untitled1-1

April 13, 2024

[1]:
```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-
packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (24.0)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.62.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (3.0.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-
packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.6)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-
auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
```

[124]:
```
# Importing essential libraries and functions
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from numpy import array
import tensorflow as tf
```

[3]:
```
from keras.preprocessing.text import one_hot, Tokenizer
from keras.models import Sequential
from keras.layers import Activation, Dropout, Dense
from keras.layers import Flatten, GlobalMaxPooling1D, Embedding, Conv1D, LSTM
from sklearn.model_selection import train_test_split
```

[4]:
```
from keras.preprocessing.sequence import pad_sequences
```

[5]:
```
pip install keras-preprocessing
```

```
Collecting keras-preprocessing
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
                             42.6/42.6 kB
1.9 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.9.1 in
/usr/local/lib/python3.10/dist-packages (from keras-preprocessing) (1.25.2)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from keras-preprocessing) (1.16.0)
Installing collected packages: keras-preprocessing
Successfully installed keras-preprocessing-1.1.2
```

[9]:
```
!wget https://raw.githubusercontent.com/skillcate/
↪sentiment-analysis-with-deep-neural-networks/main/
↪b1_SentimentAnalysis_with_NeuralNetwork.ipynb
```

```
--2024-04-13 08:12:57--  https://raw.githubusercontent.com/skillcate/sentiment-
analysis-with-deep-neural-
networks/main/b1_SentimentAnalysis_with_NeuralNetwork.ipynb
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.111.133, 185.199.110.133, 185.199.109.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.111.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 215432 (210K) [text/plain]
Saving to: ‘b1_SentimentAnalysis_with_NeuralNetwork.ipynb’

b1_SentimentAnalysi 100%[===================>] 210.38K  --.-KB/s    in 0.01s

2024-04-13 08:12:57 (21.3 MB/s) -
‘b1_SentimentAnalysis_with_NeuralNetwork.ipynb’ saved [215432/215432]
```

[14]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-
packages (from pandas) (1.25.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

[15]:
```
import pandas as pd
```

[123]:
```
#Loading dataset
```

[122]:
```
# Importing IMDb Movie Reviews dataset
movie_reviews = pd.read_csv(r'/content/a1_IMDB_Dataset.csv')
```

[17]:
```

```

```
head: cannot open '/content/movie_reviews.csv' for reading: No such file or
directory
```

[121]:
```
# Dataset exploration
movie_reviews.shape
```

[121]:
```
(50000, 2)
```

[22]:
```
movie_reviews.head(5)
```

[22]:

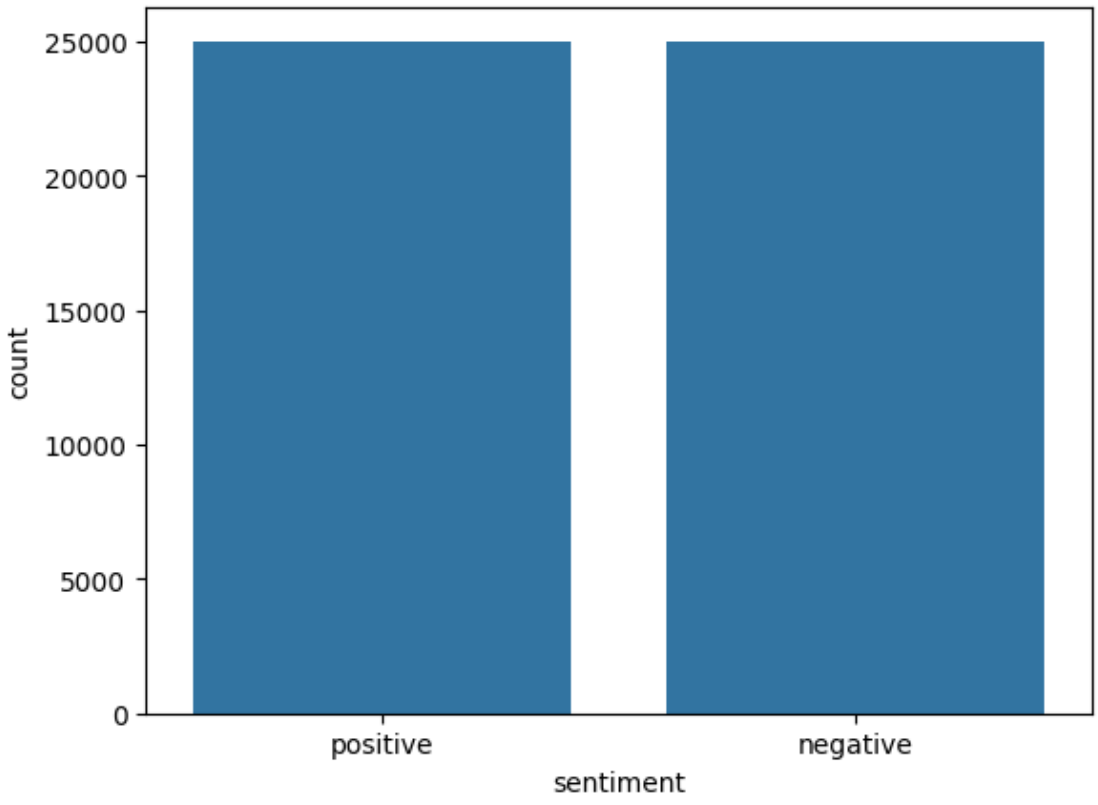
	review	sentiment
0	One of the other reviewers has mentioned that …	positive
1	A wonderful little production. The…	positive
2	I thought this was a wonderful way to spend ti…	positive
3	Basically there's a family where a little boy …	negative
4	Petter Mattei's "Love in the Time of Money" is…	positive

