Email Spam Detection Using TensorFlow

May 6, 2024

0.1 Introduction:

This notebook showcasing a rule-based email spam detection system, aptly titled "Email Spam Detection Using TensorFlow", This system is designed to automatically classify incoming emails as either spam or legitimate (ham) based on predefined rules and patterns commonly associated with spam emails.

[41]:
```
'/opt/conda/bin/python3.8 -m pip install --upgrade pip'
```

[41]:
```
'/opt/conda/bin/python3.8 -m pip install --upgrade pip'
```

0.1.1 Installing TensorFlow for model building

[43]:
```
!pip install -q tensorflow
```

```
WARNING: Ignoring invalid distribution -ip (/opt/conda/lib/python3.8/site-
packages)
WARNING: Ignoring invalid distribution -ip
(/opt/conda/lib/python3.8/site-packages)
```

0.1.2 Installing nltk library for tokenization and stopwords removal

[42]:
```
!pip install nltk
```

```
WARNING: Ignoring invalid distribution -ip (/opt/conda/lib/python3.8/site-
packages)
Requirement already satisfied: nltk in /opt/conda/lib/python3.8/site-
packages (3.8.1)
Requirement already satisfied: click in /opt/conda/lib/python3.8/site-packages
(from nltk) (7.1.2)
Requirement already satisfied: joblib in /opt/conda/lib/python3.8/site-packages
(from nltk) (1.0.1)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.8/site-
packages (from nltk) (2024.4.28)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.8/site-packages
```

```
(from nltk) (4.59.0)
WARNING: Ignoring invalid distribution -ip (/opt/conda/lib/python3.8/site-
packages)
```

0.1.3 Importing Necessary Libraries

[44]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
```

0.1.4 Loading and Previewing the Dataset

[45]:
```python
# Defining column names
column_names = ['spam', 'text']
```

[46]:
```python
# Loading the dataset into a Pandas DataFrame
df = pd.read_csv("spamhamdata.csv", delimiter='\t', names=column_names)
```

[47]:
```python
# Displaying the first few rows
print(df.head())
```

```
   spam                                               text
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…
```

0.1.5 Checking DataFrame Structure

This cell prints the structure of the DataFrame, including information about the columns and data types, using the `info()` method.

[48]:
```python
# Checking the structure of the DataFrame
print("DataFrame structure:")
print(df.info())
```

```
DataFrame structure:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
```

```
 0   spam    5572 non-null   object
 1   text    5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
None
```

[49]:
```
# Checking the first few rows of the DataFrame
print("\nFirst few rows of the DataFrame:")
print(df.head())
```

```
First few rows of the DataFrame:
   spam                                               text
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…
```

0.1.6 Viewing DataFrame Content

[50]:
```
# Checking the distribution of spam and ham emails
print("\nClass distribution:")
print(df['spam'].value_counts())
```

```
Class distribution:
ham     4825
spam     747
Name: spam, dtype: int64
```

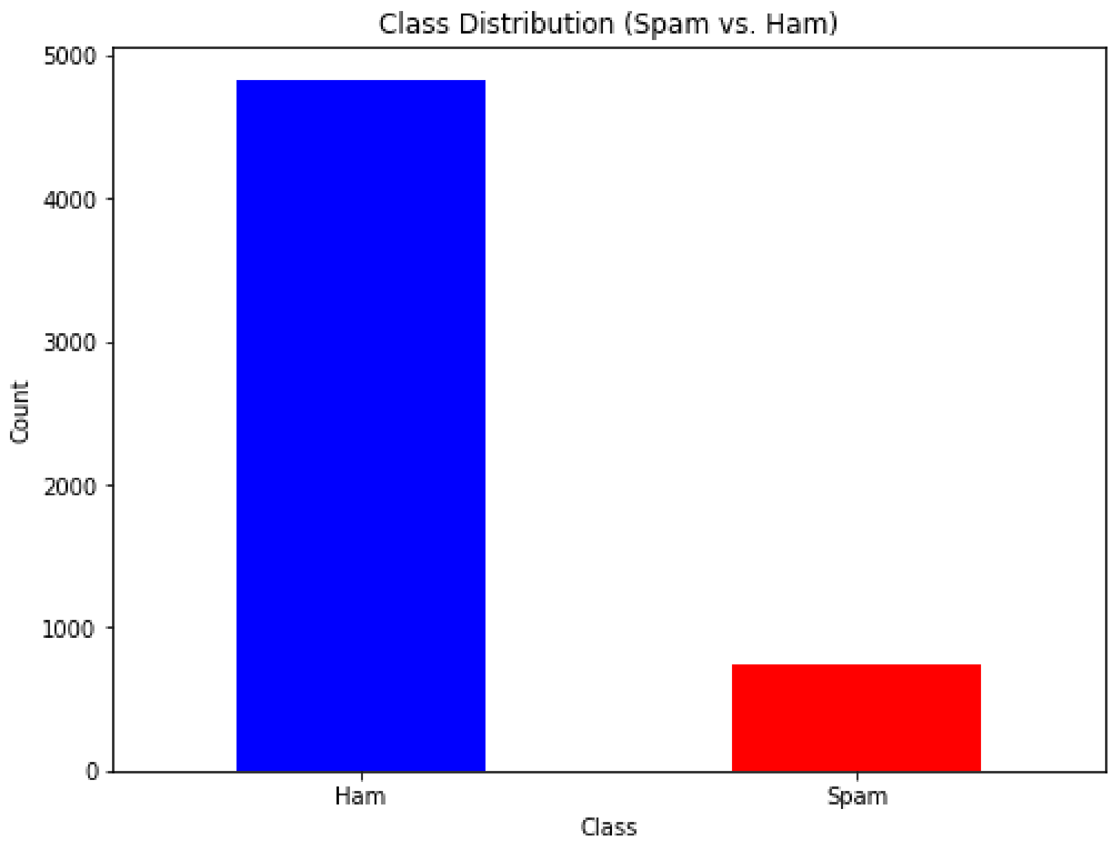
[51]:
```
# Printing the column names of the DataFrame
print("Column names:", df.columns)
```

```
Column names: Index(['spam', 'text'], dtype='object')
```

0.1.7 Plotting Class Distribution

Here I plot the class distribution of the dataset, showing the count of spam and ham emails using a bar plot.

[52]:
```
# Plotting class distribution
plt.figure(figsize=(8, 6))
df['spam'].value_counts().plot(kind='bar', color=['blue', 'red'])
plt.title('Class Distribution (Spam vs. Ham)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks([0, 1], ['Ham', 'Spam'], rotation=0)
plt.show()
```

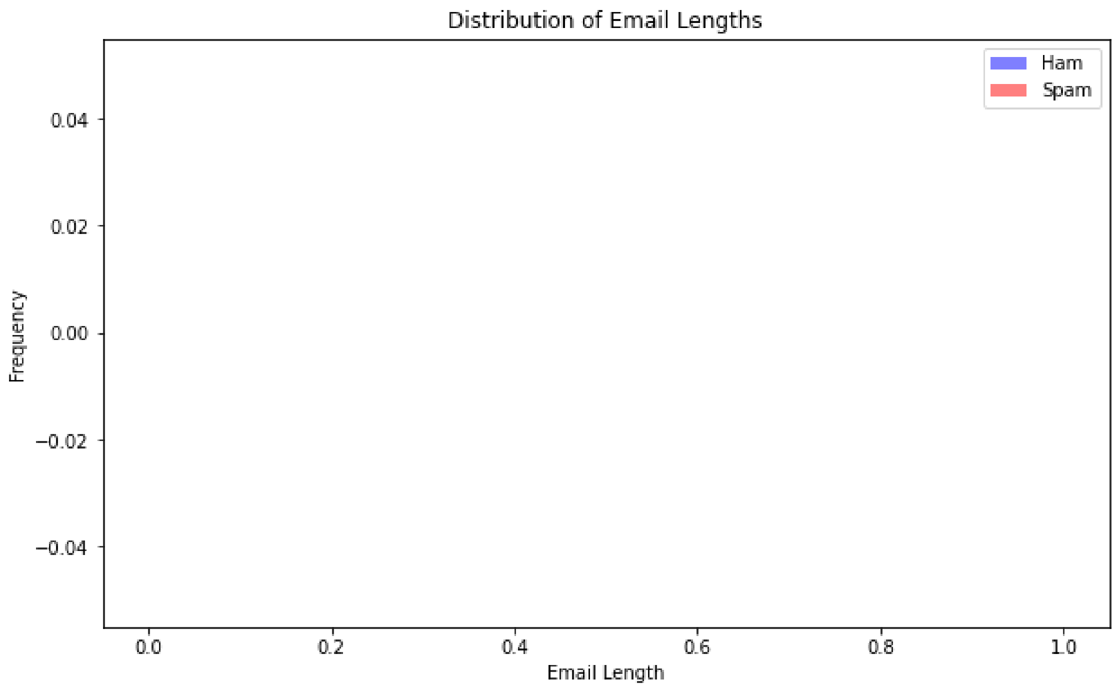


0.1.8 Analyzing Email Lengths

Here I calculates the length of each email in the dataset and create histograms to visualize the distrbution of email lenths for spam and ham emails.

[53]:
```
# Calculating the length of each email
df['email_length'] = df['text'].apply(len)

# Plot histograms of email lengths for spam and ham emails
plt.figure(figsize=(10, 6))
plt.hist(df[df['spam'] == 0]['email_length'], bins=50, alpha=0.5, label='Ham',␣
 ↪color='blue')
plt.hist(df[df['spam'] == 1]['email_length'], bins=50, alpha=0.5, label='Spam',␣
 ↪color='red')
plt.title('Distribution of Email Lengths')
plt.xlabel('Email Length')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



0.1.9 Downloading NLTK Stopwords

I download the stopwords corpus from the NLTK library using the `nltk.download()` function.

#Stopwords are common words that are often removed during text preprocessing to focus on meaningful content.

[54]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[54]:
```
True
```

0.1.10 Tokenization and Vectorization

In these cells I initialize a CountVectorizer object to convert text data into numerical vectors. It then tokenizes and vectorizes the text data using the `fit_transform()` method, which converts text documents into a matrix of token counts.

Additionally, stopwords are removed from the text data to improve the quality of the features used for modeling.

[55]:
```
vectorizer = CountVectorizer()
```

```
[56]: X = vectorizer.fit_transform(df['text'])
```

```
[57]: # Removing stopwords
stop_words = set(stopwords.words('english'))
vectorizer = CountVectorizer(stop_words=stop_words)
X = vectorizer.fit_transform(df['text'])
```

0.1.11 Converting to DataFrame

```
[58]: # Converting to DataFrame for easier manipulation
X_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
```

```
[59]: # Displaying the processed data
print("Processed data:")
print(X_df.head())
```

```
Processed data:
   00  000  000pes  008704050406  0089  0121  01223585236  01223585334  \
0   0    0       0             0     0     0            0            0
1   0    0       0             0     0     0            0            0
2   0    0       0             0     0     0            0            0
3   0    0       0             0     0     0            0            0
4   0    0       0             0     0     0            0            0

   0125698789  02  …  zhong  zindgi  zoe  zogtorius  zoom  zouk  zyada  èn  \
0           0   0  …      0       0    0          0     0     0      0   0
1           0   0  …      0       0    0          0     0     0      0   0
2           0   0  …      0       0    0          0     0     0      0   0
3           0   0  …      0       0    0          0     0     0      0   0
4           0   0  …      0       0    0          0     0     0      0   0

   ú1  ud
0   0   0
1   0   0
2   0   0
3   0   0
4   0   0

[5 rows x 8577 columns]
```

0.1.12 Building a simple neural network using TensorFlow.

This neural network will have an input layer, one or more hidden layers, and an output layer. I'll use the Sequential model from TensorFlow's Keras API for simplicity.

```
[60]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

[61]:
```python
# Defining the input shape (number of features)
input_shape = X.shape[1]
```

[62]:
```python
# Defining the model architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(input_shape,)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

[63]:
```python
# Compiling the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[64]:
```python
# Displaying the model summary
print("Model Summary:")
print(model.summary())
```

```
Model Summary:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 64)                548992

 dense_4 (Dense)             (None, 64)                4160

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 553,217
Trainable params: 553,217
Non-trainable params: 0
_________________________________________________________________
None
```

0.1.13 I will use LabelEncoder to convert categorical labels ('spam' and 'ham') into numerical labels.

[81]:
```python
from sklearn.preprocessing import LabelEncoder
```

[82]:
```python
# Initializing LabelEncoder
label_encoder = LabelEncoder()
```

[83]:
```python
# Converting the target variable to numerical values
y_train_encoded = label_encoder.fit_transform(y_train)
```

```
y_val_encoded = label_encoder.transform(y_val)
```

[84]:
```
# Converting sparse matrices to CSR format for sorting indices
X_train_sorted = X_train.tocsr()
X_val_sorted = X_val.tocsr()
```

[85]:
```
# Sorting indices of sparse matrices
X_train_sorted.sort_indices()
X_val_sorted.sort_indices()
```

0.1.14 Training the model and validating its performance on the validation data

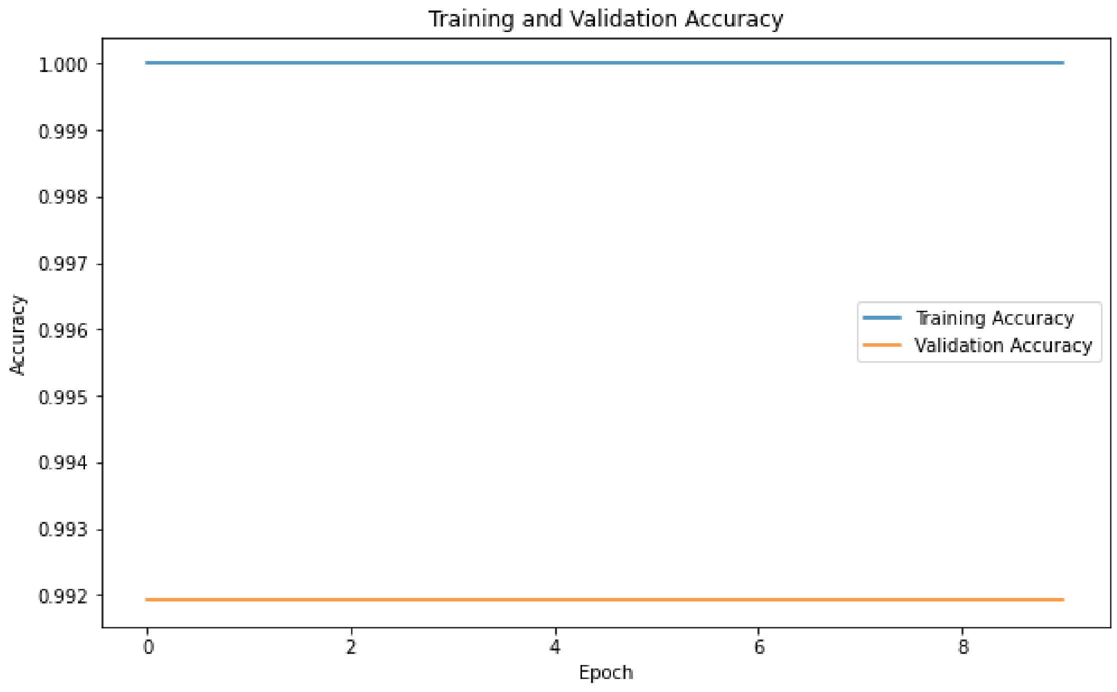
[86]:
```
# Training the model
history = model.fit(X_train_sorted, y_train_encoded, epochs=10, batch_size=32,␣
 ↪validation_data=(X_val_sorted, y_val_encoded))
```

```
Epoch 1/10
140/140 [==============================] - 1s 8ms/step - loss: 1.2630e-04 -
accuracy: 1.0000 - val_loss: 0.0655 - val_accuracy: 0.9919
Epoch 2/10
140/140 [==============================] - 1s 8ms/step - loss: 9.9521e-05 -
accuracy: 1.0000 - val_loss: 0.0678 - val_accuracy: 0.9919
Epoch 3/10
140/140 [==============================] - 1s 8ms/step - loss: 8.0070e-05 -
accuracy: 1.0000 - val_loss: 0.0696 - val_accuracy: 0.9919
Epoch 4/10
140/140 [==============================] - 1s 8ms/step - loss: 6.5715e-05 -
accuracy: 1.0000 - val_loss: 0.0711 - val_accuracy: 0.9919
Epoch 5/10
140/140 [==============================] - 1s 8ms/step - loss: 5.4318e-05 -
accuracy: 1.0000 - val_loss: 0.0727 - val_accuracy: 0.9919
Epoch 6/10
140/140 [==============================] - 1s 8ms/step - loss: 4.5686e-05 -
accuracy: 1.0000 - val_loss: 0.0741 - val_accuracy: 0.9919
Epoch 7/10
140/140 [==============================] - 1s 8ms/step - loss: 3.8708e-05 -
accuracy: 1.0000 - val_loss: 0.0756 - val_accuracy: 0.9919
Epoch 8/10
140/140 [==============================] - 1s 8ms/step - loss: 3.3232e-05 -
accuracy: 1.0000 - val_loss: 0.0771 - val_accuracy: 0.9919
Epoch 9/10
140/140 [==============================] - 1s 8ms/step - loss: 2.8559e-05 -
accuracy: 1.0000 - val_loss: 0.0780 - val_accuracy: 0.9919
Epoch 10/10
140/140 [==============================] - 1s 8ms/step - loss: 2.4690e-05 -
accuracy: 1.0000 - val_loss: 0.0794 - val_accuracy: 0.9919
```

0.1.15 Ploting the training and validation accuracy over epochs to visualize the learning progress of the model.

The x-axis represents the number of epochs, while the y-axis represents the accuracy.

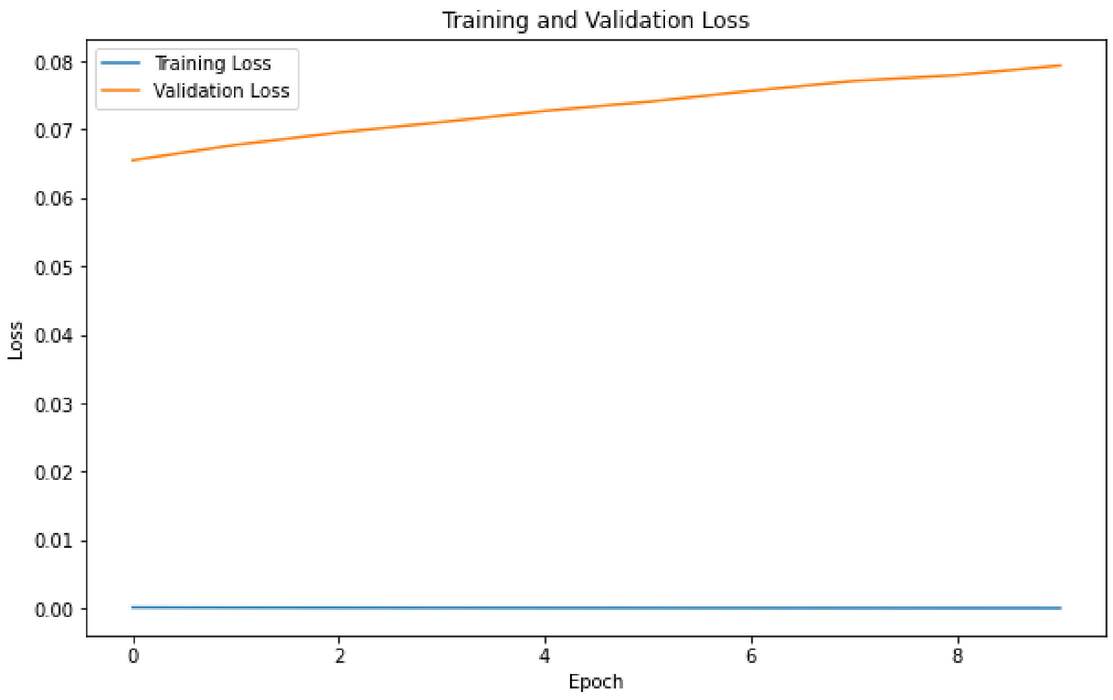
[87]:
```
# Ploting training history
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



0.1.16 Plotting the training and validation loss over epochs to visualize the learning progress of the model.

The x-axis represents the number of epochs, while the y-axis represents the loss (cross-entropy). The plot shows how well the model is minimizing the loss function on both the training and validation datasets during training.

[88]:
```
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



0.2 Summary of the Model output:

Loss and Accuracy: The loss (measured by cross-entropy) decreases consistently over the epochs, indicating that the model is learning and improving its predictions. The accuracy, which measures the proportion of correctly classified emails, remains consistently high, reaching 100% accuracy on the training data and around 99% accuracy on the validation data.

Validation Performance: The validation loss and accuracy are stable across epochs, suggesting that the model generalizes well to unseen data. The slight fluctuations in validation loss and accuracy are normal and could be attributed to the inherent variability in the data or the stochastic nature of the optimization process.

Author: Bahraleloom Abdalrahem **Email:** bahraleloom@gmail.com **GitHub:** https://github.com/Bahraleloom **Date:** 06/05/2024