Exploring Machine Learning Classifier Models For Spam Detection

Kiera Conway
Dakota State University
Seattle, Washington
Kiera.Conway@trojans.dsu.edu

Robert Chavez
Dakota State University
Bakesfield, California
robsafetyconsulting@gmail.com

***Abstract*—This report presents a comparative analysis of three machine learning classifier models for spam detection in Short Messaging Service (SMS) messages. We trained and tested K-Nearest Neighbors (KNN), Decision Trees (DT)/ Random Forests (RF), and Naïve Bayes (NB) on a publicly available Kaggle dataset. Our methods discuss dataset preparation using natural language processing (NLP) preprocessing and tokenization techniques, feature engineering with regex, feature extraction using CountVectorizer, and our model selection process. Each model was tested and compared using various performance metrics including accuracy, confusion matrices, precision, recall, and F1-scores. These results are assessed using visualizations such as ROC, Precision-Recall Curve, and Bar Plots. The results show that all three models performed well, with Naive Bayes having the highest accuracy of 98.56%, followed closely by Decision Tree/Random Forest with 97.85%, and K-Nearest Neighbors with 95.05%. Our results display the effectiveness of various machine learning algorithms for spam detection and provide insights into the strengths and weaknesses of each approach, with hopes to help guide future research in this area.**

Keywords: Spam, Ham, Naïve Bayes, Decision Trees, Random Forest, K Nearest Neighbors.

I. Introduction

A. Background

The rapid advancement of technology has transformed global communication, enabling people to instantly exchange messages across the globe through mediums such as email or SMS. However, this evolution has also resulted in an increase in spam and malware, which are often used by marketers or threat actors to send unsolicited digital communication or harm computer systems. Spam is unwanted electronic correspondence that is sent out in bulk, while malware refers to any software that intends to harm computer systems or gain unauthorized access to sensitive information, such as viruses, worms, and other code-based entities that infect a host with malicious intent. Threat actors leverage spam techniques, such as using mass emails or Short Messaging Service (SMS) messages, to send phishing attempts aimed at acquiring personally identifiable information (PII), credit card details, and other unauthorized access to computer systems.

To combat these threats on a global scale, the field of computer science and artificial intelligence has leveraged a sophisticated tool: Machine Learning. Machine Learning (ML) was created to help process large amounts of data generated by individuals, most of which are too vast for human processing.

B. Understanding Machine Learning

The ability of ML algorithms to automatically learn from data, identify patterns, and make predictions "without being explicitly programmed" [1] is what makes it an effective solution for managing and utilizing large amounts of data. In other words, instead of being given specific instructions for what to do with the data, ML algorithms are designed to learn from patterns in the data itself. Thus, by giving these algorithms large datasets to analyze, they can identify underlying patterns and make predictions based on new, unseen data. This adaptability is particularly important in the context of spam and malware, where ML algorithms can be trained to recognize patterns and identify potential threats, such as the aforementioned phishing attempts via mass emails or SMS messages.

C. Machine Learning in Cyber Security

In addition to its effectiveness in detecting and preventing spam and malware, the ability to learn and adapt to new data has made ML an invaluable tool for Cyber Security as a whole. The constantly evolving nature of cyber threats requires a dynamic and flexible solution, which traditional rule-based systems struggle to provide. Historically, rule-based systems have been the primary method of detecting threats; however, these methods often struggle to identify new and emerging threats as they rely on pre-defined rule sets that can be easily exploited. On the other hand, ML algorithms can create new rules as necessary and adapt to new and emerging threats. As new threats are constantly emerging, ML enables Cyber Security experts to stay ahead of the curve and better protect individuals and organizations from the growing threat of cyber-attacks.

Spam messages are a great example of these cyber-attacks as they pose a significant threat to Cyber Security, with many malicious intentions such as illegally accessing confidential data such as passwords and other important identifiers, or spreading malicious software with harmful links or attachments. [2] SMS messages are no exception to the threat of spam, and detecting such messages in this form of communication is crucial for protecting individuals and organizations. Due to the limitations of rule-based systems, they have traditionally struggled in the past to detect spam and other cyber threats, as they rely on suspicious patterns for detection, such as misspelled words, malicious links, and other anomalies. This is where machine learning algorithms come into play, as they can adapt and create new rules as necessary to stay ahead of the constantly evolving nature of cyber threats. Thus, it becomes trivial for an attacker to modify message content in hopes of circumventing rules, as the rules are constantly updating.