[120]:
```
# Checking for missing values
movie_reviews.isnull().values.any()
```

[120]:
```
False
```

[119]:
```
# Let's observe distribution of positive / negative sentiments in dataset
import seaborn as sns
sns.countplot(x='sentiment', data=movie_reviews)
```

[119]:
```
<Axes: xlabel='sentiment', ylabel='count'>
```



[25]:
```
movie_reviews["review"][2]
```

[25]:
```
'I thought this was a wonderful way to spend time on a too hot summer weekend,
sitting in the air conditioned theater and watching a light-hearted comedy. The
plot is simplistic, but the dialogue is witty and the characters are likable
(even the well bread suspected serial killer). While some may be disappointed
when they realize this is not Match Point 2: Risk Addiction, I thought it was
proof that Woody Allen is still fully in control of the style many of us have
grown to love.<br /><br />This was the most I\'d laughed at one of Woody\'s
comedies in years (dare I say a decade?). While I\'ve never been impressed with
Scarlet Johanson, in this she managed to tone down her "sexy" image and jumped
right into a average, but spirited young woman.<br /><br />This may not be the
crown jewel of his career, but it was wittier than "Devil Wears Prada" and more
interesting than "Superman" a great comedy to go see with friends.'
```

[26]:
```
TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    '''Removes HTML tags: replaces anything between opening and closing <> with␣
 ↪empty space'''

    return TAG_RE.sub('', text)
```

[27]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[27]:
```
True
```

[28]:
```
def preprocess_text(sen):
    '''Cleans text data up, leaving only 2 or more char long non-stepwords␣
 ↪composed of A-Z & a-z only
    in lowercase'''

    sentence = sen.lower()

    # Remove html tags
    sentence = remove_tags(sentence)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)

    # Single character removal
```

```
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)  # When we remove␣
↪apostrophe from the word "Mark's", the apostrophe is replaced by an empty␣
↪space. Hence, we are left with single character "s" that we are removing␣
↪here.

    # Remove multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)  # Next, we remove all the single␣
↪characters and replace it by a space which creates multiple spaces in our␣
↪text. Finally, we remove the multiple spaces from our text as well.

    # Remove Stopwords
    pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) +␣
↪r')\b\s*')
    sentence = pattern.sub('', sentence)

    return sentence
```

[118]:
```
# Calling preprocessing_text function on movie_reviews
X = []
sentences = list(movie_reviews['review'])
for sen in sentences:
    X.append(preprocess_text(sen))
```

[117]:
```
# Sample cleaned up movie review
X[2]
```

[117]:
```
'thought wonderful way spend time hot summer weekend sitting air conditioned
theater watching light hearted comedy plot simplistic dialogue witty characters
likable even well bread suspected serial killer may disappointed realize match
point risk addiction thought proof woody allen still fully control style many us
grown love laughed one woody comedies years dare say decade never impressed
scarlet johanson managed tone sexy image jumped right average spirited young
woman may crown jewel career wittier devil wears prada interesting superman
great comedy go see friends '
```

[31]:
```
y = movie_reviews['sentiment']

y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

[116]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42)
# The train set will be used to train our deep learning models
# while test set will be used to evaluate how well our model performs
```

[115]:
```
#Preparing embedding layer
```

[102]:
```
from keras.preprocessing.text import Tokenizer
```

[103]:
```
print(type(X_train))
```

```
<class 'numpy.ndarray'>
```

[104]:
```
X_train = X_train.tolist()
X_test = X_test.tolist()
```

[106]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.2)
```

[107]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

[107]:
```
True
```

[113]:
```
# Filter out non-text elements from X_test
X_test = [text for text in X_test if isinstance(text, str)]

# Convert texts to sequences
X_test_sequences = word_tokenizer.texts_to_sequences(X_test)
```

[114]:
```
import pandas as pd  # Assuming you have pandas installed

# Check if X_train contains strings (modify if data loading is different)
if not all(isinstance(x, str) for x in X_train):
    # Remove integer elements from X_train (assuming they're irrelevant)
    X_train = [x for x in X_train if isinstance(x, str)]  # List comprehension
 ↪for filtering

word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X_train)

X_train = word_tokenizer.texts_to_sequences(X_train)
X_test = word_tokenizer.texts_to_sequences(X_test)
```

[98]:
```
# Adding 1 to store dimensions for words for which no pretrained word␣
 ↪embeddings exist
vocab_length = len(word_tokenizer.word_index) + 1

vocab_length
```

[98]:
```
92394
```

[97]:
```
# Padding all reviews to fixed length 100
maxlen = 100

