मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

SMS Spam Detection using Machine Learning

Submitted in the partial fulfillment of the requirements of the degree of

B.Sc. Computer Science

By

Radhika Choudhary

Under the Guidance of

Ms. Kamlesh Pal

Royal College of Arts, Science and Commerce, Penkar pada Mira Road (E), Mumbai – 401107

Year: 2022-2023

ACKNOWLEDGEMENT

Achievement is finding out what you would be doing rather than what you have to do. It is not until you undertake such a project that you realize how much effort and hard work it really is, what are your capabilities and how well you can present yourself or other things. It tells us how much we rely on the efforts and goodwill of others. It gives me immense pleasure to present this report towards the fulfilment of my project.

It has been rightly said that we are built on the shoulder of others. For everything I have achieved, the credit goes to all those who had helped me to complete this project successfully.

I take this opportunity to express my profound gratitude to management of **University of Mumbai** for giving me this opportunity to accomplish this project work.

A special vote of thanks to **Prof. Kamlesh Pal** who is our professor & project guide for her most sincere, useful and encouraging contribution through out the project span.

Finally, I would like to thank entire Computer Science department who directly or indirectly helped me in completion of this project & to my family without whose support, motivation & encouragement this would not have been possible.

- RADHIKA CHOUDHARY

TABLE OF CONTENT

INTRODUCTION

SMS spam detection is a critical task in the field of natural language processing (NLP) and machine learning. With the widespread use of mobile devices and the increasing number of text messages being sent, the problem of spam messages has become a significant concern for both users and service providers. SMS spam refers to unsolicited or unwanted text messages that are sent in bulk to a large number of recipients.

Machine learning techniques provide effective solutions for identifying and filtering out SMS spam. By training a model on a labeled dataset containing examples of spam and non-spam messages, the model can learn to differentiate between the two and make predictions on new, unseen messages.

In this detail, we will explore the steps involved in SMS spam detection using machine learning, including data preprocessing, feature extraction, model training, and evaluation.

1. Data Preprocessing:

The first step in building an SMS spam detection system is to preprocess the data. This typically involves cleaning and transforming the raw SMS messages into a format suitable for machine learning algorithms. Common preprocessing steps include:

Removing any irrelevant information such as phone numbers or email addresses.

Tokenizing the messages into individual words or tokens.

Converting the text to lowercase to ensure consistency.

Removing punctuation marks and special characters.

Removing stopwords (common words like "the," "and," "is") that do not carry much meaning.

2. Feature Extraction:

After preprocessing, the next step is to extract meaningful features from the SMS messages. These features serve as inputs to the machine learning model. Some common feature extraction techniques for text data include:

Bag-of-Words (BoW): Representing each message as a vector of word frequencies. Each word in the vocabulary forms a feature, and the count of its occurrences in the message becomes the feature value.

TF-IDF (Term Frequency-Inverse Document Frequency): Similar to BoW, but it takes into account the importance of words in the entire corpus, giving higher weights to rare words that appear in specific messages.

Word embeddings: Representing words as dense vectors in a continuous vector space. Techniques like Word2Vec or GloVe can capture semantic relationships between words.

3. Model Training:

With the preprocessed data and extracted features, we can proceed to train a machine learning model for SMS spam detection. Several algorithms can be used for this task, including:

Naive Bayes: A probabilistic algorithm that assumes independence between features. It works well with text data and is computationally efficient.

Support Vector Machines (SVM): A powerful algorithm that finds an optimal hyperplane to separate spam and non-spam messages.

Random Forest: An ensemble algorithm that combines multiple decision trees to make predictions. It can handle high-dimensional feature spaces and is less prone to overfitting.

Neural Networks: Deep learning models like recurrent neural networks (RNNs) or convolutional neural networks (CNNs) can capture complex patterns in text data.

4. Model Evaluation:

After training the model, it is crucial to evaluate its performance to assess its effectiveness in detecting SMS spam. Common evaluation metrics include:

Accuracy: The proportion of correctly classified messages.

Precision: The proportion of predicted spam messages that are actually spam.

Recall: The proportion of actual spam messages that are correctly identified.

F1-score: The harmonic mean of precision and recall, providing a balanced measure of model performance.

It is also essential to split the dataset into training and testing sets, ensuring that the model's performance is evaluated on unseen data to avoid overfitting.

OBJECTIVES

The spam detection is a big issue in mobile message communication due to which mobile message communication is insecure. In order to tackle this problem, an accurate and precise method is needed to detectthe spam in mobile message communication. We proposed the applications of the machine learning-based spam detection method for accurate detection. In this technique, machine learning classifiers suchas Logistic regression (LR), K-nearest neighbor (K-NN), and decision tree (DT) are used for classification of ham and spam messages in mobile device communication. The SMS spam collection data set is used for testing the method. The dataset is split into two categories fortraining and testing the research. The results of the experiments demonstrated that the classification performance of LR is high as compared with K-NN and DT, and the LR achieved a high accuracy of99%. Additionally, the proposed method performance is good as compared with the existing state-of-the-art methods.

There are four objectives that need to be achieved in this project:

i. To study on how to use machine learning techniques for spam detection.

ii. To modify machine learning algorithm in computer system settings.

iii. To leverage modified machine learning algorithm in knowledge analysis software.

iv. To test the machine learning algorithm real data from machine learningdata repository.

The objectives of SMS spam detection are to develop effective algorithms and systems that can accurately identify and filter out unsolicited or unwanted text messages. Here are the objectives of SMS spam detection in detail:

A. Identify and Filter Spam Messages:

The primary objective is to accurately identify and filter out spam messages from legitimate messages. The system should be able to differentiate between spam and non-spam messages based on their content and characteristics. By doing so, users can have a cleaner and more relevant messaging experience.

B. Protect Users from Scams and Unwanted Content:

SMS spam often includes fraudulent schemes, phishing attempts, malicious links, and other unwanted content. The objective is to protect users from these scams and prevent them from falling victim to fraudulent activities. By detecting and blocking spam messages, the system can reduce the risks associated with such activities.

C. Improve User Experience:

Receiving a large number of unwanted messages can be annoying and disrupt the user experience. The objective is to enhance the overall user experience by reducing the amount of spam messages that users receive. This ensures that users can focus on important and relevant messages without being overwhelmed by unsolicited content.

D. Optimize Resource Utilization:

By accurately detecting and filtering spam messages, service providers can optimize the utilization of network resources. Filtering out spam at the network level reduces the bandwidth and storage required to process and store unwanted messages. This optimization allows for more efficient use of resources and improves the overall performance of the messaging infrastructure

E. Adaptability to Evolving Spam Techniques:

Spammers constantly evolve their techniques to bypass spam filters. The objective is to develop spam detection systems that can adapt and stay effective against emerging spamming techniques. By employing machine learning algorithms and continuously updating the detection models, the system can stay ahead of spammers and effectively detect new types of spam messages.

F. Ensure High Detection Accuracy:

Achieving high detection accuracy is a crucial objective in SMS spam detection. The system should minimize false positives (classifying legitimate messages as spam) and false negatives (failing to detect actual spam messages). High accuracy ensures that legitimate messages are not mistakenly filtered out and that spam messages are effectively identified and blocked.

G. Scalability and Efficiency:

SMS spam detection systems should be scalable and efficient, capable of handling a large volume of messages in real-time. The objective is to develop algorithms and architectures that can process incoming messages quickly and accurately, ensuring minimal latency and maximizing system performance.

In summary, the objectives of SMS spam detection are centered around accurately identifying and filtering out spam messages, protecting users from scams, improving user experience, optimizing resource utilization, adapting to evolving spam techniques, ensuring high detection accuracy, and maintaining scalability and efficiency.

LITERATURE REVIEW

Here's a brief literature review on SMS spam detection:

1. "An Analysis of Machine Learning Techniques for SMS Spam Detection" by Alsmadi et al. (2017): This study compares various machine learning techniques, including Naive Bayes, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and Decision Trees, for SMS spam detection. The authors evaluate the performance of these algorithms using different feature selection methods and provide insights into their effectiveness.

2. "SMS Spam Detection: A Comparative Study on Feature Selection and Classification Techniques" by Ahmed et al. (2018): This research paper presents a comprehensive comparison of feature selection techniques and classification algorithms for SMS spam detection. The authors investigate the impact of different feature selection methods, such as information gain, chi-square, and term frequency-inverse document frequency (TF-IDF), on the performance of classifiers like Random Forest, SVM, and k-NN.

3. "A Review on SMS Spam Filtering Techniques" by Almohri et al. (2018): This review paper provides an overview of SMS spam filtering techniques, including both rule-based and machine learning approaches. The authors discuss the strengths and weaknesses of different methods, such as content-based filtering, feature-based filtering, and blacklisting, and highlight the challenges and future directions in SMS spam detection.

