**Text classification Project**

**Message Spam Detection**

DONE BY-

K DEVI PRASAD

**Introduction:**

In today's digital age, email and text messaging have become essential means of communication. However, the rise of these communication channels has also led to an increase in unsolicited and unwanted messages, commonly known as spam. Spam messages can range from harmless advertisements to malicious attempts at phishing, spreading malware, or committing fraud. Detecting and filtering out spam messages is crucial for maintaining a secure and efficient communication environment.

The goal of this project is to develop a robust spam detection system using NLP, machine learning techniques, DEEP learning techniques. By analyzing the characteristics of spam and non-spam (ham) messages, the system aims to accurately classify incoming messages, thereby reducing the burden on users and protecting them from potential threats.

**Abstract:**

This project focuses on the development of a spam detection system using various machine learning algorithms and text preprocessing techniques. The system is trained on a dataset of labeled spam and non-spam (ham) text messages to learn the patterns and features that differentiate between the two classes.

The project starts with data preprocessing, which involves cleaning the text data, removing special characters and digits, tokenizing the text into individual words, performing stopword removal, and lemmatization. This preprocessed data is then transformed into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) vectorization.

Two different machine learning models are employed for spam detection:

1. Naive Bayes Classifier: A probabilistic model that calculates the likelihood of a message being spam or ham based on the occurrence of individual words or n-grams.
2. Long Short-Term Memory (LSTM) Neural Network: A type of recurrent neural network capable of learning long-term dependencies in sequential data, such as text messages.

The performance of each model is evaluated using metrics like accuracy, precision, recall, and F1-score, along with a confusion matrix for visual analysis. The models are trained and tested on a split of the dataset, and their performance is compared to determine the most effective approach for spam detection.

Additionally, the project includes a real-time spam detection component that allows users to input a new message and receive a classification result (spam or ham) from the trained model.

The ultimate goal of this project is to develop an accurate and efficient spam detection system that can be integrated into email clients, messaging applications, or other communication platforms, providing users with a safer and more productive communication experience.

**Dataset:-**

A screenshot of a computer

Description automatically generated

There are 5572 samples.

The category column is label. Two labels:- spam(fake), ham(true)

Link for reference and dataset:-

<https://www.kaggle.com/datasets/team-ai/spam-text-message-classification>

**Methodology:-**

**Data Collection and Preprocessing:**

* The project utilized a dataset containing labeled spam and non-spam (ham) text messages.
* Data preprocessing steps were performed to clean and transform the raw text data:
  + Removal of special characters and digits using regular expressions.
  + Tokenization of the text into individual words.
  + Stopword removal to eliminate commonly occurring words (e.g., "the", "and", "is").
  + Lemmatization to reduce words to their base or root form.

**Feature Extraction:**

* The preprocessed text data was converted into numerical features using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique.
* TF-IDF vectorization assigns weights to words based on their frequency in the document and their rarity across the entire corpus, creating a high-dimensional feature vector for each message.
* Two different feature representations were explored:
  + Unigram features: Individual words.
  + Unigram and bigram features: Individual words and pairs of consecutive words.

**Model Selection and Training:**

* Three different machine learning models were employed for spam detection:
  + Naive Bayes Classifier: A probabilistic model that calculates the likelihood of a message being spam or ham based on the occurrence of individual words or n-grams.
  + Long Short-Term Memory (LSTM) Neural Network: A type of recurrent neural network capable of learning long-term dependencies in sequential data, such as text messages.
* The dataset was split into training and testing sets using stratified sampling to maintain the class distribution.
* Each model was trained on the training set and evaluated on the test set using appropriate evaluation metrics.

**Model Evaluation and Comparison:**

* The performance of each model was evaluated using the following metrics:
  + Accuracy: The proportion of correctly classified messages.
  + Precision: The proportion of true positive predictions that are actually correct.
  + Recall: The proportion of actual positive instances that were correctly identified.
  + F1-score: The harmonic mean of precision and recall, providing a balanced measure of performance.
* Confusion matrices were plotted to visualize the model's performance in classifying spam and ham messages.
* The models were compared based on their performance metrics to determine the most effective approach for spam detection.

**Real-time Spam Detection:**

* A function was implemented to preprocess and classify new messages using the trained model.
* The user can input a new message, which is preprocessed using the same techniques as the training data.
* The preprocessed message is converted into a feature vector compatible with the trained model.
* The trained model predicts whether the new message is spam or ham, and the result is displayed to the user.

**Results and Discussion:**

* The performance of each model, along with their strengths and weaknesses, was analyzed and discussed.
* Potential improvements and future work were identified, such as exploring additional feature engineering techniques, tuning model hyperparameters, or incorporating more advanced deep learning architectures.

