A Comparative Study of Classification Algorithms for SMS Spam Detection in the Telecommunication Sector

Jomaina Hafiz Ahmed
School of Computer Science and Engineering
Lovely Professional University, Phagwara
Punjab, India
jomaina.ahmed@gmail.com

***Abstract*—In the modern era, the use of smartphones and mobile devices has led to an exponential increase in Short Message Service (SMS) traffic. Among these messages, a significant portion is categorized as spam, often from malicious units attempting to extract confidential information from individuals. These spam messages disrupt the normal functioning of smartphones and pose a serious threat to personal security and privacy. To combat this issue, various machine-learning techniques have been employed. In my study, I have utilized a comprehensive dataset from the UCI Machine Learning repository and applied several machine learning algorithms including K-Nearest Neighbours (KNN), Naive Bayes, Support Vector Machines (SVM), and Logistic Regression. The performance of these algorithms was evaluated based on their accuracy, precision, and recall. As the number of mobile users continues to grow, so does the volume of SMS traffic and spam messages. These spam messages are often sent for financial gain or business promotion and require effective spam classification techniques. In this paper, I have applied a variety of Machine Learning Techniques for SMS spam detection using a dataset from UCI. The experimental results indicate that the Multinomial Naïve Bayes model outperforms previous models in spam detection, achieving an accuracy of 97.09% and a precision of 100%. All implementations were carried out using Python. This study underscores the potential of machine learning in enhancing spam detection in the telecommunications sector.**

Keywords*—*Machine Learning; SMS Spam Detection; Comparative Study; K-Nearest Neighbor; Naïve Bayes Classifier; Support Vector Machine; Logistic Regression; Accuracy; Precision; Recall;

I. INTRODUCTION

As of 2024, the number of smartphone users worldwide is estimated to be around 4.88 billion with China, India, and the US leading in usage. One of the key features of these devices is the Short Message Service (SMS), a text messaging service that doesn't require internet access. This makes SMS available on both smartphones and basic mobile phones, leading to a steady increase in SMS traffic.

Spammers, individuals, or companies that send unsolicited messages, often exploit this service for their gain or to promote their businesses. These spam messages, which can be sent via SMS or email can cause annoyance and financial loss for users. Despite the existence of various

SMS spam filtering techniques, there is still a need for more advanced solutions to this growing problem. Machine Learning provides solutions for such kinds of problems. Machine Learning models provide advanced methods to

classify messages as either spam or ham based on patterns learned from data. This leads to higher precision compared to traditional rule based-based methods. These models are adaptable, efficient and scalable, providing an improved user experience. Effective and customizable spam detection reduces the number of spam messages that the user receives, leading to an improved user experience and higher satisfaction. ML-based spam detection can reduce the need for manual review, saving time, energy, and resources.

II. PROPOSED METHODOLOGY

The SMS Spam Dataset was collected from the UCI Machine Learning Repository, and different data cleaning, exploratory data analysis (EDA), and text preprocessing techniques were applied to it before applying the Machine Learning models. Various machine-learning algorithms were then implemented and evaluated. Based on the evaluation, hyperparameter tuning was applied to the different machine learning models and a predictive system was built. The proposed methodology is shown in Figure 1.

