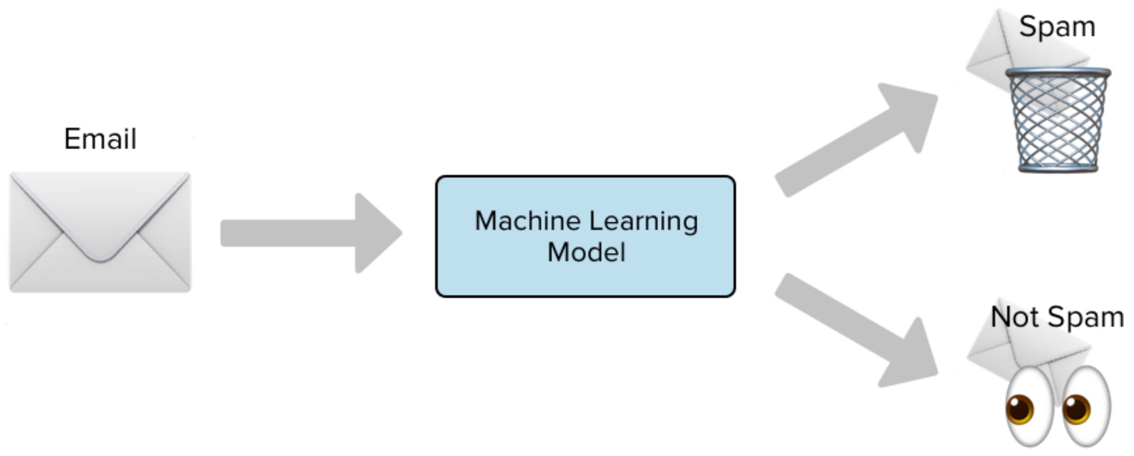
AI-Powered Spam Classifier

Introduction:

The objective of this document is to outline a project plan for building an AI-powered spam classifier for emails or text messages. The primary goal is to create a robust system that can accurately distinguish between spam and non-spam messages while minimizing false positives and false negatives. This document will detail the problem statement, the proposed approach, anticipated challenges, expected results, and a timeline for the project.



Problem Statement:

- The problem is to build an AI-powered spam classifier that can accurately distinguish between spam and non-spam messages in emails or text messages.
- The goal is to reduce the number of false positives (classifying legitimate messages as spam) and false negatives (missing actual spam messages) while achieving a high level of accuracy.

Design Thinking Process:

- **Data Collection:** We will need a dataset containing labelled examples of spam and non-spam messages. We can use a Kaggle dataset for this purpose.
- **Data Pre-processing:** The text data needs to be cleaned and pre-processed. This involves removing special characters, converting text to lowercase, and tokenizing the text into individual words.
- **Feature Extraction:** We will convert the tokenized words into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).

- **Model Selection**: We can experiment with various machine learning algorithms such as Naive Bayes, Support Vector Machines, and more advanced techniques like deep learning using neural networks.
- **Evaluation:** We will measure the model's performance using metrics like accuracy, precision, recall, and F1-score.
- **Iterative Improvement:** We will fine-tune the model and experiment with hyperparameters to improve its accuracy.

Phases of Development:

Data loading and preprocessing:

- In this phase, the program begins by loading the email dataset from a CSV file (in this case, 'spam.csv') using the pandas library. This dataset typically contains email text and labels indicating whether each email is "ham" (legitimate) or "spam" (unwanted).

Feature extraction using TF-IDF.

- The next phase involves feature extraction. In this case, a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is used. TF-IDF is a common technique for converting text data into numerical features. The TfidfVectorizer from scikit-learn is employed to transform the preprocessed email text into a TF-IDF matrix.

Model training with Multinomial Naive Bayes.

- The Multinomial Naive Bayes algorithm is suitable for text classification tasks like email classification because it can work effectively with text data and is known for its simplicity and efficiency.The model is trained using the TF-IDF features ('X_tfidf') and the corresponding labels ('y'), which represent whether each email is "ham" or "spam."

Development of a tkinter-based GUI for user interaction.

- To make the email classification accessible and user-friendly, a graphical user interface (GUI) is developed using the tkinter library.
- The GUI includes components such as labels, text entry fields, and buttons to allow users to interact with the program. Users can input a message in a text entry field.

Integration of model prediction with the GUI.

- When the user interacts with the GUI and inputs a message, the program integrates the machine learning model into the GUI. This integration includes a function ('classify_email') that preprocesses the user's input, converts it into TF-IDF features, and then uses the trained Multinomial Naive Bayes model to predict whether the input is "ham" or "spam."

Deployment of the GUI for email classification.

- After the GUI and model integration are complete, the GUI application is deployed for email classification.
- Users can use the GUI to input messages, and the program will classify them as "ham" or "spam" in real-time, providing an intuitive and practical tool for email filtering.

Data Collection:

- Collect a diverse and representative dataset of labelled email and text message data, comprising both spam and non-spam messages.
- The program loads a dataset ('spam.csv') containing email data with two columns: 'v1' (labels) and 'v2' (email text).

