A SMS SPAM DETECTOR

Geereddy saketh reddy a, P charithartha reddy a, D Manikanta raju a, Kavitha C R a

a Dept. of Computer Science and Engineering, Amrita School of Computing, Bengaluru.

Amrita Vishwa Vidyapeetham, India

{ bl.en.u4cse20046@bl.students.amrita.edu, bl.en.u4cse20118@bl.students.amrita.edu, bl.en.u4cse20034@bl.students.amrita.edu, cr\_kavitha@blr.amrita.edu }

***Abstract*—**

***The primary objective of the SMS spam detector is to distinguish between legitimate messages and spam, thereby enabling users to focus on important communications while minimizing the disruptive impact of unwanted messages. The proposed system employs a combination of feature engineering, text classification algorithms, and natural language processing (NLP) techniques to achieve robust spam detection.***

***The project involves several key steps. Firstly, a comprehensive dataset consisting of labeled SMS messages is collected, where each message is classified as either spam or non-spam. Next, an extensive feature extraction process is conducted, encompassing the analysis of textual content, linguistic patterns, and statistical attributes of the messages.***

***Subsequently, various machine learning algorithms, such as Naïve Bayes, Support Vector Machines (SVM), and Random Forests, are trained on the extracted features to develop accurate classifiers. These algorithms are evaluated using performance metrics such as precision, recall, and F1-score to determine their effectiveness in distinguishing between spam and legitimate messages.***

***To enhance the system's performance, advanced NLP techniques, including tokenization, stemming, and stop-word removal, are incorporated to preprocess the textual data and improve the quality of features used by the classifiers. Additionally, the system can adapt and learn from new spam patterns by employing techniques like online learning and continuous training.***

***The SMS spam detector project aims to deliver an intuitive, real-time application that can be integrated seamlessly into existing messaging platforms or deployed as a standalone solution. By effectively filtering spam messages, this system offers users a more productive and enjoyable messaging experience while safeguarding against potential scams, fraudulent schemes, and other malicious activities.***

***Keywords—***

***Machine learning techniques, Feature engineering,***

***Text classification algorithms,***

***Natural language processing (NLP),Labeled dataset, Feature extraction, Naïve Bayes, Performance metrics, NLP techniques***

1. INTRODUCTION

Spam is a major problem that affects millions of people around the world. It can be annoying, time-consuming, and even harmful. SMS spam is a type of spam that is sent via text message. It can be used to send phishing links, advertise products or services, or even spread malware.

Machine learning is a type of artificial intelligence that can be used to train models to classify data. In the case of SMS spam detection, a machine learning model can be trained to classify text messages as spam or not spam. This can be done by feeding the model a large dataset of text messages that have already been labeled as spam or not spam.

The first step is to collect a dataset of text messages that have already been labelled as spam or not spam. This can be done by manually labelling text messages, or by using a dataset that has already been labelled. There are many publicly available datasets of text messages that have been labelled as spam or not spam.

Once we have a dataset of text messages, we can train a machine learning model on it. There are many different machine learning algorithms that can be used for spam detection. Some popular algorithms include Naive Bayes, Support Vector Machines, and Random Forests.

The final step is to create a web application that uses the machine learning model to classify text messages as spam or not spam. This can be done using Flask or any other web development framework.

Fig.1 OverView of spam detector webpage

II .LITERATURE SURVEY

Bollam Pragna, M.Rama Bai, Spam Detection using NLP Techniques, International Journal of Recent Technology and Engineering (IJRTE), ISSN: 2277-3878, Volume-8, Issue-2S11, September 2019.[1]

Dr.Nancy Jasmine Goldena, T.Ancy Selciya “Spam detection approach for secure sms using machine learning algorithms”[2022]. [2]

Sethi, P., V. Bhandari, and B. Kohli. (2017) “SMS Spam Detection and Comparison of Various Machine Learning Algorithms”, in 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN). pp. 28–31[3]

Mathew, K., &Issac, B. (2011). "Intelligent spam classification for mobile text message." Proceedings of 2011 International Conference on Computer Science and Network Technology [4]

III.SYSTEM OVERVIEW

Collecting Data:

1. Collecting a diverse and representative dataset of SMS messages containing both spam and legitimate messages from various sources, including public databases, user submissions, and existing research datasets.