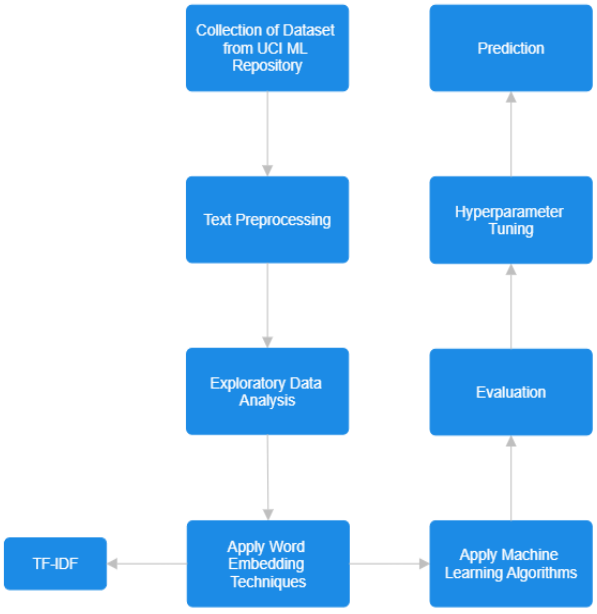

Fig 1. The Proposed Methodology for the SMS Spam Detection

A. *The Dataset and its Characteristics*

The Dataset is downloaded from the UCI repository. The dataset contains 5572 rows and 5 columns. The first column specifies whether the message is "spam" or "ham". Here, "spam" means unsolicited message, and "ham" means normal message. The second column contains the actual textual message. The last three columns contain the NAN values. The attributes are named – [v1, v2, Unnamed: 2, Unnamed: 3, Unnamed: 4]. Columns "v1" and "v2" consists of the data, whereas, the rest of the columns contain the NAN values.

Number of Rows	Number of Columns	Columns
5572	5	v1
		v2
		Unnamed: 2
		Unnamed: 3
		Unnamed: 4

Fig 2. Table showing the characteristics of the dataset.

B. *Exploratory Data Analysis (EDA)*

Exploratory Data Analysis (EDA) involves examining and analyzing datasets to understand their key characteristics, discover trends, identify outliers, remove duplicate values, and handle missing (NaN) values to clean and prepare the data for further analysis and establish relationships between variables. EDA is typically conducted as an initial phase before moving on to more formal statistical analyses or modeling.

1) Data Cleaning: To improve the dataset, data cleaning techniques are essential due to the high proportion of missing values in the last three columns of the dataset. These columns are removed, entirely. The first two columns are renamed to "Target" and "Text" from "v1" and "v2", respectively, for better comprehension of the data.

Label Encoder is used to transform the "Target" column, which consists of categorical data ("ham" and "spam"), into numeric data (since there were just two categories, "ham" is assigned "0" and "spam" as "1"). Additionally, 403 duplicate values were identified and removed from the dataset. The number of rows came down to 5169 from 5572 and columns came down to 2 from 5.

Label encoding is useful for converting non-numeric data into a format that can be understood by machine learning algorithms, which typically require numerical inputs. In this process, each unique category is assigned a unique integer value.

2) Data Visualization: This process uses graphical visualizations to represent data and provide insights into its patterns. Visualizations such as histograms, box plots, scatter plots, line plots, heatmaps, and bar charts enable the identification of trends and relationships within the data. These visual techniques help reveal the distribution of data, compare variables, and highlight key features of the dataset.

The total number of rows and columns after Data Pre-preprocessing is (5169, 2). The number of 'ham' and 'spam' in the Target column results in 4516 and 653, respectively. The pie chart shown below, in Fig-2, shows the percentage of "ham" and "spam" values in the "Target" column of the data set. The figure below shows that the data is Imbalanced, which essentially means one or more classes have significantly more data points than the other classes. In an imbalanced dataset, standard performance metrics such as accuracy can be misleading. For example, if the majority class represents 90% of the data, a model that always predicts the majority class will have high accuracy but will fail to classify the minority class correctly. In such cases, precision can be an important metric to assess the performance of a machine learning model. Using precision, along with recall together provides a more complete picture of a model's performance on imbalanced data.

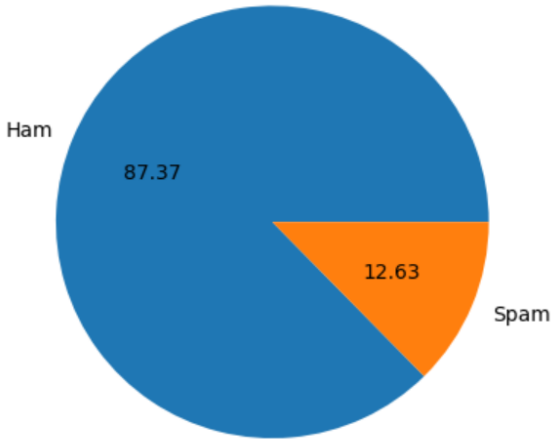

Fig 3. This is a pie chart showing the distribution of ham and spam messages after Data Preprocessing.

a) Natural Language Toolkit: The Natural Language Toolkit (nltk) is a popular Python library for natural language processing (NLP). It provides a wide range of tools and resources for working with human language data, making it easier for developers and researchers to analyze and manipulate text. Here are some key features and functionalities of the nltk library:

- Tokenization: Splitting text into smaller units such as words, sentences, or phrases.
- Stemming and Lemmatization: Reducing words to their base or root form to improve text processing.
- Stop Words: Removing common words (e.g., "the", "and", "is") that may not add much meaning to the analysis and are only important in sentence formation.

