
Revised 3 December 2019
Accepted 24 April 2020
Available online 15 May 2020

Keywords:
Software vulnerability
Classification
Feature selection
Machine learning algorithms
Severity
Term-weighting

ABSTRACT

Vulnerability classification is an important activity in software development and software quality maintenance. A typical vulnerability classification model usually involves a stage of term selection, in which the relevant terms are identified via feature selection. It also involves a stage of term-weighting, in which the document weights for the selected terms are computed, and a stage for classifier learning. Generally, the term frequency-inverse document frequency (TF-IDF) model is the most widely used term-weighting metric for vulnerability classification. However, several issues hinder the effectiveness of the TF-IDF model for document classification. To address this problem, we propose and evaluate a general framework for vulnerability severity classification using the term frequency-inverse gravity moment (TF-IGM). Specifically, we extensively compare the term frequency-inverse gravity moment, term frequency-inverse document frequency, and information gain feature selection using five machine learning algorithms on ten vulnerable software applications containing a total number of 27,248 security vulnerabilities. The experimental result shows that: (i) the TF-IGM model is a promising term weighting metric for vulnerability classification compared to the classical term-weighting metric, (ii) the effectiveness of feature selection on vulnerability classification varies significantly across the studied datasets and (iii) feature selection improves vulnerability classification.

1. Introduction

Empirical research regarding the management of vulnerabilities in software systems has recently been focused on the severity assessment of software vulnerability, with increased attention in exploiting the effectiveness of different machine learning algorithms. This is partly due to the continuous disclosure of critical-severe vulnerabilities and the increasing financial loss caused by these vulnerabilities in our healthcare, energy, defense other critical infrastructure systems. For example, the Microsoft Security Response Center disclosed that there was an overall increased in high-severe vulnerabilities by 41.7% across the industry in the second half of 2015, which accounts for 41.8% of the total largest share for software vulnerabilities in at least three years. Additionally, a report presented by Frost and Sullivan (Vulnerability 2019) in 2018 shows that there was an increase of critical and high severity vulnerabilities from 693 in 2016 to 929 in 2017, with Google project zero been the second in reporting critical and high severity vulnerabilities. More importantly, on August 14, 2019, Intel reported a warning of a high-severe vulnerability in its software that identifies the specification of Intel processors in Windows systems. According to the report, these bugs could have a significant impact on software systems, which include denial of service attacks and information disclosure. Although the firm released an update to address the bugs, an attacker can exploit these vulnerabilities to gain an escalation of privileges on a previously infected machine (O'Donnell, 2019).

In order to protect and ameliorate these vulnerabilities, a large number of organizations such as Symantec Corporations, Atlassian Security Advisories, X-Force, VUPEN Security, Forum of Incident Response and Security Teams have proposed varied techniques to improve the qualitative and quantitative analysis of vulnerabilities. For instance, the National Institute of Standards and Technology interagency report (NISTIR) (National 2020) has five main technical approaches, namely formal methods, system-level security, ad-

* Corresponding author.
E-mail address: kudjo@upsamail.edu.gh (P.K. Kudjo).

https://doi.org/10.1016/j.jss.2020.110616
0164-1212/

dictive software analysis, domain-specific techniques, and moving target defenses and automatic software diversity for reducing security vulnerabilities in software systems. Furthermore, there exist studies that aim to empirically characterize and assess the severity of security vulnerabilities. Spanos and Angelis (2018) proposed a multi-target approach to estimate vulnerability characteristics and severity scores. Their model estimates vulnerability characteristics and subsequently, calculates the vulnerability severity scores from the predicted characteristics. Furthermore, they used three machine learning algorithms, namely random forest, boosting and decision tree to build the proposed vulnerability classification model. The experimental result shows improved classification accuracy. In another study, Khazaei et al. (2016) proposed an automated framework for predicting the severity of vulnerabilities using only the textual description of vulnerabilities. They extracted vulnerability reports offered by the common vulnerability and exposures (CVE) and the common vulnerability scoring system (CVSS) base score from the open-source vulnerability database (OSVDB). Next, they constructed a feature vector to extract terms from the textual description of vulnerability reports. Additionally, they applied the principal component analysis (PCA) and linear discriminant analysis (LDA) to reduce the high dimensional feature space. Finally, they used three machine learning algorithms to predict the severity of software vulnerability. Thus, based on the aforementioned literature, it is evident that the field has witnessed significant research studies (Bozorgi et al., 2010; Han et al., 2017; Dobrovoljc et al., 2017; Zhu et al., 2017) relating to vulnerability assessment. Accordingly, the first and most vital step in building a vulnerability classification model that classifies the severity of software vulnerability is text representation. This phase aims to transform the text information into a vector in the term space so that the document can be recognized and classified by a machine learning algorithm (Lan et al., 2009). The next phase involves feature selection, which selects a subset of features in order to improve the performance of a classification model. More importantly, it also involves a stage of term-weighting, in which document weights for the selected terms are computed. Xuan and Le Quang (2014) define the term-weighting concept as assigning appropriate weights to terms in order to improve classification accuracy. Altınçay and Erenel (2010) described the concept as quantifying the importance of different terms in the selected set. In literature, the concept has been formulated as the product of the term frequency-inverse document frequency (TF-IDF) (Debole and Sebastiani, 2003) and represents the most widely used term-weighting metric for vulnerability classification. For example, Wijayasekara et al. (2014) used the TF-IDF model in stage three of their text mining process to build a binary classification model that classify vulnerabilities as hidden impact or neutral. The experimental result shows that the proposed model is capable of classifying vulnerabilities. However, they observed low classification accuracy based on the studied classifiers. Hence, they made an important recommendation by asking other researchers to improve on the feature extraction process.

Consequently, recent research in the field of automatic text categorization (ATC) (Sabbah et al., 2017; Xuan and Le Quang, 2014) shows that when term-weights are measured via the TF-IDF model, such weights are heavily compromised since the TF-IDF model is vulnerable to biases because the most important terms are typically referred to as noise, thus leading to lower term weights. Similarly, Chen et al. (2016) reiterated that the TF-IDF model is not an effective term-weighting metric for text classification because it ignores the class labels of the training document. Thus, in order to make the term-weighting process more efficient within the context of vulnerability classification, this study proposes and evaluates a general framework for software vulnerability severity classification using the term frequency-inverse gravity moment (TF-IGM) model, a theoretical extension of a concept validated by Chen et al. (2016). The study demonstrates that the TF-IGM model can be incorporated into the proposed framework to improve vulnerability classification. Our approach uses the TF-IGM model to compute the frequency and weight of each term in the textual description of vulnerability reports. Furthermore, we compare the performance of the introduced term-weighting metric with TF-IDF and information gain (IG) feature selection. Our objective is to bridge the gap between the term-weighting metrics applied in automatic text categorization and vulnerability classification. In other words, we aim to provide evidence in line with the global upsurge in the field of software engineering specifically, the broader scope of vulnerability analysis using the term frequency-inverse gravity moment, term frequency-inverse document frequency, and information gain feature selection. In order to demonstrate the effectiveness of our model, we used ten publicly available vulnerability datasets offered by the common vulnerability and exposures (CVE), national vulnerability database (NVD) and Skybox Security (2019). Additionally, the classification results of the suggested models were assessed based on the standard classification evaluation metrics.

This paper makes the following major contributions:

(i) A new and more general vulnerability classification framework within which the term frequency-inverse gravity moment can be validated is proposed in this study.
(ii) We present a comparative study of the term frequency-inverse gravity moment, term frequency-inverse document frequency and information gain feature selection on ten vulnerable software applications.
(iii) This paper evaluates the performance of five machine learning algorithms based on the introduced metrics.
(iv) We contribute new knowledge to security experts to improve vulnerability prioritization.

1.1. Research questions and outcome of the study

This paper explores the following research questions:

RQ1: How effective is the introduced term-weighting scheme?
RQ2: How do feature selection techniques compare to the introduced term-weighting metrics?
RQ3: Which machine learning algorithm performs better in the context of software vulnerability classification?

1.2. Originality and extension

This section outlines the differences in the original study (Kudjo et al., 2019) and the current study using the guidelines for experimental replication proposed by Carver (2010) for researchers and practitioners in the software engineering domain.

1.3. Original study

In the original study (Kudjo et al., 2019), we proposed a new approach that leverages an automatic text categorization concept termed the term frequency-inverse gravity moment to improve vulnerability classification. To do this, we extracted historical vulnerability reports in ten open-source applications and then we used the two term-weighting metrics on the preprocessed datasets. Next, we used three machine learning algorithms, namely decision tree, k-nearest neighbor and random forest with the intention of finding out the most appropriate model for vulnerability classification.

The preliminary results obtained from the original study (Kudjo et al., 2019) are as follows:

i. The result shows that our models achieve 38.1%−95.5% precision and 29.5%−97.4% F-measure across the applications studied.

ii. The classification models presented better precision values for the terms selected based on the term frequency-inverse gravity moment model. For example, our models presented precision values ranging from 64.3%–73.3%, 94.9%–95.5%, 41.6%–67.6%, 59.1%–61.7% and 63.1–66.8% for Mac OS, Linux Kernel, Chrome OS, Windows 10 and Windows 7 respectively.
iii. In terms of recall, the models presented better results for eight applications, namely Mac OS, Linux Kernel, Windows 10, Windows 7, Internet Explorer, Mozilla Firefox and Sea Monkey based on the term frequency-inverse gravity moment model. For instance, the recall values for these applications range from 56.3%–98.7%. The recall values for Microsoft Edge and Chrome OS were between 13.0%–50.3%.
iv. The analysis further suggests statistical significant differences between the two term-weighting metrics, namely the term frequency-inverse gravity moment and term frequency-inverse document frequency models.
v. In summary, the preliminary finding demonstrates the effectiveness and practicality of the introduced term-weighting metric for vulnerability classification.

1.3.1. Current study

In order to recommend the term frequency-inverse gravity moment as an efficient term-weighting metric for vulnerability classification, there is a need to conduct extensive experiments to validate the approach presented in (Kudjo et al., 2019). The following are the changes made and differences noted between the current work and the previous publication.

i. The original study (Kudjo et al., 2019) used ten (10) datasets offered by CVE-NVD among the category of web browsers and operating systems.
ii. The current study validates the proposed approach using the top ten (10) most vulnerable applications offered by CVE-NVD and Skybox Security (Skybox Security 2019). Furthermore, these applications have the highest number of unique confirmed vulnerabilities (Vulnerability 2019).
iii. The original study benchmark the term frequency-inverse gravity moment model with the term frequency-inverse document frequency, because, Cohen (1996) advice that we compare new techniques against the simplest possible alternative and partly due to the fact that it represents the state-of-the-art term-weighting metric for vulnerability classification. The current study performs a comparative study using information gain feature selection. We chose information gain as the benchmark technique because the term frequency-inverse gravity moment model leverages class labels for term-weighting, which is similar to feature selection.
iv. The original study used Yuen's test (Yuen, 1974) to test the statistical significant differences between the two term-weighting metrics. The current study utilizes Yuen's test and Cliff's δ effect size to validate the results of the study.
v. The original study used three machine learning algorithms with different parameter settings. The current study used five machine learning techniques that have been validated in previous studies (Bozorgi et al., 2010; Han et al., 2017; Dobrovoljc et al., 2017).

1.4. Organization of the study

The paper is organized as follows. Section 2 presents the background and definitions that are vital for understanding the study. Section 3 presents the empirical research framework and methodology. The experimental design is presented in Section 4. Section 5 details the research questions and empirical results. Section 6 gives a summary of related work relating to software vulnerability classification and the severity assessment of software vulnerability. Section 7 presents threats to validity. Section 8 summarizes the study and provides future research directions.

2. Preliminaries and problem definition

2.1. Terminology and notations

- **Vulnerability**: is a flaw, weakness or error in a software system that could cause an implicit or explicit failure of confidentiality, integrity, and availability of that system (Kudjo, 2019).
- **Vulnerability assessment**: It is an explicit attempt to discover software vulnerabilities that could be exploited by an adversary. Vulnerability assessment models aim to suggest pragmatic countermeasures to mitigate software vulnerability (Johnston, 2010).
- **Severity**: It is the degree of impact a defect has on the development of a software application being tested (Dobrovoljc et al., 2017; Zhu et al., 2017).
- **Risk score**: It is a technique that captures the characteristics of a software vulnerability and produces numerical scores (values) indicating its severity. The numerical values (i.e. 0.0–10.0) are translated into a qualitative scheme such as low, medium, high and critical to guide vulnerability prioritization (Mell et al., 2006).
- **Critical severity** (9.0–10.0): Vulnerabilities classified as critical allows remote code execution without user interaction.
- **High severity (**7.0–8.9): Vulnerabilities categorized as high are very difficult to exploit and normally results compromise the integrity, confidentiality, and availability of data.
- **Medium severity** (4.0–6.9): Vulnerabilities categorized as a medium requires that an attacker gain access to the local network in order to control the system device.
- **Low severity** (0.1–3.9): Vulnerabilities categorized as low have a less significant impact on software systems.

2.2. Software vulnerability classification

Researchers have used different techniques to classify software vulnerability. A typical vulnerability classification model is categorized into two groups, i.e. binary classification and multi-class classification. In the binary classification type, the extracted vulnerability records are classified as severe or non-severe (Lamkanfi et al., 2011), or based on the severity levels and scores (Han et al., 2017), and essential or non-essential (X. Li et al., 2017). In the multi-class framework, each document *di* is classified as a label in G*, where G* represents the characteristics of security vulnerability {access vector, access complexity, authentication, confidentiality impact, integrity impact, and availability impact}. Generally speaking, there are three main stages for building these models: (i) feature extraction and indexing, (ii) feature selection and (iii) classification. Feature extraction is the process of extracting numerical information from a raw text document. The most widely used feature extraction technique is the bag-of-words approach (Salton et al., 1975). The feature extraction process usually generates several hundred or thousands of features (terms) that increase and degrade the classification accuracy in terms of running time and cost-effectiveness. This problem is often referred to as the "*curse of dimensionality*" in-text classification (Uysal and Gunal, 2014). To address such a challenge, a number of techniques such as pre-processing and feature selection have been proposed to improve the quality of text representation and to develop high-quality classification models. Feature selection involves selecting a subset of features from the original feature sets to improve classification accuracy. At the same time, term-weighting plays a crucial role in vulnerability analysis and

can directly affect classification accuracy. It is the process of assigning document weights to the selected terms. In other words, the weight of a feature shows its importance to the document based on the introduced weighting technique (Lan et al., 2008). The most popular term-weighting metric is the TF-IDF model (Debole and Sebastiani, 2003). It is an information retrieval technique that represents documents and queries as vectors, with associated weights representing the relevance of the keywords or terms in the document (Salton et al., 1975). Furthermore, it measures the relevance of a word by means of its total number of occurrences and by the number of documents. Although the term frequency-inverse document frequency model is simple and easy to implement, previous studies in (Xuan and Le Quang, 2014; Sabbah et al., 2017; Chen et al., 2016) have raised several issues regarding the effectiveness of the model (see Section 2.3). Thus, solely relying on the TF-IDF model tends to under-estimate the classification accuracy of vulnerability classification models. Given the substantial improvement that could be achieved by using other state-of-the-art term-weighting methods, this study set out to investigate the effect of term frequency-inverse gravity moment (Chen et al., 2016) on vulnerability severity classification. Note that (Leopold and Kindermann, 2002) reiterated that choosing an appropriate term-weighting method is more important than tuning a support-vector machines (SVM) for text classification. Thus, this study derives insight from the work in (Chen et al., 2016) by building a vulnerability classification model that uses the term frequency-inverse gravity moment, term frequency-inverse document frequency and information gain feature selection to improve the performance of vulnerability classification.

