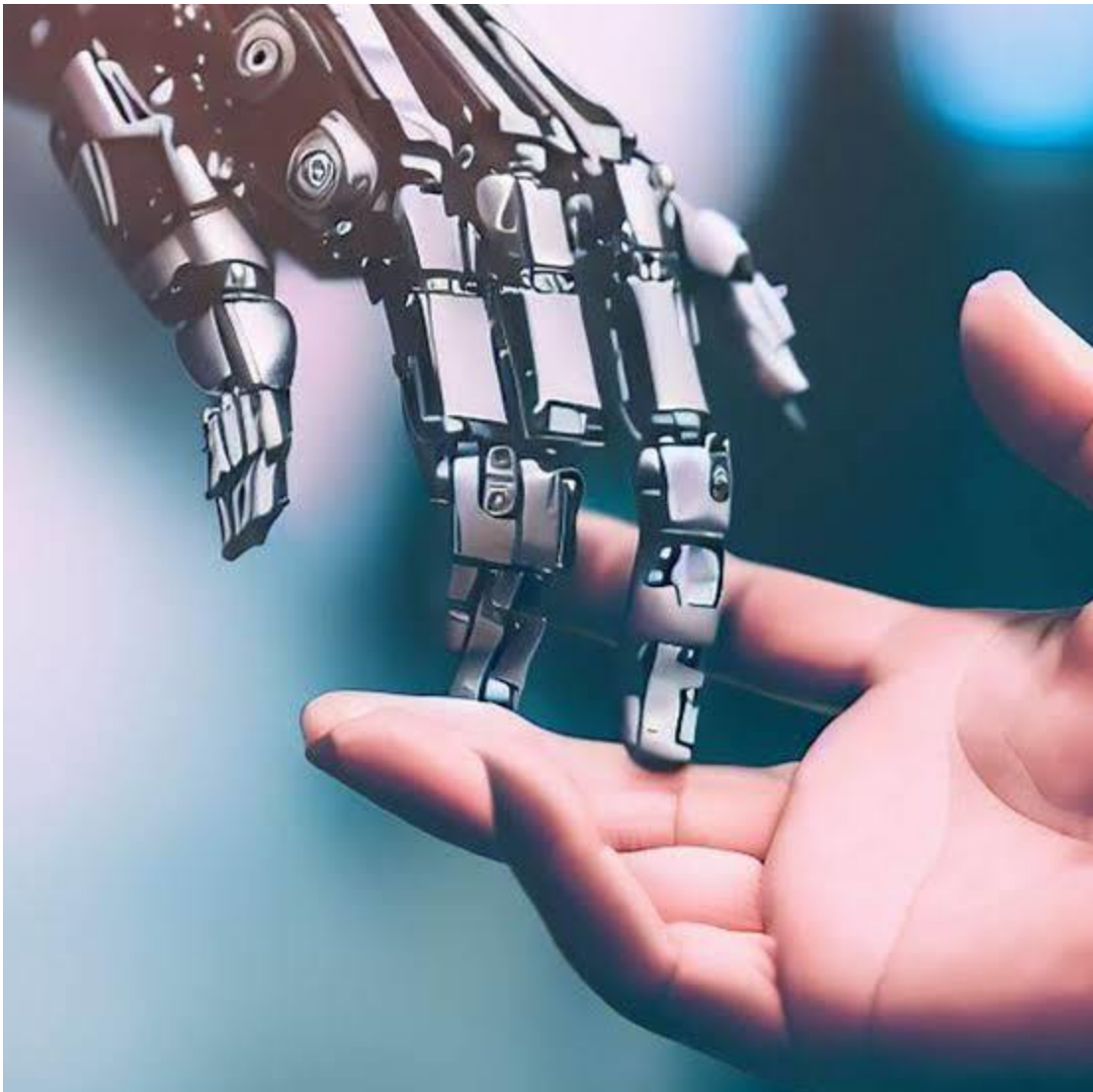


INNOVATION ON BUILDING A SMARTER AI POWERED SPAM CLASSIFIER.

Problem:

Explain in detail about the complete steps will be taken to put your design on innovation on building a smarter AI powered spam classifier.

Solution:



Spam classifier:

A spam classifier is a software or algorithm designed to automatically categorize incoming messages or content as either “spam” or “ham”

The parameters are:

Data Collection:

The classifier is trained on a dataset containing a large number of examples of both spam and non-spam messages.

Feature Extraction:

Features or attributes are extracted from the messages. these features can include words, phrases, sender information etc.

Training:

Using the labelled dataset, the classifier learns to recognize patterns and relationships between the features and spam categories. Common machine learning algorithms like Naïve Bayes, Support Vector Machines etc., can be used.