D. Research Objective and Scope

Our study aims to make a valuable contribution to the discussion on the most effective machine learning algorithm for identifying and filtering out spam messages in SMS communication. To achieve this, we will explore different methods of detecting spam using an open-source dataset from Kaggle and the Python programming language for preprocessing, training, testing, and analysis. Specifically, we will focus on three primary models, including Naïve Bayes (NB), Decision Trees (DT)/Random Forests (RF), and K-Nearest Neighbors (KNN), and provide a comprehensive comparison of their performance metrics to determine the best approach for accurate classification. Through our rigorous testing and analysis, we aim to contribute to the ongoing efforts to improve cyber security and help prevent spam messages from causing harm.

E. Related Work

During our research we did various literature reviews on similar research studies related to machine learning and cyber security. Many of these research articles show us different variations of spam removal by using a different combination of feature extraction vectorizers and machine learning models. One of the studies that particularly was of interest was a study from [3].

In this study we seen that this group of researchers utilized Convolutional Neural Network, Support Vector Machines, Artificial Neural Network, and Decision tree models. Convolutional Neural network stood out to use in this research since this type of model is used for image classification data however, in this instance it performed well with an accuracy rate of 99.10. The second model to note was the performance attained by the Artificial Neural networks which demonstrated an accuracy score of 98.39.

The results of this study will allow us to better understand why certain approaches using models such Convolutional Neural Networks may or may not offer better than expected results when taking accuracy and performance into consideration. Using what we learned from these studies we hope to try similar methods of detecting spam while trying other methods that are not commonly used for spam detection.

II. Methods

Before we begin training our models, we much first preprocess the data to ensure consistency and compatibility. We will procedurally analyze and modify our dataset to increase model quality and their resulting analytic scores, while avoiding over-generalization and over-fitting. This transformed dataset will then be trained and tested across each model individually.

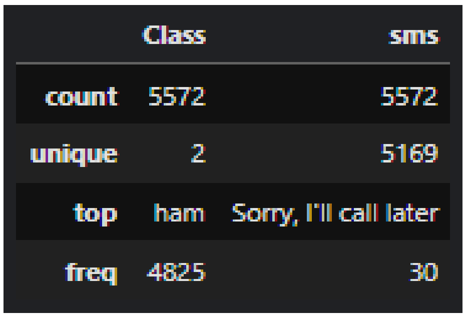
A. Introduce Dataset

We have selected the publicly available Kaggle dataset, “Spam (or) Ham,” to train and test our spam classification models [4]. It is important to note that this dataset is a condensed version of The University of California, Irvine’s (UCI) ‘SMS Spam Collection Dataset’ [5]. The original message collection was consolidated from various public sources, including 425 spam messages from Grumbletext, 3,375 ham messages from NUS SMS Corpus (NSC), 450 ham messages from Caroline Tag's PhD Thesis, and 1,002 ham and 322 spam messages from the SMS Spam Corpus v.0.1 Big [5]. The version we are using was selected due to accessibility and minor pre-processing, as two messages were formatted incorrectly and resulted in missing values. As such, our version contains 5572 Short Message Service (SMS) messages, 5169 of which are unique. Each message is split between two columns: ‘Class’ which identifies whether it is spam or ham, and ‘sms’ which contains the plain-text version of the message.

B. Data Analysis

To prepare our data for use in our ML algorithms, we first needed to review and analyze a series of properties and decide which, if any, transformations were required. We began by importing the comma-separated values (CSV) data file using the python library, Pandas. Pandas is an open-source python library which is commonly used in machine learning as it helps organize, manipulate, and analyze complex tabular data. Our process begins through a series of steps, including observing general information, checking for extraneous null values, viewing various observations, and analyzing statistical information.

We started by viewing the general and statistical information to confirm that each column, ‘Class’ and ‘sms,’ contained 5572 non-null entries. As shown in Fig. 1, each column not only contained this expected value, but the column values for the ‘count’ row were equivalent. If this were not true, it would mean we had incomplete or missing data which would need to be trimmed or modeled accordingly.



	Class	sms
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

FIGURE I – STATISTICAL ANALYSIS OF DATA

Similarly, we needed to check if there were any unnecessary null values anywhere in our dataset which could further negatively impact the ML algorithms. These are important steps as they are examples of noise which can cause overfitting and potentially “result in more complex models that miss the true pattern” [6].

