

Automatic detection of safety requests in web and mobile applications using natural language processing techniques

Salim Salmi, Lahcen Oughdir

National School of Applied Sciences, Engineering Systems and Applications Laboratory, Sidi Mohamed Ben Abdellah University, Fez, Morocco

Article Info

Article history:

Received Nov 9, 2023

Revised Feb 13, 2024

Accepted Feb 28, 2024

Keywords:

Classification

Deep learning

Machine learning

Natural language processing
safety requests

ABSTRACT

Web and mobile applications have become an essential part of our daily lives. However, as the usage of these applications increases, so does the potential for safety concerns. It is crucial for application developers to ensure that their applications are safe and secure for users. One way to achieve this is through the identification and processing of safety requests made by users. This research paper proposes a method for identifying safety requests made by users in web and mobile applications using natural language processing (NLP) and deep learning techniques. The approach involves training a machine learning and deep learning model on a dataset of user requests to identify and classify safety requests. The models are then integrated into the application's code to automatically detect and respond to safety requests. A case study on a ride-sharing application showed that the proposed approach achieved high accuracy in identifying safety requests, with an F1 score of 0.85. The proposed method can be applied to various web and mobile applications to improve safety and security, and reduce the workload of manual safety request processing.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Salim Salmi

National School of Applied Sciences, Engineering Systems and Applications Laboratory

Sidi Mohamed Ben Abdellah University

Fez, Morocco

Email: salim.salmi@usmba.ac.ma

1. INTRODUCTION

In recent years, the widespread use of web and mobile applications has led to a growing concern for user safety. As a result, developers have become increasingly aware of the need to implement safety features and provide users with the ability to request help or report unsafe situations [1]. Safety requests, such as bug reports, vulnerability reports, and feature requests [2], are essential for maintaining the security and quality of web and mobile applications [3]. Detecting these requests is a challenging task, especially for large scale applications, as manual monitoring and analysis of user feedback is time-consuming and inefficient. Therefore, there is a need for an automated approach to detect safety requests in web and mobile applications [4].

To address this issue, natural language processing (NLP) techniques have emerged as a promising solution for the automatic detection of safety requests in web and mobile applications [5]. NLP stands as a pivotal subset of artificial intelligence, specializing in the interplay between computers and human language [6]. Its core objective is to empower machines to comprehend, decipher, and produce human language, a capability highly relevant to various domains, including safety detection applications [7], [8]. The automatic detection of safety requests can significantly improve the user experience and enhance the overall safety of

web and mobile applications [9]. By automatically identifying safety-related messages, developers can quickly respond to user requests and take appropriate actions to prevent unsafe situations.

This research paper aims to investigate the effectiveness of NLP techniques for the automatic detection of safety requests in web and mobile applications. The study will involve the development of a prototype system that uses NLP algorithms to detect safety-related messages in real time. The proposed system will be evaluated using a dataset of safety-related messages collected from web and mobile applications. Overall, the findings of this research can contribute to the development of more secure and user-friendly web and mobile applications by providing developers with a powerful tool for automatic safety request detection. The study also has implications for the broader field of NLP and artificial intelligence (AI), as it demonstrates the potential for these technologies to improve user safety in a variety of contexts.

The rest of this paper is structured as follows: section 2 explores previous research on NLP techniques for identifying safety requests. In section 3, we provide a detailed explanation of our proposed approach, covering data collection, preprocessing, feature extraction, and classification. Section 4 presents the results of our evaluation of the proposed approach. Finally, in section 5, we conclude the paper and consider potential avenues for future research.

2. LITERATURE REVIEW

In the ever-evolving digital landscape of web and mobile applications, the safety of user requests is of paramount importance. These applications serve as gateways to various online activities, and the distinction between safe and unsafe requests is crucial [10]. “Safety” here encompasses the prevention of data breaches, unauthorized access, and numerous other security threats. The need to classify user requests for their safety has led to the development of advanced algorithms and models. This background provides a concise introduction to the significance of ensuring the security and well-being of users within the digital realm, emphasizing the importance of accurately classifying user requests for proactive threat mitigation and data protection.

