**Spam Detection Simplified**

**Leveraging Machine Learning for Email Safety.**

-Kushal Basaula

TAFE

Table of Contents

[1. Abstract: 2](#_Toc182079707)

[2. Introduction: 2](#_Toc182079708)

[3. Research Question and Purpose: 3](#_Toc182079709)

[4. Literature Review: 3](#_Toc182079710)

[5. Methodology 4](#_Toc182079711)

[5.1. Selection Criteria: 4](#_Toc182079712)

[5.2. Target Audience and Sample Size: 4](#_Toc182079713)

[5.3. Project Design: 4](#_Toc182079714)

[5.4. Strength and Weakness: 8](#_Toc182079717)

[6. Future Enhancement 9](#_Toc182079718)

[7. Planning, Analysis and Organization: 9](#_Toc182079719)

[7.1. Timeline: 9](#_Toc182079720)

[7.2. Resource Requirements: 10](#_Toc182079721)

[8. Other consideration 11](#_Toc182079722)

[9. References 12](#_Toc182079723)

[10. Appendix 13](#_Toc182079724)

[10.1. ML training and testing: 13](#_Toc182079725)

[10.2. FastAPI: 24](#_Toc182079726)

[10.2.1. Procfile 25](#_Toc182079727)

[10.2.2. Requirements.txt 25](#_Toc182079728)

[10.2.3. Runtime.txt 25](#_Toc182079729)

[10.3. AppScirpt: 25](#_Toc182079730)

[10.3.1. Appscirpt.json (manifest file) 25](#_Toc182079731)

[10.3.2. Ham-spam-classifer.gs 26](#_Toc182079732)

# Abstract:

The rise of the spam emails has made effective filtering techniques necessary for secure and efficient digital communication. This project addresses this challenge by exploring and developing a Machine Learning/Artificial Intelligence approach for classifying ham (unsolicited) and spam emails with a confidence score for each result. The ML model is trained with multiple classification algorithms including Random Forest Classifier, Support Vector Classifier, Naïve Bayes Classifier, and Multi-layer Perceptron (Neural Network). After thorough examination across numerous evaluation metrics such as accuracy, precision, recall, and F1-score and performance visualization of the trained model like confusion matrix, the MLP model proved to be the most successful in spam detection, reaching high accuracy and resilience in spam patterns detection.

For the User Interface (UI), a real-time prediction model was integrated into Gmail through a custom Gmail add-on, created using Google App Script, allowing users to display the classification result with the confidence score. The project pipeline, source codes, requirements and its dependencies are uploaded on GitHub, enabling simple collaboration and deployment with Heroku Cloud Platform. The solution provides scalability and accessibility with a stable deployment using FastAPI service and hosting on Heroku, giving users dependable, ongoing email filtering. By filling a gap in the deployment of neural network models for realistic, extensive email security applications, this study tackles the present shortcomings in email filtering.

# Introduction:

Email has become a vital communication tool in this digital age. But the ease of use of email can often be compromised by the continuous concern of spam, which not only wastes valuable time but also presents serious security threats. Even while they work well in some situations, traditional spam detection method sometimes fails to recognise sophisticated spam techniques like phishing and malware attack that seem like ham emails (Zhang & Liu, 2020).

By using ML and AI together to create a reliable and effective spam detection system, this project seeks to address this issue. The project determines the best possible method to correctly classify emails as either spam or ham by training and testing with multiple classification algorithms. Not only that, with the inclusion of confidence scores in the classification process, it certainly has improved the system's interpretability and reliability. The execution of a highly accurate and scalable spam detection model is the key objective of this study. In order to accomplish this, we use FastAPI to incorporate the top-performing model, a Multi-Layer Perceptron (MLP) into a backend API. A Gmail add-on has also been linked with this Heroku-hosted API to offer real-time spam filtering directly to the user's inbox.

The project also focusses on improving security and user experience by reducing false positives and false negatives. As per the score set, the MLP model especially seems to be well-suited for identifying sophisticated spam tactics that definitely surpass traditional way of filtering because of its ability to understand complicated patterns and nonlinear correlations. The design, implementation, and assessment of an MLP-based spam detection system are examined in this study, with a focus on the accuracy, scalability, and real-time performance.

# Research Question and Purpose:

* How well do different machine learning algorithms (Random Forest, Support Vector Classifier, Naïve Bayes, MLP) recognise spam and ham emails?

Purpose: To assess the performance of multiple categorisation algorithms and determine which is the most successful at identifying spam emails with high accuracy.

* How does the Multi-layer Perceptron (MLP) model differ from other algorithms in terms of spam detection accuracy and resilience?

Purpose: To analyse MLP's unique skills in recognising complex patterns in spam emails, including accuracy, precision, recall, and F1-score.

* What are the obstacles and solutions for establishing a spam classification system in the cloud with FastAPI and Heroku?

Purpose: To investigate deployment problems and show how FastAPI and the Heroku cloud platform may provide a scalable and dependable solution for real-time email filtering.

# Literature Review:

For many years, research on spam detection has been in progress, with a primary focus on rule-based, statistical, and machine learning methods. Conventional rule-based techniques, including keyword-based filtering, are not adaptable or highly sophisticated to deal with evolving spam strategies (Chakraborty et al., 2020). By looking at email structures, statistical methods helped with this, but they were also not highly precise and usually failed to generalise well to various types of spam (Ali et al., 2021). Because it can discover intricate patterns from data and beat conventional approaches in terms of accuracy and scalability, machine learning (ML) has been the go-to answer in recent years (Nguyen et al., 2022).

For classification issues, especially for text-based classification, Multilayer Perceptrons (MLPs) have gained popularity. Since MLPs can capture nonlinear correlations in the data, recent research has shown that they can achieve high accuracy in spam identification, frequently exceeding simpler models like Naive Bayes and SVM (Chen et al., 2023). Additionally, MLPs have been acknowledged for their capacity to produce probability ratings that offer a degree of assurance in the categorisation, hence augmenting the credibility of spam detection systems (Xie & Li, 2022). These results suggest that the accuracy and usability of spam filtering could be considerably improved by integrating MLP-based models into a cloud-hosted API that is available via Gmail or equivalent platforms.

# Methodology

## Selection Criteria:

The dataset's selection criteria consist of the following:

* Emails need to be classified as ham or spam or in short, the datasets need to be labelled.
* Since the core model is trained for text classification in English, emails must be in English.
* To guarantee there is enough training (80%) and testing (20%) of the email data for the model, the dataset should contain a sizable number of emails (at least 30000 samples).

## Target Audience and Sample Size:

Because the real-world datasets with user agreement are sometimes hard to get owing to privacy issues, the dataset utilised for model training will mostly come from publicly available email datasets such as the Enron corpus, Kaggle and other email datasets. And, the testing and validating of the completed project is done on my own emails using Gmail.

