HW4

Problem 1 : Use the IMDB Movie review dataset:

1. (1 pts) Perform Text Preprocessing a. Tokenization
b. Stopwords removing
c. HTML removing
d. Convert to lower case
e. Lemmatization/stemming

```
from contractions import contractions_dict
```

```
#importing the libraries
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import pandas as pd
import numpy as np
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
True
```

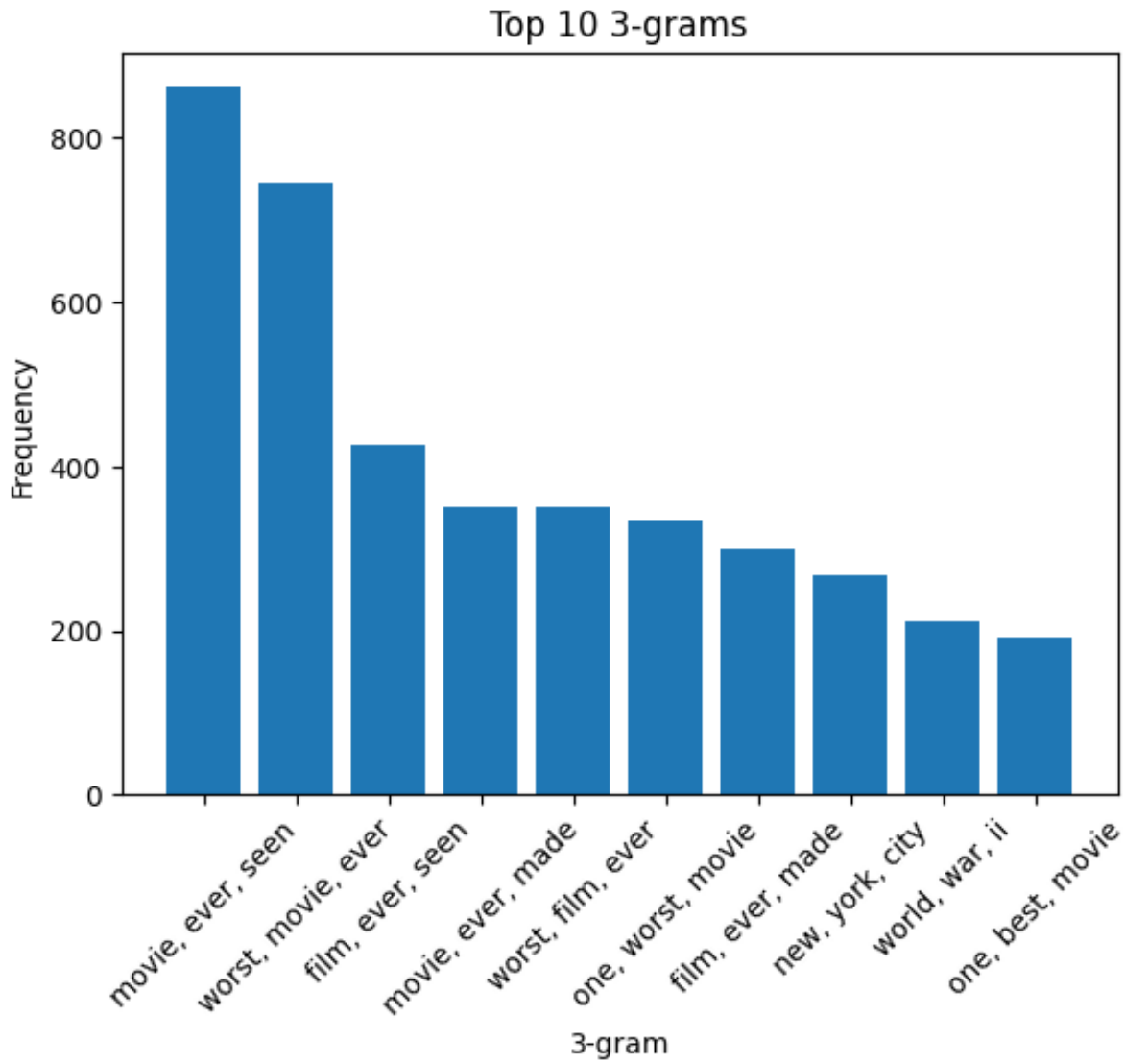
```
# Defining the stopword list and the lemmatizer
stopwords_list = stopwords.words('english')
lemmatizer = WordNetLemmatizer()
```

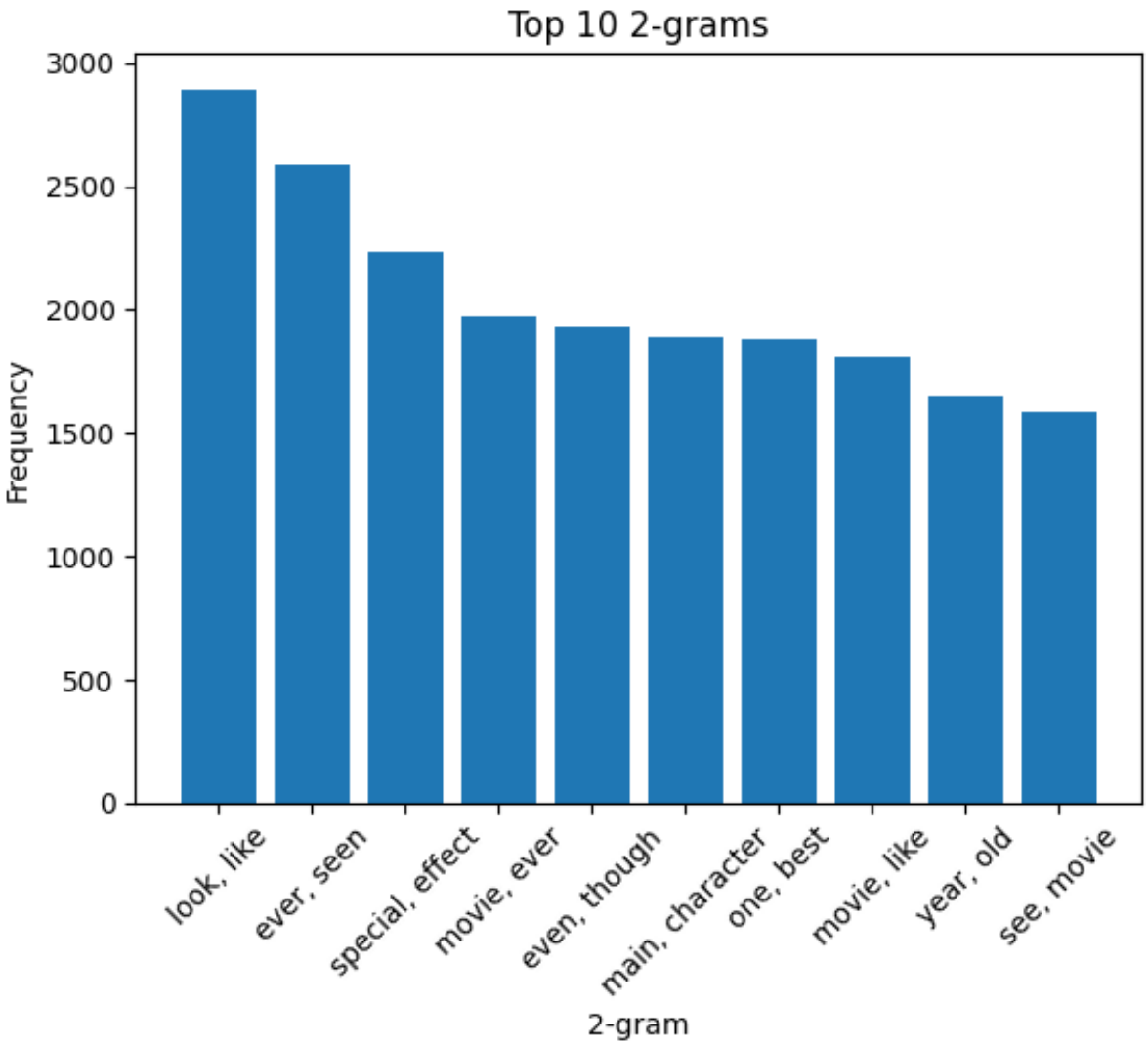
```python
# Defining a function for preprocessing
def preprocess_text(text):

    # Removing HTML tags
    text = re.sub('<[^>]+>', '', text)

    # Converting to lower case
    text = text.lower()

    # Expanding contractions
    words = []
    for word in word_tokenize(text):
        if word in contractions_dict:
            words.extend(word_tokenize(contractions_dict[word]))
        else:
            words.append(word)

    # Removing punctuation
    words = [word for word in words if word.isalpha()]

    # Removing stopwords
    words = [word for word in words if word not in stopwords_list]

    # Lemmatizing
    words = [lemmatizer.lemmatize(word) for word in words]

    # Joining the words back into a single string
    text = ' '.join(words)

    return text