SPAM CLASSIFIER USING AI:

Creating a spam classifier using AI typically involves machine learning techniques.

1.Data Collection:

Gather a dataset of emails or messages, categorizing them as either spam or non-spam. This dataset will be used to train and test your classifier.

2.Data Preprocessing:

Clean and preprocess the text data. This may involve tasks like removing special characters, lower casting, tokenization and stemming or lemmatization.

3.Feature Extraction:

Convert the text data into numerical features that machine learning models can be understand. Common techniques include TF-IDF or word embeddings can be used.



4. Model selection:

Choose a machine learning algorithm to train on your dataset.

Common choices include SVM and Recurrent Neural Networks.

5. Training:

Split your dataset into a training set and a validation set. Train your chosen model on the training data, adjusting hyperparameters as needed.

6. Evaluation:

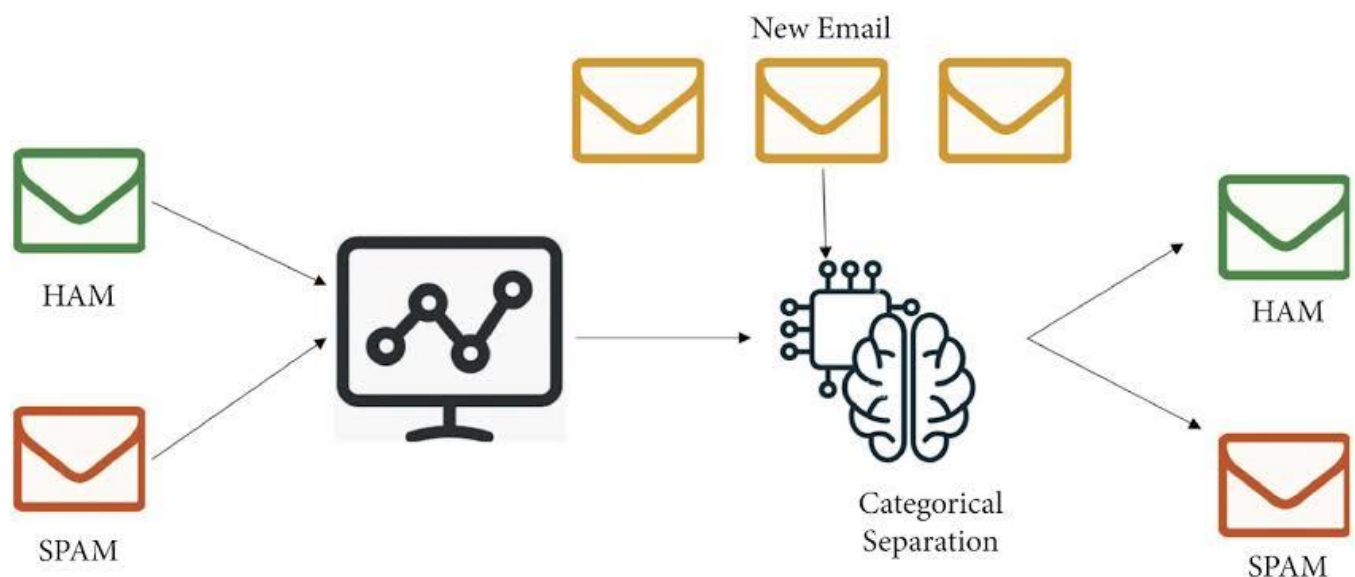
Evaluate the model's performance on a separate test dataset using metrics like accuracy, precision, recall and F1 score.

7. Deployment:

Once satisfied with the model's performance, deploy it in your application or email system to automatically classify incoming messages or not.

8. Continuous improvement:

Spam patterns change over time, so it is essential to monitor and update your classifier.



Innovation of building a smarter AI powered spam classifier:

To innovate building a smarter AI powered spam classifier requires a combination of advanced techniques and data driven approaches.

1. Collect Diverse Data:

Gather a diverse and extensive dataset of spam and non-spam messages. This should include different languages, formats and sources to improve the classifier's robustness.

2. Feature Engineering:

Extract relevant features from the text such as word frequency, n-grams, sender information, and email structure.