Using the Natural Language Toolkit (nltk), three new columns have been created in the dataset: `num_characters`, `num_words`, and `num_sentences`. Each of these columns represents a different aspect of the messages in the dataset. The `num_characters` column contains the total number of

characters in each message, including spaces and punctuation. The `num_words` column contains the number of words in each message by using the "tokenize" function. The "word_tokenize" function breaks down a message into a list of words, while the "sent_tokenize" function breaks down a message into a list of sentences. The `num_sentences` column contains the number of sentences in each message using sentence tokenization provided by nltk. These new columns provide additional insights into the text data and can be useful for further analysis and modeling.

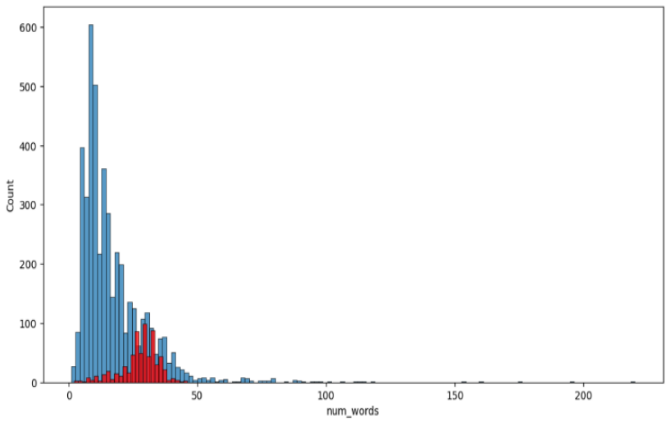

Fig 4. This is a histogram displaying the number of words in "ham" or legitimate messages and "spam", unsolicited messages.

b) Histogram: A general observation is made that spam messages tend to be larger and have more characters, words, and sentences, which can be seen in the previous figures as well. The histogram you're referring to displays the word count distribution in 'ham' (non-spam) and 'spam' text messages. It indicates that spam messages typically contain a greater number of words than ham messages. This trend might be due to the nature of spam messages, which often include more elaborate language to convey deceptive offers or instructions, as opposed to the more concise language used in regular, legitimate messages. A histogram is also plotted using the Seaborn library to better understand the number of characters, words, and sentences used in each message. [Fig 4-5]

The histogram shows that "spam" messages use much more characters and words than "ham" messages, which are usually shorter and to the point. The red bars depict the number of characters in "spam" messages and the blue shows the number of characters in the "ham" messages. [Fig 4] The red bars depict the number of words in "spam" messages and the blue shows the number of words in the "ham" messages. [Fig 5]

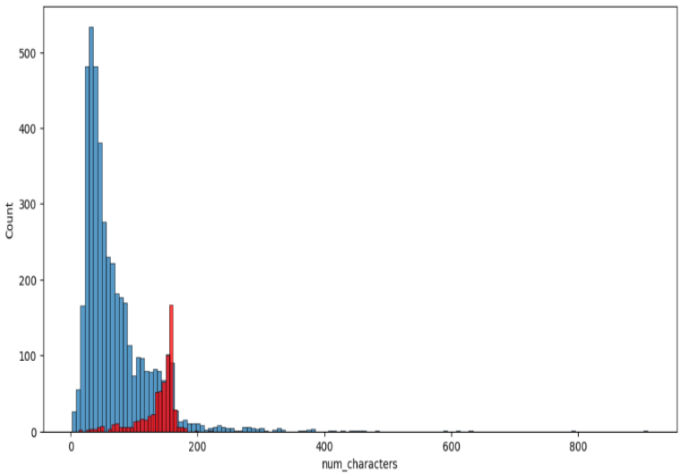

Fig 5. This is a histogram displaying the number of characters in "ham" or legitimate messages and "spam", also called unsolicited messages.

c) Correlation Analysis: The relationship between variables in a dataset can be explained by looking at the correlation between them. Correlation measures how the change in one variable is linked to the change in another variable, ranging from 1 to -1. A correlation of 1 means there is a strong positive relationship, where the two variables increase or decrease together. A correlation of -1 means there is a strong negative relationship, where one variable increases as the other decreases. A correlation of 0 means there is no clear relationship between the variables.