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

[96]:
```
# Load GloVe word embeddings and create an Embeddings Dictionary
from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()
glove_file = open('/content/a2_glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

[95]:
```
# Create Embedding Matrix having 100 columns
# Containing 100-dimensional GloVe word embeddings for all words in our corpus.
embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

[40]:
```
embedding_matrix.shape
```

[40]:
```
(92394, 100)
```

[94]:
```
#Simple Neural Network
```

[93]:
```
# Neural Network architecture
snn_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix],␣
 ↪input_length=maxlen , trainable=False)
```

```
snn_model.add(embedding_layer)

snn_model.add(Flatten())
snn_model.add(Dense(1, activation='sigmoid'))
```

[92]:
```
# Model compiling
snn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

print(snn_model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 100)          9239400

 flatten (Flatten)           (None, 10000)             0

 dense (Dense)               (None, 1)                 10001

=================================================================
Total params: 9249401 (35.28 MB)
Trainable params: 10001 (39.07 KB)
Non-trainable params: 9239400 (35.25 MB)
_________________________________________________________________
None
```

[43]:
```
snn_model_history = snn_model.fit(X_train, y_train, batch_size=128, epochs=6,␣
↪verbose=1, validation_split=0.2)
```

```
Epoch 1/6
250/250 [==============================] - 3s 9ms/step - loss: 0.5563 - acc:
0.7161 - val_loss: 0.5051 - val_acc: 0.7582
Epoch 2/6
250/250 [==============================] - 1s 5ms/step - loss: 0.4481 - acc:
0.7958 - val_loss: 0.4914 - val_acc: 0.7655
Epoch 3/6
250/250 [==============================] - 1s 5ms/step - loss: 0.4118 - acc:
0.8158 - val_loss: 0.4984 - val_acc: 0.7678
Epoch 4/6
250/250 [==============================] - 2s 6ms/step - loss: 0.3918 - acc:
0.8282 - val_loss: 0.5070 - val_acc: 0.7638
Epoch 5/6
250/250 [==============================] - 2s 8ms/step - loss: 0.3748 - acc:
0.8361 - val_loss: 0.5191 - val_acc: 0.7600
Epoch 6/6
250/250 [==============================] - 3s 12ms/step - loss: 0.3628 - acc:
```

```
0.8411 - val_loss: 0.5305 - val_acc: 0.7579
```

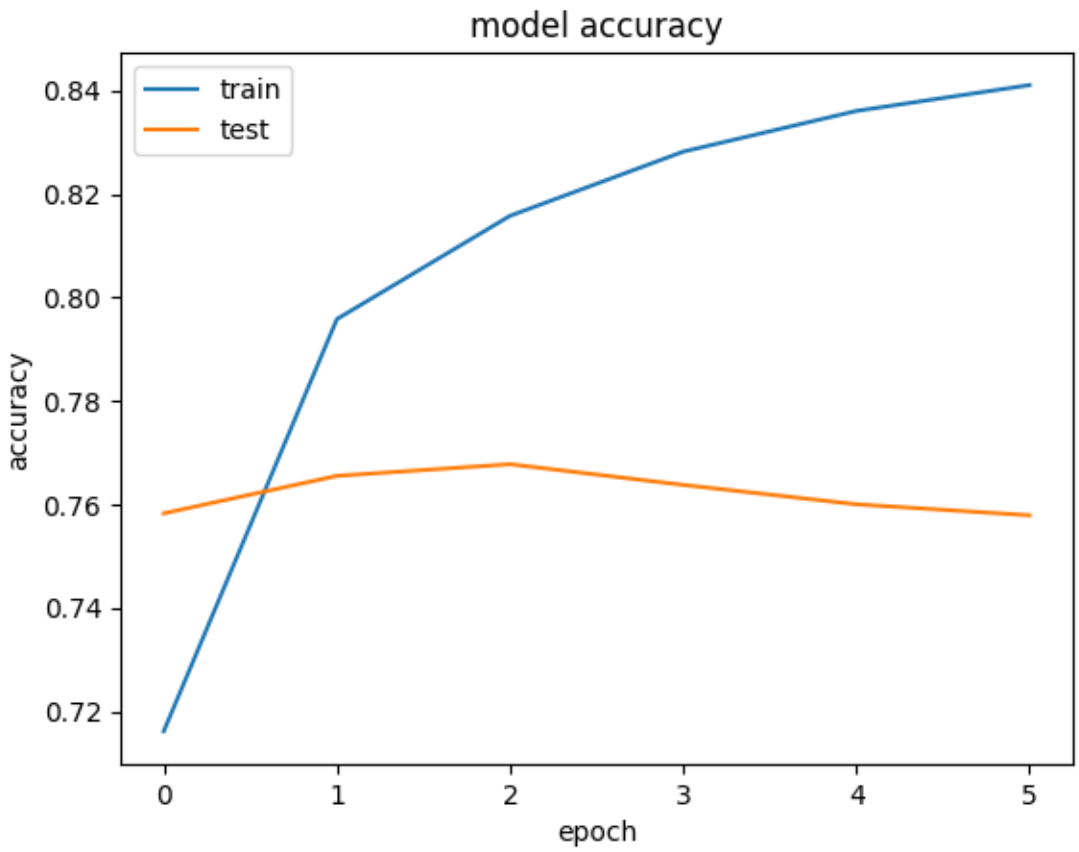
[91]:
```
# Predictions on the Test Set
score = snn_model.evaluate(X_test, y_test, verbose=1)
```

