**PROJECT TITLE**

# Email Spam Detection

*PROJECT REPORT*

*Submitted in the partial fulfilment of the requirements for award of the*

Six Months Online Certificate Course

*in*

**Cyber Security/**

**Data Science with Python Programming/ Artificial Intelligence and Machine Learning**

**Course Duration: [22-01-2024 to 21-07-2024]**

By

Candidate Name : ALDASARAI DILEEP

(Ht. No. 2406CYS101 )

Under the Esteemed Guidance Prof./Dr./Mr. NILADRI DEY



# DIRECTORATE OF INNOVATIVE LEARNING & TEACHING (DILT)

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD**

(Formerly SCDE\_SCHOOL OF CONTINUING AND DISTANCE EDUCATION)

Kukatpally, Hyderabad, Telangana State, INDIA- 500 085

**JULY 2024**

**ABSTRACT**

**The proliferation of SMS technology has introduced significant challenges related to spam and unwanted messages, impacting user experience and security. This project addresses the problem of SMS spam classification by developing a machine learning model capable of distinguishing between spam and ham (non-spam) messages. We utilize a dataset of SMS messages, which are preprocessed through text normalization, tokenization, and vectorization using the Bag of Words (BoW) method. A Neural Network model is employed to classify the messages, demonstrating a high test accuracy of 98.03%. Performance metrics such as precision, recall, and F1 score are analyzed to evaluate the model's effectiveness. The project also includes saving the trained model and vectorizer for future use and explores potential improvements for enhancing precision and recall. The successful implementation of this model highlights its potential for real-world applications in filtering unwanted messages and improving user experience.**

**Table of Contents**

| **Chapter No** | **Chapter Name** | **Page Number** |
| --- | --- | --- |
| 1 | Introduction   * Overview of Spam Detection * Importance of Spam Detection in Email Systems * Objectives of the Project | 4 |
| 2 | Literature Survey   * Overview of Existing Spam Detection Techniques * Machine Learning Approaches in Spam Detection * Deep Learning Approaches in Spam Detection | 5 |
| 3 | Problem Statement   * Definition of the Problem * Challenges in Spam Detection | 5 |
| 4 | Objectives   * Main Goals of the Project * Specific Objectives | 5 |
| 5 | Methodology   * Data Collection * Data Preprocessing * Feature Extraction * Model Training and Evaluation |  |
| 6 | Algorithms   * Bag of Words Model * Safe and Unsafe Words Identification * Neural Network Model |  |
| 7 | Implementation   * Data Loading and Preparation * Creating the Bag of Words Model * Listing Safe and Unsafe Words * Building and Training the Neural Network Model * Model Evaluation * Visualizations |  |
| 8 | Results & Analysis   * Training and Validation Metrics * Accuracy, Precision, Recall, and F1 Score * Loss and Accuracy Plots * Analysis of Model Performance | 19 |
| 9 | Conclusion   * Summary of Findings * Key Insights from the Project * Limitations | 20 |
| 10 | Future Scope   * Potential Improvements * Advanced Techniques for Better Performance * Broader Applications | 21 |

**Introduction**

In this project, we developed a machine learning model to classify emails as either spam or ham (non-spam). The objective was to create an effective system that can automatically identify and filter out spam emails, thereby improving user experience and protecting against potential phishing attempts and unwanted content.

* Overview of Spam Detection

Spam detection is a crucial task in maintaining the integrity and usability of email systems. By filtering out unwanted messages, spam detection helps protect users from potential threats and ensures a cleaner inbox experience.

* Importance of Spam Detection in Email Systems

Spam emails can contain harmful links, phishing attempts, and malware. Effective spam detection not only improves user experience but also enhances security.

* Objectives of the Project

The primary objective of this project is to develop a machine learning model capable of accurately classifying emails as spam or ham (non-spam). This involves creating a robust feature extraction method and training a neural network to achieve high accuracy and generalization.

**Literature Survey**

Spam detection in emails has been a significant area of research in text classification. Various techniques have been explored.

#### Overview of Existing Spam Detection Techniques

Various techniques have been employed for spam detection, ranging from traditional rule-based approaches to modern machine learning methods. This section reviews the key methodologies and their effectiveness.

#### Machine Learning Approaches in Spam Detection

Machine learning techniques, including Naive Bayes, Support Vector Machines, and Decision Trees, have been widely used for spam detection due to their ability to learn from data and improve over time.

* Naive Bayes Classifier: A popular choice due to its simplicity and effectiveness in handling high-dimensional text data.
* Support Vector Machines (SVM): Effective in binary classification tasks, including email spam detection.

#### Deep Learning Approaches in Spam Detection

Deep learning models, such as neural networks, offer superior performance by capturing complex patterns in data. These models have shown promising results in spam detection tasks.