Fig. 1 also displays statistical information for both the 'Class' and 'sms' columns, which includes the number of all unique observations, the most common value, and the frequency of the most common value. We use this information to assess the completeness of our data and verify the values contained are as expected and acceptable.
As we can see in the 'unique' row, the 'Class' column has the expected value of 2 for the number of unique observations, with 'spam' and 'ham' being these corresponding values. However, the 'sms' column indicates an unexpected value of 5169 for the number of unique observations. As this value is less than the expected 5572, we must conclude that there are identical messages present in the dataset. The remaining top and frequency values confirm this hypothesis by showing that the most frequent message, "Sorry, I'll call later," occurs 30 times. As $5572\text{-}5169 \neq 30$, we must further conclude that

there are additional repeated messages. Finally, the 'Class' column indicates that most messages are classified as `ham`, with 4825 occurrences, which leaves the remaining 747 messages to be categorized as `spam`.

The last step before transforming our data involves observing a few examples and their corresponding features. This will provide additional context which enables us to understand how the information is formatted and deduce if transformation is necessary. During this step, we were able to conclude that the messages were unstructured, containing a mix of lower and upper case, punctuation, and stop words. Before we could continue with testing and training our models, these messages will require natural language preprocessing (NLP). Furthermore, our observations show that the identifying classes are also in categorical form, e.g. ‘ham’ and ‘spam’. Since many ML algorithms execute complex mathematical computations, categorical data is not ideal and will require both columns to be converted to an equivalent numerical form, e.g. ‘0’ for ‘ham’ and ‘1’ for ‘spam’.

C. *Dataset Preparation*

The conversion to numerical form for Class is a simple process as there are only two categories, ‘spam’ and ‘ham.’ To achieve this conversion, we used the LabelEncoder function from the Python library sklearn, which can be used to encode categorical labels as integers that are more easily used as input for ML models. This function enabled us to create a new column ‘is_spam’ where we mapped each classification to its corresponding encoded values; each ‘spam’ classification was encoded as a 1 for true, and each ‘ham’ classification was encoded as a 0 for false.
However, the conversion for the ‘sms’ column is more complicated due to its textual nature and the nuances between legitimate text and spam. First, it is important to consider that there are many variations between the two, such as excessive punctuation, web addresses, phone numbers, or promotional content, which often result in longer, more complex messages. For example, Fig. 2 shows the relationship between message lengths in both spam (red) and ham (green) messages.

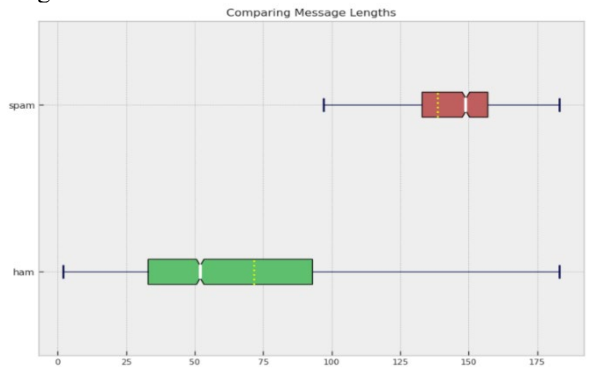


FIGURE II – COMPARING MESSAGE LENGTHS (OUTLINERS NOT SHOWN)

As shown by the dotted yellow line in each box plot, spam messages average length of ≈139 is significantly higher than ham messages average length of ≈71. While ham messages come in a larger variation of sizes, Fig. 2 shows that the entire interquartile range (IQR) of all ham messages in our dataset have message lengths that fall below the minimum value for spam messages, excluding outliers. This suggests that message length can be a useful feature for distinguishing between spam and ham messages, with longer message lengths being a potential indicator of spam. As such, we performed feature extraction by creating a new column in our data, sms_len, to capture the message lengths.

Although message length can provide valuable insights into distinguishing between spam and ham messages, it is not the only feature that can help accurately classify these messages. In addition to length, the words used in a message are also critical for effective classification. However, not all words carry significance within the context for a message. For example, consider the sentence "I went to the store to buy milk." We could remove the words "I," "to," and "the," and still convey the original message meaning. These removed words are called 'stop words’ and they are removed during preprocessing to reduce noise. Since they occur “very frequently and their presence doesn't have much impact on the sense of the sentence,” the noise they create can cause bias within a model [7]. Thus, the remaining sentence still contains the necessary useful information while being more efficient for the ML algorithms to analyze. To remove these words from out dataset, we imported the stopword corpus provided by the Natural Language Toolkit (NLTK). This corpus includes a vast collection of the most frequently occurring words, allowing us to remove them efficiently without manually creating a list.

