Assignment 6 - Exercise 1

Daniel Mehta

In [1]:
```
#imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import re
import numpy as np
from collections import Counter
from nltk.corpus import stopwords
from tqdm import tqdm

import torch
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
```

In [2]:
```
# download stopwords
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\danie\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]: True

In [3]:
```
# load dataset
file_name = 'IMDB Dataset.csv'
df = pd.read_csv(file_name)
df.head()
```

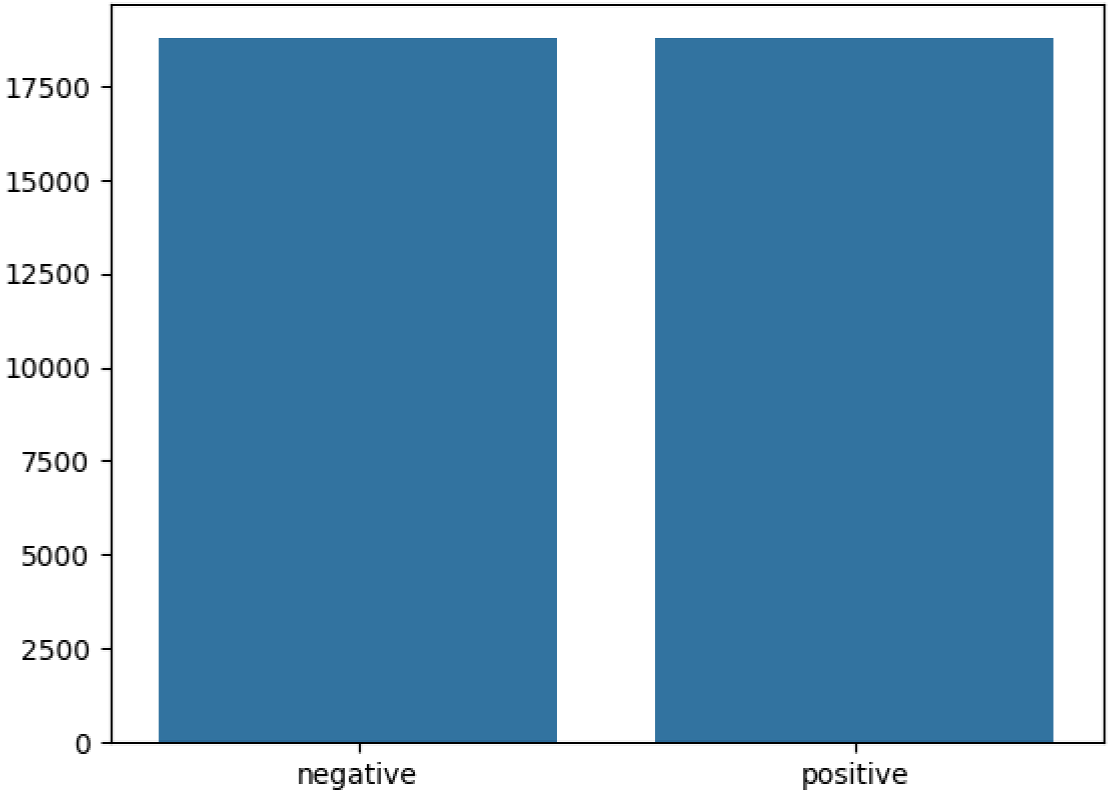
Out[3]:

	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

In [4]:

```
#split data
X,y = df['review'].values,df['sentiment'].values
x_train,x_test,y_train,y_test = train_test_split(X,y,stratify=y)
print(f'train data shape: {x_train.shape}')
print(f'test data shape: {x_test.shape}')

# Plot sentiment
dd = pd.Series(y_train).value_counts()
sns.barplot(x=np.array(['negative','positive']),y=dd.values)
plt.show()
```