2.3. Term weighting and the classical term-weighting metric

Term-weighting is a fundamental concept in data mining or text analysis tasks. It is used to convert documents as vectors in the term space and plays a vital role in automatic text categorization (Lan et al., 2005). Recent studies have shown that the performance of a classification model can be affected when term-weights are measured via an inappropriate term-weighting metric. To this aim, several term-weighting methods have been proposed and have successfully been applied in diverse domains of studies such as stock market volatility analysis (Conrad and Loch, 2015), biomedical information retrieval (Wang et al., 2016), and text classification (Chen et al., 2016). The term-weighting methods are generally categorized as supervised-based and unsupervised-based term-weighting methods. The supervised-based term-weighting uses the information on the membership of training documents to calculate the weight of the term. Again, the supervised-based term-weighting techniques use metrics that are available for feature selection problems, and this includes TF-CHI, TF-IG, and TF-GR (Lan et al., 2008). The unsupervised term-weighting methods are usually formulated as the product of the term frequency-inverse document frequency (Debole and Sebastiani, 2003). The term frequency-inverse document frequency model is made of two main elements: term frequency (TF)-inverse document frequency (IDF), TF represents the number of times a term occurs in a document and idf represents the importance of a particular term (Lamkanfi et al., 2011). The TF is defined in Eq. (1)

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

where n_{ij} is the number of occurrences of the term in document d_j, and the denominator is the sum of the number of occurrences of all terms in document d_j. The inverse document frequency is a measure of the general importance of the term obtained by dividing the number of all documents by the number of documents containing the term and then taking the logarithm of that quotient and this is given in Eq. (2).

$$\mathrm{idf}_i = \log \frac{|D|}{|\{d : t_i \in d\}|} \tag{2}$$

where |D| is the total number of documents in the corpus and |{d: $t_i \in$ d}| denotes the number of documents where the term ti appears as given in Eq. (3).

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i \tag{3}$$

Furthermore, there are several ways of representing the TF (term-frequency) and IDF (inverse document frequency). For instance, the TF can be expressed as log (TF), log (TF+1) and log (TF) +1. The IDF can also be expressed as log (N/n_i +1), log (N/n_i) +1 and log (N/n_i −1) where n_i denotes the number of documents that include the ith term. Table 1 depicts the four widely used term frequency factors, which include a binary weight, a normal term frequency, a logarithm of term frequency and an inverse term frequency as suggested in (Lan et al., 2009). The normalized form of TFI-$DF_{i,j}$ as defined in (Fattah, 2015) is given in Eq. (4). Although the term frequency-inverse document frequency is simple and easy to implement, there are some deficiencies pertaining to its effectiveness: (i) it ignores the available class labels for training document, hence the computed weight cannot fully denote the term's importance in text classification (Chen et al., 2016), (ii) the space problem where each document has to be represented using the entire words in the dictionary (Al-Anzi and AbuZeina, 2018), (iii) it ignores the fact that features or terms play different roles in different document categories (Deng et al., 2004), (iv) it does not consider the words which are synonyms of each other and, (v) Lastly, it assumes that the counts of different words provide independent evidence of similarity (Kumari et al., 2016). From a practical perspective, Tian et al. (2015) applied the TF-IDF model (i.e. in stage 2) of their automated bug report classification using bug description and summary. However, the performance of the TF-IDF model was sub-optimal; hence they used only the term frequency, which represents the number of times a word token occurs in a description for the term-weighting. In order to resolve the aforementioned challenges, previous studies (Xuan and Le Quang, 2014; Sabbah et al., 2017; Chen et al., 2016) have proposed varied term-weighting methods, which are extended from the standard TF-IDF and TFIDF formulas. These include the term frequency (TF)-inverse class frequency, term frequency-inverse gravity moment, mutual information, improved Gini index, balanced term weighting scheme, odds ratio (Russo et al., 2019), and Gini index. Thus, this study investigates one of such term-weighting methods which have been proven reliable in the context of text classification.

$$\mathrm{tfidf}_{i,j}^{norm} = \frac{\mathrm{tfidf}_{i,j}}{\max_{i,j}\mathrm{tfidf}_{i,j}} \tag{4}$$

Table 1
Term frequency metrics.

Term Frequency Metrics	Symbol	Description
1.0	Binary	Binary weight equal to 1 for terms present in a vector.
Term frequency	*tf*	Number of times a term occurs in a document.
log(1 + tf)	log *tf*	The logarithm of the term frequency.
$1 - \frac{r}{r+tf}$	*itf*	Inverse term frequency, usually $r = 1$.

2.4. The concept of term frequency-inverse gravity moment

To the best of our knowledge, the term frequency-inverse gravity moment metric was first considered by Chen et al. (2016) in the context of text classification. Their aim was to develop a new term-weighting method to (i) characterize the inter-class distribution and measure the class distinguishing power of a term in the corpus so that terms with stronger class distinguishing power are assigned greater weights than others in text representation, (ii) capture the fine-grained inter-class distribution of a term across different classes of a text so that the calculated weight can reflect the term's importance in text classification, (iii) provide adjustable parameters to achieve optimal performance.

In effect, their focus was to develop a model that can make term-weighting more reasonable and further improve text classification. Hence, the authors (Chen et al., 2016) argued that the weight of a term should be determined mainly by its class distinguishing power, which is embodied primarily by its uneven distribution across different classes and not necessarily the term intra-class distribution. Additionally, they stated that the weight of a term in a document should be determined by its importance in the document and its contribution to the text classification, which correspond respectively to the local and global weighting factors in term-weighting. Using the aforementioned assertion, a theoretical foundation based on proven postulations was established that the term inter-class distribution is more important than the intra-class distribution when determining the class distinguishing power of a term. To this aim, Chen et al. (Chen et al., 2016) proposed a new term-frequency model that incorporates a statistical model called inverse gravity moment to effectively characterize the inter-class distribution. Alternatively, the inverse gravity moment model measures the non-uniformity or concentration level of a term's inter-class distribution, which represents the term's class distinguishing power. The inverse gravity moment model is defined in Eq. (5).

$$igm(t_k) = \frac{f_{k_i}}{\sum_{r=1}^{m} f_{k_r}.r} \tag{5}$$

where igm (t_k) represents the inverse gravity moment of the class interclass distribution of term t_k, and f_{kr} ($r = 1,2\ldots,m$) are the frequencies of t_k's occurring in different classes, which are sorted in descending order with r being the rank. Similarly, the frequency, f_{kr}, refers to the class-specific document frequency (df), which is the number of documents containing the term t_k in the r-th class denoted as df_k. Again, since the first element, $f_{k1,}$ is the maximum in the list of $\{f_{kr} | r = 1,2,\ldots,m\}$, the descending order of the inverse gravity moment is given in Eq. (6). The formula (i.e. Eq. (6)) shows the inverse of the total gravity moment calculated from the normalized frequencies of the term's occurrence in all the individual classes.

$$igm(t_k) = \frac{1}{\sum_{r=1}^{m} \frac{f_{kr}}{\max_{1\le i\le m}(f_{ki})}.r} \tag{6}$$

2.5. Feature selection techniques

Feature selection is an important preprocessing activity in machine learning and data mining and has been widely applied in a plethora of studies such as bug prediction (Hamdy and El-Laithy, 2019), software effort estimation (Silhavy et al., 2018), vulnerability prediction (Kudjo, 2019). Khazaei et al. (2016) defined the concept as selecting a subset of features from the original subset for model training. There are three main types of feature selection techniques, namely filter, wrapper, and embedded methods. The filter methods commonly referred to as the "relevance index", "relevance metric" uses statistical techniques to assign scores to features independently of the machine learning algorithm. Again, the filtering methods score each feature according to a particular feature selection metric and then take the best k features (Forman, 2003). The scoring process involves counting the occurrences of a feature in training positive and negative class training examples separately and then computing a function of these. Document frequency, information gain, gini index, mutual information, odds ratio, and chi-square are some of the approaches or feature selection techniques in this category. The wrapper method employs a search strategy to select the relevant feature and evaluate the obtained subset by using a learning mechanism (Abualigah et al., 2017). Finally, the embedded methods perform variable selection during the process of training and are generally specific to a given learning algorithm. The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, which can be removed without any significant effect on the machine learning models (Bermingham et al., 2015).

2.5.1. Benchmark feature selection method

Information gain (IG) measures the decrease in entropy obtained for a category prediction by knowing the presence or absence of a term in a document. In other words, the metric measures how much a term can be used for classification of information, in order to assess the importance of lexical items for the classification task (Lei, 2020). The formula for the metric is expressed in Eq. (7) as suggested in Lei (Lei, 2020).

$$\begin{aligned} G(D,t) = &-\sum_{i=1}^{m} P(C_i)\log P(C_{i)} \\ &+P(t)\sum_{i=1}^{m} P(C_i|t)\log P(C_i|t) \\ &+P(\bar{t})\sum_{i=1}^{m} P(C_i|\bar{t})\log P(C_i|\bar{t}) \end{aligned} \tag{7}$$

where C denotes a set of document collection, in which there is not the feature t. In this approach, the information gain of each feature is computed, and the features with larger information gain are selected for the classification process.

3. Proposed approach

This section describes the proposed framework for characterizing and classifying the severity of software vulnerability using the two-term weighting metrics and information gain feature selection. Fig. 1. shows a graphical overview of the overall framework of the proposed model. The first phase shows the open-source applications utilized in this study. To ensure optimal performance of the proposed framework, the standard preprocessing techniques were applied to the textual description (i.e. long description of vulnerability) of software vulnerability. The next stages detail the feature extraction, term-weighting, and the feature selection process. The final phase details the classification process involving the five machine learning algorithms.

3.1. Dataset collection

This section describes the vulnerable software applications used for the experimental analysis. Specifically, we used vulnerability reports offered by CVE-NVD. Table 2 describes the studied datasets. Each vulnerability report contains metadata features such as unique ID, summary (i.e. short description usually a keyword), description (i.e. long description), severity scores, product name

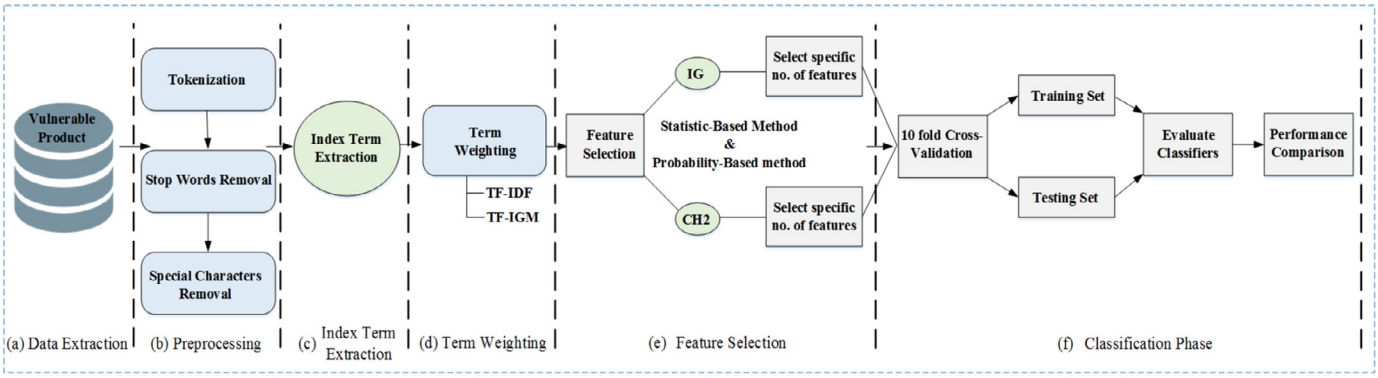

Fig. 1. The overall framework of our experimental procedure.

Table 2
Dataset description.

Applications	Vendor	#Records	Date	% of Exploits	Unique Vul. Types	Description
Windows 10	Microsoft	5624	2005–2019	89	13	An operating system managed by Microsoft as part of its Windows NT family operating systems.
QuickTime	Apple	240	2005–2017	3	7	A multimedia tool for development, storage, and playback.
Oracle Business suite	Oracle	283	2005–2019	–	9	An integrated set of business applications for ERP, CRM, and SCM.
Adobe Flash Player	Adobe	1075	2005–2019	5	9	Software used to view multimedia content, rich internet applications and streaming of audio and video.
Safari	Apple	1020	2005–2019	9	10	An open-source graphical web browser managed by Apple, which is based on the Webkit engine.
Enterprise Linux	Red Hat	536	2005–2019	11	12	An open-source operating system used to execute programs and emerging technologies.
Linux Kernel	Linux	2174	2005–2019	29	8	An open-source monolithic, Unix-like operating system used to execute programs.
Foxit Reader	Foxit Software	298	2008–2019	3	7	Efficient, fast, free and versatile software used to create, view, edit and print pdf files.
Microsoft Office	Microsoft	520	2005–2019	2	8	A suite of desktop productivity applications managed by Microsoft that include Word, Excel, Access.
Google Chrome	Google	1854	2008–2019	2	10	An open-source internet browser developed by Google.
Total	–	27,248	–	153	93	–

Skybox Id	CVE Id	Vendor	Severity	Reporting D...	Last Modified	Description
83857	CVE-2017-18071	Google	Critical	04/02/2018	05/15/2018	Google Android 8.1 and earlier, before 2018-04-05, is prone to a Critical severity unspecified vulnerability in a Qualcomm closed-source component. AKA Android internal bug 68326813.
83815	CVE-2017-13267	Google	Critical	04/02/2018	06/10/2018	Google Android 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2, 8.0, and 8.1, before 2018-04-01, is prone to a critical remote code execution vulnerability. AKA Android internal bug 69479009.
83870	CVE-2017-18136	Google	Critical	04/02/2018	05/13/2018	Google Android 8.1 and earlier, before 2018-04-05, is prone to a High severity unspecified vulnerability in a Qualcomm closed-source component. AKA Android internal bug 68989810.
83871	CVE-2017-18140	Google	Critical	04/02/2018	05/13/2018	Google Android 8.1 and earlier, before 2018-04-05, is prone to a High severity unspecified vulnerability in a Qualcomm closed-source component. AKA Android internal bug 68989811.

