Email Spam Classifier

This model will help us filter out unwanted and potentially harmful emails from our inbox. We will follow standard data science procedures, including data loading, preprocessing, feature extraction, model training, evaluation, and prediction, to achieve this goal.

Importing Necessary Libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Load and Explore the Dataset

```
# Load the dataset
df = pd.read_csv("/kaggle/input/sms-spam-collection-dataset/spam.csv", encoding='ISO-8859-1')


# Display the first few rows of the dataset
df.head()
```

Data Preprocessing

```
# Display the column names of the DataFrame
print(df.columns)


# Convert 'spam' and 'ham' to binary labels
```

```
df['v1'] = df['v1'].map({'spam': 0, 'ham': 1})

# Split the data into features (X) and target (Y)

X = df["v2"]

Y = df["v1"]

# Split the data into training and test sets

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.35,

random_state=3)
```

Feature Extraction - TF-IDF

```
# TF-IDF feature extraction

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

= tfidf_vectorizer.fit_transform(X_train)

X_test_features = tfidf_vectorizer.transform(X_test)
```

Model Training (Random Forest)

```
# Model training

model = RandomForestClassifier(n_estimators=100, random_state=3)

model.fit(X_train_features, Y_train)
```

Model Evaluation (Random Forest)

```
prediction_on_training_data = model.predict(X_train_features)

accuracy_on_training_data = accuracy_score(Y_train, prediction_on_training_data)

prediction_on_test_data = model.predict(X_test_features)

accuracy_on_test_data = accuracy_score(Y_test, prediction_on_test_data)


#Print accuracy


print('Accuracy on training data: {:.2f} %'.format(accuracy_on_training_data * 100))

print('Accuracy on test data: {:.2f} %'.format(accuracy_on_test_data * 100))
```

Confusion Matrix Visualization(Random Forest Classifier)

```
# Confusion Matrix Visualization

conf_matrix = confusion_matrix(Y_test, prediction_on_test_data) plt.figure(figsize=(8, 6))
```

```
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Spam', 'Ham'], yticklabels=['Spam', 'Ham'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Classification Report (Random Forest Classifier)

```
classification_rep = classification_report(Y_test, prediction_on_test_data, target_names=['Spam', 'Ham'])
print("Classification Report:")
print(classification_rep)
```

Feature Importance Visualization (Random Forest)

```
feature_importance = model.feature_importances_
feature_names = tfidf_vectorizer.get_feature_names_out()
sorted_idx = np.argsort(feature_importance)[-20:]  # Top 20 important features

plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align="center")
plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx])
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("Top 20 Important Features (Random Forest)")
plt.show()
```

Make Predictions on New Input (Random Forest Classifier)

```
input_your_mail = "Keep yourself safe for me because I need you and I miss you already and I envy
everyone that see's you in real life"
input_data_features = tfidf_vectorizer.transform([input_your_mail])
prediction = model.predict(input_data_features)

if prediction[0] == 1:
    print("Ham Mail")
else:
    print("Spam Mail")
```

```python
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Load the dataset
df = pd.read_csv("spam.csv", encoding='ISO-8859-1')

# Display the first few rows of the dataset
df.head()

# Display the column names of the DataFrame
print(df.columns)

# Convert 'spam' and 'ham' to binary labels
df['v1'] = df['v1'].map({'spam': 0, 'ham': 1})

# Split the data into features (X) and target (Y)
X = df["v2"]
Y = df["v1"]

# Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.35,
random_state=3)

# TF-IDF feature extraction
tfidf_vectorizer = TfidfVectorizer(min_df=1, stop_words='english',
lowercase=True)
X_train_features = tfidf_vectorizer.fit_transform(X_train)
X_test_features = tfidf_vectorizer.transform(X_test)

# Model training
model = RandomForestClassifier(n_estimators=100, random_state=3)
model.fit(X_train_features, Y_train)

prediction_on_training_data = model.predict(X_train_features)
accuracy_on_training_data = accuracy_score(Y_train,
prediction_on_training_data)

prediction_on_test_data = model.predict(X_test_features)
accuracy_on_test_data = accuracy_score(Y_test, prediction_on_test_data)
```

```
#Print accuracy

print('Accuracy on training data: {:.2f} %'.format(accuracy_on_training_data *
100))
print('Accuracy on test data: {:.2f} %'.format(accuracy_on_test_data * 100))

# Confusion Matrix Visualization
conf_matrix = confusion_matrix(Y_test, prediction_on_test_data)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Spam', 'Ham'], yticklabels=['Spam', 'Ham'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

classification_rep = classification_report(Y_test, prediction_on_test_data,
target_names=['Spam', 'Ham'])
print("Classification Report:")
print(classification_rep)

feature_importance = model.feature_importances_
feature_names = tfidf_vectorizer.get_feature_names_out()
sorted_idx = np.argsort(feature_importance)[-20:]  # Top 20 important features

plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx],
align="center")
plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx])
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("Top 20 Important Features (Random Forest)")
plt.show()