```
train data shape: (37500,)
test data shape: (12500,)
```

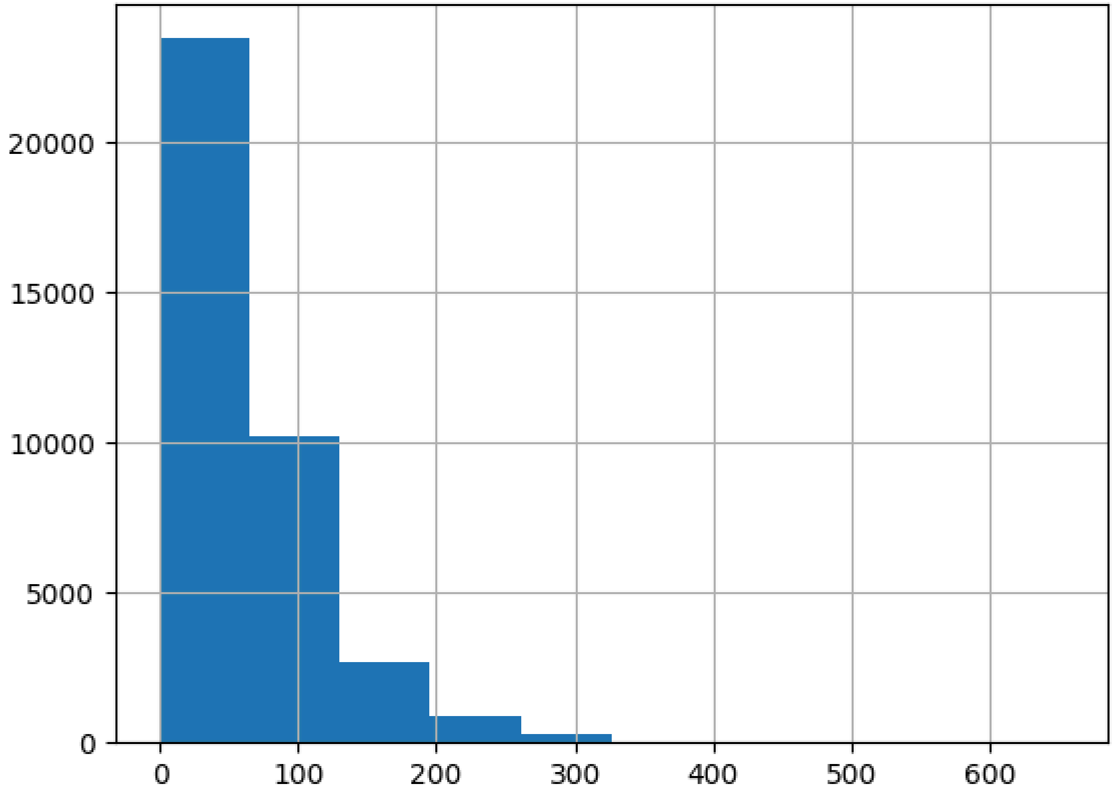


In [5]:
```
# Text preprocessing

def preprocess_string(s):
    # Remove all non-word characters (everything except numbers and letters)
    s = re.sub(r"[^\w\s]", '', s)
    # Replace all runs of whitespaces with no space
    s = re.sub(r"\s+", '', s)
    # replace digits with no space
    s = re.sub(r"\d", '', s)
    return s


def tokenize(x_train,y_train,x_val,y_val):
    word_list = []
```

```
    stop_words = set(stopwords.words('english'))
    for sent in x_train:
        for word in sent.lower().split():
            word = preprocess_string(word)
            if word not in stop_words and word != '':
                word_list.append(word)

    corpus = Counter(word_list)
    # sorting on the basis of most common words
    corpus_ = sorted(corpus,key=corpus.get,reverse=True)[:1000]
    # creating a dict
    onehot_dict = {w:i+1 for i,w in enumerate(corpus_)}

    # tokenize
    final_list_train,final_list_test = [],[]
    for sent in x_train:
            final_list_train.append([onehot_dict[preprocess_string(word)] for word in sent.lower().split()
                                     if preprocess_string(word) in onehot_dict.keys()])
    for sent in x_val:
            final_list_test.append([onehot_dict[preprocess_string(word)] for word in sent.lower().split()
                                    if preprocess_string(word) in onehot_dict.keys()])

    encoded_train = [1 if label =='positive' else 0 for label in y_train]
    encoded_test = [1 if label =='positive' else 0 for label in y_val]

