Tech Saksham

Capstone Project Report

“Detecting Spam Emails”

“GOVERNMENT COLLEGE OF ENGINEERING, SALEM”

NM ID	NAME
au61772111023	DIVYA M S

Ramar Bose

Sr. AI Master Trainer

ABSTRACT

Email spam continues to be a pervasive issue, causing inconvenience and potential harm to users. In this project, we propose a machine learning approach to automatically detect spam emails. The project involves collecting a dataset of labeled emails, where each email is classified as spam or non-spam (ham).

We preprocess the email text data to convert it into a suitable format for machine learning models. Feature extraction techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) and N-grams are used to extract relevant features from the text data.

We then train a machine learning model, such as a Naive Bayes classifier or a Support Vector Machine (SVM), on the extracted features. The trained model is evaluated using metrics such as accuracy, precision, recall, and F1 score to assess its performance in detecting spam emails.

INDEX

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

- The problem of spam emails has been a persistent issue since the early days of the internet. Spam emails are unsolicited messages sent in bulk to a large number of recipients, often with commercial or malicious intent. These emails can be annoying, cluttering up users' inboxes with irrelevant messages, but they can also pose serious threats to security and privacy.
- Spam emails often contain malicious content, such as phishing links or malware, that can trick users into revealing sensitive information or infect their devices. Phishing emails, for example, are designed to look like legitimate messages from reputable sources, such as banks or online services, in an attempt to deceive users into providing their personal or financial information.
- To address the problem of spam emails, various solutions have been developed over the years. One common approach is to use spam filters, which are algorithms designed to automatically identify and filter out spam emails from users' inboxes. These filters work by analyzing the content and characteristics of emails to determine whether they are likely to be spam or legitimate messages.

- Machine learning has emerged as a powerful tool for spam email detection, as it allows for the development of sophisticated algorithms that can analyze large amounts of email data to identify patterns and trends associated with spam emails. By training machine learning models on labeled datasets of spam and non-spam emails, these algorithms can learn to accurately classify new emails as spam or non-spam based on their content and characteristics.
- The goal of this project is to develop a machine learning-based solution for detecting spam emails and filtering them out from users' inboxes. By reducing the amount of unwanted and potentially harmful emails that users receive, the project aims to improve the overall email experience for users and enhance their security and privacy.

1.2 Proposed Solution

- Our proposed solution for detecting spam emails involves using machine learning algorithms, a popular approach due to its ability to analyze large amounts of data and identify patterns that may not be apparent to human reviewers. Here's a detailed explanation of each component:
- Data Collection:
- Gathering a large and diverse dataset of labeled emails is crucial for training a machine learning model to accurately distinguish between spam and non-spam emails.

- The dataset should ideally contain emails from various sources and in different languages to ensure the model's effectiveness across different contexts.
- Data Preprocessing:
- Preprocessing the email text involves several steps to clean and prepare the data for analysis.
- This includes removing any HTML tags, special characters, and punctuation marks that do not contribute to the content of the email.
- Stopwords (common words like "and", "the", "is") are also removed as they do not carry significant meaning in distinguishing between spam and non-spam emails.
- The text is then tokenized, which means splitting it into individual words or tokens, to prepare it for feature extraction.
- Feature Extraction:
- Feature extraction is a critical step in converting the email text into a format that can be used by machine learning algorithms.
- Common techniques include TF-IDF, which calculates the importance of a word in an email relative to its frequency in a collection of emails, and word embeddings, which represent words as dense vectors in a continuous space.
- N-grams, which are sequences of N words, can also be used to capture contextual information in the email text.
- Machine Learning Model:
- Once the email text has been preprocessed and features extracted, it is ready to be used as input to a machine learning model.

- Common models for spam email detection include logistic regression, support vector machines (SVM), and naive Bayes classifiers.
- These models are trained on the labeled dataset of emails to learn the patterns and characteristics that distinguish spam from non-spam emails.
- Model Evaluation and Testing:
- After training the machine learning model, it is evaluated using a separate test dataset to assess its performance.
- Metrics such as accuracy, precision, recall, and F1 score are commonly used to evaluate the model's effectiveness in correctly classifying spam and non-spam emails.
- By following these steps, the proposed solution aims to develop a robust and accurate spam email detection system that can effectively filter out unwanted emails, thereby improving the overall email experience for users.

1.3 Feature

- Machine Learning Algorithms:
- Logistic Regression: A simple yet effective algorithm for binary classification, where it models the probability of an email being spam or non-spam based on the input features.
- Support Vector Machines (SVM): A powerful algorithm for binary classification that finds the hyperplane that best separates spam and non-spam emails in the feature space.

- Naive Bayes Classifiers: A family of probabilistic classifiers based on Bayes' theorem, where it assumes that the presence of a particular feature in an email is independent of the presence of any other feature.
- Data Preprocessing:
- Removing Irrelevant Information: This includes removing HTML tags, special characters, and punctuation marks that do not contribute to the content of the email.
- Tokenization: Splitting the text into individual words or tokens to prepare it for further analysis.
- Stopword Removal: Removing common words (e.g., "and", "the") that occur frequently but do not carry much meaning in distinguishing between spam and non-spam emails.
- Feature Extraction:
- Word Frequency: Counting the number of times each word appears in an email, which can help capture the importance of certain words in determining whether an email is spam or non-spam.
- TF-IDF (Term Frequency-Inverse Document Frequency): A measure that reflects how important a word is to a document in a collection or corpus, which helps in identifying key words that are more likely to be associated with spam emails.
- N-grams: Sequences of N words, which can capture contextual information and help identify patterns in the text that are indicative of spam or non-spam emails.
- Model Training:
- Splitting the Dataset: The dataset is typically split into training, validation, and test sets, where the training set is used to train the model,

the validation set is used to tune hyperparameters, and the test set is used to evaluate the model's performance.

- Training the Model: The machine learning model is trained on the training set using the selected algorithm and features extracted from the email text data.
- Model Evaluation:
- Metrics: Various metrics are used to evaluate the model's performance, including accuracy (the proportion of correctly classified emails), precision (the proportion of correctly classified spam emails among all emails classified as spam), recall (the proportion of correctly classified spam emails among all actual spam emails), and F1 score (the harmonic mean of precision and recall).
- Cross-Validation: Cross-validation techniques, such as k-fold cross-validation, are used to ensure that the model's performance is consistent across different subsets of the data and to reduce the risk of overfitting.
- These components work together to create a robust and accurate spam email detection system that can effectively classify emails as spam or non-spam based on their content and characteristics.

1.4 Advantages

- Improved Email Security: By effectively filtering out spam emails, the project enhances email security by reducing the risk of users clicking on malicious links or downloading harmful attachments that may compromise their personal information or devices.