Fig. 2. An example of a vulnerability report.

vendor, publish date, update date, etc. Fig. 2. gives an example of a vulnerability description in the Skybox research vulnerability database Skybox Security 2019). Note that we used this example because the organizations build their database using more than 30 sources, which include NVD. In total, we collected 27,248 security vulnerabilities in ten vulnerable software applications, namely Adobe Flash Player, Enterprise Linux, Linux Kernel, Foxit Reader, Safari, Windows 10, Microsoft Office, Oracle Business Suites, Chrome, and QuickTime. These products represent the top ten ((10) most vulnerable applications and contain the highest number of unique confirmed vulnerabilities (Skybox Security 2019; Vulnerability 2019).

This is why we chose to validate the proposed approach using the datasets described in Table 2. Furthermore, CVE-NVD was selected for this study because it is the most widely used vulnerability database for empirical research (Bozorgi et al., 2010; Han et al., 2017; Dobrovoljc et al., 2017; Zhu et al., 2017).

3.2. Data preprocessing

After the data extraction phase, we performed preprocessing to address the data quality issues in vulnerability data. These include special characters, numbers, stop words and punctuation characters which occur frequently in the dataset. These elements have no significant effect on the text mining process, rather removing them from the text data ensures optimal performance of the text mining algorithms (Zhang et al., 2011). The following activities describe the preprocessing phase:

- **Tokenization**: This involves breaking a stream of text into words, phrases, syllabus, or other meaning elements termed tokens (i.e. tokenization). This process reduces textual data by removing unnecessary words for improved performance of the models.
- **Stop Words Removal**: Stop words which occur frequently in our datasets were all removed from the extracted corpus based on the list of stop words in (46).
- **Stemming**: We utilized the porters stemming algorithm) (Porter, 1980) to convert derived words to their base words following a similar procedure by Sharma (2015). Note that we used the WV Tool and ICTCLAS following a similar procedure by Xia et al. (2014) and Chen et al. (2016). The WV tool is a Java Library for statistical language modeling, which is used to create word vector representations for text documents. Its functions include tokenization, word filter (stop words), snowball stemmer and term-frequency. Furthermore, ICTCLAS is a Chinese lexical analysis system for tokenization, keyword extraction, new word identification, entity recognition, and tagging. It is compatible with all operating systems such as Windows, Linux, Android, and IOS and can be invoked by all programming languages such as Java, Python, C, and C#. In summary, after the preprocessing phase, we recorded a total number of 27,092 security vulnerabilities.

3.3. Index term extraction

Keyphrase extraction is an important activity in information retrieval, text mining, and natural language processing applications. There are two main techniques for the identification of keyphrase, that is keyphrase extraction and keyphrase assignments, which is also known as index term assignment (Merrouni et al., 2019). The keyphrase extraction identifies phrases that occur in a document using intrinsic properties such as frequency and length of the phrase. In the term assignment approach, documents are classified according to their content into classes that correspond to the element of vocabulary of terms. Thus, after the preprocessing stage, which involves word/sentence segmentation (i.e. the process of breaking coherent vulnerability descriptions into one word, which transforms the entire textual reports into the smallest semantic unit) stemming, tokenization and stop-words removal, we followed the procedures outlined in (Merrouni et al., 2019; Huang et al., 2019; Witten and Medelyan, 2006) to extract candidate phrases from the textual description of software vulnerability.

3.4. Term-Weight computation

This stage of the text mining process involves term-weighting. Thus, we assign term-weights to the document based on the two term-weighting metrics adopted in this study. Alternatively, we used the term frequency and inverse gravity moment and term frequency-inverse document frequency model to compute the frequency and weight of each term in the textual description of vulnerability reports.

3.5. Feature selection

Generally speaking, feature selection techniques rank features according to a specific metric such that the most useful indicators can be selected to improve classification accuracy. This study applied information gain (IG) feature selection on our problem in order to assess its effect on the proposed classification framework.

4. Experimental design

This section describes the experimental procedure for the proposed classification model. Specifically, this section provides an overview of the dependent and independent variables, machine learning techniques, and evaluation metrics applied in Sections 4.2, 4.3, and 4.4 respectively.

4.1. Experimental setting

We conducted a series of experiments to study the performance of the two term-weighting metrics and compared them with information gain feature selection. We run the experiment on an Intel® Core™ i3-4030 U with 8GB RAM desktop running Windows 7 (64-bit). Again, the study utilized the stratified 10-fold cross-validation approach following a similar procedure in (Han et al., 2017; Dobrovoljc et al., 2017; Zhu et al., 2017).

This approach split the data into 10-folds, where each time, nine parts are used for training purposes and one part is used for validation purposes. The process is repeated ten times and the results obtained after the iterations are averaged and a single result is computed. The stratified 10-fold cross-validation is a standard evaluation setting that is widely applied in software engineering.

4.2. Dependent and independent variables

The dependent variable used in our study is the same as that in (Spanos et al., 2017), which is the severity scores of vulnerabilities. Specifically, we utilized the CVSS metrics to characterize the severity of software vulnerability. The independent variable used in this paper is also the same as those in (Spanos et al., 2017). That is vulnerability reports (i.e. long description) mined from the CVE-NVD. Our approach is based on the assumption that the numerical scores used to describe the severity of vulnerability can potentially help us to distinguish between severe and non-severe vulnerabilities. For example, if the textual description of vulnerability reports explicitly state the following: “allows user-assisted remote attackers to execute arbitrary code” or allows attackers to leverage renderer access to cause a denial of service (application crash), then the hypothesis is that we are dealing with a severe vulnerability.

On the other hand, a description that states “An unspecified function in the JavaScript implementation in Google Chrome creates and exposes a "temporary footprint" when there is a current login to a web site” is perceived as non-severe vulnerability. Note that this has been the focus and practice for researchers when seeking to predict the severity of vulnerabilities (Lamkanfi et al., 2011; Spanos et al., 2017).

4.3. Machine learning algorithms

In order to measure the effectiveness of the term weighing metrics and information gain feature selection on the performance of the classification models, we used five classification algorithms for the experimental analysis. These models were selected because of their effectiveness, fast learning mechanism and their capability of classifying highly multi-dimensional datasets (Wijayasekara et al., 2014). Again, these classifiers represent the most widely used classification algorithms in prior studies (Khazaei et al., 2016; Lamkanfi et al., 2011; Spanos et al., 2017). Note that we used the implementation of the studied classifiers as provided by the Waikato Environment for Knowledge Analysis (Weka 3.8) (Hall et al., 2009) toolkit.

- Random Forest (RF): It is an ensemble-learning algorithm that makes its prediction based on the majority vote of a set of weak decision trees. It is quite stable than the simple decision tree model in terms of prediction accuracy (Khoshgoftaar et al., 2007). We used the number of trees and predictors specified in (M. Feurer et al., 2015).
- K-Nearest Neighbor (kNN): It is a non-parametric learning algorithm that is used for classification and regression problems. The *k*-nearest neighbor first identifies *k* nearest training samples for test instances and then predicts the test instances with the major class among the *k* nearest training samples. It uses a similarity metric such as the Euclidean distance metric to measure the similarity between training instances and test instances. We chose this classifier because it is easy to implement and it is capable of classifying unknown instances by considering its nearest neighbors (Lessmann et al., 2008). The parameters for the *k*-nearest neighbor were varied in the range of 1, 2, 5…17.
- Decision Tree (DT): It is a simple decision tree model that is widely applied for document classification. The algorithm generates a classification decision tree for a given dataset by recursively dividing the available data.

In other words, the algorithm recursively splits data into subsets with rules that maximize information gain (Kohavi and John, 1995). One of the benefits of decision trees is that it is very fast and well suited for categorical data. We utilized the parameter settings in (Kohavi and John, 1995).

- Naïve Bayes (NB): Uses the Bayes theorem, a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data (Lessmann et al., 2008). The default parameter settings were applied.
- Support Vector Machines (SVM): It is a statistical machine learning algorithm that performs well on a wide range of problems. It is usually applied for classification and regression problems. However, it is mostly used in classification problems. SVM was selected because Lessmann et al. (2008) indicated that the algorithm performs well as the Naïve Bayes classifier. We used an appropriate C value and the Radial Basis Function (RBF).
- Multilayer Perceptron (MLP): It is a class of feedforward artificial neural networks model that generates a set of outputs from a set of inputs (Mirjalili, 2019). The following parameter settings were used L (0.1, 0.3, 0.5) M (0.5, 0.6, 0.7, 0.8).
- Logistic Regression (LR): It is a binary classification model or regression-based analysis model used for modeling when the dependent variable is a dichotomy.

Note that at the beginning of the experiment, we conducted a preliminary study to evaluate the performance of seven machine learning algorithms (as discussed above) based on the studied datasets. However, the preliminary finding shows that the best results were obtained when RF, KNN, DT, NB, and SVM are used on the ten vulnerable software applications. Hence, we focused our efforts on these classifiers following a similar approach in (Shin and Williams, 2013; Pang et al., 2017; Lessmann et al., 2008). For instance, Shin and Williams (2013) observed no significant differences between three machine learning algorithms tested on their vulnerability prediction and fault prediction model, hence, they reported only the results of the Logistic Regression. Similarly, Pang et al. (2017) observed no significant differences between the LR, NNN, DT, and RF, thus, they presented only the results of the SVM.

4.4. Evaluation metrics

To compare the performance of the classification algorithms, we used precision, recall, accuracy, and F-measure to compute the overall performance of the classifiers. These metrics represent the most widely used classification evaluation metrics (Bozorgi et al., 2010; Han et al., 2017; Dobrovoljc et al., 2017; Zhu et al., 2017). First, we constructed the confusion matrix and computed the precision and recall measures. That is, we provide evidence of the true positives (TP), false negatives (FN), true negatives (TN) and false positives (FP). The confusion matrix comprising of the aforementioned taxonomies that were used to compute precision, and recall values are defined in Table 3. **Precision:** denotes the number of correctly classified positive instances divided by the number of instances labeled by the classifier as positive. Alternatively, it is the ratio of correctly classified severe vulnerabilities over all the severe vulnerabilities. High precision means less false positives, while low precision means more false positives. **Recall**: represents the number of correctly classified positive instances divided by the number of positive instances in the studied data. That is the ratio of correctly classified severe vulnerabilities over all of the actual severe vulnerabilities. Higher recall means less false negatives, while lower recall means more false negatives. **F-measure:** represents the harmonic mean of precision and recall (Stuckman et al., 2017). According to Davari et al. (2017), a vulnerability classification model with low precision will not be used by security experts because of high false-positives, and also a model with low recall means it does not predict the type of vulnerability accurately. Thus, it is impossible to achieve high recall and high precision at the same time. To this aim, the current study focused on the F-measure for the final judgment.

Accuracy: denotes the overall effectiveness of a classifier. In this research, it represents the number of both severe and non-severe security vulnerabilities that are correctly classified among all the extracted vulnerabilities. The formulas for precision, recall, F-measure and accuracy are given in Eqs. (8), (9), (10) and (11) respectively. Additionally, we compute the area under the curve

Table 3
Confusion matrix.

	Sample size	Classification Outcome	
		Severe	Non-severe
Actual outcome	Severe	True positive (TP)	False negative (FN)
	Non-Severe	False positive (FP)	True negative (TN)

(AUC) values. Wahono and Suryana (Khoshgoftaar et al., 2007) advocated for the AUC because, (i) it separates predictive performance from operating conditions, (ii) represents a general measure of a prediction task, (ii) it has a clear statistical interpretation. The AUC values range between 0–1, where a perfect algorithm has an AUC of 1, and a random classifier has an AUC of 0.5. The interpretation of the AUC scores are as follows: (0.90–1.00-excellent classification), (0.80–0.90- good classification), (0.70–0.80- fair classification), (0.60- 0.70- poor classification), (0.50–0.60- failure). We then compute the Matthews correlation coefficient (MCC) (as defined in 12), which is a regression coefficient that evaluates all the four quadrants of the confusion matrix (M. Feurer et al., 2015). We chose this metric because it is relatively easy to interpret with a range from -1 to $+1$, where a score of 1 indicates a perfect prediction whereas a score of -1 indicates that all the data were wrongly predicted. Finally, we performed paired comparison tests using Yuen's test and Cliff's δ effect size as recommended by Kitchenham et al. (2017). Cliff's δ effect size is used to estimate the magnitude. It is a non-parametric effect size measure that makes no assumption of a particular distribution. Note that Cliff's δ effect size was used to determine how good our models are in finding the actual severe vulnerabilities.

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

$$\mathrm{F-measure} = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{10}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{11}$$

$$\mathrm{MCC} = \frac{tp * tn - fp * fn}{\sqrt{(tp + fnn)(tp + fp)(tn + fp)(tn + fn)}} \tag{12}$$

5. Empirical results

This section presents the experimental results of the study. We focus on the performance of the introduced term-weighting metric and answer the following research questions. In particular, we inspect the performance of the term frequency-inverse gravity moment, term frequency-inverse document frequency, and information gain feature selection.

RQ1: How effective is the introduced term-weighting scheme?

Motivation: To validate the effectiveness of the term frequency-inverse gravity moment model, we need to compare it with the benchmark techniques, namely TF-IDF and information gain.

Approach: We first used different machine learning algorithms, namely decision tree, k-nearest neighbor and random forest on ten applications among the category of web browsers and operating systems. Additionally, an experiment was conducted to further validate the effectiveness and practicality of the proposed approach using the top ten (10) vulnerable software applications. The performance of the classifiers was assessed using the classification evaluation metrics defined in Section 4.4. As indicated earlier, these metrics are the most commonly applied evaluation metrics Finally, in order to provide a more robust and convincing approach, we used the 10-fold cross-validation approach ten times and report the average results.

Results: Table 4–5 shows the performance of the TF-IGM and TF-IDF model based on three learning algorithms. From these tables, we can conclude several points. Fig. 3–5 shows the area under the curve values for the three classification models (i.e. KNN, RF, and DT) based on the open-source applications. Note that in this approach, the area under the curve metric is used to measure the probability that a classifier rank a randomly chosen severe vulnerability higher than a randomly chosen non-severe vulnerability. In analyzing the performance of the classifiers, we noticed the area under the curve values for the terms selected by

Table 4
Classification performance of machine learning models based on TF-IGM.