    # Converted to Numpy arrays with dtype=object to handle variable length sequences without padding
    return np.array(final_list_train, dtype=object), np.array(encoded_train), np.array(final_list_test, dtype=object), np.arra


x_train,y_train,x_test,y_test,vocab = tokenize(x_train,y_train,x_test,y_test)
```

In [6]:
```
rev_len = [len(i) for i in x_train]
pd.Series(rev_len).hist()
```

Out[6]: <Axes: >



In [7]:
```
def padding_(sentences, seq_len):
    # Creating a 2D numpy array of zeros with shape
    features = np.zeros((len(sentences), seq_len),dtype=int)

    # Loop through each sentence
    for ii, review in enumerate(sentences):
        if len(review) != 0:
            # if the sentence is shorter than seq_len, fill from the end
            # if the sentence is longer than seq_len, cut it short
            features[ii, -len(review):] = np.array(review)[:seq_len]
    return features


# Pad training and test data to a fixed length of 500
```

```
x_train_pad = padding_(x_train,500)
x_test_pad = padding_(x_test,500)
```

In [8]:
```
# create Tensor datasets
train_data = TensorDataset(torch.from_numpy(x_train_pad), torch.from_numpy(y_train))
valid_data = TensorDataset(torch.from_numpy(x_test_pad), torch.from_numpy(y_test))


# dataloaders
batch_size = 50


# make sure to SHUFFLE your data
train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)


# obtain one batch of training data
dataiter = iter(train_loader)
sample_x, sample_y = next(dataiter)


print('Sample input size: ', sample_x.size()) # batch_size, seq_length
print('Sample input: \n', sample_x)
print('Sample output: \n', sample_y)
```

```
Sample input size:  torch.Size([50, 500])
Sample input: 
 tensor([[  0,   0,   0,  ..., 597,  32,   2],
        [  0,   0,   0,  ...,  14, 165,  30],
        [  0,   0,   0,  ..., 142, 662,   2],
        ...,
        [  0,   0,   0,  ..., 414, 304, 612],
        [  0,   0,   0,  ..., 710, 475, 302],
        [  0,   0,   0,  ..., 311,  91,  42]])
Sample output: 
 tensor([1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
        1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1,
        1, 1])
```

LSTM Model

In [9]:
```
class SentimentRNN(nn.Module):
    def __init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5):
        super(SentimentRNN,self).__init__()

        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.no_layers = no_layers
        self.vocab_size = vocab_size

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        #lstm
        self.lstm = nn.LSTM(input_size=embedding_dim,hidden_size=self.hidden_dim,
                           num_layers=no_layers, batch_first=True)

        # dropout layer
        self.dropout = nn.Dropout(0.3)

        # linear and sigmoid layer
        self.fc = nn.Linear(self.hidden_dim, output_dim)
        self.sig = nn.Sigmoid()

    def forward(self,x,hidden):
        batch_size = x.size(0)
        # embeddings and lstm_out
        embeds = self.embedding(x)  # shape: B x S x Feature   since batch = True
        #print(embeds.shape)  #[50, 500, 1000]
        lstm_out, hidden = self.lstm(embeds, hidden)

        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)
```

```
        # sigmoid function
        sig_out = self.sig(out)

        # reshape to be batch_size first
        sig_out = sig_out.view(batch_size, -1)


        sig_out = sig_out[:, -1] # get last batch of labels

        # return last sigmoid output and hidden state
        return sig_out, hidden

    def init_hidden(self, batch_size):
        ''' Initializes hidden state '''
        # Create two new tensors with sizes n_layers x batch_size x hidden_dim,
        # initialized to zero, for hidden state and cell state of LSTM
        h0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
        c0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
        hidden = (h0,c0)
        return hidden
```

In [10]:

```
no_layers = 2
vocab_size = len(vocab) + 1 #extra 1 for padding
embedding_dim = 64
output_dim = 1
hidden_dim = 256

model = SentimentRNN(no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5)

#moving to cpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print(model)
print(device)
```

