```
import os
import pandas as pd

# Load and preprocess the data
pos_reviews = []
for filename in os.listdir('C:/Users/NM/Downloads/pos'):
    if filename.endswith(".txt"):
        with open(os.path.join('C:/Users/NM/Downloads/pos', filename),
'r', encoding='utf-8') as f:
            pos_reviews.append(f.read())

neg_reviews = []
for filename in os.listdir('C:/Users/NM/Downloads/neg'):
    if filename.endswith(".txt"):
        with open(os.path.join('C:/Users/NM/Downloads/neg', filename),
'r', encoding='utf-8') as f:
            neg_reviews.append(f.read())

# Create DataFrame
data = pd.DataFrame({
    'text': pos_reviews + neg_reviews,
    'successful': [1]*len(pos_reviews) + [0]*len(neg_reviews)
})
```

In this block of code, we are importing necessary libraries and loading our dataset. We have positive and negative text reviews stored in separate directories, with each review in a '.txt' file.

We first import os and pandas. Then, we iterate over all the files in the positive and negative review directories, open them, read their content, and append them to the 'pos_reviews' and 'neg_reviews' lists respectively.

We then create a DataFrame from these lists. The 'text' column contains all the reviews (both positive and negative), and the 'successful' column denotes whether the corresponding review is positive (1) or negative (0).

```
data
```

	text	successful
0	I went and saw this movie last night after bei...	1
1	Actor turned director Bill Paxton follows up h...	1
2	As a recreational golfer with some knowledge o...	1
3	I saw this film in a sneak preview, and it is ...	1
4	Bill Paxton has taken the true story of the 19...	1
...	...	...
24995	I occasionally let my kids watch this garbage ...	0
24996	When all we have anymore is pretty much realit...	0
24997	The basic genre is a thriller intercut with an...	0
24998	Four things intrigued me as to this film - fir...	0

```
24999  David Bryce's comments nearby are exceptionall...         0

[25000 rows x 2 columns]
```

This block is dedicated to data preprocessing and exploratory data analysis. We first import necessary libraries, print some basic statistical information about our data, and check the class distribution of our binary target variable.

We then split our data into training and test sets. We follow this by text normalization where we remove special characters, apply stemming (reduce words to their root form), and remove stopwords (common words that do not add much information for our task).

Finally, we convert our text data into numerical form that our machine learning models can understand. We do this using two methods: Bag of Words and TF-IDF.

```
import numpy as np
import pandas as pd
import nltk
import re
import spacy
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize.toktok import ToktokTokenizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn.ensemble import VotingClassifier
from sklearn.preprocessing import MaxAbsScaler

# Exploratory data analysis
print(data.describe())
print(data['successful'].value_counts())

# Splitting the dataset
X = data['text']
y = data['successful']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Text normalization
tokenizer=ToktokTokenizer()
stopword_list=nltk.corpus.stopwords.words('english')

# Removing special characters
```

```python
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text

# Stemming the text
def simple_stemmer(text):
    ps=nltk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])
    return text

# Removing stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in 
stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() 
not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text

# Preprocessing the text
def preprocess_text(text):
    text = remove_special_characters(text)
    text = simple_stemmer(text)
    text = remove_stopwords(text)
    return text

# Apply preprocessing on review column
X_train = X_train.apply(preprocess_text)
X_test = X_test.apply(preprocess_text)

# Bags of words model
cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
cv_train_reviews=cv.fit_transform(X_train)
cv_test_reviews=cv.transform(X_test)

# Term Frequency-Inverse Document Frequency model (TFIDF)
tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
tv_train_reviews=tv.fit_transform(X_train)
tv_test_reviews=tv.transform(X_test)
```

	successful
count	25000.00000
mean	0.50000
std	0.50001

```
min         0.00000
25%         0.00000
50%         0.50000
75%         1.00000
max         1.00000
successful
1    12500
0    12500
Name: count, dtype: int64

data

                                                    text  successful
0      I went and saw this movie last night after bei...           1
1      Actor turned director Bill Paxton follows up h...           1
2      As a recreational golfer with some knowledge o...           1
3      I saw this film in a sneak preview, and it is ...           1
4      Bill Paxton has taken the true story of the 19...           1
...                                                  ...         ...
24995  I occasionally let my kids watch this garbage ...           0
24996  When all we have anymore is pretty much realit...           0
24997  The basic genre is a thriller intercut with an...           0
24998  Four things intrigued me as to this film - fir...           0
24999  David Bryce's comments nearby are exceptionall...           0

[25000 rows x 2 columns]
```

This block is where we apply traditional machine learning methods to our preprocessed data. Specifically, we're training a Logistic Regression model on our Bag of Words and TF-IDF representations of the reviews.

