**Phase 1: Problem Definition and Design Thinking**

|  |  |
| --- | --- |
| PROJECT TITLE | AI SPAM CLASSIFIER |
| NAME | SAKTHI VELU S |
| TEAM ID | 5232 |
| TEAM NAME | Proj\_204227\_Team\_1 |
| COLLEGE CODE-NAME | 9238-MANGAYARKARASI COLLEGE OF ENGINEERING PARAVAI, MADURAI. |
| GROUP | 5 |
| GITHUB REPOSITORY LINK | <https://github.com/Sakthi576/IBM-NAAN-MUTHALVAN-AI.git> |

**AI SPAM CLASSIFIER**

* **PROJECT DEFINITION:**
* **PROJECT OBJECTIVES:**

Our project aims to create a highly effective AI-powered spam classifier, capable of accurately differentiating between spam and non-spam messages in emails or text messages. The primary objective is to minimize both false positives (wrongly identifying legitimate messages as spam) and false negatives (missing actual spam messages), while maintaining a high level of classification accuracy.

* **DESIGN THINKING:**

To address this problem effectively, we will follow a structured approach involving the following key steps:

* **DATA COLLECTION:**

Data collection is the first step in building our spam classifier. In this phase, we need to gather a dataset that contains examples of both spam and non-spam (ham) messages. The dataset linked from Kaggle (SMS Spam Collection Dataset) will serve as our source of labelled data. This dataset likely consists of text messages, each labelled as either spam or non-spam.



* **DATA PREPROCESSING**

Before feeding the text data into our machine learning model, we need to prepare it. Data preprocessing includes the following key steps:

**Text Cleaning:** We clean the text data by removing any special characters, HTML tags, or other noisy elements that may not contribute to the classification task. This step ensures that the text data is in a more standardized format.

**import re**

**cleaned\_text = re.sub(r"<.\*?>", "", text) # Remove HTML tags**

**Lowercasing:** Converting all the text to lowercase helps in achieving consistency. It ensures that the model treats "Spam" and "spam" as the same word, reducing ambiguity.

**lowercased\_text = text.lower()**

**Tokenization:** Tokenization is the process of splitting the text into individual words or tokens. This step breaks down the text into its smallest meaningful units, making it easier for the model to work with.

**import nltk**

**from nltk.tokenize import word\_tokenize**

**tokens = word\_tokenize(text)**

* **FEATURE EXTRACTION:**
* Machine learning algorithms typically work with numerical data, so we need to convert our text data into numerical features. To achieve this, we will use the TF-IDF (Term Frequency-Inverse Document Frequency) technique:

**TF-IDF:** TF-IDF assigns a numerical value to each word in the text based on its frequency within a specific message (Term Frequency) and its importance across the entire dataset (Inverse Document Frequency). This creates a numerical representation of each message, where words that are common in a specific message but rare in the dataset receive higher values.

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

**# Create a TfidfVectorizer object**

**tfidf\_vectorizer = TfidfVectorizer()**

**# Fit and transform the documents to TF-IDF vectors**

**tfidf\_matrix = tfidf\_vectorizer.fit\_transform(documents)**

**# Get the feature names (words) corresponding to the columns in the TF-IDF matrix**

**feature\_names = tfidf\_vectorizer.get\_feature\_names\_out()**

**# Convert the TF-IDF matrix to a dense array (optional)**

**tfidf\_matrix\_dense = tfidf\_matrix.toarray()**

**# Create a dictionary to store TF-IDF values for each document**

**tfidf\_results = {}**

* **MODEL SELECTION:**

Choosing the right machine learning algorithm is crucial for building an effective spam classifier. We explore various options:

* Naive Bayes: This probabilistic algorithm is suitable for text classification tasks. It calculates the probability of a message being spam or non-spam based on word frequencies.

**from sklearn.naive\_bayes import MultinomialNB**

**classifier = MultinomialNB()**

**classifier.fit(X\_train, y\_train)**

* Support Vector Machines (SVM): SVMs are effective for linear and nonlinear classification tasks. They aim to find a decision boundary that best separates spam from non-spam messages.

**from sklearn.svm import SVC**

**classifier = SVC(kernel='linear')**

**classifier.fit(X\_train, y\_train)**

* Deep Learning: Deep learning, particularly using neural networks, offers a more complex approach to text classification. It can automatically learn intricate patterns in the data, potentially leading to high accuracy.

**from tensorflow.keras.preprocessing.text import Tokenizer**

**from tensorflow.keras.preprocessing.sequence import pad\_sequences**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout**

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

**model = Sequential()**

**model.add(Embedding(max\_words, 128, input\_length=max\_sequence\_length))**

**model.add(LSTM(128, dropout=0.2, recurrent\_dropout=0.2))**

**model.add(Dense(1, activation='sigmoid'))**

**# Compile the model**

**model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])**

**# Train the model**

**model.fit(X\_train, y\_train, epochs=5, batch\_size=32)**

* **EVALUATION**

The performance of the developed model will undergo a rigorous evaluation using pertinent evaluation metrics, including:

* **Accuracy**: Quantifying the proportion of correctly predicted cases.

**from sklearn.metrics import accuracy\_score, classification\_report**

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

* **Precision**: Assessing the model's capability to correctly identify individuals with diabetes among those predicted to have it.

**From sklearn.metrics import precision\_score**

**precision = precision\_score(true\_labels, predicted\_labels)**

* **Recall**: Gauging the model's ability to identify all individuals with diabetes within the dataset.

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

**recall = recall\_score(true\_labels, predicted\_labels)**

* **F1-Score**: Determining the harmonic mean of precision and recall, offering a balanced assessment.

from sklearn.metrics import f1\_score

f1 = f1\_score(true\_labels, predicted\_labels)

* **Iterative Improvement**

Building a robust spam classifier may require refining and optimizing the initial model. Some strategies for iterative improvement include:

**Hyperparameter Tuning:** Experimenting with different model parameters to find the settings that optimize performance.

**Feature Engineering:** Exploring additional features or advanced text preprocessing techniques to enhance the model's understanding of the data.

**Regularization:** Implementing techniques like dropout or L2 regularization to prevent overfitting, where the model performs well on training data but poorly on new data.

**Ensemble Methods:** Combining multiple models (e.g., using ensemble techniques like bagging or boosting) to improve overall accuracy and reliability.

Portions that are well-explained In this section, we’ll talk about concentrating more on the major findings and conclusions of the research Supervised machine learning has a high acceptance rate. Throughout the review, the approach can be noticed. This strategy is effective. is employed primarily because it produces more accurate findings. With less fluctuation, this strategy has a high level of consistency. Aside from that, we’ve discovered that certain algorithms work better than others. When compared to other techniques, such as Nave Based and SVM, there is a strong demand for them.