

Phase 5

PROJECT DOCUMENTATION & SUBMISSION

Date	31-10-2023
Team ID	1025
Project Name	6112-Building a Smarter AI-Powered Spam Classifier

Project Title: AI-Powered Spam Classifier

Problem Statement:

Objective: Develop an AI-Powered spam classifier using natural language processing (NLP) and Machine learning techniques to accurately distinguish between spam and non-spam messages in email or text messages.

Problem identified:

Spam messages can clutter communication channels, create security risks (e.g., phishing attacks), and lead to a poor user experience. A robust spam classifier can help filter out unwanted content and improve the quality of communication.

Introduction:

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

Our communication channels are inundated with a deluge of unwanted and unsolicited messages, commonly known as "spam." These spam messages disrupt our online experience, introduce security risks, and hinder our ability to efficiently sift through essential communication. To combat this ever-growing issue, we present the "AI-Powered Spam Classifier" project.

Our project is dedicated to developing a robust and intelligent solution that can autonomously differentiate between spam and non-spam (ham) messages. By harnessing the power of artificial intelligence and machine learning, we aim to create a sophisticated classifier capable of accurately categorizing incoming messages, ensuring that legitimate messages reach their intended recipients while promptly identifying and quarantining spam.

The significance of this project lies in its potential to enhance the quality of digital communication across various platforms, including emails, messaging apps, online forums, and comment sections. With the AI-Powered Spam Classifier, we aim to provide users with a powerful tool that can adapt and evolve to meet the ever-changing landscape of spam messages, thereby maintaining a clean, secure, and efficient communication environment.

Our journey through the phases of ideation, design, development, and model training will lead us to a final solution that empowers users and organizations to regain control over their communication channels, minimize the risk of falling victim to phishing and fraud, and improve the overall digital experience.

In this project, we will share our strategies, methodologies, and insights as we embark on the mission to create a smarter, more efficient spam classifier that leverages the capabilities of artificial intelligence to make digital communication safer and more enjoyable.

Data: The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to being ham (legitimate) or spam.

LITERATURE SURVEY

1. “Detection of Social Network Spam Based on Improved Extreme Learning Machine” , Zhijie Zhang

With the rapid advancement of the online social network, social media like Twitter has been increasingly critical to real life and become the prime objective of spammers. Twitter spam detection refers to a complex task for the involvement of a range of characteristics, and spam and non-spam have caused unbalanced data distribution in Twitter. To solve the mentioned problems, Twitter spam characteristics are analyzed as the user attribute, content, activity and relationship in this study, and a novel spam detection algorithm is designed based on regularized extreme learning machine, called the Improved Incremental Fuzzy-kernel-regularized Extreme Learning Machine (I2FELM), which is used to detect the Twitter spam accurately. As revealed from the experience validation results, the proposed I2FELM can efficiently identify the balanced and unbalanced dataset. Moreover, with few characteristics taken, the I2FELM can more effectively detect spam, which proves the effectiveness of the algorithm.

2. “A Spam Transformer Model for SMS Spam Detection ”, Haoye Lu

In this paper, we aim to explore the possibility of the Transformer model in detecting the spam Short Message Service (SMS) messages by proposing a modified Transformer model that is designed for detecting SMS spam messages. The evaluation of our proposed spam Transformer is performed on SMS Spam Collection v.1 dataset and UtkMI’s Twitter Spam Detection Competition dataset, with the benchmark of multiple established machine learning classifiers and state-of-the-art SMS spam detection approaches. In comparison to all other candidates, our experiments on SMS spam detection show that the proposed modified spam Transformer has the optimal results on the accuracy, recall, and F1-Score with the values of 98.92%, 0.9451, and 0.9613, respectively. Besides, the proposed model also achieves good performance on the UtkMI’s Twitter

dataset, which indicates a promising possibility of adapting the model to other similar problems.

3. “A Comprehensive Survey for Intelligent Spam Email Detection” , Asif Karim

The tremendously growing problem of phishing e-mail, also known as spam including spear phishing or spam borne malware, has demanded a need for reliable intelligent anti-spam e-mail filters. This survey paper describes a focused literature survey of Artificial Intelligence (AI) and Machine Learning (ML) methods for intelligent spam email detection, which we believe can help in developing appropriate countermeasures. In this paper, we considered 4 parts in the email's structure that can be used for intelligent analysis: (A) Headers Provide Routing Information, contain mail transfer agents (MTA) that provide information like email and IP address of each sender and recipient of where the email originated and what stopovers, and final destination. (B) The SMTP Envelope, containing mail exchangers' identification, originating source and destination domains\users. (C) First part of SMTP Data, containing information like from, to, date, subject – appearing in most email clients (D) Second part of SMTP Data, containing email body including text content, and attachment. Based on the number the relevance of an emerging intelligent method, papers representing each method were identified, read, and summarized. Insightful findings, challenges and research problems are disclosed in this paper. This comprehensive survey paves the way for future research endeavors addressing theoretical and empirical aspects related to intelligent spam email detection

4. “ A YouTube Spam Comments Detection Scheme Using Cascaded Ensemble Machine Learning Model ”.

This paper proposes a technique to detect spam comments on YouTube, which have recently seen tremendous growth. YouTube is running its own spam blocking system but continues to fail to block them properly. Therefore, we examined related studies on YouTube spam comment screening and conducted classification experiments with six different machine learning techniques (Decision tree, Logistic regression, Bernoulli Naïve Bayes, Random Forest, Support vector machine with linear kernel, Support vector machine with Gaussian kernel) and two ensemble models (Ensemble with hard voting,

Ensemble with soft voting) combining these techniques in the comment data from popular music videos - Psy, Katy Perry, LMFAO, Eminem and Shakira

5. “Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms “ , Biju Issac

Electronic mail has eased communication methods for many organisations as well as individuals. This method is exploited for fraudulent gain by spammers through sending unsolicited emails. This article aims to present a method for detection of spam emails with machine learning algorithms that are optimized with bio-inspired methods. A literature review is carried to explore the efficient methods applied on different datasets to achieve good results. An extensive research was done to implement machine learning models using Naïve Bayes, Support Vector Machine, Random Forest, Decision Tree and Multi-Layer Perceptron on seven different email datasets, along with feature extraction and pre-processing. The bio-inspired algorithms like Particle Swarm Optimization and Genetic Algorithm were implemented to optimize the performance of classifiers. Multinomial Naïve Bayes with Genetic Algorithm performed the best overall. The comparison of our results with other machine learning and bio-inspired models to show the best suitable model is also discussed.

DESIGN THINKING

Design Thinking Approach:

Empathize:

- Understand the needs and pain points of users who receive spam emails.
- Conduct user interviews, surveys, and analyze user feedback.
- Create user personas to represent different types of users and their unique challenges related to spam

Evaluation:

We will measure the models performance using metrics like accuracy, precision, recall and f1-score.

Iterative Improvements:

We will fine-tune the model and experiment with hyper parameters to improve its accuracy. **Prototype** Create a prototype of the machine learning model and the user interface for separating spam and non-spam messages.

Actions:

- Develop a Jupyter Notebook or Python script for data pre-processing, model training, and evaluation.
- Create a simple web interface using tools like Flask or Django to allow users to separate spam and non-spam messages.
- Test the prototype with a subset of the dataset to ensure it meets performance objectives.

Test:

Evaluate the model's performance using appropriate dataset and gather feedback from users.

Actions:

- Split the dataset into training and testing sets.
- Train the model on the training set and evaluate it on the testing set.
- Collect user feedback on the web interface for usability and accuracy.

Implement:

Once the prototype meets the defined objectives and receives positive feedback, proceed with full implementation.

Actions:

- Train the final machine learning model on the entire dataset.
- Deploy the model as part of a production-ready web application.
- Conduct thorough testing to ensure the application is robust and user-friendly.

Iterate:

Continuous improvement is essential. Gather user feedback and iterate on the model and interface to enhance accuracy and usability.

Actions:

- Monitor the model's performance and retrain it periodically with updated data.
- Address user feedback and make necessary improvements to the web interface.

PHASES OF DEVELOPMENT

1.Idea Generation and Problem Identification:

- In this initial phase, you define the problem you aim to solve, which is classifying spam and non-spam messages in your case.
- Conduct research and ideation to outline the project's goals and objectives.

2.Design and Strategy Formulation:

- Design the architecture and system components of your spam classifier.
- Formulate innovation strategies for distinguishing spam from legitimate messages.

3.Data Collection and Preparation:

- Collect a dataset of spam and non-spam messages from various sources.
- Preprocess the data, including cleaning, tokenization, and label encoding.

4.Feature Engineering:

Extract relevant features from the text data, which may include TF-IDF, word embeddings, or other text representation methods.

5.Model Selection and Development:

- Choose a machine learning algorithm for text classification (e.g., Naive Bayes, SVM, deep learning).
- Develop and train the machine learning model using the preprocessed data.

6.Model Evaluation and Validation:

- Assess the model's performance using metrics like accuracy, precision, recall, and F1-score.
- Employ techniques such as cross-validation to validate the model's generalization capabilities.

7.Deployment:

- Deploy the trained model as an API or web service that can classify messages in real-time.
- Develop a user interface for interaction with the spam classifier.

8.Scalability and Infrastructure:

- Ensure the system can scale to handle a large number of incoming messages.
- Use cloud services for scalability, reliability, and cost-effectiveness.

9.Security and Privacy:

Implement robust security measures to protect user data, especially when processing real messages.