Correlation analysis aims to describe the relationships between different attributes in the data and identify which attributes are strongly related. This information can be used to select a set of features for further analysis. One common method for calculating correlation is the Pearson Product-Moment Correlation coefficient, which measures the linear relationship between two variables. Other notable methods and references include Bravais's work on correlation analysis and Rousseau, Egg he, and Guns' research on correlation in data.

$$\frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Equation 1: Karl Pearson's Coefficient

i. x_i: x-variable values in the sample.
ii. $\bar{x}$: mean of values of the x-variable.
iii. y_i: y-variable values in the sample.
iv. $\bar{y}$: mean of values of the x-variable.

Utilizing the Pearson Product-Moment Correlation, a heatmap was generated to visualize the correlation between the variables and to choose viable features for the feature set as shown below. [Fig 6]

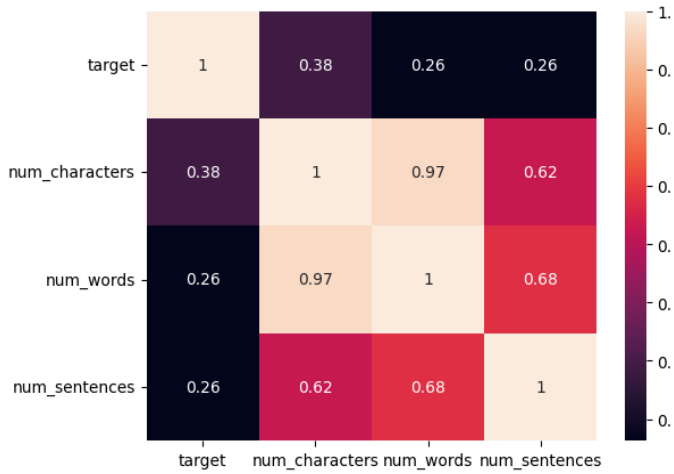

Fig 6. This figure shows the multicollinearity among the different attributes of the dataset.

Upon investigation of the heatmap, the following attributes were used to curate the feature set – "num_characters", followed by the target variable, which would be whether the message received by a user is spam or ham.

C. *Text Preprocessing*

Data preprocessing is a crucial step in natural language processing (NLP) tasks, as it helps clean and prepare text data for analysis and modeling. Proper preprocessing can lead to better model performance and more accurate analysis results.

a) Lower Case Conversion: Converting all text to lowercase ensures consistency in the data by treating words uniformly regardless of their case. This step is crucial because it prevents words with different cases (e.g., "Hello", "hello", and "HELLO") from being considered distinct tokens. By transforming the text to lowercase, you can standardize the data and avoid redundancy in the tokenization process.

b) Tokenization: Tokenization involves splitting the text into individual tokens, which can be words or sentences. Functions like "word_tokenize" and "sent_tokenize" from the "nltk" library can be used for this purpose. Tokenization is the foundation of text analysis because it allows the transformation of raw text into a structured format that can be analyzed and processed. For example, the sentence "Hello world!" would be tokenized into the list ["Hello", "world", "!"], separating words and punctuation for further processing. If we apply Lower case conversion also, we would get ["hello", "world", "!"].

c) Removing Special Characters: Special characters such as symbols, can clutter the text and may not contribute meaningful information for analysis. Removing these characters helps clean the data and focus on the relevant textual content. For instance, a sentence like "Hello, world$" can be cleaned to "Hello world," eliminating the extraneous dollar symbol and making the text more straightforward for further processing.