3. Machine Learning Algorithms:

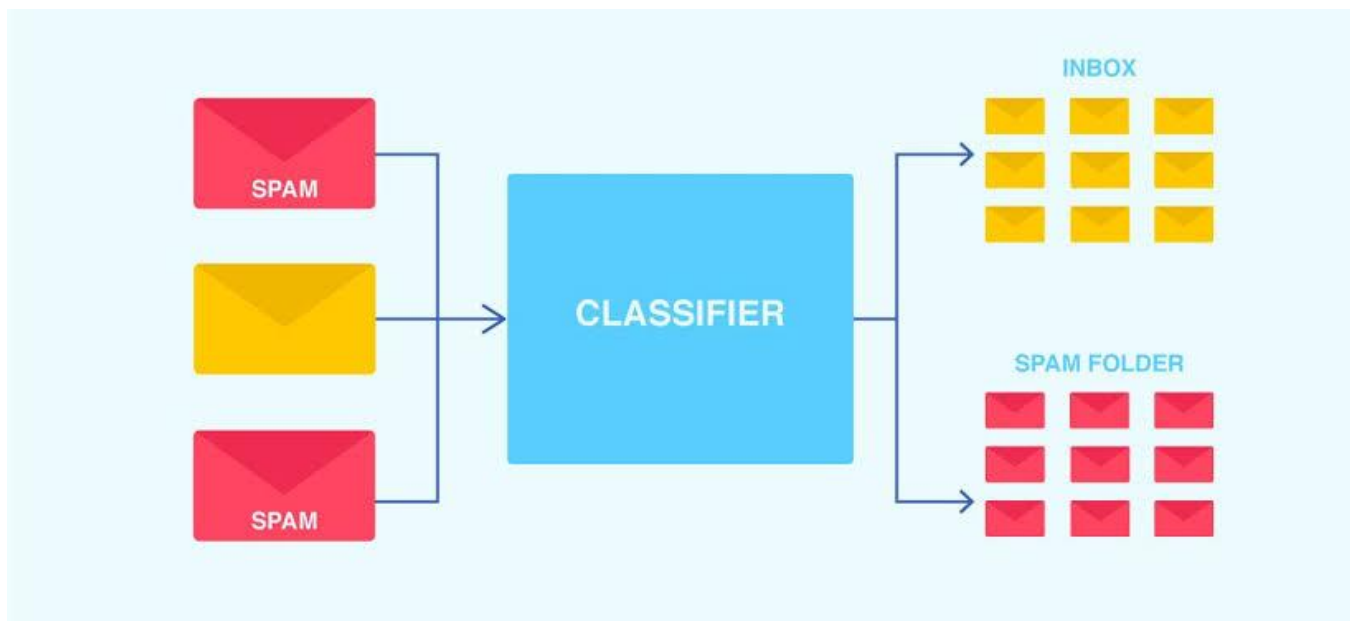
Experiment with various machine learning algorithms such as SVM, Random forests to optimize performance.

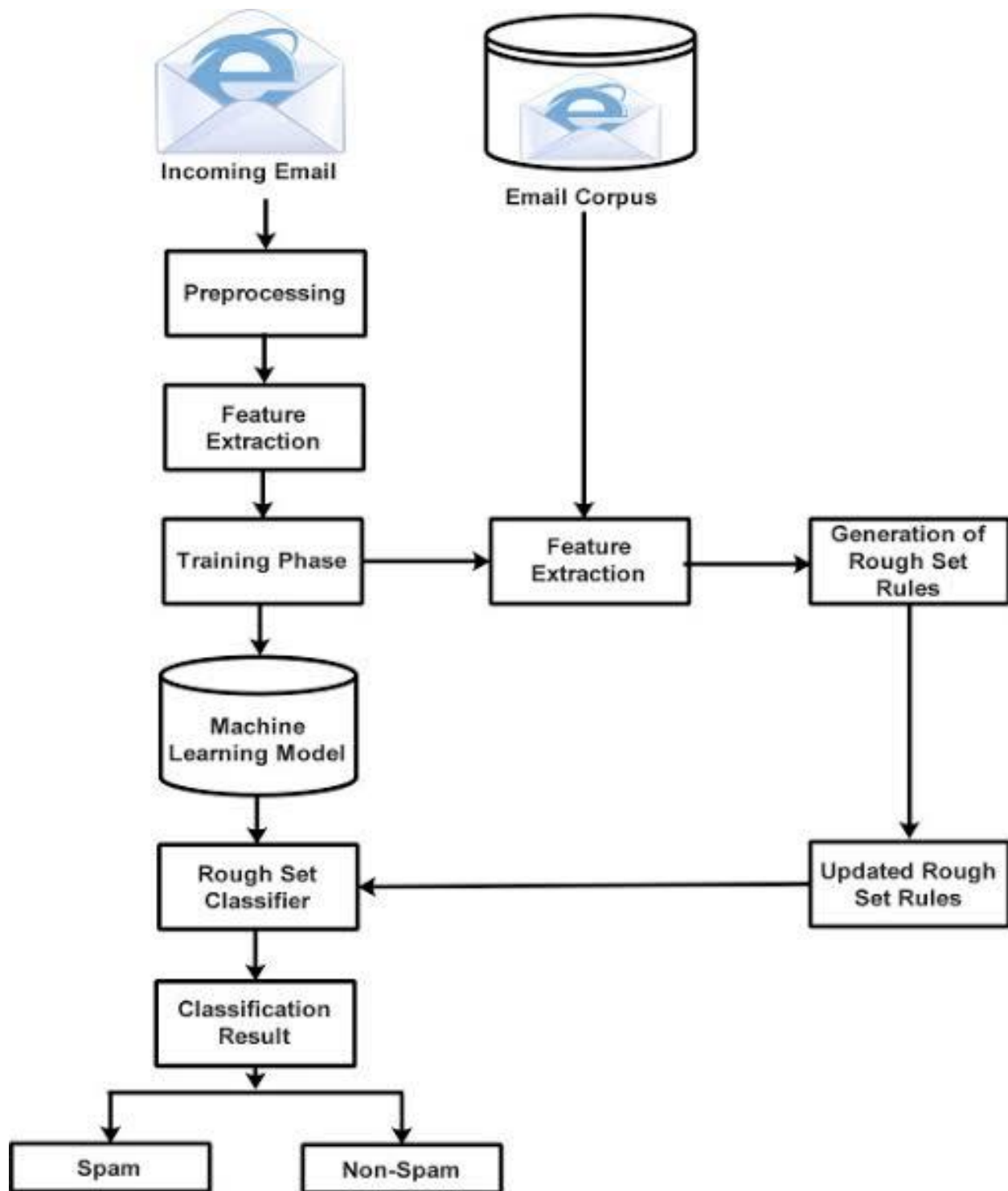
4. Ensemble Methods:

Combine multiple models using ensemble techniques like bagging, boosting or stacking to improve classification accuracy and reduce over fitting.

5. Anomaly Detection:

Implement anomaly detection techniques to identify unusual patterns in email content.







By accurately identifying and filtering spam, individuals and organizations can focus on important emails and mitigate potential risks associated with malicious content. In conclusion, email spam detection using machine learning offers a promising solution to the pervasive problem of unwanted and harmful emails.

Loading and Pre-processing the Dataset:

Dataset Choice:

To effectively tackle the issue of spam detection in SMS messages, we've opted to leverage a dataset provided by Kaggle. This dataset contains a substantial collection of SMS messages, each meticulously labelled as either 'spam' or 'non-spam' (commonly referred to as 'ham'). Choosing the right dataset is a critical step as it directly influences the quality and performance of our spam classifier.

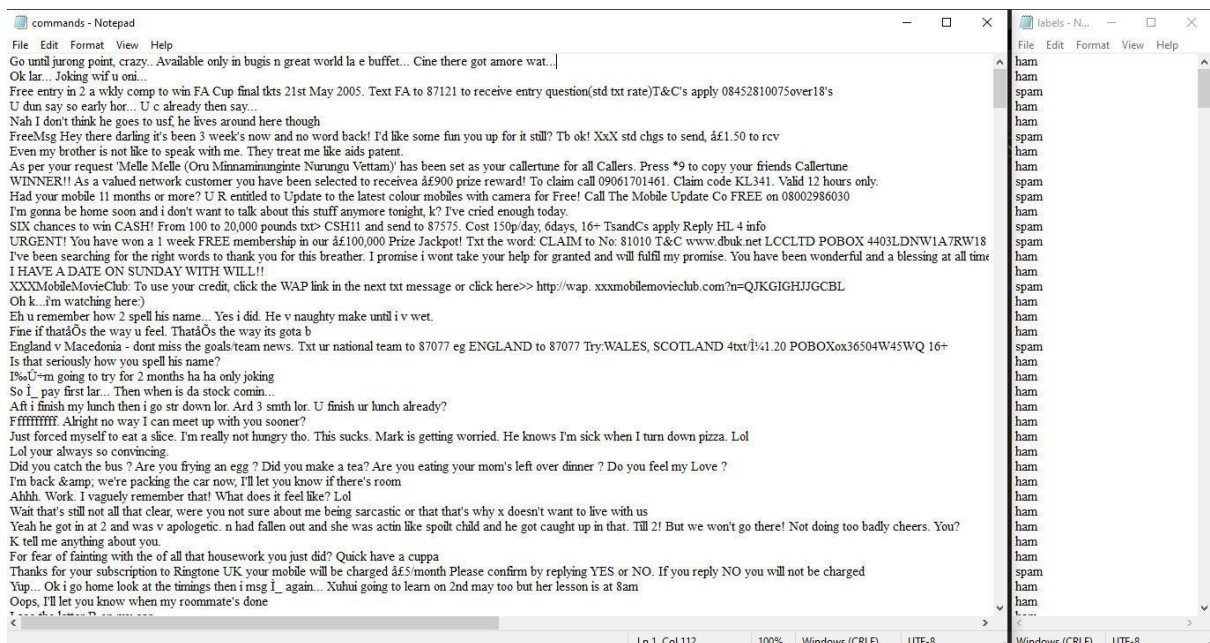
Importance of Data:

The dataset is the lifeblood of our project. It's the raw material from which our spam classifier will learn to distinguish between unwanted spam messages and legitimate ones. By analysing patterns and characteristics in this dataset, our model will become capable of making predictions in real-world scenarios.

Loading the Data:

The first part of this phase involves loading the dataset into our project environment. This step ensures that we have access to the SMS messages and their corresponding labels, enabling us to work with the data effectively. We may use programming languages like Python and libraries like Pandas for this task.

Pre-processing the Data:



Once the dataset is loaded, we need to pre-process it. This involves a series of tasks such as:

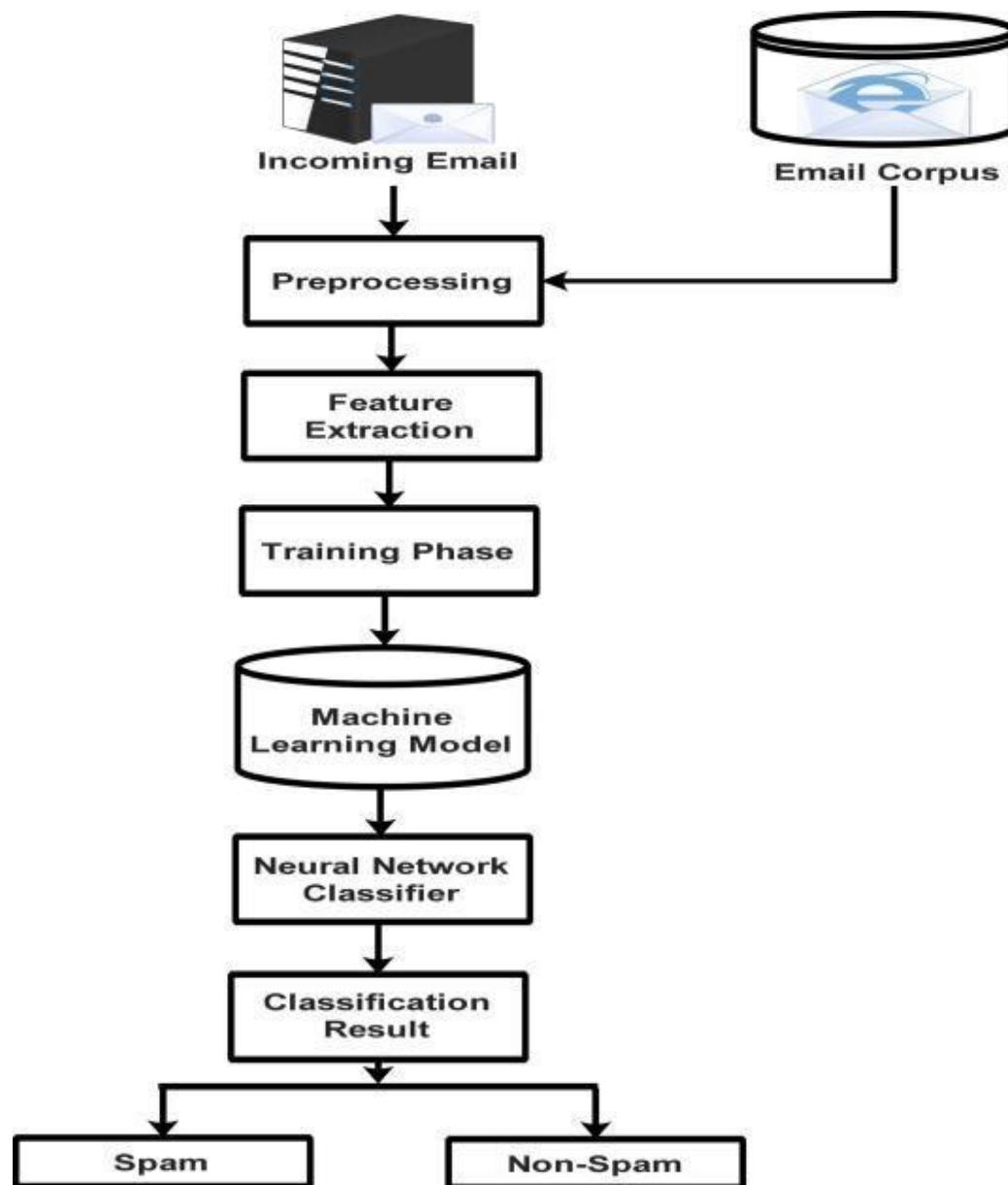
1. **Data Cleaning:** Removing any inconsistencies, missing values, or irrelevant information that could hinder our analysis.
2. **Text Tokenization:** Breaking down the SMS messages into individual words or tokens. This is a fundamental step for natural language processing.
3. **Text Normalization:** Ensuring that text is consistent by converting it to lowercase, removing punctuation, and handling issues like stemming or lemmatization.
4. **Feature Extraction:** Converting the text data into numerical features that machine learning algorithms can work with. Common techniques include TFIDF (Term Frequency-Inverse Document Frequency) or word embeddings.

