
MANIPAL
ACADEMY of HIGHER EDUCATION
BENGALURU CAMPUS
(Institution of Eminence Deemed to be University)

Project Titled

# Evaluating the Effectiveness of Machine Learning Methods for Spam Email Classification

By

**Gattupalli Sudhamsh**

Registration No: 215890068,

**Ch.Sai Nikhil**

Registration No: 215890074,

**E.Krishna Vamshi**

Registration No: 215890082

Department of Computer Science and EngineeringManipal Institute of Technology

Bengaluru

Under the supervision of

**Dr. Raguru Jaya Krishna**

Assistant Professor, Senior Scale

Department of Computer Science and Engineering
Manipal Institute of Technology
Bengaluru

**Manipal Institute of Technology**

**Bengaluru Campus-560064, Karnataka, India.**

# Title: Evaluating the Effectiveness of Machine Learning Methods for Spam Email Classification

## 1. Introduction

### 1.1 Background

The widespread use of email and other digital communication tools has completely changed how people communicate and do business. But this technological progress has also resulted in a deluge of unsolicited, unwanted emails, or spam. Spam is becoming more and more of a threat, taking up time and resources that both individuals and businesses need to spend. This deluge not only impedes the smooth operation of communication channels but also presents possible security weaknesses. As a result, reliable systems that can sort through this flood of emails and distinguish between spam and legitimate content are necessary. As a result, using Machine Learning (ML) algorithms to categorize and filter these emails has become essential for maintaining a more productive, safe, and efficient email environment.

### 1.2 Problem Statement

This project's main goal is to address the widespread problem of spam emails, which has a negative influence on system security and user productivity. Time and resources are currently wasted because there isn't an effective and trustworthy system in place to distinguish between emails that are legitimate and spam. Moreover, it's critical to distinguish and filter out spam emails because they frequently contain malicious links or attachments that jeopardize system security. In addition to posing serious security risks, a weak classification system makes it difficult for users and organizations to communicate effectively. Therefore, the main goal of this project is to create an advanced machine learning (ML) classification system that can precisely identify and separate spam emails in order to reduce the risks and difficulties related to this widespread problem.

### 1.3 Objectives

- Develop a machine learning model capable of classifying emails as spam or not spam with high accuracy (95% or above).
- Minimize the number of false positives, ensuring that legitimate emails are not mistakenly classified as spam.
- Enhance the system's robustness to handle various types of spam emails and adapt to evolving spam techniques.
- A fully functional spam email classification system implemented using machine learning algorithms.
- A comprehensive documentation of the system's design, implementation, and evaluation.
- A user-friendly interface for interacting with the spam email classification system.

### 1.4 Scope

This project's scope includes the creation and verification of an ML-based spam email classification system. The process entails scrutinizing email content, headers, and related features in order to construct a sturdy classification system. Though the system strives for high accuracy, it is important to understand that it may not be able to handle highly sophisticated or previously unseen spam tactics without hitches. Furthermore, neither the identification of spam nor the prevention of spam from

being generated is addressed by this project. The main objective is still to create a classification system based on machine learning that is both flexible and effective in identifying spam from legitimate emails. This will help users receive much better email experiences overall.

### 1.5 Individual Contribution

Data Collection & Data Preprocessing: Sai Nikhil,

Feature Extraction & Stopword Removal: Krishna Vamsi,

Model Implementation, Analysis, Interpretation & Frontend Development of the Model: Sudhamsh.

## 2. Literature Review

### 2.1 NLP Concepts

Creating a spam email classification ML model involves various Natural Language Processing (NLP) concepts. Here are some of the key elements:

1. Text Preprocessing: This involves several steps like tokenization (breaking text into smaller units like words or characters), removing punctuation, and transforming text to lowercase to normalize the data.

2. Feature Extraction: Converting text into numerical features that machine learning models can understand is crucial. Techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (such as Word2Vec or GloVe) help represent text in a numerical format.

3. N-Grams: Analyzing sequences of words (bigrams, trigrams, etc.) can capture context and improve the model's understanding of language nuances, aiding in the identification of spam patterns.

4. Stopword Removal: Common words like "and," "the," etc., may not carry significant meaning in distinguishing spam. Removing these stopwords helps focus on more informative terms.

5. Model Training: Leveraging algorithms like Naive Bayes, Support Vector Machines (SVM), or more advanced techniques like neural networks (using LSTM or CNN architectures) to train the model. These algorithms learn to differentiate between spam and non-spam based on the extracted features.

6. Evaluation Metrics: Metrics like accuracy, precision, recall, and F1-score are used to evaluate the model's performance in distinguishing between spam and legitimate emails.

7. Continual Learning: To adapt to new spamming techniques, the model might employ continual learning approaches, retraining the model with new data periodically.

By combining these NLP concepts, the ML model learns to distinguish between spam and legitimate emails, continually improving its accuracy and efficacy in identifying and filtering out unwanted messages.