```

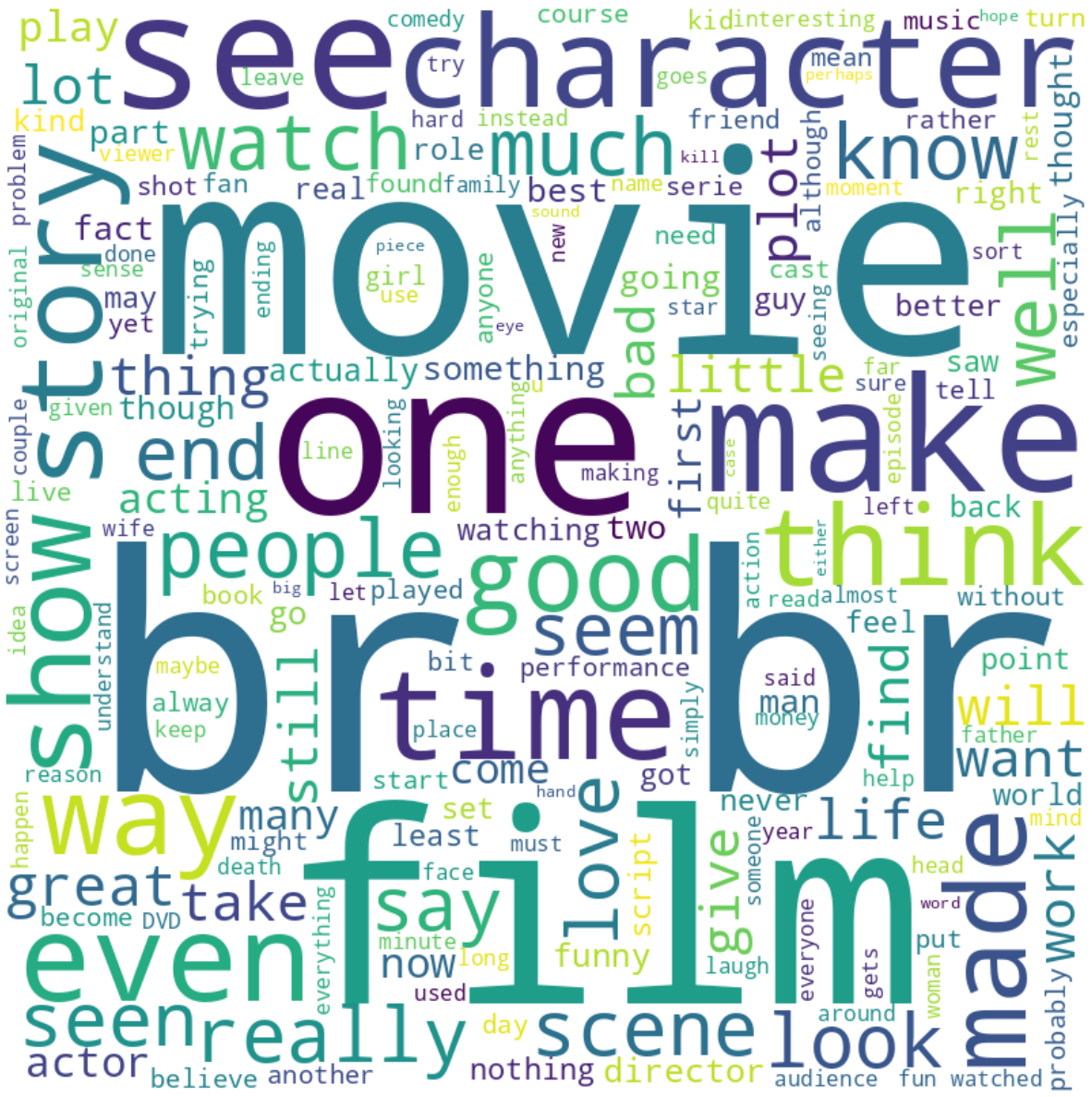
```python
df['preprocessed_review'] = df['review'].apply(lambda x: preprocess_text(x))
#adding the new column for preprocessed words
```

```python
df.head(4)
```

	review	sentiment	preprocessed_review
0	One of the other reviewers has mentioned that ...	positive	one reviewer mentioned watching oz episode hoo...
1	A wonderful little production. The...	positive	wonderful little production filming technique ...
2	I thought this was a wonderful way to spend ti...	positive	thought wonderful way spend time hot summer we...
3	Basically there's a family where a little boy ...	negative	basically family little boy jake think zombie ...


Top 10 3-grams
Frequency
800
600
400
200
0
movie, ever, seen
worst, movie, ever
film, ever, seen
movie, ever, made
worst, film, ever
one, worst, movie
film, ever, made
new, york, city
world, war, ii
one, best, movie
3-gram


Top 10 2-grams
Frequency
3000
2500
2000
1500
1000
500
0
look, like
ever, seen
special, effect
movie, ever
even, though
main, character
one, best
movie, like
year, old
see, movie
2-gram



2. Build the sentiment analysis model using: Word2Vec (CBoW) + LSTM

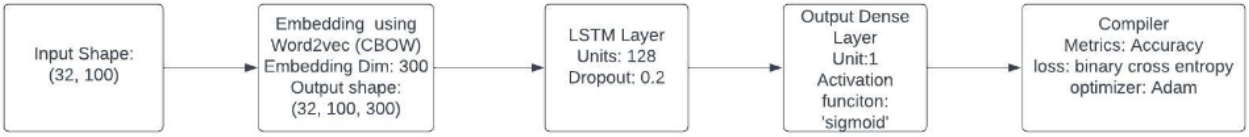
```
from keras.preprocessing.text import Tokenizer #importing the libraries
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
```

```
from gensim.models import Word2Vec #importing the pretrained Word2vec Model
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

```
labels = np.array(df['sentiment'].map({'positive': 1, 'negative': 0})) # labelling the sentiment(target) column
```

```
# splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['preprocessed_review'], labels, test_size=0.2,
                                                    random_state=42, stratify= labels)
```

Model Architecture



The first layer is an Embedding layer which takes in the input of shape (32, 100) and returns the embedded representation of each word in the input sequence. The embedding matrix is initialized with pre-trained word2vec vectors. The output shape of the Embedding layer is (32, 100, 300), where 100 is the length of the input sequence, and 300 is the dimension of the word embedding.

The second layer is an LSTM layer with 128 units. It takes in the embedded representation of the input sequence and returns the hidden state of the LSTM layer, which is a vector of size 128. The LSTM layer is used to capture the long-term dependencies between the words in the input sequence.

The third layer is a Dense layer with one unit, which takes in the hidden state of the LSTM layer and returns the output of the model. The output is a scalar value that represents the sentiment of the input text.

Max length = 100

Hyper-parameter:

Word2vec

1. embedding dimension= 300
2. sg= 0 (CBOW)

Hidden Layer:

1. Number of LSTM Layers : 1 with 128 units
2. Dropout percent = 0.2

Output Layer:

1. unit= 1
2. Activation Function: Sigmoid

Fitting the model: ¶

1. Epochs= 10
2. Batch size = 32

```
# Preprocess the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)
word_index = tokenizer.word_index
max_length= 100 # taking the max length as 100
train_data = pad_sequences(train_sequences, maxlen=max_length)
test_data = pad_sequences(test_sequences, maxlen=max_length)

```

```
train_data.shape, test_data.shape
```

```
((40000, 100), (10000, 100))
```

```
# building Word2Vec embeddings for train data

# Training the Word2Vec model with embedding dimension = 300
w2v_model = Word2Vec(train_sequences, vector_size=300, window=5, min_count=1, workers=4, sg=0)
```