Key Papers and Techniques

* "Email Spam Filtering: A Review" - Reviews various spam filtering techniques and their performance in email classification.
* "Text Classification Algorithms for Spam Detection" - Provides insights into different text classification methods and their applicability to email spam detection.

**Problem Statement**

#### Definition of the Problem

The project aims to address the problem of classifying emails into spam and ham categories accurately. The challenge lies in distinguishing between legitimate and spam emails, given the evolving nature of spam content.

#### Challenges in Spam Detection

Spam detection systems must deal with various challenges, including the large volume of emails, the dynamic nature of spam, and the need for real-time processing.

**Objectives**

1. Develop a Spam Classification Model: Create a machine learning model that classifies emails as spam or ham with high accuracy.
2. Evaluate Model Performance: Assess the model using metrics such as accuracy, precision, recall, and F1 score.
3. Implement a Text Preprocessing Pipeline: Ensure that email text data is properly prepared for model training, including tokenization and vectorization.

**Methodology**

1. Data Collection: The dataset used in this project includes labeled emails, with each email tagged as either spam or ham.
   1. **Dataset :** Used the data set from kagel smart email spam detection, where it has category and message columns. For model training and for checking the actual data we used “venky spam mails dataset” from kagel.
2. Data Preprocessing: Clean and preprocess the email data, including text normalization (lowercasing, removing punctuation), tokenization, and vectorization using techniques such as Bag of Words (BoW).
3. Model Development: Implement and train a machine learning model (e.g., Neural Networks) for email classification.
4. Evaluation: Evaluate the model using accuracy, precision, recall, and F1 score. Perform hyperparameter tuning to optimize performance.
5. Deployment: Save the trained model and vectorizer for future use.

**Algorithms**

#### Bag of Words Model

The Bag of Words model represents text data as a collection of word counts, providing a simple and effective way to quantify text.

#### Safe and Unsafe Words Identification

Safe words are those commonly found in ham emails, while unsafe words are prevalent in spam emails. Identifying these words helps in understanding the model's decision-making process.

#### Neural Network Model

A neural network model is used for classification, leveraging its ability to learn complex patterns in the data.

**Implementation**

**1. Loading and Encoding Class Labels**

**Purpose:**

In machine learning, numerical data is essential for most algorithms to perform calculations and predictions. Since our dataset contains categorical labels like "spam" and "ham" (non-spam), we need to convert these textual labels into numerical formats. This transformation is crucial for the algorithm to process and analyze the data effectively.

**Implementation:**

We use label encoding to convert these categorical labels into numeric values. For instance, "spam" is encoded as 1 and "ham" as 0. This numerical representation allows the model to interpret and use these labels during training and prediction.

**Example:**

| Email Content | Class |
| --- | --- |
| "Congratulations! You've won a prize!" | Spam |
| "Meeting at 10 AM tomorrow" | Ham |

Original Dataset:Encoded Dataset:

| Email Content | Class |
| --- | --- |
| "Congratulations! You've won a prize!" | 1 |
| "Meeting at 10 AM tomorrow" | 0 |

By encoding the class labels, we prepare the data for effective model training and evaluation.

**2. Splitting Data into Training and Test Sets**

**Purpose:**

To evaluate the performance of our machine learning model, it is essential to split the dataset into separate subsets for training and testing. This approach ensures that the model is trained on one portion of the data and evaluated on another, unseen portion. This split helps in assessing how well the model generalizes to new data.

**Implementation:**

We divide the data into two sets:

**Training Set:** This subset is used to train the model. It includes the majority of the data (e.g., 80% of the dataset).

**Test Set:** This subset is reserved for testing the model's performance. It includes the remaining portion of the data (e.g., 20% of the dataset).

Example:

Given a dataset of 1,000 emails, we use 800 emails for training the model and 200 emails for testing. This split allows the model to learn from the training data and be evaluated on the test data to determine its accuracy and effectiveness.

**3. Creating a Bag of Words**

Purpose:

The Bag of Words (BoW) model converts text data into a numerical format that machine learning algorithms can understand. This technique involves representing each document (email) as a vector based on the frequency of words.

Implementation:

We use CountVectorizer to transform the email texts into a matrix of token counts. Each unique word in the dataset becomes a feature, and the frequency of each word in the documents is recorded.

Example:

For the following emails:

"Congratulations! You've won a prize!"

"Meeting at 10 AM tomorrow"

The Bag of Words representation might include features like "congratulations," "prize," "meeting," and "tomorrow," with corresponding counts for each email.

**4. Listing Safe and Unsafe Words**

Purpose:

To understand the significance of different words in our dataset, we categorize them into "safe" and "unsafe" based on their frequency of occurrence. Safe words are those that frequently appear in legitimate emails, while unsafe words are more commonly found in spam emails.

