In [6]:
```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud,STOPWORDS
from bs4 import BeautifulSoup
import re,string,unicodedata

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, plot_confusion_matrix, plot_roc_curve,
from xgboost.sklearn import XGBClassifier

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense,Input, Embedding,LSTM,Dropout,Conv1D, MaxPooling1D, GlobalMaxPooling1D,Dropout,Bid
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
import transformers
import tokenizers
```

importing csv

In [7]:
```
df= pd.read_csv(r"C:\Users\LENOVO\Desktop\IMDB Dataset.csv")
df.head()
```

Out[7]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

CLEANING

In [8]:
```
# Finding Duplicates in the data
num_duplicates = df.duplicated().sum()
print(f"the number of duplicates in the the data", format(num_duplicates))
review = df['review']
duplicate_review = df[review.isin(review[review.duplicated()])].sort_values('review')
duplicate_review.head()

# dropping the duplicate values
df.drop_duplicates(inplace =True)
```

```
the number of duplicates in the the data 418
```

In [9]:
```
# Finding Duplicates in the data
num_duplicates = df.duplicated().sum()
print(f"the number of duplicates in the the data", format(num_duplicates))
review = df['review']
duplicate_review = df[review.isin(review[review.duplicated()])].sort_values('review')
duplicate_review.head()

# dropping the duplicate values
df.drop_duplicates(inplace =True)
```

```
the number of duplicates in the the data 0
```

In [10]:
```
# Converting the text values of sentiments into numbers
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
df['sentiment']= label_encoder.fit_transform(df['sentiment'])
```

In [11]:
```
stop = stopwords.words('english')
wl = WordNetLemmatizer()
```

In [12]:
```
mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot",
           "'cause": "because", "could've": "could have", "couldn't": "could not",
           "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't": "had not",
           "hasn't": "has not", "haven't": "have not", "he'd": "he would","he'll": "he will",
           "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll": "how will",
           "how's": "how is",  "I'd": "I would", "I'd've": "I would have", "I'll": "I will",
           "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i would",
           "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have",
           "i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it would",
           "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have",
           "it's": "it is", "let's": "let us", "ma'am": "madam", "mayn't": "may not",
           "might've": "might have","mightn't": "might not","mightn't've": "might not have",
           "must've": "must have", "mustn't": "must not", "mustn't've": "must not have",
           "needn't": "need not", "needn't've": "need not have","o'clock": "of the clock",
           "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not",
           "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she would",
           "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have",
           "she's": "she is", "should've": "should have", "shouldn't": "should not",
```

```
            "shouldn't've": "should not have", "so've": "so have","so's": "so as", "this's": "this is",
            "that'd": "that would", "that'd've": "that would have", "that's": "that is",
            "there'd": "there would", "there'd've": "there would have", "there's": "there is",
            "here's": "here is","they'd": "they would", "they'd've": "they would have",
            "they'll": "they will", "they'll've": "they will have", "they're": "they are",
            "they've": "they have", "to've": "to have", "wasn't": "was not", "we'd": "we would",
            "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have",
            "we're": "we are", "we've": "we have", "weren't": "were not",
            "what'll": "what will", "what'll've": "what will have","what're": "what are",
            "what's": "what is", "what've": "what have", "when's": "when is", "when've": "when have",
            "where'd": "where did", "where's": "where is", "where've": "where have", "who'll": "who will",
            "who'll've": "who will have", "who's": "who is", "who've": "who have", "why's": "why is",
            "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have",
            "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have",
            "y'all": "you all", "y'all'd": "you all would","y'all'd've": "you all would have",
            "y'all're": "you all are","y'all've": "you all have","you'd": "you would",
            "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have",
            "you're": "you are", "you've": "you have" }
```

In [13]:
```
clean data
rt BeautifulSoup
ing,unicodedata
t(text,lemmatize = True):
autifulSoup(text, "html.parser") #remove html tags
up.get_text()
'.join([mapping[t] if t in mapping else t for t in text.split(" ")]) #expanding chatwords and contracts clearing contractions
an= re.compile("["
             u"\U0001F600-\U0001F64F"  # emoticons
             u"\U0001F300-\U0001F5FF"  # symbols & pictographs
             u"\U0001F680-\U0001F6FF"  # transport & map symbols
             u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
             u"\U00002702-\U000027B0"
             u"\U000024C2-\U0001F251"
             "]+", flags=re.UNICODE)
oji_clean.sub(r'',text)
.sub(r'\.(?=\S)', '. ',text) #add space after full stop
.sub(r'http\S+', '', text) #remove urls
.join([word.lower() for word in text if word not in string.punctuation]) #remove punctuation
 re.split('\W+', text) #create tokens
ize:
= " ".join([wl.lemmatize(word) for word in text.split() if word not in stop and word.isalpha()]) #Lemmatize

= " ".join([word for word in text.split() if word not in stop and word.isalpha()])
xt