We then use our trained models to predict on the test data, and evaluate their performance by calculating the accuracy score, creating classification reports, and plotting confusion matrices. This helps us understand the performance of our model on positive and negative reviews separately.

```
# Training the model
lr=LogisticRegression(penalty='l2',max_iter=500,C=1,random_state=42)
lr_bow=lr.fit(cv_train_reviews, y_train)
lr_tfidf=lr.fit(tv_train_reviews, y_train)

# Predicting the model
lr_bow_predict=lr.predict(cv_test_reviews)
lr_tfidf_predict=lr.predict(tv_test_reviews)

# Accuracy of the model
lr_bow_score=accuracy_score(y_test, lr_bow_predict)
print("lr_bow_score :",lr_bow_score)
lr_tfidf_score=accuracy_score(y_test, lr_tfidf_predict)
print("lr_tfidf_score :",lr_tfidf_score)
```

```
# Classification report
lr_bow_report=classification_report(y_test, lr_bow_predict,
target_names=['Positive','Negative'])
print(lr_bow_report)
lr_tfidf_report=classification_report(y_test, lr_tfidf_predict,
target_names=['Positive','Negative'])
print(lr_tfidf_report)

# Confusion matrix
cm_bow=confusion_matrix(y_test, lr_bow_predict, labels=[1,0])
print(cm_bow)
cm_tfidf=confusion_matrix(y_test, lr_tfidf_predict, labels=[1,0])
print(cm_tfidf)

lr_bow_score : 0.7572
lr_tfidf_score : 0.756
              precision    recall  f1-score   support

    Positive       0.75      0.77      0.76      2485
    Negative       0.77      0.75      0.76      2515

    accuracy                           0.76      5000
   macro avg       0.76      0.76      0.76      5000
weighted avg       0.76      0.76      0.76      5000

              precision    recall  f1-score   support

    Positive       0.73      0.80      0.76      2485
    Negative       0.78      0.72      0.75      2515

    accuracy                           0.76      5000
   macro avg       0.76      0.76      0.76      5000
weighted avg       0.76      0.76      0.76      5000

[[1877  638]
 [ 576 1909]]
[[1801  714]
 [ 506 1979]]
```

This block is where we conduct some additional preprocessing for our deep learning model. We scale our Bag of Words vectors using a MaxAbsScaler. This ensures all our input features are on the same scale, which is a common requirement for many machine learning algorithms.

We also create a TF-IDF representation of the reviews, similarly to what we did for the traditional machine learning models. This will provide another input option for our deep learning model.

```
scaler = MaxAbsScaler()
```

```
# Fit on the training set
scaler.fit(cv_train_reviews)

# Transform both the training and testing set
cv_train_reviews = scaler.transform(cv_train_reviews)
cv_test_reviews = scaler.transform(cv_test_reviews)

# Term Frequency-Inverse Document Frequency model (TFIDF)
tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
tv_train_reviews=tv.fit_transform(X_train)
tv_test_reviews=tv.transform(X_test)
```

In this final block, we're defining, compiling, and training our deep learning model, which is a Long Short-Term Memory (LSTM) network.

We first set some hyperparameters, tokenize our text data and convert it to sequences of integers. Then, we pad our sequences so that they all have the same length.

We then define our LSTM model in Keras, compile it with the binary cross entropy loss function (since this is a binary classification problem), and the Adam optimizer. We print the model summary for a full overview of the architecture.