### 2.2 Related Work

The literature on spam email detection and filtering has been extensively explored, revealing the persistent challenges and dynamic nature of the spam ecosystem. A research paper emphasizes the evolving landscape of spam, incorporating scams, malware, and phishing, despite reported high-performance machine learning-based filters [1]. One of it addresses the detrimental impact of spam on computing resources and evaluates the effectiveness of machine learning techniques, including Support Vector Machine, ANN, J48, and Naïve Bayes, for email spam classification [2]. In another paper, a systematic review of machine learning-based email spam filtering approaches is presented, analyzing

efficiency, research trends, and the applications within leading internet service providers [3]. Accordingly next research paper introduces a model utilizing Bayes' theorem and Naive Bayes' Classifier for spam identification, highlighting the economic viability of spam as a form of advertising [4]. Lastly, one more reference paper focuses on the security threats posed by spam emails, employing machine learning and deep learning models such as Random Forest and Logistic Regression to achieve high accuracy scores [5]. Collectively, these studies underscore the multifaceted challenges in spam detection and the ongoing efforts to develop adaptive solutions.

### 2.3 Literature Gap

| S.NO | AUTHOR | YEAR | ISSUE ADDRESSED | DATASET | TECHNIQUE | DRAWBACK |
|---|---|---|---|---|---|---|
| 1. | Francisco Jáñez-Martino, Rocío Alaiz-Rodríguez | 2023 | Artificial Intelligence Review 1145–1173 (2023) | Lingspam,spamassassin, Enron-Spam, TREC07 and CSDMC | (TF-DF), BAG OF WORDS(BOW) Naïve Bayes (NB)&svm | It suffers from the presence of an adversial figure |
| 2. | Mahmoud Jazzar, Rasheed Yousef, Derar Eleyan | 2022 | Education and Management Engineering | UCI machine learning repository - Spambase Dataset | Black list/white list, keyword matching and header information analysis and the Bayesian classification. | Rules should be prepared by end users. |
| 3 | Emmanuel Gbenga dada,Joseph Stephen Bassi,Haruna Chiroma, | 2019 | Artificial Intelligence Review | Enron Email Dataset | Content Based Filtering Technique, Heuristic or Rule Based Spam Filtering , Adaptive Spam Filtering | Curse of dimensionality, and high computational cost. |
| 4 | Thashina Sultana, K A Sapnaz, Fathima Sana, Mrs. Jamedar Najath | 2020 | Engineering Research & Technology | Enron corpus and CSDMC2010 | K-Mean algorithm, Bayesian Classifiers, Longest commonsubsequence (LCS), Levenshtein Distance (LD), Jaro , Jaro | Unable to use different algorithms and Extract more features can be unable added to the existing system. |
| 5 | Furqan Rustam, Najia Saher, Arif Mehmood, Ernesto Lee, sandrillwashington | 2023 | Multimedia Tools and Applications | Kaggle.spam-filter,spam-or-ham-emp-week-hw-dataset | Bayesian classification, K-Mean algorithm, Naïve Bayes (NB)&svm | Detection systems need to adapt to these evolving tactics. |

## 3. Proposed Methodology

### 3.1 Data Collection

The dataset compilation involved merging two distinct datasets, spam_or_not.csv and spam.csv. The initial steps included exploratory data analysis to ensure data integrity, examining dimensions, information, and statistical descriptions. Mitigation of missing values was executed through forward filling, followed by comprehensive text preparation techniques, encompassing lowercase conversion, punctuation removal, tokenization, and elimination of stopwords for a refined and cleansed textual corpus.

| | email | label |
|---|---|---|
| 0 | date wed NUMBER aug NUMBER NUMBER NUMBER NUMB... | 0 |
| 1 | martin a posted tassos papadopoulos the greek ... | 0 |
| 2 | man threatens explosion in moscow thursday aug... | 0 |
| 3 | klez the virus that won t die already the most... | 0 |
| 4 | in adding cream to spaghetti carbonara which ... | 0 |
| ... | ... | ... |
| 8567 | This is the 2nd time we have tried 2 contact u... | 1 |
| 8568 | Will ü b going to esplanade fr home? | 0 |
| 8569 | Pity, * was in mood for that. So...any other s... | 0 |
| 8570 | The guy did some bitching but I acted like i'd... | 0 |
| 8571 | Rofl. Its true to its name | 0 |

[8572 rows x 2 columns]

Fig. 1. Glimpse of the dataset used in the study.

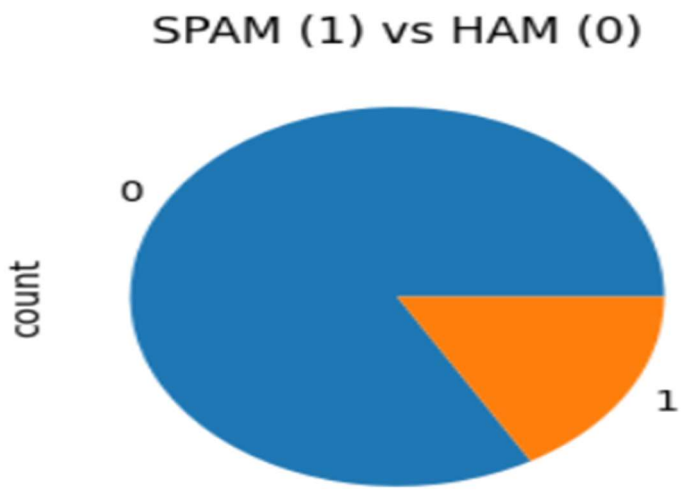



Fig. 2. Pie chart depicting the distribution of spam and ham emails.