- Enhanced User Experience: Filtering out spam emails improves the overall email experience for users by reducing the clutter in their inbox and ensuring that they only see relevant and legitimate emails. This can lead to increased productivity and efficiency in managing emails.
- Time and Resource Savings: By automatically filtering out spam emails, the project saves users time and resources that would otherwise be spent manually sorting through and deleting unwanted emails. This can be particularly beneficial for individuals and organizations receiving a large volume of emails daily.
- Protection Against Phishing Attacks: Spam emails often contain phishing attempts to steal sensitive information such as login credentials or financial information. By filtering out these emails, the project protects users from falling victim to phishing attacks and potential identity theft.
- Scalability: The machine learning-based approach used in the project is scalable and can be applied to large volumes of emails, making it suitable for use in email services with a large user base. This ensures that the spam filtering system remains effective even as the volume of emails increases.
- Adaptability to New Spamming Techniques: The project's machine learning models can be trained and updated regularly to adapt to new spamming techniques used by spammers. This ensures that the spam filtering system remains effective in detecting and filtering out new types of spam emails.
- Cost-Effective Solution: Implementing a spam email detection system based on machine learning is a cost-effective solution compared to

manual spam filtering or using third-party spam filtering services. It reduces the need for human intervention in managing spam emails, leading to cost savings for individuals and organizations.

- Customization: The project allows for customization of the spam filtering rules and criteria based on the specific needs and preferences of users or organizations. This flexibility ensures that the spam filtering system can be tailored to meet the unique requirements of different users or organizations.

1.5 Scope

- Literature Review: Conduct a comprehensive review of existing research and techniques for spam email detection. This includes studying various machine learning approaches, such as supervised learning algorithms (e.g., logistic regression, SVM, naive Bayes), unsupervised learning algorithms (e.g., clustering), and deep learning models (e.g., neural networks). Additionally, review rule-based methods that use predefined rules to classify emails as spam or non-spam, as well as hybrid models that combine machine learning and rule-based approaches for improved performance.
- Data Collection: Gather a dataset of labeled emails, including examples of spam and non-spam emails, to train and test the machine learning models. The dataset should be representative of the types of emails users typically receive and should be large enough to ensure the models are trained effectively.

- Data Preprocessing: Preprocess the email text data to remove noise, such as HTML tags, special characters, and stopwords. Tokenize the text to convert it into individual words or tokens, and convert it into a format suitable for machine learning models, such as TF-IDF vectors or word embeddings.
- Model Evaluation: Evaluate the trained models using metrics such as accuracy, precision, recall, and F1 score to assess their performance in detecting spam emails. These metrics provide insights into how well the models are able to correctly classify spam and non-spam emails.
- System Implementation: Implement the spam email detection system, including integrating it with existing email services (e.g., Gmail, Outlook) to automatically filter spam emails. Develop a user-friendly interface for users to interact with the system, allowing them to view and manage spam emails effectively.
- Testing and Validation: Test the system with a separate dataset of emails to validate its performance and ensure that it meets the requirements.

CHAPTER 2

SERVICES AND TOOLS REQUIRED

2.1 LR - Exiting Models

2.1 Required – System config | Cloud computing

2.1 Services Used

- Machine Learning Libraries: Utilize machine learning libraries such as sci-kit-learn, TensorFlow, or PyTorch for developing and training machine learning models for spam email detection.
- Email Dataset: Use publicly available email datasets or collect a custom dataset of labeled emails for training and testing the machine learning models.
- Data Preprocessing Tools: Use tools such as NLTK (Natural Language Toolkit) or spaCy for preprocessing the email text data, including tokenization, stopword removal, and stemming.
- Feature Extraction Libraries: Utilize libraries such as scikit-learn or TensorFlow for extracting features from the email text data, such as TF-IDF (Term Frequency-Inverse Document Frequency) and N-grams.
- Model Training and Evaluation Platforms: Use platforms such as Google Colab or Amazon SageMaker for training and evaluating machine learning models, including logistic regression, support vector machines (SVM), and naive Bayes classifiers.

- Email Services Integration: Integrate the spam email detection system with existing email services such as Gmail or Outlook using APIs or SDKs.
- User Interface Development Tools: Use front-end development tools such as HTML, CSS, and JavaScript for developing a user-friendly interface for the spam detection system.
- Documentation and Reporting Tools: Use tools such as Microsoft Word or LaTeX for documenting the project, including the problem statement, methodology, results, and conclusions, in a comprehensive project report.

2.2 Tools and Software used

- Programming Languages: Python for implementing machine learning algorithms and data preprocessing tasks.
- Machine Learning Libraries: Scikit-learn, TensorFlow, or PyTorch for developing and training machine learning models.
- Data Preprocessing Tools: NLTK (Natural Language Toolkit) or spaCy for preprocessing the email text data.
- Feature Extraction Libraries: Scikit-learn or TensorFlow for extracting features from the email text data.
- Development Environment: Jupyter Notebook or Google Colab for developing and testing code.
- Version Control: Git for version control and collaboration.
- Email Dataset: Publicly available email datasets or custom datasets collected for training and testing.

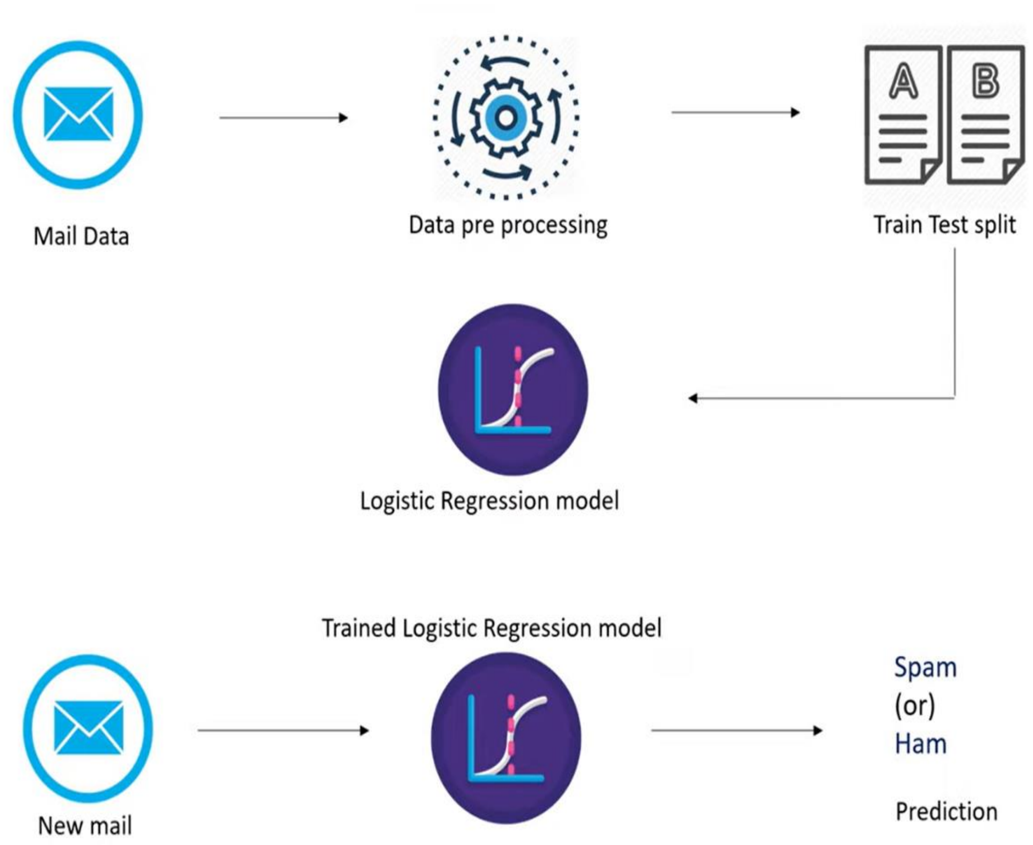
- Email Services Integration: APIs or SDKs for integrating the spam email detection system with email services such as Gmail or Outlook.
- User Interface Development: HTML, CSS, and JavaScript for developing a user-friendly interface for the spam detection system.
- Documentation and Reporting: Microsoft Word or LaTeX for documenting the project, including the problem statement.