In addition to removing stop words, we also removed phone numbers and replaced them with a special identifier. Since phone numbers are often unique and do not provide additional context regarding the text's sentiment or topic, we found that engineering a feature identification for phone numbers greatly improved our models’ scores. By replacing them with a common ID, we could remove their impact on the model's output while still maintaining a record of their occurrence. As such, when this textual data is converted to a numerical format, they have a common identifier which ensures that they will all match regardless of their original format; this prevents them from being treated as separate and unrelated features. Not only does enable the model to identify patterns and correlations between text messages that contain phone numbers more easily, but this also decreases the number of unique features in the dataset, thus reducing the dimensionality.

Another important step in our preprocessing was the removal of punctuation. Instead of a basic punctuation removal process, we opted to replace all occurrences with a space. This was important because many spam messages include links for victims to follow, and if the punctuation was removed, the links would be joined together and would not be identified as separate words. For example, by replacing the punctuation with a space, ‘www.here.com' becomes [‘www’, 'here', 'com'] instead of 'wwwherecom'. When words are converted to a numerical ID later, this will enable the common identification of the top-level domains; if we were to simply remove the punctuation, there would be no commonality between ‘here.com’ and ‘there.com,’ despite both originally containing ‘.com.’ This approach to punctuation removal is more effective in capturing the underlying meaning of text while also retaining essential information that can impact the model's performance.

In the preprocessing phase, it is common to convert all messages to lowercase to ensure consistency in the text.

However, we found that certain types of spam messages intentionally use uppercase letters to attract user attention. By leaving the original letter casing, we are able to provide the algorithm with more information to differentiate between spam and ham messages. For instance, if we converted the ham message “call me now” and the spam message “CALL NOW!” to lowercase, they would look identical (‘call’,’now’) after preprocessing. By preserving the casing, our models are able to recognize the all-caps text as a possible spam message.

Overall, our preprocessing decisions aimed to reduce noise in the dataset while still preserving essential information that can impact the model's output. By removing stop words, engineering a phone number ID, replacing punctuation with a space, and performing lemmatization to reduce words to their base form, we were able to create a more accurate representation of the underlying meaning of each message. Once the messages have been preprocessed in this manner, they are saved to a new column named ‘sms_clean’ and are almost ready to be transformed into numerical features for input into the machine learning algorithms.

D. Pre-Training Setup

Before we can transorm our messages, we must first split our training and testing data to avoid overfitting and ensure our models are able accurately predict new, unseen data. Furthermore, if we do not conduct a train/test split prior to the conversion, we could inadvertently introduce bias to our model as it would have access to information from the testing set during the training phase, which it should not have access to. As such, we will begin by defining our variables, splitting the training and testing data, and then finish executing our data transformation.

Since the goal of our algorithms is to classify whether a message is spam or ham, and this classification is dependent on each message, the discrete value from ‘is_spam’ becomes our dependent variable, y, and the string value from ‘sms_clean’ becomes our independent variable, X. After identifying and assigning our variables, we split the data into training and testing sets using sklearn's train_test_split function, allocating 75% for training and 25% for testing. To verify our split was successful, we compared the shapes of our new sets to confirm that our observation and feature values matched. Our training set (X_train and y_train) contained 4179 observations and 1 feature, while our testing set (X_test and y_test) contained 1393 observations and 1 feature. We can verify that these values are correct by noting that 75% of 5572 is 4179 (5572 * 0.75 = 4179) and 25% of 5572 is 1393 (5572 * 0.25 = 1393). This split ensures that our model is able to accurately predict new, unseen data without being biased by information from the testing set during the training phase. We can now proceed with data transformation after the split to avoid overfitting.

Our method of categorical conversion utilizes sklearn’s CountVectorizer, which utilizes our tokenized messages and maps each one to a distinct numerical identification, which it then stores in a sparse matrix. It is important to note that word order is disregarded in this approach and the main focus is on frequency. This technique is commonly known as the bag-of-words (BOW) approach where text data is represented as a "bag" of tokens. For example, consider our data contained the following messages: 'The quick brown fox' and 'The lazy dog'. If we disregarded stopwords and lemmatization, this approach would produce the matrix shown in Table 1.