Proper data pre-processing is pivotal because it sets the stage for the subsequent phases of our project. Clean, organized, and structured data allows us to build effective machine learning models. The quality of our spam classifier heavily depends on how well we manage the data in this phase.

By the end of this phase, we will have a well-organized dataset ready for analysis, model development, and evaluation. This foundational work is essential for harnessing the power of machine learning and natural language processing to distinguish spam from legitimate SMS messages accurately. Our success in building an effective spam classifier largely hinges on the precision and care with which we handle the data in this phase.

Advantages of pre-processing the data:

1. Improved model performance.
2. Noise reduction.
3. Handling the missing data.
4. Data standardization.
5. Better handling of categorical data.
6. Enhanced robustness.



PYTHON CODE:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score import joblib
print("All the librarices imported successfully\n")
```

```
labels =open("labels.txt","r") labels =
labels.read() labels =
list(labels.split("\n")) commands =
open("commands.txt","r") commands =
commands.read() commands =
list(commands.split("\n")) print("Dataset
read successfully\n")
```

```
vectorizer = TfidfVectorizer() X =
vectorizer.fit_transform(commands)
```

```
X_train, X_val, y_train, y_val = train_test_split(X, labels,
test_size=0.2, random_state=42) model = SVC(kernel='linear')
print("Training is started. . . . .\n")
model.fit(X_train, y_train) print("Training is
```

```
completed successfully\n") predictions =  
model.predict(X_val) accuracy =  
accuracy_score(y_val, predictions)  
print("Accuracy score is:", accuracy, "\n")
```

```
joblib.dump(model, "model.pkl") print("Model  
saved successfully!")
```

```
IDLE Shell 3.10.1  
File Edit Shell Debug Options Window Help  
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>> == RESTART: E:\others\i am chief minister project\pandi1\spam_classifier.py ==  
All the libraries imported successfully  
  
Dataset read successfully  
  
Training is started. . . . .  
  