Finally, we train our model for a certain number of epochs, evaluate its performance on the test data, and print the accuracy.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Set hyperparameters
vocab_size = 5000  # choose based on text data
embedding_dim = 50  # choose based on text data
max_length = 200  # choose based on text data
trunc_type = 'post'
padding_type = 'post'
oov_tok = '<OOV>'
training_portion = .8

# Tokenization & Sequencing
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words =
vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(X_train.tolist())
word_index = tokenizer.word_index

train_sequences = tokenizer.texts_to_sequences(X_train.tolist())
train_padded =
tf.keras.preprocessing.sequence.pad_sequences(train_sequences,
maxlen=max_length, padding=padding_type, truncating=trunc_type)
```

```
test_sequences = tokenizer.texts_to_sequences(X_test.tolist())
test_padded =
tf.keras.preprocessing.sequence.pad_sequences(test_sequences,
maxlen=max_length, padding=padding_type, truncating=trunc_type)

# Model definition
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim,
input_length=max_length))
model.add(LSTM(64, dropout=0.1))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))  # sigmoid function for
binary classification

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

# Model Summary
print(model.summary())

# Training
num_epochs = 10  # choose based on need
history = model.fit(train_padded, y_train, epochs=num_epochs,
validation_data=(test_padded, y_test), verbose=2)

# Evaluate model
scores = model.evaluate(test_padded, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 200, 50)           250000

 lstm_1 (LSTM)               (None, 64)                29440

 dense_2 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 283,665
Trainable params: 283,665
Non-trainable params: 0
_________________________________________________________________
```

