

Research Article

Enhanced MNB Method for SPAM E-mail/SMS Text Detection Using TF-IDF Vectorizer

Anoushka Dasgupta, Shideh Yavary Mehr*

Department of Computer Science, University of Wisconsin-Milwaukee, Milwaukee, USA

Abstract

Spam, whether in the form of SMS or email, poses significant threats by compromising user privacy and potentially leading to unauthorized access to personal data. In the era of smartphones, where users store sensitive information, the risk of cyber-attack through spam messages is heightened. This research addresses the pressing issue of spam SMS and email detection using a dataset comprising 5574 messages from reputable sources. The collection includes contributions from the National University of Singapore SMS Corpus, Grumble text Website, Caroline Tag's PhD Theses, and SMS Spam Corpus v.0.1 Big. With a meticulous approach encompassing data cleaning, balancing, preprocessing, and exploratory data analysis, the research employs the TF-IDF (Term Frequency and Inverse Document Frequency) vectorizer to enhance the model's ability to capture the importance of individual words. This foundational work sets the stage for evaluating various machine learning models, including Support Vector Machine, Multinomial Naïve Bayes, Decision Tree, Logistic Regression, Random Forest, AdaBoost, K-Nearest Neighbors, XGBoost, Gradient Boost Classifier, Bagging Classifier, and Extra Tree Classifier. Notably, the Multinomial Naïve Bayes model emerges as a standout performer with 100% accuracy and 97% precision in phishing detection. The research introduces an intuitive user interface, facilitating real-time interactivity for model assessment and offering valuable insights for cybersecurity applications. The study contributes to the advancement of robust cybersecurity systems, emphasizing precision and accuracy in spam SMS and email text detection.

Keywords

Spam Detection, SMS, Email, Cybersecurity, TF-IDF Vectorizer, Machine Learning, Multinomial Naïve Bayes

1. Introduction

In recent times, the internet has become an integral aspect of our lives, leading to a surge in email users. However, this increased reliance on email has given rise to challenges posed by unsolicited bulk email messages commonly known as spam. The proliferation of spam emails is primarily driven by their effectiveness as a medium for advertisements. Spam emails, unwanted messages sent to recipients, often result from sharing our email addresses on unauthorized websites. The consequences of spam are manifold, including inbox

clutter, degradation of internet speed, theft of valuable information, manipulation of search results, and a significant waste of time. Despite numerous studies on spam detection methods, accurately identifying spammers and spam content remains a challenging task. The existing approaches often struggle to distinguish spam accurately, and none of them provide a comprehensive understanding of the benefits of each element removed. Despite the advancements in network communication and the utilization of memory space, spam

*Corresponding author: yavaryme@uwm.edu (Shideh Yavary Mehr)

Received: 8 March 2024; **Accepted:** 3 April 2024; **Published:** 28 April 2024



Copyright: © The Author(s), 2024. Published by Science Publishing Group. This is an **Open Access** article, distributed under the terms of the Creative Commons Attribution 4.0 License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

messages continue to pose a threat and are exploited for various attacks. Spam emails, categorized as non-self, are not only used for advertising but may also contain links to phishing or malware-hosting websites aimed at stealing confidential information. To address these issues, various spam filtering techniques are employed. The spam filtering techniques are accustomed protect our mailbox from spam mails.

This study incorporates various ML (machine learning) methods, including SVM (Support Vector Machine), MNB (Multinomial Naïve Bayes), DT (Decision Tree), LR (Logistic Regression), RF (Random Forest), AdB (AdaBoost), KNN (K-Nearest Neighbors), XGBoost, GBC (Gradient Boost Classifier), BC (Bagging Classifier), and ETC (Extra Tree Classifier). Notably, the Multinomial Naïve Bayes model stands out, achieving 100% accuracy and 97% precision in phishing detection. The research introduces an intuitive user interface, enabling real-time interaction for model assessment and providing valuable insights for cybersecurity applications. The study contributes to the enhancement of robust cybersecurity systems, emphasizing precision and accuracy in the detection of spam SMS and email text.

The paper is organized as follows: Section II covers “Background and Related Work”. Implementation is defined in the next section (III). The Results are shown in Section IV. We complete the paper in Section V which outlines our future work.

2. Background and Related Work

The authors in [1] focus on email header features as efficient indicators for identifying and classifying spam messages. These features, selected based on their demonstrated performance in spam detection, are standardized across major email providers such as Yahoo Mail, Gmail, and Hotmail. The objective was to propose a generic mechanism for spam message detection that can be universally applied.