input_your_mail = "Keep yourself safe for me because I need you and I miss you
already and I envy everyone that see's you in real life"
input_data_features = tfidf_vectorizer.transform([input_your_mail])
prediction = model.predict(input_data_features)
if prediction[0] == 1:
    print("Ham Mail")
else:
    print("Spam Mail")
```

Transforming Design into an Innovative Spam Classifier

Introduction:

This document outlines the detailed steps for transforming the design of a smarter AI-powered spam classifier into an innovative solution that effectively addresses the problem of spam detection. Spam emails, messages, and unwanted content continue to inundate our inboxes and online spaces. Traditional spam filters have made significant strides in reducing the onslaught, but as spammers evolve, so must our defenses. This is where smarter AI-powered spam classifiers come into play. Leveraging advanced machine learning techniques and natural language processing, these cutting-edge solutions aim to not only filter out unwanted content but also adapt and learn from new spam tactics, ensuring a more effective and efficient spam detection and prevention system.

Step 1: Data Collection and Preparation

Data Collection:

- Source Selection: Choose reliable sources for your data, whether it's from databases, surveys, web scraping, or sensors.
- Data Types: Identify the types of data you're working with (e.g., structured, unstructured, text, images, time-series).
- Data Volume: Determine how much data you need for your analysis or model.

Data Cleaning:

- Handling Missing Data: Decide on a strategy for dealing with missing values (e.g., imputation, removal).
- Duplicate Removal: Eliminate duplicate records to ensure data integrity.
- Outlier Detection: Identify and handle outliers that can skew results.

Data Transformation:

- Feature Engineering: Create new features from existing data that may be more informative for your task.
- Scaling and Normalization: Standardize numerical features to have a consistent scale.
- Categorical Encoding: Convert categorical data into numerical format (e.g., one-hot encoding).

Data Exploration:

- Descriptive Statistics: Calculate summary statistics to understand the data's central tendencies and variability.
- Data Visualization: Create plots and charts to visualize patterns and relationships in the data.

Data Splitting:

- Training and Testing Data: Split the dataset into training and testing sets to evaluate model performance.

Data Preprocessing:
- Data Standardization: Apply preprocessing steps consistently to training and testing data.
- Dimensionality Reduction: If necessary, reduce the number of features to avoid the curse of dimensionality.

Dealing with Imbalanced Data:
- In classification tasks, address class imbalance using techniques like oversampling or undersampling.

Data Quality and Ethics:
- Ensure data privacy and comply with ethical standards, especially when dealing with personal or sensitive data.

Version Control:
- Maintain a record of changes made to the dataset to track data versioning and reproducibility.

Documentation:
- Keep detailed records of data preprocessing steps, as well as any insights or challenges encountered during the process.

Data Storage:
- Decide how and where to store the prepared data for easy access and future use.

Iterative Process:
- Data preparation is often an iterative process, and adjustments may be needed as you analyze the data and build models.

Step 2: Model Selection

Goal-Oriented: Model selection is driven by the specific goals of your machine learning task, whether it's classification, regression, clustering, or something else.

Data-Driven: The choice of model should be heavily influenced by the characteristics of your dataset. Small or large, clean or noisy data will impact your decision.

Complexity vs. Simplicity: Consider the trade-off between model complexity and simplicity. Simpler models are often preferred due to Occam's razor, but sometimes complex models are necessary for high accuracy.

Overfitting and Regularization: Guard against overfitting by using regularization techniques. This is particularly important when using complex models.

Algorithm Options: You have a wide range of algorithms to choose from, including linear regression, decision trees, random forests, support vector machines, neural networks, and more.

Cross-Validation: Employ cross-validation methods to assess how well your chosen model generalizes to unseen data.

Hyperparameter Tuning: Fine-tune model hyperparameters to optimize performance. Techniques like grid search and random search are common for this.

Ensemble Methods: Ensemble models can combine the strengths of multiple models to improve accuracy and robustness.

Interpretability vs. Performance: Depending on your application, you may need to balance model interpretability with performance. Simpler models tend to be more interpretable.

Domain Expertise: Incorporate domain knowledge when selecting a model. Domain-specific insights can guide you to the most suitable choice.

Step 3: Model Development

Problem Definition: Clearly define the problem you want to solve. Understand the goals, constraints, and metrics for success.

Feature Engineering: Create meaningful features from the data. Feature selection and extraction can significantly impact model performance.

Model Selection: Choose an appropriate machine learning algorithm or model architecture based on the nature of your problem (e.g., regression, classification, deep learning, etc.).

Model Training: Train the model using the training data. Optimize hyperparameters and monitor for overfitting or underfitting.

Validation: Use the validation set to fine-tune the model and assess its performance. Adjust as needed.

Evaluation: Evaluate the model's performance using appropriate metrics (e.g., accuracy, F1-score, RMSE, etc.) on the test set.

Hyperparameter Tuning: Tune hyperparameters to optimize the model's performance. Techniques like grid search or random search can be helpful.

Model Interpretability: Understand how the model makes predictions, especially for critical applications. Tools like SHAP values and LIME can help with interpretability.

Deployment: If applicable, deploy the model in a real-world environment. This may involve containerization, setting up APIs, or integrating with other systems.

Monitoring and Maintenance: Continuously monitor the model's performance in production and retrain it periodically as new data becomes available.

Step 4: Evaluation and Validation

Evaluation and validation are crucial processes in various fields, such as research, software development, and data analysis.

Evaluation:

Define Clear Objectives: Clearly state the goals and objectives you want to achieve through the evaluation process.

Choose Appropriate Metrics: Select relevant metrics or criteria to measure the performance or success of the system, project, or product.

Data Collection: Gather data using appropriate methods, such as surveys, observations, or automated tools, depending on the context.

Data Analysis: Analyze the collected data to draw meaningful conclusions and insights. Statistical analysis and visualization tools can be helpful.

Interpret Results: Interpret the results in the context of your objectives and metrics. Are you meeting your goals? What do the numbers or feedback mean?

Iterate and Improve: Use the evaluation results to make improvements or adjustments as needed. It's often an iterative process.

Validation:

Define Validation Criteria: Specify the criteria that a system or product must meet to be considered valid or acceptable.

Testing and Verification: Use testing methods to verify that the system or product conforms to the specified criteria. This can involve various testing types (functional, performance, security, etc.).

Validation Testing: Conduct validation testing to ensure that the system or product meets user or stakeholder needs and expectations.

User Feedback: Collect feedback from users or stakeholders to validate whether the system or product is meeting their requirements.

Compliance and Standards: Ensure that the system complies with relevant industry standards and regulations.

Documentation: Maintain clear and comprehensive documentation of the validation process and results for transparency and future reference.

Revalidation: Periodically revalidate systems or products to ensure they continue to meet their intended purposes and remain relevant.

Step 5: Continuous Learning

Continuous learning is a commitment to lifelong education and skill development.

Lifelong Process: Continuous learning is not limited to formal education but encompasses ongoing self-improvement throughout one's life.

Professional Growth: It's essential for career development, as industries and technologies evolve rapidly.

Various Forms: Learning can take various forms, including online courses, workshops, reading, mentoring, and on-the-job experiences.

Adaptability: Continuous learning fosters adaptability, helping individuals stay relevant and effective in changing environments.

Personal Motivation: Self-motivation and discipline are crucial for maintaining a continuous learning habit.

Technology's Role: Technology plays a significant role in enabling continuous learning, with e-learning platforms, webinars, and educational apps.

Goal-Driven: Setting clear goals and objectives for learning helps individuals stay focused and measure progress.

Feedback Loop: Regularly seeking feedback and reflecting on one's learning journey is essential for improvement.

Networking: Building a network of mentors and peers can provide valuable support and opportunities for learning.

Benefits: Continuous learning leads to personal growth, increased job opportunities, and a better understanding of the world.

Step 6: Model Interpretability

Model interpretability is the ability to understand and explain how a machine learning model makes predictions. It's a critical aspect of deploying machine learning models in real-world applications, and it can be achieved through various methods and techniques.

Feature Importance: Understanding which features or variables have the most significant influence on a model's predictions is essential. Feature importance techniques, such as permutation feature importance or SHAP values, help quantify this.

Local vs. Global Interpretability: Interpretability can be applied at both the local and global levels. Local interpretability explains an individual prediction, while global interpretability focuses on understanding the model's overall behavior.

Feature Visualization: Visualizing data and model outputs can be powerful. Techniques like partial dependence plots, LIME (Local Interpretable Model-agnostic Explanations), and saliency maps help visualize the relationship between features and predictions.

Simpler Models: Using simpler models like linear regression or decision trees instead of complex ones like deep neural networks can inherently provide more interpretability.

Model-Specific Techniques: Some models, like decision trees, can be directly interpreted. For instance, you can trace a decision tree to understand how a prediction is made.

SHAP Values: SHAP (SHapley Additive exPlanations) is a widely used method for explaining individual predictions by quantifying the contribution of each feature to a prediction.

Ethical Considerations: Interpretability is crucial for detecting and mitigating bias in models, ensuring fairness, and addressing ethical concerns related to AI and machine learning.

Domain Knowledge: Incorporating domain knowledge can enhance interpretability. Experts in a particular field can help make sense of the model's behavior and decisions.

Tools and Libraries: Various tools and libraries, such as SHAP, LIME, and Interpretable ML, are available to assist in model interpretability.

Regulatory Requirements: In some industries, there are legal and regulatory requirements for model interpretability and transparency, such as GDPR in Europe.

Trade-offs: There is often a trade-off between model complexity and interpretability. Simplifying a model for better interpretability may result in reduced predictive performance.

Step 7: Deployment

Deployment is the process of making a software application or system operational in a specific environment.

- Types: Deployment can occur on-premises, in the cloud, or on edge devices.

- Models: Common deployment models include on-premises, IaaS, PaaS, and SaaS.

- CI/CD: CI/CD pipelines automate the deployment process for efficient and consistent releases.

- Strategies: Strategies like Blue-Green, Canary, and Rolling deployments help manage version releases.

- Version Control: Use version control systems to track changes in source code and configurations.

- Testing: Rigorous testing, including unit, integration, and user acceptance tests, is vital.

- Scalability: Consider how the application will scale to handle increased loads.

- Security: Implement security measures to protect against vulnerabilities and data breaches.

- Monitoring: Real-time monitoring and logging are crucial for issue detection and troubleshooting.

- Rollback Plan: Always have a plan for reverting to a previous version in case of deployment failures.

- Documentation: Detailed documentation is essential for replication and understanding of the deployment process.

- Compliance: Ensure deployment adheres to industry and legal standards (e.g., GDPR, HIPAA).

- Maintenance: Plan for regular software maintenance and updates to keep it secure and up to date.

- User Support: Provide user training and support as necessary for new software or updates.

- Feedback: Gather user feedback post-deployment for continuous improvement.

- Variability: Deployment processes vary depending on technology stack and target environment.

Step 8: Monitoring and Maintenance

Monitoring and maintenance are crucial for the smooth operation of various systems and equipment.

Regular Inspections: Establish a schedule for routine inspections to identify potential issues before they become major problems.

Documentation: Maintain detailed records of all maintenance activities, including dates, tasks performed, and any replacements or repairs.

Predictive Maintenance: Implement predictive maintenance techniques using data and sensors to anticipate when equipment may fail.

Emergency Protocols: Develop clear procedures for responding to unexpected breakdowns or emergencies to minimize downtime.

Spare Parts: Keep an inventory of critical spare parts to reduce downtime and expedite repairs.

Training: Ensure that maintenance personnel are properly trained to perform their tasks safely and efficiently.

Environmental Factors: Consider environmental conditions that may impact equipment, such as temperature, humidity, and exposure to corrosive elements.

Budgeting: Allocate funds for maintenance and monitoring, understanding that preventive maintenance often saves costs in the long run.

Software Solutions: Use maintenance management software to streamline and track maintenance activities.

Safety: Prioritize safety in all maintenance activities to protect both personnel and equipment.

Step 9: Legal and Ethical Considerations

Legal Considerations:

Compliance with Laws: Ensure that your actions, whether in business or personal matters, adhere to all relevant local, national, and international laws and regulations.

Privacy: Respect individuals' privacy by handling their personal data in accordance with data protection laws (e.g., GDPR in Europe, HIPAA in the US).

Intellectual Property: Avoid infringing on intellectual property rights, such as patents, copyrights, and trademarks, and respect licensing agreements.

Contracts: Honor the terms of any agreements or contracts you enter into, and seek legal advice if necessary.

