CODEXCUE

DEEP LEARNING

Simple Sentiment Analysis:

```
# 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

from keras.preprocessing.text import one_hot, Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, GlobalMaxPooling1D, Embedding, Conv1D, LSTM
from sklearn.model_selection import train_test_split
```

Loading dataset

```
# Importing IMDb Movie Reviews dataset

movie_reviews = pd.read_csv("a1_IMDB_Dataset.csv")

# dataset source: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
```

```
# Dataset exploration

movie_reviews.shape
```

```
(50000, 2)
```

```
movie_reviews.head(5)
```

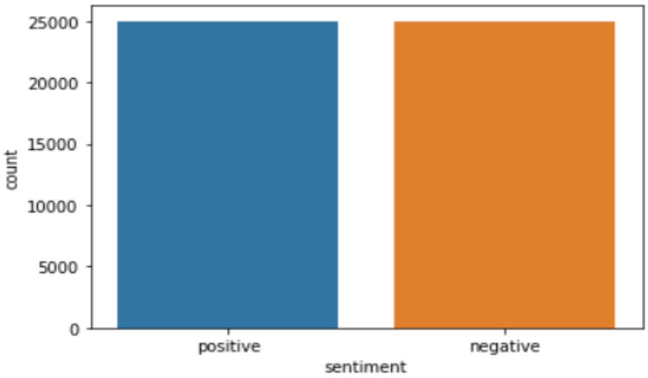
	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...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

```
# Checking for missing values

movie_reviews.isnull().values.any()
```

```
False
```

```
# Let's observe distribution of positive / negative sentiments in dataset

import seaborn as sns
sns.countplot(x='sentiment', data=movie_reviews)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe491790490>
```




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

def remove_tags(text):

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

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

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

```
# Calling preprocessing_text function on movie_reviews

X = []
sentences = list(movie_reviews['review'])
for sen in sentences:
    X.append(preprocess_text(sen))
```

```
# Sample cleaned up movie review

X[2]

# Since we are using Word Embeddings, we do not perform stemming or lemmatization as part of the preprocessing steps.
```

```
'thought wonderful way spend time hot summer weekend sitting air conditioned theater watching light hearted comedy plot simplistic dialogue witt
y 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 '
```

```
# Converting sentiment labels to 0 & 1

y = movie_reviews['sentiment']

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

```
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
```

Preparing the embedding layer

```
[ ]
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)
```

```
[ ] # Adding 1 to store dimensions for words for which no pretrained word embeddings exist

vocab_length = len(word_tokenizer.word_index) + 1

vocab_length
```

```
92394
```

```
[ ] # 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)
```

```
[ ] # Load GloVe word embeddings and create an Embeddings Dictionary

from numpy import asarray
from numpy import zeros
```

```
embeddings_dictionary = dict()
glove_file = open('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()
```

```
[ ]
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
```

```
[ ] embedding_matrix.shape
```

```
(92394, 100)
```

Model Training with:

Simple Neural Network

```
[ ] # 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'))
```

```
[ ] # 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: 9,249,401
Trainable params: 10,001
Non-trainable params: 9,239,400
_________________________________________________________________
None
```