In [2] authors introduce a novel approach based on the frequency of word repetition in incoming emails. The key sentences containing specific keywords are tagged, and the grammatical roles of the words in these sentences are determined. Subsequently, a vector is constructed to assess the similarity between received emails. The classification of emails is carried out using the K-Means algorithm based on the determined vectors.

Another study in [3], delves into the prevalence of cyber-attacks, particularly how phishers and malicious attackers exploit email services to send deceptive messages, leading to potential financial and reputational losses for target users. The study employs Bayesian Classifiers, considering each word in the email and adapting dynamically to evolving forms of spam. This adaptive approach enhances the system’s resilience against new spam patterns.

3. Implementation

The proposed SPAM detection system using machine learning has two phases as shown in Figure 1.

A. Training Phase

In this phase, Spam detection’s training phase involves a thorough process to boost model accuracy. Beginning with exploratory data analysis (EDA) for insights, the next crucial step is featuring extraction. Techniques like lowercase conversion, tokenization, special character removal, stop words and punctuation elimination, and stemming collectively craft meaningful features that encapsulate spam essence. Additional data preprocessing, such as TF-IDF vectorization, refines the dataset, enhancing the model’s pattern recognition. The processed data undergoes evaluation, and the top-performing model, often determined by accuracy, is selected for subsequent phases.

B. Detection Phase

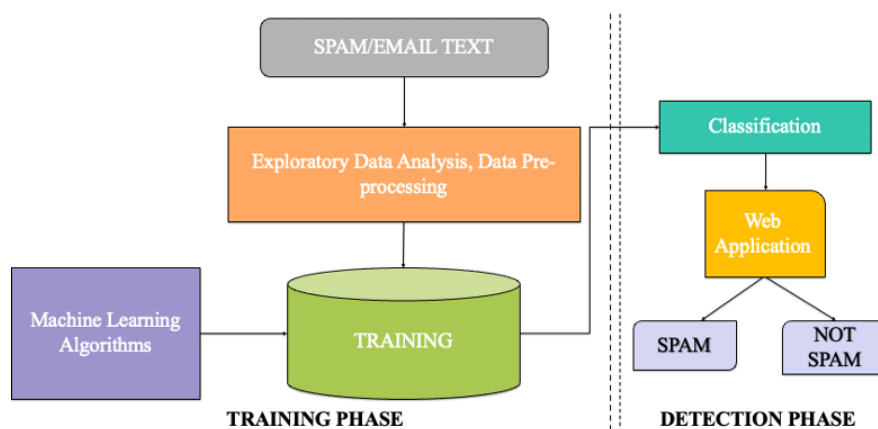


Figure 1. SPAM SMS/Email text detection model.

The detection phase focuses on identifying spam in text messages using features extracted during training. Employing

a sophisticated approach, a voting classifier combines multiple models for enhanced predictive performance. Imple-

mented on a web application, users input text messages for spam detection, receiving classifications as spam or not spam. The system further offers insights into potential threats by analyzing the top contributing words, enhancing transparency in the detection process. This user-friendly solution aims to efficiently identify and mitigate spam in text messages.

3.1. Dataset

The dataset is [4] a collection of 5574 spam and legitimate English text messages from various reputable sources. The dataset includes contributions from the National University of Singapore SMS Corpus, Grumbletext Website, Caroline Tag's PhD Theses, and SMS Spam Corpus v.0.1 Big. Comprising 4827 legitimate messages and 747 spam messages, the corpus is publicly accessible, facilitating further investigation into the realm of spam detection.