Employment Law: Comply with labor laws when hiring and managing employees, including fair wages, working conditions, and anti-discrimination measures.

Ethical Considerations:

Transparency: Be honest and open in your communications and actions, and disclose any potential conflicts of interest.

Fair Treatment: Treat all individuals fairly and without discrimination, irrespective of factors like race, gender, or religion.

Environmental Responsibility: Consider the environmental impact of your actions and strive to minimize harm through sustainable practices.

Social Responsibility: Engage in philanthropy or socially responsible initiatives that contribute positively to the community.

Confidentiality: Respect the confidentiality of sensitive information and maintain the trust of clients, customers, and colleagues.

Avoid Harm: Strive to do no harm and take precautions to prevent harm, especially in industries where safety is a concern.

Step 10: User Experience

User-Centered Design: UX design starts by focusing on the needs and preferences of the end users. It involves understanding user behaviors, expectations, and pain points.

Usability: Usability is a critical component of UX. It pertains to how easy and efficient it is for users to achieve their goals when using a product or service.

User Interface (UI) Design: UI design focuses on the specific visual and interactive elements of a product, including buttons, menus, and forms.

Aesthetics and Visual Design: Visual appeal can significantly impact the user experience. Good design, including layout, colors, and typography, can enhance the overall UX.

User Feedback: Gathering feedback from users through surveys, interviews, or analytics is crucial for understanding their needs and making improvements.

Mobile and Responsive Design: With the prevalence of mobile devices, designing for various screen sizes and ensuring responsive layouts is essential.

Performance Optimization: Slow-loading websites or applications can lead to a poor user experience. Optimizing performance is critical.

Iterative Design: UX design is an iterative process that involves continuous improvement based on user feedback and testing.

User Testing: Conducting usability testing and user testing helps evaluate how well a product meets user expectations and needs.

Ethical Considerations: UX designers should consider the ethical implications of their work, including data privacy and user well-being.

Content Strategy: Delivering the right content at the right time to users is integral to a positive user experience.

Conclusion:

Transforming the design of a smarter AI-powered spam classifier involves leveraging advanced machine learning techniques, including deep learning and natural language processing, to enhance its accuracy. This document provides a comprehensive roadmap for successfully implementing the spam classifier, Additionally, ongoing training and adaptation to new spam patterns are crucial. This evolution aims to provide users with more effective protection against spam and ensure a cleaner digital communication environment.

Building a Smarter AI-Powered Spam Classifier

The use of internet has been extensively increasing over the past decade and it continues to be on the ascent. Hence it is apt to say that the Internet is gradually becoming an integral part of everyday life. Internet usage is expected to continue growing and e-mail has become a powerful tool intended for idea and information exchange. Negligible time delay during transmission, security of the data being transferred, low costs are few of the multifarious advantages that e-mail enjoys over other physical methods. However there are few issues that spoil the efficient usage of emails. Spam email is one among them [1]. In recent years, spam email or more properly, Unsolicited Bulk Email (UBE) is a widespread problem on the Internet. Spam email is so cheap to send, that unsolicited messages are sent to a large number of users indiscriminately. When a large number of spam messages are received, it is necessary to take a long time to identify spam or non-spam email and their email messages may cause the mail server to crush.

To solve the spam problem, there have been several attempts to detect and filter the spam email on the client-side. In previous research, many Machine Learning (ML) approaches are applied to the problem, including Bayesian classifiers as Naive Bayes, C4.5 and Support Vector Machine (SVM) etc. In these approaches, Bayesian classifiers obtained good results by many researchers so that it widely applied to several filtering software's. However, almost approaches learn and find the distribution of the feature set in only the spam and the non-spam messages.

Smarter AI-Powered Spam Classifier is an advanced email filtering system designed to enhance your email experience by effectively identifying and managing spam messages. This innovative solution leverages cutting-edge artificial intelligence and machine learning technologies to accurately detect and categorize spam, minimizing the clutter in your inbox while ensuring that legitimate emails reach your primary folder. With its adaptive learning capabilities, the Smarter AI-Powered Spam Classifier continually refines its algorithms to stay ahead of evolving spam techniques, providing you with a more efficient and secure email communication environment. Say goodbye to the frustration of sifting through unwanted emails, thanks to this intelligent spam classification system.

What is spaCy

spaCy is an open-source natural language processing (NLP) library designed for processing and analyzing text data. It is written in Python and provides tools and resources for various NLP tasks, including tokenization, part-of-speech tagging, named

entity recognition, dependency parsing, and more. spaCy is known for its speed and efficiency, making it a popular choice for NLP tasks in research and industry applications. It also supports multiple languages and has pre-trained models for various languages, making it a versatile tool for working with text data.

Naive Bayes

Naive Bayes is a classification algorithm that's particularly well-suited for text classification tasks, like spam detection or sentiment analysis. It's based on Bayes' theorem, which is a fundamental concept in probability theory. Naive Bayes can be quite effective in text classification and spam detection. It calculates the probability of a data point belonging to a particular class based on the probabilities of its individual features. There are different variants of Naive Bayes, including Gaussian, Multinomial, and Bernoulli, which are suitable for different types of data.

Training: You start by training the model with a labeled dataset. For example, if you're building a spam filter, you'd provide it with a dataset of emails labeled as spam or not spam. The algorithm learns from this data.

Feature Independence: Naive Bayes makes the "naive" assumption that all features (words in the case of text) are independent of each other. This means that the presence or absence of one word doesn't affect the presence or absence of another word.

Calculating Probabilities: Given a new, unlabeled data point (e.g., a new email), Naive Bayes calculates the probability that it belongs to each class (e.g., spam or not spam) based on the occurrence of its features (words). It uses Bayes' theorem to do this.

Classification: The algorithm assigns the data point to the class with the highest probability.

Decision Tree C4.5

C4.5 (Classifier for 4.5) is a decision tree algorithm used in machine learning and data mining. It is designed for classification tasks and is named after its creator, Ross Quinlan. C4.5 constructs a decision tree from a dataset, where each internal node represents a test on an attribute, each branch represents an outcome of that test, and each leaf node represents a class label.

Attribute Selection: C4.5 selects the best attribute to split the dataset based on a metric called information gain. It measures how much information an attribute provides about the classification.

Recursive Partitioning: The dataset is divided into subsets based on the selected attribute. This process continues recursively until a stopping criterion is met, such as a maximum tree depth or a threshold for data purity.

Pruning: After the initial tree is constructed, C4.5 employs pruning to reduce the tree's complexity and improve its generalization by removing branches that don't contribute significantly to accuracy.

Handling Missing Values: C4.5 can handle missing attribute values during the tree construction process.

Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected neurons, including an input layer, one or more hidden layers, and an output layer. Each neuron in one layer is connected to every neuron in the subsequent layer.
MLPs are used for a variety of machine learning tasks, including classification and regression. They can learn complex relationships within data by adjusting the weights of the connections between neurons during training. The hidden layers in MLPs enable them to model non-linear patterns in data.
MLPs have been a fundamental component of deep learning and are often used in more complex neural network architectures. They are known for their versatility and have been applied to a wide range of applications in fields like image recognition, natural language processing, and more.

RELATED WORK

Email spam is one of the major problems of the today’s Internet, bringing financial damage to companies and annoying individual users. Among the approaches developed to stop spam, filtering is the one of the most important technique. Spam mail, also called unsolicited bulk e-mail or junk mail that is sent to a group of recipients who have not requested it. The task of spam filtering is to rule out unsolicited e-mails automatically from a user's mail stream. These unsolicited mails have already caused

many problems such as filling mailboxes, engulfing important personal mail, wasting network bandwidth, consuming users time and energy to sort through it, not to mention all the other problems associated with spam [11]. Two methods of machine classification were described in paper [4]. The first one is done on some rules defined manually. The typical example is the rule based expert systems. This kind of classification can be used when all classes are static, and their components are easily separated according to some features. The second one is done using machine learning techniques. Paper [15] formalizes a problem of clustering of spam message collection through criterion function. The criterion function is a maximization of similarity between messages in clusters, which is defined by k-nearest neighbor algorithm. Genetic algorithm including penalty function for solving clustering problem is offered. Classification of new spam messages coming to the bases of antispam system. A novel distributed data mining approach, called Symbiotic Data Mining (SDM) [7] that unifies ContentBased Filtering (CBF) with Collaborative Filtering (CF) is described. The goal is to reuse local filters from distinct entities in order to improve personalized filtering while maintaining privacy. In paper [26] the effectiveness of email classifiers based on the feed forward back propagation neural network and Bayesian classifiers are evaluated. Results are evaluated using accuracy and sensitivity metrics. The results show that the feed forward back propagation network algorithm classifier provides relatively high
accuracy and sensitivity that makes it competitive to the best known classifiers. A fully Bayesian approach to soft clustering and classification using mixed membership models based on the assumptions on four levels: population, subject, latent variable, and sampling scheme was implemented in [8]. In paper [1]-[3], automatic anti-spam filtering becomes an important member of an emerging family of junk-filtering tools for the Internet, which will include tools to remove advertisements. The author separate distance measures for numeric and nominal variables, and are then combined into an overall distance measure. In another method, nominal variables are converted into numeric variables, and then a distance measure is calculated using all variables. Paper [24] analyzes the computational complexity and scalability of the algorithm, and tests its performance on a number of data sets from various application domains. The social networks of spammers [12] by identifying communities of harvesters with high behavioral similarity using spectral clustering. The data analyzed was collected through Project Honey Pot
[14], a distributed system for monitoring harvesting and spamming. The main findings are (1) that most spammers either send only phishing emails or no phishing emails at all, (2) that most communities of spammers also send only phishing emails or no phishing emails at all, and (3) that several groups of spammers within communities exhibit coherent temporal behavior or have similar IP addresses [5].

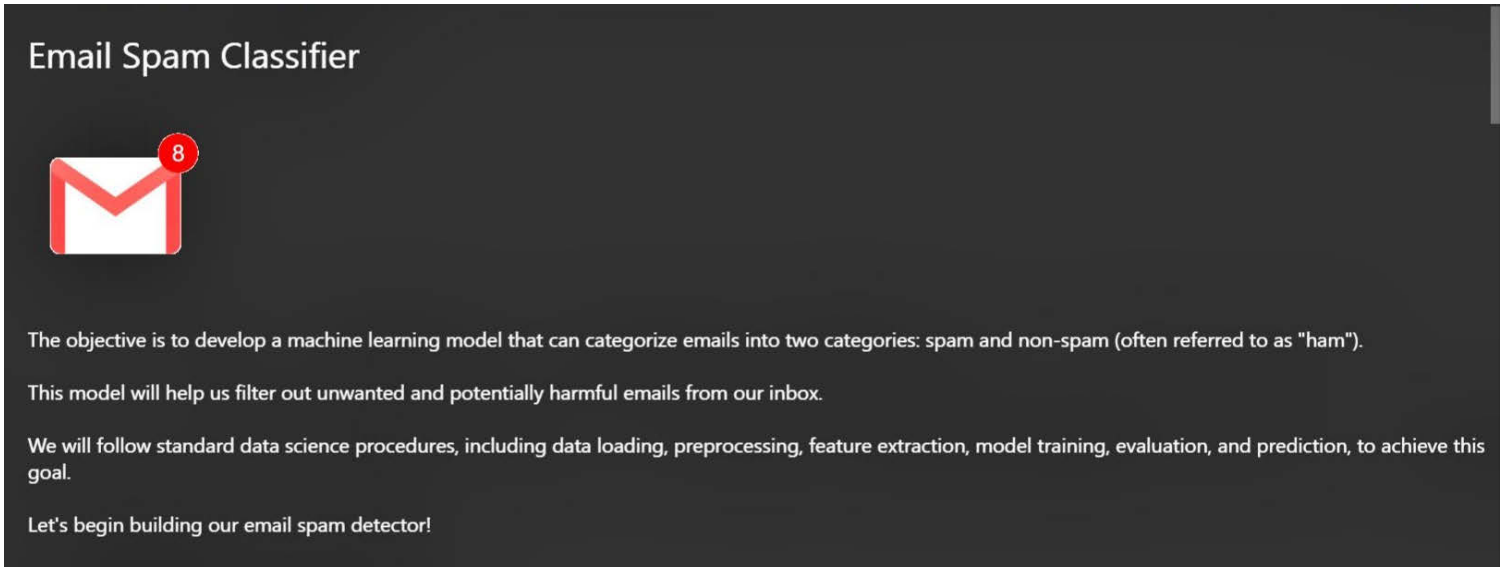
It is demonstrated that both methods obtain significant generalizations from a small number of examples; that both methods are comparable in generalization performance on problems of this type; and that both method asset reasonably efficient, even with fairly large training sets [6]. Spam classification [21] is done through Linear Discriminant Analysis by creating a bag-of words document for every Web site.

The Proposed Approach For Spam Email Classifi-cation

In this proposed model, the input is two publicly available datasets and one proposed email dataset is pre-processed using tokenization, stemming and lemmatization techniques. Afterward, the dataset is splited into train, and test by 80%, 20% respectively. Subsequently,
The spam detection is a text classification problem, That'swhy only standard ML models are used for feature engineering rather than Deep Learning (DL) techniques, dueto the viable execution proficiency interest. Furthermore, the features are extracted using two sate of the art modules namely Count-Vectorizer and TFIDF-Vectorizer and use different Hyper parameter optimization techniques for selecting the best set of parameters for a classification model.

Datasets

In this study, we use two publicly available text-based datasets. And also prepared the proposed dataset with the help of state-of-the-art, web scrapping, and personal email data. The proposed dataset is labeled with ham or spam emails. The datasets were in CSV format. The following are the two publicly available datasets, Ling Spam[14]. And UCI SMS SPAM dataset [15]. Both of these were not pre-processed and included links, numbers, and other noise in them.



Importing Necessary Libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Load and Explore the Dataset

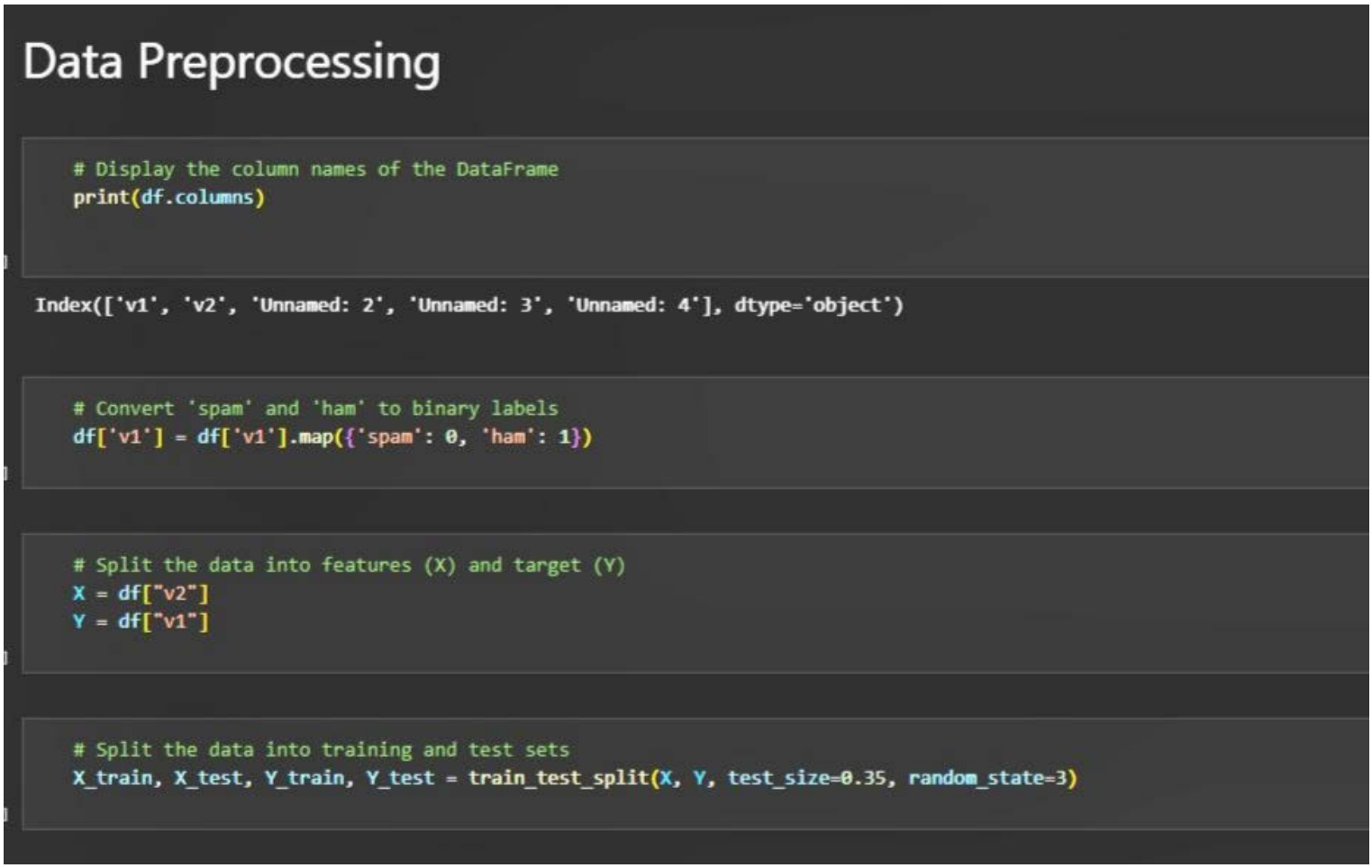
```
# Load the dataset
df = pd.read_csv("/kaggle/input/sms-spam-collection-dataset/spam.csv", encoding='ISO-8859-1')
```

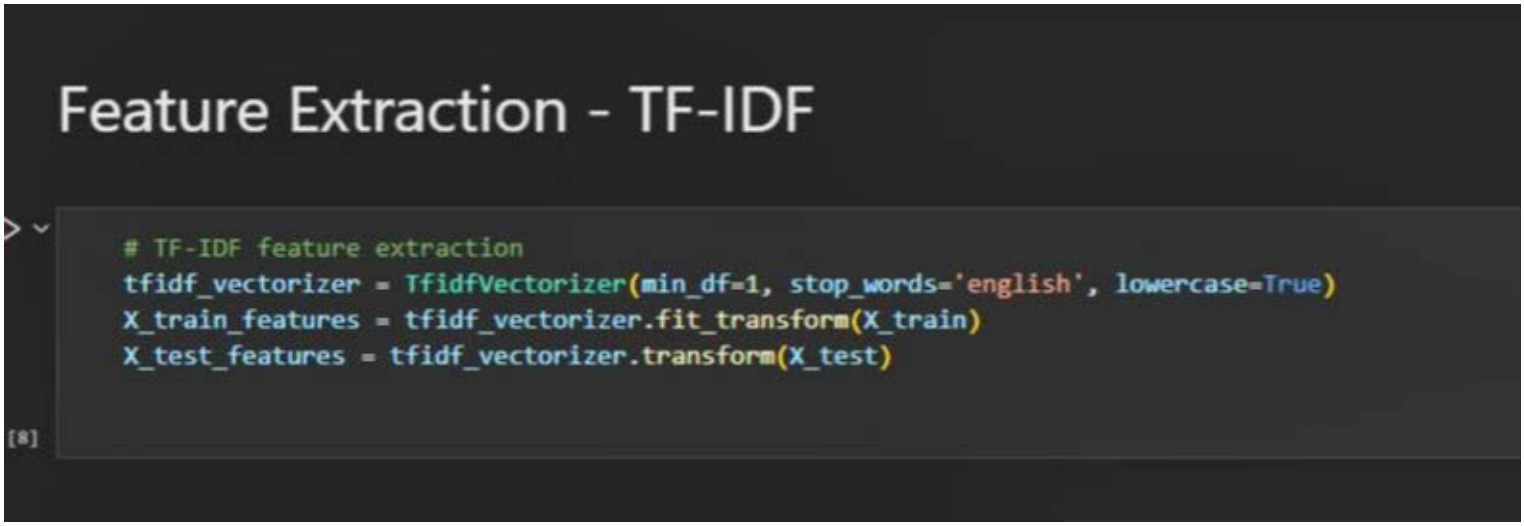
```
# Display the first few rows of the dataset
df.head()
```

	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

Data Pre-processing

In filtering of spam, the pre-processing of the textual information is very critical and important. Main objective of text data pre processing is to remove data which do not give useful information regarding the class of the document. Furthermore we also want to remove data that is redundant. Most widely used data cleaning steps in the textual retrieval tasks are removing of stop words and performing stemming to reduce the vocabulary [2]. In addition to these two steps we also removed the words that have length lesser than or equal to two.





Model Training (Random Forest)