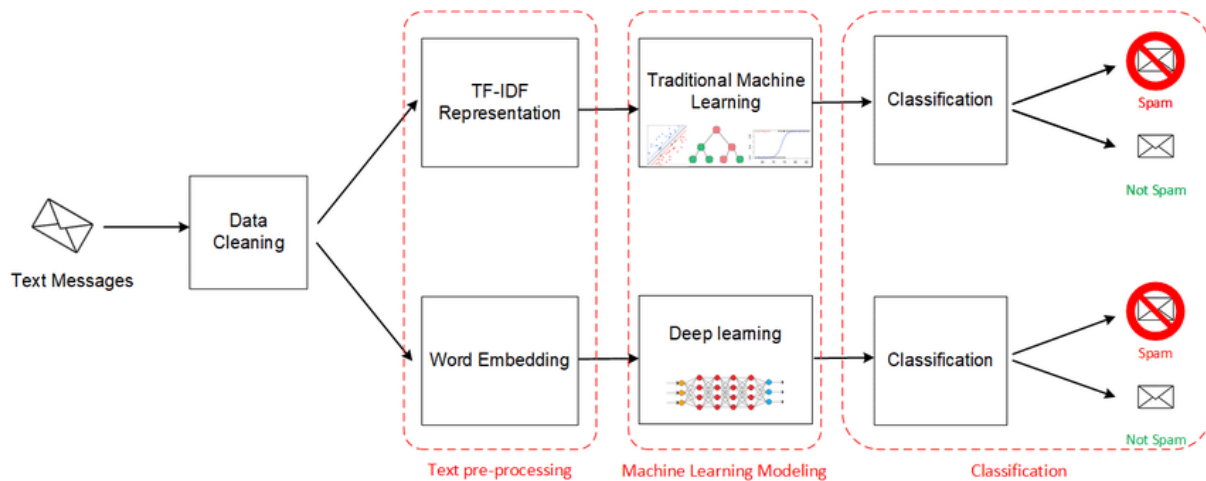
10.Continuous Monitoring and Maintenance:

- Continuously monitor the performance of the classifier.
- Implement regular updates to the model as spam patterns change.
- Address bug fixes, system improvements, and user feedback.

11.Documentation:

- Create comprehensive documentation for users, administrators, and developers.
- Explain how to use, maintain, and troubleshoot the system.

TECHNOLOGY ARCHITECTURE



1. Data Collection and Storage:

- **Data Sources:** Collect messages from various sources, such as emails, text messages, or other communication channels.
- **Data Storage:** Store the collected data in a secure and scalable database system, which can be traditional relational databases, NoSQL databases, or cloud-based data storage solutions.

2. Data Preprocessing:

- **Text Preprocessing:** Use natural language processing (NLP) techniques to clean, tokenize, and transform the text data into a format suitable for machine learning.
- **Label Encoding:** Convert categorical labels (e.g., "spam" and "ham") into numerical format.

3. Feature Extraction:

Vectorization: Transform the processed text data into numerical feature vectors using methods like TF-IDF or word embeddings (e.g., Word2Vec).

4. Machine Learning Model:

Model Selection: Choose an appropriate machine learning algorithm for text classification. Common choices include Naive Bayes, Support Vector Machines (SVM), or deep learning models like Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs).

Training: Train the selected model using the preprocessed and vectorized data.

5. Model Evaluation:

- **Performance Metrics:** Evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC curves.
- **Cross-Validation:** Perform cross-validation to assess how well the model generalizes to new data.

6. Deployment:

- **API or Web Service:** Deploy the trained model as an API or web service that can receive messages and return spam/ham classifications in real-time.
- **User Interface:** Create a user-friendly interface for users to interact with the spam classifier.

7. Infrastructure:

- **Cloud Services:** Utilize cloud platforms (e.g., AWS, Azure, GCP) for scalability, reliability, and ease of deployment. This includes cloud-based databases, virtual machines, and serverless computing for API deployment.
- **Compute Resources:** Depending on the scale of the project, provision adequate computing resources for model training and inference.

8. Security:

Implement robust security measures to protect data privacy, especially when processing real messages, which might contain sensitive information.

9. Continuous Monitoring and Maintenance:

- **Continuously monitor** the performance of the classifier and update the model as needed to adapt to changing spam patterns.
- **Address bug fixes, improvements, and model retraining** based on user feedback.

10. Documentation:

Provide comprehensive documentation for users, administrators, and developers, explaining how to use and maintain the system.

Dataset used

A dataset for an AI-powered spam classifier typically consists of labeled messages, with "ham" messages representing legitimate or non-spam content and "spam" messages representing unsolicited or unwanted content.

1.Message Content:

The dataset contains text-based messages. These messages can be in the form of emails, text messages, or other text communication types. The content of these messages varies and can include a wide range of topics and languages.

2.Labeling:

Each message in the dataset is labeled as either "ham" (non-spam) or "spam" (unsolicited or unwanted) .Labeling is essential for training and evaluating the spam classifier, as it provides the ground truth for the algorithm to learn from.

3.Data Distribution:

The dataset should have a balanced or representative distribution of "ham" and "spam" messages. Having an imbalanced dataset can affect the model's performance and may require additional techniques like oversampling or undersampling.

4.Variety of Spam:

The "spam" category should encompass various types of spam, including but not limited to:

- Email spam promoting products or services.
- Phishing attempts that aim to steal personal information.
- Malware distribution attempts.
- Fake news, scams, or fraudulent messages.
- Unwanted marketing or advertising messages.

5.Text Preprocessing:

The dataset may undergo text preprocessing to make it suitable for machine learning. This preprocessing can include:

- Tokenization: Splitting text into individual words or tokens.
- Stop-word Removal: Eliminating common words (e.g., "the," "and") that may not provide significant information.
- Lowercasing: Converting text to lowercase to ensure consistency.
- Lemmatization or Stemming: Reducing words to their base forms.
- Handling Special Characters: Dealing with punctuation, symbols, and other non-textual characters.

Data Preprocessing

Data preprocessing is a crucial step in preparing a dataset for training a machine learning model, particularly for tasks like spam classification. It involves a series of operations that clean, transform, and structure the data to make it suitable for analysis and model training.

Data Cleaning:

- Removing Duplicates: Check for and eliminate any duplicate messages to ensure that the dataset is not skewed by repetition.
- Handling Missing Data: Address any missing values in the dataset. This is important to prevent issues during analysis or model training.

Text Preprocessing:

- Tokenization: Split the text data into individual words or tokens.
Tokenization makes it easier to analyze and work with text data.
- Lowercasing: Convert all text to lowercase to ensure consistency in the text data. This prevents the model from treating "Spam" and "spam" differently.
- Stop-word Removal: Eliminate common words that are often irrelevant for classification, such as "the," "and," "in," etc.

- **Lemmatization or Stemming:** Reduce words to their base form. For example, "running," "ran," and "runs" might all be reduced to "run." This reduces the dimensionality of the data.
- **Handling Special Characters and Symbols:** Address punctuation, symbols, or special characters as needed. You can remove or replace them, depending on their relevance to the task.

Label Encoding:

- Convert categorical labels (e.g., "ham" and "spam") into numerical format. This step is essential for machine learning algorithms that require numeric input.

Data Splitting:

Split the dataset into at least two subsets: a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate its performance.

Feature Extraction Techniques

1. Bag of Words (BoW):

- BoW represents text as a collection of words, ignoring word order and structure.
- Each word is treated as a separate feature, and the presence or absence of words in a document is used as features.
- It's a simple and effective technique but doesn't consider word order or semantics.

2. Term Frequency-Inverse Document Frequency (TF-IDF):

- TF-IDF measures the importance of a word within a document relative to a collection of documents.
- It takes into account the frequency of a word within a document (TF) and its rarity across all documents (IDF).
- Common words that appear in many documents get lower TF-IDF scores, while unique or important words get higher scores.

3.Word Embeddings (Word2Vec, GloVe, FastText):

- Word embeddings are dense vector representations of words that capture semantic relationships between words.
- Word2Vec, GloVe, and FastText are popular algorithms that generate word embeddings.
- They allow the model to capture meaning and context from words.

4.N-grams:

- N-grams capture sequences of 'n' consecutive words in a text.
- For instance, bigrams (2-grams) would consider pairs of consecutive words.
- N-grams help capture some word order and context information.

5.Word Frequency Features:

- Besides TF-IDF, you can create features based on word frequencies, such as the count of specific words or characters in a document.
- For spam classification, certain words or characters may be more common in spam messages.

Machine Learning Algorithm

- **Machine Learning Algorithm:**

For a spam classifier project, you've mentioned that you've used the Naive Bayes classifier, which is a suitable choice. However, the choice of algorithm can depend on various factors

- **Naive Bayes:**

Naive Bayes is particularly well-suited for text classification tasks like spam detection because it's simple, efficient, and works well with high-dimensional data. It assumes that features (words) are conditionally independent, which often holds reasonably well for text data.

- **Other Algorithms:**

Depending on the complexity of your data and the project's requirements, you might also consider other algorithms such as Support Vector Machines (SVM), decision trees, random forests, or more advanced models like deep learning approaches (e.g., convolutional neural networks or recurrent neural networks).

- **Ensemble Methods:**

Ensemble methods like random forests or gradient boosting can be used to combine multiple classifiers for improved accuracy and robustness.

The choice of algorithm depends on the dataset, the nature of spam messages, computational resources, and the trade-off between model complexity and performance.

Model Training

Model training involves the process of teaching the selected machine learning algorithm to make accurate predictions. Here are the steps involved in model training:

Data Preparation:

Preprocess the dataset, which includes cleaning, text tokenization, feature extraction (e.g., TF-IDF or word embeddings), and label encoding.

Data Splitting:

Divide the dataset into a training set and a testing set. The training set is used to teach the model, while the testing set is used to evaluate its performance.

Algorithm-Specific Training:

For Naive Bayes, the training primarily involves calculating the probabilities and statistics required for classification based on the data's features and labels.

Hyperparameter Tuning:

Experiment with different hyperparameters (e.g., smoothing parameters for Naive Bayes) to find the optimal configuration.

Cross-Validation:

Consider using cross-validation techniques to assess how well the model generalizes to new data and to detect overfitting.