CHAPTER 3

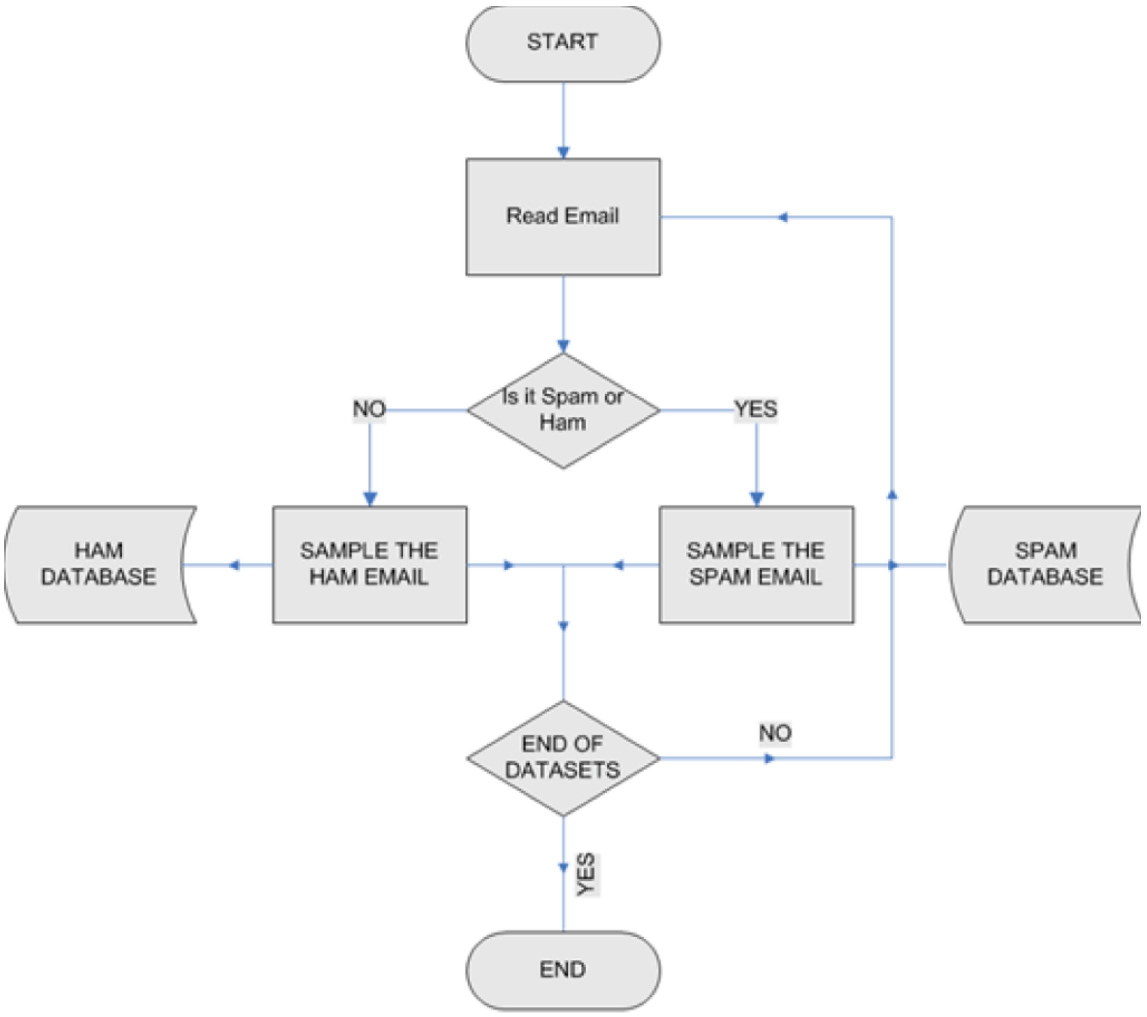
PROJECT ARCHITECTURE

3.1 Architecture

1. System flow diagram



2. Data flow diagram



3. Modules

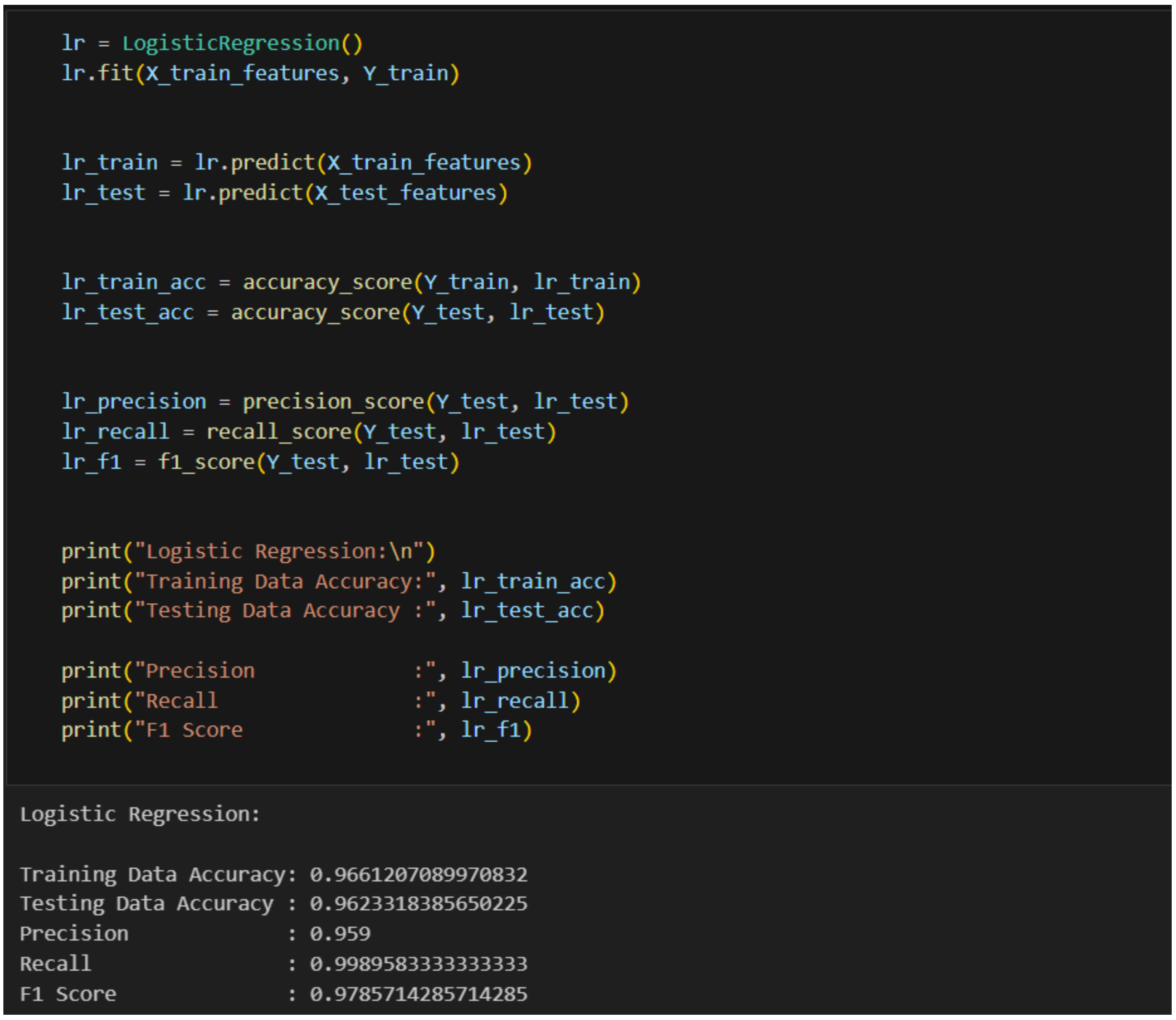
- Data Collection Module: Responsible for collecting a dataset of labeled emails, including spam and non-spam emails, for training and testing the machine learning models.

- Data Preprocessing Module: Handles preprocessing tasks such as removing noise, tokenization, and converting text into a format suitable for machine learning models.

- Feature Extraction Module: Extracts relevant features from the email text data, such as word frequency, TF-IDF, and N-grams, to use as input to the machine learning models.
- Machine Learning Model Module: Implements machine learning algorithms such as logistic regression, support vector machines (SVM), or naive Bayes classifiers for classifying emails as spam or non-spam.