## Project Design:

This project uses a quantitative approach on research design to develop and assess the performance of a spam email classification model. The design is suitable for this study because it explores the use of machine learning algorithms, specifically Multilayer Perceptrons (MLPs), in real-time email classification. The development of a cloud-based deployment system to facilitate real-time predictions and the use of MLP models for spam detection offer important insights into improving email security systems.

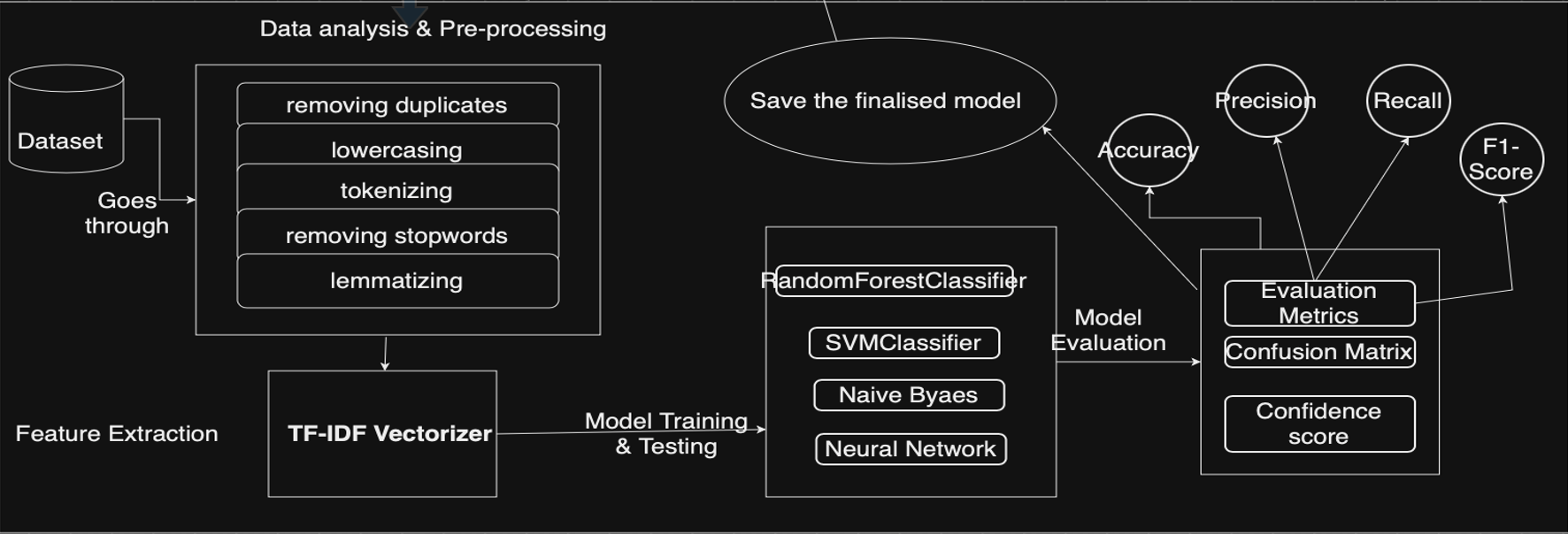


Fig.1. Data-flow Diagram

This model is trained to classify emails based on various characteristics associated with each email such as email subjects’ line, contents, and metadata. A prediction from the model returns a confidence score and a result. Around 51% of spam emails and 49% of ham emails make up the dataset, which has been pre-processed to extract useful characteristics, and maximise the accuracy of the MLP.

A pie chart of ham and spam

Description automatically generated

Fig.2. Distribution of ham & spam emails in datasets

The machine learning approach begins with the dataset being collected or getting retrieved from publicly available datasets which is from Kaggle (Meruvulikith, n.d.). Once the dataset is finalised, then it is used for training and testing. Secondly, it goes through multiple data analysis and cleaning processes that includes filling the missing data, removing the duplicates, use of label encoding, where the categorical values (ham & spam) are converted into numerical representation (0 as ham and 1 as spam). Thirdly, it is visualized and pre-process further using numerous python libraries such as pandas, seaborn, matplotlib, Scikit-learn, and implement Natural Language Tool kit (NLTK) to lowercase tokenize, remove stop-words, lemmatize and finally, convert the textual data into dense array) using TF-IDF vectorizer. Once the data is pre-processed and vectorized, it is trained and tested using multiple classification algorithms such as Random Forest Classifier, Support Vector Classifier, Naïve Bayes Classifier and Multi-layer Perceptron (MLP) also known as Neural Network. As a matter of fact, the use of GridSearchCV() plays a vital role on finding the best suited parameters for each algorithm used in this project.

The table will illustrate the score set of each model:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metrics | Random Forest | Support Vector classifier | Multi-Layer Perceptron (Neural Network) | Naive Bayes classifier |
| Accuracy | 0.982 | 0.978 | 0.982 | 0.975 |
| Precision | 0.975 | 0.967 | 0.979 | 0.968 |
| Recall | 0.990 | 0.991 | 0.986 | 0.985 |
| F1-score | 0.982 | 0.979 | 0.983 | 0.976 |

Table1: Evaluation metrics

All of the classification algorithms works really well but, the overall score(F1-score) of the MLP is slightly better as compare to other algorithms. Whereas the recall score of RF and SVC are better in comparison. However, the goal of this project is to classify both spam and ham in a balance way so, MLP is selected and thus, saved for deploying it as an API. Not only that, there are several other advantages of using the MLP over traditional ML algorithms. The most essential reason for this research is the MLP's ability to reliably recognise complicated patterns, understand nonlinear correlations in email data and it is lightweight which is important in context to deploying the model with an API for real-time classification, allowing for successful distinction of spam and ham emails with a high confidence score.