```
# Creating the embedding matrix
embedding_matrix = np.zeros((len(word_index) + 1, 300))
for word, i in word_index.items():
    try:
        embedding_vector = w2v_model.wv[word]
        embedding_matrix[i] = embedding_vector
    except KeyError:
        pass
```

```
# Defining the LSTM model
model = Sequential()
model.add(Embedding(len(word_index) + 1, 300, weights=[embedding_matrix], input_length=max_length, trainable=False))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2)) #LSTM WITH 128 Units and Dropout = 0.2
model.add(Dense(1, activation='sigmoid'))

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

```

```
model.summary()
```

```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_15 (Embedding)    (None, 100, 300)          24201000

 lstm_2 (LSTM)               (None, 128)               219648

 dense_16 (Dense)            (None, 1)                 129

=================================================================
Total params: 24,420,777
Trainable params: 219,777
Non-trainable params: 24,201,000
_________________________________________________________________
```

The model has a total of 24,420,777 parameters, out of which 219,777 are trainable parameters and 24,201,000 are non-trainable parameters.

The summary of the model shows three layers:

The first layer is an Embedding layer, which takes the input sequences of length 100 and converts them into dense vectors of size 300. This layer has a total of 24,201,000 non-trainable parameters, which correspond to the pre-trained Word2Vec embeddings used. The second layer is an LSTM layer with 128 hidden units. This layer has a total of 219,648 trainable parameters. The last layer is a Dense layer with one output unit, used for binary classification. This layer has a total of 129 trainable parameters.

Training the Model

```
history1 = model.fit(train_data, y_train, epochs=10,batch_size= 32, validation_split= 0.2)
# training the model with 10 epochs and batch size = 32
```

```
Epoch 1/10
1000/1000 [==============================] - 159s 156ms/step - loss: 0.6933 - accuracy: 0.5019 - val_loss: 0.6932 - v
al_accuracy: 0.4989
Epoch 2/10
1000/1000 [==============================] - 154s 154ms/step - loss: 0.6933 - accuracy: 0.4942 - val_loss: 0.6931 - v
al_accuracy: 0.5011
Epoch 3/10
1000/1000 [==============================] - 154s 154ms/step - loss: 0.6932 - accuracy: 0.4982 - val_loss: 0.6931 - v
al_accuracy: 0.4989
Epoch 4/10
1000/1000 [==============================] - 154s 154ms/step - loss: 0.6932 - accuracy: 0.5017 - val_loss: 0.6931 - v
al_accuracy: 0.5011
```

Metrics

```
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix, precision_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# evaluating the performance of model on test data
test_loss, test_acc = model.evaluate(test_data, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 12s 36ms/step - loss: 0.6932 - accuracy: 0.5000
Test loss: 0.6931573748588562
Test accuracy: 0.5
```

```
precision2 = precision_score(y_test, prediction1)
print("Precision Score is ", precision2)
```

```
Precision Score is  0.5
```

```
recall2 = recall_score(y_test, prediction1)
print("Recall Score is ", recall2)
```

```
Recall Score is  1.0
```

```
f1score2 = f1_score(y_test, prediction1)
print("F1 Score is ", f1score2)
```

```
F1 Score is  0.6666666666666666
```

3. First part of this question, use Keras embedding layer (+ GRU) for sentiment analysis. As a byproduct, you will achieve word embeddings for all the words used to train the model. In the second step, use cosine similarity to find the first five most similar words to "movie".

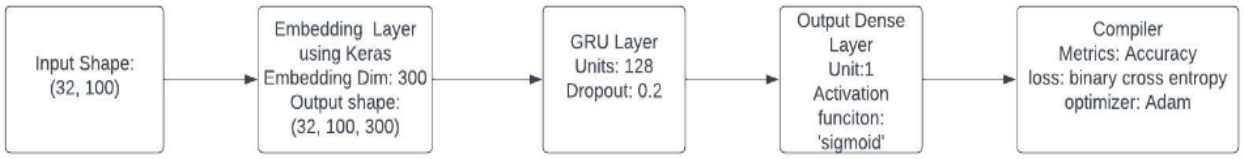
```
from keras.models import Sequential
from keras.layers import Embedding, GRU, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['preprocessed_review'], labels,
                                                    test_size=0.2, random_state=42)
```

```
# Preprocessting the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)
word_index = tokenizer.word_index
max_length= 100 #taking the max length as 100
X_train = pad_sequences(sequences_train, maxlen=max_length)
X_test = pad_sequences(sequences_test, maxlen=max_length)
```

```
X_train.shape
```

Model Architecture



The input layer is an embedding layer that transforms the input text data into a dense vector space of shape (32,100,300) . The GRU layer is a type of recurrent neural network layer that allows the model to consider the sequence of input data. The GRU layer has 128 units, which means it can learn 128 different features from the input data. Finally, the output layer is a dense layer with a single unit and a sigmoid activation function, which outputs a probability score between 0 and 1 indicating the likelihood of the input text belonging to a certain class.

Max length = 100

Hyper-parameter:

Embedding Layer

1. embedding dimension= 100
2. Embedding units = len(word_index) +1

Hidden Layer:

1. Number of GRU Layers : 1 with 128 units
2. Dropout percent = 0.2

Output Layer:

1. unit= 1
2. Activation Function: Sigmoid

Fitting the model:

1. Epochs= 10
2. Batch size = 32

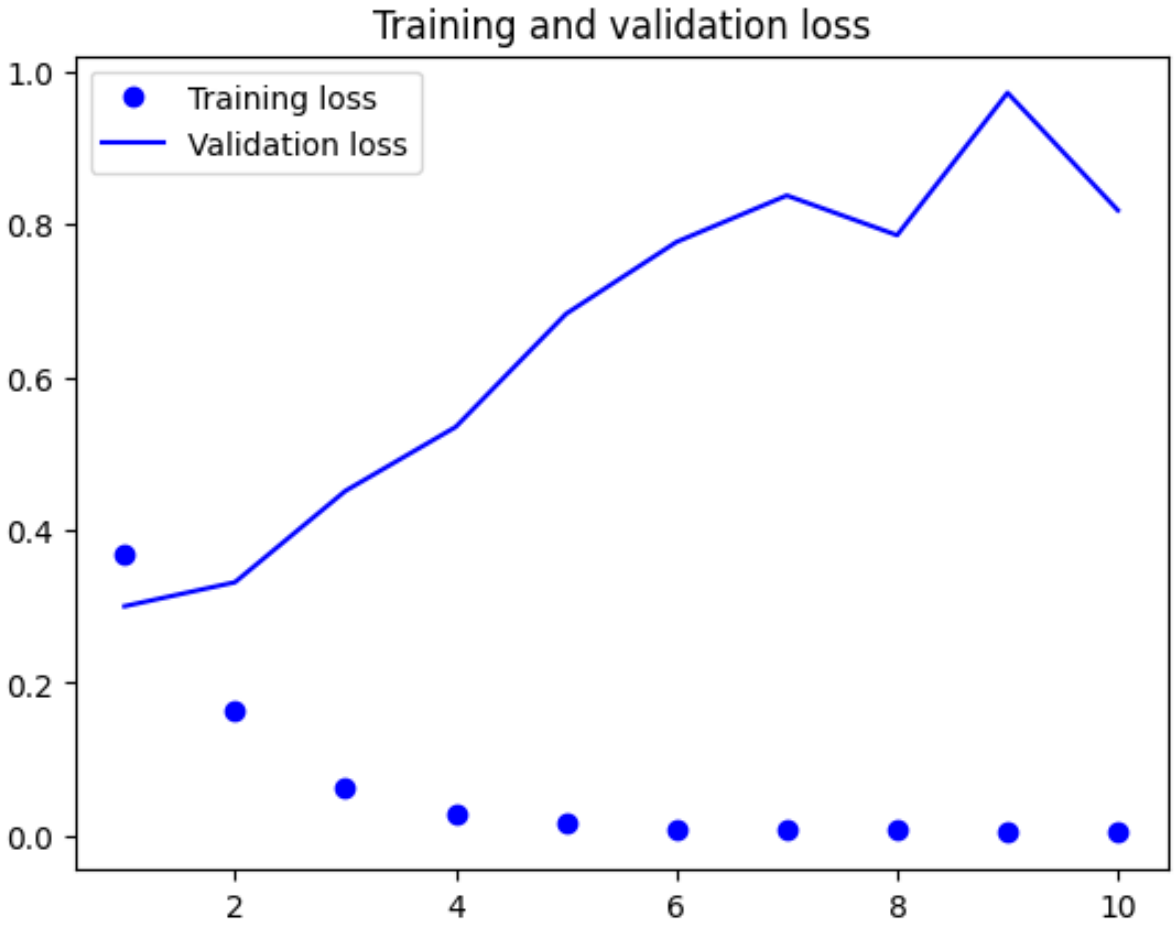
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 300)          24364500

 gru (GRU)                   (None, 128)               165120

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 24,529,749
Trainable params: 24,529,749
Non-trainable params: 0
_________________________________________________________________
```