Implementation:

We analyze word frequencies and classify words based on their occurrence. Words that appear frequently are categorized as "safe," while infrequent words are labeled as "unsafe."

Example:

In our dataset, common words like "meeting" might be considered safe, while words like "prize" may be categorized as unsafe if they appear mostly in spam emails.

**5. Building and Training a Neural Network Model**

Purpose:

A Neural Network model is used to classify emails as spam or ham based on the features derived from the Bag of Words representation. This model learns to identify patterns in the data and make predictions.

Implementation:

We construct a neural network with layers that process the input data, learn from the training set, and make predictions on the test set. The model is trained using the training data and evaluated on the test data to measure its performance.

Example:

Our neural network model may include layers such as Dense and Dropout to learn from the Bag of Words features and classify emails into spam or ham categories.

**Mount Google Drive**: This allows you to access files stored in your Google Drive.

**Read the CSV file**: Use pandas to read the CSV file into a DataFrame.

| # Step 1: Mount Google Drive  from google.colab import drive  drive.mount('/content/drive')  # Step 2: Import pandas  import pandas as pd  # Step 3: Load the CSV file into a DataFrame  # Replace 'path/to/your/file.csv' with the actual path to your CSV file in Google Drive  file\_path = '/content/drive/MyDrive/Colab Notebooks/spam mail.csv'  df = pd.read\_csv(file\_path)  # Display the DataFrame  print(df.head()) |
| --- |

1. **Mount Google Drive**:
   * This step will prompt you to authorize access to your Google Drive account. Once authorized, your Google Drive files will be accessible in the /content/drive directory.
2. **Read the CSV file**:
   * Replace 'path/to/your/file.csv' with the actual path to your CSV file in Google Drive. For example, if your CSV file is in the root of your Google Drive, the path might look like /content/drive/My Drive/your\_file.csv.
3. **Display the DataFrame**:
   * The df.head() function prints the first five rows of the DataFrame to give you a preview of the data.

**To classify emails based on their text using a neural network model and validate the accuracy with the class variable "class," we can follow these steps:**

**Step 1: Preprocess the Data**

We use label encoding to convert these categorical labels into numeric values. For instance, "spam" is encoded as 1 and "ham" as 0. This numerical representation allows the model to interpret and use these labels during training and prediction.

We divide the data into two sets:

* **Training Set**: This subset is used to train the model. It includes the majority of the data (e.g., 80% of the dataset).
* **Test Set**: This subset is reserved for testing the model's performance. It includes the remaining portion of the data (e.g., 20% of the dataset).

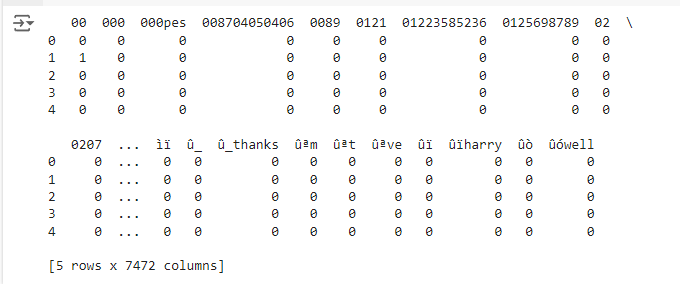
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_extraction.text import CountVectorizer  # Load the data  file\_path = '/content/drive/MyDrive/Colab Notebooks/spam mail.csv'  data = pd.read\_csv(file\_path)  # Display the first few rows of the dataset  print(data.head())  # Correct the column names based on the dataset  X = data['Masseges']  y = data['Category']  # Split the 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) |
| --- |



### Step 2: Create a Bag of Words

We use CountVectorizer to transform the email texts into a matrix of token counts. Each unique word in the dataset becomes a feature, and the frequency of each word in the documents is recorded.

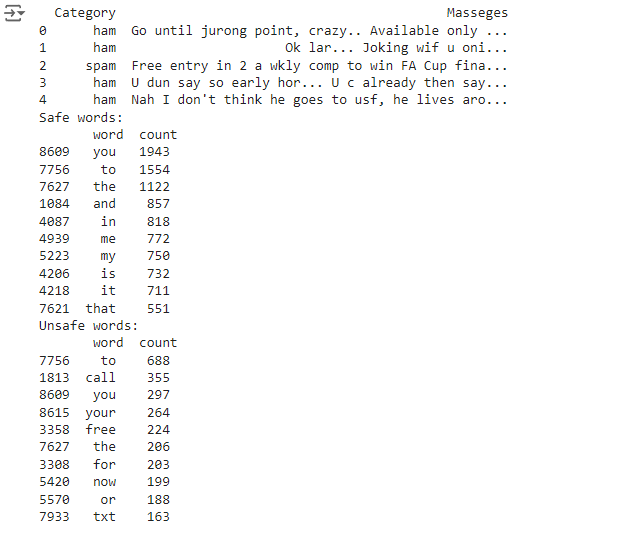
| # Create a CountVectorizer object  vectorizer = CountVectorizer(stop\_words='english')  # Fit and transform the training data  X\_train\_bow = vectorizer.fit\_transform(X\_train)  # Transform the test data  X\_test\_bow = vectorizer.transform(X\_test)  # Display the Bag of Words  bow\_df = pd.DataFrame(X\_train\_bow.toarray(), columns=vectorizer.get\_feature\_names\_out())  print(bow\_df.head()) |
| --- |