```
# Model training
model = RandomForestClassifier(n_estimators=100, random_state=3)
model.fit(X_train_features, Y_train)
```

[9]

```
RandomForestClassifier
RandomForestClassifier(random_state=3)
```

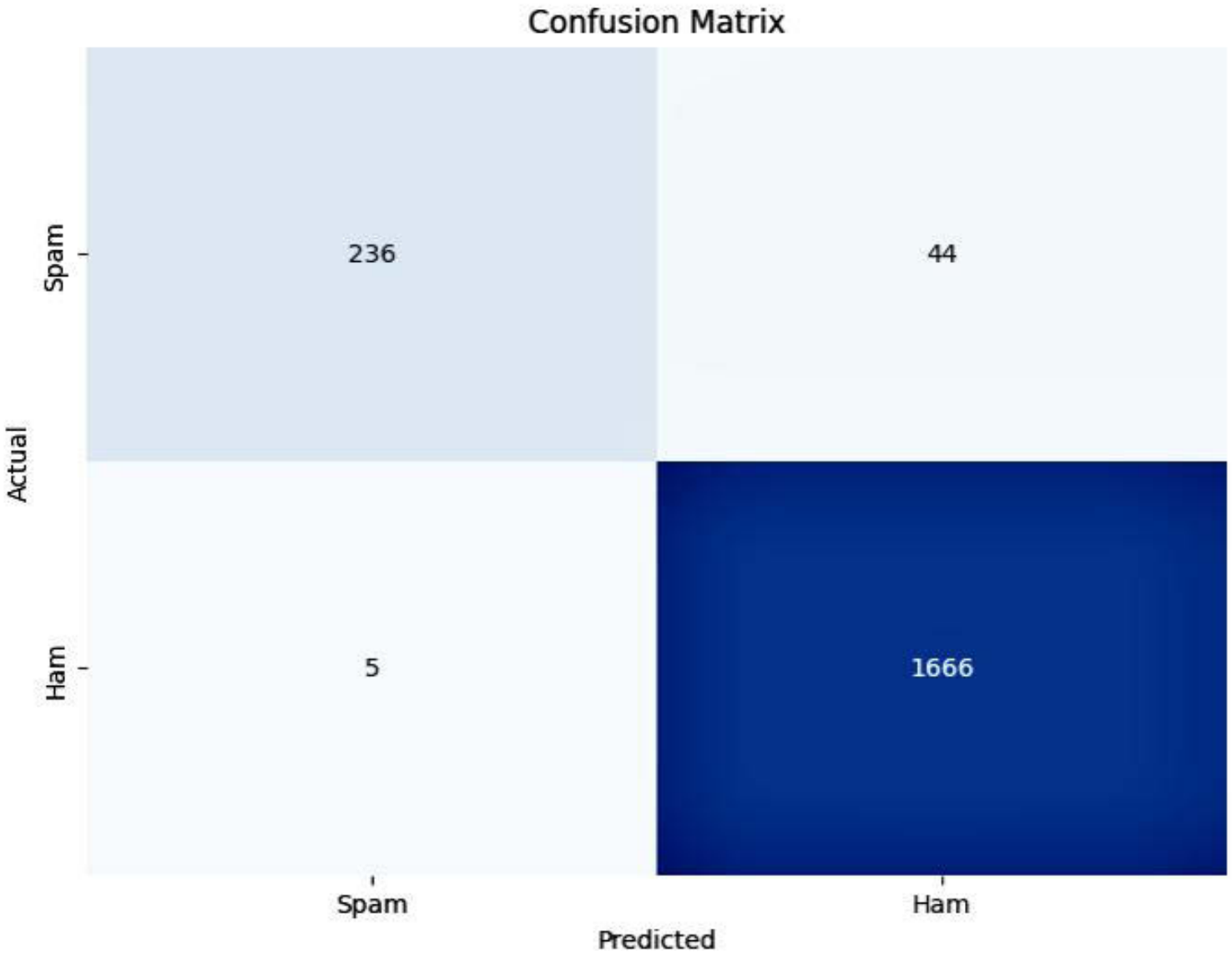
Model Evaluation (Random Forest)