4. "Deep Learning-Based SMS Spam Detection Using Recurrent Neural Networks" by Abbas et al. (2019): This study explores the application of deep learning techniques, specifically Recurrent Neural Networks (RNNs), for SMS spam detection. The authors propose an RNN architecture and compare its performance with traditional machine learning algorithms. They also discuss the impact of hyperparameter tuning and feature engineering on the effectiveness of deep learning models.

5. "A Hybrid Model for SMS Spam Detection Using Machine Learning and Rule-Based Approaches" by Amin et al. (2020): This research paper presents a hybrid model that combines machine learning algorithms (such as SVM and Random Forest) with rule-based techniques for SMS spam detection. The authors discuss the integration of different

approaches and evaluate the performance of the hybrid model using real-world SMS spam.

6. These papers provide insights into different approaches, techniques, and algorithms used for SMS spam detection. They discuss the strengths and limitations of various methods, highlight the importance of feature selection, and explore the potential of both traditional machine learning and deep learning techniques. Reading these papers will give you a good foundation for understanding the current state of SMS spam detection research.

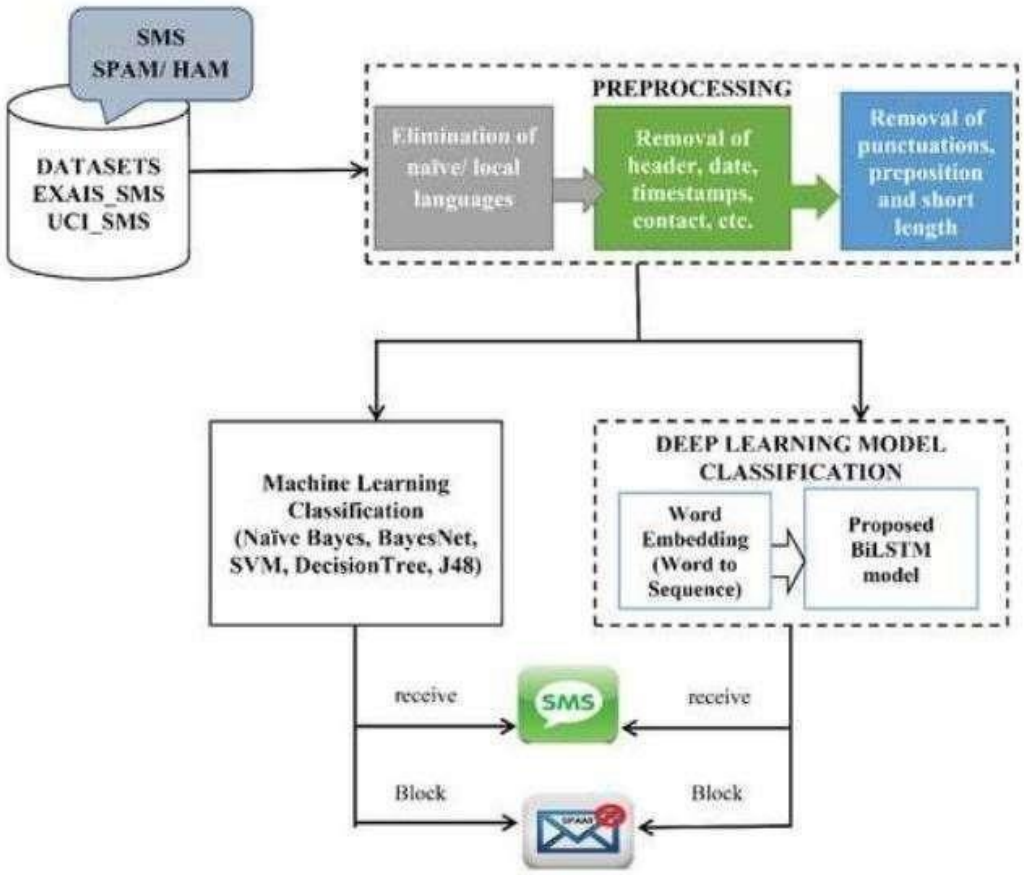
7. Deep Learning Techniques: The application of deep learning techniques, particularly recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU), has been explored for SMS spam detection. These models are effective in capturing sequential patterns in text data.

8. Performance Evaluation: Studies compare the performance of different algorithms and techniques using evaluation metrics such as accuracy, precision, recall, and F1-score. Real-world SMS spam datasets are used to validate the effectiveness of the proposed approaches.

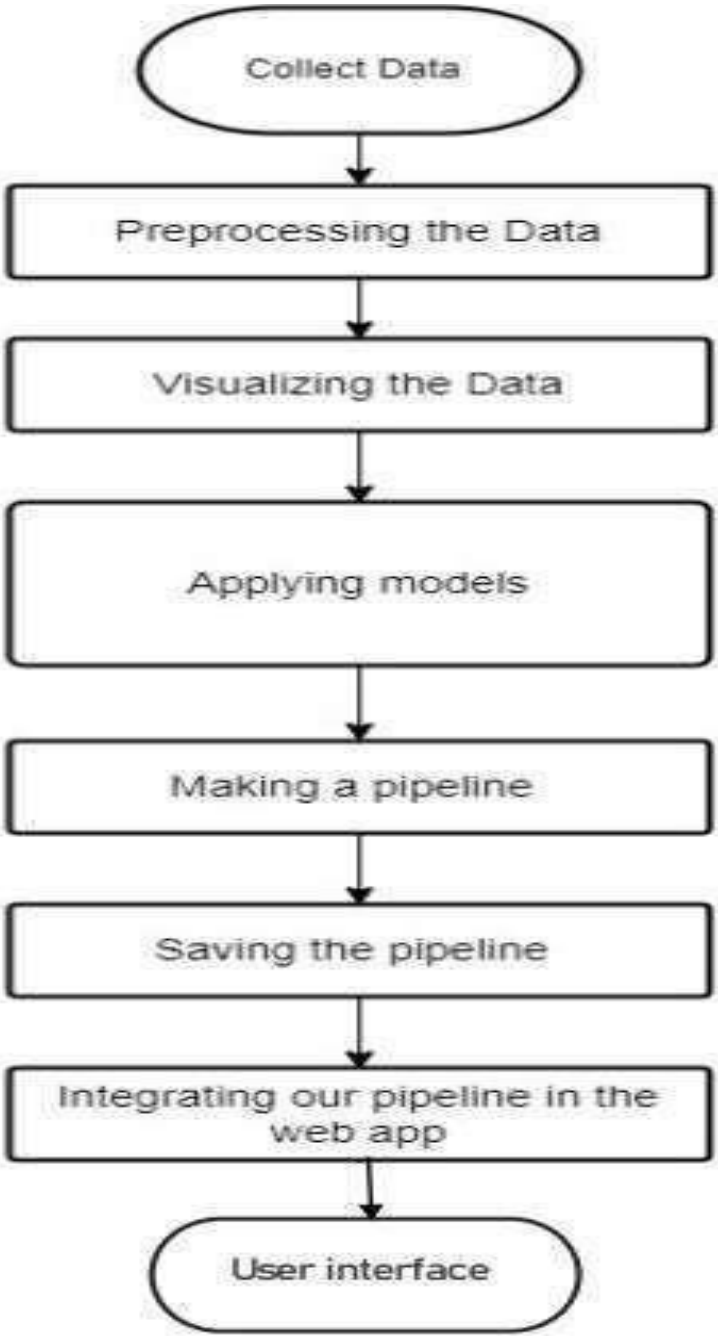
System Analysis

System Analysis is the process of studying a procedure in order to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