d) Removing Stop Words and Punctuation: Stop words are common words such as "and," "the" and "is" which often appear frequently in text, because it helps in the sentence formation, but do not carry significant meaning for analysis. By removing these words using the `nltk` stopwords corpus, removing punctuation marks using regular expressions helps further clean the data and prepare it for modeling. For example, the sentence "Hello, world! This is an SMS" can be transformed into a list of words such as ["hello," "world," "sms"], if all of the above steps are applied together, by defining a function. You can also download the stop words by using the "nltk.corpus.stopwords.words("english")". It would print all the stop words present in the English language, like, ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your' etc]. String module from python consists of all the punctuation marks that are used in the English language. By removing them, the model can focus more on the words and patterns in the sms rather than being distracted by symbols. Some of them are !"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~.

d) Stemming: Stemming is the process of reducing words to their base form, also known as the root form or lemma. This is achieved by removing affixes such as prefixes and suffixes from words, resulting in a smaller vocabulary size and improved generalization. For instance, the words "running" and "jumps" would stem to their base forms, "run" and "jump," respectively. Stemming helps in standardizing words and allows the model to focus on the core meaning of the text, rather than variations of the same word. Tools such as the "PorterStemmer" class from the "nltk.stem.porter" module are used for this purpose.

After performing all the above steps, a new column "transformed_text" is added to the data frame, which would be used instead of the actual text as it has gone through various steps of text preprocessing.

2) Word Cloud: A word cloud is a visual representation of text data that shows the frequency or importance of words within a dataset, with larger words indicating higher frequency. These visualizations offer a quick summary of the main topics and themes in a text, making them useful for text analysis, keyword identification, and comparative studies. It is applied to the "transformed_text" column in the data set to understand the pattern and trends in the SMS. Word Clouds are a versatile tool across various fields for analyzing and visualizing text data.

In this context, terms like "text," "stop," "claim," "call," "free," "offer," "mobile," and "prize" suggest themes of communication, offers, or instructions, which are common in spam messages, as shown below in Figure 7. This can help in identifying the most frequent words in spam messages, which can be used as features in a machine learning model to classify new messages as spam or ham.

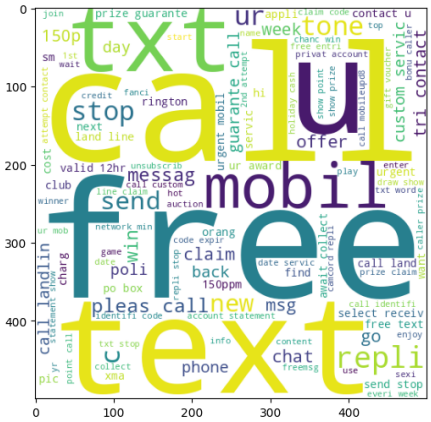

Fig 7. A Word Cloud showing the frequency of words being used in spam messages. The larger the word, the more its importance and frequency in the dataset.

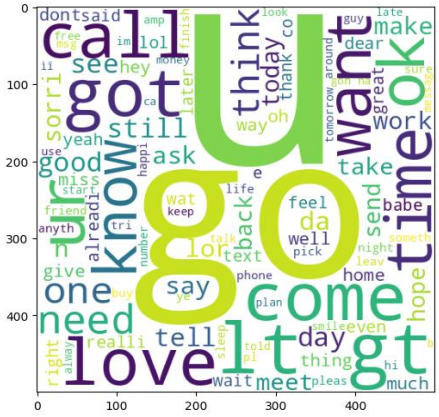

Fig 8. A Word Cloud showing the frequency and importance of words being used in ham messages.

The most prominent word in Figure 8 is “love,” which suggests that it is a key theme or frequently used word within the “ham” messages. Other large words include “know,” “think,” “want,” “say,” “go,” “u,” “come,” “ok,” and “time,” which are common words used in ham or normal messages by the users.

III. MODEL BUILDING

A. *TF-IDF Vectorizer*

Term frequency Inverse document frequency (TFIDF) is a statistical formula to convert text documents into vectors based on the relevancy of the word. It is based on the bag of the words model to create a matrix containing the information about less relevant and most relevant words in the document.