```
prediction_on_training_data = model.predict(X_train_features)
accuracy_on_training_data = accuracy_score(Y_train, prediction_on_training_data)

prediction_on_test_data = model.predict(X_test_features)
accuracy_on_test_data = accuracy_score(Y_test, prediction_on_test_data)
```

[10]

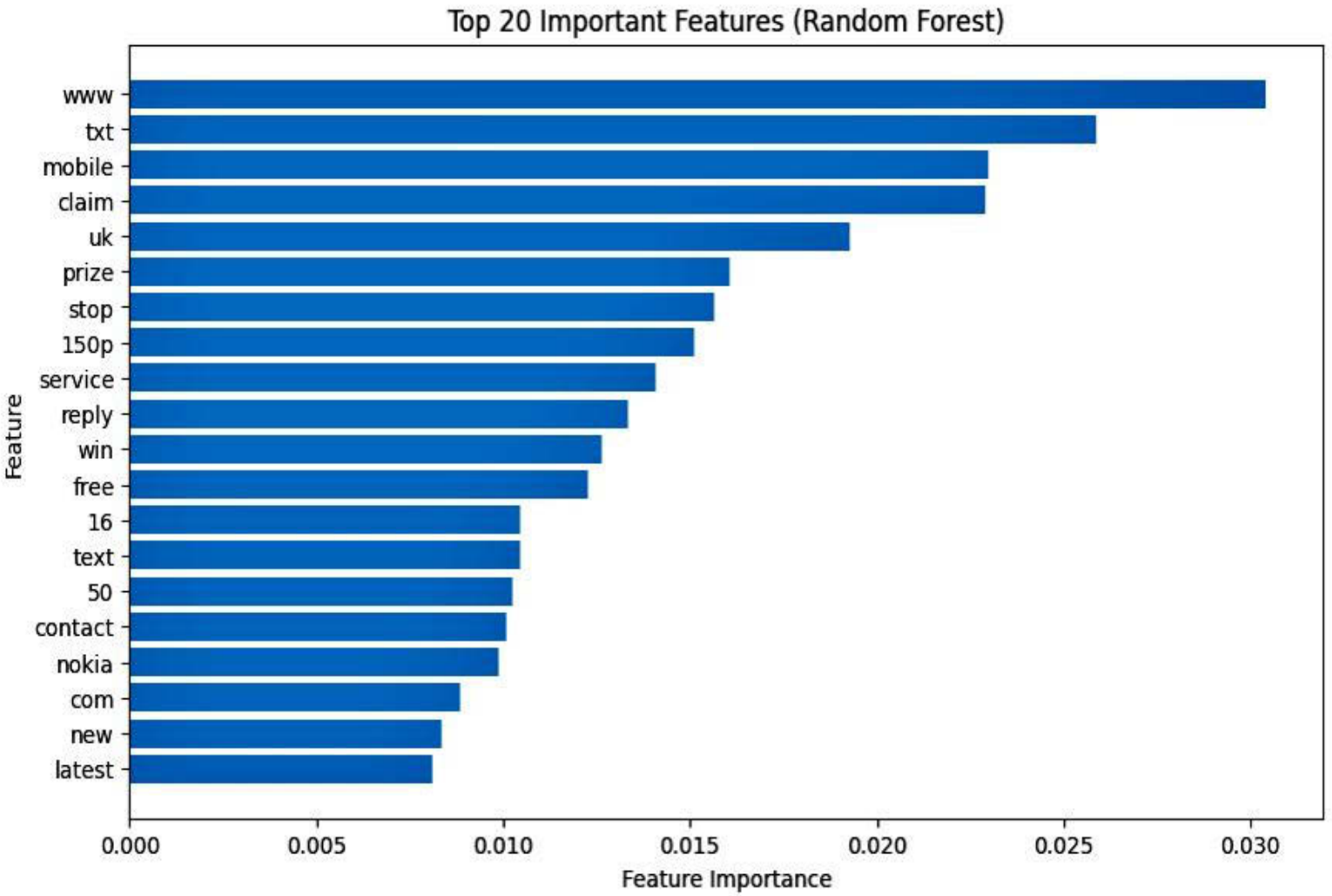
Confusion Matrix Visualization(Random Forest Classifier)

```
# Confusion Matrix Visualization
conf_matrix = confusion_matrix(Y_test, prediction_on_test_data)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Spam', 'Ham'], yticklabels=['Spam', 'Ham'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Feature Importance Visualization (Random Forest)

```
feature_importance = model.feature_importances_
feature_names = tfidf_vectorizer.get_feature_names_out()
sorted_idx = np.argsort(feature_importance)[-20:]  # Top 20 important features

plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align="center")
plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx])
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("Top 20 Important Features (Random Forest)")
plt.show()
```



Make Predictions on New Input (Random Forest Classifier)

```
input_your_mail = "Keep yourself safe for me because I need you and I miss you already and I envy everyone that see's you in real life"
input_data_features = tfidf_vectorizer.transform([input_your_mail])
prediction = model.predict(input_data_features)
if prediction[0] == 1:
    print("Ham Mail")
else:
    print("Spam Mail")
```

```
Ham Mail
```

Removal of Stopwords in Text Pre-processing

Stopwords are a set of words that do not value a text example ‘a’,’an’,’the’ these are the words that occur very frequently in our text data, but they are of no use. Many libraries have compiled stop words for various languages and we can use them directly and for any specific use case if we feel we can also add a more specific set of stop words to the list.

```
from nltk.corpus import
stopwords
"
".join(stopwords.words('english'
```