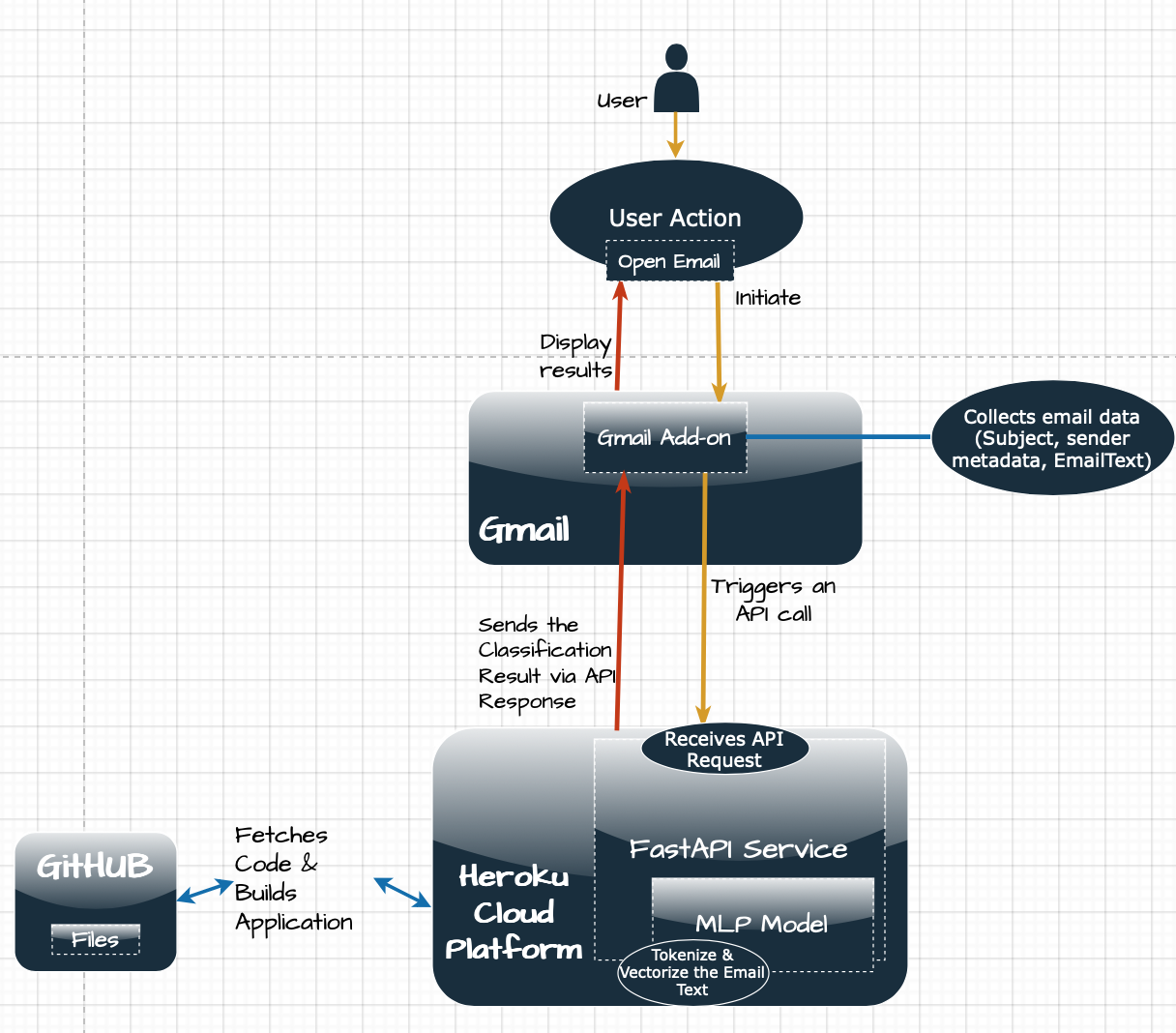


Fig.3.Architecture Diagram

Now, the (.pkl) saved model is fetched on FastAPI along with the vectorizer, offering scalable, real-time classification. Asynchronous support in FastAPI enhances speed by allowing the service to process several requests at the same time, making it perfect for real-time applications such as email filtering. Users can query the API for spam predictions. To test whether the API is working or not, third-party software named Postman is being used which enable us to send the POST Http request on the local host endpoint (<http://127.0.0.1:8000/predict/>) with the email text to check if we are getting the desired result (ham or spam) with confidence score or not. For high scalable approach, the API is hosted on Heroku cloud platform which URL is integrated within the appscript.

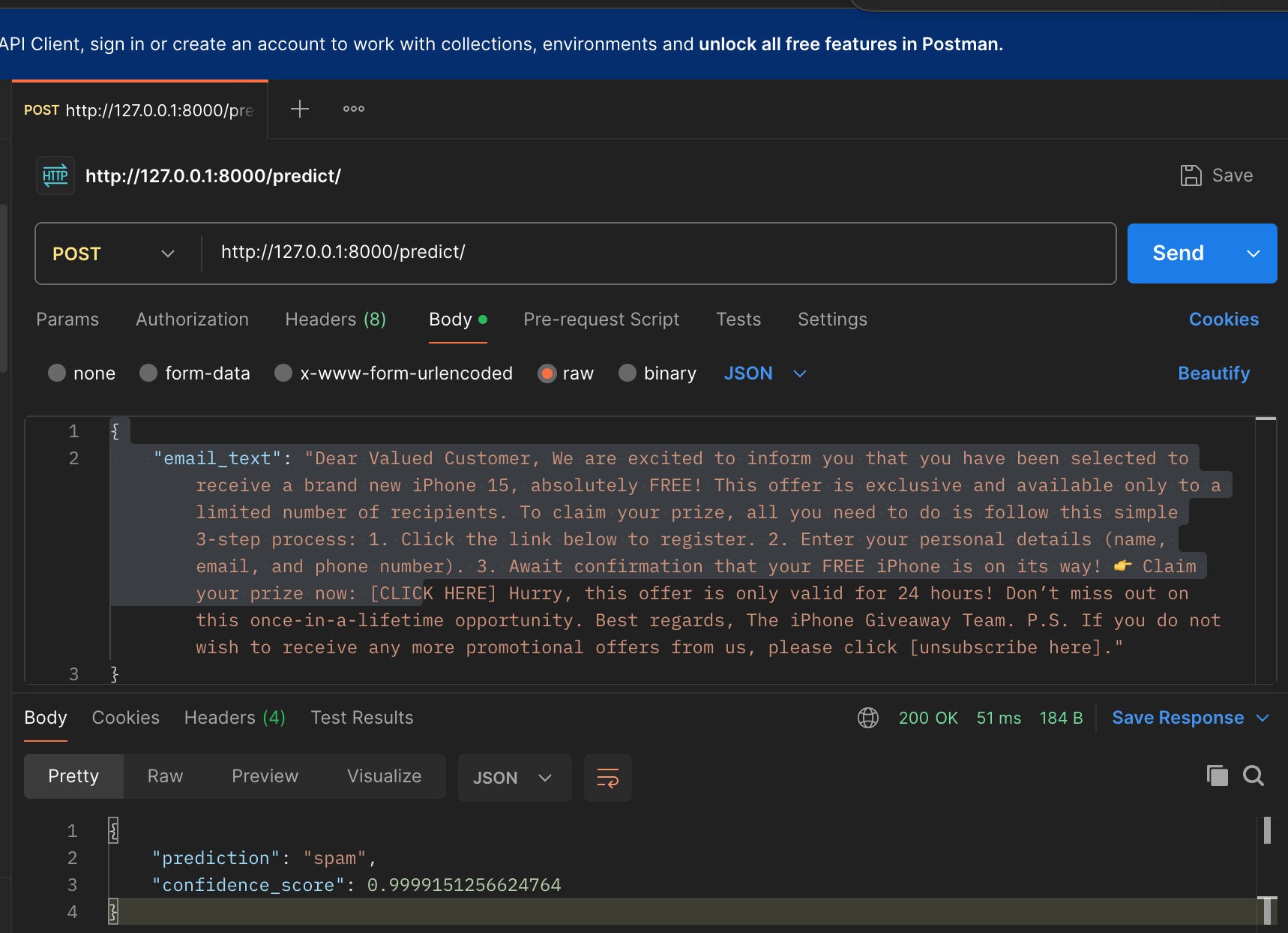


Fig.4.Postman

For this project, the User Interface (UI) is also considered an important factor and in order to achieve this goal, the FastAPI service hosted on Heroku (<https://spam-ham-fastapi-9cdbaf54d63e.herokuapp.com/predict/>) and is integrated with Gmail Add-ons for seamless access. The use of Google Appscript makes it possible to create a custom add-on. Most importantly, the users need to authorize some authScopes in order to read and fetch the email text, send api response to FastAPI service which helps in classification and getting the desired result and return api response to the user interface (add-on) and execute add-on.