TABLE I – VECTORIZER/BOW EXAMPLE.

	the	**quick**	**brown**	**fox**	**lazy**	**dog**
1	1	1	1	1	0	0
2	1	0	0	0	1	1

Since our dataset is quite large, this method produced a sparse matrix that is 4179 rows × 6859 columns for the training set, and 1393 rows × 6859 columns for the testing set. Since the row values still match our dependent variables, y_train and y_test, we can confirm the transformation maintained a 75:25 split and was a success. As our data has now been split and transformed successfully, we convert it to a numpy array, a ML preferred numerical optimized data structure, and begin training our models.

III. MODELS

After analyzing and preprocessing our dataset, we began training and testing our selected ML algorithms to analyze and classify the messages. For our analysis, we chose three popular classification algorithms: Naive Bayes, Decision Tree/Random Forest, and K-Nearest Neighbors (KNN). Naive Bayes is a probabilistic algorithm which “naively assumes independency between features,” Decision Trees create a hierarchical “a tree-like structure” structure of rules to classify data points, while Random Forests extend Decision Trees by combining multiple trees to improve performance, and KNN classify a data point based on the most common label within a predefined k-nearest neighbors proximity [8]. Each algorithm has unique strengths and weaknesses, and we analyzed their corresponding prediction metrics to explore how they performed on our dataset. By comparing these results, we were able to determine the most accurate and efficient algorithm for spam/ham classification.

A. Naïve Bayes

Naive Bayes is a probabilistic algorithm that employs Bayes' theorem, a mathematical probability formula based on prior evidence. In the case of spam detection, Naive Bayes computes the probability of a message being spam or ham based on the presence or absence of specific words in the message. This classifier is often considered naïve because it must assume all features, or words in the message, are equally important and independent of each other. While this may not always be the case, Naïve Bayes remains a popular method for many text classification tasks, including spam filtering. It can also be a beneficial simplification method when dealing with large, high-dimensional datasets like ours. When combined with our BOW approach, Naive Bayes can effectively capture the frequency of each word in the message and use it to make predictions.

To implement Naive Bayes, we used sklearn's MultinomialNB, which is specifically designed for text classification tasks like spam filtering. After fitting our model to the training data, we made predictions on the test data and used these to create a confusion matrix and calculate the accuracy score. Our confusion matrix shows that we correctly classified 1197 ham messages and 176 spam messages, with only 10 false positives and negatives. The accuracy score of

0.9856 indicates that our Naive Bayes model performed very well on this task.

B. Decision Tree

Decision Trees create a hierarchical structure of fixed rules to classify data points. The algorithm begins at the root of the structure and recursively splits the data based on the decisions made at the internal feature nodes, where each node corresponds to a decision or question about a feature, and each edge represents a possible answer. The resulting values of these decisions are indicated by edges, which lead to other internal nodes or leaves, which represent a final classification [9]. This node-edge traversal repeats until the algorithm reaches a leaf or other stopping criterion, such as a maximum depth or a minimum number of samples per leaf. At each leaf, the algorithm outputs the predicted class label for the data point that was passed down through the decision tree. Furthermore, Random Forests are an extension of Decision Trees, in which they combine multiple trees to improve performance and reduce overfitting. Instead of using a single tree, the algorithm creates a collection of trees by randomly selecting a subset of features and data points for each tree. Each tree performs an its own classification and the final prediction is then determined by combining all of the predictions, such as by taking the majority vote [9].

For our code, we used the sklearn functions DecisionTreeClassifier and RandomForestClassifier to create and fit multiple models, including a standard tree model, a model with entropy, a pre-pruned model with entropy, and a Random Forest model. After fitting and training each model, we evaluated their performance through a series of predictions on the test set, which were then used to calculate a confusion matrix and accuracy score. The accuracy score is the proportion of correct predictions out of all predictions made. Table 2 shows the layout of a confusion matrix, which provides the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each model.

TABLE II – CONFUSION MATRIX LAYOUT

	Predicted Value [0]	**Predicted Value [1]**
True Value [0]	True Negative (TN)	False Positive (FP)
	Predicted HAM Correctly	Predicted SPAM, was HAM
True Value [1]	False Negative (FN)	True Positive (TP)
	Predicted HAM, was SPAM	Predicted SPAM Correctly