Code for removal of stop words Before stop words removal we need tokenized text

```
#defining function for
tokenization
import re
#whitespace tokenizer
from nltk.tokenize import
WhitespaceTokenizer
def tokenization(text):
    tk = WhitespaceTokenizer()
    return tk.tokenize(text)

#applying function to the
column for making tokens in
both Training and Testing data
df['tokenised_clean_msg']=
df['clean_msg'].apply(lambda
x: tokenization(x))
```

Now stop words removal

```
#importing nlp library
import nltk
nltk.download('stopwords')
#Stop words present in the
library
stopwords =
nltk.corpus.stopwords.words('eng

#defining the function to
remove stopwords from
tokenized text
def remove_stopwords(text):
    output= [i for i in text
if i not in stopwords]
    return output

#applying the function for
removal of stopwords
df['cleaned_tokens']=
df['tokenised_clean_msg'].apply(
x:remove_stopwords(x))
```

Stemming in Text Pre-processing

It is a text standardization technique where a word is reduced to its stem/base word. Example: “jabbing” → “jab” and “kicking” → “kick”. The main aim for stemming is that we can reduce the vocab size before inputting it into any machine learning model.

```
#importing the Stemming
function from nltk library
from nltk.stem.porter import
PorterStemmer
#defining the object for
stemming
porter_stemmer =
PorterStemmer()

#defining a function for
stemming
def stemming(text):
  stem_text =
[porter_stemmer.stem(word) for
word in text]
  return stem_text

# applying function for
stemming
df['cleaned_tokens']=df['cleaned
x: stemming(x))
```

Spam detection

Spam detection is the process of identifying and filtering out unsolicited, irrelevant, or potentially harmful messages or content, often found in various communication channels such as email, messaging apps, or online forums. It involves the use of algorithms, techniques, and heuristics to automatically recognize and prevent spam, ensuring that users are protected from unwanted or potentially malicious content.Once the various pre-processing operations (second step) have been performed on the input data and the quality datahas been obtained, different machine learning algorithms can be used to train and evaluate the results.

		Predicted classes				
		TRUE	FALSE		TP	135
					TN	177
Actual classes	TRUE	135	53	188	FP	10
	FALSE	10	177	187	FN	53
		145	230			

Fig. 4. Confusion matrix extraction from the final Naive Bayes (NB) model

		Predicted classes				
		TRUE	FALSE		TP	148
					TN	185
Actual classes	TRUE	148	32	180	FP	10
	FALSE	10	185	195	FN	32
		158	217			

Fig. 5. Confusion matrix extraction from the final Decision Tree C45 model

CONCLUSION

Email spam classification has received a tremendous attention by majority of the people as it helps to identify the unwanted information and threats. Therefore, most of the researchers pay attention in finding the best classifier for detecting spam emails. From the obtained results, fisher filtering and runs filtering feature selection algorithms performs better classification for many classifiers. The Rnd tree classification algorithm applied on relevant features after fisher filtering has produced more than 99% accuracy in spam detection. This Rnd tree classifier is also tested with test dataset which gives accurate results than other classifiers for this spam dataset.

Building a Smarter AI-Powered Spam Classifier

Introduction

We are all very well experienced with receiving spam emails in our mail. We can see separate Spam mail box also. Do you ever wonder how those mails are categorized into spam without notifying us. If that feature not available and if we get those mails as normal mail in our inb ox there is a possibility to click and get trapped. So by using our model we will classify whether a mail received is Spam or not using Machine Learning, from this historical data of mails.

Load the Required Modules.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# modelling lib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Algorithms
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import r2_score
```

Load the Dataset

df = pd.read_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding='latin-1')

df.head()

Exploratory Data Analysis

df.info()

From above we can see unnamed 2, unnamed 3, unnamed 4 are NA. so we can drop them.

df.describe()

we can drop unnamed 2, unnamed 3, unnamed 4

df. drop(columns= ["Unnamed: 2","Unnamed: 3","Unnamed: 4"], axis=1, inplace=True)

df.head()

df['v2'].value_counts()

We can see there are repaeted duplicate values. we can remove duplicates.

duplicates

df[df.duplicated]

#Removing duplicates

df = df.drop_duplicates()

df.head()

Analyze after basic clean of data

df.info()

description

df.describe()

unique values

df.nunique()

balanced or imbalanced dataset

df['v1'].value_counts()

null values

df.isnull().sum()

Data Visualization

Univariate analysis

we have only v1 as categorical so we can use countplot for frequency

sns.countplot(data= df, x='v1')

As we can see from above data set we can see spam has very less spam emails.

Label Encoding

converting v1 into numerical from categorical

le = LabelEncoder()

df['v1']= le.fit_transform(df['v1'])

df.head()

ML Model Training

X and y features

X= df['v2']

y=df['v1']

X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=20)

X_train.shape,X_test.shape,y_train.shape,y_test.shape

Converting Text into Numerical using TF-IDF vectorizer

from sklearn.feature_extraction.text import TfidfVectorizer

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

train and test into numerical

X_train_feat = extract.fit_transform(X_train)

X_test_feat = extract.transform(X_test)

Convert the target values into 0 and 1

y_train = y_train.astype(int)

y_test = y_test.astype(int)

print(X_train_feat)

Training with Logistic Regression

lr = LogisticRegression()

lr.fit(X_train_feat,y_train)

y_pred= lr.predict(X_test_feat)

print("Logistic Regression Accuracy score", accuracy_score(y_test,y_pred)*100)

print(confusion_matrix(y_pred,y_test))

print("R-Squared value : ", r2_score(y_test,y_pred))

Accuracy score of 95%

Training with Decision tree Classification

dc = DecisionTreeClassifier(random_state=0)

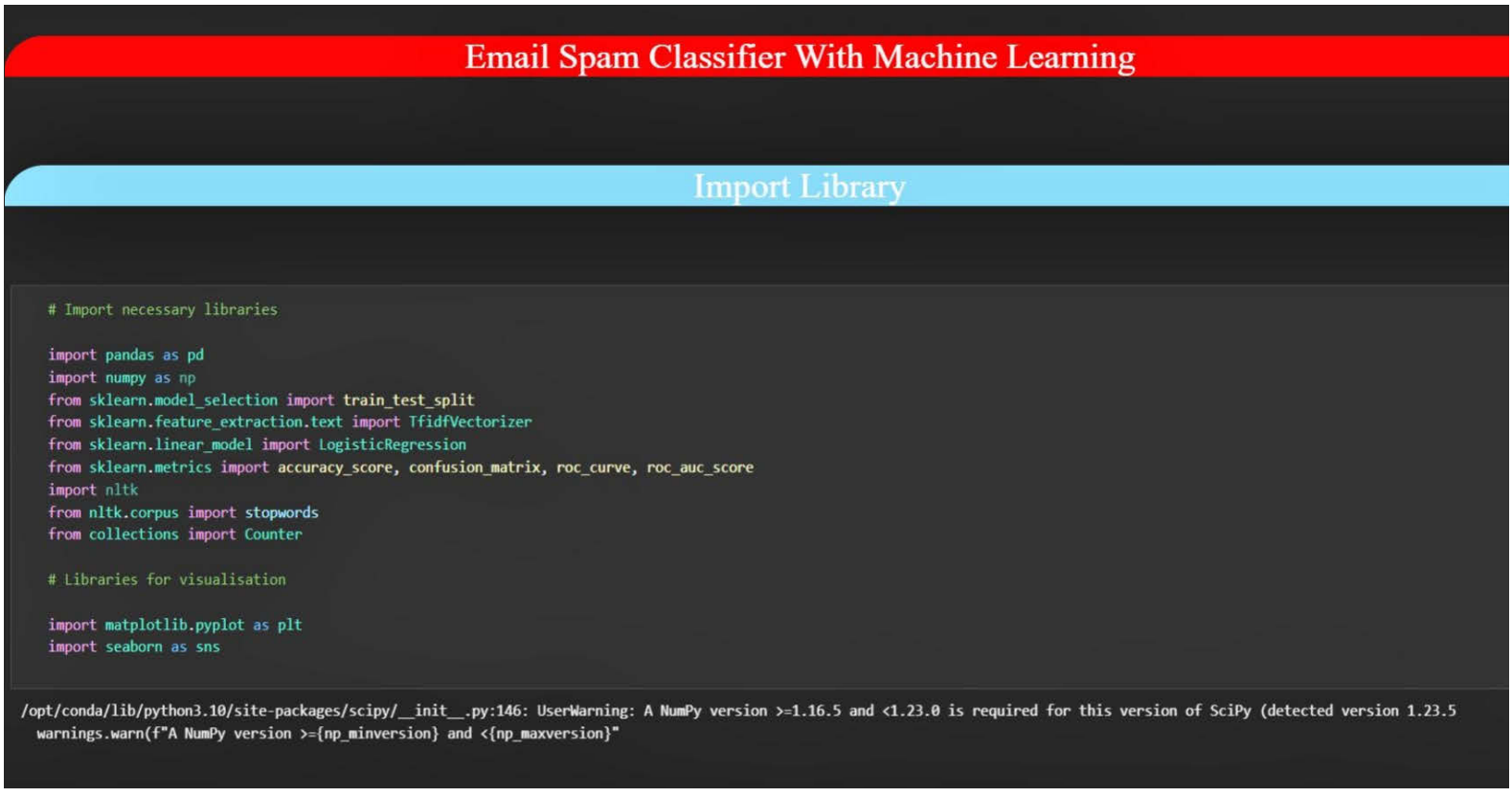
dc.fit(X_train_feat,y_train)

y_pred= lr.predict(X_test_feat)

print("Decision tree Classification Accuracy score", accuracy_score(y_test,y_pred)*100)

print(confusion_matrix(y_pred,y_test))

print("R-Squared value : ", r2_score(y_test,y_pred))



Load The Dataset And Explore

```
# Read the CSV file containing email data into a DataFrame

df = pd.read_csv("/kaggle/input/sms-spam-collection-dataset/spam.csv",encoding='latin-1')
df
```

	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 ì_ 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 × 5 columns

```
df.info()
```

```
<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
```

```
# Drop unnecessary columns from the DataFrame

columns_to_drop = ["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"]
df.drop(columns=columns_to_drop, inplace=True)
```

```
df
```

	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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
# Rename the columns "v1 and "v2" to new names

new_column_names = {"v1":"Category","v2":"Message"}
df.rename(columns = new_column_names,inplace = True)
```

```
df
```

	Category	Message
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
# Replace any NaN values in the DataFrame with a space

data = df.where((pd.notnull(df)), ' ')
```

```
data.head(10)
```

	Category	Message
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...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...