Datasets	ML TECH.	precision	recall	f-measure	MCC	AUC	accuracy (%)
Mac Os	KNN	**0.733**	**0.688**	**0.71**	**0.454**	0.671	**72.7**
	RF	0.667	0.625	0.655	0.332	0.676	66.6
	DT	0.643	0.563	0.600	0.271	**0.721**	63.6
Linux kernel	KNN	**0.955**	0.994	**0.974**	**0.557**	**0.715**	**95.0**
	RF	0.949	0.994	0.971	0.485	0.694	94.0
	DT	0.951	**0.984**	0.968	0.452	0.628	93.8
Chrome os	KNN	0.558	0.137	0.200	0.246	0.777	91.3
	RF	**0.676**	**0.172**	**0.274**	**0.314**	**0.850**	**91.8**
	DT	0.416	0.151	0.221	0.210	0.652	90.5
windows 10	KNN	**0.617**	0.720	0.664	**0.181**	**0.596**	60.0
	RF	0.613	0.737	0.670	0.179	0.592	**61.0**
	DT	0.591	**0.815**	**0.685**	0.147	0.552	58.8
windows 7	KNN	**0.668**	0.769	0.715	**0.201**	0.610	**63.0**
	RF	0.661	0.767	0.713	0.182	0.609	62.2
	DT	0.631	**0.876**	**0.737**	0.149	**0.616**	62.3
chrome	KNN	0.609	0.628	0.618	0.176	**0.607**	59.0
	RF	**0.610**	0.596	0.603	0.168	0.686	58.5
	DT	0.605	**0.801**	**0.689**	**0.234**	0.583	**61.8**
edge	KNN	0.526	0.170	0.257	0.287	0.750	97.0
	RF	0.287	**0.503**	**0.365**	0.355	**0.885**	94.7
	DT	**0.690**	0.194	0.303	**0.357**	0.816	**97.3**
explorer	KNN	0.75	**0.907**	0.821	0.282	0.710	72.5
	RF	0.760	0.898	**0.823**	**0.312**	0.791	**73.3**
	DT	**0.764**	0.848	0.804	0.284	**0.729**	71.3
firefox	KNN	0.726	**0.795**	0.759	0.304	**0.714**	68.3
	RF	0.734	0.446	**0.938**	**0.293**	0.687	67.0
	DT	**0.749**	0.776	0.762	0.345	0.706	**69.6**
seamonkey	KNN	0.751	0.957	0.842	0.066	**0.630**	73.1
	RF	**0.770**	0.869	0.817	**0.136**	0.598	70.9
	DT	0.745	**0.993**	**0.851**	0.016	0.622	**74.1**

Table 5
Classification performance of machine learning models based on TF-IDF.

Datasets	ML TECH.	precision	recall	f-measure	MCC	AUC	accuracy (%)
Mac os	KNN	**0.667**	**0.625**	0.645	**0.332**	0.678	66.6
	RF	**0.667**	**0.625**	0.645	**0.332**	**0.691**	**66.7**
	DT	0.588	0.525	0.606	0.213	0.585	60.6
Linux kernel	KNN	0.946	0.987	0.966	0.395	0.690	93.5
	RF	**0.955**	0.994	**0.974**	**0.557**	**0.703**	**95.0**
	DT	0.929	**0.991**	0.959	0.074	0.610	92.1
Chrome os	KNN	0.358	**0.340**	**0.349**	0.287	0.644	88.6
	RF	**0.604**	0.219	0.322	**0.330**	**0.828**	**91.7**
	DT	0.381	0.019	0.036	0.068	0.650	90.9
windows 10	KNN	**0.624**	0.715	**0.667**	**0.196**	**0.599**	**60.6**
	RF	0.617	0.725	**0.667**	0.183	0.589	60.1
	DT	0.590	**0.788**	0.678	0.136	0.546	58.2
windows 7	KNN	0.645	0.748	0.692	0.154	0.592	60.5
	RF	**0.656**	0.775	0.711	**0.172**	0.609	**62.9**
	DT	0.635	**0.867**	**0.733**	0.143	**0.614**	61.9
chrome	KNN	0.610	**0.596**	0.603	**0.168**	**0.586**	**58.5**
	RF	**0.612**	**0.596**	0.603	**0.168**	**0.586**	**58.5**
	DT	0.608	0.593	0.601	0.165	0.585	58.3
edge	KNN	0.311	0.274	0.291	0.271	0.628	95.9
	RF	0.201	**0.555**	0.295	0.301	**0.870**	92.0
	DT	**0.510**	0.316	**0.39**	**0.387**	0.838	**97.0**
explorer	KNN	**0.777**	0.820	0.798	**0.303**	0.710	71.2
	RF	0.776	**0.93**	**0.816**	0.205	0.650	70.8
	DT	0.765	0.844	0.802	0.284	**0.728**	71.2
firefox	KNN	0.743	0.713	0.727	0.295	0.669	66.5
	RF	**0.748**	0.723	0.725	0.269	**0.676**	65.7
	DT	0.743	**0.790**	**0.766**	**0.341**	0.648	**69.7**
seamonkey	KNN	0.700	0.841	0.804	0.121	0.584	69.4
	RF	**0.766**	0.869	0.815	**0.118**	**0.601**	70.5
	DT	0.744	**0.985**	**0.848**	0.015	0.551	**73.6**

Bold *figures indicate the best performance in terms of precision, recall and F-measure, AUC, and accuracy across each learner, ML: Machine learning.*

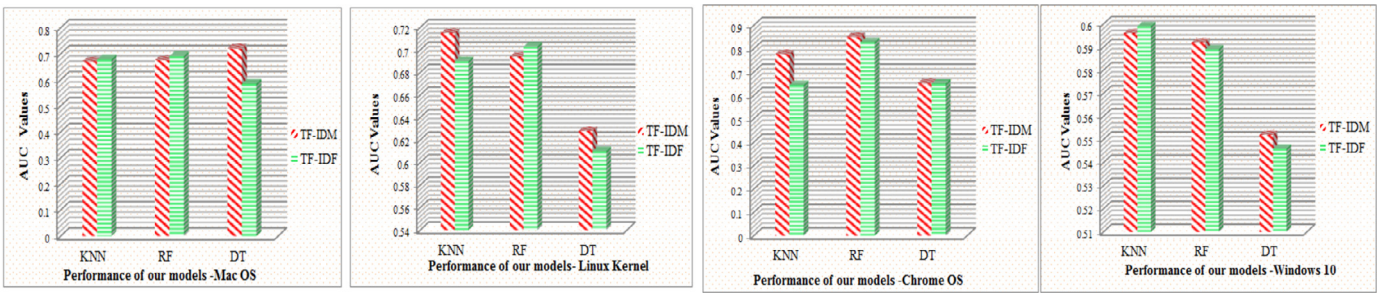

Fig. 3. The area under the curve values for Mac OS, Linux Kernel datasets and Windows 10.

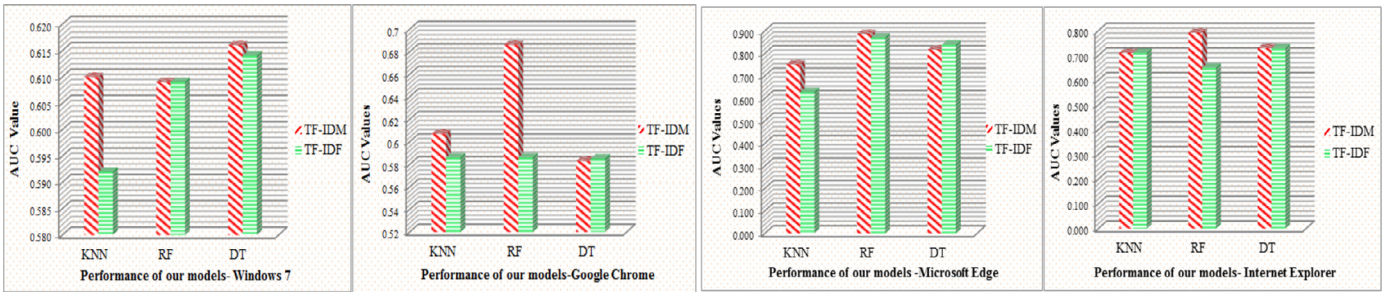

Fig. 4. The area under the curve values for Windows 7, Chrome OS, Edge and Explorer datasets.

the TF-IGM model was better than the terms selected by the TF-IDF model. For example, the KNN, RF and DT model presented AUC values of 0.7, 0.8 and 0.8 respectively for Microsoft Edge. The same discussion can be held for Internet Explorer and Mozilla Firefox. We recorded AUC values of 0.7. However, the values for the other web application systems were low. Thus, based on the classification scheme provided in Section 4.4 regarding the AUC computation, we can confirm that on average, our approach yielded better classification accuracy (i.e. measured in terms of AUC) than the TF-IDF approach. We think that the characteristics of the various datasets studied (data richness) could account for the varied performance achieved by the various classification models. For in-

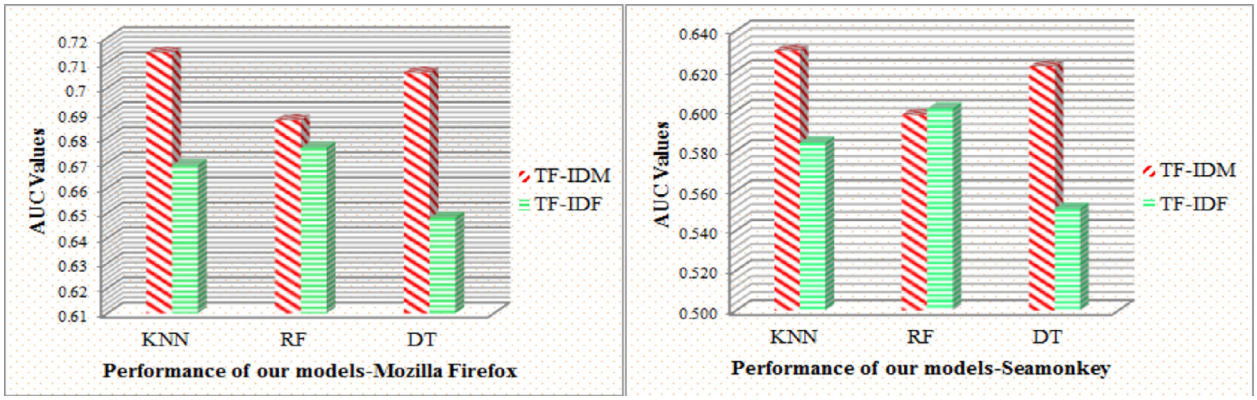

Fig. 5. The area under the curve values for Firefox and SeaMonkey datasets.

stance, the AUC values for Internet Explorer, and Mozilla Firefox, Linux Kernel and Microsoft Edge indicate that the various applications have different meta-features. In terms of precision, the classifiers presented better precision values for the terms selected by the TF-IGM model. Taking the Mac OS, Linux, Chrome OS, Windows 10 and Windows 7 as an example, the classifiers presented precision values ranging from 64.3%–73.3%, 94.9%–95.5%, 41.6%–67.6%, 59.1%–61.7%, and 63.1–66.8% respectively for these applications. On the other hand, the TF-IDF approach presented precision values of 58.8–66.7%, 92.9%–95.5, 38.1%–60.4%, 59%–62.4%, and 63.5–65.5% for Mac OS, Linux, Chrome OS, Windows 10 and Windows 7 respectively. In discussing the result regarding the two categories of applications studied (i.e. the web browsers and operating systems), it can be seen that the classification models presented precision values ranging from 28.7%–77.5% across the entire web applications based on the terms selected by the TF-IGM model and 20.1%–77.7% for the terms selected by TF-IDF model. In relation to the operating systems applications, the models presented precision values ranging from 41.6%–95.5% for the TF-IGM model and 38.1%–95.4% for the TF-IDF model. Hence, a general conclusion that can be derived is that the TF-IGM model is useful and effective than the classical term-weighting metric.

In terms of recall, the machine learning algorithms presented better results for eight applications, namely Mac OS, Linux Kernel, Windows 10, Windows 7, Internet Explorer, Firefox and SeaMonkey for the TF-IGM model. For instance, the recall values for these applications range from 56.3%–98.7%. The recall values for Microsoft Edge and Chrome OS were in the range of 13.0%–50.3%. The recall values for the TF-IDF model were consistent across the datasets. Although we achieved low recall values for some applications based on the two techniques, what really matters for comparing the performance of the models is the F-measure, which measures the harmonic mean of precision and recall. The F-measure values presented in Table 4-5 shows a significant improvement in the F-measure values for eight applications based on the TF-IGM model. Similarly, the classifiers presented better F-measure values for a handful of the applications studied based on the terms selected by the TF-IDF model.

In addition, the decision tree model was nearly as accurate as of the random forest model for the terms selected by the TF-IGM model. The contrast of the obtained results can be explained by the TF-IDF model, the random forest classifier seems to be the most accurate model in terms of F-measure values, followed by the k-nearest neighbor. While observing the results in terms of the MCC values, we can see that the TF-IGM approach presents the best average MCC values of 0.26 and 22.3 for the TF-IDF approach. In conclusion, the preliminary findings suggest that the term frequency-inverse gravity moment model is reliable and can effectively be applied to improve software vulnerability classification. Furthermore, we used Yuen's test (Yuen, 1974) as recommended by Kitchenham et al. (2017) to investigate the statistical significant differences between the term frequency-inverse gravity moment and the term frequency-inverse document frequency for each of the three models across all 10 datasets. We performed a statistical test at a 5% asymptotic significance level and report the *t-values* and *p-values* as depicted in Table 6. The findings demonstrate that there exist statistically significant differences between the term frequency-inverse gravity moment and the term frequency-inverse document frequency across a large number of the applications studied. The existence of statistically significant differences confirms the findings presented in Fig. 3–5 that the TF-IGM model outperforms the TF-IDF based model.

Table 6
Yuen's test for statistical significant difference between TF-IGM and TF-IDF.