Training is completed successfully  
  
Accuracy score is: 0.7428571428571429  
  
Model saved successfully!  
>>>
```

1. Importing Libraries:

The code starts by importing necessary libraries, including:

- `TfidfVectorizer` from `sklearn.feature_extraction.text` to convert text data into TF-IDF (Term Frequency-Inverse Document Frequency) features.

- `SVC`` (Support Vector Classification) from ``sklearn.svm`` for the SVM classifier.
- ``train_test_split`` from ``sklearn.model_selection`` to split the dataset into training and validation sets.
- ``accuracy_score`` from ``sklearn.metrics`` to calculate the accuracy of the classifier.
- ``joblib`` for saving the trained model to a file.

2. Reading Data:

- The code opens two text files: "labels.txt" and "commands.txt."
- It reads the content of these files into two separate variables
- ``labels``: This variable will store the labels or categories for each command.
- ``commands``: This variable will store the textual commands or data.

3. TF-IDF Vectorization:

- The code initializes a ``TfidfVectorizer`` as ``vectorizer``.
- It uses the ``fit_transform`` method to convert the text data in the ``commands`` variable into a TF-IDF representation. This process creates a numerical feature matrix ``X`` where each row corresponds to a command, and each column corresponds to a TF-IDF-weighted term in the command.

4. Data Splitting:

The code uses the ``train_test_split`` function to split the data into training and validation sets. The splitting is done in an 80-20 ratio, with 80% of the data used for training (``X_train`` and ``y_train``) and 20% for validation (``X_val`` and ``y_val``).

5. Model Training:

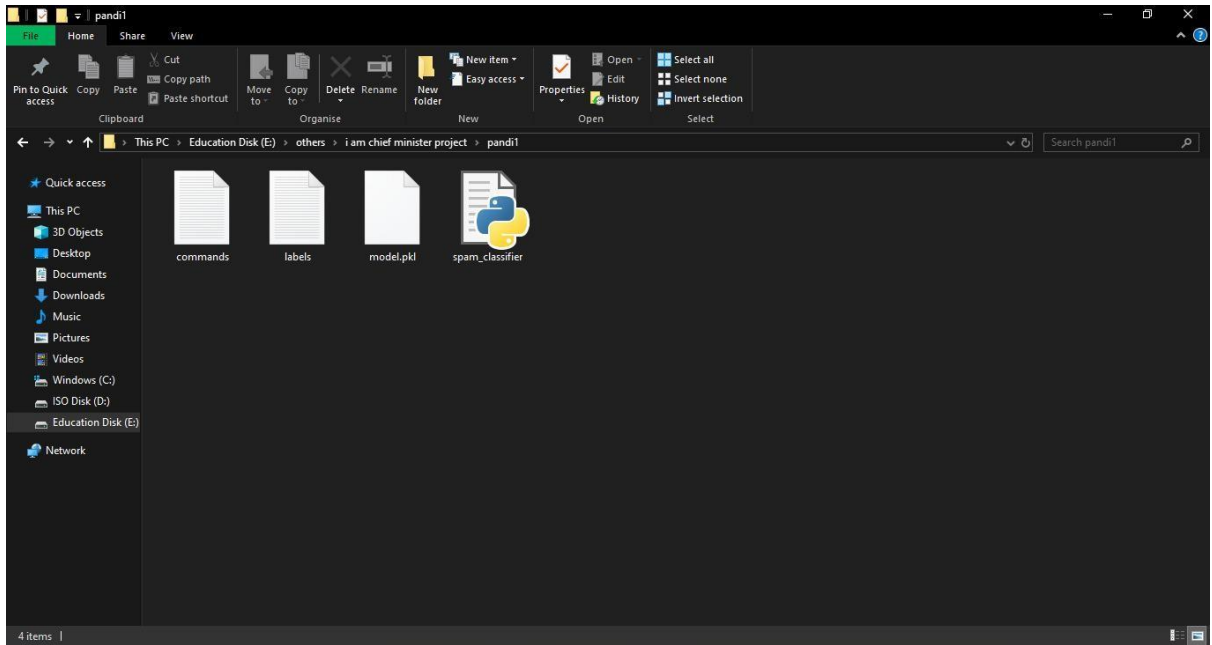
- An SVM classifier is initialized with a linear kernel and assigned to the ``model`` variable.
- The code then fits the model to the training data using the ``fit`` method. This is where the actual training of the classifier takes place.

6. Model Evaluation:

- The code makes predictions on the validation data using the trained model and stores the predictions in the ``predictions`` variable.
- It calculates the accuracy of the model's predictions using the ``accuracy_score`` function and prints the accuracy score to the console.

7. Model Saving:

The trained SVM model is saved to a file named "model.pkl" using the ``joblib.dump`` function. This allows you to later load and use the trained model for making predictions on new data without having to retrain it.



In conclusion, a spam classifier is a valuable tool for filtering unwanted and potentially harmful messages from users' inboxes. It involves the development of a machine learning model trained on labelled data, and its performance can be continually improved through rigorous evaluation, feature engineering, and user feedback. Successful spam classification helps protect users from unwanted and potentially harmful content, enhancing their digital experience.

The primary aim of a spam classifier is to enhance the user experience and protect users from unwanted, irrelevant, or potentially harmful messages by efficiently categorizing and filtering messages into the following categories:

Spam: Messages that are clearly unsolicited and have the potential to be fraudulent, contain phishing attempts, or deliver malware. The aim is to keep these out of the user's inbox.

Ham : Messages that are not spam and are important or desired by the user. The aim is to ensure that these messages are delivered to the user's inbox.

PROGRAM:

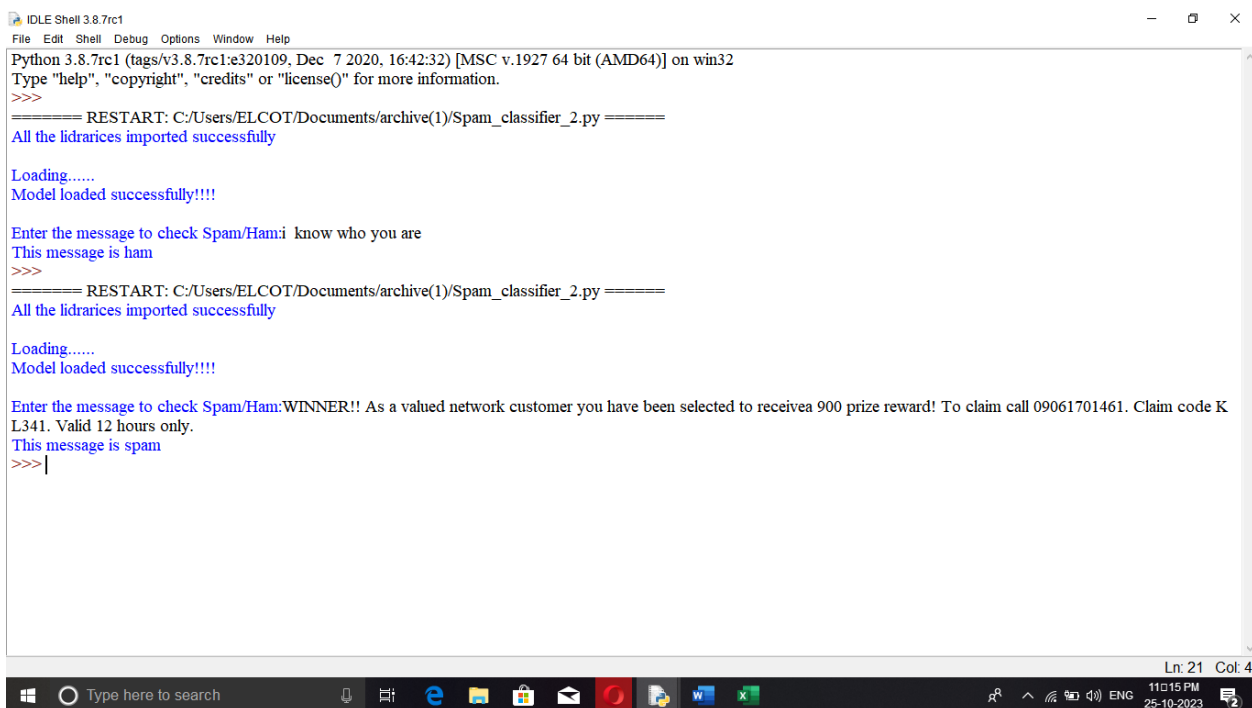
```
import joblib
print("All the libraries imported successfully\n")