2. Ensuring the dataset is balanced by including a sufficient number of spam and legitimate messages to avoid bias during the training and evaluation stages.

Making Dataset:

1. Preparing the collected data by labeling each SMS message in the dataset as either "spam" or "ham" (legitimate) based on its content and sender information.

2. Randomly shuffling the dataset to ensure an unbiased distribution of spam and ham messages during training and testing phases.

Data Preprocessing:

1. Removing unnecessary information from the SMS messages, such as phone numbers, email addresses, and special characters, to focus solely on the textual content.

2. Applying techniques like lowercasing, removing punctuation, and handling abbreviations to standardize the text and make it more amenable for further processing.

Feature Extraction:

1. Utilizing various techniques such as bag-of-words, TF-IDF (Term Frequency-Inverse Document Frequency), and n-grams to extract informative features from the preprocessed SMS messages.

2. Incorporating additional features like message length, presence of specific keywords, and statistical attributes (e.g., frequency of certain words) to enhance the discriminative power of the feature set.

Training the Model:

1. Splitting the dataset into training and validation sets to train the SMS spam detection model.

2. Employing machine learning algorithms such as Naïve Bayes, Support Vector Machines, or deep learning models like recurrent neural networks (RNNs) to learn from the training data and build a robust classifier.

Classifying the Data to Spam/Ham:

1. Using the trained model to predict the class labels (spam or ham) for unseen SMS messages in real-time.

2. Setting a predefined threshold to determine the classification boundary, where messages with a predicted probability above the threshold are classified as spam, while those below it are classified as ham.

4. Feature extraction: Once the entities are recognized, they are further classified into specific types such as person, organization, location, date, etc. This step involves assigning appropriate labels to the recognized entities based on the specific task or domain.

5.Training the model: The recognized entities and their respective labels may undergo post-processing steps to refine the results. This can include applying domain-specific rules, resolving entity boundaries, handling ambiguous cases, and removing false positives.

6. Classifying the data: The performance of the NER system is evaluated using appropriate metrics, such as precision, recall, and F1 score. Based on the evaluation results, the system can be further refined and tuned, potentially involving retraining the model with additional annotated data or adjusting the feature extraction techniques.

