### Description

## Spam Classification Using Artificial Intelligence

**Abstract:- Spam Classification using Artificial Intelligence –**For business purposes, email is the most widely utilized mode of official communication. Despite the availability of other forms of communication, email usage continues to rise. In today’s world, automated email management is critical since the volume of emails grows by the day. More than 55 percent of all emails have been recognized as spam. This demonstrates that spammers waste email users’ time and resources while producing no meaningful results. Spammers employ sophisticated and inventive strategies to carry out their criminal actions via spam emails. As a result, it is critical to comprehend the many spam email classification tactics and mechanisms. The main focus of this paper is on spam classification using machine learning algorithms. Furthermore, this research includes a thorough examination and evaluation of research on several machine learning methodologies and email properties used in various Machine Learning approaches. Future study goals and obstacles in the subject of spam classification are also discussed, which may be valuable to future researchers.

**Objective: –**

Machine learning algorithms use statistical models to classify data. In the case of spam detection, a trained machine learning model must be able to determine whether the sequence of words found in an email is closer to those found in spam emails or safe ones.

**Introduction: –**

For the majority of internet users, email has become the most often utilized formal communication channel. In recent years, there has been a surge in email usage, which has exacerbated the problems presented by spam emails. Spam, often known as junk email, is the act of sending unsolicited mass messages to a large number of people. ‘Ham’ refers to emails that are meaningful but of a different type. Every day, the average email user receives roughly 40-50 emails. Spammers earn roughly 3.5 million dollars per year from spam, resulting in financial damages on both a personal and institutional level. As a result, consumers devote a large amount of their working time to these emails. Spam is said to account for more than half of all email server traffic, sending out a vast volume of undesired and uninvited bulk emails.

They squander user resources on useless output, lowering productivity. Spammers use spam for marketing goals to spread malicious criminal acts such as identity theft, financial disruptions, stealing sensitive information, and reputational damage.

**The existing model of the system: –**

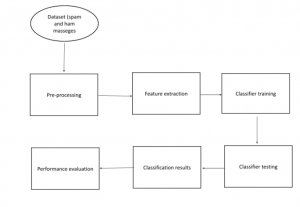
Spam refers to the term, which is related to undesired content with low-quality information, Spam referred to the major drawback of mobile business. When comes to spam detection in the campus network they did the analysis using Incremental Learning. For Collecting Spam detection on web pages. Moreover Sending out a Spam message was also analyzed. Data Collection was done privately by a limited company. From the data Collection. There also anti-spam filter system was evolved. Many parallel and distributed computing system has also processed this spam system. Machine learning algorithm provides accurate result. Text Mining analysis done separates ham and spam separately

**Proposed model of the system: –**

As we look at spam detection systems that use Machine Learning (ML) techniques, it’s vital to take a look at the history of ML in the field as well as the many methods that are now used to identify spam. Researchers have discovered that the content of spam emails, as well as their operational procedures, evolve with time. As a result, the tactics that are currently effective may become obsolete in the near future. The conceptual drift [8] is a term used to describe this occurrence. Machine Learning is an engineering approach that allows computational instruments to behave without being explicitly programmed. Because of the ML system’s ability to evolve, limiting concept drift, this strategy is a significant help in detecting and combating spam.

In the next section, we’ll go through a variety of machine learning techniques, approaches, and algorithms, as well as the benefits of each, using Supervised, Unsupervised, and Semi-Supervised Machine Learning algorithms Approaches.

**System Architecture: –**

Spam classification using Artificial Intelligence

**System Requirements: –**

**Hardware**

* OS – Windows 7, 8, and 10 (32 and 64 bit)
* RAM – 4GB

**Software:**

* Python
* Anaconda navigator
* Python built-in module

**Conclusion: –**

Following a thorough examination of the chosen study, Several study findings and observations have been identified as a result of our studies. These were previously discussed in detail.

portions that are well-explained In this section, we’ll talk about concentrating more on the major findings and conclusions of the research Supervised machine learning has a high acceptance rate. Throughout the review, the approach can be noticed. This strategy is effective. is employed primarily because it produces more accurate findings. With less fluctuation, this strategy has a high level of consistency. Aside from that, we’ve discovered that certain algorithms work better than others. When compared to other techniques, such as Nave Based and SVM, there is a strong demand for them. Machine Learning  Algorithms that aren’t as well-known. The employed multi-algorithm. n order to achieve a better result, systems are increasingly commonly used. rather than a single algorithm

### **Design buliding smarter AI-powered spam classifier**

ChatGPT

Designing a smarter AI-powered spam classifier involves a combination of data, algorithms, and continuous improvement. Here's a high-level overview of the process:

1. **Data Collection**:
   * Gather a large and diverse dataset of emails, messages, or content, including both spam and non-spam examples.
   * Annotate and label the data to distinguish between spam and non-spam accurately.
2. **Data Preprocessing**:
   * Clean and preprocess the data by removing irrelevant information, formatting, and normalizing text.
   * Feature extraction: Convert text data into numerical features, such as TF-IDF vectors, word embeddings, or other representations.
3. **Selecting Algorithms**:
   * Consider various machine learning and deep learning algorithms, such as Naive Bayes, Support Vector Machines, Random Forest, or deep neural networks (e.g., LSTM, CNN).
   * Choose the algorithm that suits your dataset and problem best.
4. **Training the Model**:
   * Split your dataset into training, validation, and test sets.
   * Train the chosen model using the training data while fine-tuning hyperparameters.
   * Monitor metrics like precision, recall, and F1-score to optimize model performance.
5. **Feature Engineering**:
   * Continuously analyze the performance and consider adding new features or improving existing ones.
6. **Regular Updates**:
   * Keep the dataset and model updated with new spam patterns and evolving techniques used by spammers.
7. **Ensemble Methods**:
   * Combine multiple models (e.g., stacking or bagging) to improve overall classification accuracy.
8. **Threshold Tuning**:
   * Adjust the classification threshold to balance between false positives and false negatives, depending on your application's requirements.
9. **User Feedback Loop**:
   * Implement a feedback mechanism where users can report false positives/negatives, and use this feedback to retrain and improve the model.
10. **Incorporate AI Explainability**:
    * Ensure that your model's decisions are interpretable, and you can explain why a particular email was classified as spam or not.
11. **Scalability and Speed**:
    * Optimize the model for real-time processing, especially if you are dealing with a high volume of messages.
12. **Deployment**:
    * Deploy the AI-powered spam classifier as an API, service, or integrated component within email platforms, messaging apps, or wherever it's needed.
13. **Monitoring and Maintenance**:
    * Continuously monitor the classifier's performance and retrain it periodically with fresh data.
    * Keep an eye on false positives and false negatives, and make adjustments as necessary.
14. **Security**:
    * Ensure that your AI system is secure to prevent attacks, such as adversarial attacks or model poisoning.
15. **Legal and Ethical Considerations**:
    * Comply with privacy laws and ethical guidelines for handling user data and content.

## **PYTHON CODE:**

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import classification\_report

# Step 1: Load and preprocess your dataset

# Assuming you have a CSV file with 'text' and 'label' columns

data = pd.read\_csv('spam\_dataset.csv')

# Step 2: Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['text'], data['label'], test\_size=0.2, random\_state=42)

# Step 3: Vectorize the text data using TF-IDF

tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) # You can adjust the number of features

X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

# Step 4: Train a classification model (e.g., Naive Bayes)

spam\_classifier = MultinomialNB()

spam\_classifier.fit(X\_train\_tfidf, y\_train)

# Step 5: Evaluate the model

y\_pred = spam\_classifier.predict(X\_test\_tfidf)

print(classification\_report(y\_test, y\_pred))

# Step 6: Fine-tune the model, adjust hyperparameters, and iterate for better performance

# Step 7: Deploy the model in your application

# You can use joblib or another library to save and load your trained model for future use.

## **BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER**

**INTRODUCTION**:

Artificial Intelligence (AI) and Machine Learning (ML) are becoming the new tool for developers to create a more efficient and life-changing models brining an intelligence into machines to perform various tasks into business operations and household without help of humans.

And to develop the AI and ML model, a precise training data is required that help algorithms to understand the certain patterns or series of outcomes comes to a given question. And training data can consist texts, images or videos which are mainly labeled to make it recognizable to computer vision and understandable to machines.