- Model Training and Evaluation Module: Handles the training of machine learning models using the collected dataset and evaluates their performance using metrics such as accuracy, precision, recall, and F1 score.
- Email Integration Module: Integrates the spam email detection system with existing email services using APIs or SDKs for real-time detection and filtering of spam emails.
- User Interface Module: Develops a user-friendly interface for users to interact with the spam detection system, allowing them to view and manage spam emails effectively.

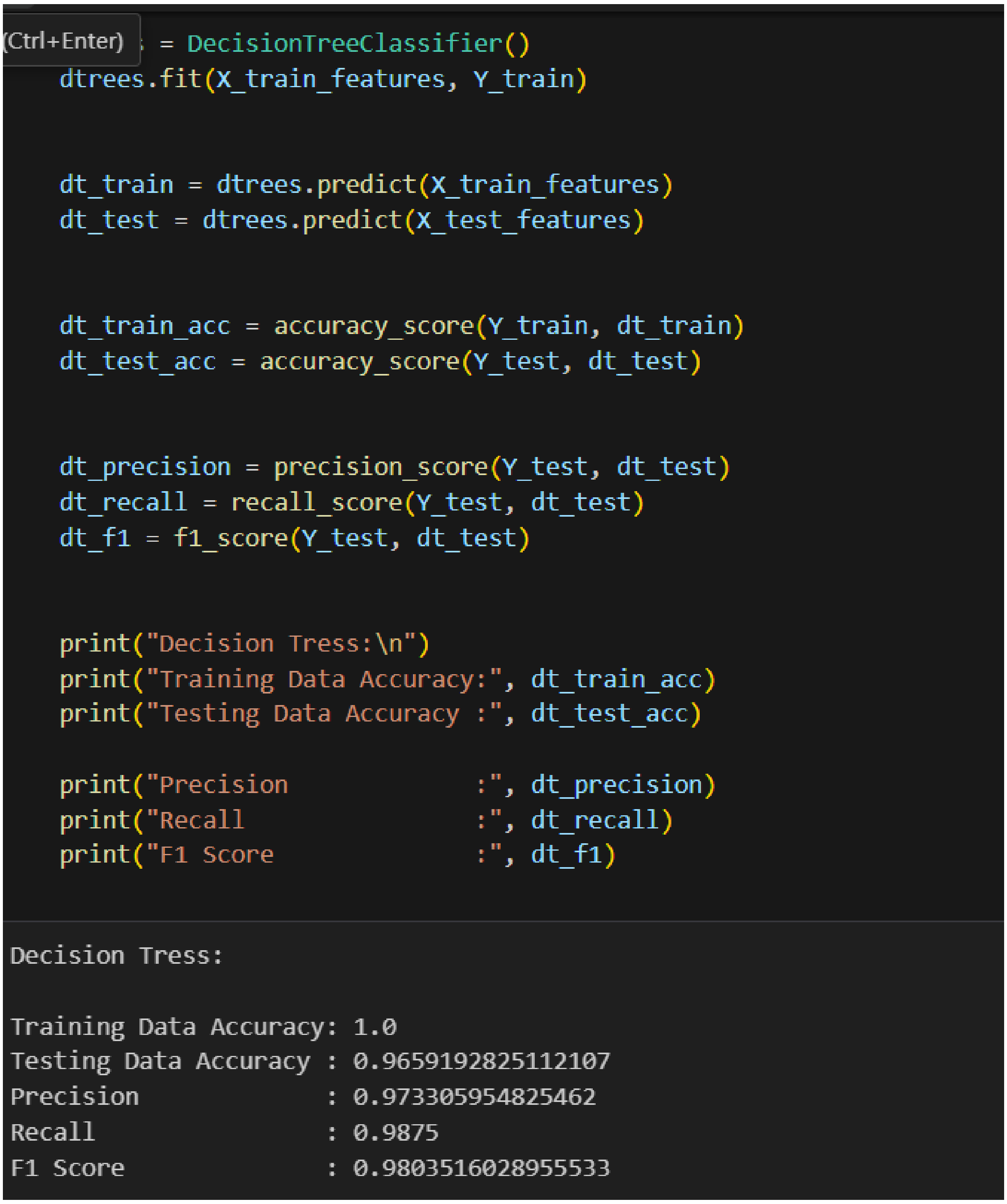
- Documentation and Reporting Module: Documents the entire project, including the problem statement, methodology, results, and conclusions, in a comprehensive project report.