```
SentimentRNN(
  (embedding): Embedding(1001, 64)
  (lstm): LSTM(64, 256, num_layers=2, batch_first=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
cuda
```

In [11]:
```
# loss and optimization functions

lr=0.001

criterion = nn.BCELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# function to predict accuracy
def acc(pred,label):
    pred = torch.round(pred.squeeze())
    return torch.sum(pred == label.squeeze()).item()
```

Training

In [12]:
```
clip = 5
epochs = 5
valid_loss_min = np.inf

epoch_tr_loss, epoch_vl_loss = [], []
epoch_tr_acc, epoch_vl_acc = [], []

for epoch in range(epochs):
    train_losses = []
    train_acc = 0.0
    model.train()
    h = model.init_hidden(batch_size)

    print(f'\nEpoch {epoch+1}/{epochs}')
    print('Training...')
```

```
    for inputs, labels in tqdm(train_loader):
        inputs, labels = inputs.to(device), labels.to(device)
        h = tuple([each.data for each in h])

        model.zero_grad()
        output, h = model(inputs, h)

        loss = criterion(output.squeeze(), labels.float())
        loss.backward()
        train_losses.append(loss.item())

        accuracy = acc(output, labels)
        train_acc += accuracy

        nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

    val_h = model.init_hidden(batch_size)
    val_losses = []
    val_acc = 0.0
    model.eval()

    print('Validating...')
    for inputs, labels in tqdm(valid_loader):
        val_h = tuple([each.data for each in val_h])
        inputs, labels = inputs.to(device), labels.to(device)

        with torch.no_grad():
            output, val_h = model(inputs, val_h)
            val_loss = criterion(output.squeeze(), labels.float())
            val_losses.append(val_loss.item())
            accuracy = acc(output, labels)
            val_acc += accuracy

    epoch_train_loss = np.mean(train_losses)
    epoch_val_loss = np.mean(val_losses)
    epoch_train_acc = train_acc / len(train_loader.dataset)
    epoch_val_acc = val_acc / len(valid_loader.dataset)

    epoch_tr_loss.append(epoch_train_loss)
    epoch_vl_loss.append(epoch_val_loss)
```

```
    epoch_tr_acc.append(epoch_train_acc)
    epoch_vl_acc.append(epoch_val_acc)

    print(f'\ntrain_loss: {epoch_train_loss:.4f}  val_loss: {epoch_val_loss:.4f}')
    print(f'train_accuracy: {epoch_train_acc*100:.2f}%  val_accuracy: {epoch_val_acc*100:.2f}%')

    if epoch_val_loss <= valid_loss_min:
        torch.save(model.state_dict(), 'state_dict.pt')
        print(f'Validation loss decreased ({valid_loss_min:.6f} --> {epoch_val_loss:.6f}). Saving model...')
        valid_loss_min = epoch_val_loss

    print('==' * 25)
```

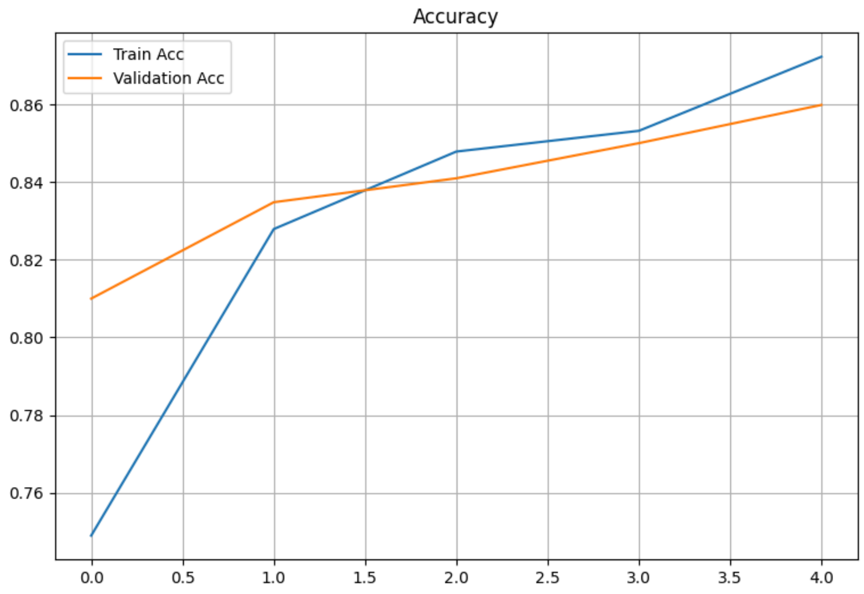
```
Epoch 1/5
Training...
100%|██████████| 750/750 [00:31<00:00, 23.93it/s]
Validating...
100%|██████████| 250/250 [00:04<00:00, 58.82it/s]
train_loss: 0.5134  val_loss: 0.4223
train_accuracy: 74.90%  val_accuracy: 81.00%
Validation loss decreased (inf --> 0.422256). Saving model...
==================================================