Dataset Link: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Import necessary libraries:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
import tkinter as tk
from tkinter import *
```

- pandas is imported to handle data manipulation and loading the dataset.
- sklearn.feature_extraction.text is imported to create a TF-IDF vectorizer.
- sklearn.naive_bayes is imported to create a Multinomial Naive Bayes classifier.
- tkinter is imported to create the graphical user interface.

Load the dataset:

```
data = pd.read_csv('spam.csv', encoding='ISO-8859-1')
```

- The code loads a dataset named 'spam.csv' using the pd.read_csv function from pandas.
- The dataset contains email data with two columns: 'v1' (label) and 'v2' (email text). It's assumed that the dataset is in CSV format and uses the ISO-8859-1 encoding.

Data preprocessing :

```
# Data preprocessing function
def preprocess_text(text):
    # Example preprocessing steps: lowercasing and removing special characters
    text = text.lower()
    text = ''.join(e for e in text if (e.isalnum() or e.isspace()))
    return text
```

- **Text Cleaning:** The 'preprocess_text' function is defined to preprocess the text data by converting it to lowercase and removing special characters,keeping only alphanumeric characters and spaces.
- This preprocessing is applied to the 'v2' column to clean the email text.
- Data is split into features (X, email text) and labels (y, 'ham' or 'spam').

```
# Apply data preprocessing to the 'v2' (email_text) column
data['v2'] = data['v2'].apply(preprocess_text)
```

The preprocess_text function is applied to the 'v2' column of the dataset, cleaning and preprocessing the email text.

Split the data:

```
# Split the data into features (X) and labels (y)
X = data['v2']  # Email text
y = data['v1']  # Labels ('ham' or 'spam')
```

The code splits the preprocessed data into features (X) and labels (y). 'X' contains the preprocessed email text, and 'y' contains the labels ('ham' or 'spam').

Feature Extraction:

```
# Create a TfidfVectorizer for feature extraction
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(X)
```

- The TF-IDF matrix is created and used for training and classification.
- A TfidfVectorizer is created to convert the preprocessed text data into TF-IDF features. It is fitted to the email text in the 'X' variable, creating a TF-IDF matrix called 'X_tfidf.'

Train the model:

```
# Train the model
model = MultinomialNB()
model.fit(X_tfidf, y)
```

- A Multinomial Naive Bayes classifier is chosen as the machine learning algorithm. This algorithm is suitable for text classification tasks, and it is efficient in handling high-dimensional data like TF-IDF matrices.
- The model is trained on the TF-IDF features and corresponding labels.
- A Multinomial Naive Bayes classifier is created and trained using the TF-IDF features ('X_tfidf') and the labels ('y').

Define a function to classify user input:

```
# Function to classify user input
def classify_email(user_input):
    user_input = preprocess_text(user_input)  # Apply preprocessing to user input
    user_input_tfidf = tfidf_vectorizer.transform([user_input])
    prediction = model.predict(user_input_tfidf)
    return prediction[0]
```

- The classify_email function takes user input, preprocesses it, and uses the TF-IDF vectorizer to transform it into TF-IDF features. It then predicts whether the input is 'ham' or 'spam' using the trained Naive Bayes model and returns the prediction.

Create a tkinter GUI window:

```
# Create a tkinter GUI window
window = tk.Tk()
window.title("Email Classifier")
```

- A tkinter GUI window is created with the title "Email Classifier."

Create GUI elements:

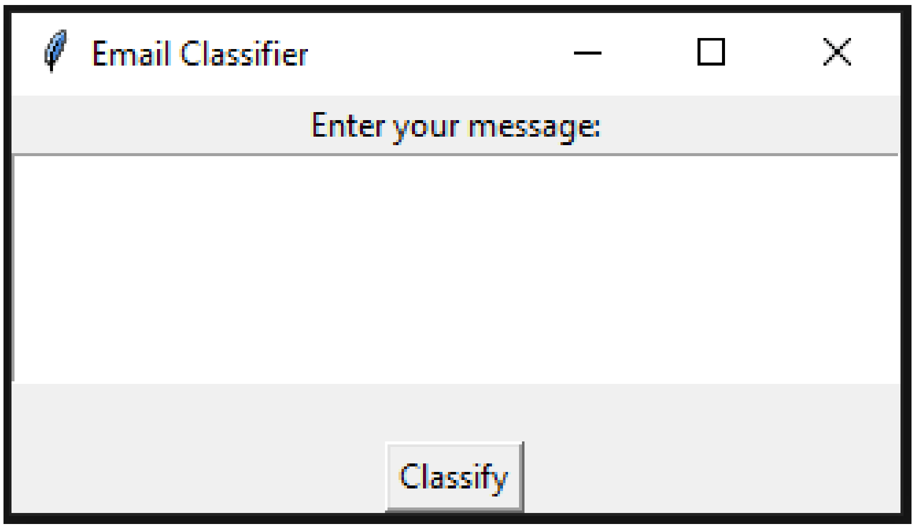
```
def classify_button_click():
    user_input = text_entry.get("1.0", "end-1c")  # Get text from the Text widget
    result = classify_email(user_input)
    result_label.config(text=f"This message is classified as '{result}'.")
    text_entry.delete(1.0, END)  # Clear the text entry for the next input
```

- Labels, a Text widget (for entering messages), and a Button labeled "Classify" are added to the GUI window.

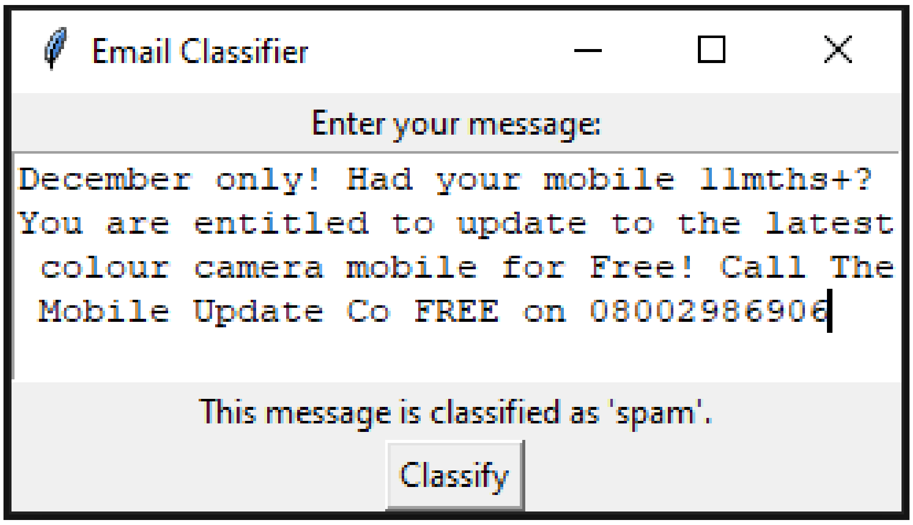
```
classify_button = tk.Button(window, text="Classify", command=classify_button_click)
classify_button.pack()

window.mainloop()
```

- The classify_button_click function is called when the "Classify" button is clicked. It retrieves the user's input from the Text widget, uses the classify_email function to predict the class of the input, and displays the result in a label. The input field is cleared for the next input.
- The window.mainloop() function is called, which starts the tkinter GUI application and allows users to interact with it.

Final output Interface:



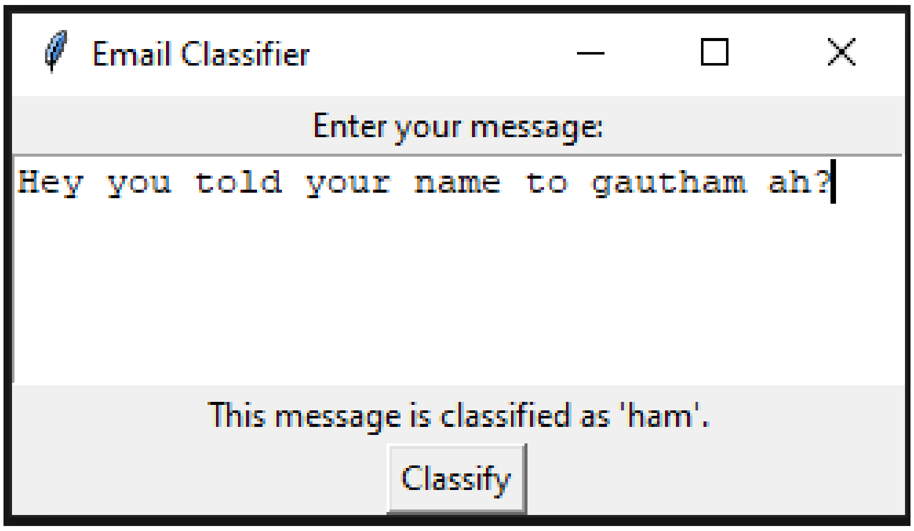
Example for spam message:



```
December only! Had your mobile 11mths+?
You are entitled to update to the latest
 colour camera mobile for Free! Call The
 Mobile Update Co FREE on 08002986906
```

This message is classified as 'spam'.

Example for ham message:



```
Hey you told your name to gautham ah?
```

This message is classified as 'ham'.

Conclusion:

- Building an AI-powered spam classifier is a complex but essential task to ensure the efficiency and security of communication channels. By following the proposed approach and continually improving the model, we aim to reduce false positives and false negatives while achieving a high level of accuracy in distinguishing between spam and non-spam messages.
- The project will involve data collection, pre-processing, feature engineering, model selection, and rigorous evaluation. The goal is to create a reliable spam filter that improves the email and messaging experience by reducing unwanted messages while ensuring legitimate ones are not mistakenly classified as spam