Rong *et al.* [11] propose a relearning-equipped malicious request detection system using an enhanced convolutional neural network (CNN) model. Notable features include a character-level embedding layer for improved understanding of request parameters and customized CNN filters for refined feature extraction. Empirical tests demonstrate superior performance compared to traditional methods like support vector machines (SVM) and random forest (RF), with a lower false positive rate. The model presents a promising solution for enhancing web application security by mitigating web parameter injection attacks.

Salmi and Oughdir [12] introduce an adaptive deep learning system to identify web-based code-injection attacks, emphasizing the four most common categories of code-injection attacks. The system allows periodic updates with new queries but is noted for its limited focus on local features, which are crucial for accurate detection outcomes. Recognizing the significance of hypertext transfer protocol (HTTP) requests, the design of an efficient and resilient HTTP request analyzer is paramount to ensure the detection and prevention of malicious requests. Jemal *et al.* [13] introduce a novel approach called code embedding for processing HTTP requests within a convolutional neural network, enhancing web attack detection efficiency. Empirical results highlight its superiority over previous methods, achieving an impressive 98.12% accuracy rate.

Luo *et al.* [14] propose an innovative system for detecting malicious URLs in web security, employing autoencoders and deep learning for classification. The approach integrates NLP and autoencoders for automated feature extraction, demonstrating robust capability in identifying anomalous URLs across datasets—a significant advancement in web security through the application of deep learning and NLP methods. Similarly, Junior *et al.* [15] introduce a novel method for identifying potential attacks within HTTP requests using machine learning. Their model utilizes bidirectional encoder representations from transformers (BERT) and bidirectional long short-term memory (BiLSTM) to detect anomalies, consistently achieving accuracy rates exceeding 95% in attack detection through comprehensive experiments on established datasets and a newly created HTTP request dataset. Together, these studies highlight advancements in leveraging cutting-edge technologies for enhancing web security.

3. METHOD

In the following section, we outline our proposed work, which centers on the utilization of two powerful text classification algorithms: the bag of words (BoW) and term frequency-inverse document frequency (TF-IDF) methods. These algorithms are fundamental in NLP and text analysis. Our objective is to explore

their effectiveness for safety requests in web and mobile application categorization and evaluate their performance in different contexts.

Additionally, we will compare the results obtained from the BoW and TF-IDF methods with those from a deep learning approach. This comparative evaluation will provide insights into the strengths and limitations of each approach, aiding in the informed selection of the most suitable algorithm for request classification tasks. Algorithm 1 describes the methodology steps for this work.

Algorithm 1: Requests classification methodology algorithm

Data: Input text data
Result: Classified request
 Initialize the training dataset with labeled data;
 Preprocess the text data by removing stopwords, punctuation, and stemming;
for each text document in the dataset **do**
 Create a feature vector using the Bag of Words (BoW) method;
 Create a feature vector using the TF-IDF method;
 Train a classification model using the BoW feature vector;
 Train a classification model using the TF-IDF feature vector;
end
Output: Select the best-performing model based on evaluation metrics

3.1. DataSet description

The dataset at hand is sourced from user requests in web and mobile applications, encompassing various features. Among the features, "Body title," "Body description," and "isSafe" are the notable components. In Table 1, it becomes evident that these features possess unique values that can be pivotal in the analysis.

In the provided dataset, each request includes a request payload, and there is a corresponding "isSafe" field. The purpose of the "isSafe" field is to indicate whether the request is safe or not for the application. If the "isSafe" field has a value of "False," it signifies that the request should be blocked. This determination is based on the evaluation of the fields within each request. A request is labeled as "not safe" if any of its fields contain data that could be exploited for malicious purposes. In such cases, the request should be blocked or handled with caution.