TF-IDF is beneficial in NLP tasks, topic modeling, and machine learning tasks. It helps algorithms to use the importance of the words to predict outcomes.

a) Term Frequency: It is the ratio of the occurrence of the word (w) in document (d) per the total number of words present in the document (d). With this simple formulation, we are measuring the frequency of a word in the document.

For example, if the sentence has 6 words and contains two “the”, the TF ratio of “the” word would be (2/6). It can also be said as the probability of a word occurring in the document as it is always between 0 and 1. It gives more weight to the terms that are more unique to a document in the collection.

$$tf(w,d) = \frac{occurence\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

Equation 2: Term Frequency

b) Inverse Document Frequency (IDF): IDF calculates the importance of a word in a corpus D. The most frequently used words like “of, we, are” have little to no significance. It is calculated by dividing the total number of documents in the corpus by the number of documents containing the word.

$$idf(w,D) = ln(\frac{total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w})$$

$$\mathrm{idf}(\mathrm{w},\mathrm{D}) = \log(\frac{\mathrm{N}}{\mathrm{f}(\mathrm{w},\mathrm{D})})$$

Equation 3: Inverse Document Frequency

c) Term Frequency Inverse Document Frequency (TFIDF): TF-IDF is the product of term frequency and inverse document frequency. It gives more importance to the word that is rare in the corpus and common in a document.

$$\mathrm{tfidf}(\mathrm{w},\mathrm{d},\mathrm{D}) = \mathrm{tf}(\mathrm{w},\mathrm{d}) \times \mathrm{idf}(\mathrm{w},\mathrm{D})$$

Equation 4: Term Frequency Inverse Document Frequency

B. *Train Test Split:*

The “train_test_split” function is used to split the features (X) and labels (y) into training and testing sets. 20% of the data will be used for the test set and the rest 80% is used for the training set. The “random_state=2” parameter ensures that the splits generate the same data set every time the code is run. As the Naïve Bayes Classifier gives the best output in terms of textual data, Three different types of Naïve Bayes Classifier are imported – Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes Classifier, along with other classification algorithms like, Support Vector Machine, Logistic Regression and K-Nearest Neighbors.

a) Gaussian Naïve Bayes: Gaussian Naive Bayes (GNB): GNB is used when the input data is continuous and follows a Gaussian distribution. It calculates the mean and standard deviation of each feature for each class and uses these values to calculate the probability of a new instance belonging to a class. This approach is particularly useful when the data is normally distributed and there are multiple features involved in the classification process.

b) Bernoulli Naive Bayes (BNN): BNN is used when the input data is binary (0/1 or yes/no) and follows a Bernoulli distribution. It models the presence or absence of a feature and calculates the probability of a new instance belonging to a class based on the presence or absence of each feature. This approach is particularly useful when the data is binary and there are multiple features involved in the classification process.

c) Multinomial Naïve Bayes (MNB): Multinomial Naive Bayes (MNB) is a variant of the Naive Bayes classifier tailored for text classification tasks, assuming features follow a multinomial distribution. Specifically designed for discrete data like word counts in documents, MNB's strength lies in

its ability to efficiently handle large datasets with many features, making it popular for spam detection and sentiment analysis applications. By assuming feature independence given the class label, MNB can effectively filter out irrelevant features, providing interpretable probabilities for document classification. This contrasts with Gaussian Naive Bayes, better suited for continuous data.

d) K-Nearest Neighbour (KNN): The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method that classifies or predicts based on the similarity of data points. It works by finding the K nearest neighbors to a given data point using a distance metric like Euclidean distance, and then determines the class or value of the data point based on the majority vote or average of the K neighbors. KNN is versatile, adapting to different patterns and local structures in the data, making it effective for classification and regression tasks.

e) Logistic Regression: Logistic regression is a supervised machine learning algorithm commonly used for binary classification tasks, predicting the probability of an instance belonging to a specific class. It employs the logistic or sigmoid function to map predictions to probabilities between 0 and 1, making it ideal for binary outcomes like spam detection or disease diagnosis. Logistic regression assumes linearity between independent variables and the log odds ratio, with considerations for multicollinearity and error independence. It offers three main types: Binomial, Multinomial, and Ordinal, catering to different classification scenarios. Model evaluation involves metrics like accuracy, precision, recall, and AUC-ROC.