```
data.describe()
```

	Category	Message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

```
data.shape
```

```
(5572, 2)
```

```
# Convert the "Category" column values to numerical representation (0 for "spam" and 1 for "ham")

data.loc[data["Category"] == "spam", "Category"] = 0
data.loc[data["Category"] == "ham", "Category"] = 1
```

```
# Separate the feature (message) and target (category) data

X = data["Message"]
Y = data["Category"]
```

```
print(X)
```

```
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                             ...
5567    This is the 2nd time we have tried 2 contact u...
5568                Will Ì_ b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                          Rofl. Its true to its name
Name: Message, Length: 5572, dtype: object
```

```
print(Y)
```

```
0       1
1       1
2       0
3       1
4       1
       ..
5567    0
5568    1
5569    1
5570    1
5571    1
Name: Category, Length: 5572, dtype: object
```

```
# 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 = 3)
```

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

```
(5572,)
(4457,)
(1115,)
```

```
# Create a TF-IDF vectorizer to convert text messages into numerical features

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

```
# Convert the training and testing text messages into numerical features using TF-IDF

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

```
# Convert the target values to integers (0 and 1)

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

```
print(X_train)
```

```
3075    Mum, hope you are having a great day. Hoping t...
1787                          Yes:)sura in sun tv.:)lol.
1614    Me sef dey laugh you. Meanwhile how's my darli...
4304                Yo come over carlos will be here soon
3266                    Ok then i come n pick u at engin?
                              ...
789                         Gud mrng dear hav a nice day
968            Are you willing to go for aptitude class.
1667    So now my dad is gonna call after he gets out ...
3321    Ok darlin i supose it was ok i just worry too ...
1688                    Nan sonathaya soladha. Why boss?
Name: Message, Length: 4457, dtype: object
```

```
print(X_train_features)
```

```
  (0, 741)      0.3219352588930141
  (0, 3979)     0.2410582143632299
  (0, 4296)     0.3891385935794867
  (0, 6599)     0.20296878731699391
  (0, 3386)     0.3219352588930141
  (0, 2122)     0.38613577623520473
  (0, 3136)     0.440116181574609
  (0, 3262)     0.25877035357606315
  (0, 3380)     0.21807195185332803
  (0, 4513)     0.2909649098524696
  (1, 4061)     0.380431198316959
  (1, 6872)     0.4306015894277422
  (1, 6417)     0.4769136859540388
  (1, 6442)     0.5652509076654626
  (1, 7443)     0.35056971070320353
  (2, 933)      0.4917598465723273
  (2, 2109)     0.42972812260098503
  (2, 3917)     0.40088501350982736
  (2, 2226)     0.413484525934624
  (2, 5825)     0.4917598465723273
  (3, 6140)     0.4903863168693604
  (3, 1599)     0.5927091854194291
  (3, 1842)     0.3708680641487708
  (3, 7453)     0.5202633571003087
  (4, 2531)     0.7419319091456392
...
  (4456, 6117)  0.5304350313291551
  (4456, 6133)  0.5304350313291551
  (4456, 1386)  0.4460036316446079
  (4456, 4557)  0.48821933148688146
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Logistic Regression Model and Train it on the Training Data

```
# Create a logistic regression model and train it on the training data

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

```
LogisticRegression()
```

```
# Make predictions on the training data and calculate the accuracy

prediction_on_training_data = model.predict(X_train_features)
accuracy_on_training_data = accuracy_score(Y_train, prediction_on_training_data)
```

```
print("Accuracy on training data:",accuracy_on_training_data)
```

```
Accuracy on training data: 0.9661207089970832
```

```
# Make predictions on the test data and calculate the accuracy

prediction_on_test_data = model.predict(X_test_features)
accuracy_on_test_data = accuracy_score(Y_test,prediction_on_test_data)
```

```
print("Accuracy on test data:",accuracy_on_test_data)
```

```
Accuracy on test data: 0.9623318385650225
```

```
# Test the model with some custom email messages

input_your_mail = ["Congratulations! You have won a free vacation to an exotic destination. Click the link to claim your prize now!"]
input_data_features = feature_extraction.transform(input_your_mail)
prediction = model.predict(input_data_features)
print(prediction)


# Print the prediction result

if (prediction)[0] == 1:
  print("Ham Mail")
else:
  print("Spam Mail")
```

```
[0]
Spam Mail
```

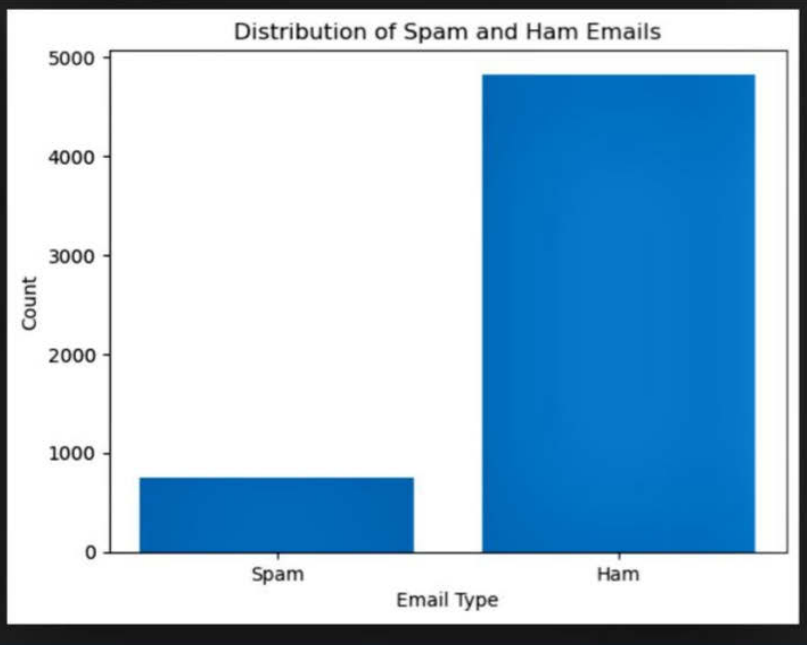
Test the model with some custom email messages

```
input_your_mail = ["Meeting reminder: Tomorrow, 10 AM, conference room. See you there!"]
input_data_features = feature_extraction.transform(input_your_mail)
prediction = model.predict(input_data_features)
print(prediction)


# Print the prediction result

if (prediction)[0] == 1:
  print("Ham Mail")
else:
  print("Spam Mail")
```

```
[1]
Ham Mail
```

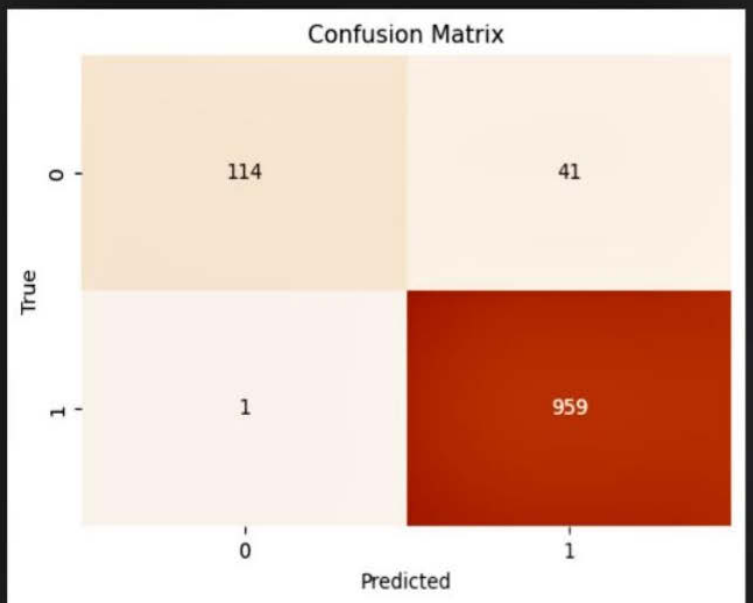
Distribution of Spam and Ham Emails

```
# Data visualization - Distribution of Spam and Ham Emails

spam_count = data[data['Category'] == 0].shape[0]
ham_count = data[data['Category'] == 1].shape[0]

plt.bar(['Spam', 'Ham'], [spam_count, ham_count])
plt.xlabel('Email Type')
plt.ylabel('Count')
plt.title('Distribution of Spam and Ham Emails')
plt.show()
```