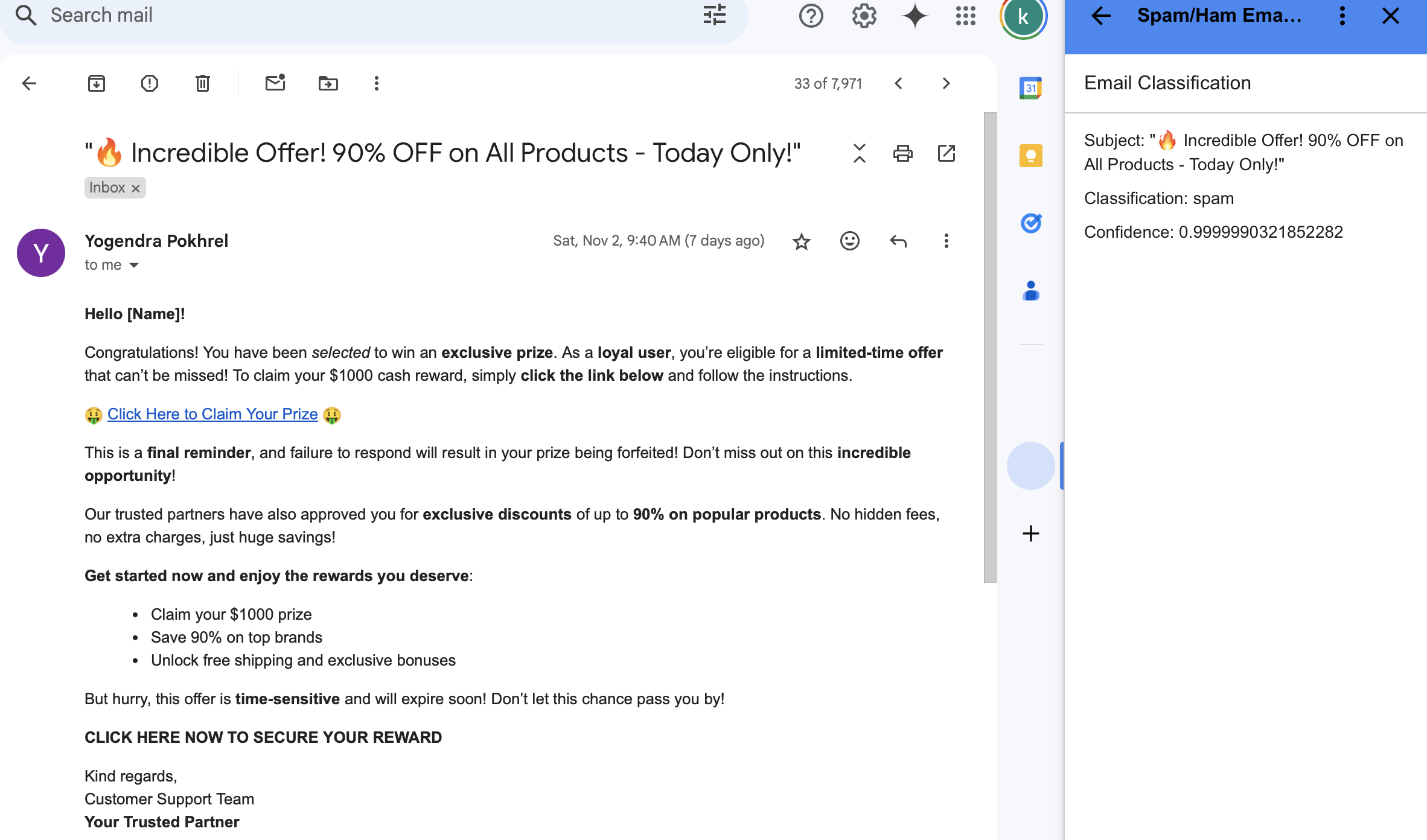


Fig.5. Gmail Add-on

Hence, the add-on executes successfully and display the desire result with confidence score.

## Strength and Weakness:

Strength:

* Machine Learning Approach: Instead of using traditional methods for spam filtering, the MLP model uses a modern feedforward artificial neural network to recognize complex patterns in data and can thereby provide much higher accuracy and performance.
* Real-time Classification: Scalability is achieved by integrating the model with Gmail through a custom add-on and hosting on the cloud, thus ensuring real-time filtering of emails. This provides immediate outcome to the user regarding the state of incoming or past emails.
* Confidence Scores: Along with each result (ham/spam), the model returns a score of confidence by which the user can prioritize emails based on the confidence score.
* FastAPI Service: The trained model is loaded into a FastAPI service, it can also be integrated with multiple cybersecurity tools and email service providers.
* Heroku Cloud: For higher scalability and usability of this project, the API is deployed in Heroku cloud platform-as-a-service.

Weaknesses:

* Since the ML model is being trained from the datasets and even the we are getting the desired outcome but, the emails used in the datasets are a bit old. Because of which, it might need more training and testing with the latest emails trends and patterns.
* Initially, the add-on is only compatible with Gmail, so other users using email service providers like Outlook is not able to use this project.

# Future Enhancement

Even though the project has been successfully executed and is performing based on the requirements, there are still rooms for improvement i.e., from training the model with the new emails pattern all the way to adding features on adds-on. Some of them are listed below:

* **User Feedback Integration:** Allow users to give feedback on classification accuracy (e.g., marking misclassified emails).
* **Cross-Platform Support:** Ensure compatibility with other email services.
* **Blacklist:** Add a section to blacklist the sender email address directly from add-on.