3.2. Data Preprocessing

In the data cleaning phase, a meticulous process was employed to ensure the integrity and coherence of the dataset. Initial scrutiny using `df.info()` revealed five columns, with 'Unnamed: 2,' 'Unnamed: 3,' and 'Unnamed: 4' containing a notable number of missing values. To streamline the dataset, these columns were promptly dropped [5]. Renaming the remaining columns to 'target' and 'text' for clarity ensued. The target variable ('ham' or 'spam') was then encoded numerically using the Label-Encoder from scikit-learn. Inspection for missing values and duplicates revealed none in the target and text columns. However, duplicate entries were identified and systematically removed, resulting in a final dataset of 5169 entries and 2 columns ('target' and 'text'). This meticulous data cleaning process laid the groundwork for subsequent analyses and model development. In the exploratory data analysis (EDA) phase, our examination of the preprocessed dataset began with a perusal of the initial entries through `df.head()`, offering an initial glimpse into the dataset's structure. Investigating the distribution of the target variable ('ham' or 'spam') uncovered a notable class imbalance, with 87% classified as 'ham' and 13% as 'spam,' as visualized in a pie chart. This imbalance underscores the need for careful consideration in subsequent analyses. To delve deeper into the dataset's textual features, we extracted statistical insights such as the number of characters, words, and sentences in each entry. Descriptive statistics were computed for the entire dataset and separately for 'ham' and 'spam' categories, revealing nuanced patterns [6]. The utilization of histograms and pair plots further elucidated the distribution and relationships among these features. Particularly, the `numcharacters` and `numwords` features exhibited distinctive patterns for 'ham' and 'spam' categories, suggesting potential discriminative power in these features for classification. Moreover, a correlation heatmap was generated to assess relationships between numerical features.

The resulting visualizations provided a holistic under-

standing of the dataset's structural characteristics. The pie chart underscored the imbalanced nature of the data, setting the stage for subsequent feature extraction through natural language processing (NLP) [7]. Histograms depicting the distribution of 'numcharacters', 'numwords', and 'numsentences' for 'ham' and 'spam' categories were instrumental in revealing variations in text length between the two classes. The heatmap further illustrated correlations between these numerical features. These methodologies and their corresponding results serve as indispensable tools in comprehending the dataset's intricacies, aiding in the identification of potentially discriminatory features for spam classification. The imbalanced class distribution, highlighted by the pie chart, signals the necessity for strategic handling during model development. The histograms and pair plots offer visual insights into the variations in textual characteristics between 'ham' and 'spam,' providing a basis for informed feature selection. The correlation heatmap contributes to the understanding of inter-feature relationships, guiding subsequent steps in model refinement. In essence, this comprehensive approach to EDA is instrumental in shaping the trajectory of the subsequent model development process and underscores the significance of nuanced insights derived from diverse analytical techniques.

We also use different preprocessing approaches for the different classifiers. These approaches are based on their requirement of input data. The following is a brief description of these approaches. The preprocessing approach is based on the following items:

A) Using Term Frequency

Inverse Document Frequency (tf-idf): The frequency of a word's occurrence within a specific document is referred to as Term Frequency. Conversely, the Inverse Document Frequency measures how often a word appears across a collection of documents, as noted in [8]. The TF-IDF method assigns weights to words based on their significance, giving less weight to common words and more weight to rare ones. Our initial steps included eliminating stopwords, uppercase letters, non-alphanumeric symbols, and extraneous punctuation. Subsequently, we normalized similar words (for instance, changing 'desks' to 'desk'). Following this preprocessing, we employed sklearn's Tfidf Vectorizer to transform the refined text into TF-IDF attributes, creating a vocabulary of up to 5000 terms for each entry. This process involves generating a count vector and then a TF-IDF matrix.

B) Using Tokenizer

When processing textual data, a fundamental step is to divide the text into individual words, referred to as tokens. This division is known as tokenization. Keras offers a Tokenizer class that serves to vectorize texts, essentially converting texts into sequences of integers where each integer represents a word's index based on its frequency rank in the dataset. To break down text into a list of words, Keras provides the `text_to_word_sequence()` function. By default, this function separates words based on spaces, transforms all text to low-

ercase, and removes punctuation. The tokenization process is tailored to only include the most frequent words in the corpus, setting a limit at 5000 words for our purposes. After implementing one of these tokenization strategies on our datasets for various classifiers, we modified the representation of the first column. Specifically, we converted 'ham' and 'spam' categories into numerical values, 0 and 1, respectively, using sklearn's LabelEncoder.

3.3. Design

The experimental setup consists of a DELL laptop with an 12th Gen Intel® Core™ i7-12700H CPU (24 MB cache, 14 cores, 20 threads, up to 4.70 GHz Turbo) processor with 16GB of RAM and 512GB of Solid-State Drive. Python is the programming language used. Jupyter Notebook, which supports over 40 programming languages including Python, is used. Jupyter is an interactive web-based interface for the development of notebooks, code, and data management, providing a flexible platform to tailor and organize data science projects.