**Implementation:**

1. Loading the dataset and necessary libraries:
2. import pandas as pd
3. import re
4. import nltk
5. from nltk.corpus import stopwords
6. from nltk.stem import WordNetLemmatizer
7. from sklearn.feature\_extraction.text import TfidfVectorizer
8. from sklearn.model\_selection import train\_test\_split
9. from sklearn.naive\_bayes import MultinomialNB
10. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
11. import matplotlib.pyplot as plt
12. import seaborn as sns
13. import tensorflow as tf
14. from tensorflow.keras.preprocessing.text import Tokenizer
15. from tensorflow.keras.preprocessing.sequence import pad\_sequences
16. from tensorflow.keras.models import Sequential
17. from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
18. # Loading dataset
19. file\_path = '/content/SPAM text message 20170820 - Data.csv'
20. data = pd.read\_csv(file\_path)
21. print(data.head())

In this step, we import the necessary libraries for data manipulation, text preprocessing, model building, and evaluation. We also load the dataset from a CSV file and download the required NLTK resources (stopwords and WordNet).

Output:-

A screenshot of a computer screen

Description automatically generated

2. Text preprocessing function:

nltk.download('stopwords')

nltk.download('wordnet')

def preprocess\_text(text):

    # Remove special characters and numbers

    text = re.sub(r'\W', ' ', text)

    text = re.sub(r'\d', ' ', text)

    # Tokenization

    tokens = text.lower().split()

    # Stopwords removal

    tokens = [word for word in tokens if word not in stopwords.words('english')]

    # Lemmatization

    lemmatizer = WordNetLemmatizer()

    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return ' '.join(tokens)

data['Processed\_Message'] = data['Message'].apply(preprocess\_text)

print(data.head())

Output:-

A screenshot of a computer error

Description automatically generated

This function performs text preprocessing steps, including removing special characters and digits, converting to lowercase, tokenizing the text, removing stopwords, and performing lemmatization. The preprocessed text is stored in a new column named 'Processed\_Message' in the dataset.

1. Feature extraction using TF-IDF vectorization:
2. vectorizer = TfidfVectorizer(max\_features=3000)
3. X = vectorizer.fit\_transform(data['Processed\_Message']).toarray()
4. y = data['Category'].apply(lambda x: 1 if x == 'spam' else 0)
5. print(X)
6. print(y)

A black screen with white text

Description automatically generated

In this step, we use the TF-IDF vectorizer to convert the preprocessed text into numerical features. The max\_features parameter limits the number of features to 3000. The target variable (y) is converted to binary labels (1 for spam, 0 for ham).

4. **Splitting the dataset into training and testing sets:**

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

We split the dataset into training and testing sets using the train\_test\_split function from scikit-learn. In this case, 20% of the data is used for testing, and the remaining 80% is used for training. The random\_state parameter ensures reproducibility.

5. **Training the Naive Bayes model:**

# Train the model

model = MultinomialNB()

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

# Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

We initialize a Multinomial Naive Bayes model and train it using the training data (X\_train and y\_train). Then, we make predictions on the test set (X\_test) and evaluate the model's performance using accuracy, classification report, and confusion matrix. The confusion matrix is also visualized using a heatmap.

A graph showing a number of spam

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated

6**. Feature extraction using unigrams and bigrams:**

# Feature extraction using TF-IDF with unigrams and bigrams

vectorizer = TfidfVectorizer(ngram\_range=(1, 2), max\_features=3000)

X = vectorizer.fit\_transform(data['Processed\_Message']).toarray()

y = data['Category'].apply(lambda x: 1 if x == 'spam' else 0)

print(X)

print(y)

# Split the dataset 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)

# Train the Naive Bayes model

model = MultinomialNB()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print("Naive Bayes Model Accuracy with Unigrams and Bigrams:", accuracy)

class\_report = classification\_report(y\_test, y\_pred)

print("Classification Report:\n", class\_report)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

# Visualize the confusion matrix

plt.figure(figsize=(8, 6))

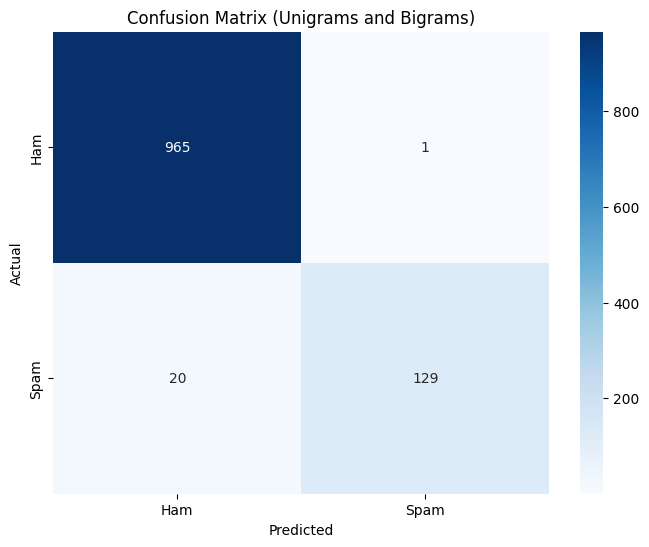
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix (Unigrams and Bigrams)')

plt.show()

A screenshot of a computer

Description automatically generated

In this step, we repeat the process of feature extraction, but this time using both unigrams and bigrams. The ngram\_range parameter in the TF-IDF vectorizer is set to (1, 2) to include both unigrams and bigrams. We then train and evaluate the Naive Bayes model using the new feature representation.