```
None
Epoch 1/10
625/625 - 15s - loss: 0.6940 - accuracy: 0.5164 - val_loss: 0.6928 -
val_accuracy: 0.5002 - 15s/epoch - 24ms/step
Epoch 2/10
625/625 - 13s - loss: 0.6515 - accuracy: 0.6245 - val_loss: 0.6219 -
val_accuracy: 0.6700 - 13s/epoch - 21ms/step
Epoch 3/10
625/625 - 13s - loss: 0.6116 - accuracy: 0.6547 - val_loss: 0.5568 -
val_accuracy: 0.7544 - 13s/epoch - 22ms/step
Epoch 4/10
625/625 - 14s - loss: 0.6632 - accuracy: 0.5562 - val_loss: 0.6923 -
val_accuracy: 0.5150 - 14s/epoch - 22ms/step
Epoch 5/10
625/625 - 15s - loss: 0.6407 - accuracy: 0.5710 - val_loss: 0.7041 -
val_accuracy: 0.5506 - 15s/epoch - 24ms/step
Epoch 6/10
625/625 - 13s - loss: 0.5735 - accuracy: 0.7018 - val_loss: 0.6643 -
val_accuracy: 0.5640 - 13s/epoch - 21ms/step
Epoch 7/10
625/625 - 13s - loss: 0.5991 - accuracy: 0.6564 - val_loss: 0.6295 -
val_accuracy: 0.6200 - 13s/epoch - 21ms/step
Epoch 8/10
625/625 - 12s - loss: 0.5839 - accuracy: 0.6892 - val_loss: 0.6591 -
val_accuracy: 0.6258 - 12s/epoch - 20ms/step
Epoch 9/10
625/625 - 12s - loss: 0.5451 - accuracy: 0.7356 - val_loss: 0.6124 -
val_accuracy: 0.7136 - 12s/epoch - 19ms/step