```

In [14]:
```
data_copy = df.copy()
```

In [16]:
```
# df['review']=df['review'].apply(clean_text,lemmatize = True)
```

In [17]:
```
df.head()
```

Out[17]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

text processing

In [18]:
```
def convert_lowercase(column):
    column = column.str.lower()
    return column

df['review'] = convert_lowercase(df['review'])
```

In [19]:
```
def remove_html_tags(text):
    re_html = re.compile('<.*?>')
    return re_html.sub(r'', text)

df['review'] = df['review'].apply(remove_html_tags)
```

In [20]:
```
def remove_url(text):
    re_url = re.compile('https?://\S+|www\.\S+')
    return re_url.sub('', text)

df['review'] = df['review'].apply(remove_url)
```

In [21]:
```
df.head()
```

Out[21]:

	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production. the filming tec...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically there's a family where a little boy ...	0
4	petter mattei's "love in the time of money" is...	1

In [22]:
```
df.to_csv('imdb_rough_practicemetrices.csv')
```

TEST AND TRAIN

In [23]:
```
###################
from sklearn.model_selection import train_test_split
train_text, test_text, train_labels, test_labels = train_test_split(df["review"],df["sentiment"], test_size=0.33)
```

In [24]:
```
########################
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

#Creates and fits a TensorFlow Tokenizer
tokenizer = Tokenizer(num_words=10_000, oov_token='<OOV>')
tokenizer.fit_on_texts(train_text)
```

```
#Creates sequences of numeric representations of words
training_sequences = tokenizer.texts_to_sequences(train_text)
#pads sequences so they all have the same length
training_sequences = pad_sequences(training_sequences, maxlen=20)

#Process test data in the same way for later evaluation
testing_sequences = tokenizer.texts_to_sequences(test_text)
testing_sequences = pad_sequences(testing_sequences, maxlen=20)


#prints a sample of the new sequences
print('PROCESSED TEXT DATA')
print('========================')
for i in range(5):
    print(training_sequences[i], '\n')
```

```
PROCESSED TEXT DATA
========================
[   4 1391   11   19    7   27    5    2   88  312  199  122  106    3
  199  106    4  168    5   98]

[6230    5    1   48    1    7  636   11   19    7   27    5   57  342
  867  874  104    5   29   55]

[   4  406 1316   15   10 4650   29  344   16    4  337  726  839   62
   11   15   17   12  305  396]

[  19   17   29    2 4739    3 1044    5   32 1899    1  527    4 2928
    1   21    4    1    1  828]

[  20   44 1559  388   37   12   12   94    2  394  658   21   39    4
 3293 2061   18  246   14   70]
```

In [25]:
```
train_labels.reset_index(inplace=True, drop=True)
```

callback

In [85]:
```
# CUSTOM CALLBACK:
from tensorflow.keras.callbacks import Callback

class CustomCallback(Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('accuracy') > 0.93:
            print("Accuracy over 95%... Stopping training")
            self.model.stop_training = True

my_callback = CustomCallback()
```

In [86]:
```
# PRE-DEFINED CALLBACK:
from tensorflow.keras.callbacks import LearningRateScheduler

#creates a function that updates the learning rate based on the epoch number
def scheduler(epoch, lr):
    if epoch < 2:
        return 0.01
    else:
        return lr * 0.99

lr_scheduler = LearningRateScheduler(scheduler)
```

model architecture:

In [87]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
from tensorflow.keras.optimizers import Adam

#input dimensions is equal to number of words tokenized (defined above)
input_dim = 10_000
# input length will be the length of our padded sequences
input_length = 20


#defines a text classifier model
model = Sequential([
    Embedding(input_dim=input_dim, output_dim=64, input_length=input_length),
    Bidirectional(LSTM(150)),
    Dropout(0.4),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(
    loss='binary_crossentropy',
    optimizer=Adam(weight_decay=0.08),
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 20, 64)            640000

 bidirectional_3 (Bidirectio  (None, 300)              258000
 nal)

 dropout_6 (Dropout)         (None, 300)               0

 dense_6 (Dense)             (None, 512)               154112

 dropout_7 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 1)                 513