Epoch 2/5
Training...
100%|██████████| 750/750 [00:30<00:00, 24.30it/s]
Validating...
100%|██████████| 250/250 [00:04<00:00, 58.95it/s]
train_loss: 0.3961  val_loss: 0.3862
train_accuracy: 82.79%  val_accuracy: 83.48%
Validation loss decreased (0.422256 --> 0.386165). Saving model...
==================================================

Epoch 3/5
Training...
100%|██████████| 750/750 [00:31<00:00, 23.98it/s]
Validating...
100%|██████████| 250/250 [00:04<00:00, 58.77it/s]
```

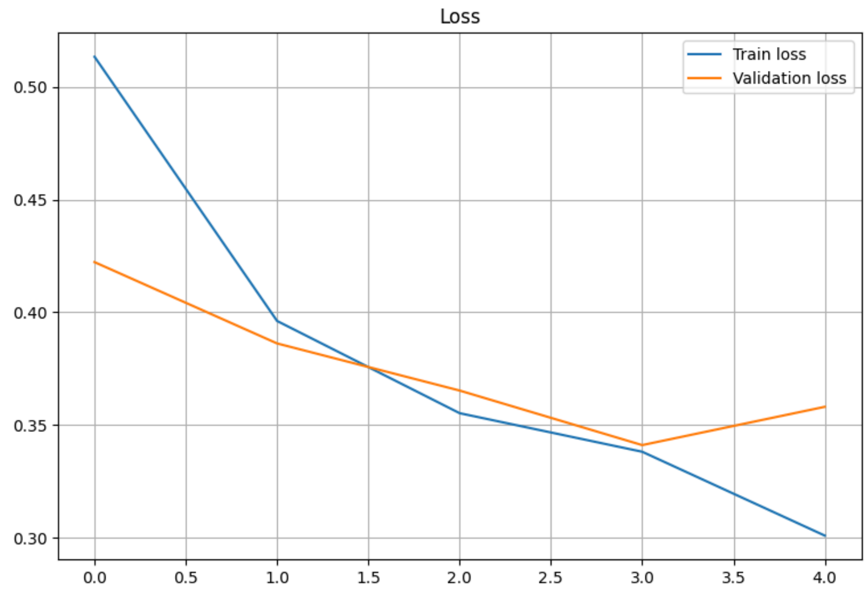
```
train_loss: 0.3552  val_loss: 0.3653
train_accuracy: 84.78%  val_accuracy: 84.10%
Validation loss decreased (0.386165 --> 0.365253). Saving model...
==================================================

Epoch 4/5
Training...
100%|██████████| 750/750 [00:31<00:00, 24.07it/s]
Validating...
100%|██████████| 250/250 [00:04<00:00, 58.37it/s]
train_loss: 0.3381  val_loss: 0.3410
train_accuracy: 85.32%  val_accuracy: 85.00%
Validation loss decreased (0.365253 --> 0.341039). Saving model...
==================================================

Epoch 5/5
Training...
100%|██████████| 750/750 [00:30<00:00, 24.24it/s]
Validating...
100%|██████████| 250/250 [00:04<00:00, 59.02it/s]
train_loss: 0.3009  val_loss: 0.3580
train_accuracy: 87.22%  val_accuracy: 85.98%
==================================================
```

In [13]:

```
fig = plt.figure(figsize = (20, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_tr_acc, label='Train Acc')
plt.plot(epoch_vl_acc, label='Validation Acc')
plt.title("Accuracy")
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(epoch_tr_loss, label='Train loss')
plt.plot(epoch_vl_loss, label='Validation loss')
plt.title("Loss")
plt.legend()
plt.grid()
```