3.4. Machine Learning Algorithms

3.4.1. Logistic Regression (LR)

Logistic regression is a type of supervised learning. According to the dataset, it is used to predict the categorical dependent variable "label." A categorical dependent variable's output is predicted using logistic regression. Binomial Logistic regression is used to solve classification problems by fitting an "S"-shaped logistic function that predicts two maximum values (0 or 1). The Logistic Regression equation with probability $p(x)$ is as below:

$$P(x) = \frac{e^{a+bx}}{1+e^{a+bx}}; -\infty < x < \infty$$

3.4.2. K-Nearest Neighbor (KNN)

The k-nearest neighbor (KNN) algorithm, a supervised machine learning technique, addresses both classification and regression issues. It is effective for datasets that are clearly separated or non-linear. KNN employs a voting system to assign a class to a new observation, where the majority vote dictates the class assignment. For instance, with K set to one, the class is determined by the closest neighbor. If K is set to ten, the classification is influenced by the ten closest neighbors [9, 10].

3.4.3. Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a popular discriminative classifier employed extensively for classification purposes. This algorithm represents each piece of data as a point within an n-dimensional space, with each feature's value corresponding to a specific coordinate's value. It proceeds to establish a hyperplane that divides the dataset into two distinct groups, each representing a different classification. The points

that are closest to the hyperplane from both groups are positioned as far away from it as possible, effectively maximizing the margin between the two classes.

3.4.4. Multinomial Naive Bayes (MNB)

This classification method utilizes Bayes' theorem, underpinning its approach with the assumption of predictor independence. Essentially, the Naive Bayes (NB) classifier presupposes that the presence of a specific feature within a class is independent of any other feature's presence. This means that, regardless of potential dependencies among features or their relevance to the existence of another feature, the NB classifier treats each characteristic as contributing independently to the overall probability. This assumption allows the Naive Bayes classifier to perform effectively, especially in scenarios involving high-dimensional input data. It is celebrated for its simplicity and robustness. An evolved variant of the NB classifier is the Multinomial Naive Bayes (MNB), which distinguishes itself by discounting the correlation between document length and its classification. It employs a multinomial distribution, making it particularly effective for discrete data, like word counts in text documents. In essence, while the NB classifier operates on the principle of conditional independence among features, the MNB classifier, as a specific iteration of NB, leverages a multinomial distribution for feature handling.

3.4.5. Decision Tree (DT)

Decision Trees (DT) are a supervised learning algorithm often used for classification tasks, effective with both categorical and continuous variables. They initiate by dividing the population into several homogeneous groups based on the most crucial attributes or independent variables. As a non-parametric method, Decision Trees do not require the assessment of outlier presence or the necessity for data to exhibit linear separation.

3.4.6. Random Forest (RF)

Random Forest is a terminology applied to a group of decision trees. This classifier embodies an ensemble learning approach by aggregating multiple decision trees. To classify a new item, a voting mechanism is employed where each tree casts a vote for a particular class. The class receiving the majority of votes determines the final classification label.

3.4.7. AdaBoost (AdB)

AdaBoost, short for Adaptive Boosting, is a composite machine learning technique designed to enhance the efficacy of a classification algorithm by integrating multiple weak classifiers into a single robust classifier. The overall prediction of the boosted classifier is derived from the weighted aggregate of all weak classifiers' predictions. One limitation of this method is its increased model construction time, despite achieving higher accuracy in its predictions.

3.4.8. XGBoost

XGBoost, short for Extreme Gradient Boosting, is a powerful and widely-used machine learning algorithm known for its exceptional performance in predictive modeling tasks. It belongs to the ensemble learning family, specifically gradient boosting, and employs a boosting technique to combine the strengths of multiple weak learners, often decision trees, to create a robust and accurate predictive model. XGBoost introduces regularization terms and parallel processing to enhance its efficiency and generalization capability. Renowned for its scalability, speed, and ability to handle complex datasets, XGBoost has become a popular choice across various domains, including classification, regression, and ranking problems, making it a staple in machine learning pipelines and data science competitions [14, 17].

3.4.9. Extra Trees Classifier (ETC)

ETC is an ensemble learning algorithm that belongs to the Random Forest family. Like Random Forests, ETC builds multiple decision trees during training and outputs the mode of the classes for classification tasks. What sets ETC apart is its extreme randomization strategy, where it selects random subsets of features at each decision node, leading to a more diverse set of trees. This heightened level of randomness enhances the model's robustness against overfitting and makes it particularly effective in handling noisy or high-dimensional datasets. ETC is widely utilized for classification tasks due to its simplicity, efficiency, and ability to deliver competitive results without requiring extensive hyperparameter tuning [16].