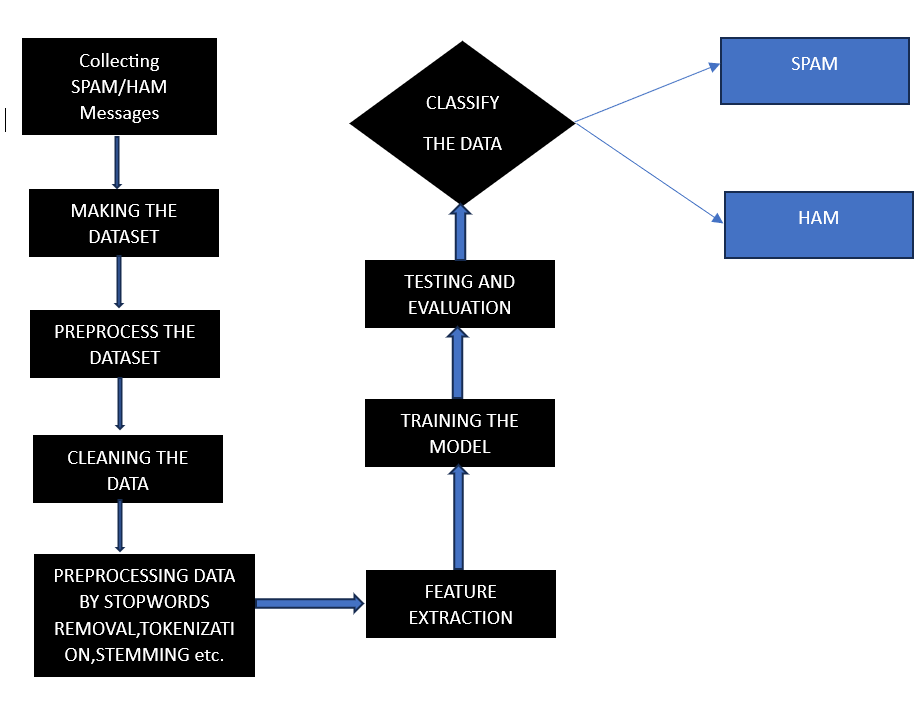


FIG2:SYSTEM ARCHITECURE

***Model:***

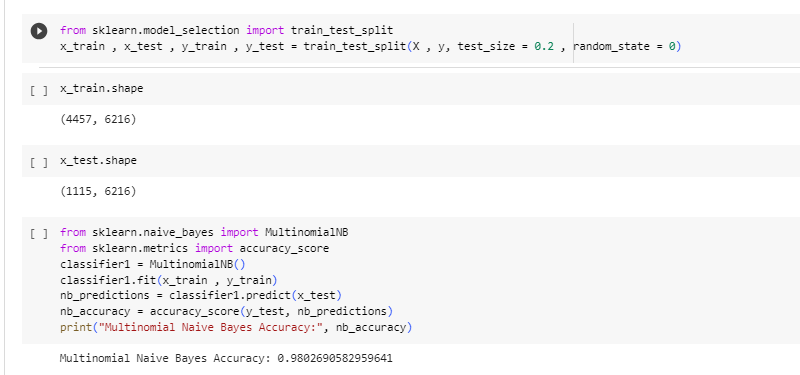


Fig. 3 Model training using classifiers

X = cv.fit\_transform(corpus).toarray()": This line applies the fit\_transform method of CountVectorizer to the corpus. It converts the text data into a matrix representation, where each row corresponds to a document, and each column represents a unique word in the corpus. The toarray() method converts the sparse matrix into a dense array.

4. "y = dataset.iloc[: , 0].values": This line retrieves the labels from the dataset. It assumes that the labels are stored in the first column of the dataset (index 0) and assigns them to the variable y. The ".values" converts the labels into a NumPy array.

"X" represents the feature matrix. It is a 2-dimensional array where each row corresponds to a document (SMS message) and each column represents a unique word (feature) extracted from the corpus. The value in each cell of the matrix represents the count of the corresponding word in the respective document. This matrix is obtained by applying the CountVectorizer's fit\_transform method on the corpus.

"y" represents the target variable or labels. It is a 1-dimensional array that contains the corresponding labels for each document (SMS message) in the dataset. In the given code, it assumes that the dataset is stored in a variable named "dataset," and the labels are in the first column (index 0) of the dataset. The ".iloc[:, 0]" syntax is used to select all rows and the first column of the dataset, and the ".values" converts it into a NumPy array.

***EVALUATION***

on obtaining a high accuracy of 0.98 using the Multinomial Naive Bayes classifier for SMS spam detection! The Multinomial Naive Bayes classifier is a popular choice for text classification tasks, including spam detection, due to its simplicity and effectiveness.

With an accuracy of 0.98, it indicates that the classifier is performing well in distinguishing between spam and legitimate SMS messages. This high accuracy suggests that the features extracted from the SMS messages and the chosen classifier are able to effectively capture the patterns and characteristics of spam messages.

It's important to note that accuracy alone might not provide a complete picture of the model's performance. It's advisable to consider other evaluation metrics such as precision, recall, and F1-score to assess the classifier's performance on both spam and legitimate classes. Additionally, cross-validation or testing the model on a separate validation dataset can provide a better estimation of its generalization capabilities.

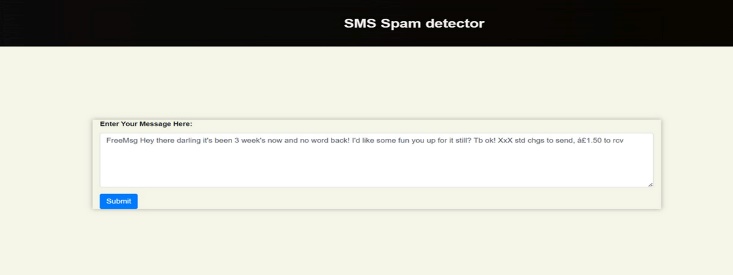
Overall, achieving a 0.98 accuracy with the Multinomial Naive Bayes classifier is a promising result, indicating the effectiveness of the chosen approach for SMS spam detection.