Evaluation Metrics

The choice of evaluation metrics is essential for assessing the model's performance. Common metrics for spam classification tasks include

Accuracy:

It measures the proportion of correctly classified messages. However, it may not be the best metric for imbalanced datasets where one class (e.g., "ham") is dominant.

Precision:

Precision calculates the ratio of true positives to the total number of predicted positives. It is useful when you want to minimize false positives.

Recall (Sensitivity):

Recall measures the ratio of true positives to the total number of actual positives. It's useful when you want to minimize false negatives.

F1-Score:

The F1-score is the harmonic mean of precision and recall. It balances precision and recall, making it a good choice for imbalanced datasets.

Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC):

These metrics are useful for binary classification problems. They measure the trade-off between true positive rate (sensitivity) and false positive rate at various classification thresholds.

Confusion Matrix:

A confusion matrix provides detailed information about true positives, true negatives, false positives, and false negatives.

The choice of evaluation metric depends on the specific project goals. For spam classification, minimizing false positives (non-spam messages classified as spam) is often more important than optimizing for accuracy.

Innovative Techniques

In the development of an AI-powered spam classifier, there are several innovative techniques and approaches that can be employed to enhance the system's effectiveness and adaptability.

Word Embeddings and Word Vectorization:

Instead of traditional TF-IDF representations, consider using word embeddings such as Word2Vec, GloVe, or FastText. These techniques capture semantic relationships between words and can improve the model's understanding of context in text data.

Deep Learning Architectures:

While Naive Bayes is a suitable choice for text classification, exploring deep learning architectures like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) can be innovative. These models can capture more complex patterns and relationships in text data.

Transfer Learning:

Apply transfer learning from pre-trained language models such as BERT, GPT-3, or similar models. Fine-tuning these models on your specific spam classification task can lead to significant improvements.

Feature Engineering:

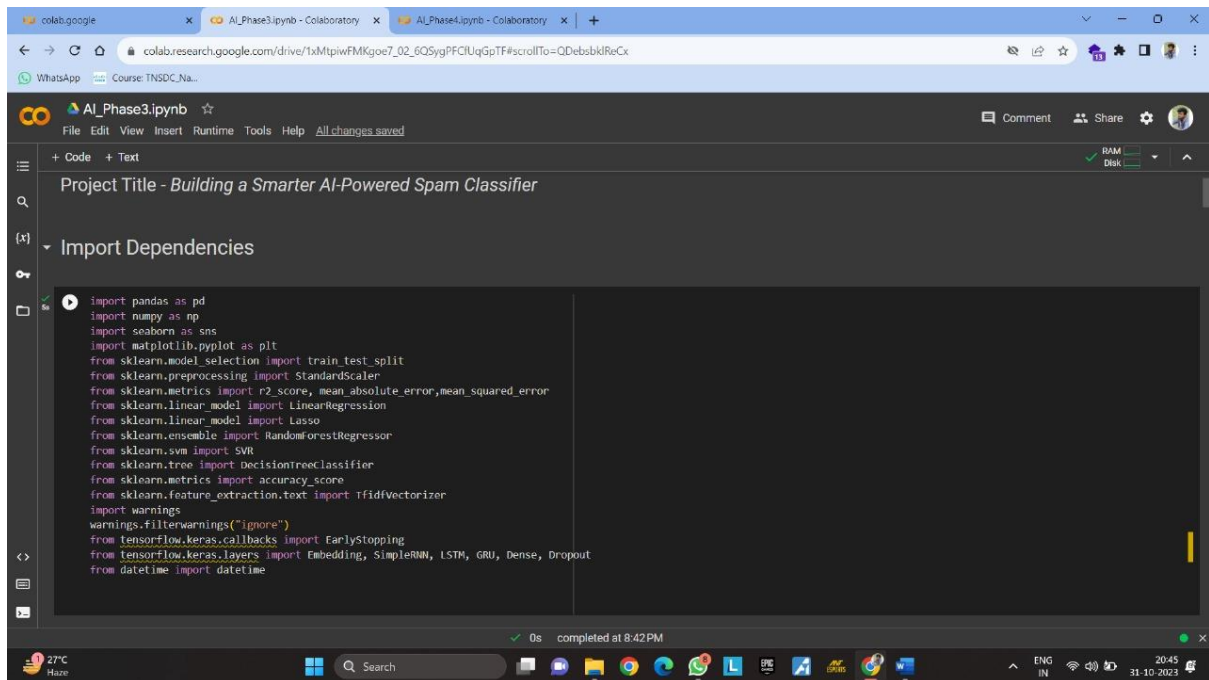
Create innovative features by considering different aspects of text data, such as sentiment analysis, readability scores, or behavioral patterns of users. These additional features can improve the model's performance.

Active Learning:

Implement an active learning approach that allows the model to select which data samples to label during training. This can be particularly helpful in reducing annotation costs and improving model performance with fewer labelled examples.

PROJECT DEVELOPMENT STEPS AND SCREENSHOTS

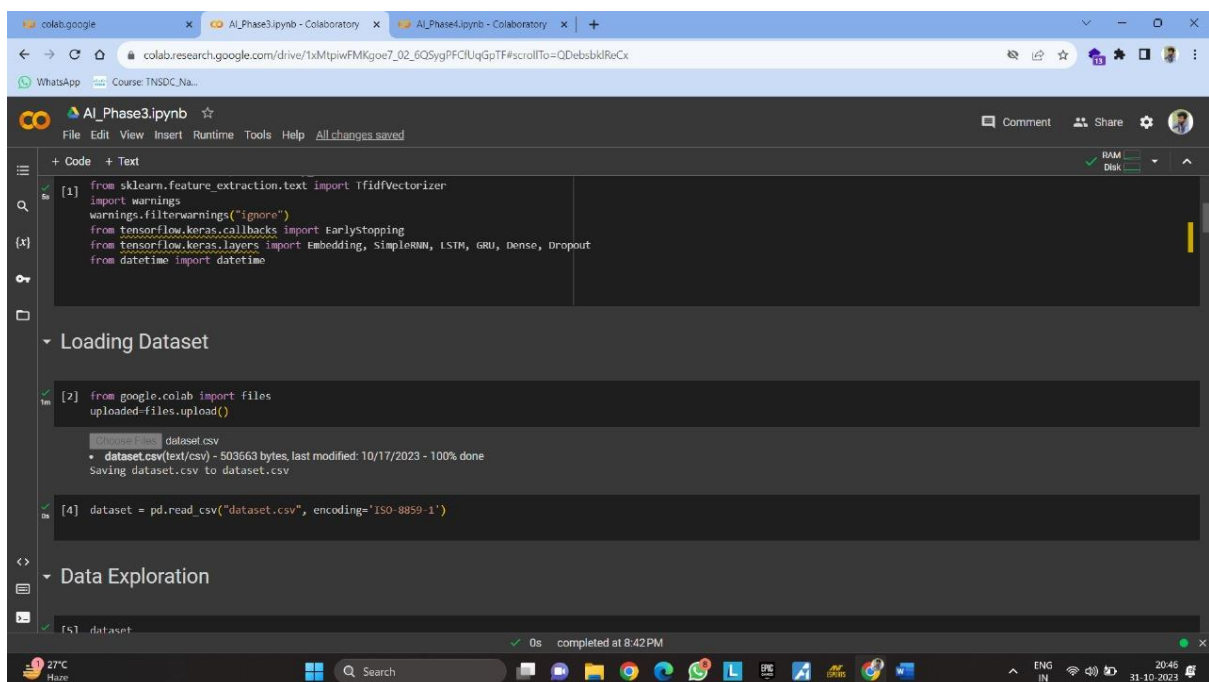
Step 1: Import the dependencies and loading the dataset:



The screenshot shows a Google Colab notebook titled "AI_Phase3.ipynb". The project title is "Building a Smarter AI-Powered Spam Classifier". The "Import Dependencies" section is expanded, showing a code cell with the following imports:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
import warnings
warnings.filterwarnings("ignore")
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense, Dropout
from datetime import datetime
```

The code cell is executed successfully, as indicated by the green checkmark and the status "completed at 8:42 PM".



The screenshot shows the same Google Colab notebook, now showing the "Loading Dataset" and "Data Exploration" sections.

The "Loading Dataset" section shows a code cell with the following code:

```
from google.colab import files
uploaded_files = files.upload()

dataset_csv = dataset.csv
dataset_csv(text/csv) - 503663 bytes, last modified: 10/17/2023 - 100% done
Saving dataset.csv to dataset.csv