3.4.10. Bagging Classifier (BC)

The Bagging Classifier is an ensemble learning algorithm that enhances the performance and stability of base classifiers by training multiple instances on different subsets of the training data. Employing a bagging (Bootstrap Aggregating) strategy, it leverages random sampling with replacement to create diverse subsets for each classifier. In scikit-learn, this meta-estimator can be applied to any base classifier, and its predictions are aggregated through averaging for regression or voting for classification. With parameters like *n* estimators determining the number of base classifiers and *max_samples* controlling the size of training subsets, the Bagging Classifier is a versatile tool for mitigating overfitting and improving generalization across various machine learning tasks [12, 15, 17].

3.4.11. Gradient Boosting Classifier (GBC)

The Gradient Boosting Classifier is a powerful ensemble learning algorithm that belongs to the family of boosting methods. It builds a series of weak learners, typically decision trees, sequentially, with each subsequent learner focusing on correcting the errors made by the previous ones. The model assigns higher weights to instances that are misclassified, creating a strong predictive model through the combination of

multiple weak learners. The algorithm's flexibility and ability to capture complex relationships in data make it well-suited for both classification and regression tasks. Key parameters include the learning rate, controlling the contribution of each tree, and the number of trees (*n* estimators). While computationally more expensive than some algorithms, the Gradient Boosting Classifier often delivers high accuracy and robust performance, particularly when fine-tuned effectively [11-13].

3.5. Execution

The algorithm outlined, encapsulates a comprehensive two-phase approach, leveraging exploratory data analysis, feature extraction, and a voting classifier ensemble during training and detection phases, ultimately enhancing the accuracy and transparency of the SPAM detection system.

Algorithm 1 SPAM Detection System

```

1: procedure TRAINING PHASE
2:   Input: Labeled dataset with spam and non-spam text messages.
3:   Perform Exploratory Data Analysis (EDA).
4:   Apply data cleaning techniques:
      • Convert text to lowercase.
      • Tokenize the text.
      • Remove special characters.
      • Eliminate stop words and punctuation.
      • Perform stemming.
5:   Implement TF-IDF vectorization for feature extraction.
6:   Split the dataset into training and testing sets.
7:   Train multiple machine learning models.
      • Evaluate model performance using accuracy.
      • Select the top-performing model.
8:   Save the selected model.
9:   Output: Trained SPAM detection model.
10: end procedure
11: procedure DETECTION PHASE
12:   Input: Unlabeled text messages for spam detection.
13:   Preprocess the input text using techniques from Training Phase.
14:   Vectorize the preprocessed text using TF-IDF.
15:   Load the pre-trained model.
16:   Use a voting classifier to classify text messages.
17:   Implement the detection system on a web application.
      • Allow users to input text messages.
      • Provide real-time classification results.
      • Analyze and display top contributing words.
18:   Output: Classification results and top contributing words.
19: end procedure

```

Figure 2. SPAM detection algorithm.

The implementation phase leverages Streamlit, a user-friendly Python library, to deploy and interact with the spam detection models seamlessly. Streamlit provides a

convenient platform for creating web applications with minimal coding effort. The user interface is designed to enable users to input text messages for classification as either

spam or not spam. This intuitive interface enhances user experience and allows for real-time interaction with the spam detection functionality.