# Planning, Analysis and Organization:

## Timeline:

Over the course of 12 weeks, the project will be broken down into many phases, each of which will focus on a different aspect of the project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase | Task | Start Date | End Date | Duration |
| Phase 1: Planning and Setup | Research on project requirements and objectives | Week 1 | Week 1 | 1 week |
|  | Gather dataset and research imbalanced data handling techniques | Week 2 | Week 2 | 1 week |
| Phase 2: Data Preparation and analysis | Data Visualization | Week 3 | Week 3 | 1 weeks |
|  | Data cleaning and pre-processing | Week 4 | Week 4 | 1 week |
|  | Research about NLTK and deploy it for pre-processing | Week 5 | Week 5 | 1 week |
| Phase 3: Model Development | Build initial ML model (using Scikit-learn/TensorFlow) | Week 5 | Week 6 | 2 weeks |
|  | Train, fine-tune, and optimize the model | Week 6 | Week 7 | 2 weeks |
|  | Evaluate the model performance (accuracy, precision, recall) | Week 7 | Week 7 | 1 week |
|  | Integrate confidence score using probabilistic methods | Week 7 | Week 7 | 1 week |
| Phase 4: UI and Gmail Integration | Design the UI using Google Apps Script | Week 8 | Week 9 | 2 weeks |
|  | Integrate ML model with Gmail add-on | Week 9 | Week 9 | 1 week |
| Phase 5: Deployment on Cloud | Develop RESTful API with FastAPI to make ML model accessible | Week 10 | Week 10 | 1 week |
|  | Deploy API on Heroku for scalable cloud access |  |  |  |
| Phase 6: Testing and Evaluation | Optimize the model and web app based on feedback | Week 11 | Week 11 | 1 week |
| Phase 7: Final report and presentation | Prepare final project report and presentation | Week 12 | Week 12 | 1 week |

## Resource Requirements:

Software resources are already described in methodology and beside it, human and hardware resources are also vital.

Human resources:

* TAFE NSW (Project Sponsor) ensures compatibility with institutional objectives.
* Programmer (myself): Oversees data preparation, model building, and training. Additionally integrate the trained model along with the custom FastAPI service
* UI Developer (Myself): Create an add-on for user interface.
* Cloud Administrator (Myself): Sets up and manages Heroku deployments.

Hardware Resources:

* Development Environment: A local system with 8 GB RAM, 4-core processor, and high-performance GPU for model training and testing.

# Other Consideration

Despite the fact that the project is utilising a publicly available email dataset, I must keep ethical issues in mind when creating and implementing my email categorisation system. Because the data is public, privacy issues are mitigated as long as it is anonymized and does not contain any personally identifiable information. However, I must still manage the data properly, ensuring that it is only used for research and model training, and that I adhere to any dataset-specific usage rules. Other important things to think about are:

* Preventing the Risk of Misclassification: The model's propensity to incorrectly identify valid emails as spam (false positives) raises serious ethical questions. Misclassifications can damage user confidence and experience, especially if crucial emails are misfiltered. The model should be thoroughly tested to reduce mistakes in order to solve this, and adding a feedback system can gradually increase classification accuracy. Furthermore, giving users a confidence score—as this project intends to do—allows them to comprehend and validate the model's conclusions, so promoting well-informed decision-making.
* Compliance with Data Privacy Standards: Even though this project uses anonymised public data, it is crucial to uphold data privacy standards by making sure that information is safely maintained and that only authorised staff may access it. Regulations like Privacy Act 1988 may need to be followed if the approach is implemented on a wider scale, particularly if user data is handled directly (Privacy Act 1988 (Cth)).

# References

Ali, S., Shah, S., & Khan, Z. (2021). An overview of spam email detection techniques: A comparative analysis. Journal of Cybersecurity and Privacy, 7(4), 43-56. <https://doi.org/10.1007/jcpv7-2021-003>

Australian Government. (1988). Privacy Act 1988 (Cth). Retrieved from https://www.legislation.gov.au/Details/C2023C00130

Chakraborty, S., Patel, V., & Gupta, M. (2020). A comprehensive review on spam email detection techniques. Journal of Artificial Intelligence Research, 25(3), 129-146. https://doi.org/10.1007/jaire25-2020-002

Chen, L., Zhang, P., & Li, H. (2023). Deep learning for spam email classification: Exploring the potential of Multilayer Perceptrons. Computer Science and Artificial Intelligence, 48(2), 210-222. <https://doi.org/10.1007/csa48-2023-016>

Meruvulikith, M. (n.d.). 190K spam/ham email dataset for classification [Data set]. Kaggle. <https://www.kaggle.com/datasets/meruvulikith/190k-spam-ham-email-dataset-for-classification>

Nguyen, H., Tran, T., & Le, T. (2022). A machine learning-based spam detection system: Comparative performance of Naive Bayes, SVM, and neural networks. Journal of Computational Intelligence, 29(3), 145-159. https://doi.org/10.1007/jci29-2022-002

Xie, J., & Li, Z. (2022). Spam email detection using deep learning techniques: A Multilayer Perceptron approach. Artificial Intelligence Review, 42(1), 123-138. <https://doi.org/10.1007/air42-2022-001>

# Appendix

* 1. ML training and testing:  
       
     # -\*- coding: utf-8 -\*-

"""ham\_spam\_classifierwconfidencescore.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1zGEZ5X2iRInAL-v4wu7sAM3JMM2zMwCq

"""

#Python library

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

#Data preprocessing

from sklearn.preprocessing import LabelEncoder

import nltk

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('omw-1.4')

import string

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize, sent\_tokenize

import re

from collections import Counter

from wordcloud import WordCloud

from sklearn.preprocessing import LabelEncoder

# Model Building

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.metrics import accuracy\_score, confusion\_matrix,precision\_score, recall\_score, f1\_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.linear\_model import LogisticRegression

from sklearn.neural\_network import MLPClassifier

from sklearn.naive\_bayes import MultinomialNB

#uploading the data

data = pd.read\_csv('/content/emails.csv',encoding='latin-1')

data.sample(10)

data.info()

data.rename(columns={'Spam/Ham': 'result', 'Message': 'emails'}, inplace=True)

data = data.drop\_duplicates(keep='first')

data.tail()

import pandas as pd

from nltk.tokenize import word\_tokenize, sent\_tokenize

# Fill missing values in Subject and Message

data['Subject'] = data['Subject'].fillna('')

data['emails'] = data['emails'].fillna('')

# Ensure all values are strings

data['Subject'] = data['Subject'].astype(str)

data['emails'] = data['emails'].astype(str)

# Calculate lengths and token counts for the Message column

data['Length'] = data['emails'].apply(len)

data['num\_words'] = data['emails'].apply(word\_tokenize).apply(len)

data['num\_sentence'] = data['emails'].apply(sent\_tokenize).apply(len)

# Display the first 10 rows of the updated DataFrame

print(data.head(10))

data = data.drop(index=0).reset\_index(drop=True)

# Calculate lengths and token counts for the Message column

data['Length'] = data['emails'].apply(len)

data['num\_words'] = data['emails'].apply(word\_tokenize).apply(len)

data['num\_sentence'] = data['emails'].apply(sent\_tokenize).apply(len)

# Display the first 10 rows

print(data.head(10))

plt.pie(data['result'].value\_counts(), labels=['Ham', 'Spam'], autopct="%0.2f")

plt.title("Distribution of ham and spam emails")

plt.axis('equal')

plt.show()

data['result'].unique()

avg\_length\_spam = data[data['result'] == 'spam']['Length'].mean()

avg\_length\_ham = data[data['result'] == 'ham']['Length'].mean()

print("Average Length of Spam Emails:", avg\_length\_spam)

print("Average Length of Ham Emails:", avg\_length\_ham)