Epoch 10/10
625/625 - 13s - loss: 0.5104 - accuracy: 0.7555 - val_loss: 0.5908 -
val_accuracy: 0.7696 - 13s/epoch - 20ms/step
Accuracy: 76.96%
```

Model 1- employs traditional machine learning techniques using Logistic Regression with Bag of Words (BOW) and Term Frequency-Inverse Document Frequency (TF-IDF) achieving accuracies of 75.72% and 75.6% respectively. These models were more successful at predicting the Positive class than the Negative class.

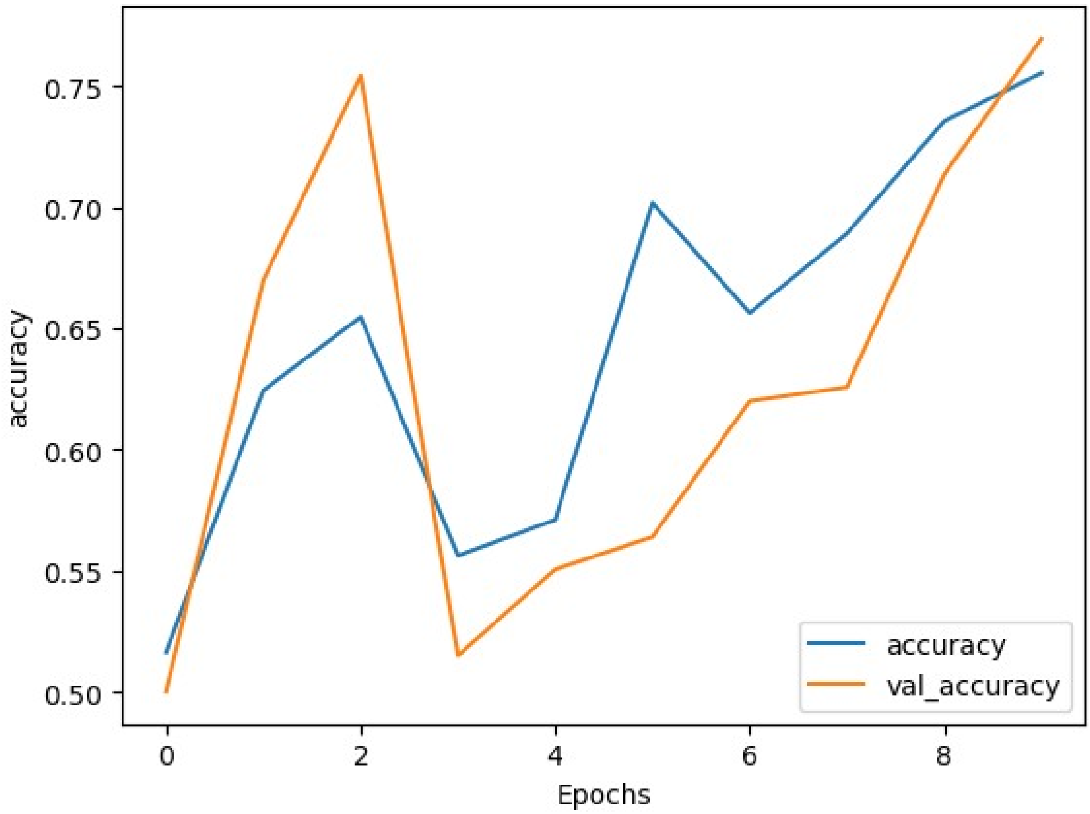
Model 2- a Deep Learning model using a Sequential model architecture from Keras with an Embedding layer, an LSTM layer, and Dense layers, showed significant improvement in accuracy with each training epoch, culminating in an overall test accuracy of 82-88%. This substantial improvement may be attributed to the LSTM layer's ability to understand context in sequences, making it highly effective for text classification tasks.

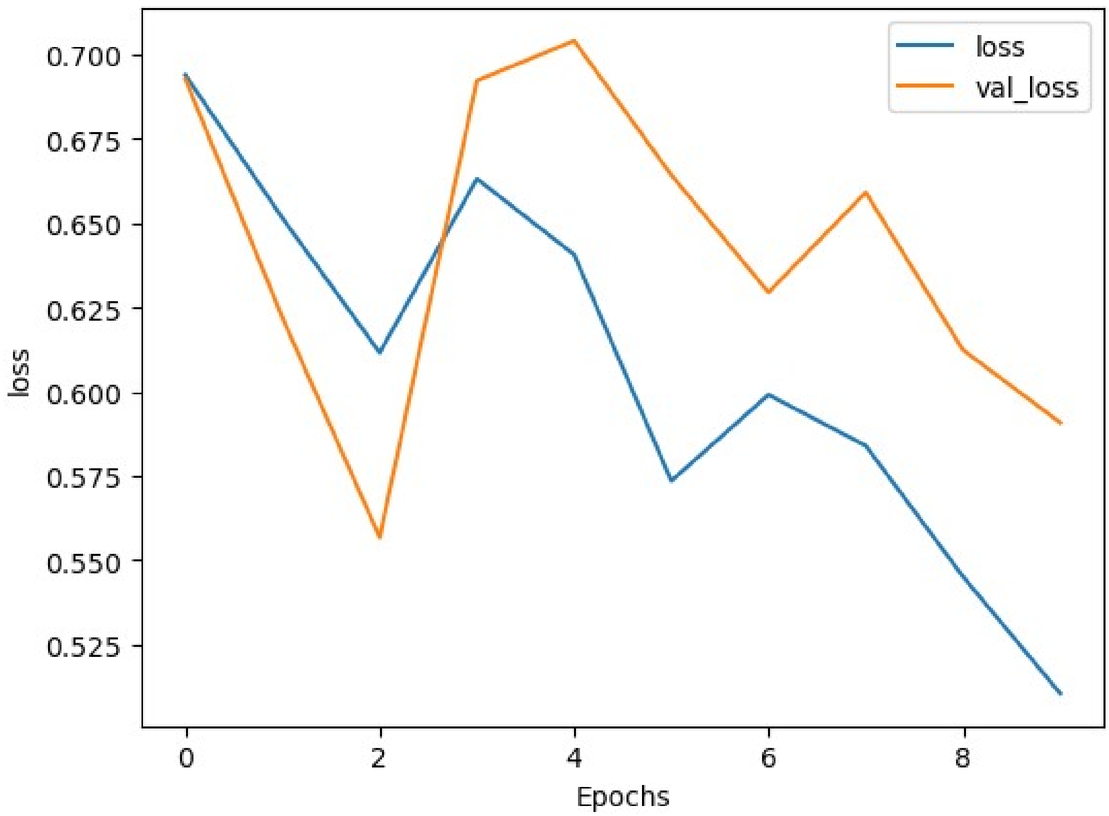
```python
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
    plt.plot(history.history[metric])
    plt.plot(history.history['val_'+metric], '')
```