Table 1. Dataset samples

Body title	Body description	isSafe
Tina Johnson	Top recognize eat. Fact whom spend area thing ...	True
Clayton Cooper	As possible American many prepare four strong....	True
Curtis Wolfe	Tuesday Notes or 2 like 2 XSP Class	False
Laura Fisher	State third represent energy campaign not forg...	True
Tyler Santos	Us enjoy since. Time identify image position o...	False

Remarkably, all the other features in the dataset maintain a uniform, single value. This uniformity renders them unsuitable for distinguishing whether a request is potentially an attack by a malicious actor or not. A notable aspect of this dataset is the distribution of the "isSafe" feature, which serves as a critical indicator of request safety. Out of the total dataset (1000) requests, 428 requests are marked as unsafe, while 572 requests are identified as safe.

3.2. Bag of words

The BoW text classification algorithm is a foundational approach to analyze and classify text documents [16]. It initiates data preprocessing, tokenizing the text and removing common stopwords to retain significant content [17]. Next, it creates a vocabulary from the unique words in the corpus, counting word occurrences in each document to build frequency vectors. These vectors represent documents numerically, capturing word frequencies as features. The dataset is then split into training and testing sets for model evaluation. A Bernoulli naive Bayes classifier is utilized to build a model using the BoW features. Ultimately, the model is trained on the training data and assessed for accuracy in classifying requests. All the preceding steps are detailed in Algorithm 2.

Algorithm 2: Request Classification Using BoW

Data: Corpus of text documents
Result: Trained classification model
Data Preprocessing;
for *each document in the corpus* **do**
 | Tokenize, remove stopwords;
end
Feature Extraction with BoW;
Create a vocabulary, vectorize text;
Split Data: Divide into training and testing sets;
Model Selection;
if *selected model is BernoulliNB()* **then**
 | Model Training: Train using BernoulliNB() classifier;
 | Model Evaluation: Evaluate model accuracy on test data;
end
else
 | Select a valid classification model;
end

The BoW model represents a text document as a numerical vector based on the frequency of words in the document. Description of the formula in (1).

$$\text{BoW}(D) = (\text{count}(\text{word}_i, D), \dots, \text{count}(\text{word}_N, D)) \quad (1)$$

Where $\text{count}(\text{word}_i, D)$ is the count or frequency of word_i in document D . $\text{BoW}(D)$ is a numerical vector with N elements, where each element represents the count of a specific word from vocabulary V in document D .

3.3. Term frequency-inverse document frequency

Text classification using TF-IDF is a robust method for categorizing text documents [18]-[20]. The process involves tokenization, eliminating common stopwords, and constructing a vocabulary of unique words. Term frequency (TF) and inverse document frequency (IDF) values are computed for each term, converting documents into numerical vectors for TF-IDF features. The dataset is split into training and testing sets, and a multinomial naive Bayes classifier is applied to build a model based on these features. The model's accuracy in document classification is then assessed on the test data. Algorithm 3 outlines the detailed steps, with t calculated as the ratio of a term's occurrences to the total terms in a document, as described in (2).

$$TF = \frac{\text{word_count}}{\text{total_terms}} \quad (2)$$

Algorithm 3: Request classification using TF-IDF

Data: Text Corpus
Result: Trained Classifier
Data Preprocessing;
foreach *document in the corpus* **do**
 | Tokenize and remove stopwords;
end
TF-IDF Feature Extraction;
Create vocabulary and calculate TF-IDF values;
foreach *document in the corpus* **do**
 if *document is not empty* **then**
 | Vectorize the document;
 end
end
Split Data into training and testing sets;
Model Selection: MultinomialNB() using TF-IDF features;
Model Training: Train the model on the training data;
Model Evaluation: Evaluate accuracy on the test data;

IDF is calculated as the logarithm of the ratio of the total number of documents N to the number of documents containing the term (doc_count). Description of the formula in (3).

$$IDF = \log \left(\frac{N}{doc_count} \right) \quad (3)$$

The TF-IDF score for a term in a document is the product of its TF and IDF values, a description of the formula in (4).

$$TF - IDF = TF \times IDF \quad (4)$$

The purpose of TF-IDF is to assign a weight to each term in a document that reflects its importance not only in that document but also in the entire corpus. Terms that are common within a document but rare across the corpus will have higher TF-IDF scores, indicating their significance in the document.