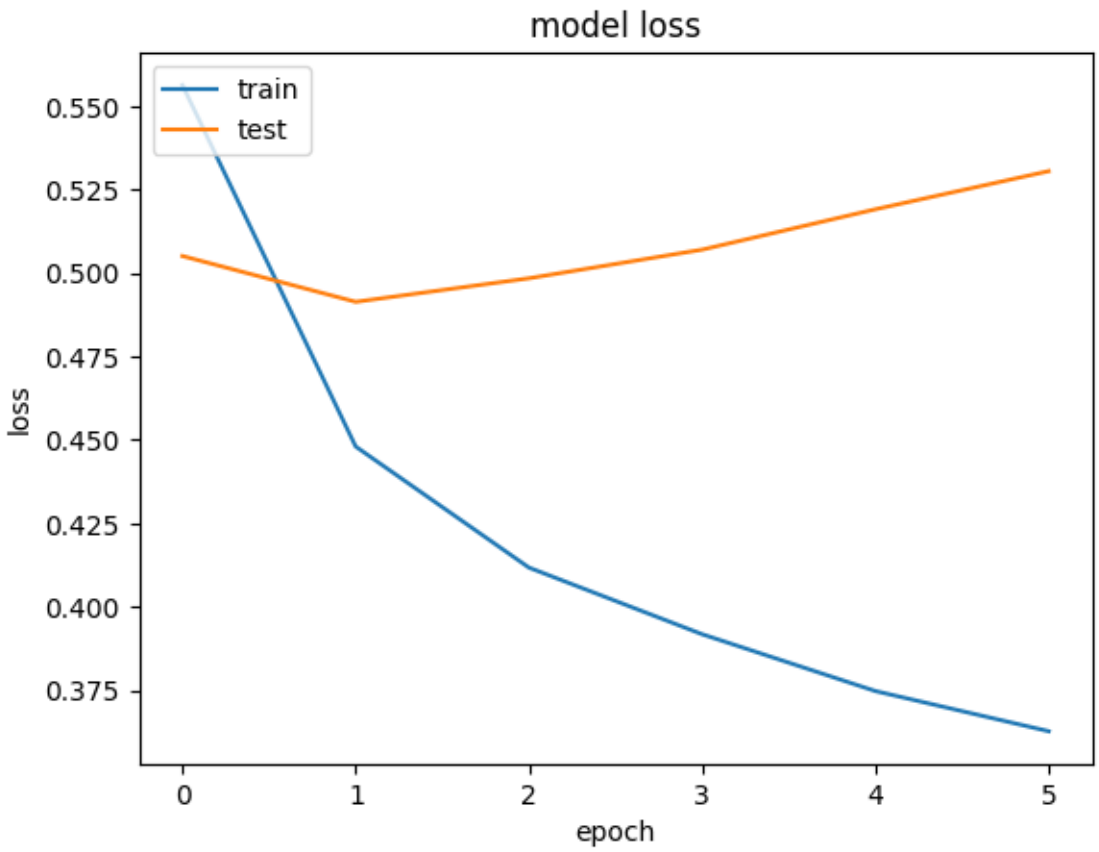
```
313/313 [==============================] - 2s 5ms/step - loss: 0.5588 - acc:
0.7501
```

[90]:
```
# Model Performance
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.33937615156173706
Test Accuracy: 0.8569999933242798
```

[89]:
```
# Model Performance Charts
import matplotlib.pyplot as plt

plt.plot(snn_model_history.history['acc'])
plt.plot(snn_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(snn_model_history.history['loss'])
plt.plot(snn_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```


model accuracy
train
test
0.84
0.82
0.80
0.78
0.76
0.74
0.72
accuracy
0
1
2
3
4
5
epoch



[88]:
```
#Convolutional Neural Network
```

[47]:
```
from keras.layers import Conv1D
```

[87]:
```
# Neural Network architecture
cnn_model = Sequential()

embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix],␣
↪input_length=maxlen , trainable=False)
cnn_model.add(embedding_layer)

cnn_model.add(Conv1D(128, 5, activation='relu'))
cnn_model.add(GlobalMaxPooling1D())
cnn_model.add(Dense(1, activation='sigmoid'))
```

[86]:
```
# Model compiling
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(cnn_model.summary())
```

```
Model: "sequential_1"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 100)          9239400

 conv1d (Conv1D)             (None, 96, 128)           64128

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 9303657 (35.49 MB)
Trainable params: 64257 (251.00 KB)
Non-trainable params: 9239400 (35.25 MB)
_________________________________________________________________
None
```

[85]:
```
# Model training
cnn_model_history = cnn_model.fit(X_train, y_train, batch_size=128, epochs=6,␣
 ↪verbose=1, validation_split=0.2)
```