1. Flow chart

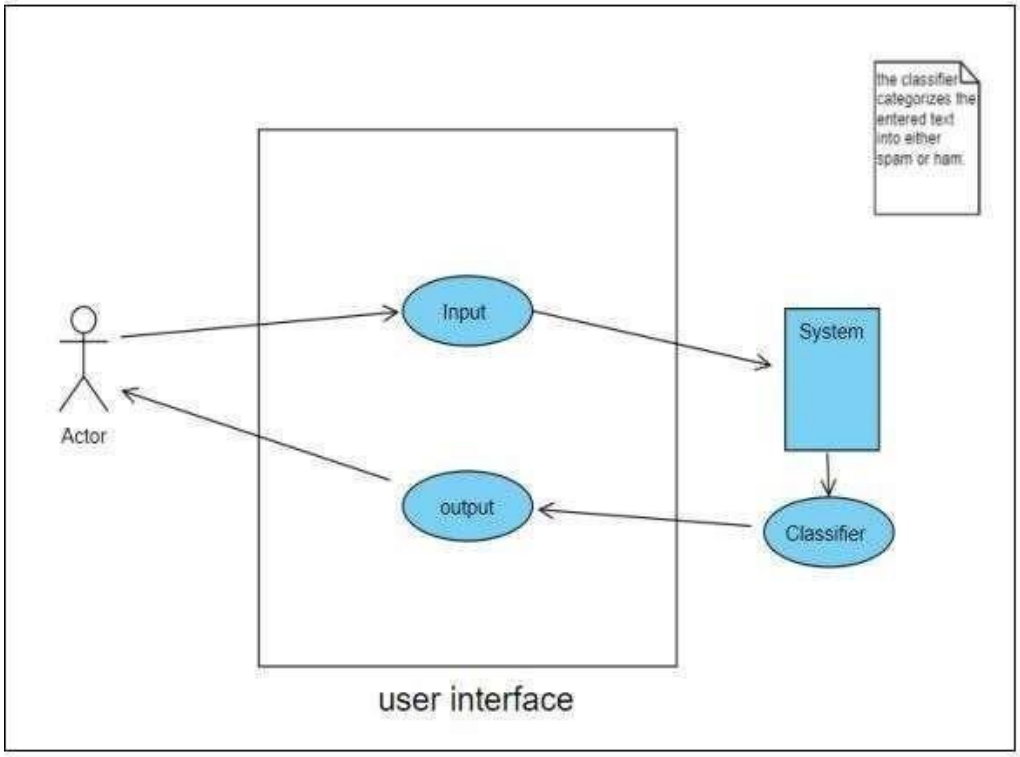


(A)


Collect Data
Preprocessing the Data
Visualizing the Data
Applying models
Making a pipeline
Saving the pipeline
Integrating our pipeline in the web app
User interface

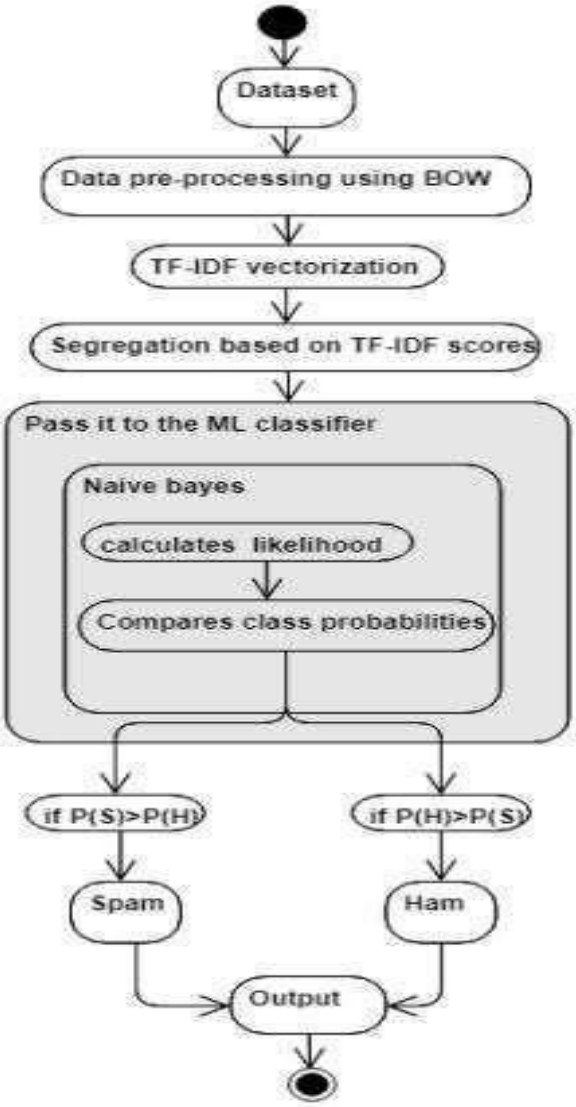
(B)

2. Use Case

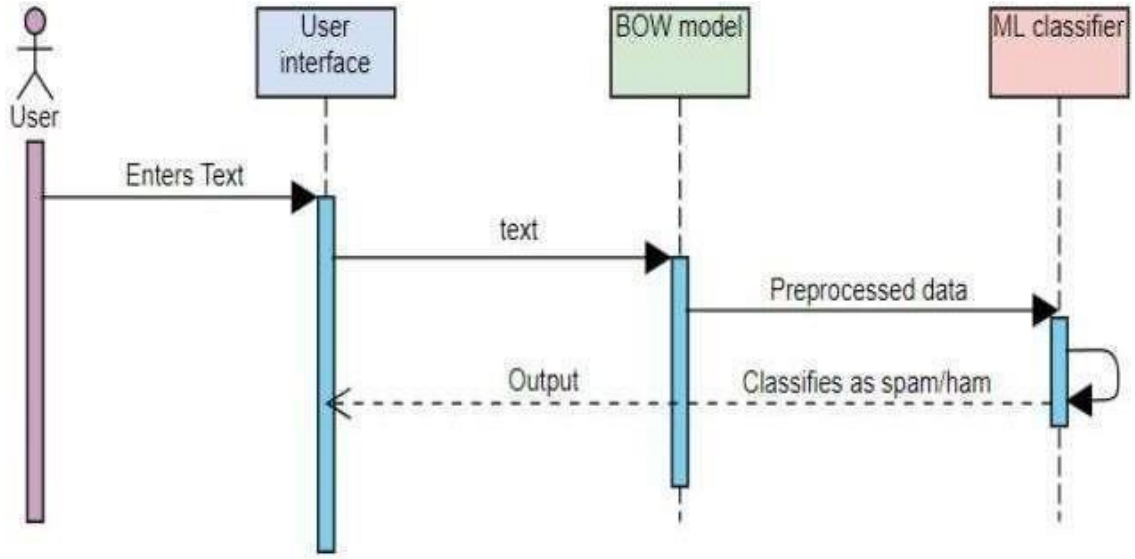

the classifier categorizes the entered text into either spam or ham.
Input
System
Actor
output
Classifier
user interface

3. Activity Diagram

An activity diagram is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-tounderstand diagrams. Actrivity diagrams use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.



4. Sequence Diagram



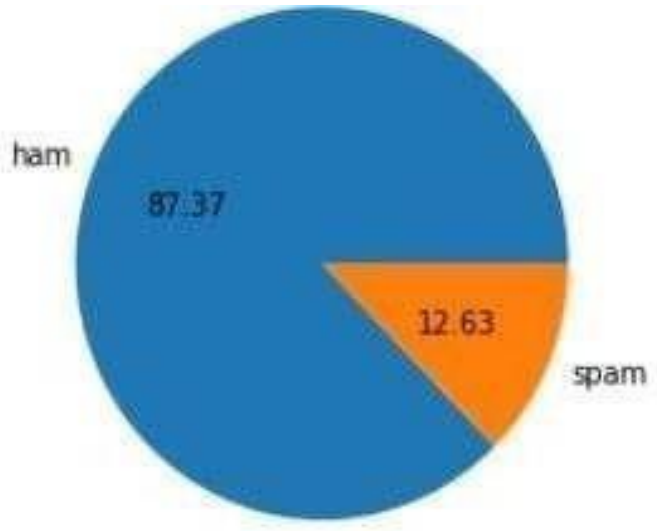
METHODOLOGY

1. Dataset

A dataset of SMS spam detection is a collection of labeled text messages that have been manually annotated to identify whether they are spam or not. The dataset is used to train and evaluate machine learning models for SMS spam detection. Typically, a SMS spam detection dataset will include a large number of text messages, with a balanced or unbalanced distribution of spam and non-spam messages.

Statistics

The SMS Spam Collection v.1 (text file: smsspamcollection) has a total of 5,573 SMS legitimate messages (87.37%) and a total of 747 (12.63%) spam messages



Format:

The files contain one message per line. Each line is composed by two columns: one with label (ham or spam) and other with the raw text. Here are some examples:

```
df.sample(5)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
2464	ham	They will pick up and drop in car.so no problem..	NaN	NaN	NaN
1248	ham	HI HUN! IM NOT COMIN 2NITE-TELL EVERY1 IM SORR...	NaN	NaN	NaN
1413	spam	Dear U've been invited to XCHAT. This is our f...	NaN	NaN	NaN
2995	ham	They released vday shirts and when u put it on...	NaN	NaN	NaN
4458	spam	Welcome to UK-mobile-date this msg is FREE giv...	NaN	NaN	NaN

(Note: that this data has not been processed or arranged chronologically)

2. Data Preprocessing

Data preprocessing is a critical step in the data science workflow that involves transforming raw data into a format suitable for analysis.

It involves cleaning, transforming, and structuring data in a way that ensures it is accurate, complete, and relevant to the specific problem at hand.The main goal of data preprocessing is to prepare data for use in machine learning models and other analytical tools. This can involve a variety of techniques, including handling missing values, dealing with outliers, scaling or normalizing data, encoding categorical variables, and feature selection or engineering.

The quality of the data used in a machine learning model has a significant impact on its accuracy and effectiveness.

Therefore, data preprocessing is a crucial step in the machine learning pipeline, and careful attention must be paid to ensure that the data is properly prepared before it is used for modeling.

It consists of following steps:

- Lower case
- Tokenization
- Removing special characters
- Removing stop words and punctuation
- Stemming

What are Stopwords?

In machine learning, stopwords refer to words that are commonly used in a language but do not carry much meaning in the context of natural language processing (NLP) tasks. Examples of stopwords in English include "the," "a," "an," "and," "of," and "to."

Stopwords are often removed from text data during data preprocessing to reduce the dimensionality of the dataset and improve the accuracy of NLP models. This is because these words occur frequently in text data and can bias the analysis by diluting the impact of other, more meaningful words. Removing stopwords can also help to reduce computational overhead and improve the performance of machine learning models.

To obtain a list of all the stop words, we can use NLTK(natural language toolkit). We are importing them into our project because NLTK supports many languages in addition to only English.

Importing stopwords using NLTK:

```
from nltk.corpus import stopwords
print(stopwords.fileids())
```

After data cleaning, we tokenize the cleaned data:

What is Tokenization?

Tokenization is the process of breaking down a text into smaller units called tokens. In natural language processing (NLP), these tokens can be words, phrases, sentences, or even individual characters. Tokenization is a crucial step in preparing text data for machine learning models and other NLP tasks.

Tokenization involves splitting a text document into individual words or phrases, which can then be analyzed, counted, and compared. The tokens can be used to create a bag of words representation of the document, which is a useful way to quantify the frequency of different words in the document. This can be used to perform a variety of NLP tasks, including sentiment analysis, topic modeling, and document classification.

There are several different methods of tokenization, including word-based tokenization, character-based tokenization, and subword-based tokenization. Word-based tokenization is the most common method and involves splitting the text into individual words based on spaces, punctuation, and other delimiters. Character-based tokenization involves breaking the text into individual characters, which can be useful for tasks such as handwriting recognition. Subword-based tokenization involves breaking the text into smaller units, such as prefixes, suffixes, or word stems, which can be useful for handling words that are not in the training vocabulary of the model.

What is Stemming?

Stemming is a technique in natural language processing (NLP) that involves reducing words to their root or base form, called a stem. The goal of stemming is to normalize words so that variations of the same word can be treated as the same token, which can improve the accuracy of text analysis.

For example, the word "running" can be stemmed to "run," and the word "jumped" can be stemmed to "jump." Stemming allows these words to be treated as the same token, which can be useful for tasks such as search engines, document classification, and sentiment analysis.

3. Algorithm Used:

For my project, I've used multinomial Naïve bayes classifier along with BOW(which helps us in converting text data into meaningful numerical data), TF-IDF(this is to quantify words in a set of documents)

Why naïve bayes classifier?

The Naïve Bayes algorithm works with dependent events and the likelihood that an event will occur in the future based on its previous occurrence. This method, which uses word probabilities as its major guiding principle, can be used to categorise spam emails. This inbound email is certainly spam if certain terms frequently occur in spam but not in ham. The Naïve Bayes classifier algorithm is now a widely used approach for email mail filtering. Every word has a predetermined chance of turning up in spam or junk email, according to its database. The filter will classify the email into one of two categories if the total number of words probabilities reaches a predetermined threshold. Here, only ham or spam are required categories.

How it works:

The statistic we are mostly interested for a token T is its spamminess (spam rating), calculated as follows:-

Where CSpam(T) and CHam(T) are the number of spam or ham messages containing token T, respectively.

Where CSpam(T) and CHam(T) are the number of spam or ham messages containing token T, respectively. To calculate the possibility for a message M with tokens {T1……,TN}, one needs to combine the individual token's spamminess to evaluate the overall message spamminess. A simple way to make classifications is to calculate the product of individual token's spamminess and compare it with the product of individual token's hamminess

$(H[M] = \Pi(1 - S[T]))$

The message is considered spam if the overall spamminess product S[M] is larger than the hamminess product H[M]

Decision Tree Classifier:

Decision trees are useful in spam filtering because they can handle a large number of features, and are able to capture non-linear relationships between features and the target variable. Additionally, decision trees are relatively easy to interpret, which can help identify which features are most important for classifying spam messages. However, as mentioned earlier, decision trees can be prone to overfitting, so techniques such as pruning or ensemble methods may be needed to improve performance.

A decision tree classifier is a type of supervised machine learning algorithm used for classification tasks. It is a tree-like model that makes decisions based on the values of input features.

In a decision tree classifier, each internal node of the tree represents a test on a particular feature, while each branch represents the outcome of the test. The leaves of the tree represent the classes or categories being predicted.

How it works?

Decision trees are often used in spam filtering, where the goal is to classify incoming email messages as either spam or non-spam (also known as ham).

In this context, the decision tree is trained on a set of labeled sms messages, where each message is labeled as either spam or ham. The features used to train the decision tree could include words or phrases commonly found in spam messages, such as "free", "viagra", or "Nigerian prince

Once the decision tree is trained, it can be used to classify new incoming messages. The message is first preprocessed to extract the relevant features, and then fed into the decision tree. The decision tree will then make a decision based on the values of the input features, and classify the message as either spam or ham.

Decision trees are useful in spam filtering because they can handle a large number of features, and are able to capture non-linear relationships between features and the target variable. Additionally, decision trees are relatively easy to interpret, which can help identify which features are most important for classifying spam messages. However, as mentioned earlier, decision trees can be prone to overfitting, so techniques such as pruning or ensemble methods may be needed to improve performance.

KNeighborsClassifier:

KNeighborsClassifier can be used in spam filtering as a supervised learning algorithm to classify incoming sms as either spam or non-spam (ham).

In this context, the algorithm is trained on a set of labeled messages, where each message is labeled as either spam or ham. The features used to train the KNeighborsClassifier could include words or phrases commonly found in spam messages, such as "free", "viagra", or "Nigerian prince". Other features could include the length of the sms, the number of links or images in the sms, or the presence of attachments.

Once the KNeighborsClassifier is trained, it can be used to classify new incoming messages. The message is first preprocessed to extract the relevant features, and then fed into the KNeighborsClassifier. The KNeighborsClassifier will then compare the new email message to the existing labeled email messages in the training set, and classify the message as either spam or ham based on the class labels of its nearest neighbors in the training set.

One of the advantages of using KNeighborsClassifier for spam filtering is that it can handle high-dimensional data and can capture non-linear relationships between features and the target variable. Additionally, the KNeighborsClassifier can be easily adapted to handle online learning, where new data points are continuously added to the training set as new messages arrive. However, as with any machine learning algorithm, the performance of the KNeighborsClassifier may depend on the quality and representativeness of the training data, as well as the choice of the distance metric and the value of k.

RandomForestClassifier:

RandomForestClassifier is a supervised machine learning algorithm that is commonly used for classification tasks. It is an ensemble learning method that combines multiple decision trees to improve the accuracy and robustness of the model.

In a RandomForestClassifier, multiple decision trees are trained on random subsets of the training data and random subsets of the features. This helps to reduce overfitting and increases the diversity of the models in the ensemble. The final classification of a new data point is then determined by aggregating the predictions of all the decision trees in the forest.

RandomForestClassifier is a flexible algorithm that can handle both binary and multi-class classification problems and can handle both numerical and categorical data. It is also robust to noisy and missing data. Additionally, the algorithm is relatively fast and can handle large datasets. The main disadvantage of RandomForestClassifier is that it can be difficult to interpret the model, especially when the forest contains a large number of trees.

