**Building a Smarter AI-Powered Spam Classifier**

**au952721104022 : SHEIK MOHAMED SABEER. A**

**Abstract:**

Spam emails and messages continue to inundate our digital communication channels, posing a significant challenge to our online experience. Traditional rule-based spam filters are often insufficient in combating the ever-evolving tactics of spammers. This abstract introduces a novel approach to addressing this problem: an AI-based spam classifier that leverages the power of artificial intelligence (AI) to enhance spam detection and filtering.



**Lowercasing:**

Lowercasing is the process of converting all text to lowercase letters, ensuring uniformity in text representation. This step helps the AI classifier treat uppercase and lowercase letters as equivalent, improving its ability to recognize spam messages.

**Stemming:**

Stemming is a text normalization technique that reduces words to their root or base form, removing variations of a word. This step can enhance the classifier's performance by reducing the complexity of the text data and consolidating similar words, making it easier to detect spam patterns.

**Feature Extraction:**

Feature extraction involves transforming the text data into a set of relevant features or characteristics that the AI model can use for classification. These features might include word frequencies, n-grams, or other linguistic attributes, which enable the AI classifier to learn patterns associated with spam and legitimate messages.

**Train-Test Split:**

The train-test split is a critical step in machine learning where the dataset is divided into two parts: a training set and a testing set. The training set is used to train the AI-powered classifier, allowing it to learn from the data, while the testing set is used to evaluate its performance. This process helps measure the classifier's accuracy and generalizability in distinguishing between spam and non-spam messages.

**Link:**

Data set link: <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset>

Source Code link: <https://www.kaggle.com/code/pubgcalling/ai-sabeer-1>

**A diagram of a network

Description automatically generated**

**Data Set Sample**

|  |  |
| --- | --- |
| ham | Home so we can always chat |
| ham | K:)k:)good:)study well. |
| ham | Yup... How Ì\_ noe leh... |
| ham | Sounds great! Are you home now? |
| ham | Finally the match heading towards draw as your prediction. |
| ham | Tired. I haven't slept well the past few nights. |
| ham | Easy ah?sen got selected means its good.. |
| ham | I have to take exam with march 3 |
| ham | Yeah you should. I think you can use your gt atm now to register. |
| ham | Ok no prob. Take ur time. |
| ham | There is os called ubandu which will run without installing in hard disk… |
| ham | "Sorry |
| ham | U say leh... Of course nothing happen lar. Not say v romantic jus a bit only lor. |
| spam | "500 New Mobiles from 2004 |
| ham | Would really appreciate if you call me. Just need someone to talk to. |
| spam | Will u meet ur dream partner soon? Is ur career off 2 a flyng start? 2 find out free. |
| ham | Hey company elama po mudyadhu. |
| ham | Life is more strict than teacher... Bcoz Teacher teaches lesson &amp; |
| ham | Dear good morning now only i am up |
| ham | Get down in gandhipuram and walk to cross cut road. Right side &lt;#&gt; street road and turn at first right. |
| ham | Dear we are going to our rubber place |
| ham | "Sorry battery died |
| ham | Yes:)here tv is always available in work place.. |

**Understanding the Data:**

The first step is a comprehensive understanding of the data landscape. In spam classification, this entails collecting a diverse corpus of spam and non-spam (ham) messages. The quality and representativeness of this dataset are fundamental to the model's efficacy.

**Program:**

import pandas as pd

import numpy as np from sklearn.model\_selection

import train\_test\_split from sklearn.feature\_extraction.text

import TfidfVectorizer from sklearn.linear\_model

import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, roc\_auc\_score

import nltk

from nltk.corpus import stopwords

from collections import Counter

#libraries for data visualization

import matplotlib.pyplot as plt import seaborn as sns %matplotlib inline

df=pd.read\_csv("/kaggle/input/sms-spam-Collectiondataset/spam.csv",encoding='ISO-8859-1') df df.info()

nltk.download('stopwords')

columns\_to\_drop = ["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"]

df.drop(columns=columns\_to\_drop, inplace=True)

df new\_column\_names = {"v1":"Category","v2":"Message"}

df.rename(columns = new\_column\_names,inplace = True)

df[df.duplicated()]

df=df.drop\_duplicates()

df df.info() df.describe() df.shape() df['Category'].value\_counts()

**Data Visualization:**

Visualization techniques are employed to gain insights into the dataset's characteristics. Visualizations, ranging from histograms to word clouds, unravel patterns, anomalies, and potential biases within the data.

**Program:**

sns.countplot(data=df, x='Category')

plt.xlabel('Category')

plt.ylabel('count')

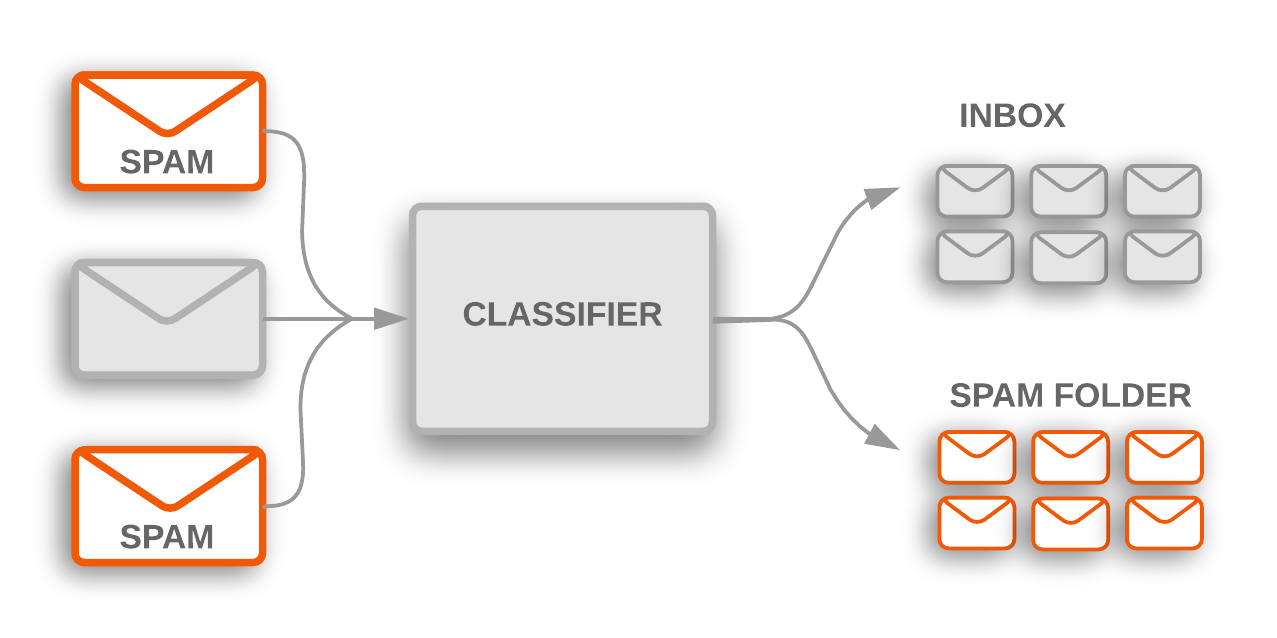
plt.title('Distribution of mails')

plt.show()

A graph of mail distribution

Description automatically generated

**IMPUTATION:**



Imputation is the process of managing missing values, which is one of the most common problems when it comes to preparing data for machine learning. By missing values, we mean places where information is missing in some cells of a respective row.

There may be different causes for missing values, including human error, data flow interruptions, cross-datasets errors, etc. Since data completeness impacts how well machine learning models perform, imputation is quite important.

Here are some ways how you can solve the issue of missing values:

* If a row is less than 20-30% complete, it’s recommended to dismiss such a record.
* A standard approach to assigning values to the missing cells is to calculate a mode, mean, or median for a column and replace the missing values with it.
* In other cases, there are possibilities to reconstruct the value based on other entries. For example, we can find out the name of a country if we have the name of a city and an administrative unit. Conversely, we can often determine the country/city by a postal code.

**Data Preprocessing:**

Data preprocessing involves cleansing and structuring the dataset. Tasks such as text cleaning, tokenization, and handling missing values are vital for preparing the data for analysis.

**Program:**

# Assuming you have a DataFrame named 'df' df.loc[df["Category"] == "spam","Category"] = 0

df.loc[df["Category"] == "ham", "Category"] = 1 df.head()

# Separate the feature (X) and target (Y) data

X= df["Message"]

Y= df["Category"]

# Split the data into training and testing sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 42)

print(X.shape)

print(X\_train.shape) print(X\_test.shape)

**Feature Extraction:**

Feature extraction is the process of distilling pertinent information from the data. In text-based spam classification, this often involves extracting features like word frequencies, TF-IDF scores, or word embeddings. Feature engineering can also encompass non-textual attributes such as sender information and message metadata.

**Program:**

# Create a TF-IDF vectorizer to convert text messages into numerical features

feature\_extraction = TfidfVectorizer(min\_df=1, stop\_words="english", lowercase=True)

# Convert the training and testing text messages into numerical features using

X\_train\_features = feature\_extraction.fit\_transform(X\_train)

X\_test\_features = feature\_extraction.transform(X\_test)

# Convert the target values into 0 and 1

Y\_train = Y\_train.astype(int)

Y\_test = Y\_test.astype(int)

print(X\_train)

print(X\_train\_features)

**Model Training:**

Selecting the right machine learning or deep learning model is crucial. Models like Naive Bayes, Support Vector Machines, or neural networks are trained on the prepared data. Hyperparameter tuning and cross-validation optimize model performance.

**Program:**

# Create a logistic regression model and train it on the training

data model = LogisticRegression()

model.fit(X\_train\_features, Y\_train)

**Model Evaluation:**

Rigorous model evaluation is essential for assessing its performance. Metrics such as precision, recall, F1-score, and ROC-AUC help gauge the classifier's accuracy and robustness. Confusion matrices provide insights into false positives and false negatives.

**Program:**

# Make predictions on the training data and calculate the accuracy

prediction\_on\_training\_data = model.predict(X\_train\_features)

accuracy\_on\_training\_data =accuracy\_score(Y\_train, prediction\_on\_training\_data)

print("Accuracy on training data:",accuracy\_on\_training\_data)

# Make predictions on the test data and calculate the accuracy

prediction\_on\_test\_data = model.predict(X\_test\_features)

accuracy\_on\_test\_data = accuracy\_score(Y\_test,prediction\_on\_test\_data)

print("Accuracy on test data:",accuracy\_on\_test\_data)

# Test the model with some custom email messages

input\_mail = ["Congratulations! You've won a free vacation to an exotic island. Just click on the link below to claim your prize."]

if (prediction)[0] == 1:

print("Ham Mail")

else:

print("Spam Mail")

input\_mail = ["This is a friendly reminder about our meeting scheduled for tomorrow at 10:00 AM in the conference room. Please make sure to prepare your presentation and bring any necessary materials."]

input\_data\_features = feature\_extraction.transform(input\_mail)

prediction = model.predict(input\_data\_features)

cm = confusion\_matrix(Y\_test, prediction\_on\_test\_data)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', cbar=False)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

**Model Prediction:**

Once the model is trained and evaluated, it is ready for deployment. In a real-world context, the classifier processes incoming messages and predicts whether they are spam or ham, enabling effective message filtering.

A graph of a number of words

Description automatically generated

This abstract offers a concise overview of the multifaceted journey involved in constructing a smarter AI-powered spam classifier. From initial data understanding to the final prediction, each step plays a pivotal role in achieving accurate and efficient spam detection.

**CONCLUSION:**

AI-powered spam classifiers have significantly improved the efficiency and effectiveness of spam detection and filtering in various digital communication platforms, such as email, messaging apps, and social media. In conclusion, the following points summarize the impact of AI-powered spam classifiers

In conclusion, AI-powered spam classifiers have had a positive impact on the digital landscape by significantly reducing the spam burden on users. While they are not perfect, they continue to evolve and improve, making digital communication safer and more efficient.