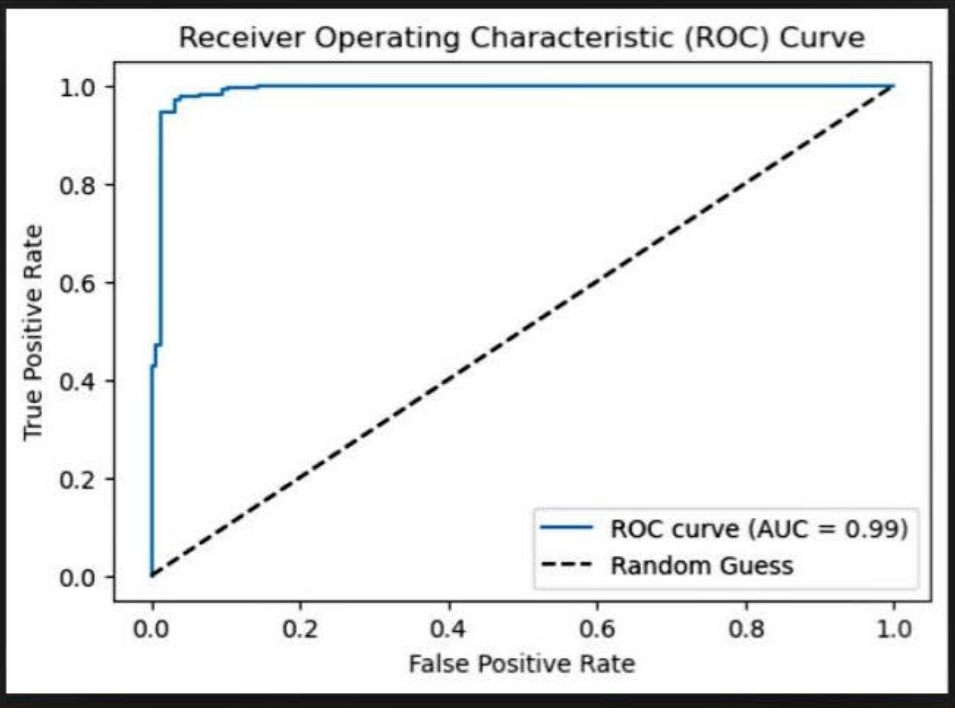
Confusion Matrix

```
cm = confusion_matrix(Y_test, prediction_on_test_data)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap='Oranges', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



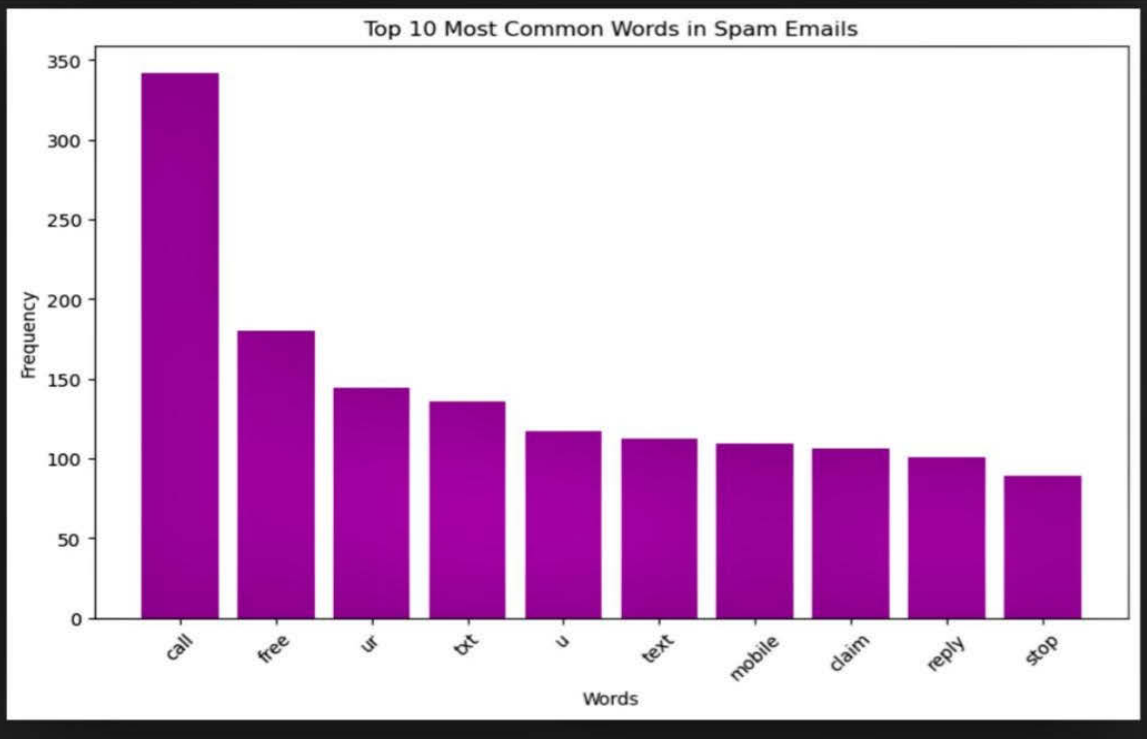
ROC Curve

```
probabilities = model.predict_proba(X_test_features)[:, 1]
fpr, tpr, thresholds = roc_curve(Y_test, probabilities)
roc_auc = roc_auc_score(Y_test, probabilities)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



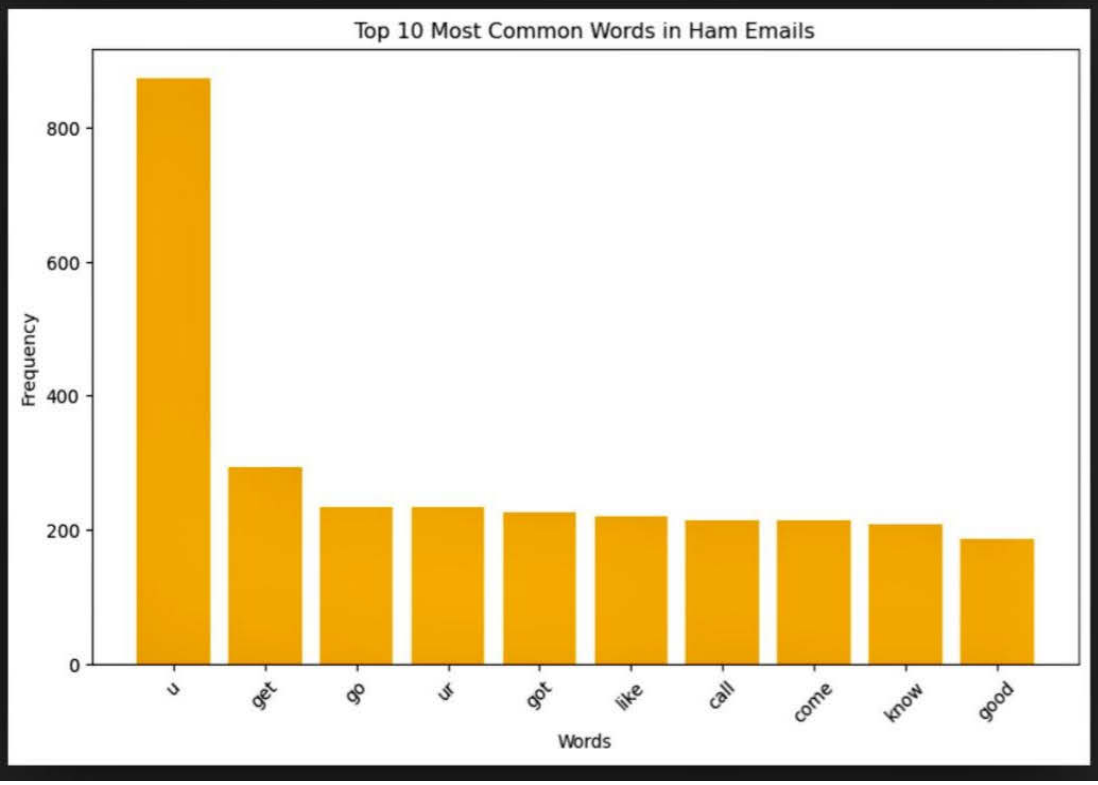
Top 10 Most Common Words in Spam Emails

```
stop_words = set(stopwords.words('english'))
spam_words = " ".join(data[data['Category'] == 0]['Message']).split()
ham_words = " ".join(data[data['Category'] == 1]['Message']).split()

spam_word_freq = Counter([word.lower() for word in spam_words if word.lower() not in stop_words and word.isalpha()])

plt.figure(figsize=(10, 6))
plt.bar(*zip(*spam_word_freq.most_common(10)), color='purple')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 10 Most Common Words in Spam Emails')
plt.xticks(rotation=45)
plt.show()
```



Top 10 Most Common Words in Ham Emails

```
ham_word_freq = Counter([word.lower() for word in ham_words if word.lower() not in stop_words and word.isalpha()])

plt.figure(figsize=(10, 6))
plt.bar(*zip(*ham_word_freq.most_common(10)), color='orange')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 10 Most Common Words in Ham Emails')
plt.xticks(rotation=45)
plt.show()
```



Conclusion

We can observe that both the ML algorithms have given acuracy rate of 95%. That is a good accuracy rate for spam email detection. Email spam classification has received a tremendous attention by majority of the people as it helps to identify the unwanted information and threats. Therefore, most of the researchers pay attention in finding the best classifier for detecting spam emails. From the obtained results, fisher filtering and runs filtering feature selection algorithms performs better classification for many classifiers. The Rnd tree classification algorithm applied on relevant features after fisher filtering has produced more than 99% accuracy in spam detection. This Rnd tree classifier is also tested with test dataset which gives accurate results than other classifiers for this spam dataset.