```
[ ] # Model training

    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 [==============================] - 2s 6ms/step - loss: 0.5589 - acc: 0.7134 - val_loss: 0.5088 - val_acc: 0.7565
Epoch 2/6
250/250 [==============================] - 1s 6ms/step - loss: 0.4514 - acc: 0.7925 - val_loss: 0.5067 - val_acc: 0.7580
Epoch 3/6
250/250 [==============================] - 1s 5ms/step - loss: 0.4159 - acc: 0.8130 - val_loss: 0.5002 - val_acc: 0.7646
Epoch 4/6
250/250 [==============================] - 1s 5ms/step - loss: 0.3918 - acc: 0.8273 - val_loss: 0.5084 - val_acc: 0.7615
Epoch 5/6
250/250 [==============================] - 1s 6ms/step - loss: 0.3753 - acc: 0.8370 - val_loss: 0.5188 - val_acc: 0.7584
Epoch 6/6
250/250 [==============================] - 1s 6ms/step - loss: 0.3624 - acc: 0.8435 - val_loss: 0.5325 - val_acc: 0.7558
```

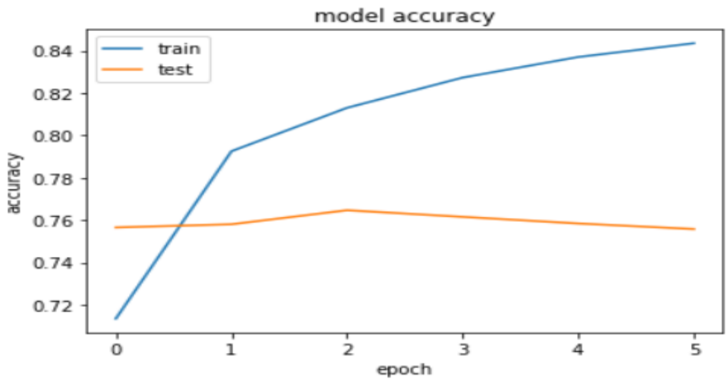
```
[ ] # Predictions on the Test Set

    score = snn_model.evaluate(X_test, y_test, verbose=1)
```

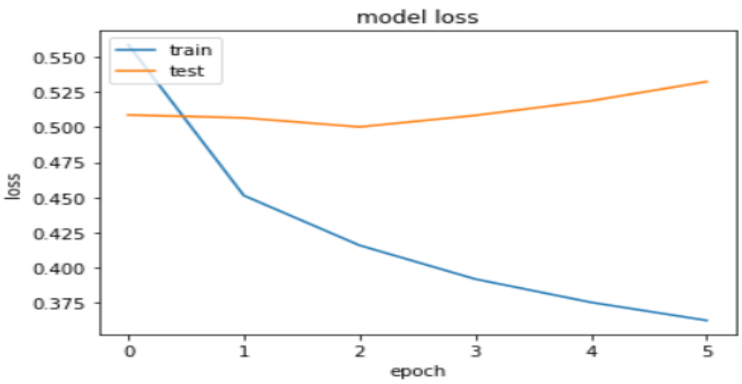
```
313/313 [==============================] - 1s 2ms/step - loss: 0.5584 - acc: 0.7499
```

```
# Model Performance

print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.5584211945533752
Test Accuracy: 0.7498999834060669
```

```
# 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()
```





Recurrent Neural Network

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

```
[ ] # 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'))
```

```
[ ] # 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: 9,356,777
    Trainable params: 117,377
    Non-trainable params: 9,239,400
    _________________________________________________________________
    None
```

```
[ ] # 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 [==============================] - 82s 320ms/step - loss: 0.5498 - acc: 0.7266 - val_loss: 0.4619 - val_acc: 0.7909
Epoch 2/6
250/250 [==============================] - 79s 317ms/step - loss: 0.4309 - acc: 0.8056 - val_loss: 0.4138 - val_acc: 0.8207
Epoch 3/6
250/250 [==============================] - 80s 319ms/step - loss: 0.3883 - acc: 0.8302 - val_loss: 0.3593 - val_acc: 0.8457
Epoch 4/6
250/250 [==============================] - 80s 318ms/step - loss: 0.3506 - acc: 0.8489 - val_loss: 0.3402 - val_acc: 0.8564
Epoch 5/6
250/250 [==============================] - 79s 317ms/step - loss: 0.3259 - acc: 0.8621 - val_loss: 0.3239 - val_acc: 0.8585
Epoch 6/6
250/250 [==============================] - 79s 317ms/step - loss: 0.3067 - acc: 0.8712 - val_loss: 0.3148 - val_acc: 0.8670
```

```
# Predictions on the Test Set

score = lstm_model.evaluate(X_test, y_test, verbose=1)
```

```
313/313 [==============================] - 11s 34ms/step - loss: 0.3194 - acc: 0.8643
```

```
# Model Performance

print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.31936636567115784
Test Accuracy: 0.864300012588501
```

```
# 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()
```