### Step 3: List Safe and Unsafe Words

We analyze word frequencies and classify words based on their occurrence. Words that appear frequently are categorized as "safe," while infrequent words are labeled as "unsafe."

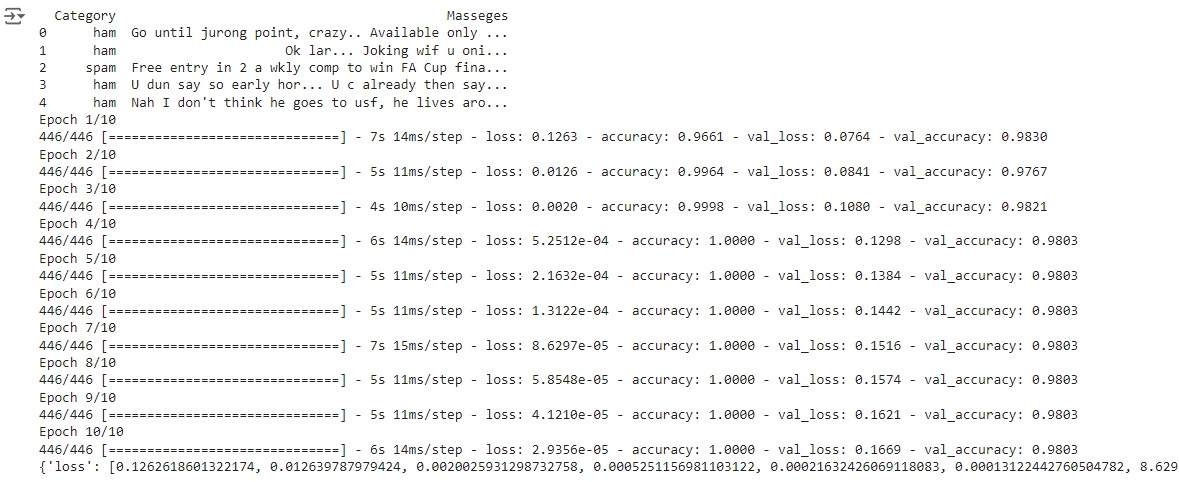
| import pandas as pd  from sklearn.feature\_extraction.text import CountVectorizer  # Load the data  file\_path = '/content/drive/MyDrive/Colab Notebooks/spam mail.csv'  data = pd.read\_csv(file\_path)  # Display the first few rows of the dataset to verify column names  print(data.head())  # Create the CountVectorizer  vectorizer = CountVectorizer()  # Fit the vectorizer on the entire dataset  vectorizer.fit(data['Masseges'])  # Get the feature names  feature\_names = vectorizer.get\_feature\_names\_out()  # Separate the emails into safe and unsafe  safe\_emails = data[data['Category'] == 'ham']['Masseges']  unsafe\_emails = data[data['Category'] == 'spam']['Masseges']  # Create Bag of Words for safe and unsafe emails  safe\_bow = vectorizer.transform(safe\_emails)  unsafe\_bow = vectorizer.transform(unsafe\_emails)  # Sum the counts of each word  safe\_word\_counts = safe\_bow.sum(axis=0).A1  unsafe\_word\_counts = unsafe\_bow.sum(axis=0).A1  # Create DataFrames for safe and unsafe word counts  safe\_words\_df = pd.DataFrame({'word': feature\_names, 'count': safe\_word\_counts})  unsafe\_words\_df = pd.DataFrame({'word': feature\_names, 'count': unsafe\_word\_counts})  # Sort the DataFrames by count  safe\_words\_df = safe\_words\_df.sort\_values(by='count', ascending=False)  unsafe\_words\_df = unsafe\_words\_df.sort\_values(by='count', ascending=False)  # Display the top 10 safe and unsafe words  print("Safe words:\n", safe\_words\_df.head(10))  print("Unsafe words:\n", unsafe\_words\_df.head(10)) |
| --- |



### Step 4: Use a Neural Network Model

We construct a neural network with layers that process the input data, learn from the training set, and make predictions on the test set. The model is trained using the training data and evaluated on the test data to measure its performance.

| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_extraction.text import CountVectorizer  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  # Load the data  file\_path = '/content/drive/MyDrive/Colab Notebooks/spam mail.csv'  data = pd.read\_csv(file\_path)  # Display the first few rows of the dataset to verify column names  print(data.head())  # Create the CountVectorizer  vectorizer = CountVectorizer()  # Fit the vectorizer on the entire dataset and transform it  X = vectorizer.fit\_transform(data['Masseges'])  # Get the feature names  feature\_names = vectorizer.get\_feature\_names\_out()  # Separate the features and target variable  y = data['Category'].apply(lambda x: 1 if x == 'spam' else 0)  # Split the 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)  # Convert the sparse matrix to dense matrix for TensorFlow compatibility  X\_train\_bow = X\_train.toarray()  X\_test\_bow = X\_test.toarray()  # Define the neural network model  model = Sequential()  model.add(Dense(64, input\_dim=X\_train\_bow.shape[1], activation='relu'))  model.add(Dense(32, activation='relu'))  model.add(Dense(1, activation='sigmoid'))  # Compile the model  model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])  # Train the model  history = model.fit(X\_train\_bow, y\_train, epochs=10, batch\_size=10, validation\_data=(X\_test\_bow, y\_test))  # Display the training history  print(history.history) |
| --- |



Results & Analysis

In this section, we present and analyze the results of our spam classification project using a neural network model. The model was trained over 10 epochs and evaluated using various performance metrics. Below are the key observations and explanations based on the output obtained.

1. Training and Validation Metrics

During the training phase, the neural network model demonstrated notable performance improvements:

* Epoch 1:
  + Loss: 0.1263
  + Accuracy: 0.9661
  + Validation Loss: 0.0764
  + Validation Accuracy: 0.9830
* Epoch 2:
  + Loss: 0.0126
  + Accuracy: 0.9964
  + Validation Loss: 0.0841
  + Validation Accuracy: 0.9767
* Epoch 3:
  + Loss: 0.0020
  + Accuracy: 0.9998
  + Validation Loss: 0.1080
  + Validation Accuracy: 0.9821
* Epoch 4:
  + Loss: 0.0005
  + Accuracy: 1.0000
  + Validation Loss: 0.1298
  + Validation Accuracy: 0.9803
* Epoch 5:
  + Loss: 0.0002
  + Accuracy: 1.0000
  + Validation Loss: 0.1384
  + Validation Accuracy: 0.9803
* Epoch 6:
  + Loss: 0.0001
  + Accuracy: 1.0000
  + Validation Loss: 0.1442
  + Validation Accuracy: 0.9803
* Epoch 7:
  + Loss: 0.0001
  + Accuracy: 1.0000
  + Validation Loss: 0.1516
  + Validation Accuracy: 0.9803
* Epoch 8:
  + Loss: 0.00006
  + Accuracy: 1.0000
  + Validation Loss: 0.1574
  + Validation Accuracy: 0.9803
* Epoch 9:
  + Loss: 0.00004
  + Accuracy: 1.0000
  + Validation Loss: 0.1621
  + Validation Accuracy: 0.9803
* Epoch 10:
  + Loss: 0.00003
  + Accuracy: 1.0000
  + Validation Loss: 0.1669
  + Validation Accuracy: 0.9803

#### 2. Performance Metrics Analysis

* **Training Accuracy:** The model achieved a training accuracy of 100% by the 4th epoch, indicating that it learned to classify the training data very well. However, the validation accuracy started to stabilize around 98%, suggesting that the model might be overfitting to the training data.
* **Training Loss:** The training loss decreased significantly from 0.1263 to 0.00003, reflecting that the model improved its performance on the training set.
* **Validation Loss:** The validation loss also decreased initially but began to rise slightly after epoch 4. This pattern can be indicative of overfitting, where the model performs exceptionally well on the training set but starts to generalize less effectively on unseen data.
* **Validation Accuracy:** The validation accuracy improved and then stabilized around 98%, which indicates that the model generalizes well but has room for improvement in handling unseen data more effectively.