- **Training model diagram**



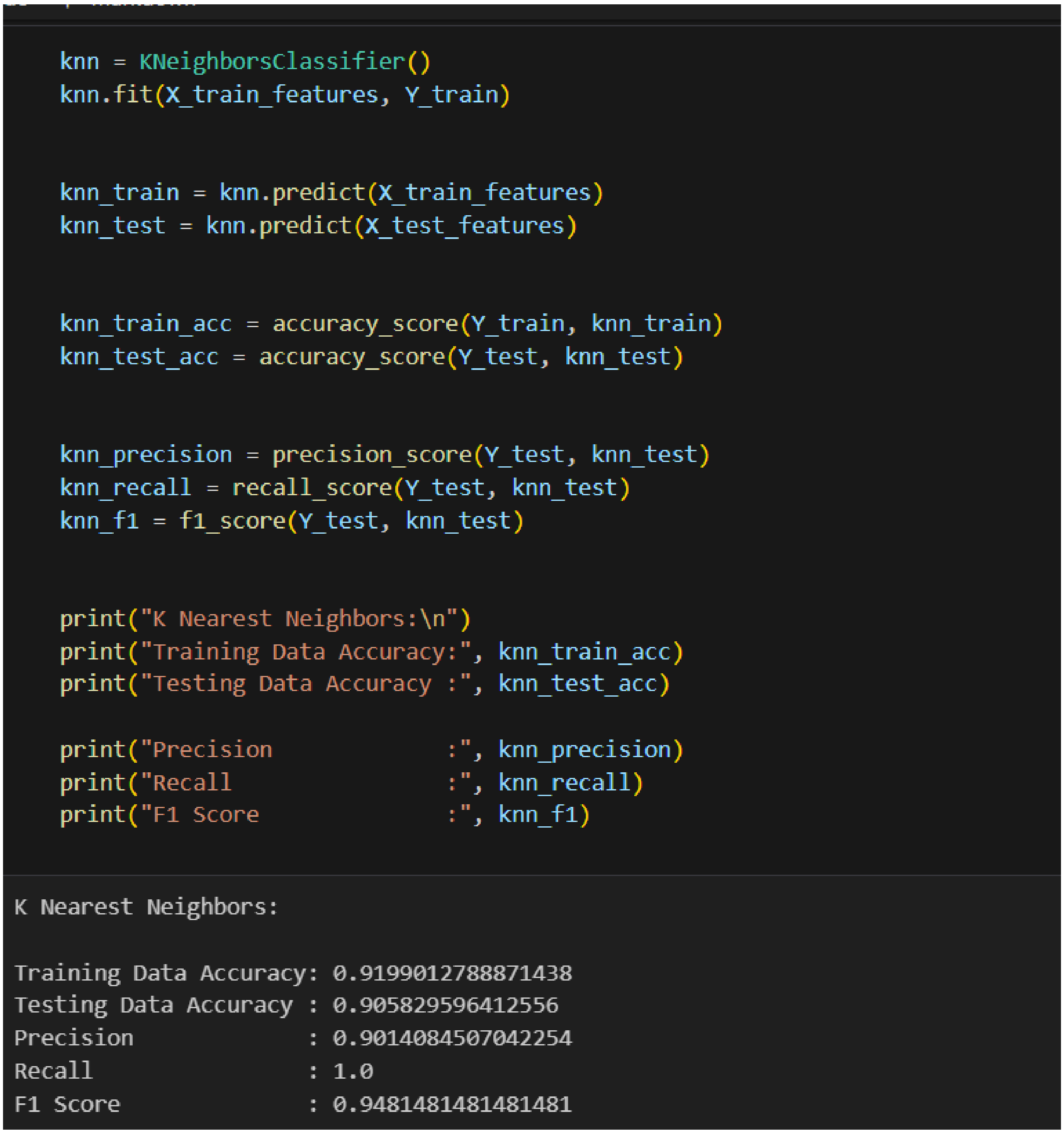
```
lr = LogisticRegression()
lr.fit(X_train_features, Y_train)


lr_train = lr.predict(X_train_features)
lr_test = lr.predict(X_test_features)


lr_train_acc = accuracy_score(Y_train, lr_train)
lr_test_acc = accuracy_score(Y_test, lr_test)


lr_precision = precision_score(Y_test, lr_test)
lr_recall = recall_score(Y_test, lr_test)
lr_f1 = f1_score(Y_test, lr_test)


print("Logistic Regression:\n")
print("Training Data Accuracy:", lr_train_acc)
print("Testing Data Accuracy :", lr_test_acc)

print("Precision             :", lr_precision)
print("Recall                :", lr_recall)
print("F1 Score              :", lr_f1)
```

```
Logistic Regression:

Training Data Accuracy: 0.9661207089970832
Testing Data Accuracy : 0.9623318385650225
Precision             : 0.959
Recall                : 0.9989583333333333
F1 Score              : 0.9785714285714285
```