**TRAINING DATA:**

Training data is basically a type of data used for training a new application, model or system through various methods depending on the project’s feasibility and requirements. And training data for AI or ML is slightly different, as they are labeled or annotated with certain techniques to make it recognizable to computer that helps machines to understand the objects.

**Types of Training Data for Machine**

machine learning training data is the key factor to make the machines recognize the objects or certain patterns and make the right prediction when used in real-life. Basically, there are three types of training data used in machine learning model development and each data has its own importance and role in building a ML model

majority of training data contains the pair of input gathered from the various resources and then organized and annotated with certain techniques with accuracy

**STRUCTURED DATA:**

Structured data is data that has a standardized format for efficient access by software and humans alike. It is typically tabular with rows and columns that clearly define data attributes. Computers can effectively process structured data for insights due to its quantitative nature.

**Structured data examples**

Here are examples of structured data systems:

• Excel files

• SQL databases

• Point-of-sale data

• Web form results

• Search engine optimization (SEO) tags

• Product directories

• Inventory control

• Reservation systems

**UNSTRUCTURED DATA:**

data is information that is not arranged according to a preset data model or schema, and therefore cannot be stored in a traditional relational database or RDBMS. Text and multimedia are two common types of unstructured content.

**Examples of unstructured**

dataUnstructured data can be created by people or generated by machines.Here are some examples of the human-generated variety:

• Email: Email message fields are unstructured and cannot be parsed by traditional analytics tools. That said, email metadata affords it some structure, and explains why email is sometimes considered semi-structured data.

• Text files: This category includes word processing documents, spreadsheets, presentations, email, and log files.

• Social media and websites: data from social networks like Twitter, LinkedIn, and Facebook, and websites such as Instagram, photo-sharing sites, and YouTube.

• Mobile and communications data: For this category, look no further than text messages, phone recordings, collaboration software, chat, and instant messaging.

• Media: This data includes digital photos, audio, and video files.

**SEMI STRUCTURED DATA:**

Semi-structured datais a form of structured data that does not obey the tabular structure of data models associated with relational databases or other forms of datatables, but nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, it is also known as self-describing structure.

**Examples of structured data**

•Dates and times.

•Cell phone numbers.

•Social security numbers.

•Banking/transaction information.

•Customer names, postal addresses, and email addresses.

•Product prices.

•Serial numbers.

### **Begin Building smarter AI power spam loading and preprocessing the dataset**

**In[1]:**

**import numpy as** **np**

*# linear algebra*

**import pandas as** **pd**

*# data processing, CSV file I/O (e.g. pd.read\_csv)*

**from nltk.corpus import** **stopwords import** **nltk**

**nltk.download('stopwords')**

**from sklearn.pipeline import** **Pipeline**

**from sklearn.naive\_bayes import** **BernoulliNB , MultinomialNB , GaussianNB**

**from sklearn.metrics import** **accuracy\_score**

**import** **os**

**for** **dirname, \_, filenames in os.walk('/kaggle/input'):**

**for** **filename in filenames:**

**print(os.path.join(dirname, filename))**

[nltk\_data] Downloading package stopwords to /usr/share/nltk\_data...

[nltk\_data] Package stopwords is already up-to-date!

/kaggle/input/sms-spam-collection-dataset/spam.csv

#### **Understand the spam collection data !**

**In [2]:**

**filepath = '/kaggle/input/sms-spam-collection-dataset/spam.csv'**

**data\_import = pd.read\_csv(filepath , encoding = 'ISO-8859-1')**

**data\_import.head()**

**Out[2]**:

|  | v1 | v2 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 |
| --- | --- | --- | --- | --- | --- |
| 0 | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| 1 | ham | Ok lar... Joking wif u oni... | NaN | NaN | NaN |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... | NaN | NaN | NaN |
| 3 | ham | U dun say so early hor... U c already then say... | NaN | NaN | NaN |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... | NaN | NaN | NaN |

### **Preprocessing !**

### **In [3]:**

**df = data\_import.drop(['Unnamed: 2' , 'Unnamed: 3' , 'Unnamed: 4'] , axis1)**

**df.head()**

**Out[3]:**

|  | v1 | v2 |
| --- | --- | --- |
| 0 | ham | Go until jurong point, crazy.. Available only ... |
| 1 | ham | Ok lar... Joking wif u oni... |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | ham | U dun say so early hor... U c already then say... |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... |