f) Support Vector Machine: Support Vector Machines (SVM) are a powerful supervised machine learning algorithm used for both classification and regression tasks. It aims to find the optimal hyperplane that maximizes the margin between the closest data points of different classes, known as the maximum-margin hyperplane. The data points closest to this hyperplane are called support vectors, and they define the decision boundary. SVM can handle both linear and non-linear classification problems by using kernel functions, such as linear, polynomial, radial basis function (RBF), and sigmoid, to map the data into a higher-dimensional feature space where it can be linearly separated. SVM is effective in high-dimensional spaces, can handle complex, non-linear data, and is memory-efficient as it uses a subset of training points (support vectors) to define the decision boundary. However, SVM can be sensitive to the choice of kernel function and its hyperparameters, and it may not perform well with large datasets or noisy data.

IV. MODEL EVALUATION

After evaluation, Multinomial Naïve Bayes, with an accuracy of 97.10% and Precision of 100% is the best model among them. Since the dataset is imbalanced, in such cases, Precision would be a better evaluation metric than accuracy. Therefore, it is advisable to try to improve the value of the Precision of the model.

Classifier	Accuracy	Precision	Recall	F1-Score
Gaussian	0.87	0.51	0.80	0.62
Multinomial	0.97	1.00	0.78	0.88
Bernoulli	0.98	0.99	0.88	0.93

Fig 9. Table of comparison displaying the evaluation metrics of different types of Naïve Bayes Classifiers

In addition to Naïve Bayes Classifiers, other models are also imported and evaluated for spam detection, like Support Vector Machine (SVM), Linear Regression, and K-Nearest Neighbours (KNN) classifiers.

Classifier	Accuracy	Precision	Recall	F1-Score
SVC	0.9758	0.9748	0.8406	0.9027
KNN	0.9052	1.0000	0.2899	0.4494
MNB	0.9710	1.0000	0.7826	0.8780
LR	0.9584	0.9703	0.7101	0.8201

Fig 10. Table of comparison displaying the evaluation metrics of different types of classifiers

A. *Hyperparameter Tuning:*

Hyperparameter tuning is a critical aspect of building efficient and high-performing machine learning models.

1) Model Parameters:

Model parameters are values that are learned from the training data during the training process. Examples include weights in a neural network or coefficients in a linear regression model.

The model's objective is to find the optimal values for these parameters that minimize the loss function and improve predictive performance.

2) Hyperparameters

Hyperparameters are values that control the behaviour of the learning process. They are not learned from the data; instead, they are set before training and can significantly impact the model's performance.

Examples include the learning rate in gradient descent, the regularization strength in linear models, the number of neighbours in K-Nearest Neighbours, and the depth of a decision tree.

B. *Grid Search*

Grid search is a method of hyperparameter tuning that involves dividing the domain of the hyperparameters into a discrete grid and trying every combination of values. This method is exhaustive, meaning it will find the best point in the domain, but it can be very slow and computationally expensive . For each point in the grid, k-fold cross-validation is performed, which requires k training steps. This makes grid search a good option when the number of hyperparameters is small and the model is simple, but it can be impractical for complex models with many hyperparameters

C. *Random Search*

Random search is similar to grid search, but instead of using all the points in the grid, it tests only a randomly selected subset of these points. The smaller this subset, the faster but less accurate the optimization. The larger this subset, the more accurate the optimization but the closer to a grid search1. Random search is a useful option when you have several hyperparameters with a fine-grained grid of values. Using a subset made by 5-100 randomly selected points, you can get a reasonably good set of values for the hyperparameters.

After applying Random Search on the different models, four machine learning algorithms: Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Multinomial Naive Bayes (NB), and Logistic Regression (LR). The SVC, with parameters 'kernel': 'linear', 'gamma': 1, and 'C': 0.1, achieved an accuracy of 0.9748, precision of 0.9710, recall of 0.8333, and F1 score of 0.8984. The KNN algorithm, with parameters 'weights': 'uniform' and 'n_neighbors': 5, demonstrated an accuracy of 0.8994, precision of 1.0000, recall of 0.2463, and F1 score of 0.39. The NB algorithm was evaluated with 'alpha': 1 as the best parameter, achieved an accuracy of 0.9710, precision of 1.00, recall of 0.7826, and F1 score of 0.8788. The LR algorithm, with 'solver': 'libfgs', 'penalty': 'l2', and 'C':10, demonstrated an accuracy of 0.9739, precision of 0.9743, recall of 0.8268, and F1 score of 0.8941.