dataset = pd.read_csv("dataset.csv", encoding='ISO-8859-1')
```

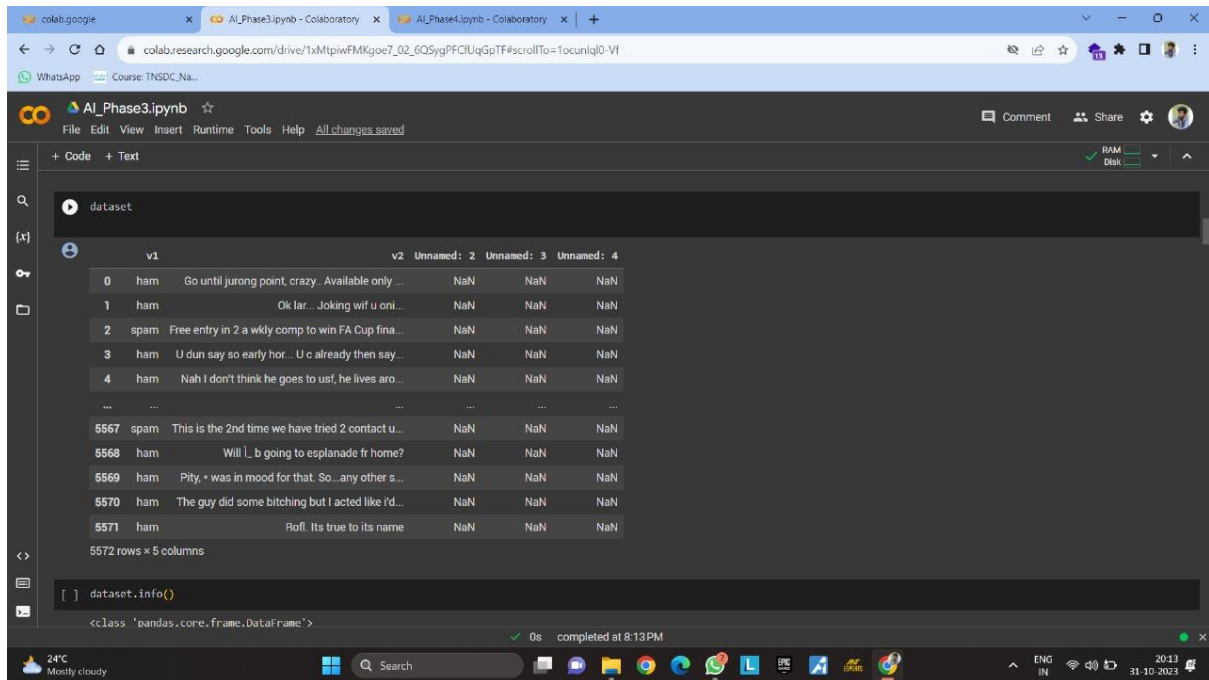
The code cell is executed successfully, as indicated by the green checkmark and the status "completed at 8:42 PM".

The "Data Exploration" section shows a code cell with the following code:

```
dataset
```

The code cell is executed successfully, as indicated by the green checkmark and the status "completed at 8:42 PM".

Step 2: Data Exploration and describe the Dataset:



The screenshot shows a Google Colab notebook titled "AI_Phase3.ipynb". The code cell contains the following text:

```
dataset
```

The output displays a preview of the dataset with 5 columns: v1, v2, Unnamed: 2, Unnamed: 3, and Unnamed: 4. The first 5 rows are shown, followed by an ellipsis and then rows 5567 to 5571. The dataset has 5572 rows and 5 columns.

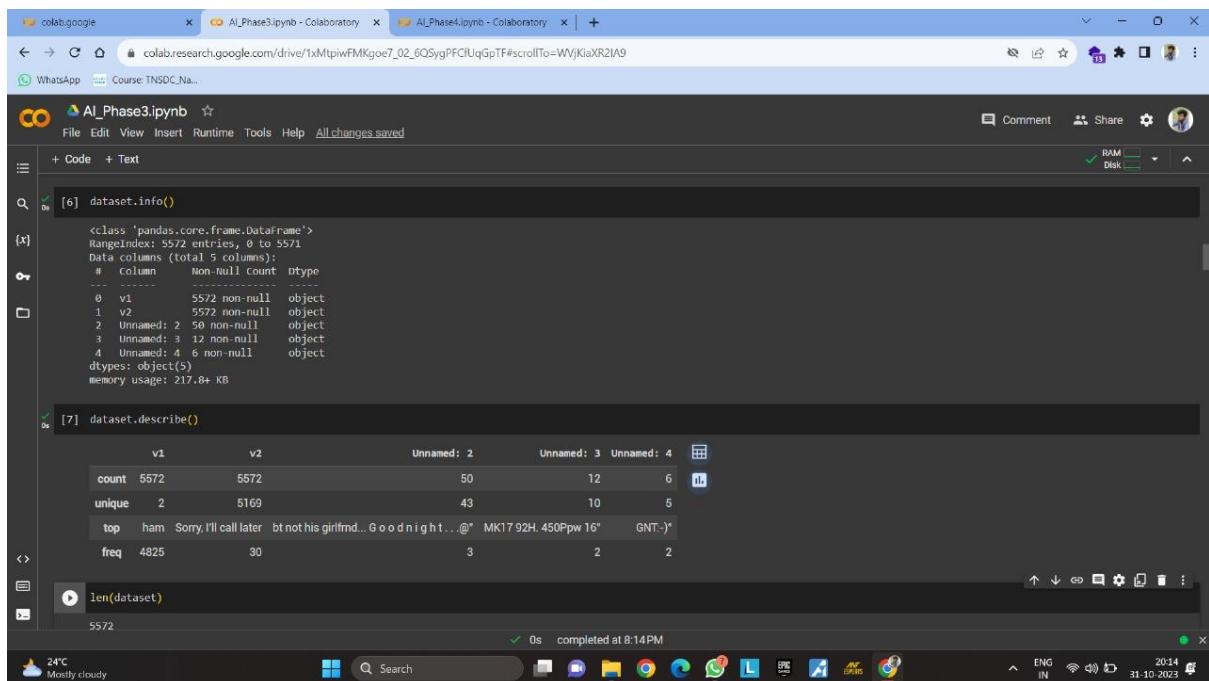
	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say ...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
...
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will l_b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN

5572 rows x 5 columns

The code cell also contains the following text:

```
[ ] dataset.info()
```

The output shows the class of the dataset: `<class 'pandas.core.frame.DataFrame'>`.



The screenshot shows a Google Colab notebook titled "AI_Phase3.ipynb". The code cell contains the following text:

```
[6] dataset.info()
```

The output displays the following information:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   column        Non-Null count  Dtype
---  -
 0    v1            5572 non-null    object
 1    v2            5572 non-null    object
 2  Unnamed: 2    50 non-null     object
 3  Unnamed: 3    12 non-null     object
 4  Unnamed: 4    6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

The code cell also contains the following text:

```
[7] dataset.describe()
```

The output displays a summary of the dataset:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
count	5572	5572	50	12	6
unique	2	5169	43	10	5
top	ham	Sorry, I'll call later	bt not his girlfriend... Good night...@	MK17 92H. 450Ppw 16"	GNT-)
freq	4825	30	3	2	2

The code cell also contains the following text:

```
len(dataset)
```

The output shows the length of the dataset: 5572.

colab.google x AI_Phase3.ipynb - Colaboratory x AI_Phase4.ipynb - Colaboratory x +

colab.research.google.com/drive/1xMtpiwFMKgoe7_02_6QSygPFClUqGpTF#scrollTo=p2MO2ct027wO

WhatsApp Course: TNSDC.Na...

AI_Phase3.ipynb

File Edit View Insert Runtime Tools Help Saving...

+ Code + Text

```
[8] len(dataset)

5572

[9] # Check for null values
dataset.isna().sum()

v1      0
v2      0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64

# Check the number of columns
dataset.columns

Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')

# Check for number of unique values
dataset.nunique()

v1      2
v2    5169
Unnamed: 2    43
Unnamed: 3    10
Unnamed: 4     5
dtype: int64
```

0s completed at 8:15PM

24°C Mostly cloudy

Search

ENG IN 20:15 31-10-2023

colab.google x AI_Phase3.ipynb - Colaboratory x AI_Phase4.ipynb - Colaboratory x +

colab.research.google.com/drive/1xMtpiwFMKgoe7_02_6QSygPFClUqGpTF#scrollTo=Waa3Js4IAY_F

WhatsApp Course: TNSDC.Na...

AI_Phase3.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
dtype: int64

[12] # Checking for balance
dataset['v1'].value_counts()

ham    4825
spam    747
Name: v1, dtype: int64

# Dropping unnamed columns
dataset = dataset[['v1', 'v2']]
dataset.head()

v1      v2
0  ham    Go until jurong point, crazy.. Available only ...
1  ham    Ok lar... Joking wif u oni...
2  spam   Free entry in 2 a wkly comp to win FA Cup fina...
3  ham    U dun say so early hor... U c already then say...
4  ham    Nah I don't think he goes to usf, he lives aro...
```