Dataset	Model	*t-value*	*p-value*
MAC OS	kNN	0.8872	0.0069*
	RF	−0.0129	0.0400*
	DT	0.5787	0.0756
Linux Kernel	kNN	0.3122	0.0617
	RF	−0.1492	0.0144*
	DT	0.4224	0.0033*
Chrome OS	kNN	−0.0334	0.0410*
	RF	−0.0151	0.9883
	DT	0.4185	0.0480*
Windows 10	kNN	0.9657	0.0440*
	RF	0.9781	0.0281*
	DT	0.0749	0.0418*
Windows 7	kNN	0.2177	0.0321*
	RF	0.9979	0.0027*
	DT	0.9807	0.0248*
Google Chrome	kNN	0.8984	0.0309*
	RF	0.9974	0.0033*
	DT	0.5479	0.0221*
Microsoft Edge	kNN	0.8318	0.2180
	RF	0.8510	0.1928
	DT	0.9403	0.0368*
Internet Explorer	kNN	0.9182	0.0540*
	RF	0.7779	0.2901
	DT	0.9923	0.0099*
Mozilla Firefox	kNN	0.2787	0.7862
	RF	−0.9673	0.0421*
	DT	0.9333	0.0858
Seamonkey	kNN	0.8206	0.0230*
	RF	0.9793	0.0266*
	DT	0.9413	0.0155*

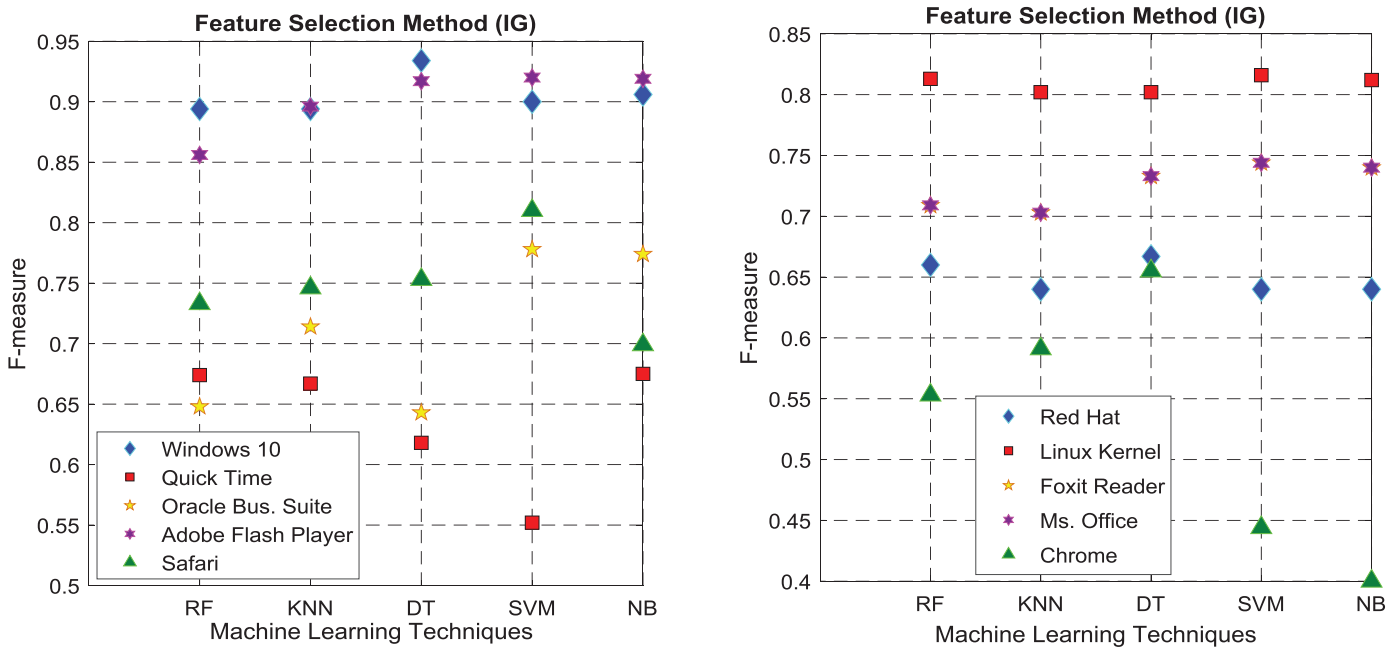

Fig. 6. Performance plot (F-measure) for feature Selection Approach across the 10 datasets.

The experimental result shows that on average, the term frequency-inverse gravity moment model can classify software vulnerability reports with high accuracy than the classical term-weighting model for a wide range of the applications studied. Thus, the preliminary findings make a strong case to further validate the effectiveness of the term frequency-inverse gravity moment model within the broader scope of software vulnerability severity classification.

Having established that term frequency-inverse gravity moment model performed better than the classical term-weighting metric, we conducted an empirical study using different datasets in order to evaluate the effectiveness of the model within the context of software vulnerability classification. In particular, we collected 27,248 security vulnerabilities in 10 vulnerable software products, namely Adobe Flash Player, Enterprise Linux, Linux Kernel, Foxit Reader, Safari, Windows 10, Microsoft Office, Oracle Business Suites, Chrome, and QuickTime. Table 7-8 summaries the results in terms of precision, recall, F-measure, area under the curve, and accuracy when the proposed techniques are applied to the ten (10) datasets. Specifically, Table 7 presents the results of the feature selection approach, whiles Table 8 shows the result for the TF-IGM approach based on the applications.

For the convenience of the reader, this study first presents the results of the empirical comparison in terms of F-measure values, which represent the harmonic mean of precision and recall. The fifth column of Table 7-8 shows the F-measure values of each classifier (RF, KNN, DT, SVM, and NB). For each application, the *best* evaluation and classification scores are shown in *bold*. This implies that the corresponding machine learning algorithm provided the best classification performance in terms of the evaluation metrics discussed in Section 4.4. From Table 7-8, it can be seen that IG train with the five machine learning techniques presented F-measure values ranging from 27.5% to 93.4% and 17.4%–90.9% for the term frequency-inverse gravity moment model. Furthermore, it can be seen from the dot plot in Fig. 6 and Fig. 7 that the feature selection approaches consistently performed better regarding the F-measure values. Again, the feature selection technique outperformed that of the term frequency-inverse gravity moment model on applications such as Adobe Flash Player, Enterprise Linux, Linux Kernel, Oracle Business Suites, QuickTime, Windows 10 and QuickTime. The F-measure values for the following applications, Chrome, and Microsoft Office were low, making the term frequency-inverse gravity moment model better compared to the feature selection approach. Alternatively, the term frequency-inverse gravity moment model outperformed the feature selection approach using the five machine learning algorithms on almost three applications. For instance, we recorded F-measure of 65.4%, 62.7.7%, 59.7%, 44.0%,61.3% for the five models trained using the features/terms selected by IG and 50.5%, 45.1%,27.2%, 35.0%44.0% respectively for RF, KNN, DT, SVM and NB for the Microsoft Office application.

Thus, from Fig. 6 and Fig. 7, we can summary the F-measure values as follows:

i. Across all classification models and datasets, the best F-measure value was achieved when the models are trained using the features/terms selected by IG. This implies that the models performed better when IG is applied to the datasets.
ii. In relation to the proposed framework, the F-measure values were negative (i.e. low values) for two datasets, namely Chrome, and Microsoft Office, We think that the characteristics of the various applications play a major role in relation to the values presented by the classifiers
iii. Red Hat and Microsoft Office provided the least F-measure value of 17.4% and 27.2% for the term frequency-inverse gravity moment and information gain feature selection respectively.
iv. On average, the best F-measure values were obtained when the models were trained on Adobe Flash Player, Enterprise Linux, Linux Kernel, Oracle Business Suites, QuickTime, Windows 10, Safari, and Foxit Reader.
v. In terms of the machine learning algorithms, the SVM, and DT Models turn to be the most accurate classifier (measured in terms of F-measure) based on two approaches. However, NB and RF are the most stable classifiers based on the ten applications studied.
vi. Furthermore, we noticed that our models presented interesting F-measure values for Linux Kernel and Red Hat based on

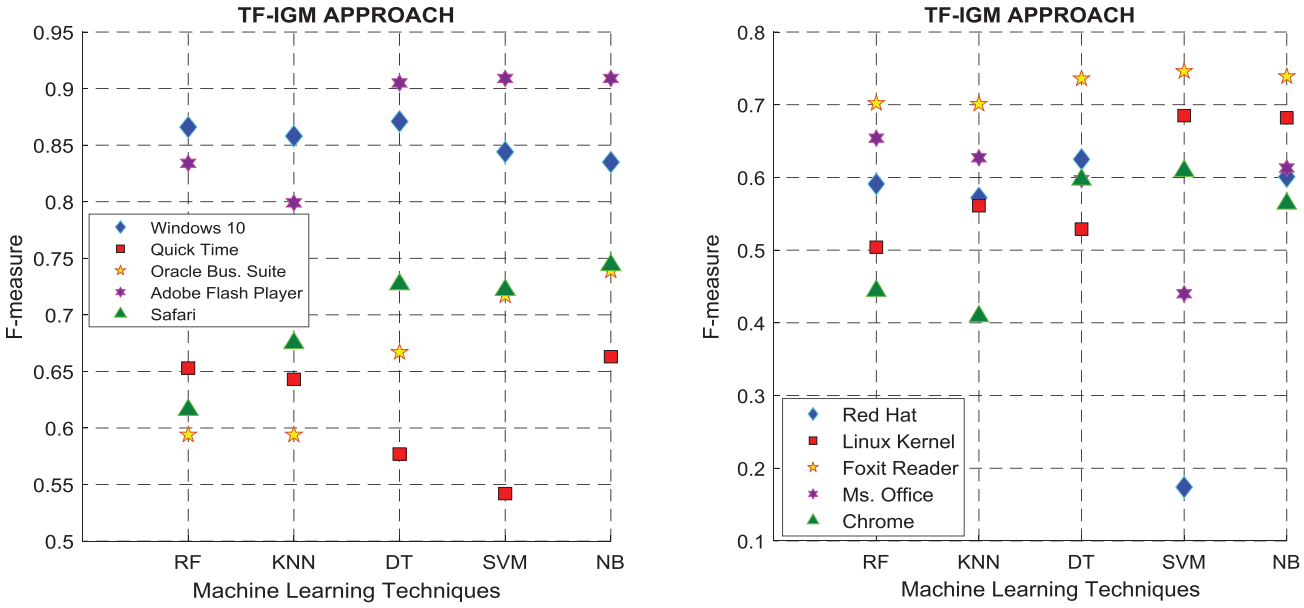

Fig. 7. Performance plot (F-measure) for TF-IGM Approach across the 10 datasets.

the feature selection approach. For instance, the F-measure values for Linux Kernel and Red Hat range from 80.2%–80.16% and 64.4%–66.7%.

vii. Lastly, the findings achieved in this study in terms of the F-measure shows the term frequency-inverse gravity moment improves software vulnerability classification and can effectively be applied for future research studies.

Answer to RQ1: On average, the best F-measure value was achieved when the classification models are trained using the features or terms selected by information gain. Nevertheless, similar to our results of the web browsers and operating systems datasets, the performance of the term frequency-inverse gravity moment model was consistent across the ten vulnerable software products. Thus, we can conclude that the TF-IGM model is an effective term-weighting metric for vulnerability classification

Next, we want to investigate the particular datasets that each of the classifiers predicts in order to show the variations in classifying the severity of software vulnerability. In particular, we present the positive predicted value, which represents the probability that a vulnerability that is classified as severe is actually severe. As indicated earlier, high precision means less false positives, while low precision means more false positives. From Fig. 8 and Fig. 9, we noticed that the feature selection approach presented precision values ranging from 53.1%–85.8% across the applications studied and 45.0%–90.5% for the term frequency-inverse gravity approach. Again, the precision values were consistent across the applications studied. In other words, we achieved high precision values for most of the applications studied. In conclusion, the results in terms of precision show that every record that was classified as severe was an actually severe vulnerability. A similar discussion can be held for the recall values, we recorded better recall values for eight applications using information gain and the term frequency-inverse gravity moment. The value for Microsoft Office and Google Chrome was low. Note that this is often a general trend in classification tasks since it is practically impossible to achieve high recall and high precision at the same time. Thus, this study focused on the F-measure values for the final judgment.

Accuracy is the most widely used evaluation metric for classification tasks. It measures or indicates the correct classification rate. In this study, it represents the number of both severe and non-severe security vulnerabilities that are correctly classified among all the extracted vulnerabilities. It is an intuitive evaluation metric for measuring the performance of classifiers, however, it is generally not an appropriate metric when learning on highly imbalanced datasets. Thus, we presented the accuracy values in this paper to further stress the point that accuracy is not an appropriate metric when dealing with a highly imbalanced vulnerability dataset. For instance, we can see from Table 7-8 that the classification models presented accuracy values ranging from 47.1%–87.6% across the applications studied, and in some cases, we recorded the same accuracy values for the classifiers. Take applications such as Adobe Flash Player, Oracle Business Suite, and Safari as an example, we noticed that the SVM and NB exhibit the same accuracy of 88.3%, and 55.9% for RF, KNN, and SVM regarding the Oracle Business Suite. Similarly, the KNN, NB and DT classifiers presented an accuracy of 59.2% for Safai based on the term frequency-inverse gravity moment approach. A similar discussion can be held when evaluating the performance of the models regarding the feature selection approach. Thus, it is evident that all the machine learning algorithms exhibited different strengths and weaknesses and that they should not be applied "*blindly*" on the studied datasets. This is why we chose not to judge the performance of our models based on the accuracy values presented in Table 7-8. At this stage, the objective of the study is to investigate the superiority of the two techniques. To this do, we performed a robust statistical test using Yuen's test and Cliff's δ effect size for pairwise differences between the two techniques. Cliff's δ effect size was used to estimate the magnitude of the differences between the two approaches. It is a non-parametric effect size measure that makes no assumption of a particular distribution. Table 10 depicts the magnitude scale values applied for the analysis. We conducted a statistical test at a 5% asymptotic significance level using the Yuen's test and Cliff's delta effect size to investigate the existence of statistical and practical differences between the two techniques across all studied datasets. Based on the results presented in Table 9, it can be seen that in all cases of the studied datasets, there exists a significant difference

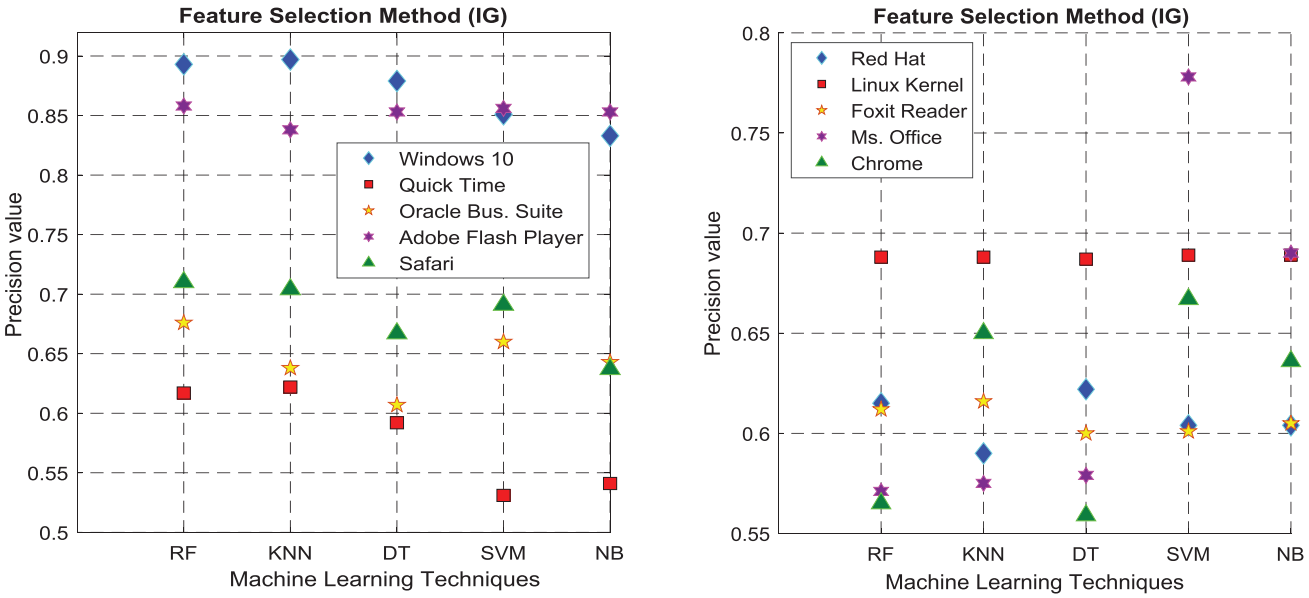

Fig. 8. Performance plot (Precision) for IG Approach across the 10 datasets.