# Plotting the graph

plt.bar(['Spam', 'Ham'], [avg\_length\_spam, avg\_length\_ham], color=['Blue', 'green'])

plt.title('Average Length of Emails for Spam and Ham')

plt.xlabel('Email Type')

plt.ylabel('Average Length')

plt.show()

avg\_word\_spam = data[data['result'] == 'spam']['num\_words'].mean()

avg\_word\_ham = data[data['result'] == 'ham']['num\_words'].mean()

print("Average Words of Spam Emails:", avg\_word\_spam)

print("Average Words of Ham Emails:", avg\_word\_ham)

# Plotting the graph

plt.bar(['Spam', 'Ham'], [avg\_word\_spam, avg\_word\_ham], color=['Blue', 'orange'])

plt.title('Average Words of Emails for Spam and Ham')

plt.xlabel('Email Type')

plt.ylabel('Average Words')

plt.show()

avg\_sentence\_spam = data[data['result'] == 'spam']['num\_sentence'].mean()

avg\_sentence\_ham = data[data['result'] == 'ham']['num\_sentence'].mean()

print("Average Sentence of Spam Emails:", avg\_sentence\_spam)

print("Average Sentence of Ham Emails:", avg\_sentence\_ham)

# Plotting the graph

plt.bar(['Spam', 'Ham'], [avg\_sentence\_spam, avg\_sentence\_ham], color=['Blue', 'black'])

plt.title('Average Sentence of Emails for Spam and Ham')

plt.xlabel('Email Type')

plt.ylabel('Average Sentence')

plt.show()

correlation\_matrix = data[['Length', 'num\_words', 'num\_sentence']].corr()

print("The Relationship between Features are ",correlation\_matrix )

# Visualize the correlation matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.title('Correlation Matrix of Features')

plt.xlabel('Features')

plt.ylabel('Features')

plt.show()

#Lowercasing

data['transform\_text'] = data['emails'].str.lower()

#Tokenization: breaking text into individual words

data['transform\_text'] = data['transform\_text'].apply(word\_tokenize)

#Removes any characters that are not letters, numbers, or whitespace from each word in the tokenized list.

data['transform\_text'] = data['transform\_text'].apply(lambda x: [re.sub(r'[^a-zA-Z0-9\s]', '', word) for word in x])

#Creates a set of stop words (common words like "the," "and," "a") from the English language.

stop\_words = set(stopwords.words('english'))

#Removing stop words and punctuation

data['transform\_text'] = data['transform\_text'].apply(lambda x: [word for word in x if word not in stop\_words and word not in string.punctuation])

#Stemming: Reducing words to their root form (stems) helps group related words together, reducing the vocabulary size and improving model generalization.

#ps = PorterStemmer()

#data['transform\_text'] = data['transform\_text'].apply(lambda x: [ps.stem(word) for word in x])

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

data['transform\_text'] = data['transform\_text'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])

# join the preprocessed text back to string

data['transform\_text'] = data['transform\_text'].apply(lambda x: ' '.join(x))

# Display the preprocessed data

print(data[['emails', 'transform\_text']].sample(4))

wc = WordCloud(width=500,height=500,min\_font\_size=10,background\_color='white')

#word cloud for spam

spam\_wc = wc.generate(data[data['result'] == 'spam']['transform\_text'].str.cat(sep = ' '))

plt.imshow(spam\_wc)

#word cloud for ham

ham\_wc = wc.generate(data[data['result'] == 'ham']['transform\_text'].str.cat(sep = ' '))

plt.imshow(ham\_wc)

#label Encoding

encoder = LabelEncoder()

data['result'] = encoder.fit\_transform(data['result'])

data.sample(10)

# Feature extraction using TF-IDF(text into numbers)

tfidf = TfidfVectorizer(max\_features=3000)

X = tfidf.fit\_transform(data['transform\_text']).toarray()

y = data['result']

X.shape

y = data['result'].values

y

# Split 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)

# Initialize Random Forest Classifier

rf = RandomForestClassifier()

# Set up the parameter grid to search for the best parameters

param\_grid\_rf = {

'n\_estimators': [50, 100, 200], # Number of trees in the forest

'max\_depth': [None, 10, 20, 30], # Maximum depth of the tree

'min\_samples\_split': [2, 5, 10], # Minimum number of samples required to split an internal node

'min\_samples\_leaf': [1, 2, 4], # Minimum number of samples required to be at a leaf node

'max\_features': ['auto', 'sqrt', 'log2'] # Number of features to consider when looking for the best split

}

# Use GridSearchCV to search for the best parameter combination

grid\_search\_rf = GridSearchCV(estimator=rf, param\_grid=param\_grid\_rf, cv=5, scoring='accuracy')

# Fit the model to the training data

grid\_search\_rf.fit(X\_train, y\_train)

# Get the best parameters and best estimator

best\_params\_rf = grid\_search\_rf.best\_params\_

best\_rf\_model = grid\_search\_rf.best\_estimator\_

# Evaluate the best Random Forest model

y\_pred\_best\_rf = best\_rf\_model.predict(X\_test)

accuracy\_best\_rf = accuracy\_score(y\_test, y\_pred\_best\_rf)

precision\_best\_rf = precision\_score(y\_test, y\_pred\_best\_rf, average='weighted')

recall\_best\_rf = recall\_score(y\_test, y\_pred\_best\_rf, average='weighted')

f1\_score\_best\_rf = f1\_score(y\_test, y\_pred\_best\_rf, average='weighted')

# Print the best parameters and evaluation metrics

print("Best Random Forest Model Parameters:", best\_params\_rf)

print("Best Random Forest Model Accuracy:", accuracy\_best\_rf)

print("Best Random Forest Model Precision:", precision\_best\_rf)

print("Best Random Forest Model Recall:", recall\_best\_rf)

print("Best Random Forest Model F1-score:", f1\_score\_best\_rf)

# Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=180, max\_depth=None, min\_samples\_split=10, min\_samples\_leaf=1, random\_state=42)

rf\_classifier.fit(X\_train, y\_train) #training the model

#testing the model

y\_pred\_rf = rf\_classifier.predict(X\_test)

#Calculate evaluation matrices

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

precision\_rf = precision\_score(y\_test, y\_pred\_rf)

recall\_rf = recall\_score(y\_test, y\_pred\_rf)

f1\_score\_rf = f1\_score(y\_test, y\_pred\_rf)

print("Random Forest:")

print("Accuracy:", accuracy\_rf)

print("Precision:", precision\_rf)

print("Recall:", recall\_rf)

print("F1-score:", f1\_score\_rf)

print("\n")

# Confusion Matrix for Random Forest

cm\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

disp\_rf = ConfusionMatrixDisplay(confusion\_matrix=cm\_rf)

disp\_rf.plot()

plt.title('Confusion Matrix for Random Forest')

plt.show()