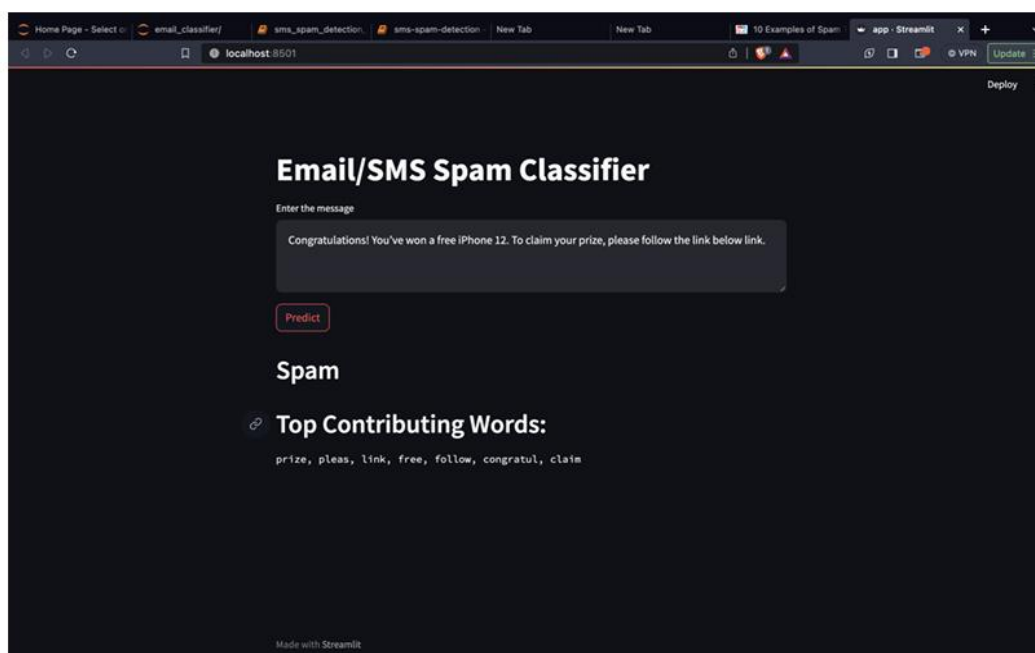


Figure 3. The above screenshot displays an example where the application detects it as spam, also provides the top contributing words supporting the decision.

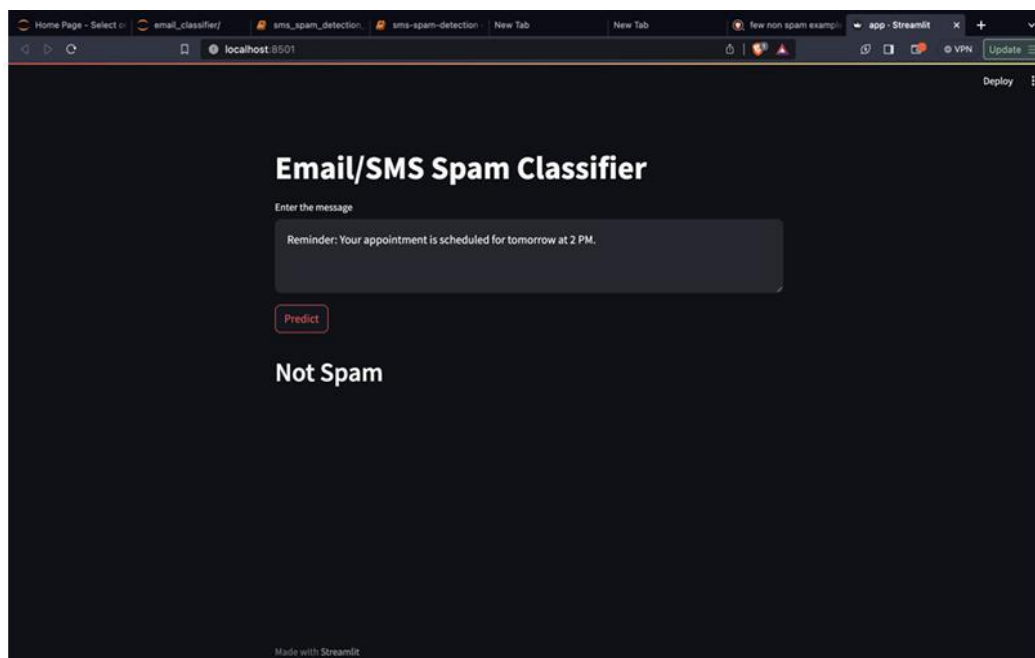


Figure 4. The above screenshot displays an example where the application detects it as spam, also provides the top contributing words supporting the decision.

The Streamlit application incorporates the best-performing models, Multinomial Naive Bayes (MNB) and Extra Tree Classifier (ETC), to ensure reliable and accurate spam detection. Users can input messages, triggering the models to pre-

dict whether the text is spam or not. Additionally, the application offers insights into potential threats by analyzing and displaying the top contributing words, contributing to transparency in the detection process. The implementation using

Streamlit serves as an accessible and user-friendly tool, catering to both researchers and practitioners in the field of cybersecurity. It not only showcases the efficacy of the models but also empowers users to actively engage with and evaluate the spam detection system in a real-world context.