**In [4]:**

*### Removing stopwords from the feature column.*

**sw = stopwords.words('english')**

**def** **stopword(text) :**

**txt = [word.lower()**

**for word in text.split()**

**if word.lower() not in sw]**

**return** **txt**

**df['v2'] = df['v2'].apply(stopword)**

**df.head()**

**Out[4]:**

|  | v1 | v2 |
| --- | --- | --- |
| 0 | ham | [go, jurong, point,, crazy.., available, bugis... |
| 1 | ham | [ok, lar..., joking, wif, u, oni...] |
| 2 | spam | [free, entry, 2, wkly, comp, win, fa, cup, fin... |
| 3 | ham | [u, dun, say, early, hor..., u, c, already, sa... |
| 4 | ham | [nah, think, goes, usf,, lives, around, though] |

### **Stemming**

### **In [5]:**

**from nltk.stem.snowball**

**import** **SnowballStemmer**

**ss = SnowballStemmer("english")**

**def** **stemming(text) :**

**text = [ss.stem(word)**

**for word in text**

**if word.split()]**

**return "".****join(text)**

**df['v2'] = df['v2'].apply(stemming)**

**In [6]:**

**df.head()**

**Out[6]:**

|  | v1 | v2 |
| --- | --- | --- |
| 0 | ham | gojurongpoint,crazy..availbugingreatworldlaebu... |
| 1 | ham | oklar...jokewifuoni... |
| 2 | spam | freeentri2wklicompwinfacupfinaltkts21stmay2005... |
| 3 | ham | udunsayearlihor...ucalreadisay... |
| 4 | ham | nahthinkgoeusf,livearoundthough |

**In [7]:**

### TF-IDF { Term Frequency , Inverse Document Frequency }

**from sklearn.feature\_extraction.text**

**import** **TfidfVectorizer**

**tfid\_vect = TfidfVectorizer()**

# Extract the tfid representation matrix of the test data.

tfid\_matrix = tfid\_vect.fit\_transform(df['v2'])

**print(f"Type :{type(tfid\_matrix)} , Matrix at 0 : {tfid\_matrix[0]} , Shape : {tfid\_matrix.shape}")**

Type :<class 'scipy.sparse.\_csr.csr\_matrix'> , Matrix at 0 : (0, 1827) 0.5056391989470028

(0, 1030) 0.5056391989470028

(0, 2166) 0.48268727087494234

(0, 3635) 0.5056391989470028 , Shape : (5572, 12124)

**In [8]:**

# Collect sparse matrix into dense

**array = tfid\_matrix.todense()**

**In [9]:**

**df1 = pd.DataFrame(array)**

**df1[df1[10] != 0].head()**

**Out[9]:**

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 12114 | 12115 | 12116 | 12117 | 12118 | 12119 | 12120 | 12121 | 12122 | 12123 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5285 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

2 rows × 12124 columns

**In [10]:**

**df1['v1'] = df['v1']**

**In [11]:**

**df1.head()**

**Out[11]:**

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **...** | **12115** | **12116** | **12117** | **12118** | **12119** | **12120** | **12121** | **12122** | **12123** | **v1** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ham |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ham |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | spam |
| 3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ham |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ham |

5 rows × 12125 columns

**In [12]:**

**from sklearn.model\_selection**

**import** **train\_test\_split**

**features = df1.drop('v1' , axis = 1)**

**label = df1['v1']**

**x\_train , x\_test , y\_train , y\_test = train\_test\_split(features , label , test\_size = 0.3)**

**print(f"X train shape : {x\_train.shape}\nY train shape : {y\_train.shape}\nX test shape : {x\_test.shape}\nY test shape : {y\_test.shape}")**