#Hyperparameter tuning using GridSearchCV

param\_grid\_svc = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly']}

grid\_search\_svc = GridSearchCV(SVC(), param\_grid=param\_grid\_svc, cv=5)

grid\_search\_svc.fit(X\_train, y\_train)

best\_params\_svc = grid\_search\_svc.best\_params\_

best\_svc\_model = grid\_search\_svc.best\_estimator\_

# Evaluate the best SVC model

y\_pred\_best\_svc = best\_svc\_model.predict(X\_test)

accuracy\_best\_svc = accuracy\_score(y\_test, y\_pred\_best\_svc)

precision\_best\_svc = precision\_score(y\_test, y\_pred\_best\_svc)

recall\_best\_svc = recall\_score(y\_test, y\_pred\_best\_svc)

f1\_score\_best\_svc = f1\_score(y\_test, y\_pred\_best\_svc)

print("Best SVC Model Parameters:", best\_params\_svc)

print("Best SVC Model Accuracy:", accuracy\_best\_svc)

print("Best SVC Model Precision:", precision\_best\_svc)

print("Best SVC Model Recall:", recall\_best\_svc)

print("Best SVC Model F1-score:", f1\_score\_best\_svc)

# Support Vector Classifier (SVC)

svc\_classifier = SVC(C=1.0, kernel= "linear", probability=True)

svc\_classifier.fit(X\_train, y\_train) #training the model

# testing the model

y\_pred\_svc = svc\_classifier.predict(X\_test)

#evaluation metrices

accuracy\_svc = accuracy\_score(y\_test, y\_pred\_svc)

precision\_svc = precision\_score(y\_test, y\_pred\_svc)

recall\_svc = recall\_score(y\_test, y\_pred\_svc)

f1\_score\_svc= f1\_score(y\_test, y\_pred\_svc)

print("Support Vector Classifier (SVC):")

print("Accuracy:", accuracy\_svc)

print("Precision:", precision\_svc)

print("Recall:", recall\_svc)

print("F1-score:", f1\_score\_svc)

print("\n")

# Confusion Matrix for SVC

cm\_svc = confusion\_matrix(y\_test, y\_pred\_svc)

disp\_svc = ConfusionMatrixDisplay(confusion\_matrix=cm\_svc)

disp\_svc.plot()

plt.title('Confusion Matrix for SVC')

plt.show()

# Naive Bayes Classifier

nb\_classifier = MultinomialNB(alpha=0.1, fit\_prior= True)

nb\_classifier.fit(X\_train, y\_train) #training

y\_pred\_nb = nb\_classifier.predict(X\_test) #testing

#Evaluation metrices

accuracy\_nb = accuracy\_score(y\_test, y\_pred\_nb)

precision\_nb = precision\_score(y\_test, y\_pred\_nb)

recall\_nb = recall\_score(y\_test, y\_pred\_nb)

f1\_score\_nb = f1\_score(y\_test, y\_pred\_nb)

print("Naive Bayes:")

print("Accuracy:", accuracy\_nb)

print("Precision:", precision\_nb)

print("Recall:", recall\_nb)

print("F1-score:", f1\_score\_nb)

print("\n")

cm\_nb = confusion\_matrix(y\_test, y\_pred\_nb)

disp\_nb = ConfusionMatrixDisplay(confusion\_matrix=cm\_nb)

disp\_nb.plot()

plt.title('Confusion Matrix for Naive Bayes')

plt.show()

# Neural Network (Multi-layer Perceptron)

mlp\_classifier = MLPClassifier(hidden\_layer\_sizes=(50,) ,max\_iter=19, random\_state=42)

mlp\_classifier.fit(X\_train, y\_train) #training

#testing the model

y\_pred\_mlp = mlp\_classifier.predict(X\_test)

#Evalution metrices

accuracy\_mlp = accuracy\_score(y\_test, y\_pred\_mlp)

precision\_mlp = precision\_score(y\_test, y\_pred\_mlp)

recall\_mlp = recall\_score(y\_test, y\_pred\_mlp)

f1\_score\_mlp = f1\_score(y\_test, y\_pred\_mlp)

print("Neural Network (MLP):")

print("Accuracy:", accuracy\_mlp)

print("Precision:", precision\_mlp)

print("Recall:", recall\_mlp)

print("F1-score:", f1\_score\_mlp)

# Confusion Matrix for Neural Network (MLP)

cm\_mlp = confusion\_matrix(y\_test, y\_pred\_mlp)

disp\_mlp = ConfusionMatrixDisplay(confusion\_matrix=cm\_mlp)

disp\_mlp.plot()

plt.title('Confusion Matrix for Neural Network (MLP)')

plt.show()

from sklearn.metrics import precision\_recall\_curve, auc

def plot\_precision\_recall\_curve(y\_test, y\_probs, model\_name):

# y\_probs: the predicted probabilities

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_probs)

auc\_pr = auc(recall, precision) # AUC of precision-recall

plt.figure(figsize=(3, 3))

plt.plot(recall, precision, label=f'{model\_name} (AUC = {auc\_pr:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title(f'Precision-Recall Curve for {model\_name}')

plt.legend(loc='best')

plt.grid(True)

plt.show()

# For Random Forest Classifier

y\_probs\_rf = rf\_classifier.predict\_proba(X\_test)[:, 1] # Get probabilities for the positive class

plot\_precision\_recall\_curve(y\_test, y\_probs\_rf, "Random Forest")

# For SVC Classifier

y\_probs\_svc = svc\_classifier.decision\_function(X\_test) # Use decision function instead of predict\_proba for SVC

plot\_precision\_recall\_curve(y\_test, y\_probs\_svc, "Support Vector Classifier")

# For Naive Bayes Classifier

y\_probs\_nb = nb\_classifier.predict\_proba(X\_test)[:, 1]

plot\_precision\_recall\_curve(y\_test, y\_probs\_nb, "Naive Bayes")

from sklearn.metrics import roc\_auc\_score, roc\_curve

# For Random Forest

# Calculate predicted probabilities

y\_pred\_proba\_rf = rf\_classifier.predict\_proba(X\_test)[:, 1]

roc\_auc\_rf = roc\_auc\_score(y\_test, y\_pred\_proba\_rf)

# Compute ROC curve

fpr\_rf, tpr\_rf, thresholds\_rf = roc\_curve(y\_test, y\_pred\_proba\_rf)

print("Random Forest ROC AUC:", roc\_auc\_rf)

# Plot ROC curve for Random Forest

plt.figure()

plt.plot(fpr\_rf, tpr\_rf, color='blue', label='Random Forest (area = %0.2f)' % roc\_auc\_rf)

plt.plot([0, 1], [0, 1], color='red', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) - Random Forest')

plt.legend(loc="lower right")

plt.grid()

plt.show()

# For Support Vector Classifier (SVC)

y\_pred\_proba\_svc = svc\_classifier.predict\_proba(X\_test)[:, 1]

roc\_auc\_svc = roc\_auc\_score(y\_test, y\_pred\_proba\_svc)