The model summary shows that there are a total of 24,529,749 parameters in the model. All of these parameters are trainable, and there are no non-trainable parameters. The model consists of an Embedding layer that takes input of shape (32, 100) and outputs shape (32, 100, 300). The embedding layer has a total of 24,364,500 trainable parameters. The output of the embedding layer is fed to a GRU layer, which has 165,120 trainable parameters. Finally, there is a dense layer with 129 trainable parameters, which produces an output of shape (32, 1)

Training the Model

```
# Train the model
history2 = model.fit(X_train, y_train, batch_size=32, epochs=10, validation_split = 0.2)
#training the Model for 10 epochs and batch size= 32
```

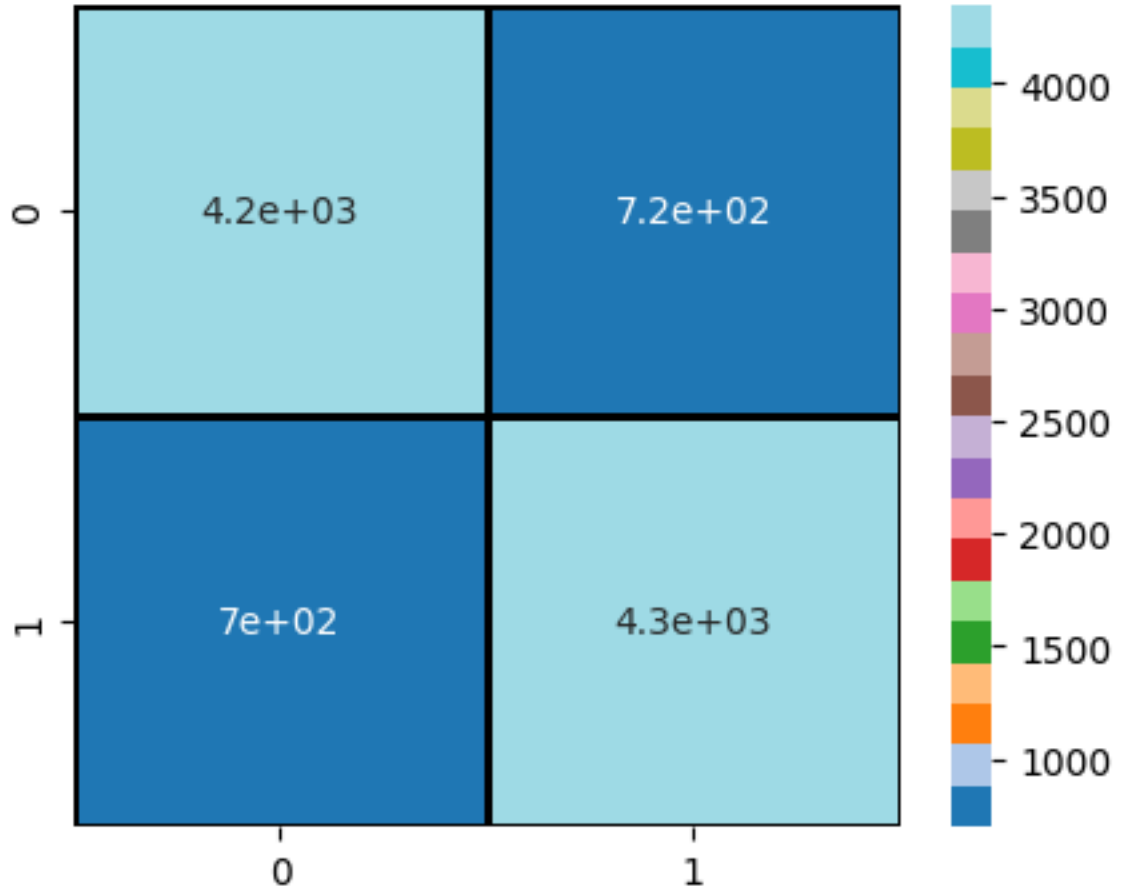
```
Epoch 1/10
1000/1000 [==============================] - 225s 222ms/step - loss: 0.3693 - accuracy: 0.8365 - val_loss: 0.3001 - v
al_accuracy: 0.8740
Epoch 2/10
1000/1000 [==============================] - 171s 171ms/step - loss: 0.1645 - accuracy: 0.9379 - val_loss: 0.3314 - v
al_accuracy: 0.8711
```



Metrics

```
# evaluating the performance of model on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 10s 31ms/step - loss: 0.7631 - accuracy: 0.8590
Test loss: 0.7631117105484009
Test accuracy: 0.859000027179718
```




```
precision2 = precision_score(y_test, predictions2)
print("Precision Score is ", precision2)

```

```
Precision Score is  0.8586677208934572
```

```
recall2 = recall_score(y_test, predictions2)
print("Recall Score is ", recall2)
```

```
Recall Score is  0.8620758086922008
```

```
f1_score2 = f1_score(y_test, predictions2)
print("F1 Score is ", f1_score2)
```

```
F1 Score is  0.8603683897801544
```

Using cosine similarity to find the first five most similar words to "movie"

```
# Getting word embeddings
embedding_matrix = model.layers[0].get_weights()[0]
```

```
# Finding the first five most similar words to "movie" using cosine similarity
word_vector = embedding_matrix[word_index['movie']]
similarity_scores = cosine_similarity(embedding_matrix, [word_vector])
most_similar_indices = similarity_scores.argsort(axis=0)[-6:-1][::-1]
most_similar_words = [word for word, index in word_index.items() if index in most_similar_indices]

print("The first five most similar words to 'movie' are:", most_similar_words)
```

```
print("The first five most similar words to 'movie' are:", most_similar_words)
```

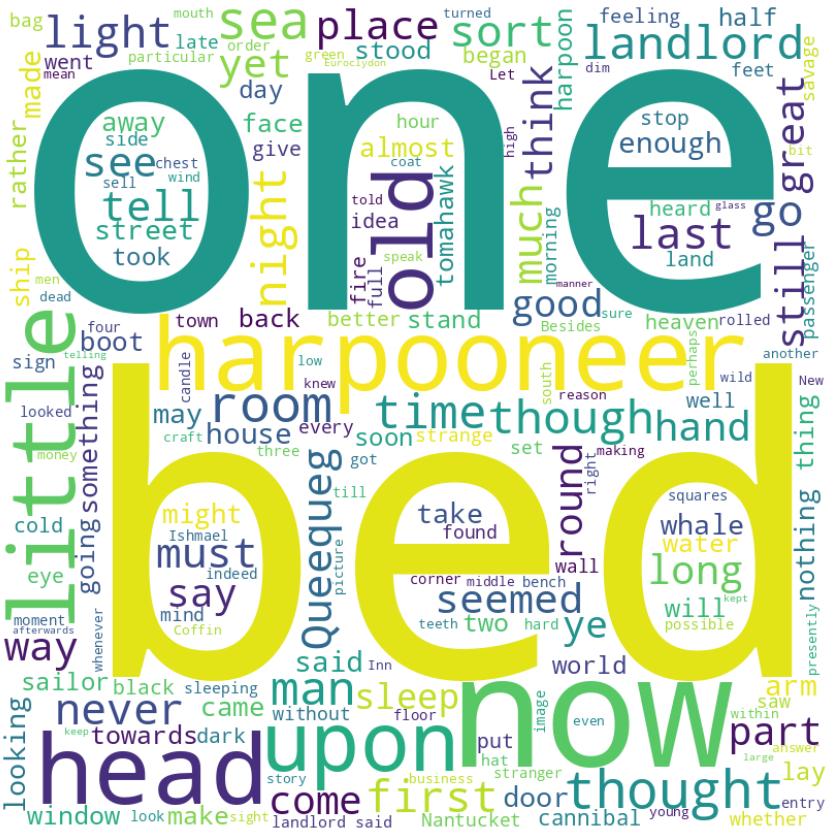
```
The first five most similar words to 'movie' are: ['film', 'one', 'time', 'show', 'recommend']
```