3.4. Multinomial naive Bayes

Multinomial naive Bayes is a variant of the naive Bayes algorithm that is specifically tailored for text classification tasks [21]. Unlike the simple naive Bayes model, which treats text as a BoW with only word presence or absence, multinomial naive Bayes explicitly models word counts, making it particularly well suited for text data [22]. In multinomial naive Bayes, the fundamental formulas for classification are adjusted to work with word counts. Description of the formula in (5).

$$P(tk|c) = \frac{N(tk, c) + 1}{N(c) + V} \quad (5)$$

Where $P(tk|c)$ denotes the conditional probability of the term tk appearing in a document of class c . $N(tk, c)$ represents the count of term tk within documents of class c . $N(c)$ indicates the total count of terms in documents of class c . V is the vocabulary size, which refers to the total number of unique terms. For further description of the formula, see (6).

$$P(c) = \frac{N(c)}{N} \quad (6)$$

Where $P(c)$ is the prior probability of a document occurring in class c , $N(c)$ is the count of documents in class c , and N is the total count of documents. Multinomial naive Bayes calculates the conditional probability of each term tk occurring in a document of class c using Laplace smoothing (adding 1 to the counts) and then combines these probabilities to classify the document into the most likely class.

3.5. The Bernoulli naive Bayes

The Bernoulli naive Bayes classifier is a variant of the naive Bayes algorithm suitable for binary text classification tasks [23], where each term or feature is treated as a binary variable indicating its presence or absence in a document [24]. Description of the formula in (7).

$$P(tk|c) = \frac{N(tk, c) + \alpha}{N(c) + 2\alpha} \quad (7)$$

Where $P(tk|c)$ represents the conditional probability of the term tk occurring in a document of class c . $N(tk, c)$ denotes the count of term tk in documents of class c . $N(c)$ signifies the total count of terms in documents of class c . α is a smoothing parameter, typically set to 1. In the case of Bernoulli naive Bayes, it shares the same P_P as multinomial naive Bayes for a document within a class.

The Bernoulli naive Bayes classifier estimates the conditional probability of each term tk occurring in a document of class c and combines these probabilities to classify the document as belonging to the most likely class [25]. The use of Laplace smoothing helps prevent zero probabilities and accounts for unobserved terms [26]. Bernoulli naive Bayes is an efficient and effective algorithm for binary text classification tasks, making it a valuable tool in NLP and machine learning.

4. RESULTS AND DISCUSSION

This section presents the outcomes of our extensive experiments focused on request classification within the domains of NLP and machine learning. Our objective was to thoroughly assess the performance of various techniques and classifiers, including deep learning models, in the intricate task of request classification.

The experiments covered diverse approaches such as feature engineering, sentiment analysis, and text categorization methods. Utilizing cutting-edge machine learning classifiers and neural networks, we examined their effectiveness in accurately classifying user requests. The experiments utilized a substantial dataset comprising user requests from diverse web and mobile applications. The results offer valuable insights into the strengths and limitations of each approach, aiding in the identification of optimal strategies for request classification across varied applications and contexts. This in-depth exploration serves as a valuable resource for practitioners and researchers aiming to enhance the effectiveness of request classification in real-world scenarios.

4.1. Bag of word model evaluation

The description provides a concise overview of implementing a text classification model using the BoW technique with a Bernoulli naive Bayes classifier. It begins by importing necessary libraries from the scikit-learn toolkit and utilizing CountVectorizer (CV) to convert text data into numerical representations. CountVector is configured with a maximum of 100 features and a unigram range (1,1) for ngrams. Data are split into training and testing sets with an 80/20 split ratio, paired with target labels. The Bernoulli naive Bayes classifier is employed for classification. To evaluate performance, an accuracy score is calculated, and a confusion matrix and classification report are printed in Figure 1 for deeper insights into the model's performance.