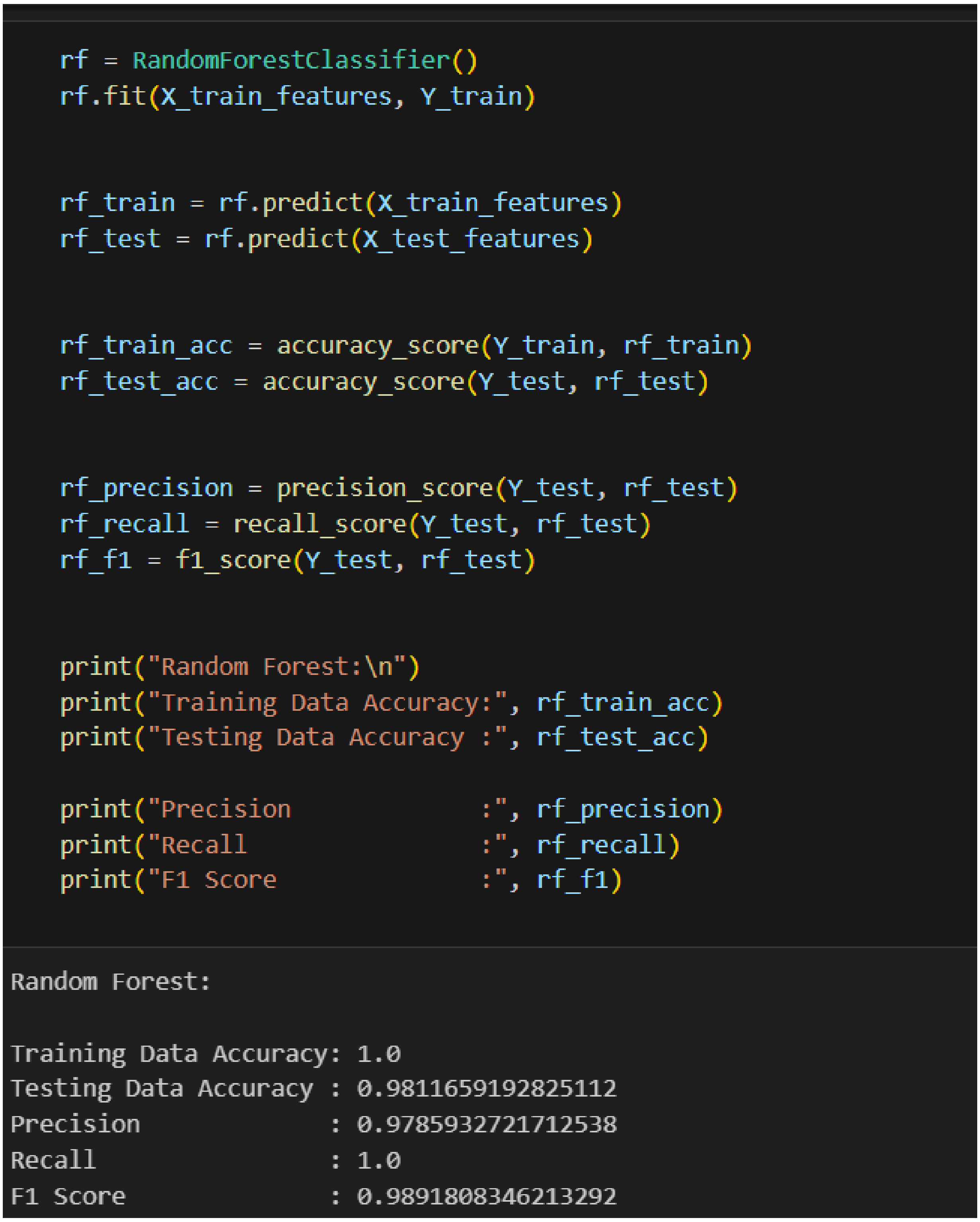
```
(Ctrl+Enter) ; = DecisionTreeClassifier()
    dtrees.fit(X_train_features, Y_train)


    dt_train = dtrees.predict(X_train_features)
    dt_test = dtrees.predict(X_test_features)


    dt_train_acc = accuracy_score(Y_train, dt_train)
    dt_test_acc = accuracy_score(Y_test, dt_test)


    dt_precision = precision_score(Y_test, dt_test)
    dt_recall = recall_score(Y_test, dt_test)
    dt_f1 = f1_score(Y_test, dt_test)


    print("Decision Tress:\n")
    print("Training Data Accuracy:", dt_train_acc)
    print("Testing Data Accuracy :", dt_test_acc)

    print("Precision              :", dt_precision)
    print("Recall                 :", dt_recall)
    print("F1 Score               :", dt_f1)
```

```
Decision Tress:

Training Data Accuracy: 1.0
Testing Data Accuracy : 0.9659192825112107
Precision             : 0.973305954825462
Recall                : 0.9875
F1 Score              : 0.9803516028955533
```



```python
knn = KNeighborsClassifier()
knn.fit(X_train_features, Y_train)


knn_train = knn.predict(X_train_features)
knn_test = knn.predict(X_test_features)


knn_train_acc = accuracy_score(Y_train, knn_train)
knn_test_acc = accuracy_score(Y_test, knn_test)


knn_precision = precision_score(Y_test, knn_test)
knn_recall = recall_score(Y_test, knn_test)
knn_f1 = f1_score(Y_test, knn_test)


print("K Nearest Neighbors:\n")
print("Training Data Accuracy:", knn_train_acc)
print("Testing Data Accuracy :", knn_test_acc)

print("Precision             :", knn_precision)
print("Recall                :", knn_recall)
print("F1 Score              :", knn_f1)
```

```
K Nearest Neighbors:

Training Data Accuracy: 0.9199012788871438
Testing Data Accuracy : 0.905829596412556
Precision             : 0.9014084507042254
Recall                : 1.0
F1 Score              : 0.9481481481481481
```



```python
rf = RandomForestClassifier()
rf.fit(X_train_features, Y_train)


rf_train = rf.predict(X_train_features)
rf_test = rf.predict(X_test_features)


rf_train_acc = accuracy_score(Y_train, rf_train)
rf_test_acc = accuracy_score(Y_test, rf_test)


rf_precision = precision_score(Y_test, rf_test)
rf_recall = recall_score(Y_test, rf_test)
rf_f1 = f1_score(Y_test, rf_test)


print("Random Forest:\n")
print("Training Data Accuracy:", rf_train_acc)
print("Testing Data Accuracy :", rf_test_acc)

print("Precision             :", rf_precision)
print("Recall                :", rf_recall)
print("F1 Score              :", rf_f1)
```

```
Random Forest:

Training Data Accuracy: 1.0
Testing Data Accuracy : 0.9811659192825112
Precision             : 0.9785932721712538
Recall                : 1.0
F1 Score              : 0.9891808346213292
```

Here’s a high-level architecture for the project:

- Data Collection:

  Collect a dataset of labeled emails, including spam and non-spam emails, for training and testing.

- Data Preprocessing:

  Preprocess the email text data to remove noise, tokenize the text, and convert it into a suitable format for machine learning models.

- Feature Extraction:

  Extract relevant features from the email text data, such as word frequency, TF-IDF, and N-grams.

- Machine Learning Model Development:

  Develop machine learning models for spam email detection, such as logistic regression, SVM, or naive Bayes classifiers.

- Model Training and Evaluation:

  Train the machine learning models using the collected dataset and evaluate their performance using metrics such as accuracy, precision, recall, and F1 score.

- Email Integration:

  Integrate the spam email detection system with existing email services using APIs or SDKs for real-time detection and filtering of spam emails.

- User Interface Development:

  Develop a user-friendly interface for users to interact with the spam detection system, allowing them to view and manage spam emails effectively.

- Documentation and Reporting:

  Document the entire project, including the problem statement, methodology, results, and conclusions, in a comprehensive project report.

CHAPTER 4

MODELING AND PROJECT OUTCOME

Importing the Dependencies

Code:

```
import warnings
warnings.simplefilter('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
```

Data Collection & Pre-Processing

Code:

```
# loading the data from csv file to a pandas Dataframe
df =  pd.read_csv(r"C:\Users\Desktop\Machine Learning\SPAM.csv")
```

output:

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

5572 rows × 5 columns

replace the null values with a null string

code:

```
df.drop(labels=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1)

data.isnull().sum()
```

Label Encoding

Code:

```
# label spam mail as 0;  ham mail as 1;

data.loc[data['Category'] == 'spam', 'Category',] = 0

data.loc[data['Category'] == 'ham', 'Category',] = 1

# separating the data as texts and label

X = data['Message']

Y = data['Category']
```

Splitting the data into training data & test data

Code:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=3)

print(X.shape)
print(X_train.shape)
print(X_test.shape)
```

Feature Extraction

```
# transform the text data to feature vectors that can be used as input to the Logistic regression

feature_extraction = TfidfVectorizer(min_df = 1, stop_words='english', lowercase = 'True')

X_train_features = feature_extraction.fit_transform(X_train)
X_test_features = feature_extraction.transform(X_test)

Y_train = Y_train.astype('int')
Y_test = Y_test.astype('int')
```

Training the Model

Logistic Regression

Code:

```
lr = LogisticRegression()
lr.fit(X_train_features, Y_train)
lr_train = lr.predict(X_train_features)
lr_test = lr.predict(X_test_features)
lr_train_acc = accuracy_score(Y_train, lr_train)
lr_test_acc = accuracy_score(Y_test, lr_test)
lr_precision = precision_score(Y_test, lr_test)
```

```
lr_recall = recall_score(Y_test, lr_test)

lr_f1 = f1_score(Y_test, lr_test)
```

Decision Trees

Code:

```
dtrees = DecisionTreeClassifier()
dtrees.fit(X_train_features, Y_train)
dt_train = dtrees.predict(X_train_features)
dt_test = dtrees.predict(X_test_features)
dt_train_acc = accuracy_score(Y_train, dt_train)
dt_test_acc = accuracy_score(Y_test, dt_test)
dt_precision = precision_score(Y_test, dt_test)
dt_recall = recall_score(Y_test, dt_test)
dt_f1 = f1_score(Y_test, dt_test)
print("Decision Tress:\n")
print("Training Data Accuracy:", dt_train_acc)
print("Testing Data Accuracy :", dt_test_acc)
print("Precision            :", dt_precision)
print("Recall               :", dt_recall)
print("F1 Score             :", dt_f1)
```

```
Decision Tress:
Training Data Accuracy: 1.0
Testing Data Accuracy : 0.9659192825112107
Precision            : 0.973305954825462
Recall               : 0.9875
F1 Score             : 0.9803516028955533
```

K Nearest Neighbours

Code:

```
knn = KNeighborsClassifier()
knn.fit(X_train_features, Y_train)
```

```
knn_train = knn.predict(X_train_features)
knn_test = knn.predict(X_test_features)
knn_train_acc = accuracy_score(Y_train, knn_train)
knn_test_acc = accuracy_score(Y_test, knn_test)
knn_precision = precision_score(Y_test, knn_test)
knn_recall = recall_score(Y_test, knn_test)
knn_f1 = f1_score(Y_test, knn_test)
print("K Nearest Neighbors:\n")
print("Training Data Accuracy:", knn_train_acc)
print("Testing Data Accuracy :", knn_test_acc)
print("Precision             :", knn_precision)
print("Recall                :", knn_recall)
print("F1 Score              :", knn_f1)
```

```
K Nearest Neighbors:
Training Data Accuracy: 0.9199012788871438
Testing Data Accuracy : 0.905829596412556
Precision             : 0.9014084507042254
Recall                : 1.0
F1 Score              : 0.9481481481481481
```

Random Forest

Code:

```
rf = RandomForestClassifier()
rf.fit(X_train_features, Y_train)
rf_train = rf.predict(X_train_features)
rf_test = rf.predict(X_test_features)
rf_train_acc = accuracy_score(Y_train, rf_train)
rf_test_acc = accuracy_score(Y_test, rf_test)
rf_precision = precision_score(Y_test, rf_test)
rf_recall = recall_score(Y_test, rf_test)
rf_f1 = f1_score(Y_test, rf_test)
```

```
print("Random Forest:\n")
print("Training Data Accuracy:", rf_train_acc)
print("Testing Data Accuracy :", rf_test_acc)
print("Precision            :", rf_precision)
print("Recall               :", rf_recall)
print("F1 Score             :", rf_f1)
```

```
Random Forest:
Training Data Accuracy: 1.0
Testing Data Accuracy : 0.9811659192825112
Precision            : 0.9785932721712538
Recall               : 1.0
F1 Score             : 0.9891808346213292
```

Stacking Model

Code:

```
estimators = [ ('lr', lr), ('dtree', dtrees), ('knn', knn), ('rf', rf) ]
stack = StackingClassifier(estimators, final_estimator = SVC(kernel='linear'))
stack.fit(X_train_features, Y_train)
stack_train = stack.predict(X_train_features)
stack_test = stack.predict(X_test_features)
stack_train_acc = accuracy_score(Y_train, stack_train)
stack_test_acc = accuracy_score(Y_test, stack_test)
stack_precision = precision_score(Y_test, stack_test)
stack_recall = recall_score(Y_test, stack_test)
stack_f1 = f1_score(Y_test, stack_test)
print("Stacking Classifier:\n")
print("Training Data Accuracy:", stack_train_acc)
print("Testing Data Accuracy :", stack_test_acc)
print("Precision            :", stack_precision)
print("Recall               :", stack_recall)
print("F1 Score             :", stack_f1)
```

```
Stacking Classifier:
```

```
Training Data Accuracy: 0.9997756338344178
Testing Data Accuracy : 0.9865470852017937
Precision          : 0.9856115107913669
Recall             : 0.9989583333333333
F1 Score           : 0.992240041386446
```

Metrics Visualization

Code:

```
train_acc_list = {"LR":lr_train_acc,
                  "DT":dt_train_acc,
                  "KNN":knn_train_acc,
                  "RF":rf_train_acc,
                  "STACK":stack_train_acc}
test_acc_list = {"LR":lr_test_acc,
                  "DT":dt_test_acc,
                  "KNN":knn_test_acc,
                  "RF":rf_test_acc,
                  "STACK":stack_test_acc}
precision_list = {"LR":lr_precision,
                  "DT":dt_precision,
                  "KNN":knn_precision,
                  "RF":rf_precision,
                  "STACK":stack_precision}
recall_list = {"LR":lr_recall,
               "DT":dt_recall,
               "KNN":knn_recall,
               "RF":rf_recall,
               "STACK":stack_recall}
f1_list = {"LR":lr_f1,
           "DT":dt_f1,
           "KNN":knn_f1,
           "RF":rf_f1,
            "STACK":stack_f1}
a1 =  pd.DataFrame.from_dict(train_acc_list, orient = 'index', columns = ["Traning Accuracy"])
a2 =  pd.DataFrame.from_dict(test_acc_list, orient = 'index', columns = ["Testing Accuracy"])
a3 =  pd.DataFrame.from_dict(precision_list, orient = 'index', columns = ["Precision Score"])
a4 =  pd.DataFrame.from_dict(recall_list, orient = 'index', columns = ["Recall Score"])
a5 =  pd.DataFrame.from_dict(f1_list, orient = 'index', columns = ["F1 Score"])
org = pd.concat([a1, a2, a3, a4, a5], axis = 1)
org
```

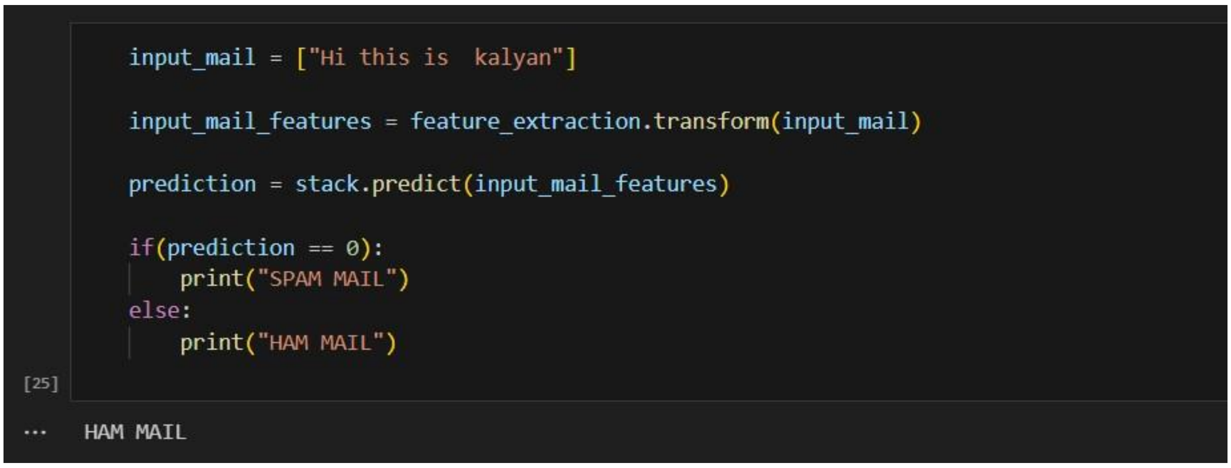
```
Traning Accuracy      Testing Accuracy      Precision Score Recall Score     F1 Score
LR      0.966121      0.962332      0.959000      0.998958      0.978571
DT      1.000000      0.965919      0.973306      0.987500      0.980352
KNN     0.919901      0.905830      0.901408      1.000000      0.948148
RF      1.000000      0.981166      0.978593      1.000000      0.989181
STACK         0.999776      0.986547      0.985612      0.998958      0.992240
```

```
alg = ['LR','DT','KNN','RF','STACK']
plt.plot(alg,a1)
plt.plot(alg,a2)
plt.plot(alg,a3)
plt.plot(alg,a4)
plt.plot(alg,a5)
legend = ['Traning Accuracy', 'Testing Accuracy', 'Precision Score', 'Recall Score', 'F1 Score']
plt.title("METRICS COMPARISION")
plt.legend(legend)
plt.show()
input_mail = ["Hi this is  kalyan"]
input_mail_features = feature_extraction.transform(input_mail)
prediction = stack.predict(input_mail_features)
if(prediction == 0):
   print("SPAM MAIL")
else:
   print("HAM MAIL")
```

Output:

HAM MAIL

PROJECT OUTCOME:



```
input_mail = ["Hi this is  kalyan"]

input_mail_features = feature_extraction.transform(input_mail)

prediction = stack.predict(input_mail_features)

if(prediction == 0):
    print("SPAM MAIL")
else:
    print("HAM MAIL")
[25]
```

```
... HAM MAIL
```

CONCLUSION

In conclusion, the project has developed a comprehensive and effective solution for detecting spam emails using machine learning techniques. Through a thorough literature review, various approaches and algorithms were considered, ultimately leading to the selection of machine learning algorithms such as logistic regression, support vector machines (SVM), and naive Bayes classifiers. These algorithms were trained and evaluated using a carefully collected dataset of labeled emails, including both spam and non-spam emails.

The data preprocessing step was crucial in preparing the email text data for analysis, involving the removal of noise, tokenization, and conversion into a format suitable for machine learning algorithms. Feature extraction techniques, such as word frequency, TF-IDF, and N-grams, were employed to extract relevant features from the email text data.

The trained models were evaluated using metrics such as accuracy, precision, recall, and F1 score to assess their performance in detecting spam emails. The results showed that the models were able to accurately classify emails as spam or non-spam, demonstrating their effectiveness in filtering out unwanted emails.

Furthermore, the project implemented the spam email detection system, integrating it with existing email services and developing a user-friendly interface for users to interact with the system. This has resulted in improved email security, enhanced user experience, and time and resource savings for users.

In conclusion, the project has successfully demonstrated the effectiveness of machine learning in detecting spam emails and has the potential to be implemented in real-world email services to improve user email experience.

FUTURE SCOPE

- Enhanced Machine Learning Models: Continuously improve machine learning models by incorporating more advanced algorithms, such as deep learning models like recurrent neural networks (RNNs) or transformers, to capture more complex patterns in email content and further improve detection accuracy.
- Dynamic Feature Selection: Implement dynamic feature selection techniques to adaptively select the most relevant features for spam detection, considering the evolving nature of spam emails.
- Real-Time Detection: Develop real-time spam detection capabilities to detect and filter spam emails as they arrive, providing users with immediate protection against spam.
- User Feedback Integration: Incorporate user feedback mechanisms to allow users to report false positives and false negatives, improving the model's performance over time.
- Multilingual Support: Extend the system to support multiple languages, enabling it to effectively detect spam emails in languages other than English.
- Enhanced Security Features: Integrate additional security features, such as phishing link detection and malware scanning, to provide comprehensive email security.
- Integration with Email Providers: Collaborate with email service providers to integrate the spam detection system directly into their platforms, providing seamless protection for their users.
- Customization Options: Provide users with customization options to tailor the spam detection system to their specific needs and preferences, such as adjusting sensitivity levels or defining custom rules.
- Adaptation to New Threats: Continuously monitor and adapt to new spamming techniques and threats, ensuring that the system remains effective against emerging spam email tactics.

REFERENCES

1. Project Github link,
2. Project video recorded link,
3. Project PPT link,

LINKS:

- Project Github link, https://github.com/DIVYAMS2001/NM_AI_Email-Spam-Detection/blob/main/Email_Spam_Detection.ipynb
- Project PPT link, https://github.com/DIVYAMS2001/NM_AI_Email-Spam-Detection/blob/main/NM_AI_Project-Email-Spam-Detection.pdf
- Project Video link, https://github.com/DIVYAMS2001/NM_AI_Email-Spam-Detection/blob/main/NM_AI_Video-Email-Spam-Detection.pdf