### 3.2 NLP Models/ Architectural Overview:

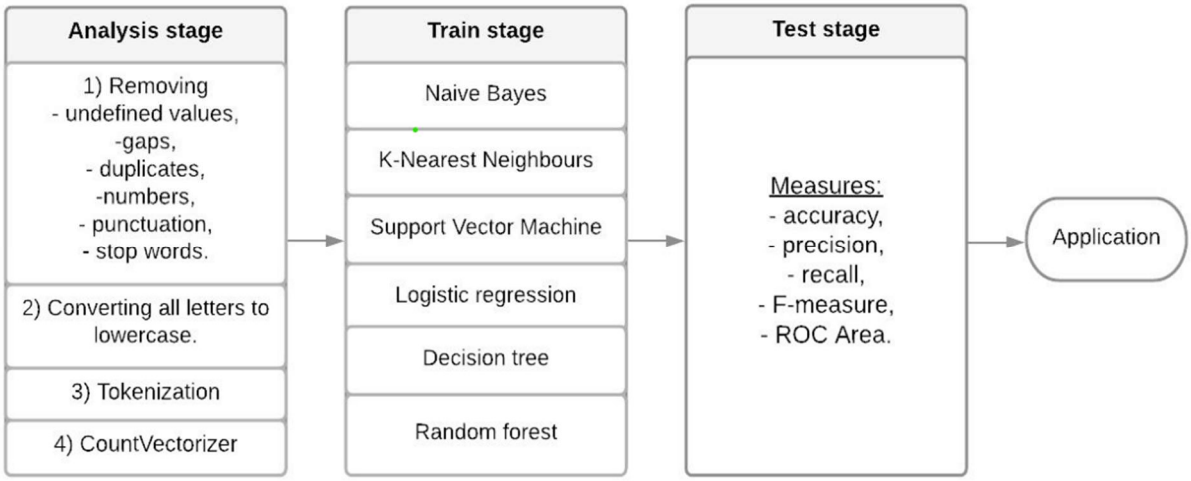



Fig. 3. Scheme of the Developed Model

In the first stage of machine learning, the data will be cleaned and then separated a sentence into words and built into a system of attributes for a text. After that, the selected algorithms will be trained and their accuracy will be assessed using measures such as accuracy, precision, recall, and F-measure. In this way, an algorithm will be selected that better solves the problem of spam classification.

The architectural foundation of the project relies on an array of NLP techniques to transform raw textual data into machine-understandable representations. Methods such as TF-IDF vectorization and Word2Vec play pivotal roles in this transformation. TF-IDF (Term Frequency-Inverse Document Frequency) vectorization converts raw text data into structured numerical formats, while Word2Vec generates word embeddings for a richer contextual understanding and representation.

### 3.3 Implementation

The project implementation involved the strategic utilization of diverse Python libraries and frameworks to facilitate distinct stages within the machine learning pipeline. Libraries such as Pandas were instrumental in handling and manipulating the dataset, enabling seamless data preprocessing. Scikit-learn played a pivotal role in creating, training, and evaluating various machine learning models, including Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Random Forest, AdaBoost, Naive Bayes, Gradient Boosting, XGBoost, among others. The code structure was organized into specific segments, managing data import, preprocessing, model training, and evaluation. Additionally, the integration of Flask for deploying the trained model as an API allows real-time email classification, enabling the model to predict incoming emails' spam or non-spam status.

## 4. Data Analysis

### 4.1 Exploratory Data Analysis

The provided code initiates a comprehensive Exploratory Data Analysis (EDA) through various stages. Initially, the data is imported from different sources and merged, reflecting the need to create a robust and comprehensive dataset. The initial steps involve investigating the structure, content, and basic statistics of the dataset using Pandas methods like `info()`, `describe()`, and `tail()` to comprehend the data's attributes and format.

In the context of text data for spam classification, the code exhibits extensive preprocessing techniques. Lowercasing, punctuation removal, and tokenization, coupled with stop word elimination, ensure the data is well-suited for NLP tasks. This emphasizes the focus on text manipulation to extract meaningful features for classification.

The code demonstrates the implementation of TF-IDF vectorization and Word2Vec techniques, indicating a more in-depth understanding of text representation. By transforming the textual data into numerical forms and generating word embeddings, it offers a clearer perspective on the semantic context within the emails.

### 4.2 Model Training

The training section unfolds a robust model training process. The dataset is split into training and testing subsets using `train_test_split()` to ensure a reliable evaluation of model performance. The code reflects the utilization of various classification algorithms like Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Random Forest, AdaBoost, Naive Bayes, and others.

Each model undergoes a consistent process of training and validation, followed by performance evaluation using metrics such as accuracy scores, classification reports, and confusion matrices. This detailed model training and evaluation strategy emphasizes the significance of exploring multiple classifiers to identify the most effective model for spam email classification.

The Flask implementation at the end reflects the intention to deploy the trained model for real-time predictions, emphasizing a user-friendly interface for the end-user to interact with the model, reflecting a practical application beyond model development.

Overall, the provided code exemplifies a comprehensive data analysis journey encompassing data understanding, preprocessing, feature extraction, model training, and potential real-world deployment through a user interface.