# Compute ROC curve

fpr\_svc, tpr\_svc, thresholds\_svc = roc\_curve(y\_test, y\_pred\_proba\_svc)

print("SVC ROC AUC:", roc\_auc\_svc)

# Plot ROC curve for SVC

plt.figure()

plt.plot(fpr\_svc, tpr\_svc, color='blue', label='SVC (area = %0.2f)' % roc\_auc\_svc)

plt.plot([0, 1], [0, 1], color='red', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) - SVC')

plt.legend(loc="lower right")

plt.grid()

plt.show()

# For Naive Bayes

y\_pred\_proba\_nb = nb\_classifier.predict\_proba(X\_test)[:, 1]

roc\_auc\_nb = roc\_auc\_score(y\_test, y\_pred\_proba\_nb)

# Compute ROC curve

fpr\_nb, tpr\_nb, thresholds\_nb = roc\_curve(y\_test, y\_pred\_proba\_nb)

print("Naive Bayes ROC AUC:", roc\_auc\_nb)

# Plot ROC curve for Naive Bayes

plt.figure()

plt.plot(fpr\_nb, tpr\_nb, color='blue', label='Naive Bayes (area = %0.2f)' % roc\_auc\_nb)

plt.plot([0, 1], [0, 1], color='red', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) - Naive Bayes')

plt.legend(loc="lower right")

plt.grid()

plt.show()

import pickle

filename = 'mlpTrained\_model.pkl'

pickle.dump(mlp\_classifier, open(filename, 'wb'))

pickle.dump(tfidf, open('mlptransform.pkl', 'wb'))

loading\_model = pickle.load(open('mlpTrained\_model.pkl', 'rb'))

tfidf = pickle.load(open('mlptransform.pkl', 'rb'))

def predict\_email(email):

# Convert email into numerical vector using the trained TF-IDF vectorizer

email\_vector = tfidf.transform([email])

# Convert sparse matrix to dense array

email\_vector\_dense = email\_vector.toarray()

# Get the probabilities for each class (ham and spam)

confidence\_scores = loading\_model.predict\_proba(email\_vector\_dense)

# Model prediction (1 for spam, 0 for ham) based on maximum probability

prediction = np.argmax(confidence\_scores[0]) # Get the class with the highest probability

# Prepare the response

result = {

"prediction": "spam" if prediction == 1 else "ham",

"confidence\_score": float(confidence\_scores[0][prediction]) # Score for predicted class

}

# Print result for the email

print(f"The email is predicted as {result['prediction']} with a confidence score of {result['confidence\_score'] \* 100:.2f}%")

# Get user input for email

user\_email = input("Enter the email text: ")

# Predict whether the input email is spam or ham

predict\_email(user\_email)

## FastAPI:

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from sklearn.neural\_network import MLPClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

import joblib

import numpy as np

# Define the request model

class EmailText(BaseModel):

email\_text: str

# Initialize FastAPI app

app = FastAPI()

# Load the pre-trained model and vectorizer

model: MLPClassifier= joblib.load('mlpTrained\_model.pkl')

vectorizer: TfidfVectorizer = joblib.load('mlptransform.pkl')

@app.post("/predict/")

async def predict(email: EmailText):

try:

# Transform the email text to vector

vectorized\_text = vectorizer.transform([email.email\_text])

# Convert sparse matrix to dense array

dense\_input = vectorized\_text.toarray()

# Get confidence scores using predict\_proba

confidence\_scores = model.predict\_proba(dense\_input)

# Model prediction (1 for spam, 0 for ham) based on maximum probability

prediction = np.argmax(confidence\_scores[0]) # Get the class with the highest probability

# Prepare the response

result = {

"prediction": "spam" if prediction == 1 else "ham",

"confidence\_score": float(confidence\_scores[0][prediction]) # Score for predicted class

}

return result

except Exception as e:

raise HTTPException(status\_code=400, detail=str(e))

# Include a root endpoint for simple health check

@app.get("/")

async def read\_root():

return {"message": "Hello, this is the spam/ham classification API"}

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)

### Procfile

web: uvicorn ml\_fastapi2:app --host=0.0.0.0 --port=${PORT:-5000}

### Requirements.txt

fastapi

uvicorn

pydantic

pickle5

scikit-learn

numpy

joblib

### Runtime.txt

python-3.9.17

## AppScirpt:

### Appscirpt.json (manifest file)

{

"timeZone": "Australia/Sydney",

"oauthScopes": [

"https://www.googleapis.com/auth/gmail.readonly",

"https://www.googleapis.com/auth/script.external\_request",

"https://www.googleapis.com/auth/gmail.addons.execute"

],

"gmail": {

"name": "Spam/Ham Email Classifier",

"logoUrl": "https://github.com/kushalbas/FastAPI\_Adds-on/blob/89a43482f6bc75781973d68781380cecd504edf5/logo.png",

"contextualTriggers": [

{

"unconditional": {},

"onTriggerFunction": "getContextualAddOn"

}

],

"primaryColor": "#4285F4",

"secondaryColor": "#34A853"

},

"urlFetchWhitelist": [

"https://spam-ham-fastapi-9cdbaf54d63e.herokuapp.com/predict/"

],

"runtimeVersion": "V8",

"exceptionLogging": "STACKDRIVER"

}

### Ham-spam-classifer.gs

// Create a card to display the classification result

function createEmailClassificationCard(subject, result, confidence) {

return CardService.newCardBuilder()

.setHeader(CardService.newCardHeader().setTitle("Email Classification"))

.addSection(CardService.newCardSection()

.addWidget(CardService.newTextParagraph().setText("Subject: " + subject))

.addWidget(CardService.newTextParagraph().setText("Classification: " + result))

.addWidget(CardService.newTextParagraph().setText("Confidence: " + confidence))

)

.build();

}

// Function to call your FastAPI service and classify the email

function classifyEmail(e) {

var messageId = e.messageMetadata.messageId;

var emailSubject = e.messageMetadata.subject;

var emailSubject = GmailApp.getMessageById(messageId).getSubject();

var apiEndpoint = "https://spam-ham-fastapi-9cdbaf54d63e.herokuapp.com/predict/";

var emailContent = GmailApp.getMessageById(messageId).getPlainBody();

try {

// Make POST request to the FastAPI endpoint

var response = UrlFetchApp.fetch(apiEndpoint, {

method: "POST",

contentType: "application/json",

payload: JSON.stringify({ "email\_text": emailContent })

});

// Parse the FastAPI response

var result = JSON.parse(response.getContentText());

var classification = result.prediction;

var confidence = result.confidence\_score;

} catch (error) {

Logger.log("Error calling the FastAPI service: " + error);

return createEmailClassificationCard(emailSubject, "Error", "failed to connect.");

}

// Create and return the card

return createEmailClassificationCard(emailSubject, classification, confidence);

}

// Main function for Gmail Add-on

function getContextualAddOn(e) {

Logger.log(JSON.stringify(e)); // Log the event object

return classifyEmail(e);

}