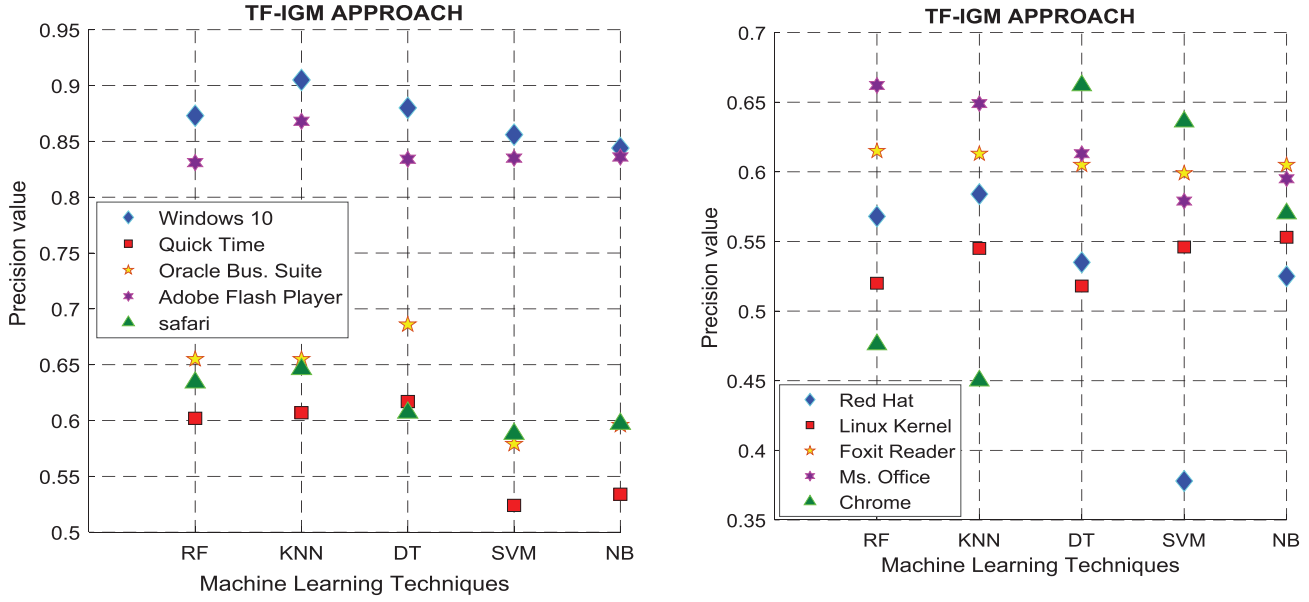

Fig. 9. Performance plot (Precision) for TF-IGM Approach across the 10 datasets.

between information gain and the term frequency-inverse gravity moment. Note that we considered practical significant difference based on Cliff's delta effect size when the effect size value is at least 0.276.

RQ2: How do feature selection techniques compare to the introduced term-weighting metrics?

Motivation: Prior studies in defect classification have shown that the high misclassification rates and low classification accuracy of defect models are often as a result of the number of features utilized prior to the training and classification phase (Ghotra et al., 2017). Similarly, in software vulnerability prediction, Pang et al. (2017) reiterated that the performance of a vulnerability prediction model heavily depends on how well the features are constructed and selected for the statistical machine learning algorithms. Thus, it is important that one is mindful of the irrelevant and redundant features in vulnerability datasets that result in a high feature set (i.e. "curse of dimensionality"). To address this issue, recent studies (Hamdy and El-Laithy, 2019; Kudjo, 2019; Pang et al., 2017) have applied varied feature selection techniques prior to training machine learning models to improve classification accuracy.

Approach: Motivated by the above theory, the current study investigates the effect of information gain feature selection on the proposed vulnerability classification model. Our aim is to compare the performance of the introduced term-weighting metrics with information gain.

Results: It can be seen from 7–8 that the selected features/terms by IG yielded improved classification performance as compared to the term frequency-inverse gravity moment model. However, there were only small differences between the values for some applications, which further reinforces the findings presented in our previous study (Kudjo et al., 2019). Thus, based on the findings presented in this paper, we can conclude that:

i. The effectiveness of feature selection techniques on vulnerability classification varies significantly across the studied datasets

Table 7
Classification performance of machine learning models based on IG.

Datasets	ML TECH.	precision	recall	f-measure	AUC	accuracy (%)
windows 10	RF	0.893	0.895	0.894	0.578	81.0.
	KNN	**0.897**	0.591	0.895	0.496	81.1
	DT	0.879	0.991	**0.934**	**0.613**	**87.6**
	SVM	0.581	**0.954**	0.900	0.558	81.9
	nb	0.833	0.993	0.906	0.605	82.8
quicktime	RF	0.617	0.744	0.674	0.227	**61.7**
	KNN	**0.622**	0.719	0.667	**0.599**	**61.7**
	DT	0.592	0.646	0.618	0.591	57.4
	SVM	0.531	0.575	0.522	0.486	50.4
	nb	0.541	**0.897**	**0.675**	0.555	54.0
oracle business suites	RF	**0.676**	0.622	0.648	0.549	57.6
	KNN	0.638	0.811	0.714	0.491	59.3
	DT	0.607	0.684	0.643	0.612	60.5
	SVM	0.660	0.946	**0.778**	0.548	**66.1**
	nb	0.643	**0.973**	0.774	**0.648**	64.4
adobe flash player	RF	**0.858**	0.855	0.856	0.570	75.5
	KNN	0.838	0.962	0.896	0.251	81.3
	DT	0.853	0.991	0.917	0.699	84.6
	SVM	0.856	0.995	**0.920**	0.742	**85.2**
	nb	0.853	**0.996**	0.919	**0.743**	85.0
Safari	RF	**0.710**	0.789	0.733	0.543	61.6
	KNN	0.704	0.793	0.746	0.560	62.4
	DT	0.677	0.865	0.753	0.564	64.4
	SVM	0.691	**0.977**	**0.810**	**0.630**	**68.0**
	nb	0.637	0.773	0.699	0.599	60.0
red hat	RF	0.615	0.714	0.660	0.598	60.9
	KNN	0.590	0.698	0.640	0.614	58.1
	DT	**0.622**	**0.719**	**0.667**	0.598	**61.7**
	SVM	0.604	0.681	0.640	**0.680**	60.1
	nb	0.607	0.691	0.640	0.654	60.1
linux kernel	RF	0.688	0.994	0.813	0.509	68.5
	KNN	0.688	0.961	0.802	0.523	67.2
	DT	0.687	0.965	0.802	0.525	67.2
	SVM	0.687	**0.999**	**0.816**	**0.530**	**68.8**
	nb	**0.689**	0.990	0.815	0.529	68.4
foxit reader	RF	0.612	0.842	0.709	0.539	58.7
	KNN	**0.616**	0.820	0.703	0.538	58.5
	DT	0.600	0.944	0.733	**0.544**	58.7
	SVM	0.601	**0.976**	**0.744**	0.518	59.7
	nb	0.605	0.951	0.740	0.532	**59.9**
microsoft office	RF	0.571	**0.452**	**0.505**	**0.615**	60.1
	KNN	0.575	0.371	0.451	0.602	59.4
	DT	0.579	0.177	0.272	0.557	57.2
	SVM	**0.778**	0.266	0.350	0.517	62.3
	nb	0.690	0.323	0.440	0.647	**63.0**
google chrome	RF	0.565	0.542	0.553	0.635	61.8
	KNN	0.650	0.564	0.591	0.651	**67.2**
	DT	0.559	**0.792**	**0.655**	0.572	63.6
	SVM	**0.667**	0.333	0.444	**0.770**	63.6
	nb	0.636	0.292	0.400	0.644	61.8

ii. Feature selection procedures improve vulnerability classification.

Answer to RQ2: To conclude the above observations, we found that the impact of feature selection on software vulnerability classification models varies across the studied datasets and also across the machine learning algorithms. The finding further suggests that feature selection improves software vulnerability classification.

RQ3: Which machine learning algorithm performs better in the context of software vulnerability classification?

Motivation: Machine learning techniques have recently made great strides in several domains of studies and widely applied by enterprises such as RapidMiner.com, DataRobot.com, Amazon Machine Learning Google's Prediction API (M. Feurer et al., 2015). To identify and classify the severity of security vulnerability, this

Table 8
Classification performance of machine learning models based on TF-IGM.

Datasets	ML TECH.	precision	recall	f-measure	AUC	accuracy (%)
windows 10	RF	0.873	0.866	0.866	**0.870**	77.2
	KNN	**0.905**	0.858	0.858	0.544	**75.7**
	DT	0.880	**0.871**	**0.871**	0.505	77.5
	SVM	0.856	0.844	0.844	0.518	73.8
	nb	0.844	0.835	0.835	0.534	72.8
quicktime	RF	0.602	0.712	0.653	**0.615**	**60.5**
	KNN	0.607	0.684	0.643	0.612	**60.5**
	DT	**0.617**	0.542	0.577	0.593	58.6
	SVM	0.524	0.56	0.542	0.499	50.6
	nb	0.534	**0.877**	**0.663**	0.554	53.7
oracle business suites	RF	0.655	0.543	0.594	0.571	55.9
	KNN	0.655	0.543	0.594	0.565	55.9
	DT	**0.686**	0.649	0.667	**0.573**	59.3
	SVM	0.579	0.943	0.717	0.36	55.9
	nb	0.596	**0.971**	**0.739**	0.52	59.3
adobe flash player	RF	0.529	0.838	0.834	0.529	72.2
	KNN	0.540	0.741	0.799	0.54	68.2
	DT	0.647	0.990	0.905	0.647	82.6
	SVM	**0.698**	0.997	**0.909**	**0.698**	83.3
	nb	0.697	**0.996**	**0.909**	0.697	83.3
safari	RF	0.634	0.600	0.616	0.492	55.2
	KNN	**0.646**	0.707	0.675	0.537	59.2
	DT	0.607	0.907	0.727	0.531	59.2
	SVM	0.588	0.933	0.722	**0.571**	56.8
	nb	0.597	**0.987**	**0.744**	0.544	59.2
red hat	RF	0.568	0.617	0.591	0.629	58.3
	KNN	**0.584**	0.560	0.572	**0.63**	58.9
	DT	0.535	**0.751**	**0.625**	0.554	55.9
	SVM	0.378	0.113	0.174	0.174	47.5
	nb	0.525	0.702	0.601	0.601	54.3
linux kernel	RF	0.520	0.489	0.504	0.459	47.1
	KNN	0.545	0.578	0.561	0.485	50.4
	DT	0.518	0.541	0.529	0.485	47.1
	SVM	0.546	**0.919**	**0.685**	**0.512**	53.6
	nb	**0.553**	0.889	0.682	0.462	54.4
foxit reader	RF	**0.615**	0.824	0.702	0.542	58.0
	KNN	0.613	0.818	0.701	0.539	58.2
	DT	0.605	0.941	0.736	0.550	59.6
	SVM	0.599	0.990	**0.746**	0.536	59.7
	nb	0.605	**0.949**	0.739	**0.653**	59.9
microsoft office	RF	**0.662**	**0.646**	**0.654**	**0.626**	**60.8**
	KNN	0.649	0.608	0.627	0.612	58.6
	DT	0.613	0.582	0.597	0.545	55.0
	SVM	0.579	0.355	0.44	0.618	59.4
	nb	0.595	0.633	0.613	0.605	54.3
google chrome	RF	0.476	0.417	0.444	0.609	54.5
	KNN	0.450	0.375	0.409	0.622	52.7
	DT	**0.662**	0.544	0.597	0.631	57.9
	SVM	0.636	**0.583**	**0.609**	**0.659**	67.2
	nb	0.570	0.558	0.564	0.628	57.7

Table 9
Statistical significant difference between information gain and tf-igm.

Dataset	Yuen's test		*Cliff's δ effect size*
	t-value	*p-value*	
Windows 10	5.0623	0.0159	0.7425*
QuickTime	0.6425	0.0432	0.5385*
Oracle	1.1713	0.0064	0.2762*
Adobe Flash Player	1.1664	0.0412	0.2770*
Safari	1.7488	0.0465	0.3185*
Red Hat	1.7013	0.0392	0.3327*
Linux Kernel	6.7548	0.0128	0.6385*
Foxit Reader	0.3278	0.0207	0.7527*
Ms Office	3.3907	0.0116	0.6430*
Chrome	0.1981	0.0042	0.8486*

Statistical Significance: $p<0.05$; Practical Significance: $*\delta \geq 0.276$.

Table 10
Magnitude thresholds for cliff's delta effect size.

Magnitude Thresholds	Effect Size
Negligible effect size	$\delta < 0.112$
Small effect size	$0.112 \leq \delta < 0.276$
Medium effect size	$0.276 \leq \delta < 0.427$
Large effect size	$\delta \geq 0.427$

study evaluated the effectiveness of seven machine learning algorithms provided by the Waikato Environment for Knowledge Analysis (Weka 3.8) (Hall et al., 2009) toolkit with an objective of investigating the most appropriate model for vulnerability classification.

Approach: Inspired by the work by Lessmann et al. (2008) in which the authors performed a comparative study of 22 classifiers on ten publicly available defect prediction datasets from the NASA metrics data repository, we selected five machine learning algorithm after the evaluation phase for the classification process.

Results: The results in Table 7–8 shows that the KNN, RF, and DT turn to be the most accurate classifiers based on the two term-weighting metrics. In comparing the performance of IG and TF-IGM in Table 7–8, we noticed that SVM and DT turn to be the most accurate classifier (measured in terms of F-measure) based on two approaches.

Answer to RQ3: The experimental analysis shows that Naïve Bayes and Random Forest are the most stable classifiers based on the ten applications studied. However, the Decision tree model is the most accurate classifier for the approach presented in this study.

6. Related work

This section presents previous research studies in relation to software vulnerability analysis. More importantly, our survey of related work focuses on two categories of software vulnerability assessment, that is, software vulnerability classification and the severity assessment of software vulnerability.

6.1. Software vulnerability classification

Russo et al. (2019) published a study in this journal in 2019 in which they presented a vulnerability classification model termed CVErizer, a model that automatically generates summaries of daily posted security vulnerabilities and categorizes them according to a taxonomy. To do this, they used a set of 3369 pre-labeled CVE records and performed an end-to-end evaluation of the CVErizer summaries using fifteen cybersecurity masters students and four professional security experts. The experimental result shows that the model can automatically classify vulnerabilities with an accuracy of 81%.