The model achieved an accuracy score of 84%. This accuracy score represents the model's effectiveness in correctly classifying text data into safe and unsafe classes. The confusion matrix in Figure 2 illustrates the model's classification performance, with 49 instances correctly classified as safe, 32 instances incorrectly classified as safe when they were actually unsafe and 119 instances correctly classified as unsafe. It appears to be a situation where the model is performing better at identifying unsafe instances than safe.

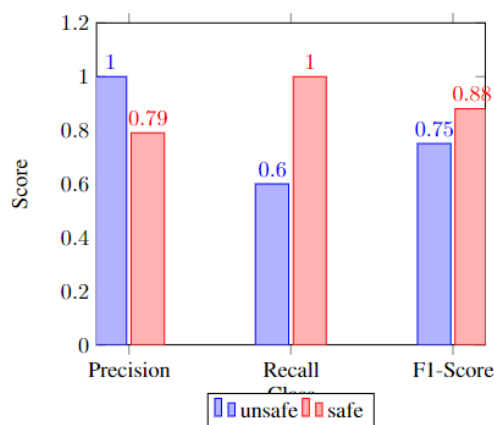


Figure 1. Precision, recall, and F1-score

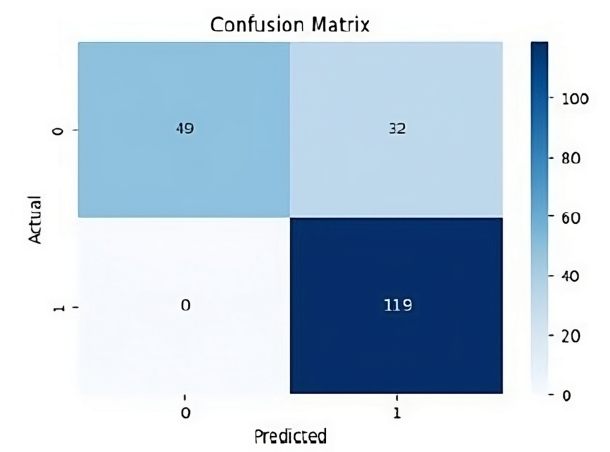


Figure 2. Bernoulli naive Bayes classifier confusion matrix

4.2. Term frequency-inverse document frequency technique evaluation

In this section, the process of implementing a text classification model is detailed, utilizing the TF-IDF vectorization method alongside a multinomial naive Bayes classifier. The initial step involves importing a versatile tool for converting textual data into numerical features. The vectorizer is configured to use unigrams (single words) with an ngram range of (1, 1) and limits the feature set to the top 3000 most important words using *max_features=3000*. The text data, stored in the 'corpus' variable, undergoes transformation into numerical features using the *fit_transform* method, resulting in a dense array. The dataset is then divided into training and testing subsets, with 20% allocated for testing and reproducibility ensured by setting *random_state = 0*. The multinomial naive Bayes classifier is chosen for text classification tasks due to its compatibility with discrete data like TF-IDF vectors. The classifier is imported, instantiated, and trained on the TF-IDF representations of the text data and their corresponding labels using the fit method.

After training, the model is used to make predictions on the test data. The model's performance is assessed by computing the accuracy score, representing the proportion of correctly classified instances. The

achieved accuracy score is 84%, demonstrating the model's effectiveness in classifying text data.

4.3. Deep learning model evaluation

This section focuses on a meticulously designed deep learning model, implemented using the TensorFlow framework, aimed at accurately categorizing incoming requests as safe or unsafe in real-time. The initial steps involve data preprocessing and tokenization, where a tokenizer is configured to recognize the top 100 most frequent words, capturing essential linguistic patterns within the request data. The model's core includes an embedding layer with an embedding dimension of 64, mapping words to dense numerical vectors to encode both semantic and syntactic information. The sequential application programming interface (API) is used to construct the model with various layers, including GlobalAveragePooling1D for efficient downsampling and dropout layers (with rates of 0.2 and 0.3) to prevent overfitting. The model concludes with a dense layer using a sigmoid activation function, suitable for binary classification tasks (safe vs. unsafe requests). Table 2 provides an overview of the deep learning model.