Our initial Decision Tree Classifier, which utilized function defaults, achieved an accuracy score of 97.84% with 162 TP's and 6 FP's, indicating that the model incorrectly classified 24 FN. While this is within an acceptable range, it was lower than the accuracy score obtained by the Naive Bayes model (98.56%). As such, we experimented with different hyperparameters, including the criterion and maximum depth by pre-pruning, to increase model metrics. Since our default model utilizes the Gini impurity criterion by default, we selected the entropy criterion in an attempt to improve the accuracy. The entropy criterion differs from the Gini impurity in that it measures the level of information gained by each split in the decision tree; alternatively, Gini impurity measures the probability of a data point being misclassified. While both criterion methods are used to determine how the data should be split, entropy tends to create more balanced trees, while Gini impurity tends to have faster computation. Our entropy model performed equally at identifying TP (162 correct), but it was much less accurate at identifying TN (1197 correct vs 1201 correct). As such, the accuracy score for this model decreased to 97.55% and we continued experimenting in an attempt to recreate a model which increased accuracy without a loss in TN or TP.

Next, pre-pruning was implemented to limit excess tree growth and prevent overfitting. Unfortunately, pre-pruning the Decision Tree classifier was still unable to improve overall accuracy and scored 97.63%. By limiting the tree from over-fitting, this model became more successful than our Entropy model, and the same as our original model at identifying Ham messages (1201 correct vs 1201 correct in original model vs 1197 in entropy model). However, this model was less successful than both other models at identifying spam messages (159 correct vs 162 correct in both our original model and entropy models). Since this model increased TN compared to our entropy model, but decreased in TP, the accuracy score was between our Entropy and Gini models at 97.63%. As the aim was to increase our overall accuracy and without sacrificing TN or TP, we attempted a random forest next.

Random Forests provided the highest accuracy score between the decision trees, scoring a 97.98%. By using multiple decision trees to make predictions, this model became more successful than all of our other decision tree models and achieved the highest TN values with no FP. While the TP score predicted was less than the other models, the TN increased significantly more than the TP decreased; thus, random forests provide the highest accuracy score - increasing from our original 97.84% to 97.98%. A summation of these values shown in Table 3.

TABLE III. COMPARISON OF DT/RF METRICS

Model Name	**Accuracy Score**	**TP**	**TN**	**FP**	**FN**
Gini (Standard) Model	0. 978464	162	1201	6	24
Entropy	0.975592	162	1197	10	24
Pre-Pruned Model	0.976310	159	1201	6	27
Random Forest	0.979899	158	1207	0	28

C. K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a classification algorithm that determines the class of an observation based on its proximity, defined as the Euclidean distance or straight line between two points, to its nearest neighbors within a region. [9] The value of k represents the number of neighbors to consider and may be calculated multiple ways. A common method for selecting k is to calculate the square root of the number of observations in the training set, as used in our initial KNN model. However, as shown in Table 4, when we observe the testing and training metrics of our dependent variable, y, we can see that the majority of observations are ham and contain the value '0'. When there is this imbalance of data, using the square root method to calculate k may result in a bias towards the majority class. This is because the majority class will have more neighbors and therefore influence the classification more than the minority class. As a result, our KNN square root model scored a below average accuracy of 86.64%, which suggests that we may need to explore alternative methods for calculating k.

TABLE IV – TESTING AND TRAINING METRICS FOR Y

	Testing Set y_test	**Training Set y_train**
count	1393.000000	4179.000000
mean	0.133525	0.134243
std	0.340263	0.340954
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	1.000000	1.000000

To address this bias, we implemented the Grid Search optimization method to find the optimal k value for our KNN model. Grid Search is an "automated hyperparameter-tuning method" that uses a k-fold cross-validation process to "brute-force" through a range of parameters using a nested for-loop approach to find the optimal combination for a given model. [10] The dataset is divided into equal subsets for each iteration of the loop, with one subset reserved for training and the rest for testing.
The average accuracy score is then calculated across all cross-validation folds, and the best k value is used to train our new model. To accomplish this, we employed the GridSearchCV function from the sklearn library and defined an upper limit of 15 for the search space due to its computationally expensive nature. While this technique improved our KNN accuracy score by 8.39% to 95.04%., successfully classifying 117 instances of spam, it remains limited in its Spam v Ham classification abilities compared to other classification algorithms.

Our models show that due to the high computational cost and sensitivity to bias, KNN is unsuitable for large datasets with many features, such as those typically encountered in spam filtering. For example, since KNN uses the Euclidean algorithm to calculate the distance between observations, As the number of features increases, the distance calculation becomes more "computationally complex," reaching "exponential time [and] could lead to computational explosions" [11]. This can result in a curse of dimensionality, where the increase of features disproportionately and dramatically reduces the model's performance.