0s completed at 8:15PM

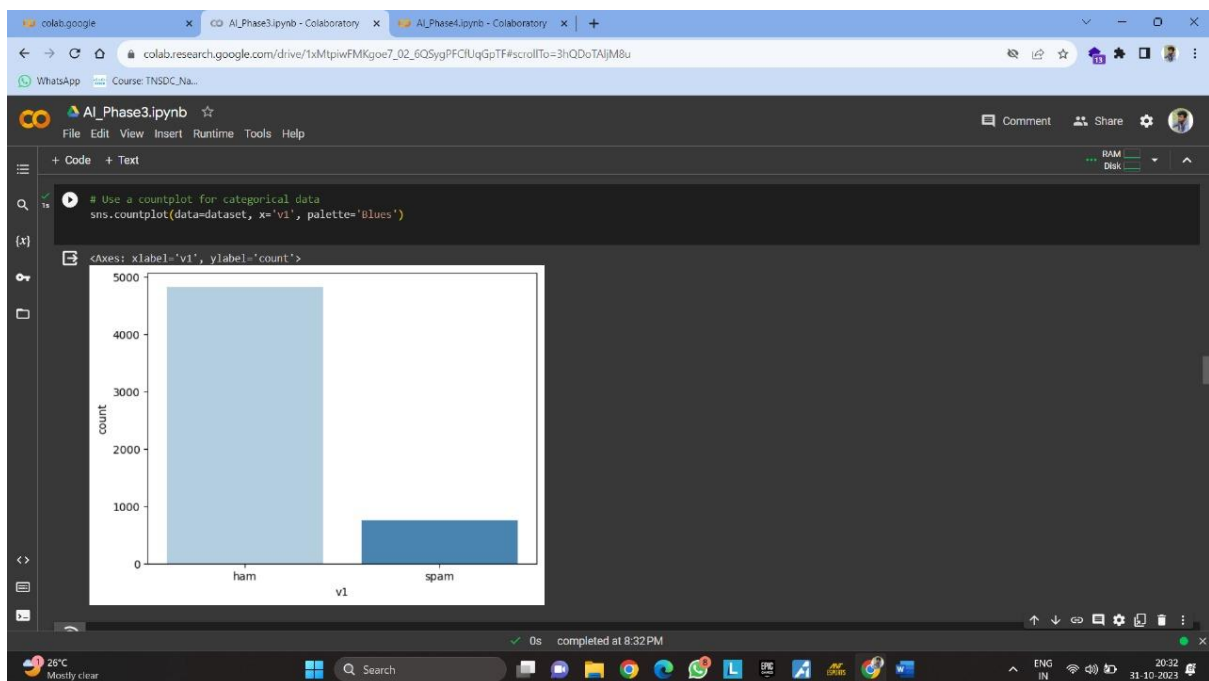
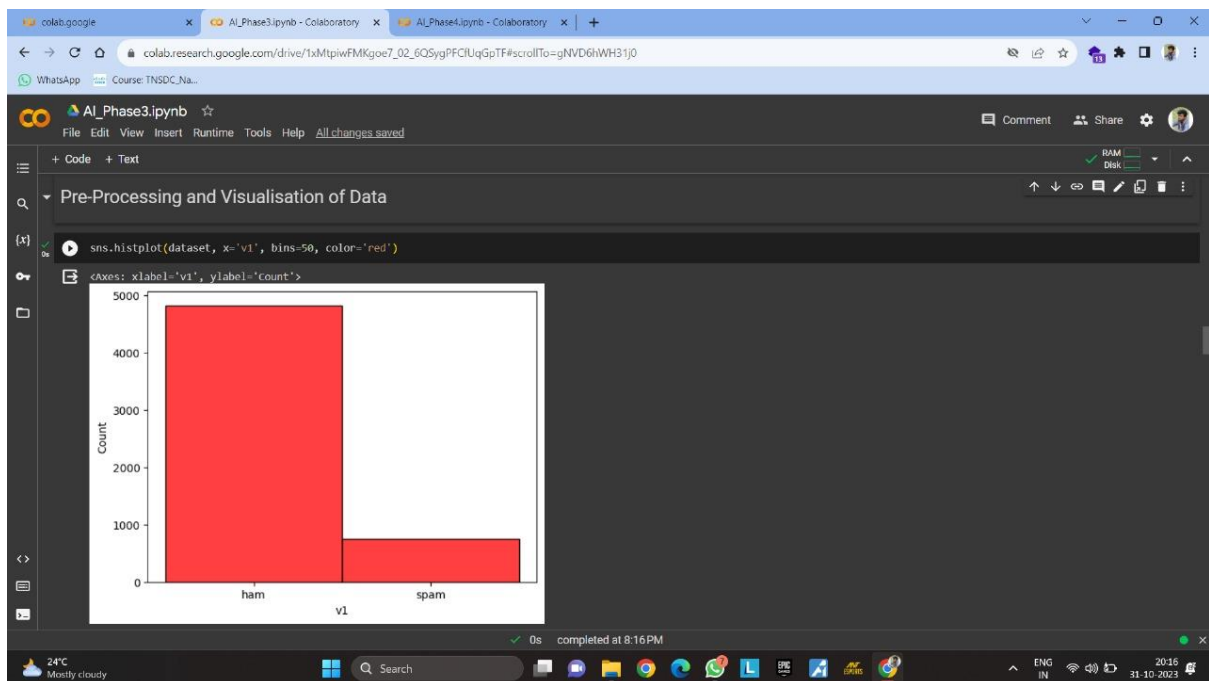
24°C Mostly cloudy

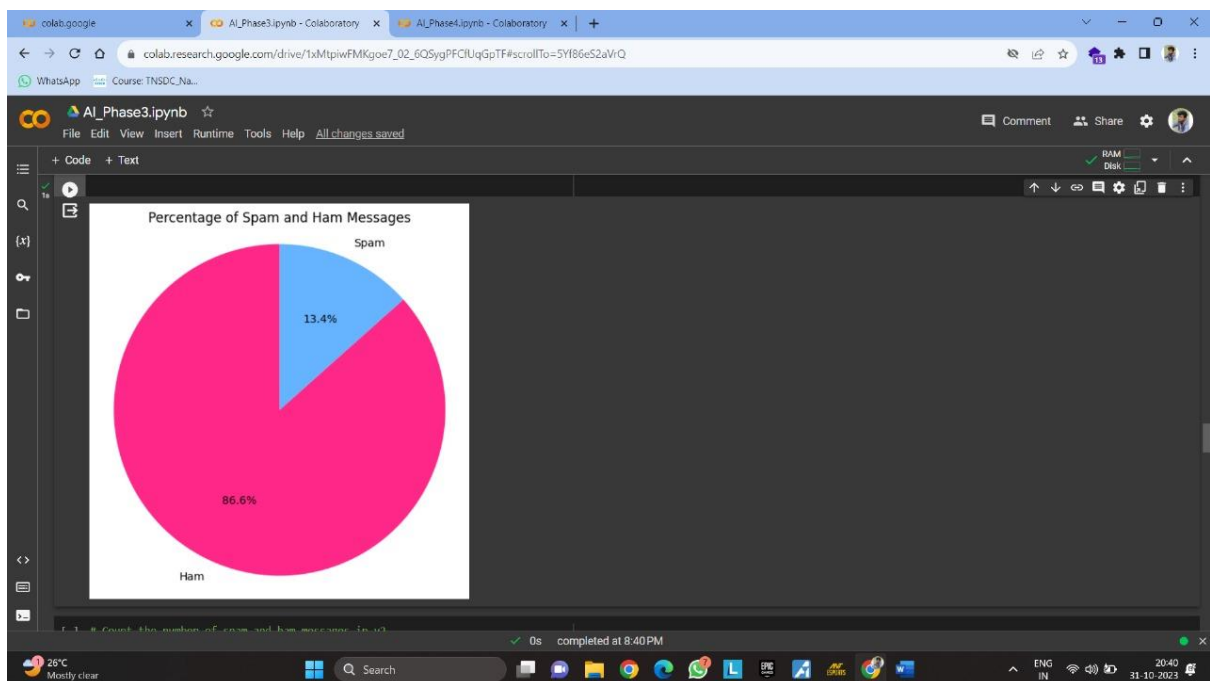
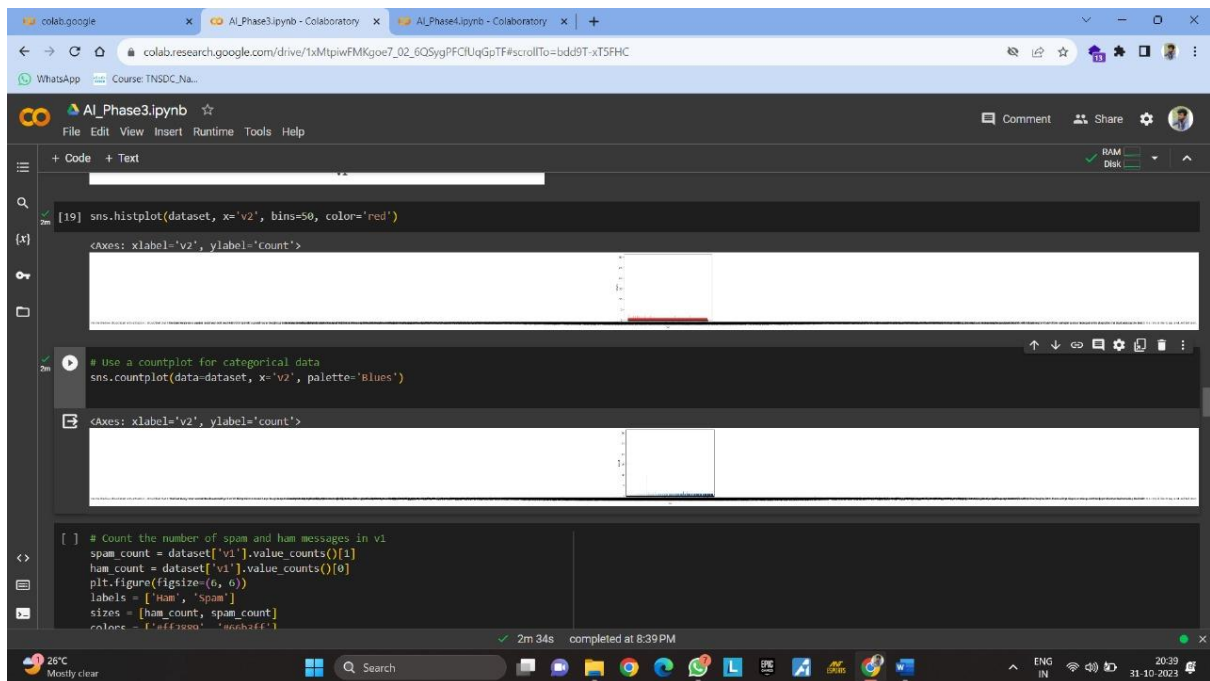
Search