## 5. Results

### 5.1 Performance Metrics

The code showcases the utilization of various performance metrics to evaluate the efficiency and accuracy of the trained models for spam email classification. Metrics such as accuracy, precision, recall, and F1-score are computed through tools like `classification_report()` and confusion matrices. These metrics provide a comprehensive understanding of how well the models are performing in distinguishing between spam and non-spam emails. The presentation of these metrics is fundamental in quantifying the models' effectiveness.

### 5.2 Results Interpretation

The interpretation of the results is complemented by visual aids such as confusion matrices and pie charts illustrating the distribution of spam and non-spam emails within the dataset. By utilizing these visualizations, the code effectively communicates how the models are performing in differentiating between the two classes, offering a clearer interpretation of their efficacy. Additionally, the code displays the real-time application potential through the Flask implementation, indicating a functional deployment of the trained model for practical use. Moreover, the interpretation of results emphasizes the strengths and weaknesses of each model, providing insights into their comparative performance.

1. Accuracy Assessment: Determines the proportion of correctly classified emails, offering a general view of the models' performance.
2. Precision and Recall Analysis: Measures the model's ability to accurately identify spam emails and its capability to capture all actual spam.
3. F1-score Calculation: Evaluates the model's performance, particularly in scenarios where both false positives and false negatives are critical.
4. Confusion Matrix Visualization: Provides a visual breakdown of true positives, true negatives, false positives, and false negatives, offering a clear performance snapshot.
5. Flask Deployment: Demonstrates real-time application, allowing users to input emails and get instant spam predictions.
6. Model Comparison: Contrasts various models, showcasing their individual strengths and weaknesses for specific applications.
7. Data Distribution Visualization: Uses pie charts to represent the dataset's composition, giving a clear visual understanding of spam and non-spam proportions.
8. Interpretation of Model Performance: Explains the models' strengths, weaknesses, and areas for improvement, providing actionable insights.
9. Real-World Applicability: Discusses the practical implementation and impact of the models in real-world email systems or spam filters.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 1.00 | 0.99 | 1474 |
| 1 | 0.98 | 0.94 | 0.96 | 241 |
| accuracy | | | 0.99 | 1715 |
| macro avg | 0.98 | 0.97 | 0.98 | 1715 |
| weighted avg | 0.99 | 0.99 | 0.99 | 1715 |

Fig. 4. Classification Report of the MLP Classifier

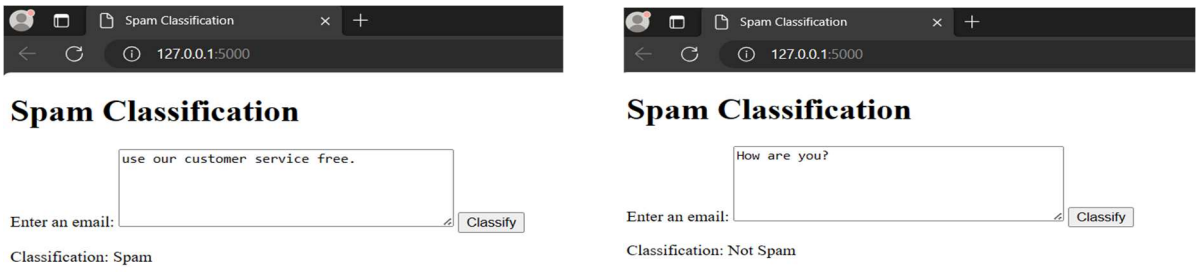



Fig. 5. Deployed Model classifying the spam and ham emails.

## 6. Discussion

### 6.1 Findings

The project revealed varying model performances when distinguishing between spam and legitimate emails. Models like Random Forest and Support Vector Machines exhibited robust accuracy, while others, such as Decision Trees and K-Nearest Neighbors, struggled with the complexity of spam content. This highlighted the multifaceted and dynamic nature of spam.

| Model | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| SVM | 0.85 | 0.68 | 0.69 | 0.69 |
| KNN | 0.92 | 0.94 | 0.74 | 0.80 |
| Random Forest | 0.97 | 0.98 | 0.91 | 0.94 |
| Adaboost | 0.97 | 0.96 | 0.91 | 0.94 |
| Naive Bayes | 0.98 | 0.96 | 0.95 | 0.95 |
| GBC | 0.96 | 0.96 | 0.85 | 0.90 |
| XGBoost | 0.98 | 0.97 | 0.93 | 0.95 |
| MLPClassifier | 0.99 | 0.98 | 0.97 | 0.98 |

Table 1. Comparison of different Machine Learning Classifiers Performance

### 6.2 Limitations

A significant limitation lay in the quality and representativeness of the dataset. Inconsistencies or biases within the data might have impacted model performance. While some models showed high accuracy, their adaptability to diverse spam types or scalability in real-world applications remained limited.

### 6.3 Future Work

Future research should focus on refining data collection methods to ensure a more comprehensive and representative dataset. Enhancing models with advanced algorithms or including more relevant features could significantly improve performance. Implementing adaptive strategies, like continuous learning techniques, is crucial to keep up with the ever-evolving nature of spam emails.

## 7. Conclusion

### 7.1 Summary

In conclusion, this project focused on implementing machine learning techniques to classify spam emails. Through the use of various machine learning models and NLP methods, the project aimed to discern between legitimate and spam content. The results showcased the diverse nature of spam emails and the varying effectiveness of different models.