```
Epoch 1/6
250/250 [==============================] - 33s 132ms/step - loss: 0.1345 - acc:
0.9656 - val_loss: 0.3443 - val_acc: 0.8541
Epoch 2/6
250/250 [==============================] - 30s 121ms/step - loss: 0.1092 - acc:
0.9770 - val_loss: 0.3487 - val_acc: 0.8528
Epoch 3/6
250/250 [==============================] - 29s 118ms/step - loss: 0.0852 - acc:
0.9880 - val_loss: 0.3539 - val_acc: 0.8533
Epoch 4/6
250/250 [==============================] - 28s 110ms/step - loss: 0.0660 - acc:
0.9936 - val_loss: 0.3725 - val_acc: 0.8512
Epoch 5/6
250/250 [==============================] - 30s 121ms/step - loss: 0.0515 - acc:
0.9967 - val_loss: 0.3719 - val_acc: 0.8512
Epoch 6/6
250/250 [==============================] - 31s 124ms/step - loss: 0.0402 - acc:
0.9987 - val_loss: 0.3890 - val_acc: 0.8503
```

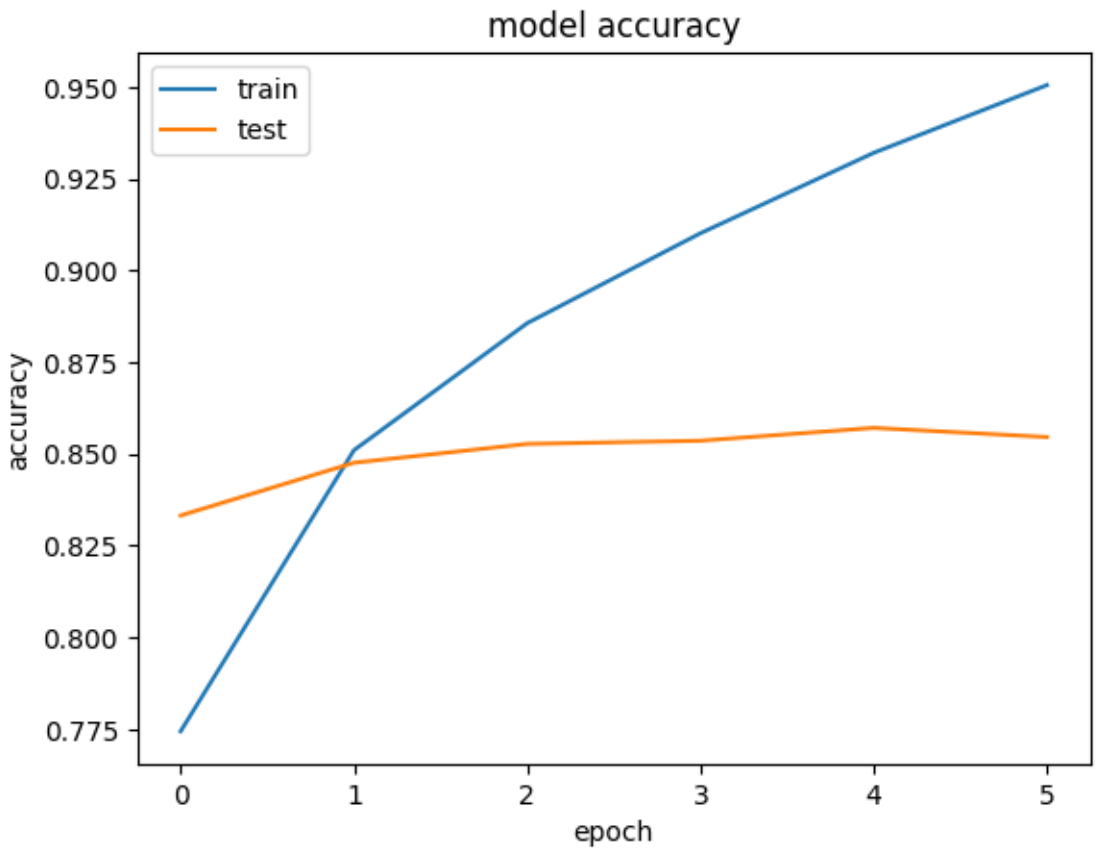
[84]:
```
# Predictions on the Test Set
score = cnn_model.evaluate(X_test, y_test, verbose=1)
```

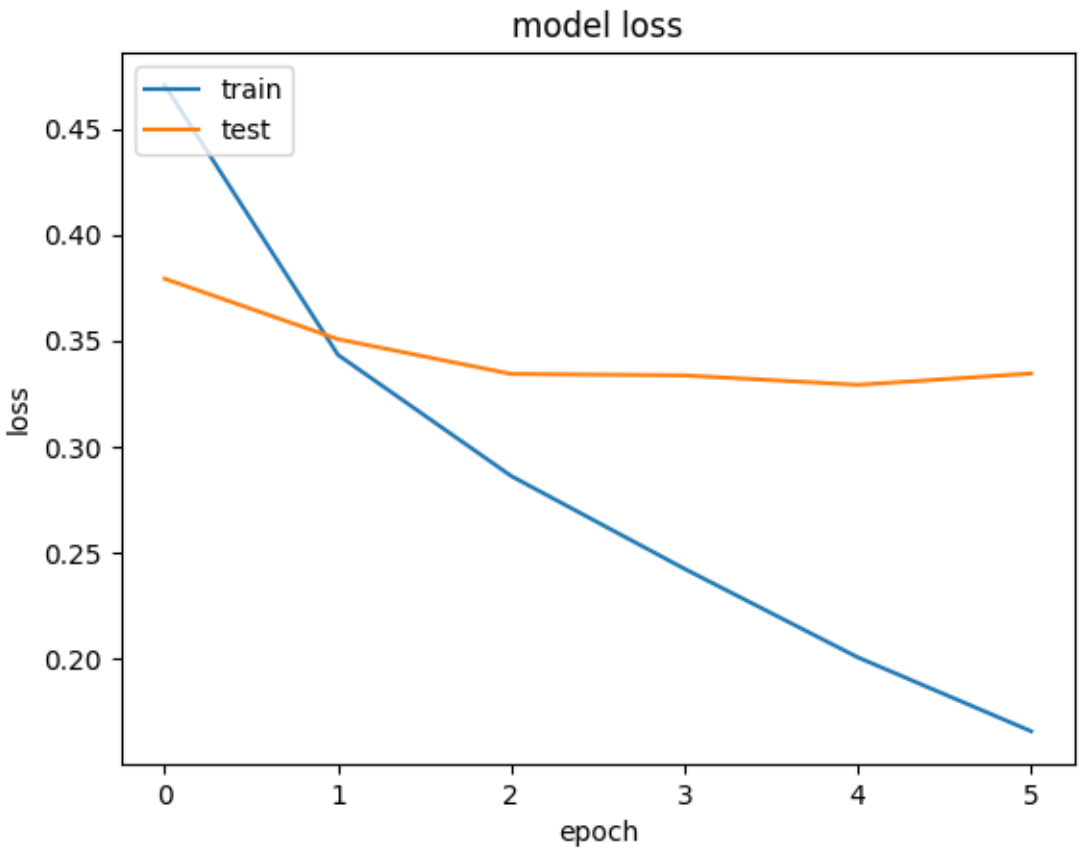
```
313/313 [==============================] - 6s 20ms/step - loss: 0.3394 - acc:
0.8570
```

[83]:
```
# Model Performance
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.31639814376831055
Test Accuracy: 0.8621000051498413
```

[82]:
```
# Model Performance Charts
import matplotlib.pyplot as plt