=================================================================
Total params: 1,052,625
Trainable params: 1,052,625
Non-trainable params: 0
```

In [88]:
```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
df['sentiment']= label_encoder.fit_transform(df['sentiment'])
```

train the model

In [89]:
```
history = model.fit(
    np.array(training_sequences), #must convert to numpy array before sending to model
    np.array(train_labels),       #must convert to numpy array before sending to model
    epochs=100,
    batch_size=128,
    callbacks=[my_callback, lr_scheduler],verbose=1)
```

```
Epoch 1/100
260/260 [==============================] - 17s 58ms/step - loss: 0.5283 - accuracy: 0.7336 - lr: 0.0100
Epoch 2/100
260/260 [==============================] - 15s 58ms/step - loss: 0.4026 - accuracy: 0.8172 - lr: 0.0100
Epoch 3/100
260/260 [==============================] - 15s 58ms/step - loss: 0.3127 - accuracy: 0.8663 - lr: 0.0099
Epoch 4/100
260/260 [==============================] - 15s 58ms/step - loss: 0.2394 - accuracy: 0.9024 - lr: 0.0098
Epoch 5/100
260/260 [==============================] - 15s 58ms/step - loss: 0.1907 - accuracy: 0.9227 - lr: 0.0097
Epoch 6/100
259/260 [============================>.] - ETA: 0s - loss: 0.1674 - accuracy: 0.9336Accuracy over 95%... Stopping training
260/260 [==============================] - 15s 58ms/step - loss: 0.1674 - accuracy: 0.9336 - lr: 0.0096
```

In [90]:
```
# precision for training data prediction
from sklearn.metrics import precision_score
predicted_train_labels = model.predict(np.array(training_sequences))
precision_train = precision_score(train_labels, predicted_train_labels.round())
print('Training data precision =',precision_train)
```

```
1039/1039 [==============================] - 5s 5ms/step
Training data precision = 0.9648892338926375
```

In [91]:
```
# recall for training data prediction
from sklearn.metrics import recall_score
predicted_train_labels = model.predict(np.array(training_sequences))
recall_train = recall_score(train_labels, predicted_train_labels.round())
print('Training recall data =',recall_train)
```

```
1039/1039 [==============================] - 5s 5ms/step
Training recall data = 0.9724963890226288
```

In [92]:
```
 # recall for training data prediction
from sklearn.metrics import f1_score
predicted_train_labels = model.predict(np.array(training_sequences))
f1_score_train = f1_score(train_labels, predicted_train_labels.round())
print('Training data f1_score =',f1_score_train)
```

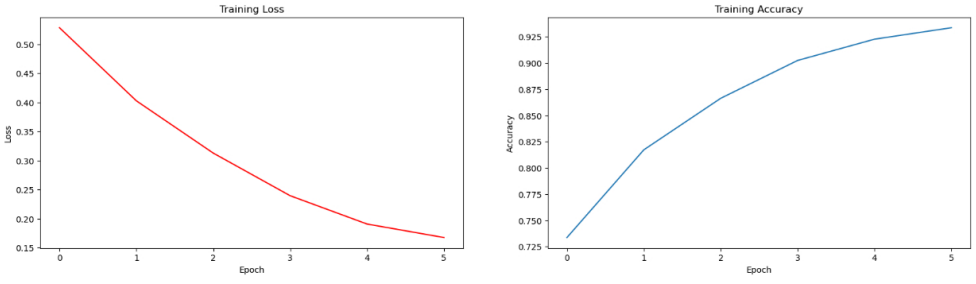
```
1039/1039 [==============================] - 5s 5ms/step
Training data f1_score = 0.968677876689746
```

In [ ]:
```

```

plotting the model loss and accuracy:

In [93]:
```
#Plots history of model training
plt.rcParams["figure.figsize"] = (20,5)
fig, axs = plt.subplots(1, 2)

axs[0].plot(history.history['loss'], color='red')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Loss')
axs[0].set_title('Training Loss')

axs[1].plot(history.history['accuracy'])
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Accuracy')
axs[1].set_title('Training Accuracy')
```

Out[93]:
```
Text(0.5, 1.0, 'Training Accuracy')
```



let's test:

In [94]:
```
NEW_REVIEW =\
"""
he commited a crime
"""
```

In [95]:
```
#Process the new review the same way the test text was processed
new_review_sequence = tokenizer.texts_to_sequences([NEW_REVIEW])
new_review_sequence = pad_sequences(new_review_sequence, maxlen=20)