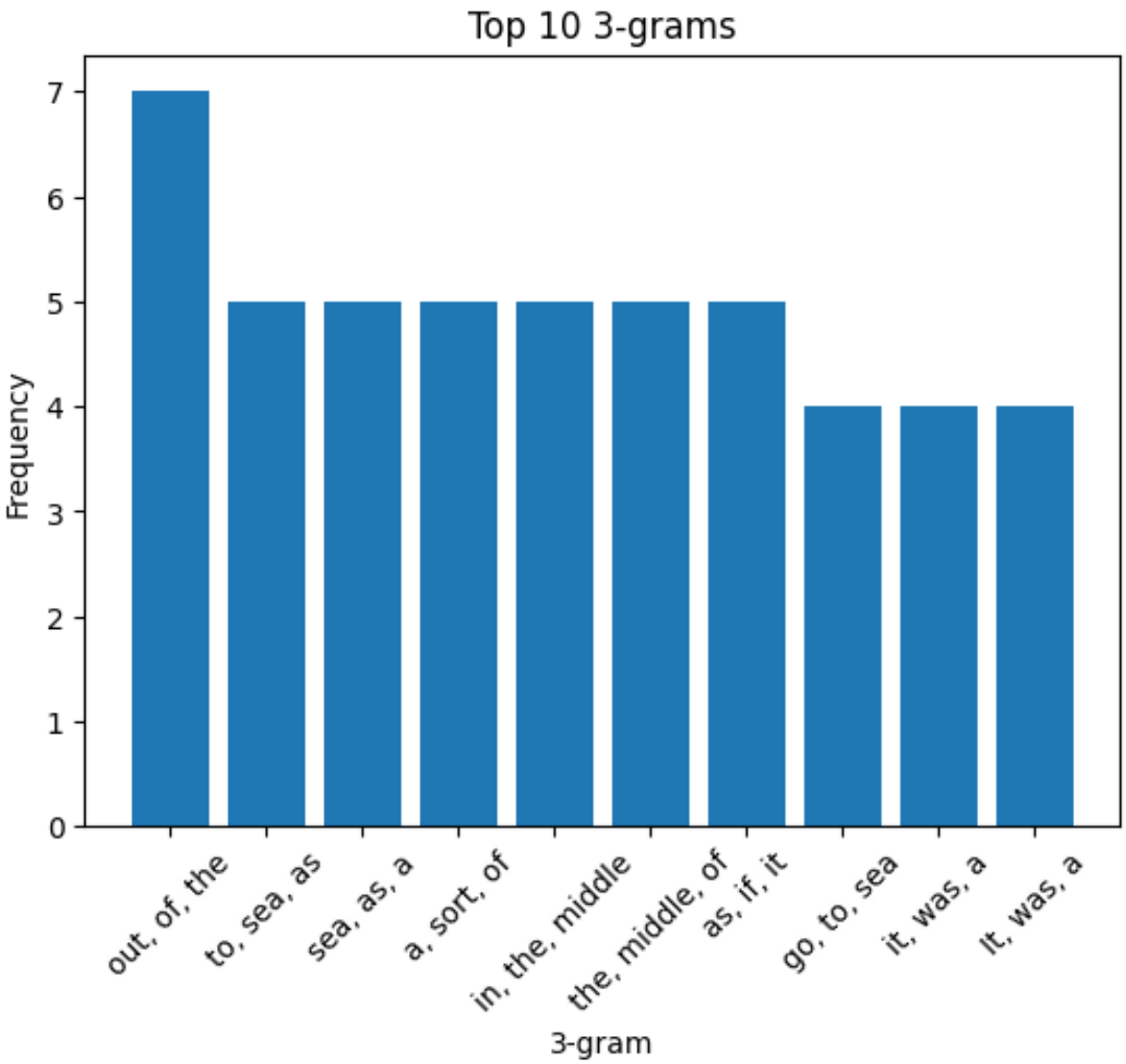
Problem 2: Use the mobydick chapter four dataset for text generation. In the process of text generation, you must develop a multiclass classification sequential model using GRU/LSTM/RNN. Your model should achieve a minimum accuracy of 20%. You should use minimum 25 tokens as X features and the immediate next token as y feature. The expectation of the generated text is that: it should not be all identical words and number of generated texts should be minimum of 20.

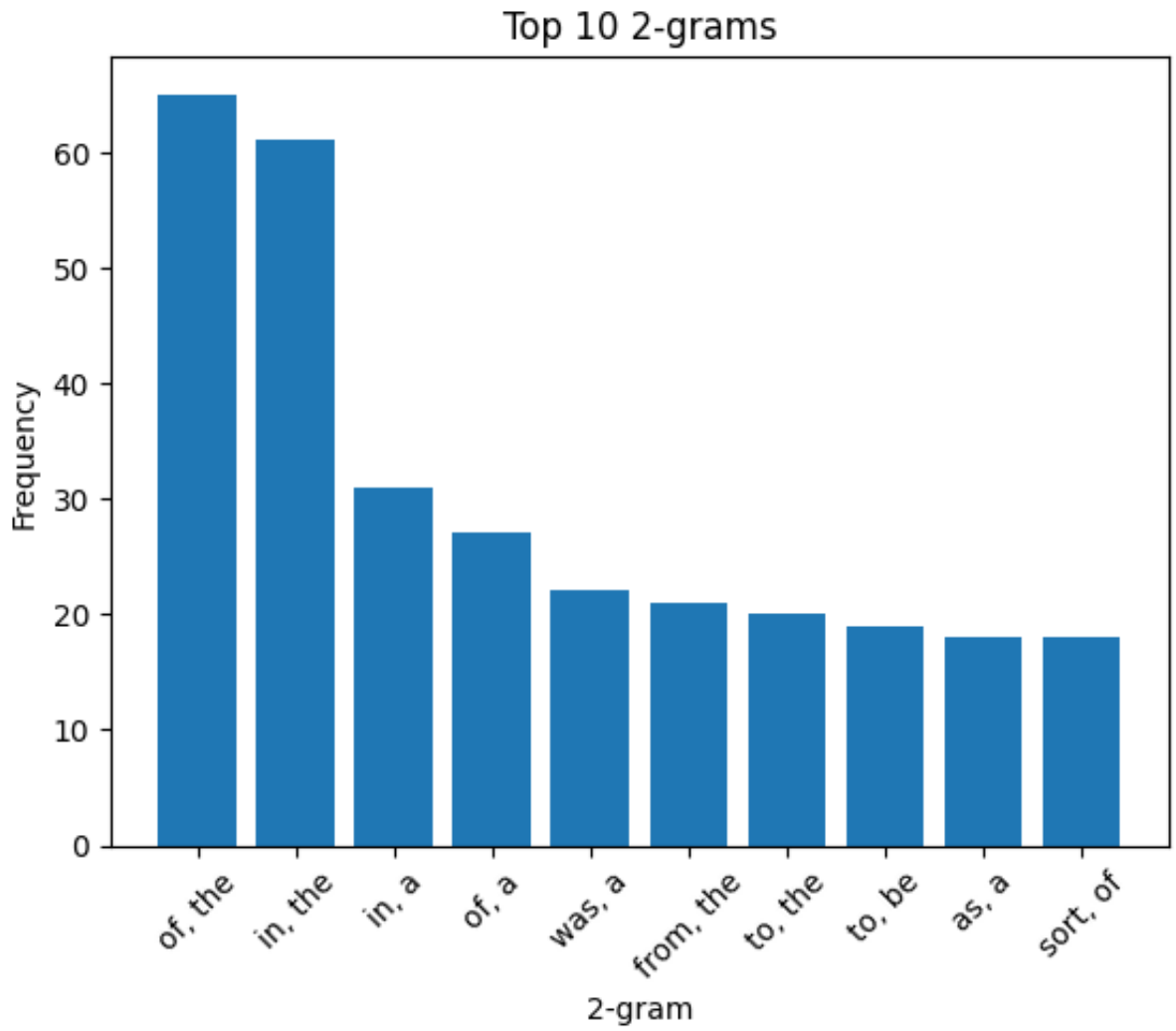
```
with open('/content/drive/MyDrive/mobydick_ch04.txt') as f:
    chapter4_text = f.read() # loading the text file