Huang et al. (2019) proposed an automated software vulnerability classification model based on deep neural networks (DNN). Their model uses the term-frequency-inverse document frequency, information gain, and the deep neural network model. The TF-IDF model is used to compute the frequency and weight of each word in the textual descriptions of software vulnerability, and information gain is applied to select an optimal set of feature words or terms. Finally, the deep neural network model is used to automatically classify the extracted vulnerability reports. The result shows improved classification accuracy compared to the benchmark techniques. Garg et al., 2019 presented a theoretical vulnerability classification approach based on different technical parameters such as frequency, susceptibility, exploits, and threats types. Their model seeks to resolve the challenges of existing vulnerability classification schemes that examine a single cause-effect of security vulnerabilities in a domain-specific application. Thus, the proposed model was developed based on a comprehensive review of vulnerabilities in different domains. Li et al. (X. Li et al., 2017) proposed a vulnerability classification framework, which is based on vulnerability characteristics, accumulation of errors, strict timing requirement, and complex interactions between environment and software. They examined seven different attack patterns and possible mapping between vulnerability types and attack patterns. The above models were used to analyze security vulnerabilities and their corresponding attacks as reported by Google project zero. The authors observed that Madel security vulnerabilities, especially non-aging related Mandel vulnerabilities account for the largest share of all classified vulnerabilities. Furthermore, they noticed that it takes more time and complex strategies to detect and fix non-aging related Mandel security vulnerabilities.

Sabetta and Bezzi (2018) used different machine learning techniques to analyze source code repositories and to automatically identify commits that are security-relevant (i.e. that are likely to fix a vulnerability). To achieve this goal, they applied the natural language processing techniques to preprocessing the source-code changes documents and then classify them using the standard classification models. The experimental results showed improved classification accuracy with precision and recall values of 80% and 43% respectively. Davari et al. (2017) proposed an automatic vulnerability classification framework based on conditions that activate vulnerabilities with the goal of helping security experts and software developers to design appropriate corrective countermeasures. The result demonstrates that the decision tree model can identify the category of unseen security vulnerabilities with 69% F-measure. Lastly, Wijayasekara et al. (2014)) proposed a hidden impact vulnerability classification approach using text mining techniques. They used the bag-of-words representation in stage one to extract terms, where they store the number of times each unique keyword (term) is found in vulnerability reports. Three machine learning algorithms, namely Naïve Bayes, Naïve Bayes Multinomial and Decision Tree were used to classify the bug reports. The experimental result shows that the models can accurately classify hidden impact vulnerabilities.

6.2. Software vulnerability severity assessment

The Symantec Security Response Center defines the severity of software vulnerability as "the impact a defect has on software systems when vulnerabilities are exploited". In other words, it is the risk associated with software vulnerability when exploited. Technically, the severity prediction of a software vulnerability is an important concept in vulnerability assessment mainly because it is used to (i) communicate the impact of vulnerabilities in software systems (ii) quantify the economic impact (i.e. in terms of monetary or reputation losses) of security vulnerabilities, (iii) determine how quickly a vulnerability should be resolved when discovered and (iv) prioritizing testing efforts towards the high severe security vulnerabilities, (iv) better understand software vulnerabilities in their historical context (Munaiah and Meneely, 2016). To this aim, the field has witnessed significant research studies. This section specifically details the seminal studies regarding the severity assessment of security vulnerabilities.

Sahin and Tosun (2019) replicated a prior study by Han et al. (2017), which aims to predict the severity of software vulnerability using only the textual description and convolutional neural networks (CNNs). The authors followed the feature extraction process in (Han et al., 2017), which is the word embeddings. Thus, in addition to the CNN model, the authors used the long short term mem-

ory (LSTM) and extreme gradient boosting (XGBoost) for predicting the severity levels and scores of software vulnerability. They evaluated the approach using a total number of 82,974 security vulnerabilities mined from NVD. Again, they conducted two experiments made of two tasks, namely classification for predicting the severity levels of security vulnerability and regression model for predicting the severity scores of vulnerability. The CNN and LSTM model yielded similar results regarding the F-measure values in predicting the severity levels. In relation to the regression model, they recorded MAPE values of 16.14%, 17.03%, 18.91% for LSTM, CNN, and XGBoost respectively. Zou et al. (2019) proposed an automatic vulnerability severity assessment model called automatic common vulnerability scoring system (AutoCVSS), which is based on the attack process. The proposed approach leverages the characteristics of vulnerabilities in order to model the CVSS base metrics. The vulnerability severity scores obtained in the analysis show that the AutoCVSS model is effective and can automatically produce the scores in accordance with those assessed manually by security experts.

Wang et al. (2019) proposed a vulnerability severity prediction model based on text mining techniques. They applied the natural language preprocessing techniques to the textual description of vulnerability reports, and then they applied the principal component analysis (PCA) to gather the spare features. Finally, the XGBoost model was used to predict the severity levels of software vulnerability. The finding indicates that the proposed model achieved improved prediction accuracy compared to the traditional machine learning algorithms. In another related study, Zhu et al. (2017) argued that the severity assessment of software vulnerability is essential because the same number of vulnerabilities might cause different degrees of damage since they have different access scope and attack approaches. To this aim, they presented a vulnerability severity model in which the probability and severity of vulnerability occurrences are determined by the logistic function and binomial distribution respectively. Spanos et al. (2017) investigated the extent to which vulnerability severity can be inferred directly from vulnerability description. They applied data-mining techniques (e.g. stop words removal, stemming, and removal of punctuation) on the textual description of vulnerability reports in order to build a model that predicts vulnerability severity. Similarly, Singh and Joshi (2016) presented a risk evaluation model that uses NVD and CVSS metrics. They focus on the part of the CVSS scores that deal with the duration of exploitation (temporal vector) and the security level of the organization (environmental vector). In other words, the authors wanted to examine the impact of the temporal vector and the environmental vector on the CVSS scores. Thus, their model estimates a security risk level from vulnerability information as a combination of a period of exploitation and frequency of occurrence in order to assess the impact derived from the CVSS metric. Joh and Malaiya (2011) presented an approach that assesses security risk measures using vulnerability lifecycle and the CVSS metric. They used a stochastic model for the vulnerability lifecycle and the CVSS metrics. Additionally, they incorporated vulnerability discovery and potential zero-day attacks into the model in order to evaluate the potential risk of software systems. Holm et al. (2012) performed an empirical analysis of system-level vulnerability metrics using real data obtained from an International Cyber Defense exercise involving over 100 participants. The data was collected by analyzing network traffic logs, attacker logs, observer logs, and network vulnerabilities. They presented a statistical analysis by showing how 18 security estimation metrics based on the CVSS metric correlate with the time-to-comprise of 34 successful attackers. The result shows that security modeling with CVSS data alone does not accurately depict the time-to-compromise of a system. The finding further shows that metrics employing more CVSS data are more correlated with time-to-compromise.

Younis et al. (2016) performed an empirical study by investigating the extent to which the independent scales used by the vulnerability reward programs (VRP) correlate with the CVSS base score. They used Mozilla Firefox and Google Chrome to evaluate the proposed approach. The result shows that the CVSS score is a useful metric for prioritizing vulnerabilities. Finifter et al. (2013) conducted a similar study by examining the characteristics of two well-known rewards program. They used Google Chrome and Mozilla Firefox as the case study programs. The finding shows that both programs are economically efficient as compared to the cost of hiring a full-time security researcher. They further observed that Firefox has far more critical-severe vulnerabilities as compared to Google Chrome. Feutrill et al. (2018) examined the effect of the CVSS metric on the time until a proof-of-concept exploit is developed. They used the NVD and the Exploit Database to show how the CVSS metrics can provide useful insight into the exploit delays. The result shows that there is a rapid decline in the median vulnerability exploit time. They also show that the median time lag associated with populating CVSS metrics will increase from a single day prior to 2017 to 19 days in 2018. Cheng et al. (2012) presented a novel framework to resolve the challenges associated with aggregating the CVSS score. That is the loss of useful semantics in handling dependency relationships and the loss of semantics from different aspects. Thus, instead of taking each base score as an input, their model drills down to the base metric level where dependency relationships have well-defined semantics. Next, the model interprets and aggregates the base metrics from three different aspects in order to preserve corresponding semantics of the individual scores. They used the Boston University Representative Internet Topology gEnerator (BRITE) to generate simulated network topologies, where vulnerability data are fed into the generated network topologies to obtain network configurations. Finally, attack graphs are generated from the configurations using the aforementioned procedures.

To summarize the main difference between the proposed framework and previous studies, we observe the following points. (i) previous studies have proposed a variety of data mining and machine learning algorithms as an important paradigm for vulnerability classification, (ii) they applied real-world data to estimate the CVSS metrics, (iii) they used the NVD-CVE datasets for characterizing and classifying vulnerabilities, (iv) again, different feature selection techniques such as information gain (IG) and principal component analysis (PCA) have been applied to improve vulnerability classification and lastly (v) various stochastic models have been proposed to evaluate the lifecycle of vulnerabilities and the CVSS metrics. Nevertheless, one limitation that we address in this study is that all the approaches do not go beyond considering how to use any of the modified term-weighting schemes to improve vulnerability classification. For instance, Huang et al. (2019) used the classical term-weighting metric (i.e. TF-IDF model) to compute the frequency and weight of each word in the bug reports, and information gain feature selection was applied to select an optimal set of feature words or terms. However, they did not extensively compare the performance of different machine learning algorithms using the term frequency-inverse gravity moment model, term frequency-inverse document frequency and information gain feature selection. Thus, the objective of the current study is to improve the body of knowledge relating to vulnerability classification based on the TF-IGM, TF-IDF, and IG feature selection. Furthermore, we compare and assess the performance of five machine learning algorithms and validated the results using appropriate statistical tests in order to determine whether there exist statistical significant differences between the various approaches presented

in this study. Finally, we have provided a completely generalizable vulnerability classification framework and repeatable experimental results in this study.

7. Threats to validity

As an empirical study, there are several potential constraints. We, therefore, discuss below the internal and external threats to the validity of the study. The threat to internal validity examines factors that may impact the result of our study. This study obtained vulnerability reports offered by CVE-NVD in ten publicly available vulnerability databases. This has been the process for vulnerability severity (Bozorgi et al., 2010; Han et al., 2017; Dobrovoljc et al., 2017; Zhu et al., 2017) but has a limitation that vulnerabilities not recorded within the period studied cannot be collected. Further research is required to improve the accuracy of the vulnerability collections process from these repositories. Additionally, this study utilized the standard vulnerability scoring metric (CVSS) to characterize the severity of software vulnerability. Nevertheless, the recently proposed severity metrics such as the vulnerability rating and scoring system (VRSS) and Weighted Impact Vulnerability Scoring System (WIVSS) suggests that further research is required to improve the severity assessment process. The threat to External validity concerns the generalization of the empirical results.

Our study used ten publicly available vulnerability datasets and five classification models which could be a source of bias to the results obtained in this study. However, the sizes of the datasets were large enough to enable us to conduct a thorough experiment. Additionally, the results from the study of the ten datasets to assess the severity of software vulnerability cannot be generalized to other datasets, especially proprietary applications because we used only open-source applications. Nevertheless, we have outlined a detailed description of the experimental setup, which will be useful for replicating our experiment using commercial software applications. We also acknowledge the availability of diverse software vulnerability classification models which could have been considered but we argue that the five models selected represent the most widely used models for vulnerability classification, thus these learning algorithms are representative of all the several learning algorithms that exist in literature. Lastly, we used precision, recall, and F-measure, and AUC to measure the performance of the model. This has been the process in previous studies (Han et al., 2017; Dobrovoljc et al., 2017; Zhu et al., 2017). However, there may be other essential metrics and representations demonstrating how well a classifier performs. Further research is required to analyze the results based on other performance evaluation metrics.

8. Conclusions and future research

In this study, we proposed an automatic vulnerability classification approach using the term frequency-inverse gravity moment. The framework involves evaluation and classification. In the evaluation phase, different machine learning algorithms are evaluated on ten vulnerable software applications. Then, in the classification phase, the best machine learning algorithms, namely RF, KNN, DT, NB, and SVM are used to build the proposed classification with all the extracted historical vulnerability reports. Finally, the classification results of the suggested models are assessed using the standard evaluation metrics. Again, we performed a comparative study using the classical term-weighting metric (i.e. term frequency-inverse document frequency) and the most widely used feature selection technique (i.e. information gain). The experimental result shows that the term frequency-inverse gravity moment model is a promising metric for vulnerability report classification compared to the classical term-weighting metric. Regarding the effectiveness of the term frequency-inverse gravity moment and the feature selection technique, the findings suggest that feature selection procedures greatly improve vulnerability classification. In addition, we observed that the effectiveness of feature selection procedures on vulnerability classification models varies significantly across the studied datasets. The findings of this research can be used by researchers and security experts to prioritize software vulnerabilities that demands urgent attention or bugs that need to be fixed in software systems. In future, we plan to integrate our approach to other vulnerability severity assessment schemes and open-source vulnerability databases to make it easier for security experts and software developers to adopt the findings presented in this study. Furthermore, we plan to investigate the impact of the wrapper and embedded feature selection techniques on software vulnerability classification models.

Declaration of Competing Interest

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

Acknowledgement

This work is partly supported by National Natural Science Foundation of China (NSFC U1836116, 6170022430 and 61872167), the Project of Jiangsu Provincial Six Talent Peaks (Grant number: XXJS-016), The Postdoctoral Science Foundation of China 1112019T120399 and the Graduate Research Innovation Project of Jiangsu Province (Grant numbers KYCX17 1807).

References

ABC http://dev.mysql.com/doc/refman/5.1/en/fulltext-stopwords.html.
Abualigah, L.M., Khader, A.T., Hanandeh, E.S., 2017. A new feature selection method to improve the document clustering using particle swarm optimization algorithm. J Comput Sci.
Al-Anzi, F.S., AbuZeina, D., 2018. Beyond vector space model for hierarchical Arabic text classification: a Markov chain approach. Inf Process Manag 54, 105–115.
Altınçay, H., Erenel, Z., 2010. Analytical evaluation of term weighting schemes for text categorization, 31, pp. 1310–1323.
Bermingham, M.L., Pong-Wong, R., Spiliopoulou, A., et al., 2015. Application of high-dimensional feature selection: evaluation for genomic prediction in man. Sci Rep 5, 10312.
Bozorgi, M., Saul, L.K., Savage, S., et al., 2010. Beyond Heuristics : learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 105–114.
Carver, J.C., 2010. Towards reporting guidelines for experimental replications: a proposal. In: 1st International Workshop on Replication in Empirical Software Engineering, 1, pp. 1–4.
Chen, K., Zhang, Z., Long, J., et al., 2016. Turning from TF-IDF to TF-IGM for term weighting in text classification. Expert Syst. App.l 66, 245–260.
Cheng, P., Wang, L., Jajodia, S., et al., 2012. Aggregating CVSS base scores for semantics-rich network security metrics. In: Proceedings of the 31st IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 31–40.
Cohen, P.R., 1996. Empirical methods for artificial intelligence. IEEE Intell. Syst. 6, 88.
Conrad, C., Loch, K., 2015. Anticipating long term stock market volatility. Journal of Applied Econometrics 30 (7), 1090–1114.
Davari, M., Zulkernine, M., Jaafar, F., 2017. An automatic software vulnerability classification framework. In: IEEE International Conference on Software Security and Assurance (ICSSA), pp. 44–49.
Debole, F., Sebastiani, F., 2003. Supervised term weighting for automated text categorization. In: Proceedings of the ACM Symposium on Applied Computing, pp. 784–788.
Deng, Z.-H., Tang, S.-W., Yang, D.-Q., et al., 2004. A comparative study on feature weight in text categorization. In: *Asia-Pacific Web Conference*„ pp. 588–597.
Dobrovoljc, A., Trček, D., Likar, B., 2017. Predicting exploitations of information systems vulnerabilities through attackers characteristics. IEEE Access 5, 26063–26075.
Fattah, M.A., 2015. New term weighting schemes with combination of multiple classifiers for sentiment analysis. Neurocomputing 167, 434–442.
Feurer, M., Klein, A., Eggensperger, K., et al., 2015a. Efficient and robust automated machine learning. Neural Information Process. Syst. 2962–2970.