plt.plot(cnn_model_history.history['acc'])
plt.plot(cnn_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(cnn_model_history.history['loss'])
plt.plot(cnn_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```


model accuracy
train
test
accuracy
epoch
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
1
2
3
4
5



[81]:
```
#Recurrent Neural Network (LSTM)
```

[54]:
```
from keras.layers import LSTM
```

[80]:
```
# Neural Network architecture
lstm_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix],␣
 ↪input_length=maxlen , trainable=False)

lstm_model.add(embedding_layer)
lstm_model.add(LSTM(128))

lstm_model.add(Dense(1, activation='sigmoid'))
```

[79]:
```
# Model compiling
lstm_model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['acc'])
print(lstm_model.summary())
```

```
Model: "sequential_2"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 100)          9239400

 lstm (LSTM)                 (None, 128)               117248

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 9356777 (35.69 MB)
Trainable params: 117377 (458.50 KB)
Non-trainable params: 9239400 (35.25 MB)

_________________________________________________________________
None
```

[78]:
```
# Model Training
lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=6,␣
 ↪verbose=1, validation_split=0.2)
```

```
Epoch 1/6
250/250 [==============================] - 100s 400ms/step - loss: 0.2792 - acc:
0.8863 - val_loss: 0.3212 - val_acc: 0.8597
Epoch 2/6
250/250 [==============================] - 95s 379ms/step - loss: 0.2606 - acc:
0.8957 - val_loss: 0.3381 - val_acc: 0.8689
Epoch 3/6
250/250 [==============================] - 92s 367ms/step - loss: 0.2428 - acc:
0.9043 - val_loss: 0.3219 - val_acc: 0.8609
Epoch 4/6
250/250 [==============================] - 94s 377ms/step - loss: 0.2219 - acc:
0.9147 - val_loss: 0.3500 - val_acc: 0.8650
Epoch 5/6
250/250 [==============================] - 92s 366ms/step - loss: 0.2028 - acc:
0.9228 - val_loss: 0.3318 - val_acc: 0.8674
Epoch 6/6
250/250 [==============================] - 90s 360ms/step - loss: 0.1824 - acc:
0.9326 - val_loss: 0.3455 - val_acc: 0.8676
```