Table 2. Summary of deep learning model

Layer (type)	Output shape	Param #
embedding (Embedding)	(None, None, 64)	6,400
GlobalAveragePooling1D	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 6)	390
dropout_1 (Dropout)	(None, 6)	0
dense_1 (Dense)	(None, 1)	7
Total params:		6,797
Trainable params:		6,797
Non-trainable params:		0

The model embarks on the training phase. The 'loss' function is defined as *binary_crossentropy*, a natural choice for binary classification tasks. 'Adam' serves as the optimizer, a versatile and efficient choice. Performance monitoring centers on 'accuracy,' a key metric that gauges the model's proficiency. The model undergoes training over 10 epochs, allowing it to adapt and learn the subtleties of the request data. The incorporation of a validation split of 20% facilitates ongoing evaluation, offering insights into the model's performance and its ability to generalize beyond the training data.

As the training process concludes, the model transitions to the evaluation phase, where its performance is meticulously assessed. It is with great pride that we report an exceptional test accuracy of 84.5%, as shown in Figure 3. This metric signifies the model's capability to effectively categorize incoming requests as safe or unsafe, instilling confidence in its ability to make rapid and accurate security decisions.

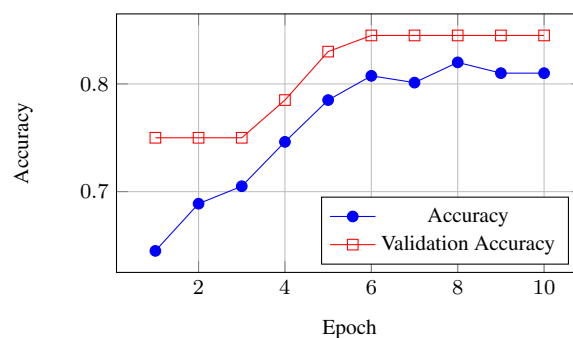


Figure 3. Training and validation accuracy over epochs

4.4. Discussion

The information in Table 3 comprehensively compares different classification models applied to text analysis, specifically, on a dataset comprising user requests from web and mobile applications. Three distinct classifiers—Bernoulli naive Bayes, multinomial naive Bayes, and a deep learning model—were utilized in this

study. The evaluation involved two feature extraction techniques: TF-IDF and BoW. The accuracy scores for each combination of classifier and feature type are detailed in Table 3.

Table 3. Classifier features and accuracy

Classifier	Features	
	BoW	TF-IDF
Bernoulli naive Bayes	0.84	0.84
Multinomial naive Bayes	0.84	0.84
Deep learning	0.845	0.845

Notably, both the Bernoulli naive Bayes and multinomial naive Bayes models demonstrated similar accuracy scores, achieving 84% for both TF-IDF and BoW features. In contrast, the deep learning model surpassed the traditional naive Bayes models, attaining an accuracy of 84.5% for both TF-IDF and BoW. This comparative analysis underscores that the integration of deep learning with either TF-IDF or BoW features yields superior performance in classifying user requests within web and mobile applications. The importance of the chosen feature extraction technique is evident, emphasizing the need to select the most suitable method based on the specific requirements of the task.

5. CONCLUSION

In this paper, we have explored the implementation of Requests classification models using two fundamental approaches: the BoW and TF-IDF vectorization methods in conjunction with various classifiers. Our work demonstrated efficiency and versatility by achieving an accuracy of 84%. Additionally, we explored deep learning techniques, which yielded a slightly higher accuracy of 84.5%. These results underscore the effectiveness of both traditional and modern approaches in request classification. By comparing the outcomes and assessing the strengths and limitations of each approach, we have provided valuable insights for practitioners and researchers seeking effective Requests classification techniques. The choice between BoW and TF-IDF, as well as the selection of an appropriate classifier, depends on the specific requirements and nature of the text classification task.