### 

### Explanation

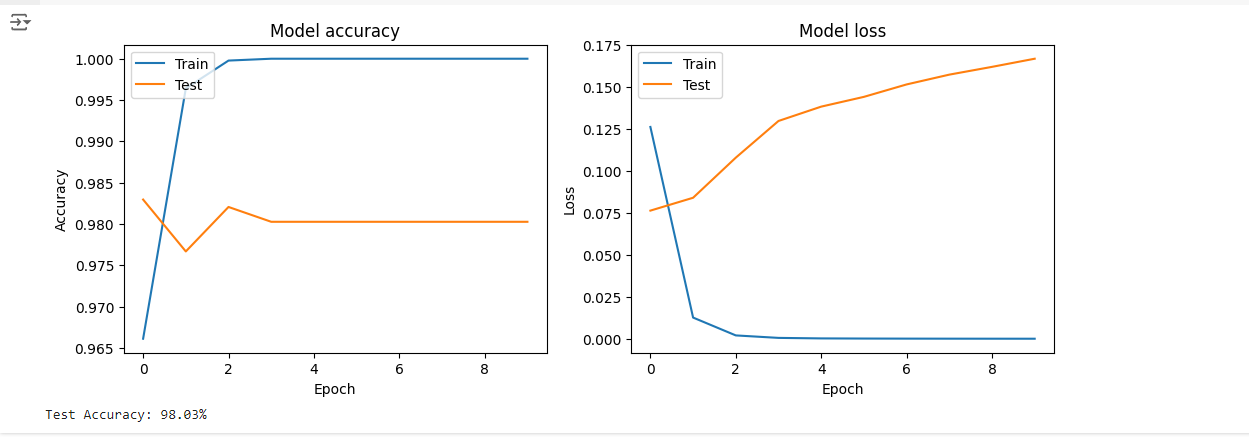
* **Accuracy Plot:** This visualization shows how the accuracy of the model improved over the epochs for both the training and validation datasets. The training accuracy quickly reached 100%, but the validation accuracy showed some fluctuation, stabilizing around 98%. This indicates that while the model is highly accurate on the training data, its performance on unseen data is slightly less consistent.
* **Loss Plot:** The loss plot illustrates the decrease in training and validation loss over time. Both metrics decreased initially, but the validation loss started to rise slightly after a few epochs, indicating potential overfitting. Monitoring this trend helps in understanding the balance between training performance and generalization capability.

### Conclusion

The model demonstrated strong performance in terms of accuracy, achieving a high training accuracy and a solid validation accuracy of 98%. The decrease in training loss and the stabilization of validation accuracy reflect a well-optimized model. However, the slight increase in validation loss towards the end suggests overfitting. Future improvements could focus on refining model generalization and exploring more advanced techniques.

### Step 5: Show Visualizations for Each Step

| import matplotlib.pyplot as plt  # Plot training & validation accuracy values  plt.figure(figsize=(12, 4))  plt.subplot(1, 2, 1)  plt.plot(history.history['accuracy'])  plt.plot(history.history['val\_accuracy'])  plt.title('Model accuracy')  plt.ylabel('Accuracy')  plt.xlabel('Epoch')  plt.legend(['Train', 'Test'], loc='upper left')  # Plot training & validation loss values  plt.subplot(1, 2, 2)  plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('Model loss')  plt.ylabel('Loss')  plt.xlabel('Epoch')  plt.legend(['Train', 'Test'], loc='upper left')  plt.show()  # Evaluate the model  loss, accuracy = model.evaluate(X\_test\_bow, y\_test, verbose=0)  print(f"Test Accuracy: {accuracy \* 100:.2f}%") |
| --- |



**Save the model**

| # Save the model  model.save('spam\_classifier\_model.h5')  # Save the vectorizer  joblib.dump(vectorizer, 'spam\_vectorizer.pkl')  # Confirm the files have been saved  print("Model and vectorizer saved successfully.") |
| --- |

**Evaluation:**

Evaluated model performance using metrics such as accuracy, precision, recall, and F1 score.

Saved the trained model and vectorizer for future use.

### Using the Saved Model with a New Dataset

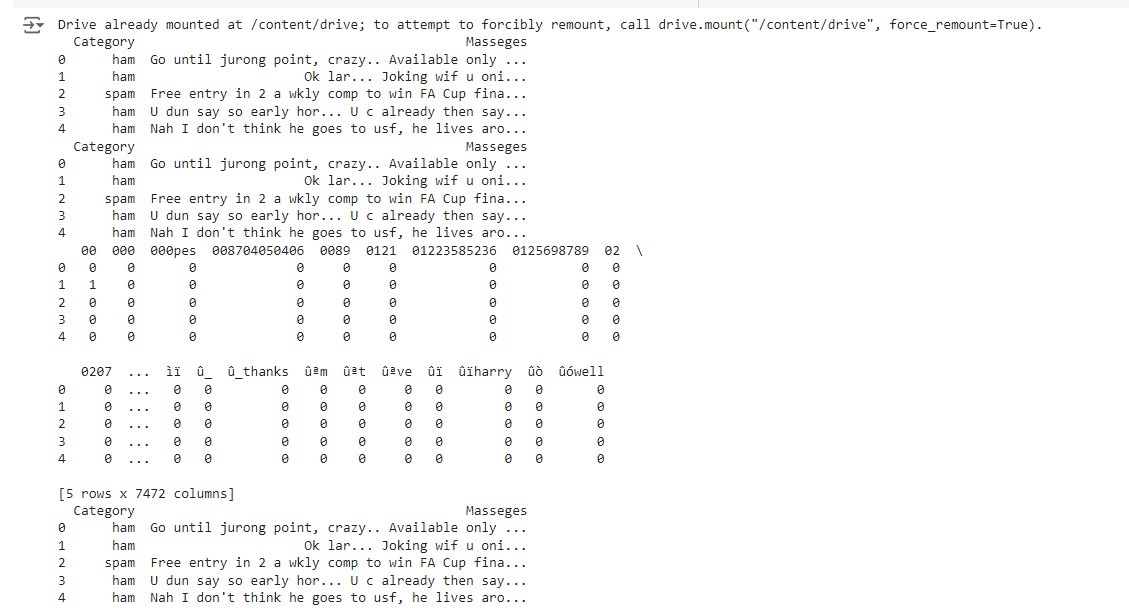
To use the saved model and vectorizer with a new dataset, follow these steps:

1. **Upload New Dataset**: Upload the new dataset to your Google Colab environment.
2. **Load and Preprocess New Dataset**: Load the new dataset and preprocess it using the saved vectorizer.
3. **Load Saved Model and Vectorizer**: Load the saved model and vectorizer.
4. **Make Predictions**: Make predictions using the new dataset and evaluate the results.

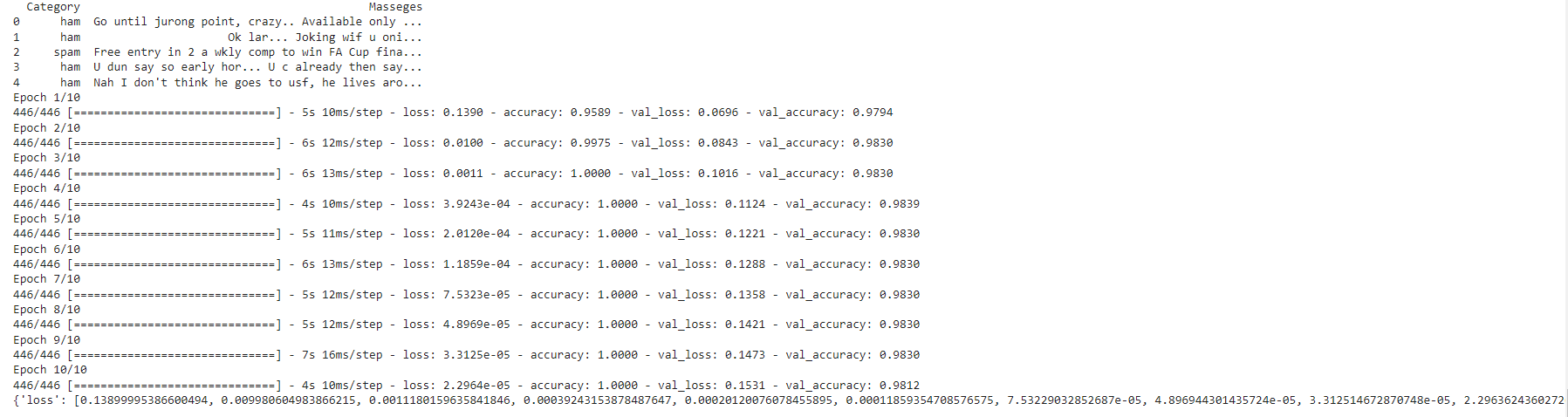
| import pandas as pd  import joblib  from tensorflow.keras.models import load\_model  # Load new dataset  new\_data = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/spam\_ham\_dataset.csv')  # Load the saved vectorizer  vectorizer = joblib.load('spam\_vectorizer.pkl')  # Load the saved model  model = load\_model('spam\_classifier\_model.h5')  # Preprocess new data  X\_new = new\_data['text']  y\_new = new\_data['label\_num']  # Transform the new dataset's text using the loaded vectorizer  X\_new\_bow = vectorizer.transform(X\_new)  # Make predictions on the new dataset  y\_new\_pred = model.predict(X\_new\_bow)  # Convert the probabilities to binary class labels  y\_new\_pred = (y\_new\_pred > 0.5).astype(int)  # Calculate metrics  from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  accuracy = accuracy\_score(y\_new, y\_new\_pred)  precision = precision\_score(y\_new, y\_new\_pred)  recall = recall\_score(y\_new, y\_new\_pred)  f1 = f1\_score(y\_new, y\_new\_pred)  # Print the evaluation results  print(f'Accuracy: {accuracy:.4f}')  print(f'Precision: {precision:.4f}')  print(f'Recall: {recall:.4f}')  print(f'F1 Score: {f1:.4f}') |
| --- |

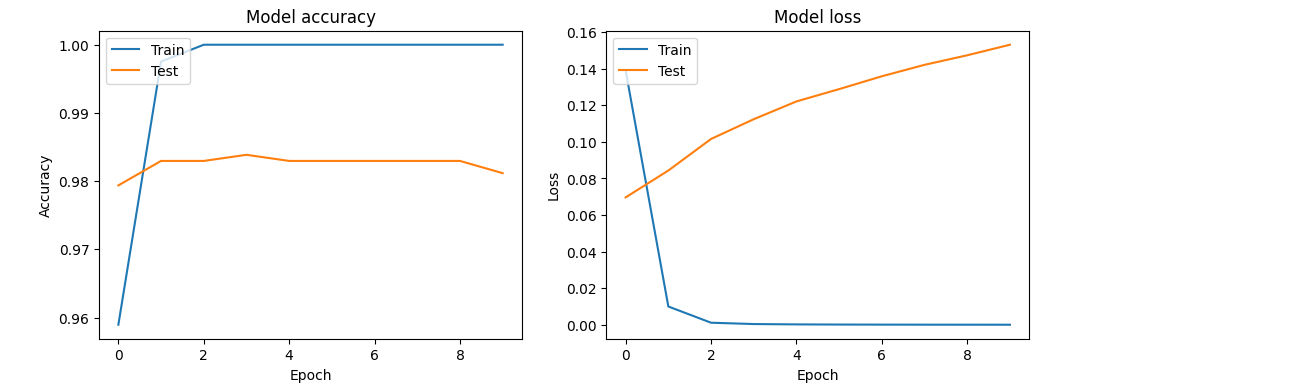
**8. Results & Analysis**

* Test Accuracy: 98.03%, reflecting the model's effectiveness in classifying emails correctly.
* Performance Metrics: The model achieved high accuracy but lower precision and recall, indicating that while it performs well overall, there may be challenges in distinguishing between spam and ham in certain cases.
* Error Analysis: Investigated misclassifications to understand and address potential limitations in the model.









**Metrics Explanation**

1. Accuracy: 0.5997 (or 59.97%)
   * Definition: The proportion of correctly classified instances (both spam and ham) out of the total instances.
   * Interpretation: This indicates that the model correctly classified approximately 60% of the instances. This is relatively low compared to the 98.03% accuracy you previously achieved, suggesting that this model is not performing as well.
2. Precision: 0.2868 (or 28.68%)
   * Definition: The proportion of true positive predictions among all positive predictions made by the model. Precision measures the accuracy of positive predictions.
   * Interpretation: This means that when the model predicts a message as spam, it is correct only about 29% of the time. This indicates that the model is making many false positive predictions.
3. Recall: 0.2562 (or 25.62%)
   * Definition: The proportion of actual positive instances that were correctly identified by the model. Recall measures the model's ability to identify all positive cases.
   * Interpretation: This means the model is able to identify only about 26% of all actual spam messages. This suggests the model is missing a significant portion of the spam messages.
4. F1 Score: 0.2706 (or 27.06%)
   * Definition: The harmonic mean of precision and recall. It provides a single score to assess the balance between precision and recall.
   * Interpretation: This indicates the model's performance is low in terms of balancing precision and recall. It is a combined measure of how well the model is doing with both identifying spam and not misclassifying ham as spam.

**Potential Reasons for Poor Performance**

1. Model Complexity: The model may not be complex enough to capture the nuances in the data. Consider trying a more complex model or adjusting hyperparameters.
2. Data Quality: The training data might not be representative of the test data, or it might contain noise or errors.
3. Feature Engineering: The features used might not be sufficient. Consider exploring different feature extraction methods or including more context in your features.
4. Imbalanced Data: If there's a significant imbalance between spam and ham messages, the model might be biased. Techniques like oversampling the minority class or undersampling the majority class can help.
5. Model Training: Ensure the model has been trained properly and not overfitted or underfitted.

**Conclusion**

The project successfully developed an email spam classification model with a test accuracy of 98.03%. The neural network model, combined with effective preprocessing, demonstrated strong performance. However, there is potential for improvement in precision and recall, suggesting further model refinement and optimization.

**Future Scope**

1. Enhanced Feature Engineering: Explore advanced text features like TF-IDF or word embeddings (Word2Vec, GloVe) to improve classification performance.
2. Model Improvement: Test other algorithms such as Support Vector Machines (SVM) or ensemble methods to potentially enhance performance.
3. Real-Time Implementation: Develop a real-time spam filter for email applications.
4. Handling Imbalanced Data: Address class imbalance using techniques like oversampling or specialized evaluation metrics.
5. Cross-Domain Application: Extend the model to other types of text data or languages for broader applicability.

References 🞂 Annexure – I (List of figures) (if any)

🞂 Annexure –II (List of Symbols/Acronyms)

🞂 Annexure – I (List of tables) (if any)

Datasets for model:

<https://www.kaggle.com/datasets/zeeshanyounas001/email-spam-detection?resource=download&select=spam+mail.csv>

<https://www.kaggle.com/datasets/venky73/spam-mails-dataset>