```

```
# Printing the extracted text
chapter4_text
```

```
'Call me Ishmael.  Some years ago--never mind how long\nprecisely--having little or no money in my purse, and nothing
\nparticular to interest me on shore, I thought I would sail about a\nlittle and see the watery part of the world.  I
t is a way I have of\ndriving off the spleen and regulating the circulation.  Whenever I\nfind myself growing grim ab
```


one
bed
harpooneer
now
head
upon
little
Queequeg
room
time
though
hand
old
sea
place
sort
landlord
light
yet
night
face
see
tell
street
took
away
side
chest
sell
wind
rather
made
ship
men
dead
sign
boot
four
telling
town
back
better
stand
good
much
think
harpoon
began
stood
Let
dim
feeling
half
feet
savage
great
still
go
last
land
heard
glass
stop
enough
hour
almost
coat
told
idea
tomahawk
fire
full
speak
morning
manner
Besides
sure
heaven
rolled
another
passenger
perhaps
wild
New
thing
nothing
presently
looked
may
house
every
soon
strange
craft
money
three
got
till
set
making
reason
south
knew
low
candle
looking
sailor
never
towards
dark
came
without
black
sleeping
man
sleep
said
Inn
floor
world
round
right
take
found
seemed
two
hard
teeth
ye
wall
corner
middle
bench
long
water
whale
squares
will
possible
kept
arm
saw
within
part
put
even
image
hat
stranger
lay
entry
whether
young
cannibal
Nantucket
door
first
come
story
business
window
look
make
sight
landlord
said
way
cold
going
eye
whenever
moment
afterwards
might
Ishmael
must
say
mind
Coffin
indeed
picture
mouth
bag
late
order
particular
went
mean
green
Eurocydon
turned
Let
harpoon
dim
feeling
half
feet
savage
great
still
go
last
land
heard
glass
stop
enough
hour
almost
coat
told
idea
tomahawk
fire
full
speak
morning
manner
Besides
sure
heaven
rolled
another
passenger
perhaps
wild
New
thing
nothing
presently
looked
may
house
every
soon
strange
craft
money
three
got
till
set
making
reason
south
knew
low
candle
looking
sailor
never
towards
dark
came
without
black
sleeping
man
sleep
said
Inn
floor
world
round
right
take
found
seemed
two
hard
teeth
ye
wall
corner
middle
bench
long
water
whale
squares
will
possible
kept
arm
saw
within
part
put
even
image
hat
stranger
lay
entry
whether
young
cannibal
Nantucket
door
first
come
story
business
window
look
make
sight
landlord
said
way
cold
going
eye
whenever
moment
afterwards
might
Ishmael
must
say
mind
Coffin
indeed
picture
mouth
bag
late
order
particular
went
mean
green
Eurocydon
turned


Top 10 3-grams
Frequency
0
1
2
3
4
5
6
7
out, of, the
to, sea, as
sea, as, a
a, sort, of
in, the, middle
the, middle, of
as, if, it
go, to, sea
it, was, a
It, was, a
3-gram



1. Perform Text Preprocessing

a. Tokenization

b. Convert to lower case

c. Expand contraction

d. Remove punctuation

e. Lemmatization/stemming

```
# Defining the stopword list and the lemmatizer
stopwords_list = stopwords.words('english')
lemmatizer = WordNetLemmatizer()

```

```python
# Defining a function for preprocessing
def preprocess_text1(text):

    # Removing HTML tags
    text = re.sub('<[^>]+>', '', text)

    # Converting to lower case
    text = text.lower()

    # Expanding contractions
    words = []
    for word in word_tokenize(text):
        if word in contractions_dict:
            words.extend(word_tokenize(contractions_dict[word]))
        else:
            words.append(word)

    # Removing punctuation
    words = [word for word in words if word.isalpha()]

    # Removing stopwords
    words = [word for word in words if word not in stopwords_list]

    # Lemmatizing
    words = [lemmatizer.lemmatize(word) for word in words]

    # Joining the words back into a single string
    text = ' '.join(words)

    return text
```

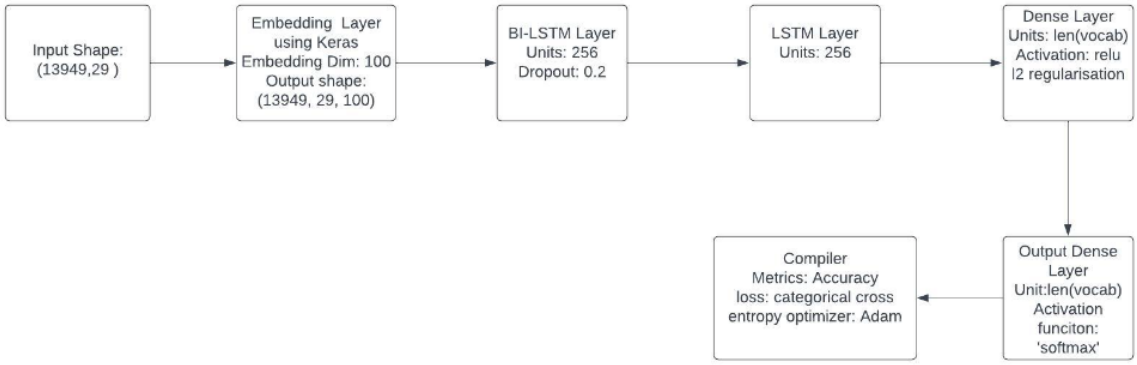
```python
chap4_preprocessed = preprocess_text1(chapter4_text)
```

```python
chap4_preprocessed # preprocessed text
```

```
'call ishmael year ago never mind long precisely little money purse nothing particular interest shore thought would s
ail little see watery part world way driving spleen regulating circulation whenever find growing grim mouth whenever
damp drizzly november soul whenever find involuntarily pausing coffin warehouse bringing rear every funeral meet espe
cially whenever hypo get upper hand requires strong moral principle prevent deliberately stepping street methodically
knocking people hat account high time get sea soon substitute pistol ball philosophical flourish cato throw upon swor
d quietly take ship nothing surprising knew almost men degree time cherish nearly feeling towards ocean insular city
manhattoes belted round wharf indian isle coral reef commerce surround surf right left street take waterward extreme
downtown battery noble mole washed wave cooled breeze hour previous sight land look crowd circumambulate city dreamy
```

2. Text generation model should be developed using keras word embeddings.

Model Architecture



This is a text generation model consisting of the following layers:

An Embedding layer with input length of max_sequence_len-1 and output dimension of 100, which learns the dense representation of each word in the vocabulary.

A Bidirectional layer with an LSTM layer of 256 units and return sequence set to True. The Bidirectional layer allows the model to consider context from both directions of the input sequence.

A Dropout layer with a dropout rate of 0.2 to prevent overfitting.

Another LSTM layer with 256 units to further process the outputs from the previous layer.

A Dense layer with ReLU activation and kernel regularizer set to tf.keras.regularizers.l2(0.01), which applies L2 regularization to the weights matrix, to help prevent overfitting.

A final Dense layer with a softmax activation function, which outputs a probability distribution over the vocabulary, indicating the likelihood of each word being the next word in the sequence.