REFERENCES

- [1] G. A. Jaafar, S. M. Abdullah, and S. Ismail, "Review of recent detection methods for http DDoS attack," *Journal of Computer Networks and Communications*, vol. 2019, 2019, doi: 10.1155/2019/1283472.
- [2] W. Zheng, J. Cheng, X. Wu, R. Sun, X. Wang, and X. Sun, "Domain knowledge-based security bug reports prediction," *Knowledge-Based Systems*, vol. 241, 2022, doi: 10.1016/j.knosys.2022.108293.
- [3] F. A. Bhuiyan, M. B. Sharif, and A. Rahman, "Security bug report usage for software vulnerability research: A systematic mapping study," *IEEE Access*, vol. 9, pp. 28471–28495, 2021, doi: 10.1109/ACCESS.2021.3058067.
- [4] J. Shahid, M. K. Hameed, I. T. Javed, K. N. Qureshi, M. Ali, and N. Crespi, "A comparative study of web application security parameters: Current trends and future directions," *Applied Sciences*, vol. 12, no. 8, 2022, doi: 10.3390/app12084077.
- [5] S. L. Blodgett, S. Barocas, H. D. III, and H. Wallach, "Language (technology) is power: A critical survey of "bias" in NLP," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 5454–5476, doi: 10.18653/v1/2020.acl-main.485.
- [6] S. Meera and S. Geerthik, "Natural language processing," in *Artificial Intelligent Techniques for Wireless Communication and Networking*, doi: 10.1002/9781119821809.ch10.
- [7] B. Bhatti, S. Mubarak, and S. Nagalingam, "Information security implications of using nlp in it outsourcing: A diffusion of innovation theory perspective," *Automated Software Engineering*, vol. 28, no. 12, 2021, doi: 10.1007/s10515-021-00286-x.
- [8] V. Varenov and A. Gabdrahmanov, "Security requirements classification into groups using nlp transformers," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 2021, pp. 444–450, doi: 10.1109/REW53955.2021.9714713.
- [9] V. Odumuyiwa and A. Chibueze, "Automatic detection of http injection attacks using convolutional neural network and deep neural network," *JCSANDM*, vol. 9, no. 4, pp. 489–514, 2021, doi: 10.13052/jcsm2245-1439.941.
- [10] R. Mohammadi, C. Lal, M. Conti, and L. Sharma, "Software defined network-based http flooding attack defender," *Computers and Electrical Engineering*, vol. 101, 2022, doi: 10.1016/j.compeleceng.2022.108019.
- [11] W. Rong, B. Zhang, and X. Lv, "Malicious web request detection using character-level CNN," in *Machine Learning for Cyber Security*, 2019, doi: 10.1007/978-3-030-30619-9_2.
- [12] S. Salmi and L. Oughdir, "Performance evaluation of deep learning techniques for DoS attacks detection in wireless sensor network," *Journal Big Data*, vol. 10, no. 17, pp. 1–25, 2023, doi: 10.1186/s40537-023-00692-w.
- [13] I. Jemal, M. A. Haddar, O. Cheikhrouhou, and A. Mahfoudhi, "Malicious http request detection using code-level convolutional neural network," in *Lecture Notes in Computer Science*, 2021, pp. 317–324, doi: 10.1007/978-3-030-68887-5_19.
- [14] C. Luo, S. Su, Y. Sun, Q. Tan, M. Han, and Z. Tian, "A convolution-based system for malicious urls detection," *CMC—Computers Materials Continua*, vol. 62, no. 1, pp. 399–411, 2020, doi: 10.1007/s10515-021-00286-x.