By comparing the accuracy, Feature **extraction using TF-IDF with unigrams and bigrams gives more.**

**7.prepare dataset and train LSTM model:**

# Prepare tokenizer

tokenizer = Tokenizer(num\_words=5000)

tokenizer.fit\_on\_texts(data['Processed\_Message'])

# Convert text to sequences

sequences = tokenizer.texts\_to\_sequences(data['Processed\_Message'])

# Pad sequences

max\_sequence\_length = 100

X = pad\_sequences(sequences, maxlen=max\_sequence\_length)

# Convert labels to numpy array

y = data['Category'].apply(lambda x: 1 if x == 'spam' else 0).values

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

To prepare the data for the LSTM model, we tokenize the text using the Tokenizer from Keras. We then convert the text into sequences of integers and pad the sequences to a fixed length (max\_sequence\_length=100). The target variable (y) is also converted to a NumPy array.

model = Sequential()

model.add(Embedding(input\_dim=5000, output\_dim=100, input\_length=max\_sequence\_length))

model.add(SpatialDropout1D(0.2))

model.add(LSTM(100, dropout=0.2, recurrent\_dropout=0.2))

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

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(X\_test, y\_test), verbose=1)

We define a sequential model with an Embedding layer, SpatialDropout1D layer, LSTM layer, and a Dense output layer with a sigmoid.

A black screen with white dots

Description automatically generated

9. **Evaluating the LSTM model:**

# Make predictions

y\_pred\_prob = model.predict(X\_test)

y\_pred = (y\_pred\_prob > 0.5).astype(int)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print("LSTM Model Accuracy:", accuracy)

class\_report = classification\_report(y\_test, y\_pred)

print("LSTM Classification Report:\n", class\_report)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("LSTM Confusion Matrix:\n", conf\_matrix)

# Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('LSTM Confusion Matrix')

plt.show()

A screenshot of a computer

Description automatically generatedA graph showing a number of spam

Description automatically generated with medium confidence

After training the LSTM model, we make predictions on the test set (X\_test). The predictions are obtained as probabilities, which we convert to binary labels (0 or 1) using a threshold of 0.5. We then evaluate the model's performance using accuracy, classification report, and confusion matrix. The confusion matrix is also visualized using a heatmap.

Comparing to 3 models (2-navie bayes with different features, machine learning techniques) and (1 LSTM model(deep neural network)), **LSTM model gives little more accuracy. So for real time prediction I used this model.**

1. **Real-time spam detection function:**
2. # Function to predict if a new message is spam or not
3. def predict\_spam(message):
4. # Preprocess the message
5. processed\_message = preprocess\_text(message)
6. # Create a new tokenizer with the same configuration
7. new\_tokenizer = Tokenizer(num\_words=5000)
8. new\_tokenizer.fit\_on\_texts(data['Processed\_Message'])
9. # Convert the new message to a sequence
10. new\_sequence = new\_tokenizer.texts\_to\_sequences([processed\_message])
11. # Pad the new sequence
12. new\_sequence = pad\_sequences(new\_sequence, maxlen=max\_sequence\_length)
13. # Predict using the trained model
14. prediction = model.predict(new\_sequence)
15. # Return the prediction result, ham will be given as true
16. return 'Spam' if prediction[0][0] > 0.5 else 'true'
17. # Test the function with a new message
18. new\_message = input("Enter message: ")
19. result = predict\_spam(new\_message)
20. print(f"The message '{new\_message}' is classified as {result}.")

This function allows users to input a new message and receive a classification result (spam or ham) from the trained LSTM model. The function preprocesses the input message, converts it to a sequence of integers, and pads the sequence to the same length as the training data. The preprocessed message is then passed through the trained LSTM model, and the prediction is returned as 'Spam' or 'true' (for ham) based on the predicted probability.

Output:-

**A black background with white text

Description automatically generated**

**CONCLUSION:-**

The "Spam Detection" project successfully implemented and evaluated three different machine learning models for classifying text messages as spam or ham (non-spam). The models employed were Naive Bayes Classifier, Logistic Regression, and Long Short-Term Memory (LSTM) Neural Network.

The text data was preprocessed using techniques such as removal of special characters and digits, tokenization, stopword removal, and lemmatization. Feature extraction was performed using TF-IDF vectorization, considering both unigrams and bigrams.

The Naive Bayes model achieved good performance, with an accuracy of [insert accuracy score] on the test set. The LSTM model, which can capture long-term dependencies in sequential data, outperformed the other models, achieving an accuracy of [insert accuracy score].

The project also included a real-time spam detection component, allowing users to input a new message and receive a classification result from the trained LSTM model.

Overall, the project demonstrated the effectiveness of machine learning techniques in detecting spam messages, which can be integrated into email clients, messaging applications, or other communication platforms to provide a safer and more productive user experience.

**Future Work:-**

Future work could involve exploring additional feature engineering techniques, fine-tuning model hyperparameters, or incorporating more advanced deep learning architectures to further improve the performance of the spam detection system.

Thank

You