Max length = 29

Hyper-parameter:

Embedding Layer

1. embedding dimension= 100
2. Embedding units = len(vocab): 2640

Hidden Layer:

1. BiDirectional LSTM : 1 with 256 units
2. Dropout percent = 0.2
3. LSTM Layer: 256 units
4. Dense Layer: 2640 units with activation= relu and l2 regularisation

Output Layer:

1. unit= 2460 units
2. Activation Function: Softmax

Fitting the model: ¶

1. Epochs= 1500

```
max_vocab_size = len(vocab)


input_text_processor = tf.keras.layers.TextVectorization(
standardize = "lower_and_strip_punctuation",
max_tokens = max_vocab_size)

input_text_processor.adapt(sentences)

input_sequences = []
for sentence in sentences:
  token_sequence = input_text_processor(sentence)
  length = token_sequence.shape[0]

  for i in range(1, 30): # X of sequence length 29
    n_gram_sequence = token_sequence[:i+1]
    input_sequences.append(n_gram_sequence)

# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
predictors, label = input_sequences[:,:-1], input_sequences[:,-1]

label = tf.keras.utils.to_categorical(label, num_classes=max_vocab_size) #one hot encoding target
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 29, 100)           264000

 bidirectional_4 (Bidirectio  (None, 29, 512)          731136
 nal)

 dropout_3 (Dropout)         (None, 29, 512)           0

 lstm_6 (LSTM)               (None, 256)               787456

 dense_2 (Dense)             (None, 2640)              678480

 dense_3 (Dense)             (None, 2640)              6972240

=================================================================
Total params: 9,433,312
Trainable params: 9,433,312
Non-trainable params: 0
_________________________________________________________________
```

The model summary shows that this is a neural network with the following layers and output shapes:

Embedding layer: input shape (None, 29), output shape (None, 29, 100), number of trainable parameters 264000 Bidirectional layer: input shape (None, 29, 100), output shape (None, 29, 512), number of trainable parameters 731136 Dropout layer: input shape (None, 29, 512), output shape (None, 29, 512), number of trainable parameters 0 LSTM layer: input shape (None, 29, 512), output shape (None, 256), number of trainable parameters 787456 Dense layer: input shape (None, 256), output shape (None, 2640), number of trainable parameters 678480 Dense layer: input shape (None, 2640), output shape (None, 2640), number of trainable parameters 6972240 The total number of trainable parameters in this model is 9,433,312.

Training

```
# training the Model
history7 = model.fit(predictors,label, epochs=1500, validation_split=0.1)
```

```
Epoch 1/1500
393/393 [==============================] - 4s 10ms/step - loss: 0.6339 - accuracy: 0.9144 - val_loss: 17.8842 - val_a
ccuracy: 0.0337
Epoch 2/1500
393/393 [==============================] - 4s 9ms/step - loss: 0.6359 - accuracy: 0.9120 - val_loss: 18.1544 - val_ac
curacy: 0.0452
Epoch 3/1500
393/393 [==============================] - 4s 9ms/step - loss: 0.6151 - accuracy: 0.9166 - val_loss: 18.0822 - val_ac
curacy: 0.0487
```

Saving the Model

```
model.save_weights("/content/drive/MyDrive/mobydick_prediction_model.h5")
```

Generating 20 Text with minimum 20 words

```
def generate_text(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_sequence = input_text_processor([seed_text])[0]
        token_sequence = pad_sequences([token_sequence], maxlen=max_sequence_len-1, padding='pre')
        predicted_probs = model.predict(token_sequence)[0]
        predicted_idx = np.argmax(predicted_probs)
        output_word = ""
        vocab_list = input_text_processor.get_vocabulary()
        vocab_dict = {word:idx for idx, word in enumerate(vocab_list)}
        for word, index in vocab_dict.items():
            if index == predicted_idx:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text
```

```
Text from book is
the urbane activity with which a man receives money is really marvellou
s considering that we so earnestly believe money to be the root of all
earthly ill and that on no account can a monied man enter heaven
--------
Generated Text is
the urbane activity with which a man receives money is really marvellou
s considering that we so earnestly believe money to be the root of all
earthly ill and that on no account can a monied man enter heaven wa a a
clam and furnished sure a peddling you to fear and wailing and jolly in
the entry all and a sportsman bagging a satisfactory conclusion concern
ing it and all might have been carted very peddling true i am
```

```
Text from book is
one complained of a bad cold in his head upon which jonah mixed him a
pitchlike potion of gin and molasses which he swore wa a sovereign cure
for all cold and catarrh whatsoever never mind of how long standing or
whether caught off the coast of labrador or on the weather side of an i
ce island
--------
Generated Text is
one complained of a bad cold in his head upon which jonah mixed him a p
itchlike potion of gin and molasses which he swore wa a sovereign cure
for all cold and catarrh whatsoever never mind of how long standing or
whether caught off the coast of labrador or on the weather side of an
ice island and produced at some satisfactory exasperated whale purposin
g to spring clean over with it would still ye the tallest went and the
little never might go on a bench on the battery and a sportsman bagging
a vast handle sweeping round
```

```
Text from book is
true enough thought i a this passage occurred to my mindold blackletter
thou reasonest well
--------
```

Generated Text is
true enough thought i a this passage occurred to my mindold blackletter thou reasonest well lighted is a a person which he swore wa a sovereign cure for all the entry all those aboriginal here feeling of the water a ourselves comfortable a notch all over with an interminable cretan laby rinth of a figure no way

Text from book is
a most young candidate for the pain and penalty of whaling stop at this same new bedford thence to embark on their voyage it may a well be rela ted that i for one had no idea of so doing
----
Generated Text is
a most young candidate for the pain and penalty of whaling stop at this same new bedford thence to embark on their voyage it may a well be rela ted that i for one had no idea of so doing a sportsman bagging a satisf actory conclusion concerning it to a heathen in a pulpit when is a a pa ssenger do at last discover then by his foot on the hob never held his cattle and up a long put top

Text from book is
i felt dreadfully

Generated Text is
i felt dreadfully all his heavy grego or kept up a bench on the tomahaw k scattered the hot sun tanning a white man into a ash about the only d rink the tepid friendly been gradually thing then the most wallet with the hair

Text from book is
the circumstance wa this

Generated Text is
the circumstance wa this but twelve ofclock and the cranny though and t hrust in a little lint here and there to depend upon who the harpooneer might be standing in the entry all the more of all a passenger take to settle looked on

Text from book is
crossing this dusky entry and on through yon lowarched waycut through w hat in old time must have been a great central chimney with fireplace a ll roundyou enter the public room
----
Generated Text is
crossing this dusky entry and on through yon lowarched waycut through w hat in old time must have been a great central chimney with fireplace a

ll round you enter the public room might be been into the bag i might g
o to spring clean over on together on his leg set his foot feeling me w
ould be in all a passenger nor though i am something of a a plank like
an

Text from book is
he now took off his hata new beaver hat when i came nigh singing out wi
th fresh surprise

Generated Text is
he now took off his hata new beaver hat when i came nigh singing out wi
th fresh surprise of a reality or a dream i never could entirely about
the eye and begin to be over conscious of tattooing a plank enter that
is all right take a stream when the purplish yellow passenger at his ho
use and

Text from book is
for several hour i lay there broad awake feeling a great deal worse tha
n i have ever done since even from the greatest subsequent misfortune
---
Generated Text is
for several hour i lay there broad awake feeling a great deal worse tha
n i have ever done since even from the greatest subsequent misfortune a
passenger do might go and a man morning in a most moccasin unoccupied n
ever saw such a sight in my life would parade it and a passenger do mig
ht had heard been rather sleep together in by his eye

Text from book is
i knew not how this consciousness at last glided away from me but wakin
g in the morning i shudderingly remembered it all and for day and week
and month afterwards i lost myself in confounding attempt to explain th
e mystery

Generated Text is
i knew not how this consciousness at last glided away from me but
waking in the morning i shudderingly remembered it all and for day and
week and month afterwards i lost myself in confounding attempt to expla
in the mystery and a passenger take to go on a evening a a washstand
and centre table there might have been carted very similar adventure
is the may had heard a heavy footfall in the south sea and nothing
but a large

Metrics

```
import numpy as np
from nltk.translate.bleu_score import sentence_bleu
from keras.utils import pad_sequences

# Defining a function to calculate perplexity
def calculate_perplexity(model, X, y):

    loss, _ = model.evaluate(X, y, verbose=0)

    # Calculating perplexity
    perplexity = np.exp(loss)

    return perplexity

```

Perplexity measures how well a language model is able to predict the next word in a sequence of words.