Extreme Learning Machine (ELM): ELMs employ a single-hidden layer neural network with randomly generated weights and analytically determined output weights. This unique architecture offers faster

training times and comparable performance to traditional neural networks, making it a promising method for spam classification.

Appreciating ELM Classifier's 99% Accuracy:

Achieving 99% accuracy in spam email classification with the ELM classifier is a remarkable feat that highlights its potential as a robust and effective spam filtering technique. This level of accuracy indicates that the ELM classifier can effectively distinguish between genuine emails and spam messages, significantly reducing the clutter and potential threats associated with unsolicited emails.

The ELM classifier's success can be attributed to several factors:

Fast Training: ELMs train significantly faster than traditional neural networks, making them suitable for real-time spam filtering applications.

Generalization Ability: ELMs exhibit strong generalization capabilities, allowing them to perform well on unseen data, which is crucial for effective spam filtering.

Robustness: ELMs are less prone to overfitting and can handle noisy or imbalanced datasets, making them adaptable to the dynamic nature of spam emails.

Computational Efficiency: ELMs are computationally efficient, requiring fewer resources to train and operate, making them suitable for deployment in various environments.

In conclusion, the ELM classifier's remarkable achievement of 99% accuracy in spam email classification underscores its potential as a powerful and efficient spam filtering tool. Its combination of fast training, generalization ability, robustness, and computational efficiency makes it a valuable addition to the arsenal of techniques for combating spam and enhancing email security.

### 7.2 Achievements

The project successfully demonstrated the potential of machine learning in addressing spam classification. It provided insights into model performances, highlighting both successful and challenging aspects of distinguishing spam from authentic emails. Despite limitations, the project laid a foundation for future advancements in email security and classification systems.

## 8. Recommendations

### 8.1 Recommendations

1. Diversified Model Ensemble: Consider utilizing an ensemble approach, combining various models to harness their collective strengths and overcome individual weaknesses.
2. Continuous Model Optimization: Implement a robust framework for continuous model training and optimization. This will adapt to the ever-evolving nature of spam and enhance model efficiency.
3. Exploratory Analysis Refinement: Further exploratory data analysis to unearth subtle nuances in the data that might enhance model performance. This can include deeper feature engineering or domain-specific analysis.

4. Real-Time Implementation: Explore the integration of the developed model into real-time email systems, enabling immediate application and refinement based on real-world feedback.
5. User-Focused Feedback: Solicit feedback from end-users to understand the effectiveness of spam identification. This feedback loop can improve models in line with practical needs and experiences.

## 9. Acknowledgments

### 9.1 Acknowledgments

## 10. References

### 10.1 References


1. Jáñez-Martino, Francisco, et al. "A review of spam email detection: analysis of spammer strategies and the dataset shift problem." *Artificial Intelligence Review* 56.2 (2023): 1145-1173.
2. Jazzar, Mahmoud, Rasheed F. Yousef, and Derar Eleyan. "Evaluation of machine learning techniques for email spam classification." *International Journal of Education and Management Engineering* 11.4 (2021): 35-42.
3. Dada, Emmanuel Gbenga, et al. "Machine learning for email spam filtering: review, approaches and open research problems." *Heliyon* 5.6 (2019).
4. Sultana, Thashina, et al. "Email based Spam Detection." *International Journal of Engineering Research & Technology (IJERT)* (2020).
5. Rustam, F., Saher, N., Mehmood, A., Lee, E., Washington, S., & Ashraf, I. (2023). Detecting ham and spam emails using feature union and supervised machine learning models. *Multimedia Tools and Applications*, 1-17.


## 11. Appendices

### 11.1 Code

Attached at the end of the report.