IV. COMPARE RESULTS

The performance of each model was evaluated in terms of accuracy, precision, recall, and F1-score. Accuracy measures the overall average of correct predictions by dividing correct by total predictions $(\mathrm{TP} + \mathrm{TN})/(\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN})$, precision calculates the proportion of true positive predictions out of all positive predictions $(\mathrm{TP})/(\mathrm{TP} + \mathrm{FP})$, and recall determines the proportion of true positive predictions out of all actual positive cases in the dataset $(\mathrm{TP})/(\mathrm{TP} + \mathrm{FN})$. Lastly, the F1-score is the harmonic mean of precision and recall $2 \times \left(\frac{Precision \times Recall}{Precision + Recall}\right)$, and is used to evaluate a model's performance in predicting both positive and negative cases. Based on these metrics, the Naive Bayes model performed the best with an accuracy of 0.9856 and F1-score of 0.9856, followed closely by the Decision Tree and Random Forest models with accuracy scores above 0.97 and F1-scores around 0.98. The KNN with Square Root model had the lowest accuracy and F1-score, indicating poor performance in predicting both positive and negative cases. A summation of these values are shown below in Table 5.

TABLE V. MODEL METRICS

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Naive Bayes	0.9856	0.9856	0.9856	0.9856
Decision Tree	0.9785	0.9783	0.9785	0.9784
Decision Tree with Entropy	0.9756	0.9752	0.9756	0.9754
Decision Tree with Pruning	0.9763	0.9761	0.9763	0.9762
Random Forest	0.9799	0.9804	0.9799	0.9801
KNN with Square Root	0.8665	0.7508	0.8665	0.8045
KNN with Grid Search	0.9505	0.9531	0.9505	0.9518

Additionally, we evaluated the performance of each model using ROC (Receiver Operating Characteristic) and PRC (Precision-Recall Curve). The ROC model plots recall $(TP/(TP + FN))$ against the false positive rate $(FP/(TN + FP))$, and is used to determine how well the model can distinguish between two classes. Alternatively, the PRC plots precision $(TP/(TP + FP))$ against the recall $(TP/(TP + FN))$ and is used to determine how well a model can identify positive samples while minimizing false positives. The area under each curve provides an overall measure of model performance. As we can see in ROC graph in Fig. 3, the Random Forest model had the highest area under the curve (AUC) at 0.998, indicating that it has the highest recall while keeping the false positive rate low. The Naive Bayes and Decision Tree with Pruning models also had high AUC scores, suggesting that they also performed well in classification. The KNN with Square Root model had the lowest AUC at 0.832.

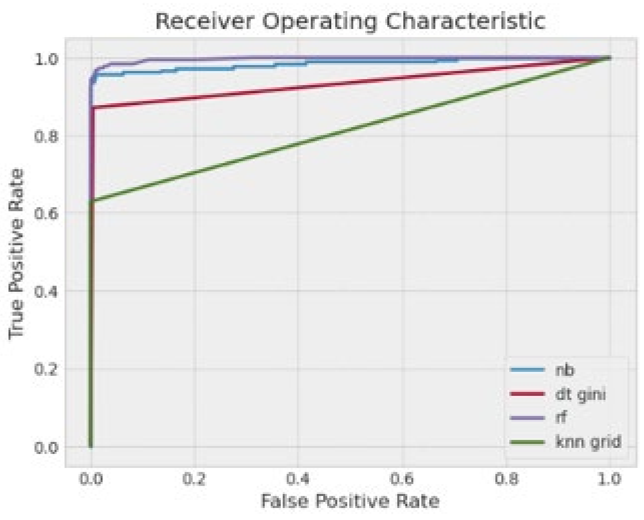


FIGURE III: ROC GRAPH FOR ALL MODELS

As shown in the PRC graph in Fig. 4, Naive Bayes had the highest AUC at 0.990, followed by Random Forest and Decision Tree with Pruning, with the KNN with Square Root model last with an AUC at 0.524.