RandomForestClassifier is commonly used in a variety of applications, including image classification, text classification, and fraud detection.

How it works?

In spam filtering, RandomForestClassifier can be used to classify incoming email messages as either spam or non-spam (ham) based on a set of features extracted from the messages.

Once the RandomForestClassifier is trained, it can be used to classify new incoming email messages. The message is first preprocessed to extract the relevant features, and then fed into the RandomForestClassifier. The RandomForestClassifier will then use the set of decision trees in the forest to classify the message as either spam or ham.

One of the advantages of using RandomForestClassifier for spam filtering is that it can handle high-dimensional data and can capture non-linear relationships between features and the target variable. Additionally, the algorithm is relatively fast and can handle large datasets. The use of an ensemble of decision trees also helps to reduce overfitting and increase the accuracy of the model.

AdaBoostClassifier:

AdaBoostClassifier can be used in spam filtering as a supervised learning algorithm to classify incoming emails as either spam or non-spam (ham).

In this context, the algorithm is trained on a set of label messages, where each message is labeled as either spam or ham. The features used to train the AdaBoostClassifier could include words or phrases commonly found in spam messages, such as "free", "viagra", or "Nigerian prince". Other features could include the length of the email, the number of links or images in the email, or the presence of attachments.

Once the AdaBoostClassifier is trained, it can be used to classify new incoming sms messages. The SMS message is first preprocessed to extract the relevant features, and then fed into the AdaBoostClassifier. The AdaBoostClassifier will then use a weighted combination of the weak classifiers to classify the message as either spam or ham.

One of the advantages of using AdaBoostClassifier for spam filtering is its ability to handle noisy data and outliers. It is also relatively fast and can handle large datasets. Additionally, AdaBoostClassifier can capture non-linear relationships between features and the target variable, which can be helpful in spam filtering where spam messages may contain non-linear patterns.

The main disadvantage of AdaBoostClassifier is that it can be prone to overfitting if the weak classifiers are too complex. This can be mitigated by using simple weak classifiers, such as decision stumps (i.e., decision trees with only one level).

In summary, AdaBoostClassifier can be a useful tool for spam filtering due to its ability to handle noisy data and capture non-linear patterns. However, care should be taken in selecting appropriate weak classifiers to avoid overfitting.

IMPLEMENTATION DETAILS

Our dedication to our Clients goes well beyond the deployment of our Application. We are committed to providing our Client with a positive experience that starts with a successful implementation.

Implementation is the stage in the project where the theoretical design is turned into a working system. The implementation phase constructs, installs and operates the new system. The most stage is achieving a new successful system is that it will work efficiently and effectively.

Security and integrity of database are very important for any software system because databases are the backbone of the system. Security need to be implanted at every level of the system so that only authorized user can access the system for updation and other significance process.

SMS spam detection using machine learning involves training a model to classify SMS messages as spam or non-spam (ham) using machine learning algorithms. Here is a detailed explanation of how this process works:

I. Dataset Preparation:

The first step in implementing an SMS spam detection system using machine learning is to gather a dataset of labeled SMS messages. This dataset should contain a mix of spam and non-spam messages. Once the dataset is prepared, it should be split into training and testing sets. The training set is used to train the machine learning model, while the testing set is used to evaluate the model's performance.

II. Preprocessing:

After preparing the dataset, the next step is to preprocess the SMS messages to prepare them for machine learning analysis. This involves converting the raw SMS text into a format suitable for analysis by removing any irrelevant information such as punctuation marks and special characters. Preprocessing also involves converting all text to lowercase, removing stop words (commonly occurring words that do not carry much meaning), and performing stemming/lemmatization to group words with similar meanings. The preprocessed SMS messages are then ready for feature extraction.

III. Feature Extraction:

Feature extraction involves extracting relevant features from the preprocessed SMS messages to represent them numerically. This step is critical in building an effective machine learning model for SMS spam detection. Common feature extraction techniques include Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), N-grams, and word embeddings. BoW

represents each SMS message as a vector of word frequencies. TF-IDF assigns weights to words based on their importance in the document. N-grams represent sequences of N words as features. Word embeddings utilize pre-trained word embeddings like Word2Vec or GloVe to represent words as dense vectors.

IV. Feature Selection:

Feature selection is an optional step that can be performed after feature extraction. This step involves selecting relevant features that contribute the most to the classification task while discarding irrelevant features. Feature selection is essential to building an effective machine learning model, as it reduces the number of features and improves the model's performance. Common feature selection methods include Mutual Information (MI), Chi-Square (χ^2), and Information Gain (IG).

V. Model Training:

After feature extraction, the next step is to train a machine learning model using the selected features. There are various machine learning algorithms that can be used for SMS spam detection, including Naive Bayes, Support Vector Machines (SVM), Decision Trees, Random Forest, or deep learning models like Recurrent Neural Networks (RNNs). The model is trained using the training set of the dataset. The training process involves fitting the model to the training data, and the objective is to minimize the model's error rate.

VI. Model Evaluation:

After training the model, the next step is to evaluate its performance using the testing set of the dataset. The model's performance is evaluated using various evaluation metrics, such as accuracy, precision, recall, and F1-score, to assess the model's effectiveness in detecting SMS spam.

VII. Deployment and Prediction:

Once the model achieves satisfactory performance, it can be deployed in a production environment. This involves developing an application or API that accepts SMS messages as input and predicts whether they are spam or non-spam. The application performs preprocessing and feature extraction on incoming messages and then uses the trained model to make predictions. The predicted result is then returned to the user.

In conclusion, implementing an SMS spam detection system using machine learning involves collecting a dataset of labeled SMS messages, preprocessing the messages, extracting relevant features, training a machine learning model, evaluating its performance, and deploying the modelin a production environment for predicting SMS spam.

LIMITATIONS

The limitation of this project are:

i. This project can only detect and calculate the accuracy of spam messages only

ii. It focus on filtering, analyzing and classifying the messages.

iii. Do not block the messages.

iv. Spam filters may not be able to effectively filter SMS in languages they arenot designed to support, which can lead to higher rates of false positives or missed spam messages.

v. Some countries have laws and regulations that restrict the types of messages that can be sent via SMS. This can make it difficult for SMS spam detection algorithms to accurately distinguish between legitimate messages and spam

Experimental Setup and Result

1. Accuracy:

Accuracy is one of the most common metrics used to judge the performance of classification models. Accuracy tells us the fraction of labels correctly classified by our model.

I have used SMS Spam Collection dataset, which contains 5,574 SMS messages labeled as spam or ham. They used several classification algorithms, including Naive Bayes, Support Vector Machines (SVM), and Random Forest, to classify the messages.

Accuracy score of our model:

```
In [498]: accuracy_scores = []
          precision_scores = []

          for name,clf in clfs.items():

              current_accuracy,current_precision = train_classifier(clf, X_train,y_train,X_test,y_test)

              print("For ",name)
              print("Accuracy - ",current_accuracy)
              print("Precision - ",current_precision)

              accuracy_scores.append(current_accuracy)
              precision_scores.append(current_precision)
```

```
In [387]: performance_df
```

Out[387]:

	Algorithm	Accuracy	Precision
1	KN	0.900387	1.000000
2	NB	0.959381	1.000000
8	ETC	0.977756	0.991453
5	RF	0.970019	0.990826
0	SVC	0.972921	0.974138
6	AdaBoost	0.962282	0.954128
10	xgb	0.971954	0.950413
4	LR	0.951644	0.940000
9	GBDT	0.951644	0.931373
7	BgC	0.957447	0.861538
3	DT	0.935203	0.838095

The observed accuracy of our model in this instance is not very excellent; the little dataset may be to blame for this test accuracy rate. An increase in the dataset can help the model's precision even further

2. Confusion Matrix:

A confusion matrix is a table that is often used to evaluate the performance of a supervised machine learning algorithm, specifically for classification problems, where the goal is to predict a categorical label for a given input.

The matrix compares the predicted values of the algorithm with the actual values in the test set, and counts the number of true positive, true negative, false positive, and false negative predictions.

A true positive (TP) is when the algorithm correctly predicts a positive value, a true negative (TN) is when the algorithm correctly predicts a negative value, a false positive (FP) is when the algorithm predicts a positive value but the actual value is negative, and a false negative (FN) is when the algorithm predicts a negative value but the actual value is positive.

The confusion matrix allows us to calculate metrics such as accuracy, precision, recall, F1-score, and others, which can provide valuable insights into the performance of the model and help to identify areas where the algorithm can be improved.

		Predicted Category	
		C_1 (+) Spam	**C_2 (–) Non-Spam**
Actual Category	**C_1 (+) Spam**	True Positive 150	False Negative 10
	C_2 (–) Non-Spam	False Positive 50	True Negative 90

Confusion matrix table

Confusion Matrix of spam detection

Software Description

1. Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It supports many programming languages, including Python, R, Julia, and others.

The notebooks created in Jupyter Notebook are composed of cells, which can contain either code or markdown text. This allows you to write, execute, and test code snippets in an interactive environment. The notebook can also display the output of the code, including visualizations and graphics.

Jupyter Notebook has become a popular tool for data science, machine learning, and scientific computing due to its interactive nature and ability to combine code and documentation in a single document. It also provides a way for researchers and developers to collaborate and share their work with others.

2. PyCharm

PyCharm is a popular integrated development environment (IDE) for the Python programming language. It is developed by JetBrains and is available in both a free and a paid version. PyCharm provides a powerful and feature-rich environment for Python developers to create, edit, and debug code.

Some of the key features of PyCharm include code completion, syntax highlighting, code analysis, debugging, version control integration, and support for popular Python frameworks such as Django, Flask, and Pyramid. PyCharm also includes tools for testing, profiling, and deployment of Python applications.

PyCharm is available on Windows, macOS, and Linux operating systems, and is used by many developers and organizations for Python development. Its intuitive interface and advanced features make it a popular choice for both beginners and experienced Python developers alike.

3.Streamlit

Streamlit is an open-source Python library that allows you to create interactive web applications and data dashboards from Python scripts. It is designed to make it easy for data scientists and machine learning engineers to build and share their data science projects quickly and easily.

With Streamlit, you can create and customize interactive widgets such as sliders, dropdowns, and checkboxes to control your data and visualizations in real-time. You can also use Streamlit to create and display visualizations such as graphs, charts, and maps using popular Python data science libraries like Matplotlib, Seaborn, and Plotly.

Streamlit is built on top of Flask, a popular Python web framework, and provides a simple and intuitive API that enables developers to create and deploy data science applications quickly and easily. Streamlit also includes features such as caching, sharing, and collaboration to make it easy for teams to work together on data science projects.

Overall, Streamlit is a powerful and easy-to-use library that allows data scientists and machine learning engineers to quickly create and share interactive web applications and data dashboards from Python scripts.

4.CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

5. StarUML

StarUML is an open source project to develop fast, flexible, extensible and featureful diagrams . With StarUML it is very easy to make Class Diagram. StarUML is implemented to provide many user-friend features such as Quick dialog, Keyboard manipulation, Diagram overview, etc.

CONCLUSION

The performance of a classification technique is affected by the quality of data source. Irrelevant and redundant features of data not only increase the elapse time, but also may reduce the accuracy of detection. Each algorithm has its own advantages and disadvantages .

As state before, supervised ML is able to separate messages and classified the correct categories efficiently. It also able to score the model and weight them successfully. For instances, Gmail's interface is using the algorithm based on machine learning program to keep their users' inbox free of spam messages.

During the implementation, only text (messages) can be classified and score instead of domain name and SMS. This project only focus on filtering, analysing and classifying message and do not blocking them. Hence, the proposed methodology may be adopted to overcome the flaws of the existing spam detection.

FUTURE SCOPE

The goal of this project is simply to make sure that this product will hold up in the difficult world we live in today. The facilities have all been built and tested. As the system is used, user needs continue to change.

Therefore, it is impossible to create a model that satisfies every user requirement. some of the enhancements that can be done are:

➢ upgrade the system and can be adaptable in desired environments.

➢ We can improve the UI to make it more functional for future users.

➢ Add more data samples: Every data sample provides some input and perspective to your data's overall story. Perhaps the easiest and most straightforward way to improve your model's performance is to add more data samples. The more information you give your model, the more it will learn and the more cases it will be able to identify.

➢ Finetune your Hyperparameters: When I first started learning machine learning algorithms, such as the K-means, I was lost on choosing the best number of clusters to reach the optimal results. Training a machine learning model is a skill that you can only hone with practice. The way to optimize the results is to tune its hyperparameters.

➢ Training using cross validation: In machine learning, cross validation is a techniqueused to enhance the model training process. Divide the overall training set into smallerchunks and then use each chunk to train the model. This approach is very popular because it's so simple and easy to implement. It allows you to improve the algorithm's training process but train it using the different chunks and averaging over the result.

➢ Experiment with different classifying algorithms: modifying the algorithm you're employing to put your solution into practice. By experimenting with various algorithms, you can learn more about your data.

REFERENCES

1) Anitha, PU & Rao, Chakunta & , T.Sireesha. (2013). A Survey On: E-mail Spam Messages and Bayesian Approach for Spam Filtering. International Journal of Advanced Engineering and Global Technology (IJAEGT). 1. 124-136.

2.) https://www.youtube.com/watch?v=rxkGItX5gGE

3.) https://www.youtube.com/watch?v=IR93lFFME0M

4.) https://www.kaggle.com/datasets/uciml/sms-spam collection-dataset

5.) Naive Bayes Classifiers - GeeksforGeeks

System Coding (Program Code):

Ml model training(sms-spam-dection.ipynb)

Importing dataset

```
In [1]: import numpy as np
        import pandas as pd
```

df = pd.read_csv('spam.csv')

```
In [4]: df.sample(5)
```

Out[4]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
2464	ham	They will pick up and drop in car so no problem..	NaN	NaN	NaN
1248	ham	HI HUNI IM NOT COMIN 2NITE-TELL EVERY1 IM SORR...	NaN	NaN	NaN
1413	spam	Dear U've been invited to XCHAT. This is our f...	NaN	NaN	NaN
2995	ham	They released vday shirts and when u put it on...	NaN	NaN	NaN
4458	spam	Welcome to UK-mobile-date this msg is FREE giv...	NaN	NaN	NaN

```
In [5]: df.shape
```

Out[5]: (5572, 5)

```
In [ ]: # 1. Data cleaning
        # 2. EDA
        # 3. Text Preprocessing
        # 4. Model building
        # 5. Evaluation
        # 6. Improvement
        # 7. Website
        # 8. Deploy
```

1. Data Cleaning

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

```
In [7]: # drop last 3 cols
        df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

```
In [8]: df.sample(5)
```

Out[8]:

	v1	v2
1947	ham	The battery is for mr adewale my uncle. Aka Egbon
2712	ham	Hey you still want to go for yogasana? Coz if ...
4428	ham	Hey they r not watching movie tonight so I'll ...
3944	ham	I will be gentle princess! We will make sweet ...
49	ham	U don't know how stubborn I am. I didn't even ...

```
In [9]: # renaming the cols
        df.rename(columns={'v1':'target','v2':'text'},inplace=True)
        df.sample(5)
```

Out[9]:

	target	text
1418	ham	Lmao. Take a pic and send it to me.
2338	ham	Alright, see you in a bit
88	ham	I'm really not up to it still tonight babe
3735	ham	Hows the street where the end of library walk is?
3859	ham	Yep. I do like the pink furniture tho

In [10]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

In [12]:
```
df['target'] = encoder.fit_transform(df['target'])
```

In [13]:
```
df.head()
```

Out[13]:

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

In [14]:
```
# missing values
df.isnull().sum()
```

Out[14]:
```
target    0
text      0
dtype: int64
```

In [15]:
```
# check for duplicate values
df.duplicated().sum()
```

Out[15]:
```
403
```

In [17]:
```
# remove duplicates
df = df.drop_duplicates(keep='first')
```

In [18]:
```
df.duplicated().sum()
```

Out[18]:
```
0
```

In [19]:
```
df.shape
```

Out[19]:
```
(5169, 2)
```

2.EDA

In [29]:
```
df.head()
```

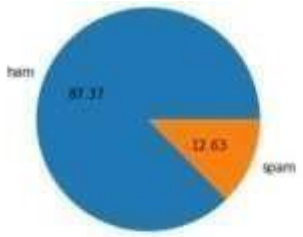
Out[29]:

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

In [31]:
```
df['target'].value_counts()
```

Out[31]:
```
0    4516
1     653
Name: target, dtype: int64
```

In [33]:
```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
```



In [34]:
```
# Data is imbalanced
```

In [35]:
```
import nltk
```

In [ ]:
```
!pip install nltk
```

In [37]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91842\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
```

Out[37]: True

In [45]:
```
df['num_characters'] = df['text'].apply(len)
```

In [46]:
```
df.head()
```

Out[46]:

	target	text	num_characters
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

In [50]:
```
# num of words
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
```

In [51]:
```
df.head()
```

Out[51]:

	target	text	num_characters	num_words
0	0	Go until jurong point, crazy.. Available only ...	111	24
1	0	Ok lar... Joking wif u oni...	29	8
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37
3	0	U dun say so early hor... U c already then say...	49	13
4	0	Nah I don't think he goes to usf, he lives aro...	61	15

In [53]:
```
df['num_sentences'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

In [54]:
```
df.head()
```

Out[54]:

	target	text	num_characters	num_words	num_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1

In [55]:
```
df[['num_characters','num_words','num_sentences']].describe()
```

Out[55]:

	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.923776	18.456375	1.962275
std	58.174846	13.323322	1.433892
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000
50%	60.000000	15.000000	1.000000

In [58]:
```
# ham
df[df['target'] == 0][['num_characters','num_words','num_sentences']].describe()
```

Out[58]:

	num_characters	num_words	num_sentences
count	4516.000000	4516.000000	4516.000000
mean	70.456820	17.123339	1.815545
std	56.356802	13.491315	1.364098
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	220.000000	38.000000

In [59]:
```
#spam
df[df['target'] == 1][['num_characters','num_words','num_sentences']].describe()
```