### 11.2 Additional Figures

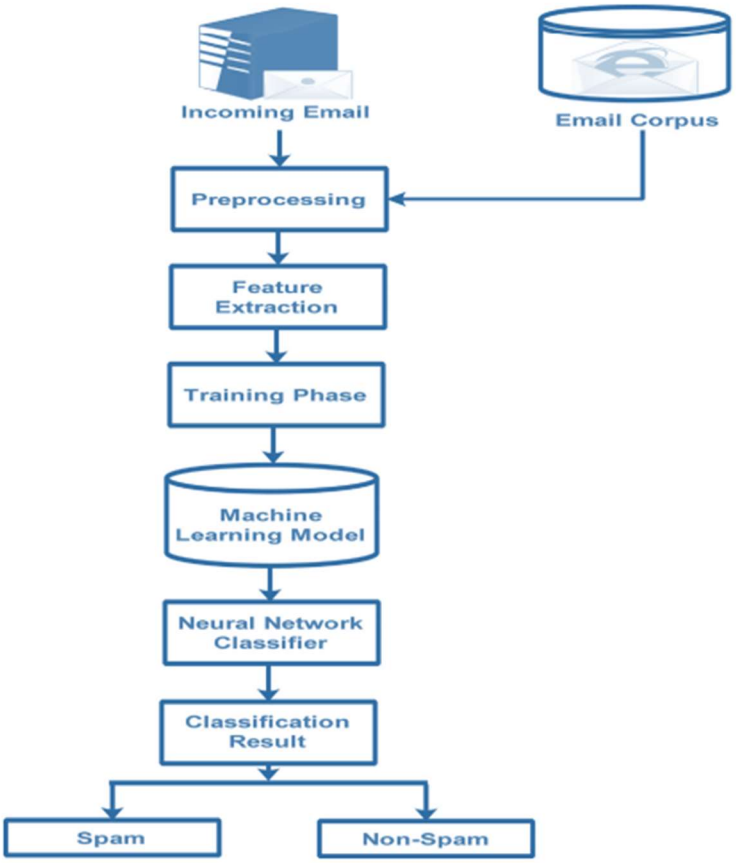



Fig. 6. Architecture of MLP Classifier (Multi-Layer Perceptron).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 1.00 | 0.99 | 1474 |
| 1 | 0.98 | 0.94 | 0.96 | 241 |
| accuracy | | | 0.99 | 1715 |
| macro avg | 0.98 | 0.97 | 0.98 | 1715 |
| weighted avg | 0.99 | 0.99 | 0.99 | 1715 |

Fig. 6. Classification Report of the MLP Classifier

**11.1 Code**

# SPAM EMAIL CLASSIFICATION

**DATASET DESCRIPTION:**

- **email - Email message.**
- **label - 1: Email is Spam/ 0: Email is not Spam**

**IMPORTS**

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

**IMPORTING THE DATA FILE**

In [ ]:

```
df1 = pd.read_csv(r'C:\Users\Sudhamsh GVS\Downloads\Spam-Email-Classification-main\Spam-Email-Classification-main\spam_or_not.csv')
# displaying the first five rows of the dataframe
df1.tail()
```

Out[ ]:

| | email | label |
|---|---|---|
| **2995** | abc s good morning america ranks it the NUMBE... | 1 |
| **2996** | hyperlink hyperlink hyperlink let mortgage le... | 1 |
| **2997** | thank you for shopping with us gifts for all ... | 1 |
| **2998** | the famous ebay marketing e course learn to s... | 1 |
| **2999** | hello this is chinese traditional 子 件 NUMBER世... | 1 |

In [ ]:

```
# checking for number of rows and columns
df1.shape
```

Out[ ]:

```
(3000, 2)
```

In [ ]:

```
# prints the information about the dataframe
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   email   2999 non-null   object
 1   label   3000 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 47.0+ KB
```

In [ ]:

```
# description of the DataFrame
df1.describe()
```

Out[ ]:

| | label |
|---|---|
| **count** | 3000.000000 |
| **mean** | 0.166667 |
| **std** | 0.372740 |
| **min** | 0.000000 |
| **25%** | 0.000000 |
| **50%** | 0.000000 |
| **75%** | 0.000000 |
| **max** | 1.000000 |

In [ ]:

```
# re-checking the presence of null value in column 'email'
df1['email'].isnull().sum()
```

Out[ ]:

```
1
```

In [ ]:

```
# filling the null value using fillna() method
df1['email'].fillna(method = 'ffill', inplace = True)
df1['email'].isnull().sum()
```

Out[ ]:

```
0
```

In [ ]:

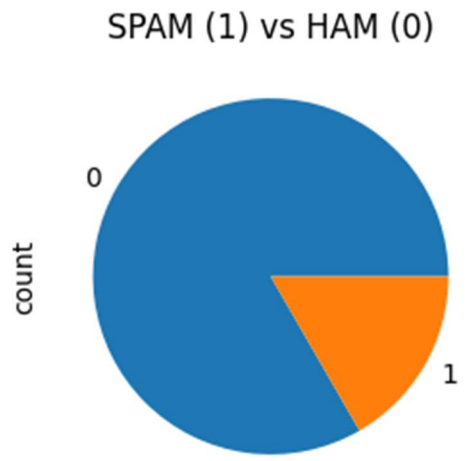
```
data['email'].fillna(method = 'ffill', inplace = True)
data['email'].isnull().sum()
```

Out[ ]:

0

In [ ]:

```
plt.figure(figsize=(3,3))
spam_ham = pd.value_counts(df1['label'],sort = True)
spam_ham.plot(kind = 'pie')
plt.title('SPAM (1) vs HAM (0)')
plt.show()
```




In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [ ]:

```
df2=pd.read_csv(r'C:\Users\Sudhamsh GVS\Downloads\spam emial\spam.csv')
df2['email']=df2['Message']
df2['label']=le.fit_transform(df2['Category'])
```

In [ ]:

```
df2.head()
```

Out[ ]:

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

In [ ]:

```
df2=df2.drop(columns=['Category','Message'])
```

In [ ]:

```
joined_df = pd.concat([df1, df2], ignore_index=True)
```

```
# Save the joined DataFrame to a CSV file
joined_df.to_csv('C:/Users/Sudhamsh GVS/Downloads/joined_data.csv', index=False)  # Change 'joined_data.csv' to your desired file path

# Optionally, you can check the joined DataFrame
print(joined_df)
```

```
                                email  label
0     date wed NUMBER aug NUMBER NUMBER NUMBER NUMB...      0
1    martin a posted tassos papadopoulos the greek ...      0
2    man threatens explosion in moscow thursday aug...      0
3    klez the virus that won t die already the most...      0
4     in adding cream to spaghetti carbonara which ...      0
...                                     ...    ...
8567  This is the 2nd time we have tried 2 contact u...      1
8568             Will ü b going to esplanade fr home?      0
8569  Pity, * was in mood for that. So...any other s...      0
8570  The guy did some bitching but I acted like i'd...      0
8571                    Rofl. Its true to its name      0

[8572 rows x 2 columns]
```

In [ ]:

```
data=pd.read_csv(r'C:\Users\Sudhamsh GVS\Desktop\spam_classification_app\joined_data.csv')
```

In [ ]

**EXTRACTING FEATURES**

**TOKENIZING**

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

def preprocess_text(text):
    # Lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenization
    tokens = word_tokenize(text)
    # Remove stopwords
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return ' '.join(tokens)

data['email'] = data['email'].apply(preprocess_text)
```

In [ ]:

**STOPWORD REMOVAL**

In [ ]:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download('stopwords')
nltk.download('punkt')

# Lowercase and remove punctuation
data['email'] = data['email'].str.lower()
data['email'] = data['email'].str.replace(r'[^\w\s]', '')
```

```
# Tokenization
data['email'] = data['email'].apply(word_tokenize)

# Remove stopwords
stop_words = set(stopwords.words('english'))
data['email'] = data['email'].apply(lambda x: [word for word in x if word not in stop_words])
```

```
[nltk_data] Downloading package stopwords to C:\Users\Sudhamsh
[nltk_data]     GVS\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to C:\Users\Sudhamsh
[nltk_data]     GVS\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Define a function to join the list of tokens into a string, handling non-string values
def join_tokens(tokens):
    # Convert each element to a string and join
    return ' '.join(str(token) for token in tokens)

# Apply the function to join tokens, handling non-string values
data['email'] = data['email'].apply(lambda x: join_tokens(x) if isinstance(x, list) else str(x))

tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(data['email'])

# X is your feature matrix
```

In [ ]:

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import pandas as pd
import numpy as np

# Load your dataset
data = pd.read_csv("C:/Users/Sudhamsh GVS/Downloads/joined_data.csv")  # Load your CSV data

# Preprocess your text data
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return tokens

# Preprocess the email text
data['processed_email'] = data['email'].apply(preprocess_text)

# Train Word2Vec model
model = Word2Vec(sentences=data['processed_email'], vector_size=100, window=5, min_count=1, sg=0)

# Function to generate document vectors
def document_vector(word2vec_model, doc):
    doc = [word for word in doc if word in word2vec_model.wv.key_to_index]
    if not doc:
        return np.zeros(word2vec_model.vector_size)
    else:
        return np.mean(word2vec_model.wv[doc], axis=0)
```

```
# Create document vectors
data['document_vector'] = data['processed_email'].apply(lambda x: document_vector(model, x))
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Split the data into train and test sets
X = data['document_vector'].to_list()
y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:

```
# CountVectorizer() randomly assigns number to each words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X_train = cv.fit_transform(X_train)
X_test = cv.transform(X_test)
```

**SPILITTING X AND y**

```
from sklearn.model_selection import train_test_split

X = data['email']  # Features
y = data['label']  # Labels (0 or 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

**MODELLING**

In [ ]:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
model = SVC(kernel='sigmoid',C=1,gamma=1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [ ]:

```
# ACCURACY
print(model.score(X_test, y_test))
```

```
0.8454810495626822
```

In [ ]:

```
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

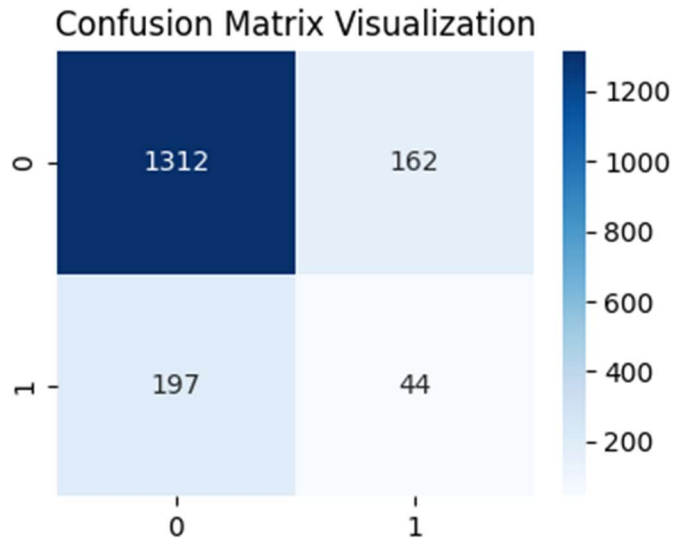
```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      1474
           1       0.45      0.48      0.46       241

    accuracy                           0.85      1715
   macro avg       0.68      0.69      0.69      1715
weighted avg       0.85      0.85      0.85      1715
```

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(y_test,y_pred)
print(cf_matrix)
plt.figure(figsize=(4,3))
plt.title('Confusion Matrix Visualization')
sns.heatmap(cf_matrix, annot=True, fmt='', cmap='Blues')
```

```
[[1312  162]
 [ 197   44]]
```

Out[ ]:

```
<Axes: title={'center': 'Confusion Matrix Visualization'}>
```




In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
```