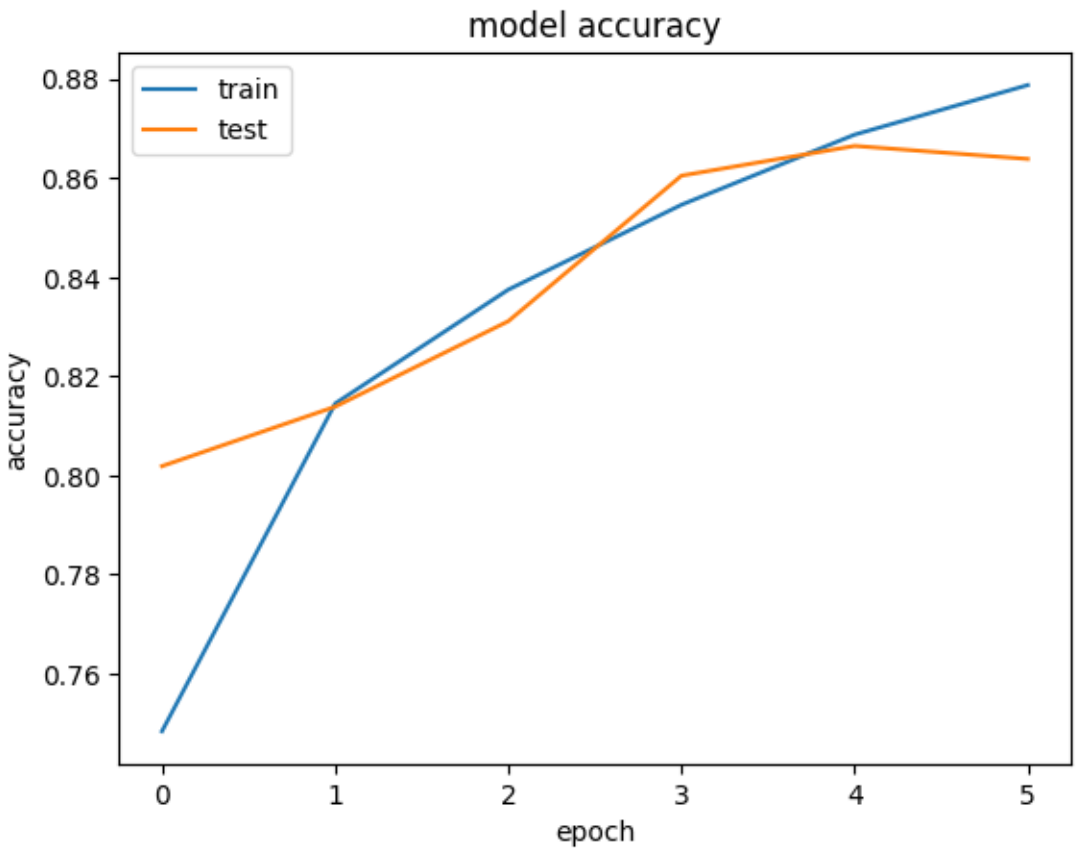
[77]:
```
# Predictions on the Test Set
score = lstm_model.evaluate(X_test, y_test, verbose= 1)
```

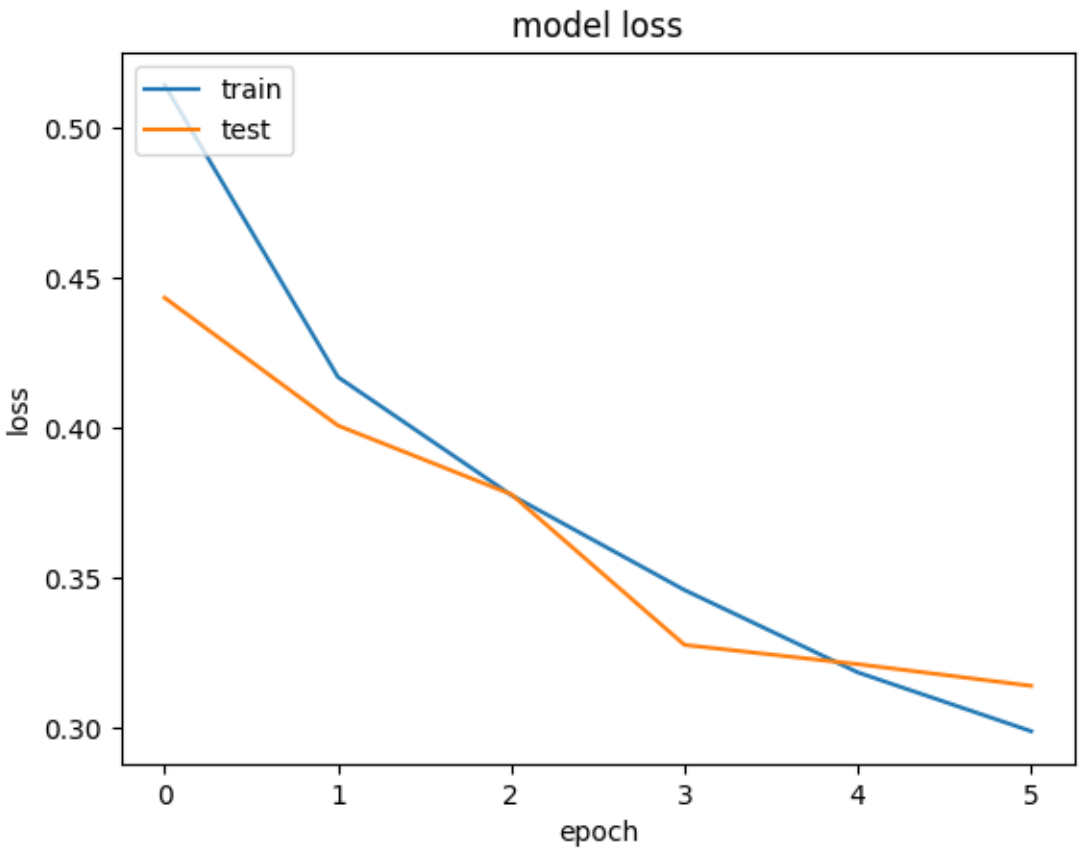
```
313/313 [==============================] - 22s 69ms/step - loss: 0.3164 - acc:
0.8621
```