4. Results

The research presents a comprehensive comparison of various machine learning models, including Support Vector Machine (SVM), Multinomial Naive Bayes (NB), Decision Tree, Logistic Regression, Random Forest, AdaBoost, K Nearest Neighbors, XGBoost, Gradient Boost Classifier, Bagging Classifier, and Extra Tree Classifier. Figure 4 visually represents the precision and recall scores of each model, with Multinomial Naive Bayes (MNB) and Extra Trees Classifier (ETC) emerging as the top performers. The evaluation metrics employed in this study encompassed Accuracy and Precision scores, providing a comprehensive assessment of model performance.

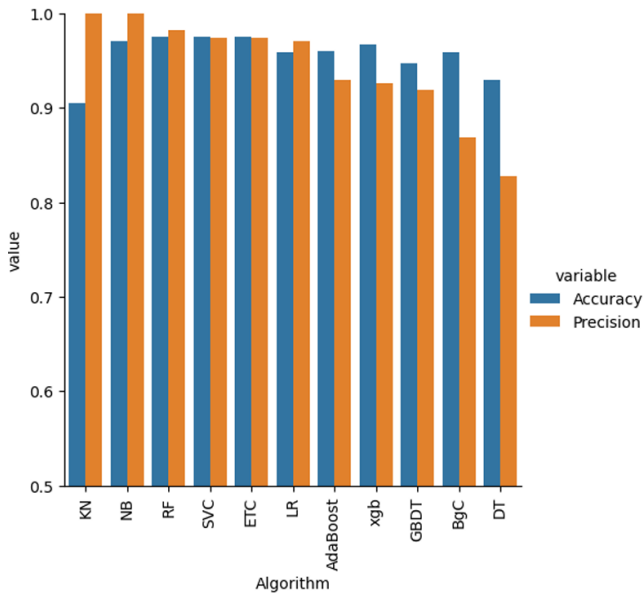


Figure 5. Graph representing the accuracies and precision of each model.

The above comparison graph and table was developed using the accuracy metric as explained below. Accuracy: Accuracy is the ratio of correct predictions out of all predictions made by an algorithm. It can be calculated by dividing precision by recall or as 1 minus false negative rate (FNR) divided by false positive rate (FPR).

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (1)$$

Precision or positive predictive value: Precision is the ratio of true positives over the sum of false positives and true neg-

atives. It is also known as a positive predictive value.

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

Where TP and TN are as follows:

TP: number of true positives, spam text that is classified as such.

TN: number of true negatives, not spam text that are classified as such.

FP: number of false positives, not spam text that are classified as bad.

FN: number of false negatives, spam text that are classified as good.

The Multinomial Naive Bayes is the best model out of all Machine Learning Models used in the proposed system, which is used to detect spam or not spam., Multinomial Naive Bayes with its high precision and high accuracy, is ideal for this classification problem.

Number	Algorithm	Accuracy	Precision
1	KNN	90.5	1.00
2	MNB	97	1.00
3	RF	97.5	0.98
4	SVM	97.5	0.97
5	ETC	97.4	0.97
6	LR	95.8	0.97
7	ADB	96	0.93
8	XGB	96.7	0.92
9	GBDT	94.6	0.92
10	GBC	95.8	0.87
11	DT	92.9	0.83

Figure 6. Table with the accuracy and precision results of each ml algorithm.

5. Conclusion and Future Work

The implemented spam detection system harnesses the power of various machine learning algorithms, such as Support Vector Machine, Multinomial Naive Bayes, Decision Tree, Logistic Regression, Random Forest, AdaBoost, K-Nearest Neighbors, XGBoost, Gradient Boost Classifier, Bagging Classifier, and Extra Tree Classifier. The utilization of exploratory data analysis, feature extraction techniques, and thorough preprocessing, including the application of TF-IDF vectorization, contributes to the system's effectiveness in distinguishing between spam and non-spam messages. Notably, the Multinomial Naive Bayes model emerges as a standout performer, exhibiting high accuracy and precision in phishing detection. The user-friendly interface developed with Streamlit enhances the user experience, allowing interactive assessments of model performance. Evaluation metrics, including accuracy and precision scores, attest to the system's efficiency and discriminatory power. The strategic choice of Multinomial Naive Bayes and

Extra Trees Classifier underscores their exceptional performance in identifying and combating spam in text messages, emphasizing the system's contribution to robust cybersecurity measures.

