Assignment No: 3

Title: Implement e-mail spam filtering.

Problem Definition:

Implement e-mail spam filtering using text classification algorithm with appropriate dataset.

Outcome:

Students will be able to,

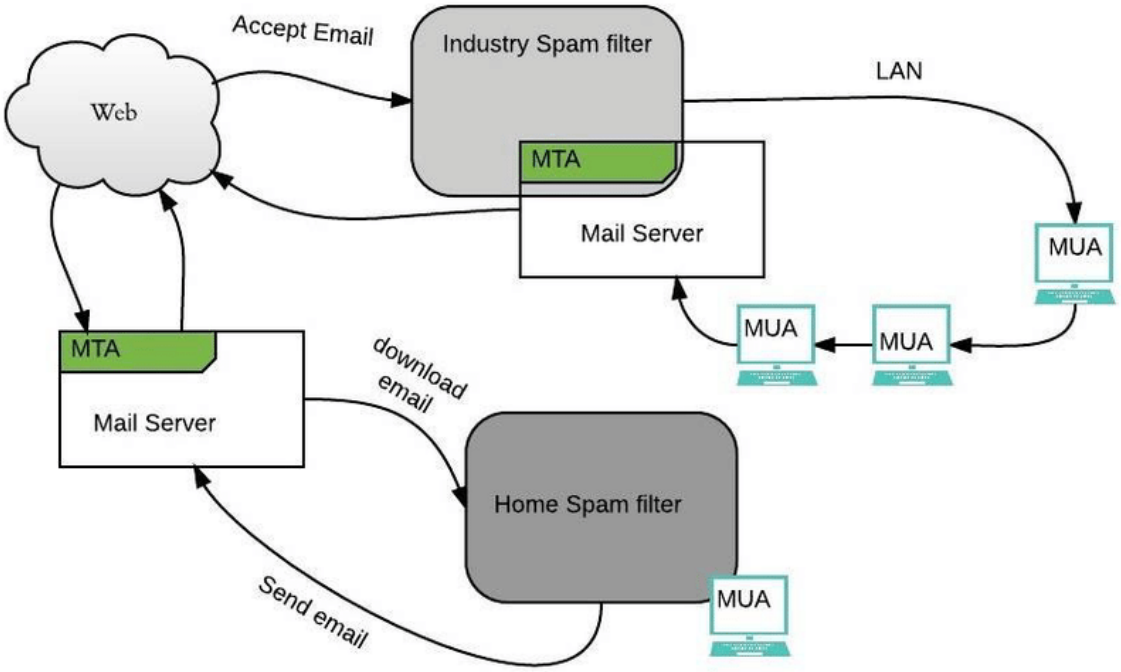
1. Apply various tools and techniques for information retrieval and web mining .
2. Evaluate and analyze retrieved information.

Theory:

Introduction of E-mail spam filtering:

Spam filters are designed to identify emails that attackers or marketers use to send unwanted or dangerous content. They use specific filtering methods to identify the content of emails or their senders and then flag the email as spam. The email can then be automatically deleted instantly or after a period of time.

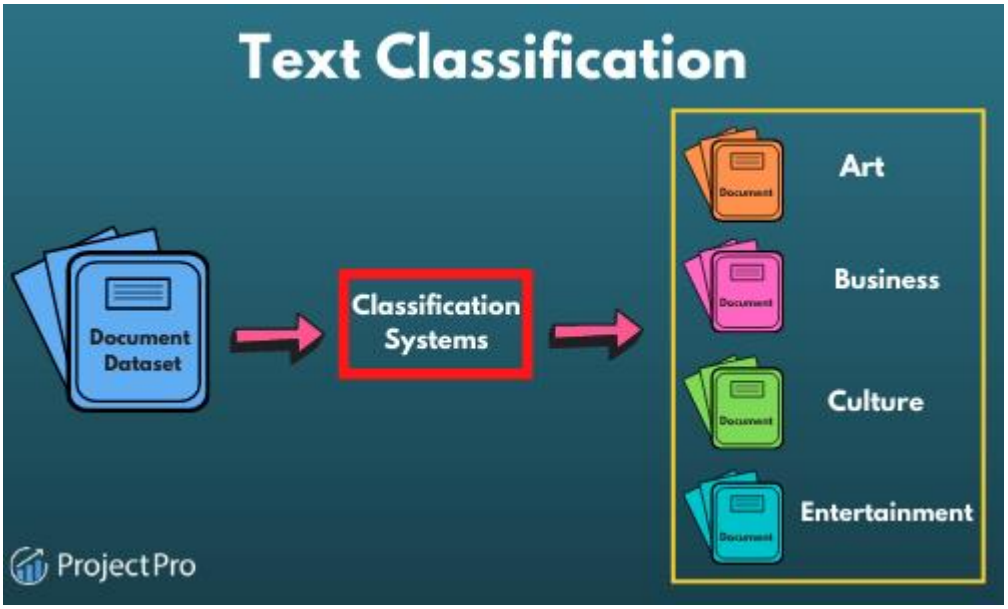
Email filtering is the processing of email to organize it according to specified criteria. The term can apply to the intervention of human intelligence, but most often refers to the automatic processing of messages at an SMTP server, possibly applying anti-spam techniques. Filtering can be applied to incoming emails as well as to outgoing ones.



Introduction of Text classification:

Text classification is a natural language processing (NLP) task in which you categorize or label a piece of text into one or more predefined classes or categories. It has a wide range of applications, such as spam email detection, sentiment analysis, document classification, topic categorization, and more.

An example of text classification is classifying emails as spam or not spam. Another is sentiment analysis, where texts are categorized as positive, negative, or neutral.



Implement e-mail spam filtering:

Implementing an email spam filter involves building a text classification model to classify emails as either spam or not spam (ham). You can use Python and libraries like scikit-learn and NLTK for this task.

Here's a step-by-step guide on how to create an email spam filter:

1. **Data Preparation:**

Choose an appropriate dataset for training your model. One popular dataset for email spam classification is the Enron-Spam dataset.

Pre-process the data, which may include removing HTML tags, special characters, and stop words. You can use libraries like NLTK or spacy for text pre-processing.

2. **Feature Extraction:**

Convert the text data into numerical features. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) and Bag of Words (Bow).

3. **Split the Data:**

Split the dataset into training and testing sets to evaluate the model's performance. A common split is 80% for training and 20% for testing.

4. **Model Selection and Training:**

Choose a machine learning or deep learning algorithm for text classification. Some popular choices are:

1. Naive Bayes
2. Support Vector Machine (SVM)
3. Logistic Regression
4. Random Forest
5. Deep Learning models (e.g., LSTM or CNN)
6. Train the selected model using the training data.

5. **Model Evaluation:**

Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score on the testing dataset.

Adjust hyper parameters and try different algorithms to improve performance if necessary.

6. **Model Deployment:**

Once you are satisfied with the model's performance, you can deploy it to a production environment to classify incoming emails.

- **Here's a Python code example using scikit-learn to build a basic spam filter using a Naive Bayes classifier:**

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
# Load the dataset
data = pd.read_csv("spam_dataset.csv")

# Preprocess the data (e.g., remove HTML tags, special characters, and stopwords)

# Split the data into features (X) and labels (y)
X = data['email_text']
y = data['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

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

# Make predictions
```

```
y_pred = classifier.predict(X_test_tfidf)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)


print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(report)
```

Conclusion:-

we have to implemente an email spam filter using a text classification algorithm with an appropriate dataset is a crucial and effective way to combat unwanted and potentially harmful emails.

Performance & Understanding	Innovation	Timely Completion	Total	Sign & Date
3	1	1	5	