[76]:
```
# Model Performance
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.31639814376831055
Test Accuracy: 0.8621000051498413
```

[75]:
```python
# Model Performance Charts
import matplotlib.pyplot as plt

plt.plot(lstm_model_history.history['acc'])
plt.plot(lstm_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(lstm_model_history.history['loss'])
plt.plot(lstm_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```


model accuracy
train
test
accuracy
epoch
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0
1
2
3
4
5



[74]:
```
# Saving the model as a h5 file for possible use later
lstm_model.save(f"./c1_lstm_model_acc_{round(score[1], 3)}.h5",␣
 ↪save_format='h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

[73]:
```
#Making Predictions on Live IMDb data
# # Load previously trained LSTM Model

# from keras.models import load_model

# model_path ='./c1_lstm_model_acc_0.856.h5'
# pretrained_lstm_model = load_model(model_path)

# # summarize model.
# pretrained_lstm_model.summary()
```

[62]:
```
sample_reviews = pd.read_csv("/content/a3_IMDb_Unseen_Reviews.csv")
```

[72]:
```
# Load sample IMDb reviews csv, having ~6 movie reviews, along with their IMDb␣
↪rating
sample_reviews.head(6)
```

[72]:

	Unnamed: 0	Movie	Review Text	IMDb Rating	Predicted Sentiments
0	0	Ex Machina	Intelligent Movie.\nThis movie is obviously al…	9	8.9
1	1	Ex Machina	Extraordinary and thought-provoking.\n'Ex mach…	10	9.9
2	2	Ex Machina	Poor story, only reasonable otherwise.\nIf I h…	3	2.8
3	3	Ex Machina	Had Great Potential.\nThis movie is one of the…	1	4.6
4	4	Eternals	Amazing visuals and philosophical concepts!\n\…	10	9.7
5	5	Eternals	Worst MCU film ever\n\nFollowing the events of…	3	0.2

[71]:
```
# Preprocess review text with earlier defined preprocess_text function
unseen_reviews = sample_reviews['Review Text']

unseen_processed = []
for review in unseen_reviews:
  review = preprocess_text(review)
  unseen_processed.append(review)
```

[70]:
```
# Tokenising instance with earlier trained tokeniser
unseen_tokenized = word_tokenizer.texts_to_sequences(unseen_processed)

# Pooling instance to have maxlength of 100 tokens
unseen_padded = pad_sequences(unseen_tokenized, padding='post', maxlen=maxlen)
```

[69]:
```
# Passing tokenised instance to the LSTM model for predictions
unseen_sentiments = lstm_model.predict(unseen_padded)

unseen_sentiments
```

```
1/1 [==============================] - 0s 32ms/step
```

[69]:
```
array([[0.8879797 ],
       [0.9914043 ],
       [0.27684546],
       [0.46238056],
```

```
       [0.97226   ],
       [0.01589395]], dtype=float32)
```

[68]:
```
# Writing model output file back to Google Drive
sample_reviews['Predicted Sentiments'] = np.round(unseen_sentiments*10,1)

df_prediction_sentiments = pd.DataFrame(sample_reviews['Predicted Sentiments'],␣
 ↪columns = ['Predicted Sentiments'])
df_movie                 = pd.DataFrame(sample_reviews['Movie'], columns =␣
 ↪['Movie'])
df_review_text           = pd.DataFrame(sample_reviews['Review Text'], columns␣
 ↪= ['Review Text'])
df_imdb_rating           = pd.DataFrame(sample_reviews['IMDb Rating'], columns␣
 ↪= ['IMDb Rating'])


dfx=pd.concat([df_movie, df_review_text, df_imdb_rating,␣
 ↪df_prediction_sentiments], axis=1)

dfx.to_csv("./c2_IMDb_Unseen_Predictions.csv", sep=',', encoding='UTF-8')

dfx.head(6)
```

[68]:

	Movie	Review Text	IMDb Rating	Predicted Sentiments
0	Ex Machina	Intelligent Movie.\nThis movie is obviously al…	9	8.9
1	Ex Machina	Extraordinary and thought-provoking.\n'Ex mach…	10	9.9
2	Ex Machina	Poor story, only reasonable otherwise.\nIf I h…	3	2.8
3	Ex Machina	Had Great Potential.\nThis movie is one of the…	1	4.6
4	Eternals	Amazing visuals and philosophical concepts!\n\…	10	9.7
5	Eternals	Worst MCU film ever\n\nFollowing the events of…	3	0.2

[ ]: