Double-click (or enter) to edit

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import pandas as pd
import os
import re
import sklearn
from sklearn import preprocessing
from sklearn import model_selection
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
import keras
from keras import preprocessing
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input, SpatialDropout1D, Embeddi
from tensorflow.keras.regularizers import l2
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_repor

pd.set_option('display.max_rows', None)
```

```
Amazon = pd.read_csv('amazon_cells_labelled.txt', sep='\t', header=None, names=['Review',
IMDB = pd.read_csv('imdb_labelled.txt', sep='\t', header=None, names=['Review', 'Sentiment
Yelp = pd.read_csv('yelp_labelled.txt', sep='\t', header=None, names=['Review', 'Sentiment

df = pd.concat(([Amazon, IMDB, Yelp]))

print(df.shape)
print(df.head())
```

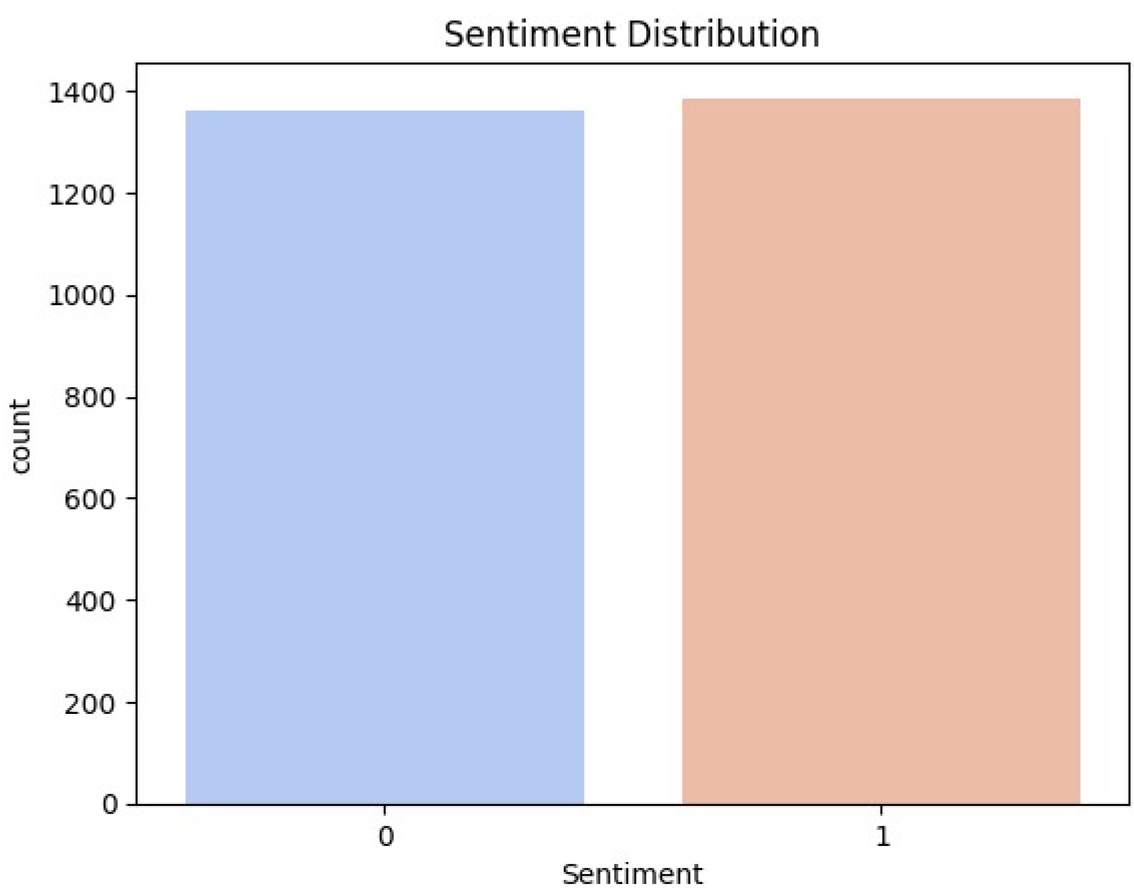
```
(2748, 2)
                                              Review  Sentiment
0  So there is no way for me to plug it in here i...          0
1                         Good case, Excellent value.          1
2                              Great for the jawbone.          1
3  Tied to charger for conversations lasting more...          0
4                                   The mic is great.          1
```

```
print(df.shape)
print(df.head())
print("  ")

# Visualize the sentiment distribution
df['Sentiment'].value_counts()
sns.countplot(x='Sentiment', data=df, palette='coolwarm')
plt.title('Sentiment Distribution')
plt.show()

print("  ")
print(df['Sentiment'].dtype)  # Should be int or float
print("  ")
print(df['Sentiment'].unique())
```

```
(2748, 2)
                                              Review  Sentiment
0  So there is no way for me to plug it in here i...          0
1                        Good case, Excellent value.          1
2                             Great for the jawbone.          1
3  Tied to charger for conversations lasting more...          0
4                                  The mic is great.          1
```




```
int64
```

```
[0 1]
```

```
df.dropna(inplace=True)
df.isnull().sum()
```

	0
Review	0
Sentiment	0

dtype: int64

```
#Initial list of words/characters in reviews
chars_list = []
for comment in df['Review']:
    for character in comment:
        if character not in chars_list:
            chars_list.append(character)
print(chars_list)
```

```
['S', 'o', ' ', 't', 'h', 'e', 'r', 'i', 's', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l',
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt_tab')
nltk.download('wordnet')

stop_words = set(stopwords.words('english'))
lemma = WordNetLemmatizer()

# Process reviews and remove unwanted characters
desc_list = []
for description in df.Review:
    description = re.sub("[^a-zA-Z]", " ", description).lower()  # Remove punctuation and
    description = word_tokenize(description)  # Tokenize text
    description = [lemma.lemmatize(word) for word in description if word not in stop_word
    desc_list.append(' '.join(description))  # Join back into a single string

print("  ")
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['Review'])
print("Vocabulary size: ", len(tokenizer.word_index) + 1)
print("  ")

wordcloud = WordCloud(stopwords=stop_words).generate(' '.join(desc_list))  # Join all pro

# Display and save the word cloud
plt.imshow(wordcloud, interpolation='bilinear')
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.savefig('wordcloud.png')
plt.show()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...

Vocabulary size:  5272
```


```
review_length = []
for char_len in df['Review']:
  review_length.append(len(char_len.split(" ")))

review_max = np.max(review_length)
review_min = np.min(review_length)
review_median = np. median(review_length)
print("Max length is: ", review_max)
print("Median length is ", review_median)
print("Min length is: ", review_min)
print("  ")
```

```
Max length is:  1393
Median length is  11.0
Min length is:  1
```

```
x = np.array(desc_list)
y = df.Sentiment.values

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state

y_train = pd.Series(y_train)
```

```
y_train = pd.Series(y_train)
y_test = pd.Series(y_test)

print("Training size: ", x_train.shape)
print("Testing size: ", x_test.shape)
```

```
Training size:  (2198,)
Testing size:  (550,)
```

```
# Tokenizer setup
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train)  # Fit tokenizer on training data
word_index = tokenizer.word_index
print(word_index)
print("  ")
print("Vocab Size: ", len(tokenizer.word_index) + 1)
print("  ")
print(f"Number of unique words: {len(tokenizer.word_index)}")
print("  ")
print(f"First 10 words in the word index: {list(tokenizer.word_index.items())[:10]}")
print("  ")

# Calculate the maximum, min, and median review length from the training data
review_length = [len(review.split()) for review in x_train]
review_max = np.max(review_length)
review_min = np.min(review_length)
review_median = np.median(review_length)

# Convert text to sequences (convert words to integers)
seq_train = tokenizer.texts_to_sequences(x_train)
padded_train = pad_sequences(seq_train, maxlen=review_max, padding='post', truncating='po

seq_test = tokenizer.texts_to_sequences(x_test)
padded_test = pad_sequences(seq_test, maxlen=review_max, padding='post', truncating='post

print("Max length is: ", review_max)
print("  ")
print("Median length is ", review_median)
print("  ")
print("Min length is: ", review_min)
print("  ")
print(f"Shape of padded training data: {padded_train.shape}")
print("  ")
print(f"Shape of padded test data: {padded_test.shape}")
print("  ")
print("Padded Training Sequences:")
print(pd.DataFrame(padded_train).head())
print("  ")
print("Padded Test Sequences:")
print(pd.DataFrame(padded_test).head())
```

```
{'good': 1, 'great': 2, 'movie': 3, 'phone': 4, 'film': 5, 'time': 6, 'food': 7, 'pla

Vocab Size:  3938

Number of unique words: 3937

First 10 words in the word index: [('good', 1), ('great', 2), ('movie', 3), ('phone',

Max length is:  677

Median length is  5.0

Min length is:  0

Shape of padded training data: (2198, 677)

Shape of padded test data: (550, 677)

Padded Training Sequences:
    0    1     2     3     4    5    6     7    8    9   ...  667  668  669  \
0  176  275     3     0     0    0    0     0    0    0  ...    0    0    0
1   38  217  1046  1662    38   23  741  1663    0    0  ...    0    0    0
2  316  367   137   163  1664    0    0     0    0    0  ...    0    0    0
3   18  317     4  1665  1666  275   50    13    2    0  ...    0    0    0
4   51  427   742     0     0    0    0     0    0    0  ...    0    0    0

   670  671  672  673  674  675  676
0    0    0    0    0    0    0    0
1    0    0    0    0    0    0    0
2    0    0    0    0    0    0    0
3    0    0    0    0    0    0    0
4    0    0    0    0    0    0    0

[5 rows x 677 columns]

Padded Test Sequences:
     0    1     2    3     4    5    6     7    8    9   ...  667  668  669  \
0    49    3   105   85  2217    0    0     0    0    0  ...    0    0    0
1  2437  233    14  155  2761  839  344  1008    0    0  ...    0    0    0
2  3301  835   168  323     0    0    0     0    0    0  ...    0    0    0
3    59   20   737  252     0    0    0     0    0    0  ...    0    0    0
4   107  498  1439    0     0    0    0     0    0    0  ...    0    0    0

   670  671  672  673  674  675  676
0    0    0    0    0    0    0    0
1    0    0    0    0    0    0    0
2    0    0    0    0    0    0    0
3    0    0    0    0    0    0    0
4    0    0    0    0    0    0    0

[5 rows x 677 columns]
```

```
training_padded = np.array(padded_train)
training_label = np.array(y_train)
```

```
test_padded = np.array(padded_test)
test_label = np.array(y_test)

pd.DataFrame(training_padded).to_csv("training_padded.csv")
pd.DataFrame(training_label).to_csv("training_label.csv")
pd.DataFrame(test_padded).to_csv("test_padded.csv")
pd.DataFrame(test_label).to_csv("test_label.csv")


early_stopping_monitor = EarlyStopping(monitor = 'val_loss', patience = 2, restore_best_w
input_dim = len(tokenizer.word_index) + 1
output_dim = 64
adam = Adam(learning_rate = 0.0005)

# Build the LSTM Model
model = Sequential()

model.add(Input(shape = (review_max,)))
model.add(Embedding(input_dim = len(tokenizer.word_index) + 1, output_dim = output_dim))
model.add(SpatialDropout1D(0.5))

model.add(Bidirectional(LSTM(128, dropout=0.6, recurrent_dropout=0.6)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.0001)))
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.0001)))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

# Print the model summary
model.summary()
print("  ")
print(f"Embedding input_dim: {input_dim}")
print(f"Embedding output_dim: {output_dim}")
```

Model: "sequential"

Layer (type)	Output Shape	Param
embedding (Embedding)	(None, 677, 64)	252,03
spatial_dropout1d (SpatialDropout1D)	(None, 677, 64)	
bidirectional (Bidirectional)	(None, 256)	197,63
batch_normalization (BatchNormalization)	(None, 256)	1,02
dropout (Dropout)	(None, 256)	
dense (Dense)	(None, 128)	32,89

dense_1 (Dense)	(None, 64)	8,25
dense_2 (Dense)	(None, 1)	6

```
Total params: 491,905 (1.88 MB)
Trainable params: 491,393 (1.87 MB)
Non-trainable params: 512 (2.00 KB)

Embedding input_dim: 3938
Embedding output_dim: 64
```

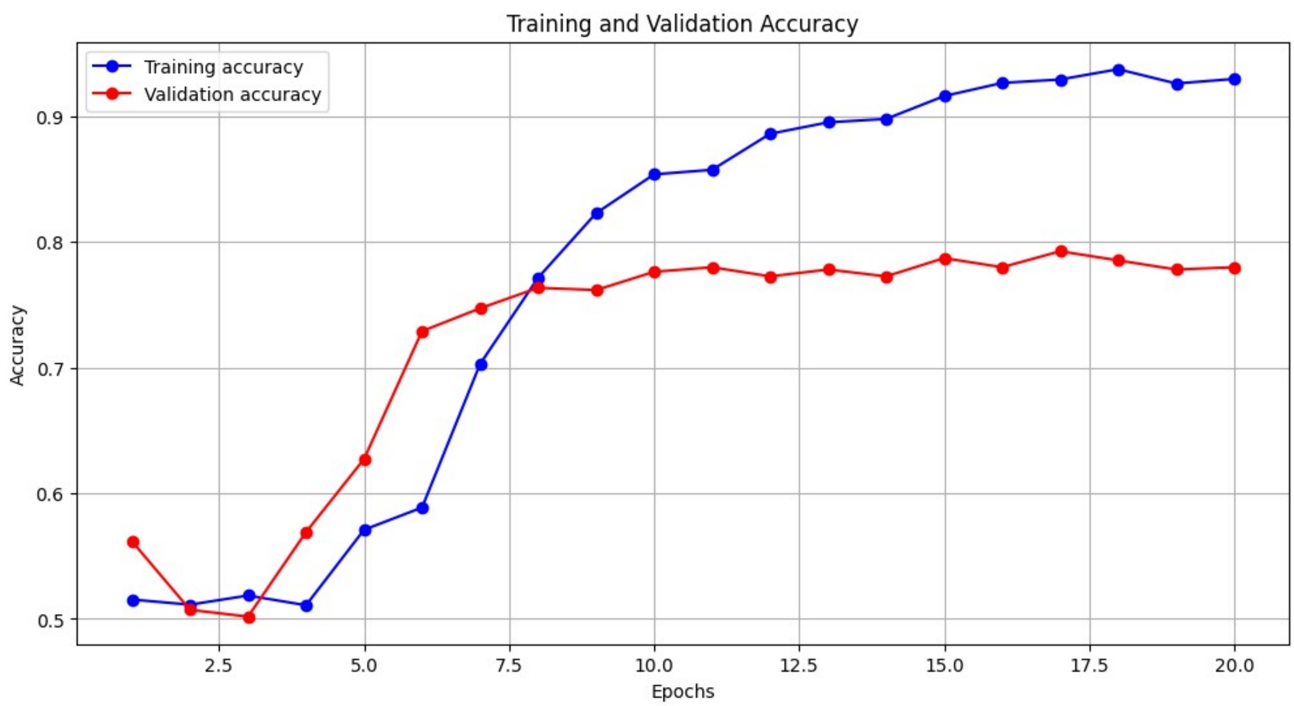
```
history = model.fit(padded_train, training_label, epochs = 20, batch_size = 128, validati
```

```
Epoch 1/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 83s 4s/step - accuracy: 0.5013 - loss: 0.7643 - val_accura
Epoch 2/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 73s 4s/step - accuracy: 0.5223 - loss: 0.7381 - val_accura
Epoch 3/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 72s 4s/step - accuracy: 0.5217 - loss: 0.7257 - val_accura
Epoch 4/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 82s 4s/step - accuracy: 0.4945 - loss: 0.7367 - val_accura
Epoch 5/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 82s 4s/step - accuracy: 0.5646 - loss: 0.7058 - val_accura
Epoch 6/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 82s 4s/step - accuracy: 0.5710 - loss: 0.7048 - val_accura
Epoch 7/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 82s 4s/step - accuracy: 0.6848 - loss: 0.6178 - val_accura
Epoch 8/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 81s 4s/step - accuracy: 0.7741 - loss: 0.5182 - val_accura
Epoch 9/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 73s 4s/step - accuracy: 0.8202 - loss: 0.4440 - val_accura
Epoch 10/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 73s 4s/step - accuracy: 0.8513 - loss: 0.3932 - val_accura
Epoch 11/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 81s 4s/step - accuracy: 0.8547 - loss: 0.3689 - val_accura
Epoch 12/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 82s 4s/step - accuracy: 0.8891 - loss: 0.3017 - val_accura
Epoch 13/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 78s 4s/step - accuracy: 0.8913 - loss: 0.2896 - val_accura
Epoch 14/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 76s 4s/step - accuracy: 0.8959 - loss: 0.2655 - val_accura
Epoch 15/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 73s 4s/step - accuracy: 0.9150 - loss: 0.2465 - val_accura
Epoch 16/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 72s 4s/step - accuracy: 0.9271 - loss: 0.2222 - val_accura
Epoch 17/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 83s 4s/step - accuracy: 0.9252 - loss: 0.2226 - val_accura
Epoch 18/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 81s 4s/step - accuracy: 0.9442 - loss: 0.1879 - val_accura
Epoch 19/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 81s 4s/step - accuracy: 0.9263 - loss: 0.2179 - val_accura
Epoch 20/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 79s 4s/step - accuracy: 0.9311 - loss: 0.1879 - val_accura
```

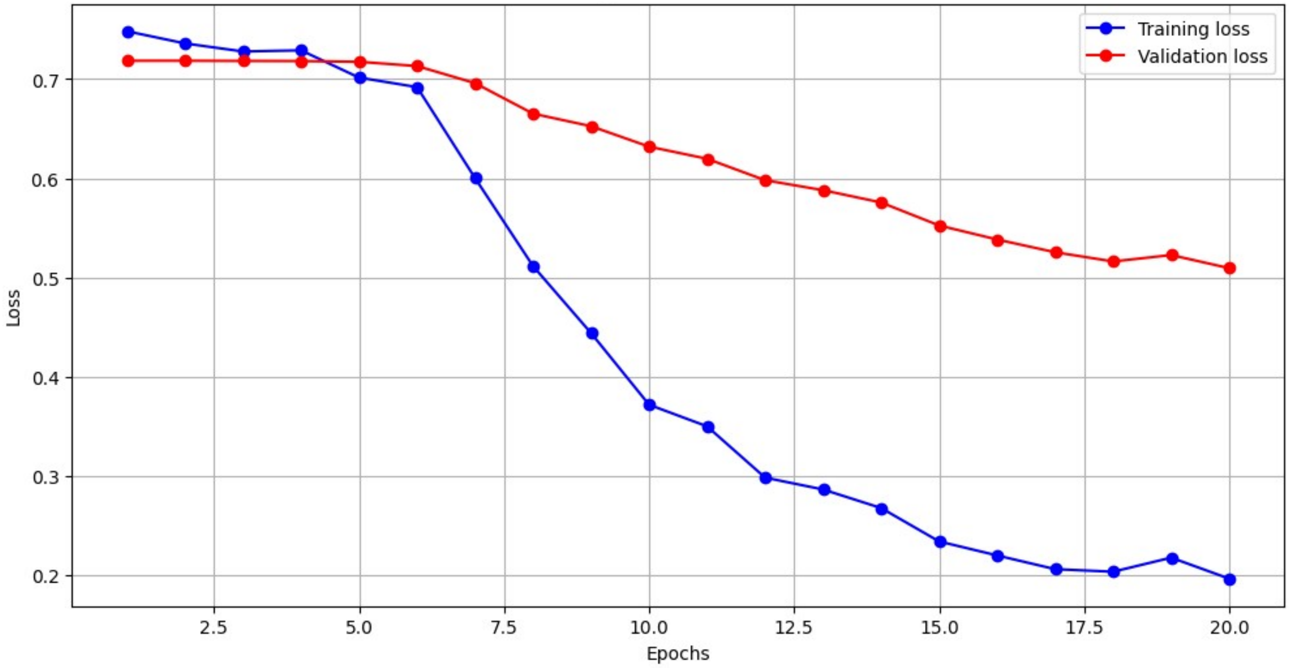
```
# Get the training and validation accuracy and loss
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_accuracy) + 1)  # Epochs range

# Plot training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(epochs, train_accuracy, 'bo-', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'ro-', label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Plot training and validation loss
plt.figure(figsize=(12, 6))
plt.plot(epochs, train_loss, 'bo-', label='Training loss')
plt.plot(epochs, val_loss, 'ro-', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



Training and Validation Loss




```
model.save('model.keras')
```