Feurer, M., Klein, A., Eggensperger, K., et al., 2015b. Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems, pp. 2962–2970.
Feutrill, A., Ranathunga, D., Yarom, Y., et al., 2018. The effect of common vulnerability scoring system metrics on vulnerability exploit delay. In: Proceedings of the 6th IEEE International Symposium on Computing and Networking (CANDAR), pp. 1–10.
Finifter, M., Akhawe, D., Wagner, D., 2013. An empirical study of vulnerability rewards programs. In: InUSENIX Security Symposium 2013, pp. 273–288.
Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 3, 1289–1305.
Garg, S., Singh, R.K., Mohapatra, A.K., 2019. Analysis of software vulnerability classification based on different technical parameters. Information Security Journal: A Global Perspective 1–19.
Ghotra, B., McIntosh, S., Hassan, A.E., 2017. A large-scale study of the impact of feature selection techniques on defect classification models. In: Proceedings of the 14th IEEE/ACM International Conference on Mining Software Repositories (MSR), pp. 146–157.
Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The WEKA data mining software: an update. ACM SIGKDD Explorations Newslett. 11, 10–18.
Hamdy, A., El-Laithy, A., 2019. SMOTE and Feature Selection for More Effective Bug Severity Prediction. International Journal of Software Engineering and Knowledge Engineering 6, 897–919.
Han, Z., Li, X., Xing, Z., et al., 2017. Learning to predict severity of software vulnerability using only vulnerability description. In: IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 125–136.
Holm, H., Ekstedt, M., Andersson, D., 2012. Empirical analysis of system-level vulnerability metrics through actual attacks. IEEE Trans. Dependable Secure Comput. 9 (6), 825–837.
Huang, G., Li, Y., Wang, Q., Ren, J., Cheng, Y., Zhao, X., 2019. Automatic Classification Method for Software Vulnerability Based on Deep Neural Network. IEEE Access 7, 28291–28298.
Joh, H., Malaiya, Y.K., 2011. Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics. In: International Conference on Security and Management (SAM), pp. 10–16.
Johnston, R.G., 2010. Being vulnerable to the threat of confusing threats with vulnerabilities. The Journal of Physical Security 4, 30–34.
Khazaei, A., Ghasemzadeh, M., Derhami, V., 2016. An automatic method for CVSS score prediction using vulnerabilities description. Journal of Intelligent & Fuzzy Systems 30 (1), 89–96.
Khoshgoftaar, T.M., Golawala, M., Van Hulse, J., 2007. An empirical study of learning from imbalanced data using random forest. In: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, pp. 310–317.
Kitchenham, B., Madeyski, L., Budgen, D., et al., 2017. Robust statistical methods for empirical software engineering. Empirical Software Eng. 22, 579–630.
Kohavi, R., John, G.H., 1995. Automatic parameter selection by minimizing estimated error. In: Machine Learning Proceedings, 1995. Elsevier, pp. 304–312.
Kudjo, P.K., Chen, J., Zhou, M., et al., 2019. Improving the Accuracy of Vulnerability Report Classification Using Term Frequency-Inverse Gravity Moment. In: Proceedings of the19th IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 248–259.
Kudjo, P.K., 2019. A cost-effective strategy for software vulnerability prediction based on bellwether analysis. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis,2019, pp. 424–427.
Kumari, M., Jain, A., Bhatia, A., 2016. Synonyms Based Term Weighting Scheme: an Extension to TF. IDF. Procedia Comput Sci 89, 555–561.
Lamkanfi, A., Demeyer, S., Soetens, Q.D., et al., 2011. Comparing mining algorithms for predicting the severity of a reported bug. In: Proceedings of the 15th IEEE European Conference on Software Maintenance and Reengineering, (CSMR), pp. 249–258.
Lan, M., Tan, C.L., Low, H.B., et al., 2005. A comprehensive comparative study on term weighting schemes for text categorization with support vector machines. In: Special Interest Tracks and Posters of the 14th International Conference on World Wide Web, pp. 1032–1033.
Lan, M., Tan, C.L., Su, J., et al., 2008. Supervised and traditional term weighting methods for automatic text categorization. IEEE Trans Pattern Anal Mach Intell 31 (4), 721–735.
Lan, M., Tan, C.L., Su, J., et al., 2009. Supervised and traditional term weighting methods for automatic text categorization. IEEE Trans. Pattern. Anal Mach. Intell. 31, 721–735.
Lei, S., 2020. A feature selection method based on information gain and genetic algorithm. In: International Conference on Computer Science and Electronics Engineering, pp. 355–358 12AD.
Leopold, E., Kindermann, J., 2002. Text categorization with support vector machines. How to represent texts in input space? Mach Learn 46, 423–444.
Lessmann, S., Baesens, B., Mues, C., et al., 2008. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Transactions on Software Eng. 34, 485–496.
Li, X., Chen, J., Lin, Z., et al., 2017a. A Mining Approach to Obtain the Software Vulnerability Characteristics. In: Proceedings of the 5th IEEE International Conference on Advanced Cloud and Big Data, pp. 296–301.
Li, X., Chang, X., Board, J.A., et al., 2017b. A novel approach for software vulnerability classification. In: Reliability and Maintainability Symposium (RAMS), pp. 1–7.
Mell, P., Scarfone, K., Romanosky, S., 2006. Common vulnerability scoring system. IEEE Secur Priv 4 (6), 85–89.
Merrouni, Z.A., Frikh, B., Ouhbi, B., 2019. Automatic keyphrase extraction: a survey and trends. J. Intell. In.f Syst. 1–34.
Mirjalili, S., 2019. Evolutionary Multi-layer Perceptron. in Evolutionary Algorithms and Neural Networks, ed: Springer 87–104.
Munaiah, N., Meneely, A., 2016. Vulnerability severity scoring and bounties: why the disconnect? In: Proceedings of the 2nd International Workshop on Software Analytics, pp. 8–14.
NationalInstitute of Standards and Technology Interagency Report 8151, https://doi.org/10.6028/NIST.IR.8151. 2020.
Lindsey O'Donnell, "Windows Users at Risk From High-Severity Intel Software Flaw," https://threatpost.com, 2019.
Pang, Y., Xue, X., Wang, H., 2017. Predicting vulnerable software components through deep neural network. In: International Conference on Deep Learning Technologies, pp. 6–10.
Porter, M.F., 1980. An algorithm for suffix stripping. Program 14 (3), 130–137.
Russo, E.R., Di Sorbo, A., Visaggio, C.A., Canfora, G., 2019. Summarizing Vulnerabilities' Descriptions to Support Experts during Vulnerability Assessment Activities. Journal of Systems and Software 156, 84–99.
Sabbah, T., Selamat, A., Selamat, M.H., et al., 2017. Modified frequency-based term weighting schemes for text classification. Appl. Soft Comput. 58, 193–206.
Sabetta, A., Bezzi, M., 2018. A practical approach to the automatic classification of security-relevant commits. In: IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 579–582.
Sahin, S.E., Tosun, A., 2019. A Conceptual Replication on Predicting the Severity of Software Vulnerabilities. In: Proceedings of the Evaluation and Assessment on Software Engineering.
Salton, G., Wong, A., Yang, C.-S., 1975. A vector space model for automatic indexing. Commun ACM 18, 613–620.
Sharma, M., 2015. The way ahead for bug-fix time prediction. In: Proceedings of the 3rd International Workshop on Quantitative Approaches to Software Quality, p. 33.
Shin, Y., Williams, L., 2013. Can traditional fault prediction models be used for vulnerability prediction? Empirical Software Eng. 18 (1), 25–59.
Silhavy, R., Silhavy, P., Prokopova, Z., 2018. Evaluating subset selection methods for use case points estimation. Inf Softw Technol 97, 1–9.
Singh, U.K., Joshi, C., 2016. Quantitative security risk evaluation using CVSS metrics by estimation of frequency and maturity of exploit. In: Proceedings of the World Congress on Engineering and Computer Science, pp. 19–21.
Skybox Security, Vulnerability and threat trends report, www.vulnerabilitycenter.com, 2019.
Spanos, G., Angelis, L., 2018. A multi-target approach to estimate software vulnerability characteristics and severity scores. Journal of Systems and Software 146, 52–166.
Spanos, G., Angelis, L., Toloudis, D., 2017. Assessment of vulnerability severity using text mining. In: Proceedings of the 21st Pan-Hellenic Conference on Informatics, p. 49.
Stuckman, J., Walden, J., Scandariato, R., 2017. The effect of dimensionality reduction on software vulnerability prediction models. IEEE Transactions on Reliability 66 (1), 17–37.
Tian, Y., Lo, D., Xia, X., Sun, C., 2015. Automated Prediction of Bug Report Priority Using Multi-Factor Analysis. Empirical Software Engineering 20 (5), 1354–1383.
Uysal, A.K., Gunal, S., 2014. Text classification using genetic algorithm oriented latent semantic features. Expert Syst Appl 41, 5938–5947.
VulnerabilityResearch Market Analysis, https://www.techrepublic.com [Online: accessed 22-April-2019]. 2019.
Wang, Y., S, Wu, Li, D., et al., 2016. A Part-Of-Speech term weighting scheme for biomedical information retrieval. J Biomed Inform 63, 379–389.
Wang, P., Zhou, Y., Sun, B., Zhang, W., 2019. Intelligent Prediction of Vulnerability Severity Level Based on Text Mining and XGBboost. In: Proceedings of the 11th IEEE International Conference on Advanced Computational Intelligence (ICACI), pp. 72–77.
Wijayasekara, D., Manic, M., Mcqueen, M., 2014. Vulnerability identification and classification via text mining bug databases. In: Proceedings of the 40th IEEE Annual Conference on Industrial Electronics Society, IECON, (IEEE, 2014), pp. 3612–3618.
Witten, I.H., Medelyan, O., 2006. Thesaurus based automatic keyphrase indexing. In: Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'06), pp. 296–297.
Xia, X., Lo, D., Qiu, W., et al., 2014. Automated configuration bug report prediction using text mining. In: Proceedings of the 38th Annual IEEE Conference on Computer Software and Applications (COMPSAC), pp. 107–116.
Xuan, N.P., Le Quang, H., 2014. A new improved term weighting scheme for text categorization. In: Knowledge and Systems Engineering. Springer, pp. 261–270.
Younis, A., Malaiya, Y.K., Ray, I., 2016. Evaluating CVSS base score using vulnerability rewards programs. In: IFIP International Information Security and Privacy Conference, pp. 62–75.
Yuen, K.K., 1974. The Two-Sample Trimmed t for Unequal Population Variances. Biometrika 60 (1), 165–170.
Zhang, S., Caragea, D., Ou, O., 2011. An empirical study on using the national vulnerability database to predict software vulnerabilities. In: International Conference on Database and Expert Systems Applications, 6860, pp. 217–231.
Zhu, X., Cao, C., Zhang, J., 2017. Vulnerability severity prediction and risk metric modeling for software. Applied Intelligence 47, 828–836.
Zou, D., Yang, J., Li, Z., Jin, H., Ma, X., 2019. AutoCVSS: an Approach for Automatic Assessment of Vulnerability Severity Based on Attack Process. In: International Conference on Green, Pervasive, and Cloud Computing, pp. 238–253.

Jinfu Chen is a Professor in the School of Computer Science and Communication Engineering, Jiangsu University, China. He received a B.E. degree from Nanchang Hangkong University, Nanchang, China, in 2004, and a Ph.D. degree from Huazhong University of Science and Technology, Wuhan, China, in 2009, both in Computer Science. His-major research interests include software testing, software analysis, and trusted software. He is a member of the ACM, IEEE, and the China Computer Federation.

Patrick Kwaku Kudjo received his B.A. in Computer Science and Management from Wisconsin International University College, awarded by University of Ghana, Legon, MSc. in Information Technology from Sikkim Manipal University, India in 2013 and PhD in Computer Application Technology at Jiangsu University, China under the supervision of Prof Jinfu Chen. His-main research interests are Information Security, Software Testing, Machine Learning, Text Mining, and Blockchain. He is a student member of ACM and IEEE.

Solomon Mensah received his BSc degree in Computer Science and Statistics from University of Ghana in 2011, MEng in Computer Engineering from University of Ghana in 2014 and Ph.D. in Computer Science at City University of Hong Kong under the supervision of Dr. Jacky Keung. He is currently a Lecturer at the Department of Computer Science, University of Ghana, Legon. His-research interests include Software Effort Estimation, Technical Debt, and Deep Learning. He has experience in data mining and the development of predictive and forecasting models.

Selasie Aformaley Brown holds an undergraduate degree in Computer Science, and a Master's degree in Health Informatics, both awarded by the University of Ghana in the years 2010 and 2012 respectively. He is an Assistant Lecturer in the Information Technology Department of the University of Professional Studies Accra (UPSA), Ghana. Selasie is also a Ph.D. candidate in Computer Science at the University of Energy and Natural Resources (UENR), Ghana. His-areas of interest are Deep Learning, Bioinformatics, Cognitive Informatics, and Information Security.

George K.S. Akorfu has a BSc (Hons) degree in Computer Science and Statistics from the University of Ghana, a Licentiate for the Institute of Management Information Systems (IMIS) London, Masters in Business Administration (MBA-MIS option) from the University Of Ghana, Legon, Accra & Vrije Universiteit Brussel, Brussels, and a Student of Chartered Institute of Management Accountants (CIMA). He is currently a graduate in Doctor of Philosophy (Ph.D.) at Open University Malaysia (OUM) in Information Technology (IT) obtained at Accra Institute of Technology (AIT). His-main research interests are Information Systems Development and Information Management. He is currently a Senior Lecturer and Head of the Business Computing Department at Wisconsin International University College, Ghana.