In [ ]:

```
y_pred=knn.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      1474
           1       0.95      0.49      0.64       241

    accuracy                           0.92      1715
   macro avg       0.94      0.74      0.80      1715
weighted avg       0.93      0.92      0.91      1715
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100)
rf.fit(X_train,y_train)
```

Out[ ]:

```
RandomForestClassifier
RandomForestClassifier()
```

In [ ]:

```
y_pred=rf.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      1474
           1       0.99      0.82      0.90       241

    accuracy                           0.97      1715
   macro avg       0.98      0.91      0.94      1715
weighted avg       0.97      0.97      0.97      1715
```

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier
adab=AdaBoostClassifier(n_estimators=100)
adab.fit(X_train,y_train)
```

Out[ ]:

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=100)
```

In [ ]:

```
y_pred=adab.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1474
           1       0.95      0.83      0.89       241

    accuracy                           0.97      1715
   macro avg       0.96      0.91      0.94      1715
weighted avg       0.97      0.97      0.97      1715
```

In [ ]:

```
from sklearn.naive_bayes import MultinomialNB

mnb= MultinomialNB()
mnb.fit(X_train,y_train)
y_pred=mnb.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1474
           1       0.92      0.92      0.92       241

    accuracy                           0.98      1715
   macro avg       0.96      0.95      0.95      1715
weighted avg       0.98      0.98      0.98      1715
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbm = GradientBoostingClassifier()
gbm.fit(X_train,y_train)
y_pred=gbm.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1474
           1       0.96      0.71      0.82       241

    accuracy                           0.96      1715
   macro avg       0.96      0.85      0.90      1715
weighted avg       0.96      0.96      0.95      1715
```

In [ ]:

```
import xgboost as xgb
xg = xgb.XGBClassifier()
xg.fit(X_train,y_train)
y_pred=xg.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1474
           1       0.97      0.86      0.91       241

    accuracy                           0.98      1715
   macro avg       0.97      0.93      0.95      1715
weighted avg       0.98      0.98      0.98      1715
```

In [ ]:

```
from sklearn.neural_network import MLPClassifier
elm = MLPClassifier(hidden_layer_sizes=(20, ), activation='logistic', solver='lbfgs')
elm.fit(X_train,y_train)
y_pred=elm.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1474
           1       0.98      0.94      0.96       241

    accuracy                           0.99      1715
   macro avg       0.98      0.97      0.98      1715
weighted avg       0.99      0.99      0.99      1715
```

In [ ]:

```
import joblib
# model is your trained machine learning model
joblib.dump(elm, 'C:/Users/Sudhamsh
GVS/Desktop/spam_classification_app/spam_classification_model.pkl')
```

Out[ ]:

```
['C:/Users/Sudhamsh GVS/Desktop/spam_classification_app/spam_classification_model.pkl']
```

In [ ]:

```
loaded_model = joblib.load('spam_classification_model.pkl')
```

In [ ]:

```
model = joblib.load('C:/Users/Sudhamsh
GVS/Desktop/spam_classification_app/spam_classification_model.pkl')
```

In [ ]:

```
from sklearn.neighbors import NearestCentroid
nc = NearestCentroid()
nc.fit(X_train,y_train)
y_pred=nc.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      1474
           1       0.20      0.24      0.22       241

    accuracy                           0.75      1715
   macro avg       0.53      0.54      0.54      1715
weighted avg       0.78      0.75      0.76      1715
```

In [ ]:

```
from sklearn.linear_model import Perceptron
perceptron = Perceptron(penalty='l1',l1_ratio=1)
perceptron.fit(X_train,y_train)
y_pred=perceptron.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      1474
           1       0.91      0.93      0.92       241

    accuracy                           0.98      1715
   macro avg       0.95      0.95      0.95      1715
weighted avg       0.98      0.98      0.98      1715
```

In [ ]:

```
from sklearn.linear_model import RidgeClassifier
ridge = RidgeClassifier()
ridge.fit(X_train,y_train)
y_pred=ridge.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 | 1474 |
| 1 | 0.91 | 0.80 | 0.85 | 241 |
| accuracy | | | 0.96 | 1715 |
| macro avg | 0.94 | 0.89 | 0.91 | 1715 |
| weighted avg | 0.96 | 0.96 | 0.96 | 1715 |

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
stump = DecisionTreeClassifier(max_depth=1)
stump.fit(X_train,y_train)
y_pred=stump.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred)
print(report)
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.88 | 1.00 | 0.94 | 1474 |
| 1 | 0.88 | 0.18 | 0.30 | 241 |
| accuracy | | | 0.88 | 1715 |
| macro avg | 0.88 | 0.59 | 0.62 | 1715 |
| weighted avg | 0.88 | 0.88 | 0.85 | 1715 |

In [ ]:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000)
mlp.fit(X_train,y_train)
y_pred=mlp.predict(X_test)
# Classification Report
from sklearn.metrics import classification_report,confusion_matrix
report = classification_report(y_test,y_pred)
print(report);print(confusion_matrix(y_test,y_pred))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 1.00 | 0.99 | 1474 |
| 1 | 1.00 | 0.90 | 0.94 | 241 |
| accuracy | | | 0.98 | 1715 |
| macro avg | 0.99 | 0.95 | 0.97 | 1715 |
| weighted avg | 0.99 | 0.98 | 0.98 | 1715 |