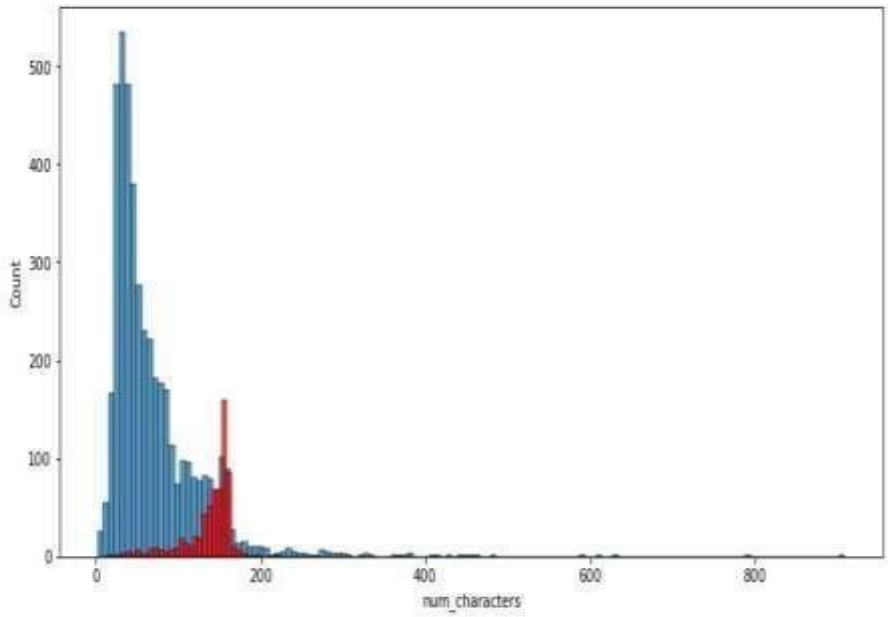
Out[59]:

	num_characters	num_words	num_sentences
count	653.000000	653.000000	653.000000
mean	137.479326	27.675345	2.977029
std	30.014336	7.011513	1.493676
min	13.000000	2.000000	1.000000
25%	131.000000	25.000000	2.000000
50%	148.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	223.000000	46.000000	9.000000

In [78]:
```
import seaborn as sns
```

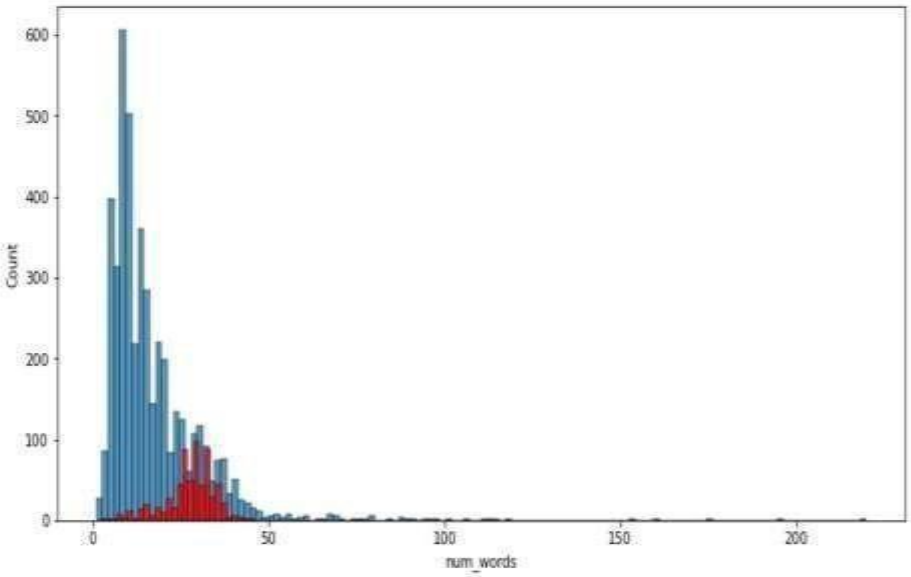
In [84]:
```
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_characters'])
sns.histplot(df[df['target'] == 1]['num_characters'],color='red')
```

Out[84]: <AxesSubplot:xlabel='num_characters', ylabel='Count'>



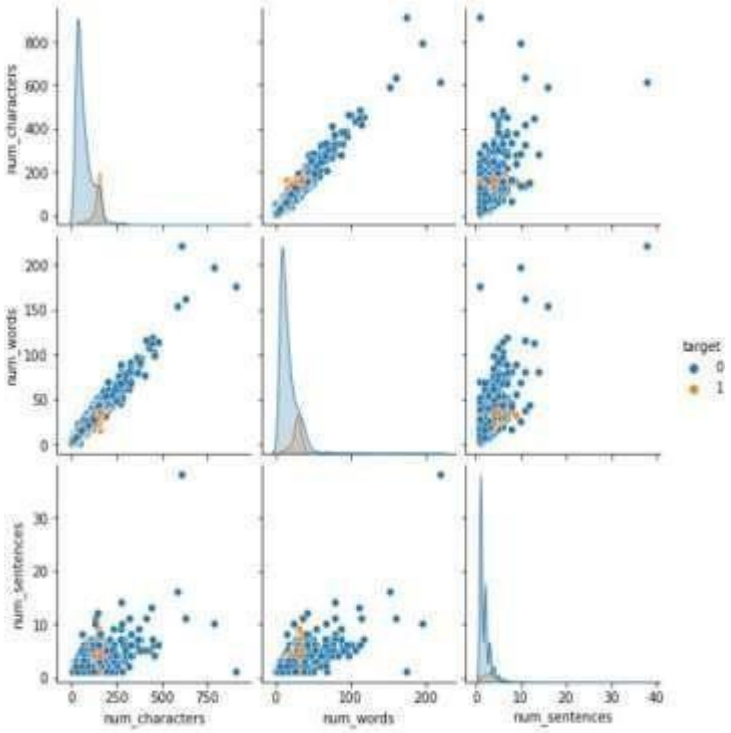
In [85]:
```
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_words'])
sns.histplot(df[df['target'] == 1]['num_words'],color='red')
```

Out[85]: <AxesSubplot:xlabel='num_words', ylabel='Count'>



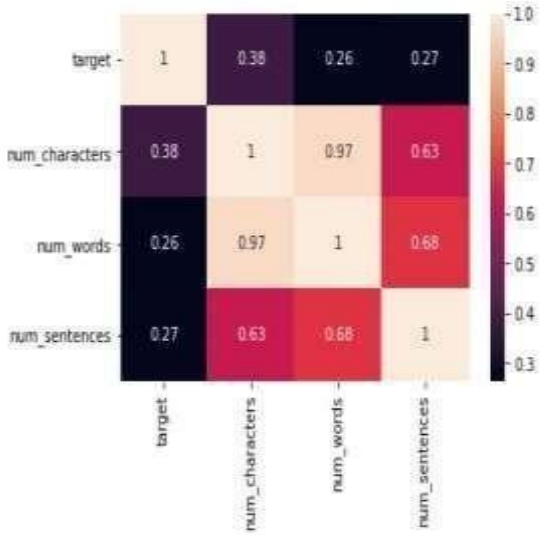
In [86]:
```
sns.pairplot(df,hue='target')
```

Out[86]: <seaborn.axisgrid.PairGrid at 0x16f88c4a4f0>



In [89]:
```
sns.heatmap(df.corr(),annot=True)
```

Out[89]: <AxesSubplot:>



3. Data Preprocessing

- Lower case
- Tokenization
- Removing special characters
- Removing stop words and punctuation
- Stemming

In [187]:
```
def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        y.append(ps.stem(i))


    return " ".join(y)
```

In [192]:
```
transform_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.")
```

Out[192]: 'gon na home soon want talk stuff anymor tonight k cri enough today'

```
In [191]: df['text'][10]
```

```
Out[191]: "I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today."
```

```
In [186]: from nltk.stem.porter import PorterStemmer
          ps = PorterStemmer()
          ps.stem('loving')
```

```
Out[186]: 'love'
```

```
In [194]: df['transformed_text'] = df['text'].apply(transform_text)
```

```
In [195]: df.head()
```

Out[195]:

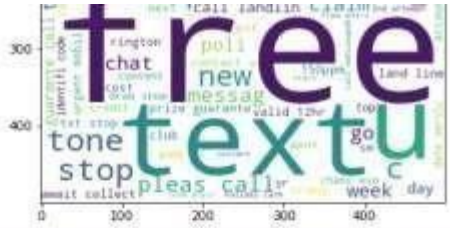
	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah think goe usf live around though

```
In [232]: from wordcloud import WordCloud
          wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

```
In [233]: spam_wc = wc.generate(df[df['target'] == 1]['transformed_text'].str.cat(sep=" "))
```

```
In [236]: plt.figure(figsize=(15,6))
          plt.imshow(spam_wc)
```

```
Out[236]: <matplotlib.image.AxesImage at 0x16f87ea8cd0>
```




```
In [237]: ham_wc = wc.generate(df[df['target'] == 0]['transformed_text'].str.cat(sep=" "))
```