**X train shape : (3900, 12124)**

**Y train shape : (3900,)**

**X test shape : (1672, 12124)**

**Y test shape : (1672,)**

**In [13]:**

**ber\_pipe = Pipeline(steps = [( 'ber\_model' ,** **BernoulliNB())])**

**multi\_pipe = Pipeline(steps = [('multi\_model' ,** **MultinomialNB())])**

**guass\_pipe = Pipeline(steps = [('guass\_model' ,** **GaussianNB())])**

**In [14]:**

**def** **model\_evaluation(model) :**

**model.fit(x\_train , y\_train)**

**y\_pred\_model = model.predict(x\_test)**

**acc\_score = accuracy\_score(y\_test , y\_pred\_model)**

**print(f"Accuracy Score of {model[0]} : {acc\_score}")**

**model\_evaluation(ber\_pipe)**

**model\_evaluation(multi\_pipe)**

**model\_evaluation(guass\_pipe)**

**Accuracy Score of BernoulliNB() : 0.8947368421052632**

**Accuracy Score of MultinomialNB() : 0.9204545454545454**

**Accuracy Score of GaussianNB() : 0.464114832535353**

**What is README file?**

In simple words, we can describe a README file as a guide that gives users a detailed description of a project you have worked on.

**How to Write a Good README – a Step by Step by procedure**

**1. Project's Title**

This is the name of the project. It describes the whole project in one sentence, and helps people understand what the main goal and aim of the project is.

**2. Project Description**

This is an important component of your project that many new developers often overlook.

Your description is an extremely important aspect of your project. A well-crafted description allows you to show off your work to other developers as well as potential employers.

The quality of a README description often differentiates a good project from a bad project. A good one takes advantage of the opportunity to explain and showcase:

What your application does,

Why you used the technologies you used,

Some of the challenges you faced and features you hope to implement in the future.

**3. Table of Contents (Optional)**

If your README is very long, you might want to add a table of contents to make it easy for users to navigate to different sections easily. It will make it easier for readers to move around the project with ease.

**4. How to Install and Run the Project**

If you are working on a project that a user needs to install or run locally in a machine like a "POS", you should include the steps required to install your project and also the required dependencies if any.

Provide a step-by-step description of how to get the development environment set and running.

**5. How to Use the Project**

Provide instructions and examples so users/contributors can use the project. This will make it easy for them in case they encounter a problem – they will always have a place to reference what is expected.

You can also make use of visual aids by including materials like screenshots to show examples of the running project and also the structure and design principles used in your project.

Also if your project will require authentication like passwords or usernames, this is a good section to include the credentials.

**6. Include Credits**

If you worked on the project as a team or an organization, list your collaborators/team members. You should also include links to their GitHub profiles and social media too.

Also, if you followed tutorials or referenced a certain material that might help the user to build that particular project, include links to those here as well.

This is just a way to show your appreciation and also to help others get a first hand copy of the project.

**7. Add a License**

For most README files, this is usually considered the last part. It lets other developers know what they can and cannot do with your project.

We have different types of licenses depending on the kind of project you are working on. Depending on the one you will choose it will determine the contributions your project gets.

The most common one is the GPL License which allows other to make modification to your code and use it for commercial purposes. If you need help choosing a license, use check out this link: https://choosealicense.com/

Up to this point what we have covered are the minimum requirements for a good README. But you might also want to consider adding the following sections to make it more eye catching and interactive.

Additional README Sections

**8. Badges**

Badges aren't necessary, but using them is a simple way of letting other developers know that you know what you're doing.

Having this section can also be helpful to help link to important tools and also show some simple stats about your project like the number of forks, contributors, open issues etc...

Below is a screenshot from one of my projects that shows how you can make use of badges:

badges

The good thing about this section is that it automatically updates it self.

Don't know where to get them from? Check out the badges hosted by shields.io. They have a ton of badges to help you get started. You may not understand what they all represent now, but you will in time.

**9. How to Contribute to the Project**

This mostly will be useful if you are developing an open-source project that you will need other developers to contribute to. You will want to add guidelines to let them know how they can contribute to your project.

Also it is important to make sure that the licence you choose for an open-source projects is correct to avoid future conflicts. And adding contribution guidelines will play a big role.

Some of the most common guidelines include the Contributor Covenant and the Contributing guide. Thes docs will give you the help you need when setting rules for your project.

**10. Include Tests**

Go the extra mile and write tests for your application. Then provide code examples and how to run them.

This will help show that you are certain and confident that your project will work without any challenges, which will give other people confidence in it, too