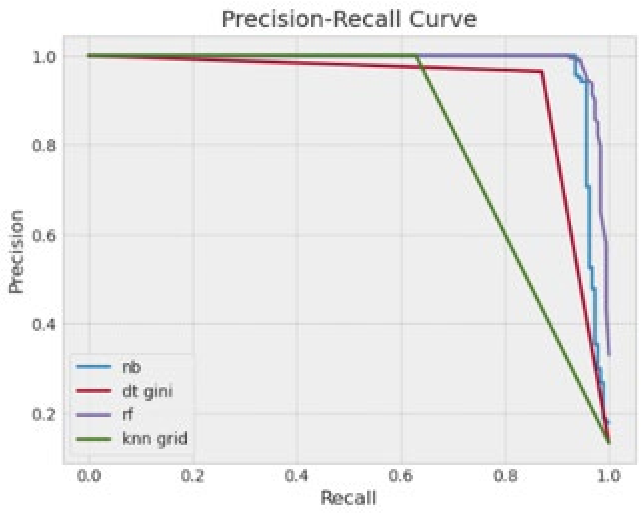


FIGURE IV. PRC GRAPH FOR ALL MODELS

The high AUC scores for Naive Bayes, Random Forest, and Decision Tree with Pruning in both curves indicate that these models are effective in distinguishing between classes and accurately identifying positive samples. On the other hand, the low AUC score for KNN with Square Root in both curves suggests that it may not be as effective in classification as other models. Overall, evaluating a model's performance using both ROC and PRC graphs can provide a more comprehensive understanding of how well it performs in classification tasks.

V. CONCLUSION

In conclusion, our project evaluated six ML classifier models for their effectiveness in detecting spam emails. Based on our evaluation, the Naive Bayes model performed the best in terms of accuracy, scoring 0.9856. Additionally, the Decision Tree models, particularly with pruning or entropy, also performed well in accurately classifying spam emails with high accuracy and precision. The Random Forest model had the highest AUC scores and was effective in distinguishing between spam and non-spam emails with high recall and low false positives. The KNN model with grid search also performed well with high accuracy and precision. However, the KNN model with square root had lower AUC score and lower precision, suggesting it may not be as effective as other models. The importance of spam detection cannot be overstated in today's world of cyber security, where the protection of sensitive information and prevention of cyber-attacks from threat actors are paramount. By utilizing effective machine learning models for spam detection, we can improve our email filtering systems and better protect our systems and data from potential threats. As such, we hope this study serves as a useful contribution towards advancing spam detection techniques and enhancing cyber security measures.

REFERENCES

[1] S. Brown, "Machine Learing, Explained," 21 April 2021. [Online].Available:https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained#:~:text=Machine%20learning%20is%20a%20subfield,learn%20without%20explicitly%20being%20programmed.

[2] T. A. F. F. D. M. Son Dinh, "Spam campaign detection analysis,and investigation," Digitial Investigation, March 2015. [Online]. Available: • https://www.sciencedirect.com/science/article/pii/S1742287615000079?ref=pdf_download&fr=RR-2&rr=7b77774e197530dd#page=10&zoom=100,0,0.

[3] M. Gupta, "A Comparative Study of Spam SMS Detection Using Machine Learning Classifiers," August 2018. [Online]. Available: • https://www.researchgate.net/publication/328907962_A_Comparative_Study_of_Spam_SMS_Detection_Using_Machine_Learning_Classifiers.

[4] A. S, "Spam (or) Ham," Kaggle, 2023. [Online]. Available: • https://www.sciencedirect.com/science/article/pii/S1742287615000079?ref=pdf_download&fr=RR-2&rr=7b77774e197530dd#page=10&zoom=100,0,0.

[5] T. A. Almeida, "SMS Spam Collection Data Set," University California Irvine, 22 June 2012. [Online]. Available: • https://www.sciencedirect.com/science/article/pii/S1742287615000079?ref=pdf_download&fr=RR-2&rr=7b77774e197530dd#page=10&zoom=100,0,0.

[6] B. Lantz, Machine Learning with R, Packt, 2019.

[7] D. G. Sohom Ghosh, Natual Language Processing Fundamentals, Packt, 2019.

[8] H. Saleh, Machine Learning Fundamentals, Packt, 2018.

[9] Z. Nagy, Artificial Intelligence and Machine Learning Fundamentals, Packt, 2018.

[10] L. Owen, Hyperparameter Tuning with Python, Packt, 2022.

[11] G. Bonaccoroso, Machine Learning Algorithms - Second Edition, Packt, 2018.

[12] C. Sinclair, "An Application of Machine Learning to Network Intrusion Detection," [Online]. Available: https://www.cs.unc.edu/~jeffay/courses/nidsS05/ai/00816048.pdf.

[13] K. Feasel, Finding Ghosts in Your Data: Anomaly Detection Techniques with Examples in Python, Apress, 2022.

[14] Y. Liu, Python Machine Learning By Example - Third Edition, Packt, 2020