Classifier	Accuracy	Precision	Accuracy after tuning	Precision after tuning
SVC	0.9758	0.9748	0.9748	0.9710
Multinomial Naïve Bayes	0.9710	1.0000	0.9710	1.0000
Logistic Regression	0.9584	0.9703	0.9739	0.9743
KNN	0.9052	1.0000	0.8994	1.0000

Fig 11. Table of comparison displaying the accuracy and precision values of different types of classifiers before and after hyperparameter tuning

V. CONCLUSION

As can be seen from the previous tables Multinomial Naïve Bayes is the best model for SMS spam detection, with an accuracy of 97.10% and precision of 100%. In this case, precision would be a better evaluation metric compared to accuracy because the data is Imbalanced, which essentially means one or more classes have significantly more data points than the other classes. In an imbalanced dataset, standard performance metrics such as accuracy can be misleading. For example, if the majority class represents 90% of the data, a model that always predicts the majority class will have high accuracy but will fail to classify the minority class correctly. In such cases, precision can be an important metric to assess the performance of a machine learning model. Using precision which provides a more complete picture of a model's performance on imbalanced data. Therefore, it is the best model. It means that 97% of the model's total predictions are correct, and all of the model's positive predictions are correct (there are no false positives).

These findings offer valuable insights into how different classification algorithms perform and can help in choosing the most suitable one for a particular application. In simpler terms, it's concluded that Multinomial Naïve Bayes is better than all the other models used in predicting whether an SMS is spam or not, with "1" meaning spam results and "0" meaning not spam results.

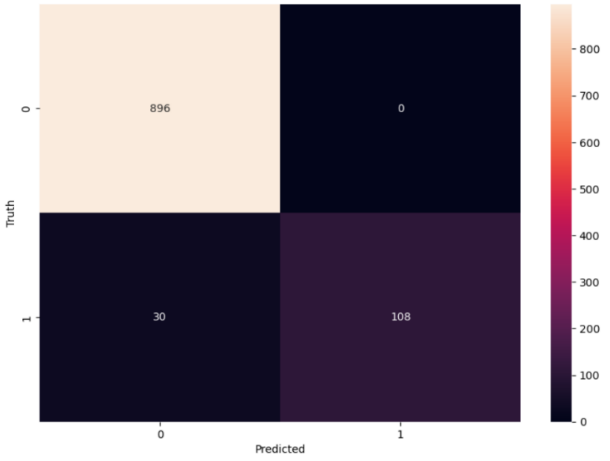

Fig 12. Confusion matrix of Multinomial Naïve Bayes Classifier

The top left quadrant (True 0, Predicted 0) shows the number 896. This represents the number of true negatives, i.e., the number of negative instances (0s) correctly predicted as negative. The top right quadrant (True 0, Predicted 1) shows the number 0. This represents the number of false positives, i.e., the number of negative instances (0s) incorrectly predicted as positive. The bottom left quadrant (True 1, Predicted 0) shows the number 30. This represents the number of false negatives, i.e., the number of positive instances (1s) incorrectly predicted as negative. The bottom right quadrant (True 1, Predicted 1) shows the number 108. This represents the number of true positives, i.e., the number of positive instances (1s) correctly predicted as positive.

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of this research paper on SMS Spam Detection using Machine Learning. This work would not have been possible without the invaluable contributions and support of the authors of the various research papers, textbooks, and resources that have been instrumental in shaping our understanding of the different models and techniques for spam detection.

REFERENCES

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] S. Gadde, A. Lakshmanarao, and S. Satyanarayana, "2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)," Coimbatore, India, 19-20 March 2021. [Online]. Available: https://doi.org/10.1109/ICACCS51430.2021.9441783. [Accessed: 03-June-2021].

[3] [online] https://www.kdnuggets.com/2022/09/convert-text-documents-tfidf-matrix-tfidfvectorizer./