```
plt.show()
```





In [14]:
```
def predict_text(text):
        word_seq = np.array([vocab[preprocess_string(word)] for word in text.split()
                         if preprocess_string(word) in vocab.keys()])
        word_seq = np.expand_dims(word_seq,axis=0)
        pad =  torch.from_numpy(padding_(word_seq,500))
        inputs = pad.to(device)
        batch_size = 1
        h = model.init_hidden(batch_size)
        h = tuple([each.data for each in h])
        output, h = model(inputs, h)
        return(output.item())


index = 30
print(df['review'][index])
print('='*70)
print(f'Actual sentiment is  : {df["sentiment"][index]}')
print('='*70)
pro = predict_text(df['review'][index])
```

```python
status = "positive" if pro > 0.5 else "negative"
pro = (1 - pro) if status == "negative" else pro
print(f'Predicted sentiment is {status} with a probability of {pro}')
```

```
Taut and organically gripping, Edward Dmytryk's Crossfire is a distinctive suspense thriller, an unlikely "message" movie using
the look and devices of the noir cycle.<br /><br />Bivouacked in Washington, DC, a company of soldiers cope with their restless
ness by hanging out in bars. Three of them end up at a stranger's apartment where Robert Ryan, drunk and belligerent, beats the
ir host (Sam Levene) to death because he happens to be Jewish. Police detective Robert Young investigates with the help of Robe
rt Mitchum, who's assigned to Ryan's outfit. Suspicion falls on the second of the three (George Cooper), who has vanished. Ryan
slays the third buddy (Steve Brodie) to insure his silence before Young closes in.<br /><br />Abetted by a superior script by J
ohn Paxton, Dmytryk draws precise performances from his three starring Bobs. Ryan, naturally, does his prototypical Angry White
Male (and to the hilt), while Mitchum underplays with his characteristic alert nonchalance (his role, however, is not central);
Young may never have been better. Gloria Grahame gives her first fully-fledged rendition of the smart-mouthed, vulnerable tram
p, and, as a sad sack who's leeched into her life, Paul Kelly haunts us in a small, peripheral role that he makes memorable.<br
/><br />The politically engaged Dmytryk perhaps inevitably succumbs to sermonizing, but it's pretty much confined to Young's re
miniscence of how his Irish grandfather died at the hands of bigots a century earlier (thus, incidentally, stretching chronolog
y to the limit). At least there's no attempt to render an explanation, however glib, of why Ryan hates Jews (and hillbillies an
d...).<br /><br />Curiously, Crossfire survives even the major change wrought upon it -- the novel it's based on (Richard Brook
s' The Brick Foxhole) dealt with a gay-bashing murder. But homosexuality in 1947 was still Beyond The Pale. News of the Holocau
st had, however, begun to emerge from the ashes of Europe, so Hollywood felt emboldened to register its protest against anti-Se
mitism (the studios always quaked at the prospect of offending any potential ticket buyer).<br /><br />But while the change fro
m homophobia to anti-Semitism works in general, the specifics don't fit so smoothly. The victim's chatting up a lonesome, drunk
young soldier then inviting him back home looks odd, even though (or especially since) there's a girlfriend in tow. It raises t
he question whether this scenario was retained inadvertently or left in as a discreet tip-off to the original engine generating
Ryan's murderous rage.
======================================================================
Actual sentiment is  : positive
======================================================================
Predicted sentiment is positive with a probability of 0.5837451219558716
```

Conclusion

- Used the IMDB movie reviews dataset (50K samples) for binary sentiment classification (positive/negative)
- Preprocessed text using custom tokenization and one hot encoding of the 1000 most frequent words
- Padded sequences to a fixed length of 500 tokens
- Built and trained an LSTM model with embedding, dropout, and a fully connected output layer
- Achieved ~86% training accuracy and ~85% validation accuracy after 5 epochs

- Slight rise in validation loss at the end suggests potential early overfitting
- Model was able to correctly predict sentiment of a nuanced test review