#sends new review to be predicted by the model
new_review_prediction = round(model.predict(np.array(new_review_sequence))[0][0])
sentiment = "NEGATIVE" if new_review_prediction == 0 else "POSITIVE"

#displays what the model thinks the sentiment of the review was
```

```
print("MOVIE REVIEW:", NEW_REVIEW)
print("MODEL PREDICTED SENTIMENT:", sentiment)
```

```
1/1 [==============================] - 0s 17ms/step
MOVIE REVIEW:
he commited a crime

MODEL PREDICTED SENTIMENT: NEGATIVE
```

eval model:¶

In [98]:
```
loss, accuracy = model.evaluate(testing_sequences, test_labels)
```

```
512/512 [==============================] - 3s 5ms/step - loss: 0.7837 - accuracy: 0.7427
```

In [99]:
```
# recall for training data prediction
from sklearn.metrics import recall_score
predicted_test_labels = model.predict(np.array(testing_sequences))
recall_test = recall_score(test_labels, predicted_test_labels.round())
print('Testing recall data =',recall_test)
```

```
512/512 [==============================] - 3s 5ms/step
Testing recall data = 0.7624576681180455
```

In [100]:
```
f1_score_test = f1_score(test_labels, predicted_test_labels.round())
print('Testing data f1_score =',f1_score_test)
```

```
Testing data f1_score = 0.7496283964563886
```

In [101]:
```
precision_score_test = precision_score(test_labels, predicted_test_labels.round())
print('Testing data precision_score =', precision_score_test)
```

```
Testing data precision_score = 0.7372237165243831
```

In [112]:
```
y_pred = model.predict(testing_sequences)

```

```
512/512 [==============================] - 2s 5ms/step
```

In [113]:
```
print(y_pred)
```

```
[[9.2250073e-01]
 [9.9997121e-01]
 [3.7033834e-02]
 ...
 [5.5549435e-06]
 [8.0901891e-08]
 [9.2241624e-03]]
```

In [114]:
```
from sklearn.metrics import confusion_matrix

# Get the true labels for the test set
y_true = test_labels


```

In [116]:
```
# Generate predicted probabilities for the test set
y_pred_prob = model.predict(testing_sequences)

# Convert probabilities to binary values
y_pred = (y_pred_prob > 0.5).astype(int)
```

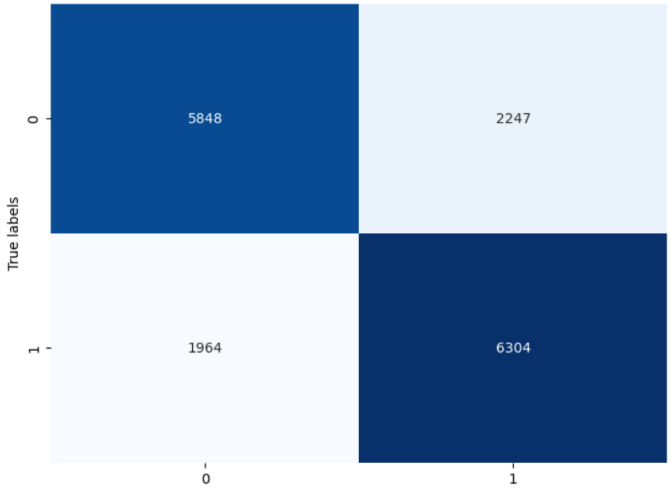
```
512/512 [==============================] - 2s 5ms/step
```

In [117]:
```

# Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Print the confusion matrix
print(cm)
```

```
[[5848 2247]
 [1964 6304]]
```

In [120]:
```
model.save('IMDB_LSTM(hululu).h5')
```

In [121]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```



Predicted labels

In [122]:
```
# Extract true positives, true negatives, false positives, false negatives
tn, fp, fn, tp = cm.ravel()

# Print the counts
print("True positives:", tp)
print("True negatives:", tn)
print("False positives:", fp)
print("False negatives:", fn)
```

```
True positives: 6304
True negatives: 5848
False positives: 2247
False negatives: 1964
```

In [ ]:
```

```