print("Loading.....")
loaded_model = joblib.load("model.pkl")
vectorizer = joblib.load("vectorizer.pkl")
print("Model loaded successfully!!!!\n")

message = []
mess = input("Enter the message to check Spam/Ham:")
message.append(mess)
```

```
VEC_MESSAGE = vectorizer.transform(message)
predictions = loaded_model.predict(VEC_MESSAGE)
print("This message is "+predictions[0])
```

OUTPUT:



```
IDLE Shell 3.8.7rc1
File Edit Shell Debug Options Window Help
Python 3.8.7rc1 (tags/v3.8.7rc1:e320109, Dec 7 2020, 16:42:32) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/ELCOT/Documents/archive(1)/Spam_classifier_2.py =====
All the libraries imported successfully

Loading.....
Model loaded successfully!!!!

Enter the message to check Spam/Ham: i know who you are
This message is ham
>>>
===== RESTART: C:/Users/ELCOT/Documents/archive(1)/Spam_classifier_2.py =====
All the libraries imported successfully

Loading.....
Model loaded successfully!!!!

Enter the message to check Spam/Ham: WINNER!! As a valued network customer you have been selected to receive a 900 prize reward! To claim call 09061701461. Claim code K
L341. Valid 12 hours only.
This message is spam
>>> |
```

EXPLANATION:

- The code starts by importing the joblib library.
- The code then loads the model and vectorizer libraries into memory.

- The next line of code creates a list called message that will hold all the messages to be checked for spam or ham.
- Next, it asks for input from the user on what they want to check as spam/ham using an input function called mess.
- Then, it uses a transform function in the vectorizer library to transform each message into a numerical value between 0 and 1 (inclusive).
- This number is fed into a predict function in loaded_model which returns predictions of how likely each word is to be classified as either spam or ham.
- Finally, this prediction is printed out with some formatting so that it can easily be read by humans.
- The code attempts to load the model and vectorizer, transform the message into a list of predictions, and print out the first prediction.
- The code above will print “This message is Ham or Spam”.

CONCLUSION:

By accurately identifying and filtering spam, individuals and organizations can focus on important emails and mitigate potential risks associated with malicious content. In conclusion, email spam detection using machine learning offers a promising solution to the pervasive problem of unwanted and harmful emails.