In the realm of future work, several promising avenues can be pursued to enhance the capabilities of the implemented spam detection system. One compelling direction is the exploration of advanced language representation models, with particular emphasis on transformers like BERT. Investigating how models like GPT (Generative Pre-trained Transformer) or RoBERTa could contribute to improved understanding and detection of nuanced linguistic patterns within spam messages holds significant potential. Additionally, the integration of novel pre-processing techniques, feature engineering methodologies, and ensemble learning strategies could further elevate the system's overall performance. Considering the dynamic nature of spam tactics, ongoing research into explainable AI (XAI) methods will be instrumental in providing transparent insights into the decision-making processes of the model. Furthermore, cross-domain adaptation studies could extend the system's applicability to diverse linguistic contexts and communication channels. Finally, exploring the integration of cutting-edge techniques in natural language processing and machine learning remains a promising avenue for advancing the effectiveness and adaptability of the spam detection framework.

Abbreviations

GPT: Generative Pre-trained Transformer
 MNB: Multinomial Naive Bayes
 ETC: Extra Tree Classifier
 SVM: Support Vector Machine
 DT: Decision Tree
 LR: Logistic Regression
 RF: Random Forest
 AdB: AdaBoost
 KNN: K Nearest Neighbors
 GBC: XGBoost, Gradient Boost Classifier
 BC: Bagging Classifier

Conflicts of Interest

The authors declare no conflict of interest.

References

- [1] H. Najadat, N. Abdulla, R. Abooraig, and S. Nawasrah, "Mobile sms spam filtering based on mixing classifiers," *International Journal of Advanced Computing Research*, vol. 1, pp. 1–7, 2014.
- [2] Y. Yang and S. Elfayoumy, "Anti-spam filtering using neural networks and Bayesian Classifiers," 2007.
- [3] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, "Contributions to the study of sms spam filtering: new collection and results," in *Proceedings of the 11th ACM symposium on Document engineering*, 2, pp. 259–262.
- [4] "Uci machine learning repository: Sms spam collection dataset." <https://www.kaggle.com/uciml/sms-spam-collectiondataset>
- [5] S. J. Delany, M. Buckley, and D. Greene, "Sms spam filtering: Methods and data," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9899–9908, 2012.
- [6] I. Murynets and R. P. Jover, "Analysis of sms spam in mobility networks," vol. 3, 2013.
- [7] P. P. Chan, C. Yang, D. S. Yeung, and W. W. Ng, "Spam filtering for short messages in adversarial environment," *Neurocomputing*, vol. 155, pp. 167–176, 2015.
- [8] T. K. Kriti Agarwal, "Email spam detection using integrated approach of naïve bayes and particle swarm optimization," 2018.
- [9] D. Sahoo and S. C. Liu, Chenghao and Hoi, "Malicious url detection using machine learning: A survey," vol. abs/1701.07179. *arXiv*, 2017.
- [10] M. Khonji, Y. Iraqi, and A. Jones, "Phishing detection: A literature survey," vol. 15, no. 4. *IEEE*, 2013, pp. 2091–2121.
- [11] A. S. Manjeri, K. R., A. M. N. V., and P. C. Nair, "A machine learning approach for detecting malicious websites using url features." *Elsevier*, 2019, pp. 555–561.
- [12] R. Patgiri, H. Katari, R. Kumar, and D. Sharma, "Empirical study on malicious url detection using machine learning," in *Distributed Computing and Internet Technology*. Springer International Publishing, 2019, pp. 380–388.
- [13] J. Khan, H. Mohammd, Q. Niyaz, V. K. Devabhaktuni, S. Guo, and U. Shaikh, "A hybrid machine learning approach to network anomaly detection," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. *IEEE*, 2019, pp. 0347–0352.
- [14] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious urls: an application of large-scale online learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. Association for Computing Machinery, 2009, p. 681–688.
- [15] B. Eshete, A. Villafiorita, and K. Weldemariam, "Binspect: Holistic analysis and detection of malicious web pages," in *Security and Privacy in Communication Networks*. Springer, 2013, pp. 149–166.
- [16] S. Purkait, "Phishing counter measures and their effectiveness – literature review," vol. 20, no. 8. Emerald Group Publishing limited, 2012, pp. 382–420.
- [17] A. Singh, "Dataset of malicious and benign webpages." *Mendeley Data*, 2020.