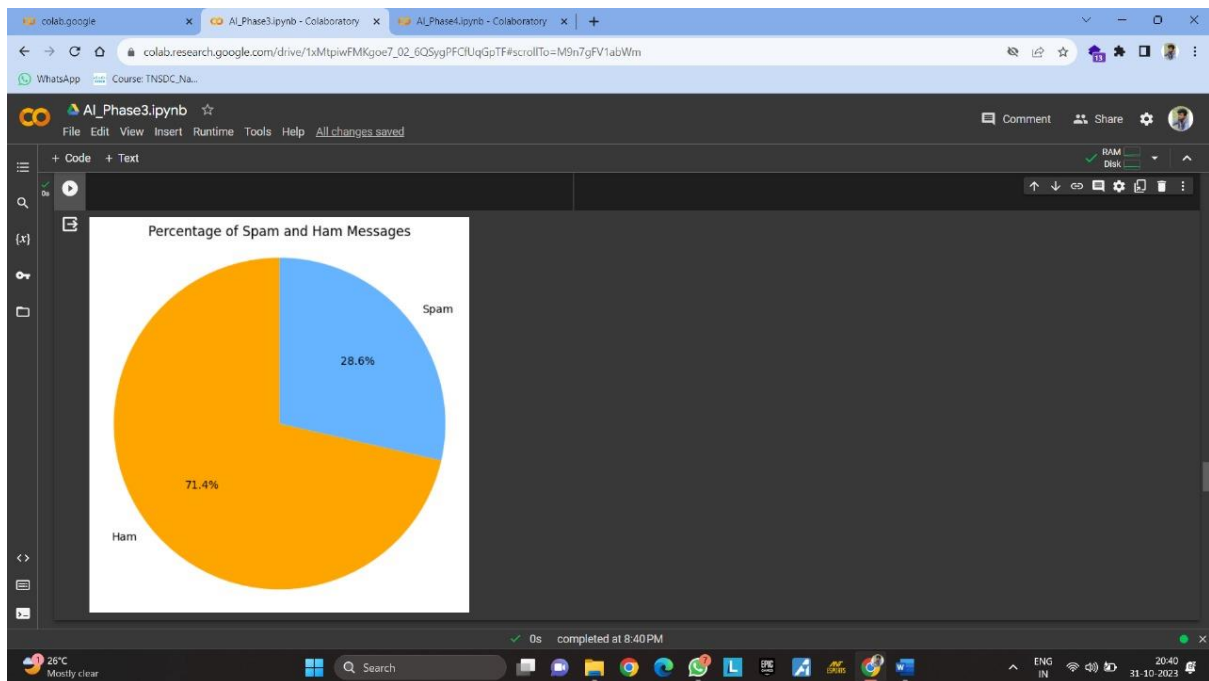
ENG IN 20:15 31-10-2023

Pre-Processing and Visualisation of Data

Step 3: Pre-processing and Visualisation of Dataset:







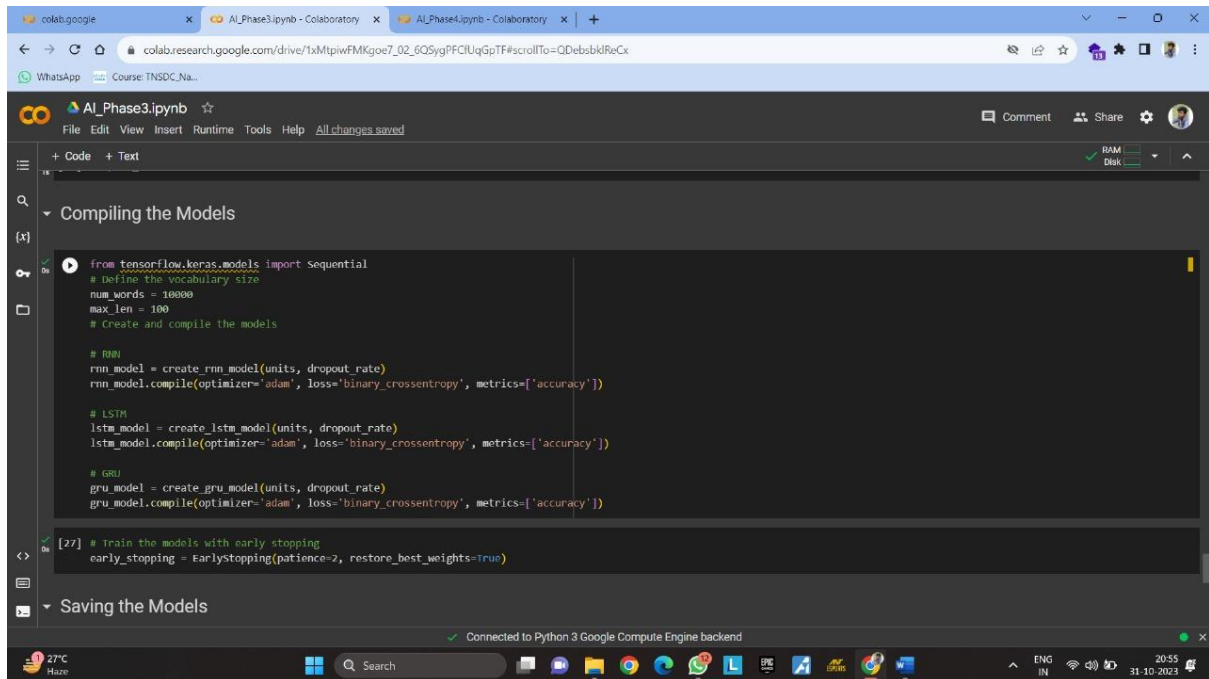
Step 4: Modeling the Dataset:

```
def create_rnn_model(units, dropout_rate):
    model = Sequential()
    model.add(Embedding(input_dim=num_words, output_dim=100, input_length=max_len))
    model.add(SimpleRNN(units))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    return model

def create_lstm_model(units, dropout_rate):
    model = Sequential()
    model.add(Embedding(input_dim=num_words, output_dim=100, input_length=max_len))
    model.add(LSTM(units))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    return model

def create_gru_model(units, dropout_rate):
    model = Sequential()
    model.add(Embedding(input_dim=num_words, output_dim=100, input_length=max_len))
    model.add(GRU(units))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    return model
```


Step 5: Compiling the Model:



```
from tensorflow.keras.models import Sequential
# Define the vocabulary size
num_words = 10000
max_len = 100
# Create and compile the models

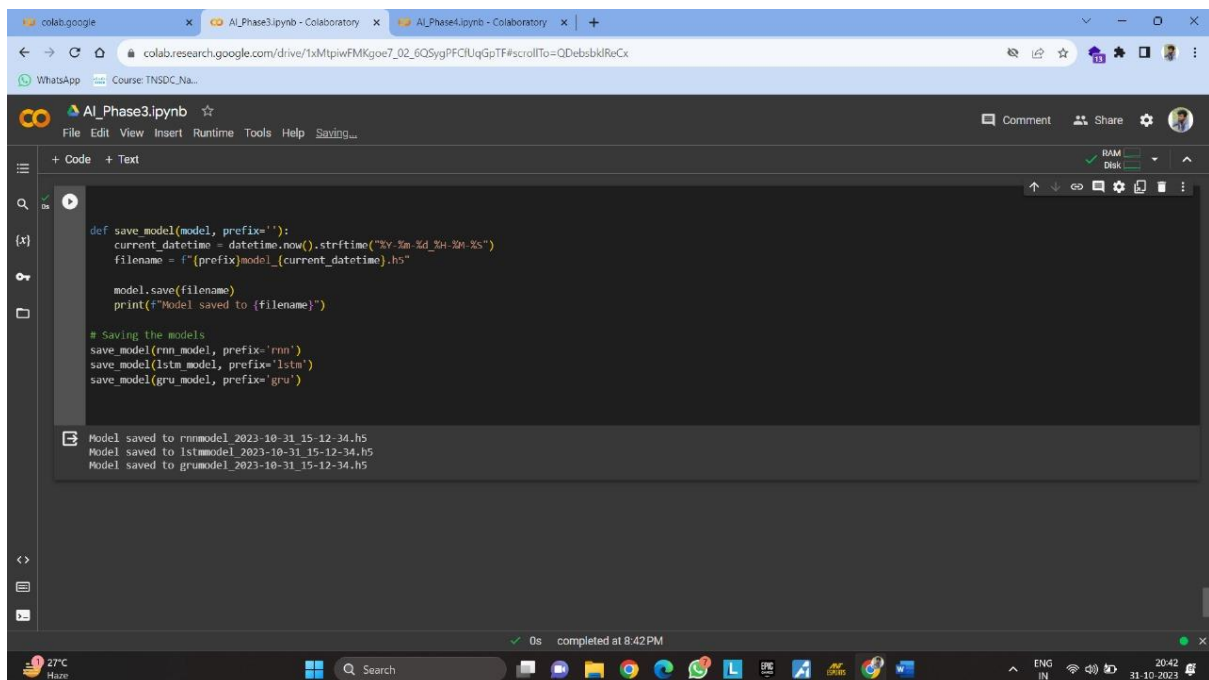
# RNN
rnn_model = create_rnn_model(units, dropout_rate)
rnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# LSTM
lstm_model = create_lstm_model(units, dropout_rate)
lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# GRU
gru_model = create_gru_model(units, dropout_rate)
gru_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

[27] # Train the models with early stopping
early_stopping = EarlyStopping(patience=2, restore_best_weights=True)
```

Step 6: Saving the Model:

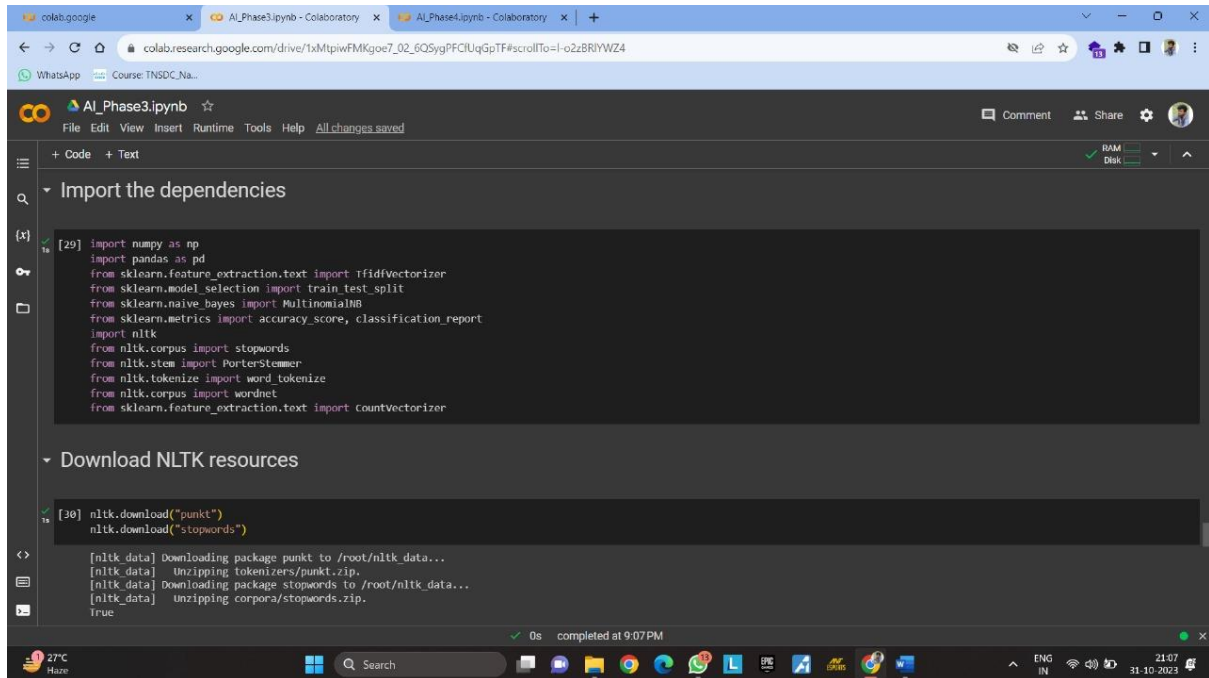


```
def save_model(model, prefix=''):
    current_datetime = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    filename = f"{prefix}model_{current_datetime}.h5"
    model.save(filename)
    print(f"Model saved to {filename}")

# Saving the models
save_model(rnn_model, prefix='rnn')
save_model(lstm_model, prefix='lstm')
save_model(gru_model, prefix='gru')
```

Model saved to rnnmodel_2023-10-31_15-12-34.h5
Model saved to lstmmodel_2023-10-31_15-12-34.h5
Model saved to grumodel_2023-10-31_15-12-34.h5

Step 7: Import the dependencies and Download the NLTK resources:



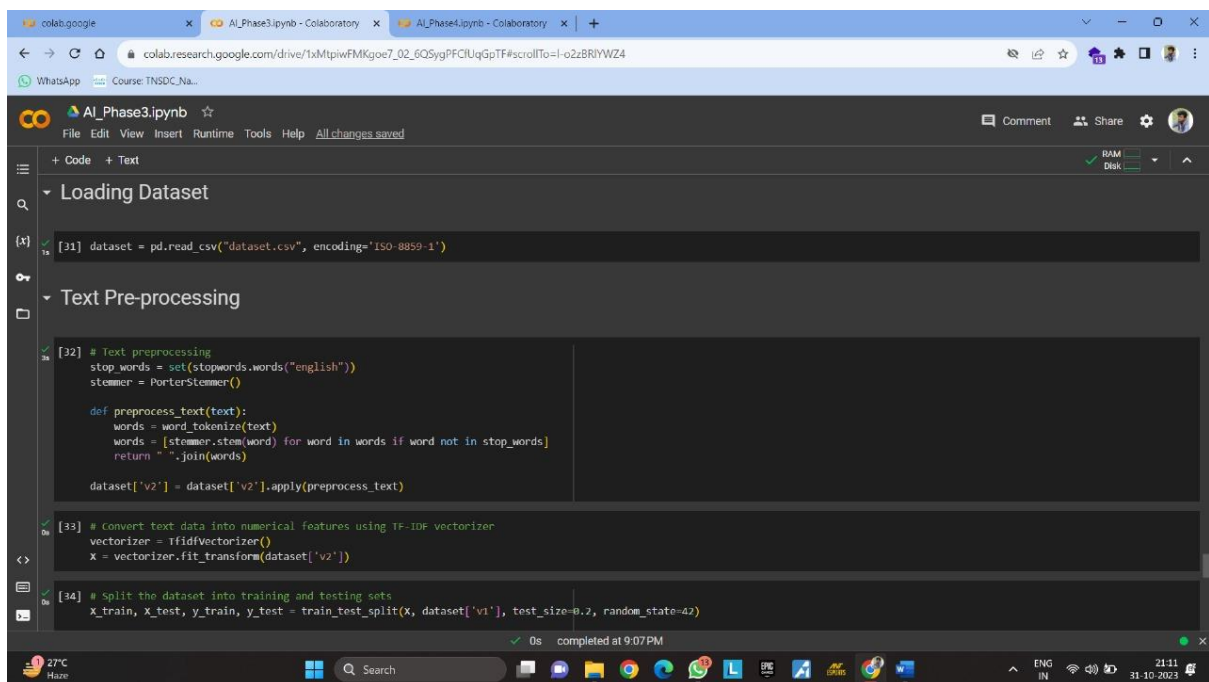
The screenshot shows a Google Colab notebook titled "AI_Phase3.ipynb". The notebook has two code cells. The first cell, labeled [29], imports various dependencies including numpy, pandas, sklearn, and nltk. The second cell, labeled [30], downloads the NLTK resources for 'punct' and 'stopwords'. The output of the second cell shows the progress of downloading and unzipping these resources.

```
[29] import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet
from sklearn.feature_extraction.text import CountVectorizer

[30] nltk.download("punct")
nltk.download("stopwords")

[nltk_data] Downloading package punct to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punct.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
True
```

Step 8: Loading the dataset and Text pre-processing of Dataset:



The screenshot shows a Google Colab notebook titled "AI_Phase3.ipynb". The notebook has three code cells. The first cell, labeled [31], loads the dataset using pandas. The second cell, labeled [32], performs text pre-processing by defining a function to tokenize and stem words, removing stop words, and applying this function to the dataset. The third cell, labeled [33], converts the text data into numerical features using a TfidfVectorizer. The fourth cell, labeled [34], splits the dataset into training and testing sets.

```
[31] dataset = pd.read_csv("dataset.csv", encoding="ISO-8859-1")

[32] # Text pre-processing
stop_words = set(stopwords.words("english"))
stemmer = PorterStemmer()

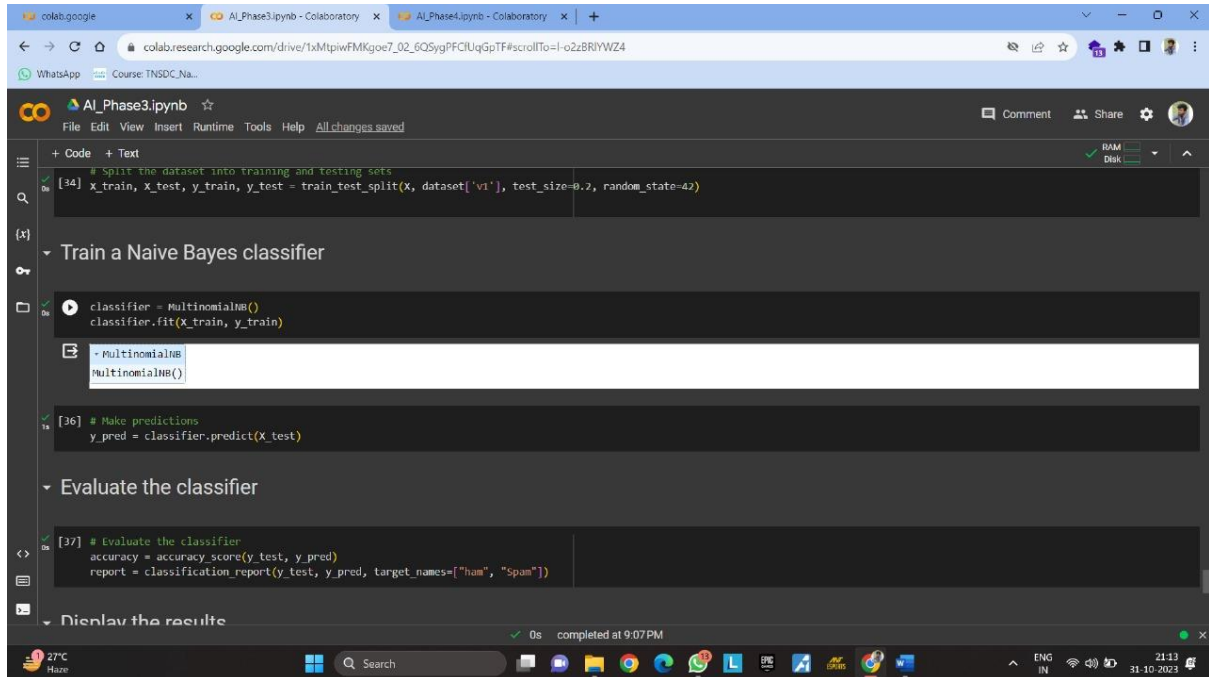
def preprocess_text(text):
    words = word_tokenize(text)
    words = [stemmer.stem(word) for word in words if word not in stop_words]
    return " ".join(words)

dataset['v2'] = dataset['v2'].apply(preprocess_text)

[33] # Convert text data into numerical features using TF-IDF vectorizer
vectorizer = TfidfVectorizer()
x = vectorizer.fit_transform(dataset['v2'])

[34] # Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, dataset['v1'], test_size=0.2, random_state=42)
```

Step 9: Train a Naïve Bayes classifier and Evaluate the classifier:



The screenshot shows a Google Colab notebook titled "AI_Phase3.ipynb". The code is as follows:

```
[34] # Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, dataset['v1'], test_size=0.2, random_state=42)

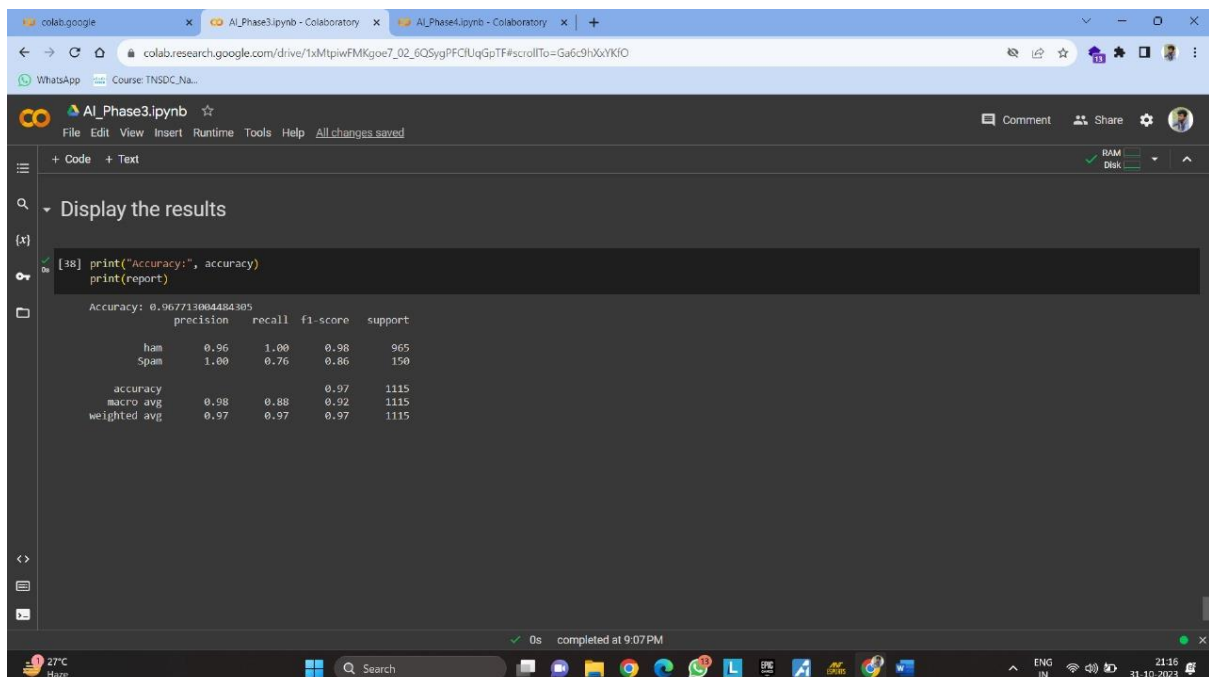
# Train a Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(x_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=["ham", "Spam"])
```

The notebook interface shows the code is executed successfully. The status bar at the bottom indicates "completed at 9:07 PM".

Step 10: Display the Results:



The screenshot shows the same Google Colab notebook, now displaying the results of the classifier. The code is as follows:

```
[38] print("Accuracy:", accuracy)
print(report)
```

The output of the code is displayed in the notebook:

```
Accuracy: 0.967713064484305
precision recall f1-score support
ham 0.96 1.00 0.98 965
Spam 1.00 0.76 0.86 150
accuracy 0.98 0.88 0.97 1115
macro avg 0.98 0.88 0.92 1115
weighted avg 0.97 0.97 0.97 1115
```

The notebook interface shows the code is executed successfully. The status bar at the bottom indicates "completed at 9:07 PM".

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score,
mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
import warnings
warnings.filterwarnings("ignore")
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU,
Dense, Dropout
from datetime import datetime
from google.colab import files
uploaded=files.upload()

dataset = pd.read_csv("dataset.csv", encoding='ISO-8859-1')

dataset
```

```
dataset.info()
```

```
dataset.describe()
```

```
len(dataset)
```

```
# Check for null values
```

```
dataset.isna().sum()
```

```
# Check the number of columns
```

```
dataset.columns
```

```
# Check for number of unique values
```

```
dataset.nunique()
```

```
# Checking for balancement
```

```
dataset['v1'].value_counts()
```

```
# Dropping unnamed columns
```

```
dataset = dataset[['v1', 'v2']]
```

```
dataset.head()
```

```
sns.histplot(dataset, x='v1', bins=50, color='red')
```

```
# Use a countplot for categorical data
```

```
sns.countplot(data=dataset, x='v1', palette='Blues')
```

```
sns.histplot(dataset, x='v2', bins=50, color='red')
```

```
# Use a countplot for categorical data
```

```
sns.countplot(data=dataset, x='v2', palette='Blues')
```

```
# Count the number of spam and ham messages in v1
```

```
spam_count = dataset['v1'].value_counts()[1]
```

```
ham_count = dataset['v1'].value_counts()[0]
```

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

```
labels = ['Ham', 'Spam']
```

```
sizes = [ham_count, spam_count]
```

```
colors = ['#ff2889', '#66b3ff']
```

```
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
```

```
plt.title('Percentage of Spam and Ham Messages')
```

```
plt.axis('equal')
```

```
plt.show()
```

```
# Count the number of spam and ham messages in v2
```

```
spam_count = dataset['v2'].value_counts()[1]
```

```
ham_count = dataset['v2'].value_counts()[0]
```

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

```
labels = ['Ham', 'Spam']
```

```
sizes = [ham_count, spam_count]
```

```
colors = ['#ffa500', '#66b3ff']
```

```
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
```

```
plt.title('Percentage of Spam and Ham Messages')
```

```
plt.axis('equal')
```

```
plt.show()
```

```
X_text = dataset['v2']
```

```
y = dataset['v2']
```

```
# Split the data into training and testing sets
```

```
X_text_train, X_text_test, y_train, y_test = train_test_split(X_text, y,  
test_size=0.2, random_state=42)
```

```
# TfidfVectorizer to convert text data into numerical features
```

```
tfidf_vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')
```

```
X_train_tfidf = tfidf_vectorizer.fit_transform(X_text_train)
```

```
X_test_tfidf = tfidf_vectorizer.transform(X_text_test)
```

```
# Create a classification model
```

```
clf = DecisionTreeClassifier()
```

```
clf.fit(X_train_tfidf, y_train)
```

```
y_pred = clf.predict(X_test_tfidf)
```

```
# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Model Accuracy: {accuracy:.2f}")
```

```
def create_rnn_model(units, dropout_rate):
```

```
    model = Sequential()
```

```
    model.add(Embedding(input_dim=num_words, output_dim=100,  
input_length=max_len))
```

```
    model.add(SimpleRNN(units))
```

```
    model.add(Dropout(dropout_rate))
```

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

```
    return model
```

```
def create_lstm_model(units, dropout_rate):
```

```
    model = Sequential()
```

```
    model.add(Embedding(input_dim=num_words, output_dim=100,  
input_length=max_len))
```

```
    model.add(LSTM(units))
```

```
    model.add(Dropout(dropout_rate))
```

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

```
    return model
```

```
def create_gru_model(units, dropout_rate):
```

```
    model = Sequential()
```

```
    model.add(Embedding(input_dim=num_words, output_dim=100,  
input_length=max_len))
```

```
    model.add(GRU(units))
```

```
    model.add(Dropout(dropout_rate))
```

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



```
return model
```

```
# Define hyperparameters
```

```
units = 64
```

```
dropout_rate = 0.5
```

```
from tensorflow.keras.models import Sequential
```

```
# Define the vocabulary size
```

```
num_words = 10000
```

```
max_len = 100
```

```
# Create and compile the models
```

```
# RNN
```

```
rnn_model = create_rnn_model(units, dropout_rate)
```

```
rnn_model.compile(optimizer='adam', loss='binary_crossentropy',  
metrics=['accuracy'])
```

```
# LSTM
```

```
lstm_model = create_lstm_model(units, dropout_rate)
```

```
lstm_model.compile(optimizer='adam', loss='binary_crossentropy',  
metrics=['accuracy'])
```

```
# GRU
```

```
gru_model = create_gru_model(units, dropout_rate)
```

```
gru_model.compile(optimizer='adam', loss='binary_crossentropy',  
metrics=['accuracy'])
```

```
# Train the models with early stopping
early_stopping = EarlyStopping(patience=2, restore_best_weights=True)

def save_model(model, prefix=''):
    current_datetime = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    filename = f"{prefix}model_{current_datetime}.h5"

    model.save(filename)
    print(f"Model saved to {filename}")

# Saving the models
save_model(rnn_model, prefix='rnn')
save_model(lstm_model, prefix='lstm')
save_model(gru_model, prefix='gru')

import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

import nltk

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
nltk.download("punkt")
```

```
nltk.download("stopwords")
```

```
dataset = pd.read_csv("dataset.csv", encoding='ISO-8859-1')
```

```
# Text preprocessing
```

```
stop_words = set(stopwords.words("english"))
```

```
stemmer = PorterStemmer()
```

```
def preprocess_text(text):
```

```
    words = word_tokenize(text)
```

```
    words = [stemmer.stem(word) for word in words if word not in  
stop_words]
```

```
    return " ".join(words)
```

```
dataset['v2'] = dataset['v2'].apply(preprocess_text)
```

```
# Convert text data into numerical features using TF-IDF vectorizer
```

```
vectorizer = TfidfVectorizer()
```

```
X = vectorizer.fit_transform(dataset['v2'])
```

```
# Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, dataset['v1'],  
test_size=0.2, random_state=42)
```

```
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=["ham",
"Spam"])

print("Accuracy:", accuracy)
print(report)
```

CONCLUSION

In conclusion, the development of an AI-powered spam classifier represents a significant stride in enhancing digital communication. By tackling the pervasive issue of spam messages, this project offers a promising solution to improve user experiences and security. Through the careful selection of algorithms, effective model training, thoughtful evaluation metrics, and innovative techniques, the project paves the way for a more reliable spam classifier. With user-friendly deployment instructions, this classifier can serve as a valuable tool to combat unwanted messages, making digital communication safer and more efficient. The journey doesn't end here; continuous adaptation, user feedback, and vigilance against emerging spam patterns will be key to maintaining the classifier's success.