- [15] L. R. Junior, D. Macedo, A. Oliveira, and C. Zanchettin, "Logbert-bilstm: Detecting malicious web requests," in *Artificial Neural Networks and Machine Learning – ICANN 2022*, 2022, doi: 10.1007/978-3-031-15934-3_58.
- [16] W. A. Qader, M. M. Ameen, and B. I. Ahmed, "An overview of bag of words; importance, implementation, applications, and challenges," in *2019 International Engineering Conference (IEC)*, 2019, pp. 200–204, doi: 10.1109/IEC47844.2019.8950616.
- [17] S. Sarica and J. Luo, "Stopwords in technical language processing," *PLoS ONE*, vol. 16, no. 8, 2021, doi: 10.1371/journal.pone.0254937.
- [18] H. Christian, M. P. Agus, and D. Suhartono, "Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF)," *ComTech: Computer, Mathematics and Engineering Applications*, vol. 7, no. 4, pp. 285–294, 2016, doi: 10.21512/comtech.v7i4.3746.
- [19] V. Kumar and B. Subba, "A TF-IDFvectorizer and svm based sentiment analysis framework for text data corpus," in *2020 National Conference on Communications (NCC)*, 2020, pp. 1–6, doi: 10.1109/NCC48643.2020.9056085.
- [20] I. Gupta, S. Mittal, A. Tiwari, P. Agarwal, and A. Singh, "TIDF-DLPM: Term and inverse document frequency based data leakage prevention model," *arXiv-Computer Science*, 2022, doi: 10.48550/arXiv.2203.05367.
- [21] S. Gan, S. Shao, L. Chen, L. Yu, and L. Jiang, "Adapting hidden naive bayes for text classification," *Mathematics*, vol. 9, no. 19, 2021, doi: 10.3390/math9192378.
- [22] L. Jiang, C. Li, S. Wang, and L. Zhang, "Deep feature weighting for naive bayes and its application to text classification," *Engineering Applications of Artificial Intelligence*, vol. 52, pp. 26–39, 2016, doi: 10.1016/j.engappai.2016.02.002.
- [23] Y. Ying, T. N. Mursitama, Shidarta, and Lohansen, "Effectiveness of the news text classification test using the naive bayes' classification text mining method," *Journal of Physics: Conference Series*, vol. 1764, no. 1, 2021, doi: 10.1088/1742-6596/1764/1/012105.
- [24] G. Singh, B. Kumar, L. Gaur, and A. Tyagi, "Comparison between multinomial and bernoulli naive bayes for text classification," *Proceedings of the 2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, pp. 593–596, 2019, doi: 10.1109/ICACTM.2019.8776800.
- [25] B. Santhi and G. Brindha, "Multinomial naive bayes using similarity based conditional probability," *Journal of Intelligent and Fuzzy Systems*, pp. 1431–1441, 2019, doi: 10.3233/JIFS-181009.
- [26] L. D. R. -Reis, "Laplace distribution: properties, regression model and simulation," *Journal of Statistics and Management Systems*, vol. 25, no. 2, pp. 489–499, 2022, doi: 10.1080/09720510.2021.1930664.

BIOGRAPHIES OF AUTHORS



Salim Salmi    received his M.Sc. degree at the National Higher School of Computer Science and Systems Analysis (ENSIAS) Mohammed V University Rabat, Morocco in 2020. He is currently pursuing his Ph.D. in Computer Science at National Schools of Applied Sciences, Sidi Mohamed Ben Abdellah University, Fez, Morocco. His research interests include cybersecurity, AI, machine learning, and neural networks. He can be contacted at email: salim.salmi@usmba.ac.ma.



Lahcen Oughdir    is a Full Professor in the ISA Laboratory, National Schools of Applied Sciences, Sidi Mohamed Ben Abdellah University, Fez, Morocco. He received a Ph.D. in Computer Science from the Sidi Mohammed Ben Abdellah University Faculty of Sciences Dhar El Mahraz, Fez in 2010. His current research interests are applied mathematics, databases, information science, and E-Learning. He can be contacted at email: pmelin@tectijuana.mx.