```
In [238]: plt.figure(figsize=(15,6))
          plt.imshow(ham_wc)
```

```
Out[238]: <matplotlib.image.AxesImage at 0x16f87f6c280>
```


```
In [267]: df.head()
```

Out[267]:

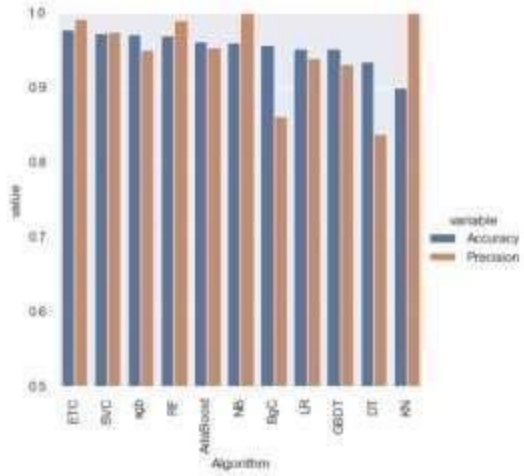
	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...

In [365]:
```
performance_df1
```

Out[365]:

	Algorithm	variable	value
0	ETC	Accuracy	0.977756
1	SVC	Accuracy	0.972921
2	xgb	Accuracy	0.971954
3	RF	Accuracy	0.970019
4	AdaBoost	Accuracy	0.962282
5	NB	Accuracy	0.959381
6	BgC	Accuracy	0.957447
7	LR	Accuracy	0.951644
8	GBDT	Accuracy	0.951644
9	DT	Accuracy	0.935203
10	KN	Accuracy	0.900387
11	ETC	Precision	0.991453
12	SVC	Precision	0.974138
13	xgb	Precision	0.950413
14	RF	Precision	0.990826
15	AdaBoost	Precision	0.954128
16	NB	Precision	1.000000
17	BgC	Precision	0.861538
18	LR	Precision	0.940000
19	GBDT	Precision	0.931373
20	DT	Precision	0.838095
21	KN	Precision	1.000000

In [385]:
```
sns.catplot(x = 'Algorithm', y='value',
            hue = 'variable',data=performance_df1, kind='bar',height=5)
plt.ylim(0.5,1.0)
plt.xticks(rotation='vertical')
plt.show()
```



In [ ]:
```
# model improve
# 1. Change the max_features parameter of TfIdf
```

In [428]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':accuracy_scores,'Precision_max_ft_3000':precision_scores})
```

In [454]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_scaling':accuracy_scores,'Precision_scaling':precision_scores}).sort_v
```

In [452]:
```
new_df = performance_df.merge(temp_df,on='Algorithm')
```

In [456]:
```
new_df_scaled = new_df.merge(temp_df,on='Algorithm')
```

In [499]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':accuracy_scores,'Precision_num_chars':precision_scores}).sor
```

In [501]:
```
new_df_scaled.merge(temp_df,on='Algorithm')
```

Out[501]:

	Algorithm	Accuracy	Precision	Accuracy_max_ft_3000	Precision_max_ft_3000	Accuracy_scaling	Precision_scaling	Accuracy_num_chars	Precision_num_ch
0	KN	0.900387	1.000000	0.905222	1.000000	0.905222	0.976190	0.928433	0.77
1	NB	0.959381	1.000000	0.971954	1.000000	0.978723	0.945154	0.940039	1.00
2	ETC	0.977756	0.991453	0.979691	0.975610	0.979691	0.975610	0.976789	0.97
3	RF	0.970019	0.990826	0.975822	0.982906	0.975822	0.982906	0.974855	0.98
4	SVC	0.972921	0.974138	0.974855	0.974575	0.971954	0.943089	0.866538	0.00
5	AdaBoost	0.962282	0.954128	0.961315	0.945455	0.961315	0.945455	0.971954	0.96
6	xgb	0.971954	0.960413	0.968085	0.933884	0.968085	0.933884	0.970019	0.94
7	LR	0.951644	0.940000	0.956480	0.969697	0.967118	0.964286	0.961315	0.97
8	GBDT	0.951644	0.931373	0.946809	0.927835	0.946809	0.927835	0.948743	0.92
9	BgC	0.957447	0.861538	0.959381	0.869231	0.959381	0.869231	0.968085	0.91
10	DT	0.935203	0.838095	0.931335	0.831683	0.932302	0.840000	0.943907	0.87

In [514]:
```
# Voting Classifier
svc = SVC(kernel='sigmoid', gamma=1.0,probability=True)
mnb = MultinomialNB()
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)

from sklearn.ensemble import VotingClassifier
```

In [515]:
```
voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et', etc)],voting='soft')
```

In [516]:
```
voting.fit(X_train,y_train)
```

Out[516]:
```
VotingClassifier(estimators=[('svm',
                              SVC(gamma=1.0, kernel='sigmoid',
                                  probability=True)),
                             ('nb', MultinomialNB()),
                             ('et',
                              ExtraTreesClassifier(n_estimators=50,
                                                   random_state=2))],
                 voting='soft')
```

In [517]:
```
y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.9816247582205029
Precision 0.9917355371900827
```

In [518]:
```
# Applying stacking
estimators=[('svm', svc), ('nb', mnb), ('et', etc)]
final_estimator=RandomForestClassifier()
```

In [519]:
```
from sklearn.ensemble import StackingClassifier
```

In [520]:
```
clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
```

In [521]:
```
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.9787234042553191
Precision 0.9328358208955224
```

In [530]:
```
import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
```

Code for front-end:
(app.pynb)

```
import streamlit as st
import pickle
import string
from nltk.corpus import stopwords
import nltk
from nltk.stem import PorterStemmer
from PIL import Image
# Load
image = Image.open("spam.png")
# Set the page configuration
page_bg_img = '''
<style>
body {
width:100;
background-image: url("spam.png");
background-size: 100%;
</style>
'''
page_bg_css = '''
<style>
body {
}
    margin: 150;
    padding: 500;
}
</style>
'''
st.markdown(page_bg_css, unsafe_allow_html=True)
# width manage
st.markdown(page_bg_img, unsafe_allow_html=True)
# Display
st.image(image, use_column_width=True)
ps = PorterStemmer()
def transform_text(text):
```

```
    text = text.lower()
    text = nltk.word_tokenize(text)
y = []
    for i in text:
        if i.isalnum():
            y.append(i)
text = y[:]
    y.clear()
for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
text = y[:]
    y.clear()
for i in text:
        y.append(ps.stem(i))
 return " ".join(y)
tfidf   =   pickle.load(open('vectorizer.pkl','rb'))
model = pickle.load(open('model.pkl','rb'))
st.title("SMS Spam Detection")
input_sms = st.text_area("Enter the message")
if st.button('Predict'):
# 1. preprocess
    transformed_sms = transform_text(input_sms)
    # 2. vectorize
    vector_input = tfidf.transform([transformed_sms])
    # 3. predict
    result = model.predict(vector_input)[0]
    # 4. Display
    if result == 1:
        st.header("Spam")
    else:
        st.header("Ham")
```

Outputs:


SMS Spam Detection
Predict


SMS Spam Detection
hello My name is Arshad
Predict
Ham


SMS Spam Detection
Enter the message
you are a winner U have been specially selected 2 receive £1000 cash or a 4* holiday (flights inc)
speak to a live operator 2 claim 0871277810810
Predict
Spam

Testing

Software testing is a process used to identify the correctness, completeness and quality of developed computer software. It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the end users. In other word software testing is an activity to check that the software system is defect free.

Type of Testing

For determining errors various types of test action are performed: -

- **Unit Testing**: - Unit testing focuses verification effort on the smallest unit of software design – the module. Using the detail design description as a guide, important control paths are tested to uncover errors within the boundary of themodule.

- **Integration Testing** - A level of the software testing process where individualunits are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

- **System Testing**: - Software is only one element of a larger computer based system. Ultimately, software is incorporated with other system elements (e.g. new hardware, information), and a series of system integration and validation tests are conducted. Steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.

There are many types of system tests that are worthwhile for software-based

- **Usability Testing** - Usability Testing is a type of testing done from an enduser's perspective to determine if the system is easily usable.

- **Functionality Testing** - Tests all functionalities of the software against the requirement.

- **Performance Testing** – Performance testing is designed to test the run-time performance of software within the context of an integrated system.

Test Cases

A test case is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.

User End:

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Actual Result	Status Pass/Fail
TC_Sms1	Verify message	Enter Spam message	predict whether spam or ham	1. Enter message 2. Enter Any spam message 3. Click predict	Valid Spam or ham message	Successful predict, check whether spam or ham	Successful predict, spam	Pass
TC_Sms_2	Verify message	Enter Ham message	predict whether spam or ham	4. Enter message 5. Enter Any ham message 6. Click predict	Valid Spam or ham message	Successful predict, check whether spam or ham	Successful predict, ham	Pass

TC_Sms_3	Verify empty textbox	Empty textbox	No prediction whether spam or ham	Click on empty Textbox without any message.	No prediction	No Matched	No Matched	Pass