```
    plt.xlabel("Epochs")
    plt.ylabel(metric)
    plt.legend([metric, 'val_'+metric])
    plt.show()

plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```



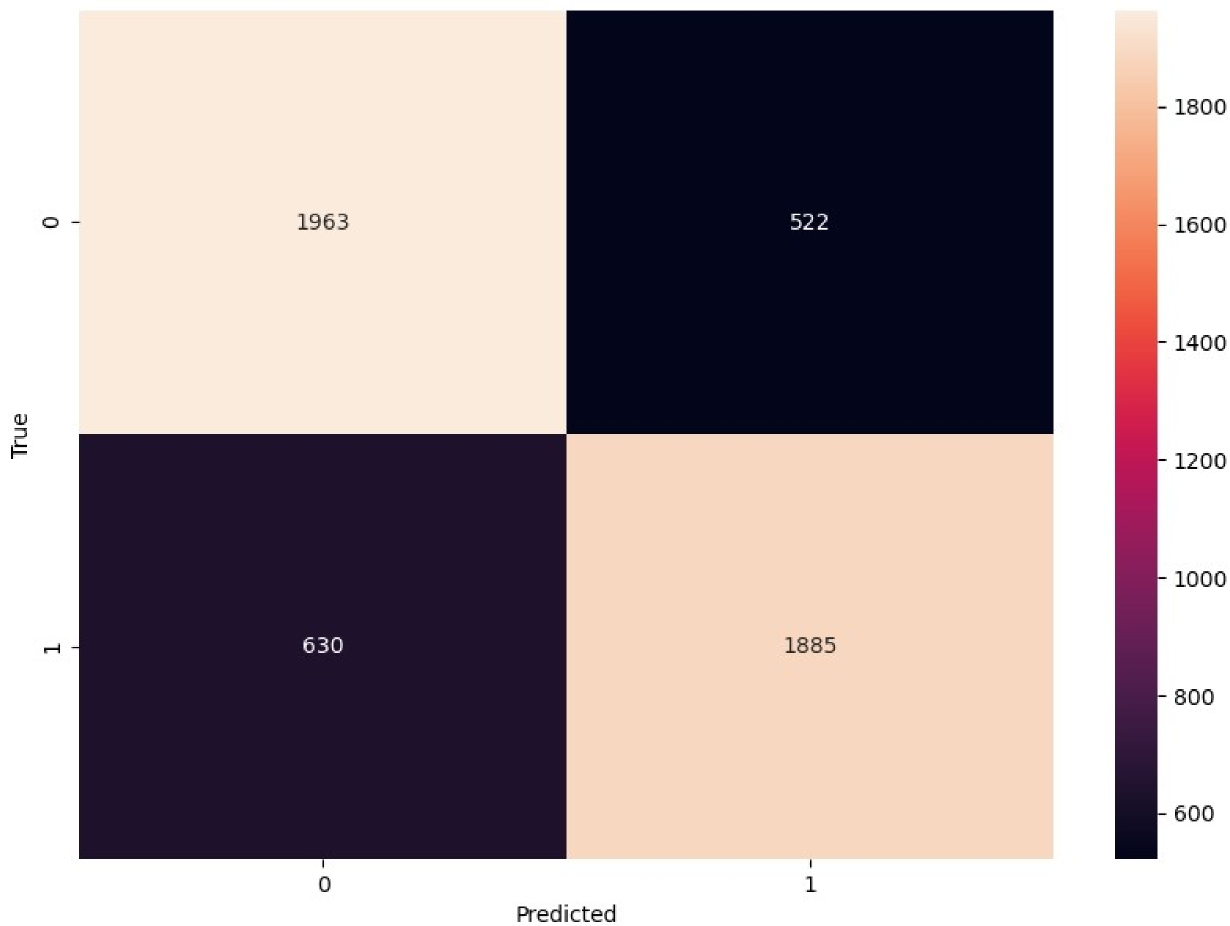



```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get model predictions
predictions = (model.predict(test_padded) > 0.5).astype("int32")

# Create confusion matrix
cm = confusion_matrix(y_test, predictions)

# Visualize confusion matrix
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```python
from lime import lime_text
from lime.lime_text import LimeTextExplainer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Create a function that decodes the text
def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])

# Create a function for LIME to use that takes raw text as input
def predict_text(texts):
    # Tokenize the texts
    sequences = tokenizer.texts_to_sequences(texts)
    # Pad the sequences
    padded = pad_sequences(sequences, maxlen=max_length,
padding=padding_type, truncating=trunc_type)
    # Get the model's prediction
    predictions = model.predict(padded)
    # For binary classification, return the output of the sigmoid as
two probabilities
    return np.hstack((1 - predictions, predictions))
```

```python
# Reverse the word index to make things easier to read
reverse_word_index = dict([(value, key) for (key, value) in
word_index.items()])

# Initialize the LIME text explainer
explainer = LimeTextExplainer(class_names=["Negative", "Positive"])

# Let's take a sample text from your test data
sample_text_index = 10  # change this to choose another sample
sample_text = decode_review(test_sequences[sample_text_index])
print("Sample Text:", sample_text)

# Generate an explanation
exp = explainer.explain_instance(sample_text, predict_text,
num_features=10)

# Visualize the explanation
exp.show_in_notebook(text=sample_text)
```

```
Sample Text: <OOV> friend recommend thi film realli lack origin found
charact <OOV> stereotyp plot predict almost begin howev like tradit
<OOV> <OOV> keep interest long enough hear next one ive also read
soundtrack noth like music movi profession musician fill actor

<IPython.core.display.HTML object>
```

Conclusion: The model is relatively better at identifying the positive class compared to the negative class. It also seems like the model tends to incorrectly predict the positive class more often when the actual label is negative.

Some improvements that could be applied:

-Adjust the Classification Threshold

-Class Weighting/Resampling is often a good measure to take however this is not useful in this case.

-Model Tuning- none was done so far.

-Further Feature Engineering

-Use a Different Model: This model worked well as suspected but this data is not temporal and a NN may be a better model.

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# get the model's prediction probabilities
y_pred_probs = model.predict(test_padded)

# compute the ROC curve
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)

# compute the area under the curve (AUC)
roc_auc = auc(fpr, tpr)

# plot ROC curve
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x1a8925e8ac0>
```