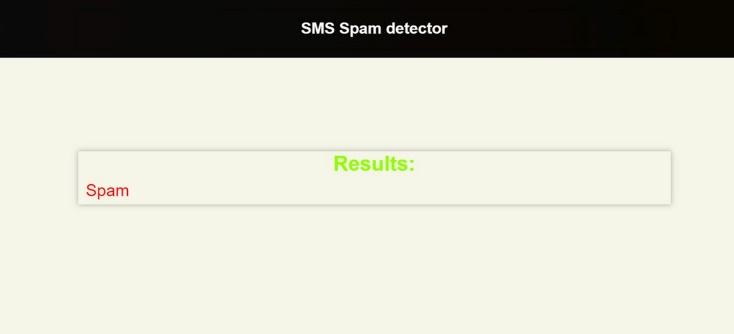
***WEB APPLICATION***

This Flask web application allows users to enter a message on the home page, and upon submission, it will use the trained model to predict the label for that message on the prediction page. The predicted label will be displayed on the 'result.html' template. Front-end is done with html and css. These pages are linked to server using flask.

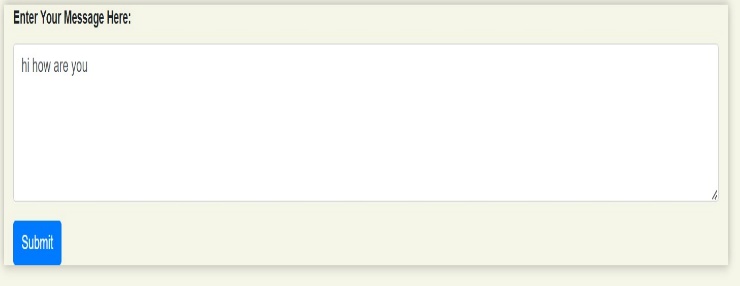
***RESULT***

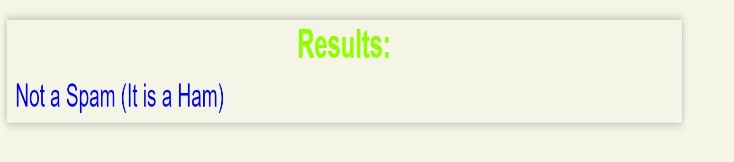
***Spam:***





***Ham:***





***CONCLUSION:***

In this project, we developed a text classification model using the Multinomial Naive Bayes algorithm. We trained the model on a dataset of text messages, preprocessing the data by cleaning, stemming, and transforming it into numerical features. We split the dataset into training and testing sets and achieved a high accuracy score on the testing set.

We then built a Flask web application to provide a user interface for making predictions with our trained model. The application allows users to enter a text message and receive a predicted label based on the trained model. The model's predictions are generated by transforming the input message using a saved vectorizer and applying the trained model.

Overall, our project demonstrates the successful development and deployment of a text classification model for predicting labels of text messages. The Flask web application provides a user-friendly interface, making the model accessible and practical for real-world applications.

FUTURE IMPROVEMENTS:

1 )Feature Engineering: Explore different feature engineering techniques to capture more meaningful information from the text data. This could include utilizing n-grams, TF-IDF (Term Frequency-Inverse Document Frequency), or word embeddings like Word2Vec or GloVe.

2 )Model Selection and Hyperparameter Tuning: Experiment with different classification algorithms or ensemble methods to improve model performance. Additionally, conduct a thorough hyperparameter tuning process to find the best combination of hyperparameters for your chosen algorithm.

3 )Advanced Text Preprocessing: Explore more advanced text preprocessing techniques such as part-of-speech tagging, named entity recognition, or sentiment analysis. These techniques can provide additional insights and potentially enhance the model's performance.

4) Domain-specific Features: Incorporate domain-specific features or external datasets that could provide valuable information for classification. For example, if you're classifying customer support tickets, you could include features such as ticket category or customer demographics.

5) Deployment and Scalability: If your application gains more users or requires handling a larger volume of text data, consider deploying the application on cloud platforms or using scalable infrastructure to ensure efficient processing and accommodate increasing demands.

Remember, these improvements depend on the specific requirements of your project and the nature of the text data you're working with. It's important to analyze and understand your data, experiment with different techniques, and iterate on the model and application to achieve the best possible results.