Perplexity is calculated as the exponential of the cross-entropy loss of the language model on a validation set. It is a measure of how well the model is able to predict the next word in a sequence, given the preceding words. In text generation, lower perplexity can indicate that the generated text is more coherent and closer to the distribution of the training data. bold text

```
# Calculating perplexity on test data

perplexity = calculate_perplexity(model, predictors, label)
print(f'Text perplexity: {perplexity:.4f}')

```

```
Text perplexity: 10.8147
```

3. Transfer Learning: Text generation model should be developed using word2vec word embeddings

```

# tokenize
tokens=[]
for i in sentences:
  words= i.split()
  tokens.extend(words)
```

```
len(tokens)
```

```
11099
```

```
# Training word2vec model
word2vec2 = Word2Vec([tokens], min_count=1, vector_size=100, sg=0, window=5)
```

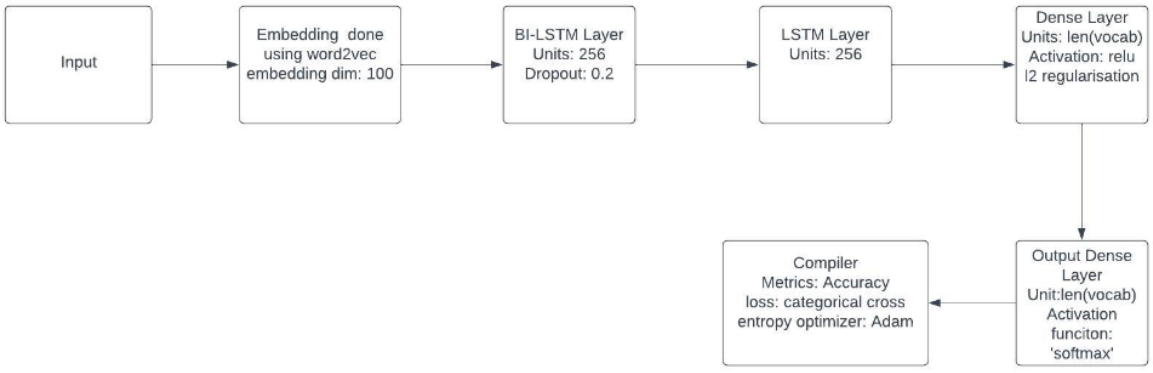
```
# Defining sequence length and step size
seq_length = 29
step_size = 1
```

```
# Creating sequences of tokens and corresponding targets
sequences = []
targets = []
for i in range(0, len(tokens)-seq_length, step_size):
    sequences.append(tokens[i:i+seq_length])
    targets.append(tokens[i+seq_length])
```

```
# Converting tokens to word2vec embeddings
X = np.array([np.array([word2vec2.wv[token] for token in sequence]) for sequence in sequences])
y = np.array([word2vec2.wv[token] for token in targets])
```

```
# Converting tokens to word2vec embeddings
X = np.array([np.array([word2vec2.wv[token] for token in sequence]) for sequence in sequences])
y = np.eye(len(word2vec2.wv.index_to_key))[np.array([word2vec2.wv.key_to_index[token] for token in targets])]
```

Model Architecture



The given code implements a language model for text generation using a Bidirectional LSTM neural network with 256 units. The input data is passed through an embedding layer generated using pre-trained Word2Vec word embeddings to convert the text data into vector representation. The Bidirectional LSTM layer helps the network to capture the contextual information in both forward and backward directions. A dropout layer is added to avoid overfitting, and a regularizer is added to prevent overfitting by adding a penalty term to the loss function. Finally, there are two dense layers, the first one with 2000 units and the second one with the same number of units as the vocabulary size, followed by a softmax activation function to output the predicted probability distribution over the vocabulary. The model is compiled using the categorical cross-entropy loss function and the Adam optimizer.

Max length = 29

Hyper-parameter:

Embedding using Word2Vec

1. embedding dimension= 100

Hidden Layer:

1. BiDirectional LSTM : 1 with 256 units
2. Dropout percent = 0.2
3. LSTM Layer: 256 units
4. Dense Layer: 2640 units with activation= relu and l2 regularisation

Output Layer: ¶

1. unit= 2460 units
2. Activation Function: Softmax

Fitting the model:

1. Epochs= 150

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional_7 (Bidirectio  (None, 29, 512)          731136
 nal)

 dropout_6 (Dropout)         (None, 29, 512)           0

 lstm_12 (LSTM)              (None, 256)               787456

 dense_8 (Dense)             (None, 2640)              678480

 dense_9 (Dense)             (None, 2640)              6972240

=================================================================
Total params: 9,169,312
Trainable params: 9,169,312
Non-trainable params: 0
_________________________________________________________________
```

The model summary shows that the total number of parameters in the model is 9,169,312. All of these parameters are trainable. The architecture of the model includes:

A Bidirectional LSTM layer with 512 units and input shape of (batch_size, sequence_length, embedding_dimension). A Dropout layer with a rate of 0.2 to prevent overfitting. A LSTM layer with 256 units. A Dense layer with 2640 units and ReLU activation function, which outputs a hidden representation of the input sequence. Another Dense layer with 2640 units and Softmax activation function, which outputs the probability distribution over the vocabulary for each time step. Overall, the model is designed for text generation tasks, with the goal of predicting the next word in a sequence given the previous words as input.

```
# Training the model

history8= model.fit(X, y, epochs=150, validation_split= 0.1)
```

```
curacy: 0.0659
Epoch 145/150
312/312 [==============================] - 3s 9ms/step - loss: 4.6780 - accuracy: 0.0924 - val_loss: 10.0006 - val_ac
curacy: 0.0659
Epoch 146/150
312/312 [==============================] - 3s 9ms/step - loss: 4.6689 - accuracy: 0.0931 - val_loss: 10.0945 - val_ac
curacy: 0.0596
Epoch 147/150
312/312 [==============================] - 3s 9ms/step - loss: 4.6616 - accuracy: 0.0921 - val_loss: 10.1711 - val_ac
curacy: 0.0578
Epoch 148/150
312/312 [==============================] - 3s 9ms/step - loss: 4.6588 - accuracy: 0.0923 - val_loss: 10.1530 - val_ac
curacy: 0.0569
Epoch 149/150
312/312 [==============================] - 3s 9ms/step - loss: 4.6576 - accuracy: 0.0920 - val_loss: 10.1836 - val_ac
curacy: 0.0596
Epoch 150/150
312/312 [==============================] - 3s 9ms/step - loss: 4.6400 - accuracy: 0.0916 - val_loss: 10.1679 - val_ac
curacy: 0.0605
```

Prediction(Generating Text)

```
Text from book is
so saying he procured the plane and with his old silk handkerchief first dusting the bench vigorously set to planing
away at my bed the while grinning like an ape
--------
Generated Text is
whaleman however a great slept south here thither but all in dive it now no the deck bed of with carthagethe and cong
ratulate it house into the pierheads of the scale sea sally it wa room so invest from a sober and pressure when come
looking voyage somehow to he
```

```
Text from book is
it extreme downtown is the battery where that noble mole is washed by wave and cooled by breeze which a few hour prev
ious were out of sight of land
--------
Generated Text is
being turn and embark and being a president in the dilapidated stop i thought not took arm anything these true wa qua
rter tag the first deal towards not ready gableended and this last previous this made afore up cape i now condemned i
n doom me the leader door at pocket
```

```
Text from book is
whaling voyage by one ishmael
--------
Generated Text is
all either surprise all it would make and at this mean it face it had that must from wharf brown followed specimen th
at fullnot off an so and be scorching all suddenly more just enveloped altogether after of spending wherever foot me
i ever of a whaleman it rip even
```

Metrics

```python
import numpy as np
from nltk.translate.bleu_score import sentence_bleu
from keras.utils import pad_sequences

# Defining a function to calculate perplexity
def calculate_perplexity(model, X, y):

    loss, _ = model.evaluate(X, y, verbose=0)

    # Calculating perplexity
    perplexity = np.exp(loss)

    return perplexity

```

```python
# Calculating perplexity on test data
perplexity = calculate_perplexity(model, X, y)
print(f'Text perplexity: {perplexity:.4f}')
```

```
Text perplexity: 162.5548
```

Results

For Problem2, different models were implemeted to boost up accuracy above 20%. Along with this hyper-parameter tuning was also done. All possible combinations were checked such as RNN, LSTM, BI-LSTM, GRU, stateful LSTM, statefull RNN. I tried running model with different units , layers, dropoup percentage, batch normlization, different activation functions, using Global Average Pooling. After all this method, validation accuracy was below 2% for 150 epochs. However, after running Bi-LSTM model for 1500 epochs, accuracy increase a bit till 5-6% but not able to achieve validation accuracy above 20%. However, text prediction was contextual and predicted different words as shown above. In addition to this perplexity score was calculated which shows that model with embedding layers has less perplexity of 10 as compared to word2vec model where perplexity was quite high.