GOODREADS BOOKS RATING PREDICTION

DEEP LEARNING
DIGITAL ASSIGNMENT- 2

TEAM MEMBERS

SAI TEJA BANDARU (20BRS1129)
SARATH ADHITHYA S (20BRS1194)

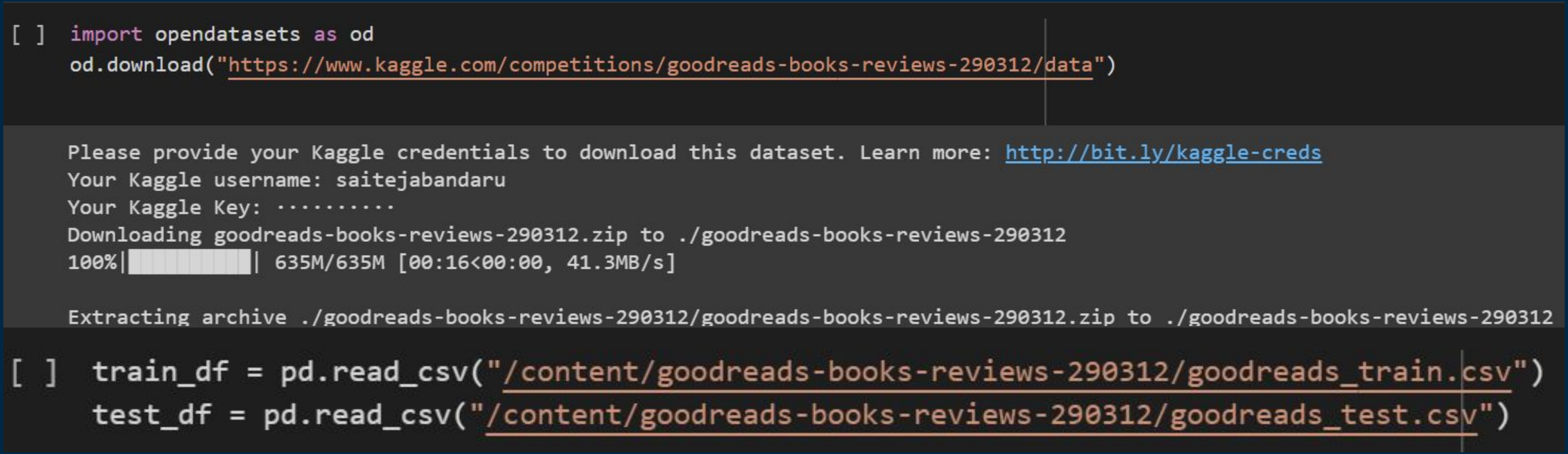
GITHUB LINK

https://github.com/SarathAdhi/DA2-DL-GoodReads-Rating-Prediction

DATA COLLECTION

The opendatasets Python library is used to download the "Goodreads-books-reviews" competition dataset on Kaggle.

The od.download() function initiates the download of the dataset and saves it to the current working directory. By default, it extracts the contents of the downloaded zip file automatically.



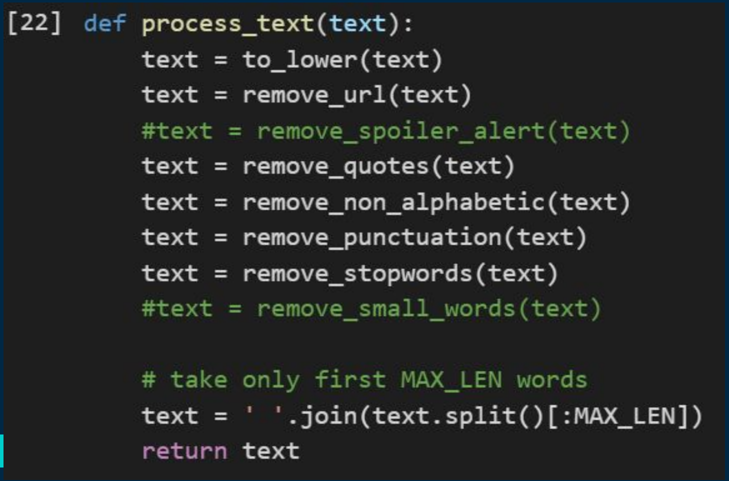
```
[ ]  import opendatasets as od
     od.download("https://www.kaggle.com/competitions/goodreads-books-reviews-290312/data")

     Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
     Your Kaggle username: saitejabandaru
     Your Kaggle Key: ··········
     Downloading goodreads-books-reviews-290312.zip to ./goodreads-books-reviews-290312
     100%|██████████| 635M/635M [00:16<00:00, 41.3MB/s]

     Extracting archive ./goodreads-books-reviews-290312/goodreads-books-reviews-290312.zip to ./goodreads-books-reviews-290312
```

```
[ ]  train_df = pd.read_csv("/content/goodreads-books-reviews-290312/goodreads_train.csv")
     test_df = pd.read_csv("/content/goodreads-books-reviews-290312/goodreads_test.csv")
```

DATA PREPROCESSING

The text preprocessing procedures include:

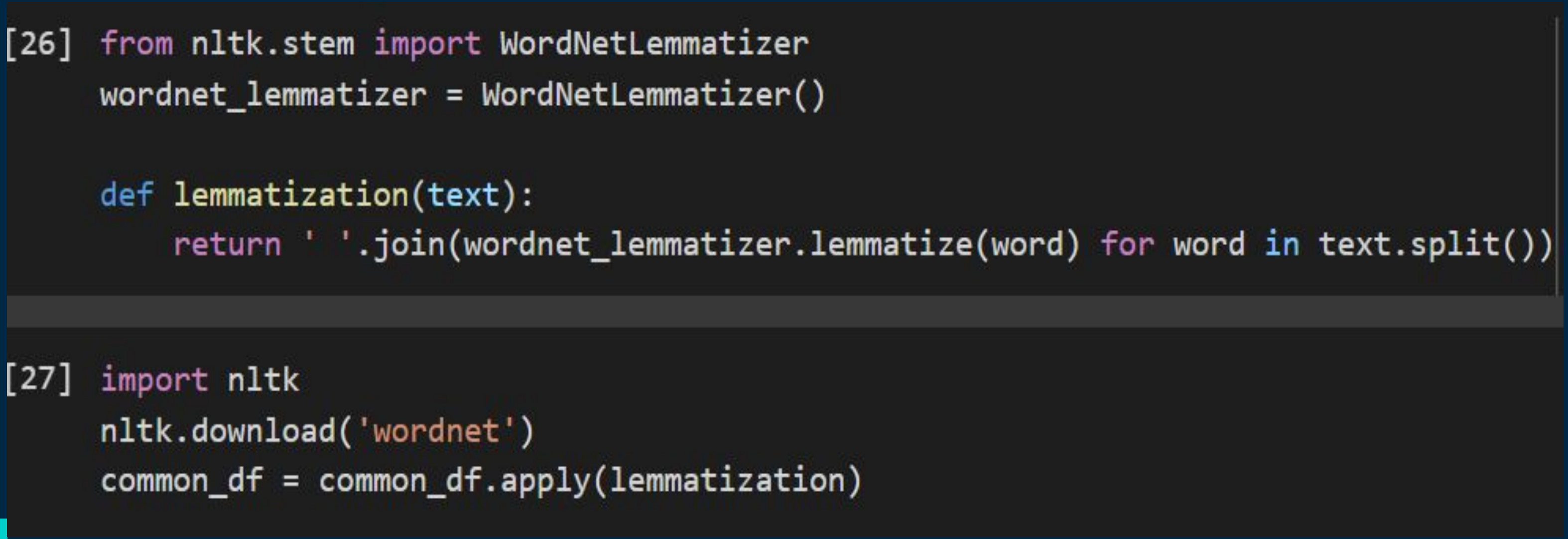
to lower(): Every text will be converted to lowercase.
remove_url(): All URLs that were present in the text were deleted.
remove quotes(): Eliminates any quotes from the content.
non_alphabetic(): Eliminate any non-alphabetic characters from the text.
remove_punctuation(): Removes all punctuation from the text.
remove_stopwords(): Removes any stopwords that were present in the text.



```
[22] def process_text(text):
         text = to_lower(text)
         text = remove_url(text)
         #text = remove_spoiler_alert(text)
         text = remove_quotes(text)
         text = remove_non_alphabetic(text)
         text = remove_punctuation(text)
         text = remove_stopwords(text)
         #text = remove_small_words(text)

         # take only first MAX_LEN words
         text = ' '.join(text.split()[:MAX_LEN])
         return text
```

DATA PREPROCESSING

Lemmatization: Reducing a word to its simplest or dictionary form is a process called lemmatization.The WordNetLemmatizer is used by the lemmatization function to apply the lemmatization function to each word after splitting the input text into its component words.

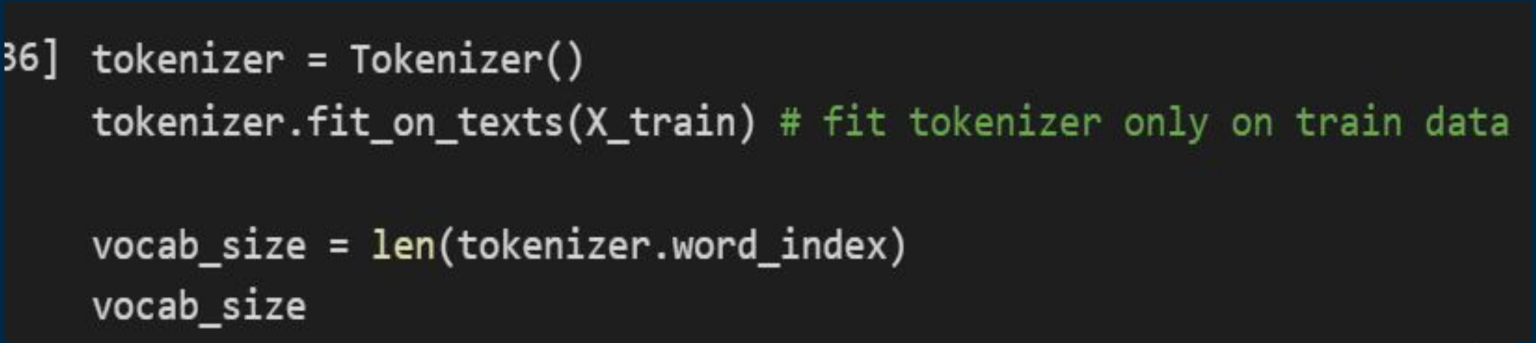


```
[26] from nltk.stem import WordNetLemmatizer
     wordnet_lemmatizer = WordNetLemmatizer()

     def lemmatization(text):
         return ' '.join(wordnet_lemmatizer.lemmatize(word) for word in text.split())
```

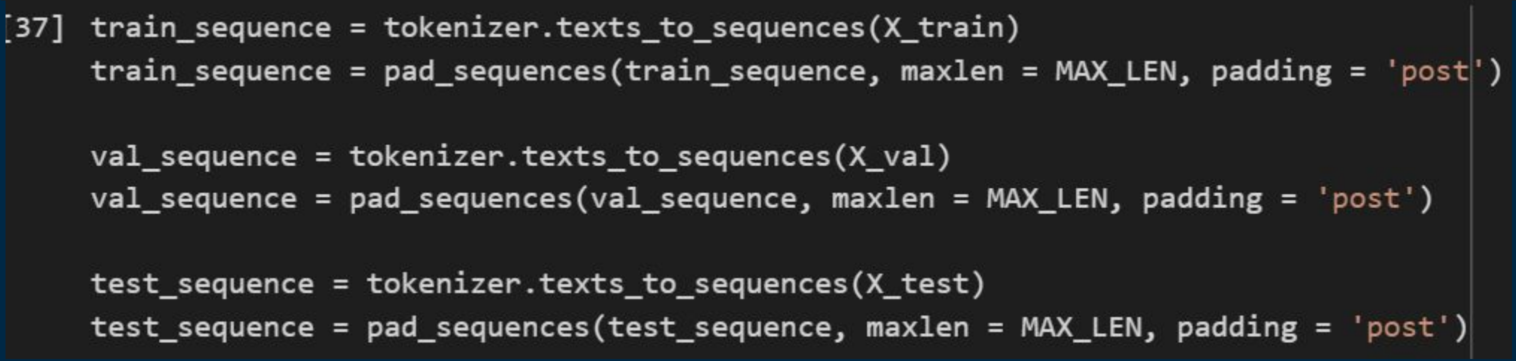
```
[27] import nltk
     nltk.download('wordnet')
     common_df = common_df.apply(lemmatization)
```

DATA PREPROCESSING

Tokenization: The Keras Tokenizer class is a strong text preparation tool that turns word sequences (text) into numerical sequences that may be used to train neural networks. It helps in drawing out relevant information from unstructured text data.



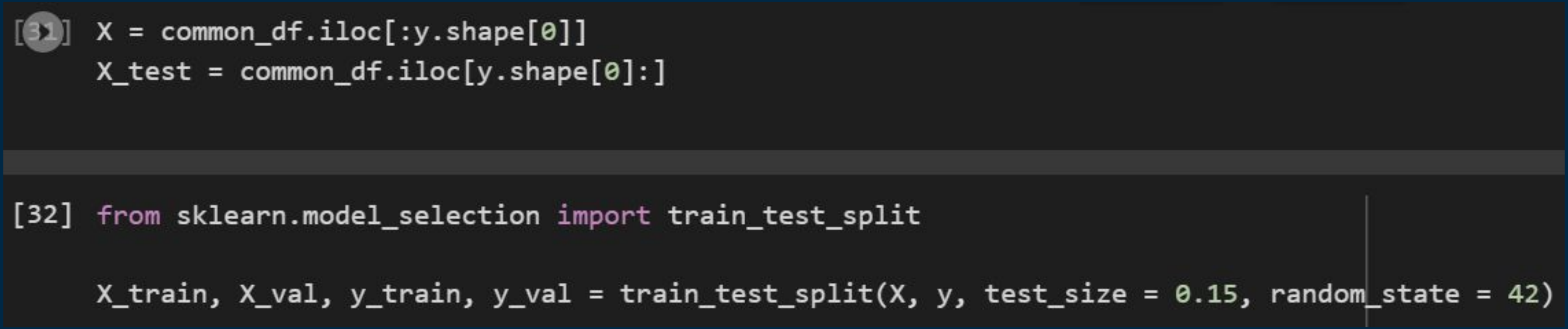
```
36]  tokenizer = Tokenizer()
     tokenizer.fit_on_texts(X_train) # fit tokenizer only on train data

     vocab_size = len(tokenizer.word_index)
     vocab_size
```



```
[37]  train_sequence = tokenizer.texts_to_sequences(X_train)
      train_sequence = pad_sequences(train_sequence, maxlen = MAX_LEN, padding = 'post')

      val_sequence = tokenizer.texts_to_sequences(X_val)
      val_sequence = pad_sequences(val_sequence, maxlen = MAX_LEN, padding = 'post')

      test_sequence = tokenizer.texts_to_sequences(X_test)
      test_sequence = pad_sequences(test_sequence, maxlen = MAX_LEN, padding = 'post')
```

SPLITTING DATASET

The data is split into training and validation sets using the train test split function from the sklearn.model selection package.
We may use the validation set to fine-tune the machine learning model's hyperparameters and assess how well it performs on new data by dividing the data into training and validation sets.



```
[31] X = common_df.iloc[:y.shape[0]]
     X_test = common_df.iloc[y.shape[0]:]
```

```
[32] from sklearn.model_selection import train_test_split

     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.15, random_state = 42)
```

COMPARISION OF MODELS

- LSTM : Long Short-Term Memory, often known as LSTM, is a form of recurrent neural network (RNN) architecture created to solve the vanishing gradient issue that affects conventional RNNs. An LSTM network's basic component is a memory cell with long-term information storage capabilities. LSTMs are especially helpful for processing sequential data, such as text and speech, where the context and dependencies between the sequence's components are crucial for deciphering the meaning of the input.

- SIMPLE RNN: A simple RNN is a kind of neural network made to handle sequential data by retaining a hidden state that records details of the sequences that have already been processed. The current input element and the previous hidden state are fed into the network as inputs at each time step, and it outputs a new hidden state along with an output. They can be used for text classification and sentiment analysis tasks and are useful for processing short sequences.

- BIDIRECTIONAL LSTM: The bidirectional LSTM employs two independent LSTMs, one of which processes the input sequence in a forward manner and the other in a backward way. The output from each time step is then concatenated to include the dependencies in both directions. The bidirectional LSTM captures long-term dependencies in the data more effectively than a simple LSTM by processing the input sequence in both directions.

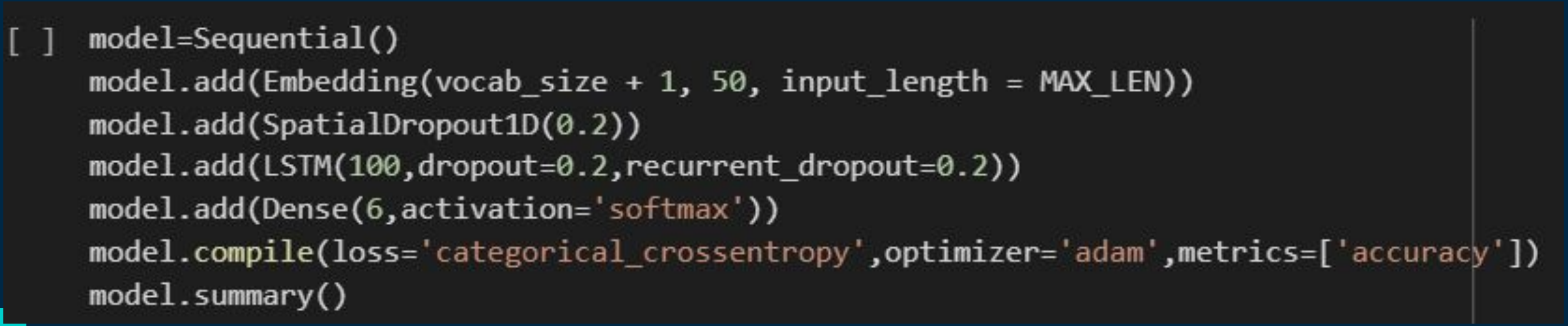
LSTM

- The first layer of the model is an Embedding layer that takes as input the vocab_size and input_length of the sequence as parameters. Each word in the input sequence is given an embedding representation by the layer.

- The SpatialDropout1D layer adds regularisation by randomly removing all of the 1D feature maps from the input during training with a probability of 0.2.

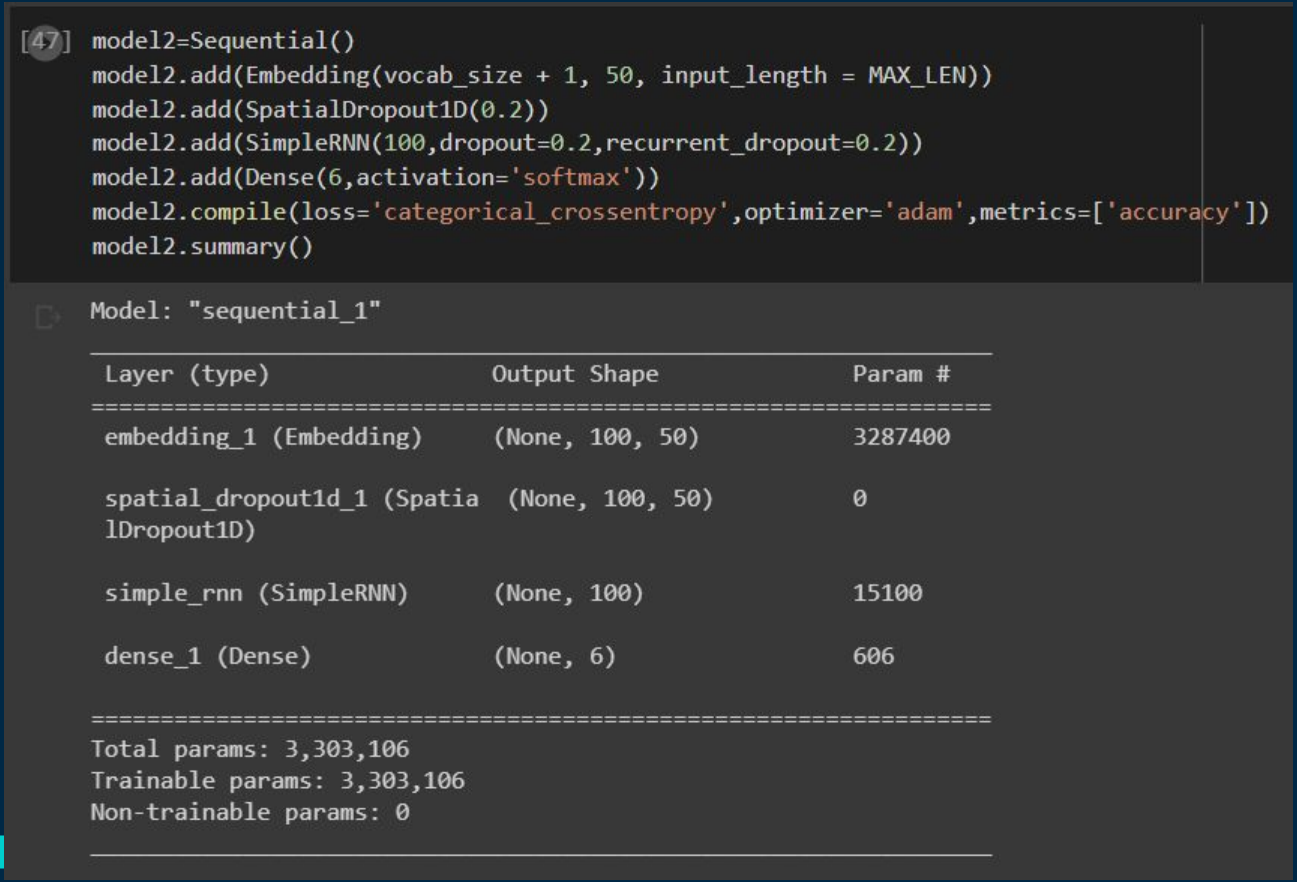
- The LSTM layer, a recurrent layer with 100 units, applies the LSTM operation to the series of embeddings discovered by the preceding layer.

- The softmax activation function is used in the sixth and final dense layer, which has six units (one for each class), to create a probability distribution over the classes.



```
[ ]  model=Sequential()
     model.add(Embedding(vocab_size + 1, 50, input_length = MAX_LEN))
     model.add(SpatialDropout1D(0.2))
     model.add(LSTM(100,dropout=0.2,recurrent_dropout=0.2))
     model.add(Dense(6,activation='softmax'))
     model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
     model.summary()
```

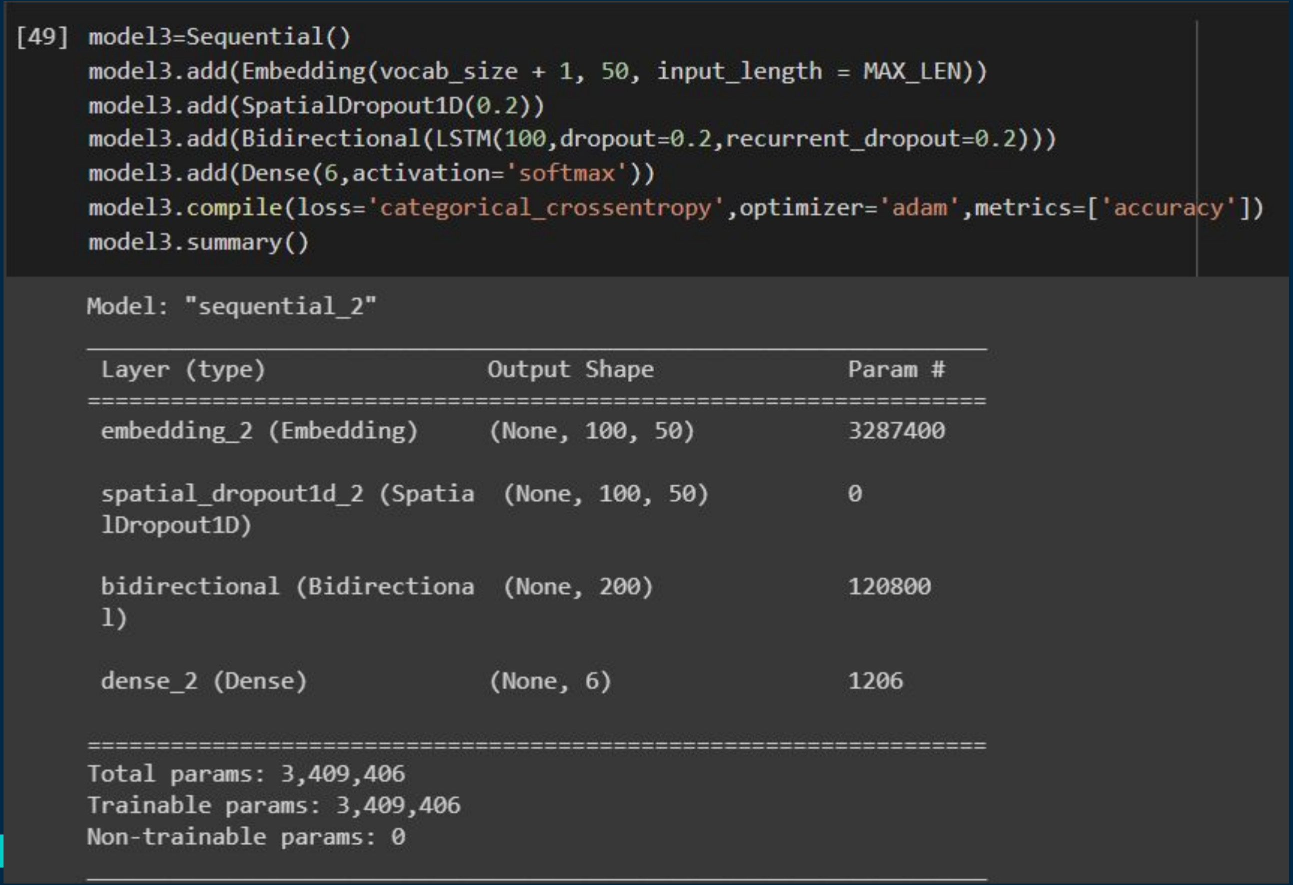
SIMPLE RNN



```
[47] model2=Sequential()
     model2.add(Embedding(vocab_size + 1, 50, input_length = MAX_LEN))
     model2.add(SpatialDropout1D(0.2))
     model2.add(SimpleRNN(100,dropout=0.2,recurrent_dropout=0.2))
     model2.add(Dense(6,activation='softmax'))
     model2.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
     model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 50)           3287400

 spatial_dropout1d_1 (Spatia  (None, 100, 50)          0
 lDropout1D)

 simple_rnn (SimpleRNN)      (None, 100)               15100

 dense_1 (Dense)             (None, 6)                 606

=================================================================
Total params: 3,303,106
Trainable params: 3,303,106
Non-trainable params: 0
_________________________________________________________________
```

BIDIRECTIONAL LSTM

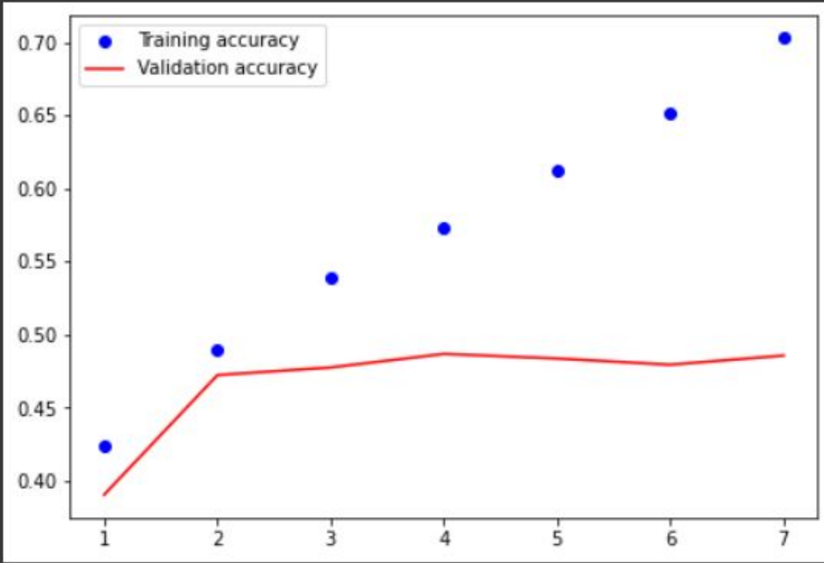


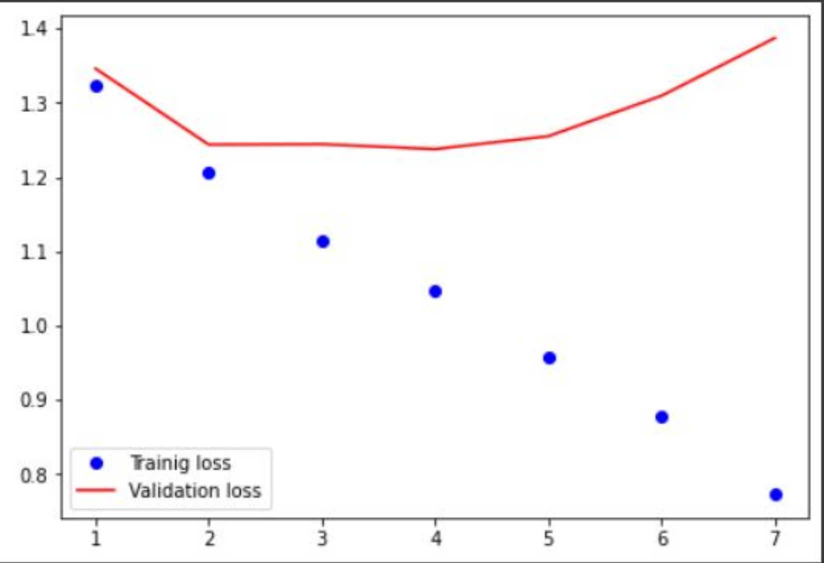
```
[49] model3=Sequential()
     model3.add(Embedding(vocab_size + 1, 50, input_length = MAX_LEN))
     model3.add(SpatialDropout1D(0.2))
     model3.add(Bidirectional(LSTM(100,dropout=0.2,recurrent_dropout=0.2)))
     model3.add(Dense(6,activation='softmax'))
     model3.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
     model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 50)           3287400

 spatial_dropout1d_2 (Spatia  (None, 100, 50)          0
 lDropout1D)

 bidirectional (Bidirectiona  (None, 200)              120800
 l)

 dense_2 (Dense)             (None, 6)                 1206

=================================================================
Total params: 3,409,406
Trainable params: 3,409,406
Non-trainable params: 0
_________________________________________________________________
```

RESULTS(LSTM)

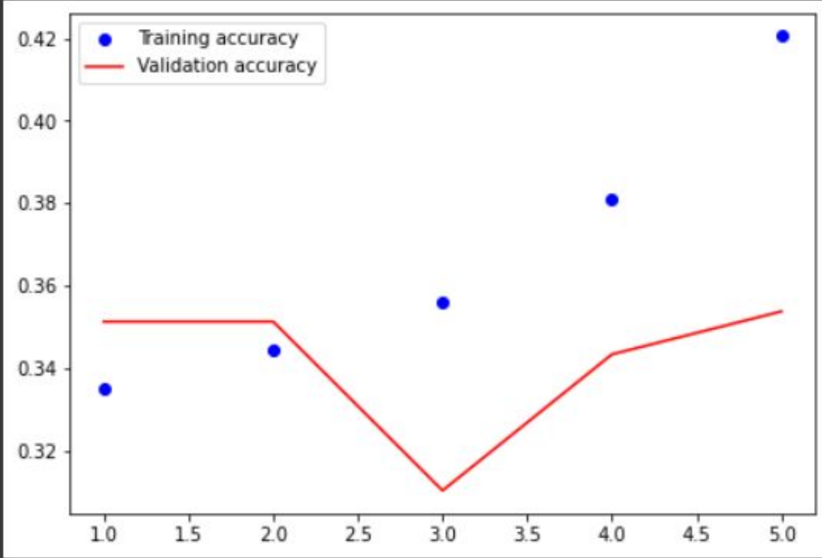
In LSTM the training accuracy has increased linearly with the number of epochs. The validation accuracy had increased upto 0.5 and remained stable after 2 epochs. The training loss is gradually decreasing with increasing epochs but the validation loss which was initially decreasing began to rise after 5 epochs.

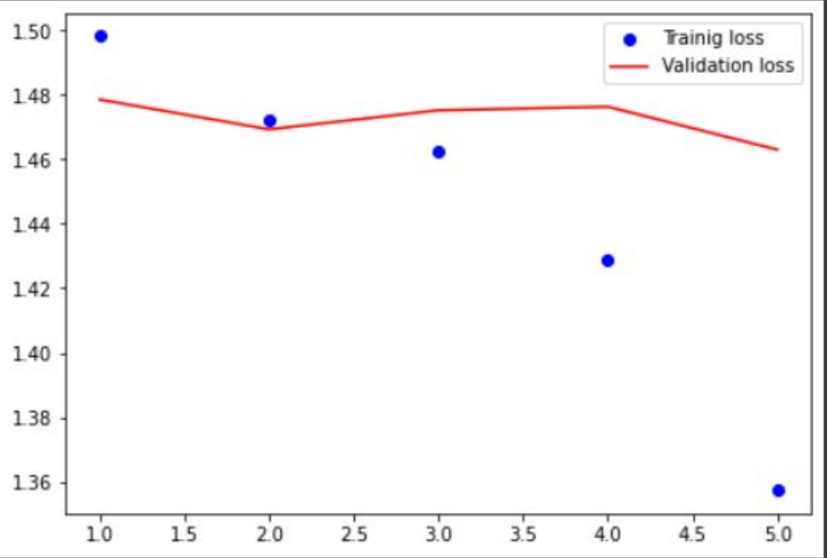




RESULTS(SIMPLE RNN)

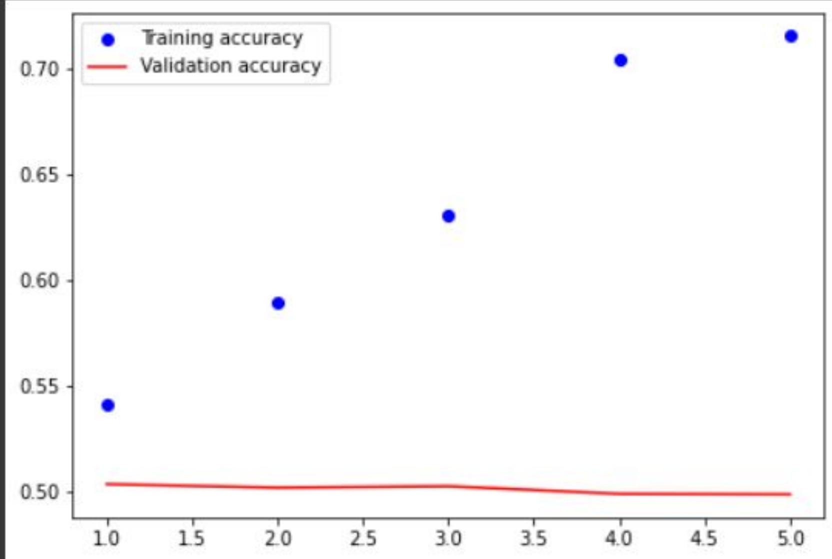
In Simple RNN, the accuracy was much lesser when compared to LSTM, but with increasing epochs, even the training accuracy started to increase while the validation accuracy is fluctuating. The training loss also significantly reduced with the increasing epochs but the validation loss remained stable at 1.48.

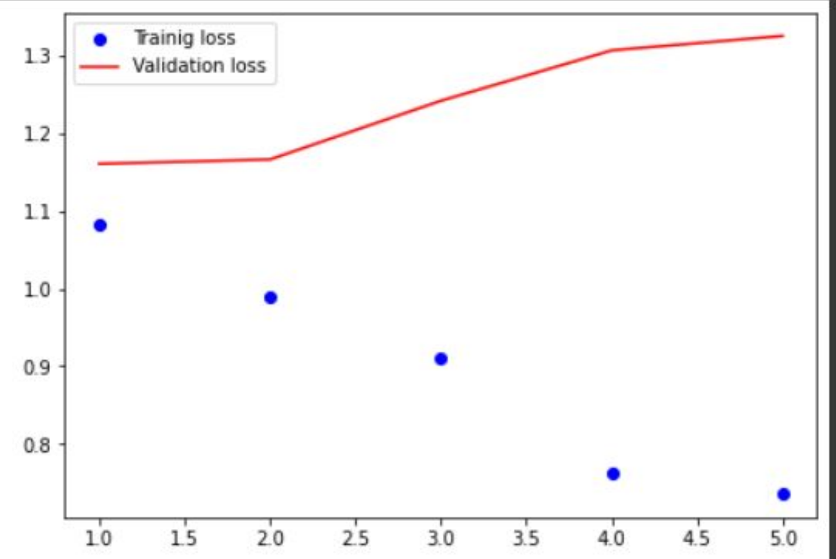




RESULTS(BIDIRECTIONAL RNN)

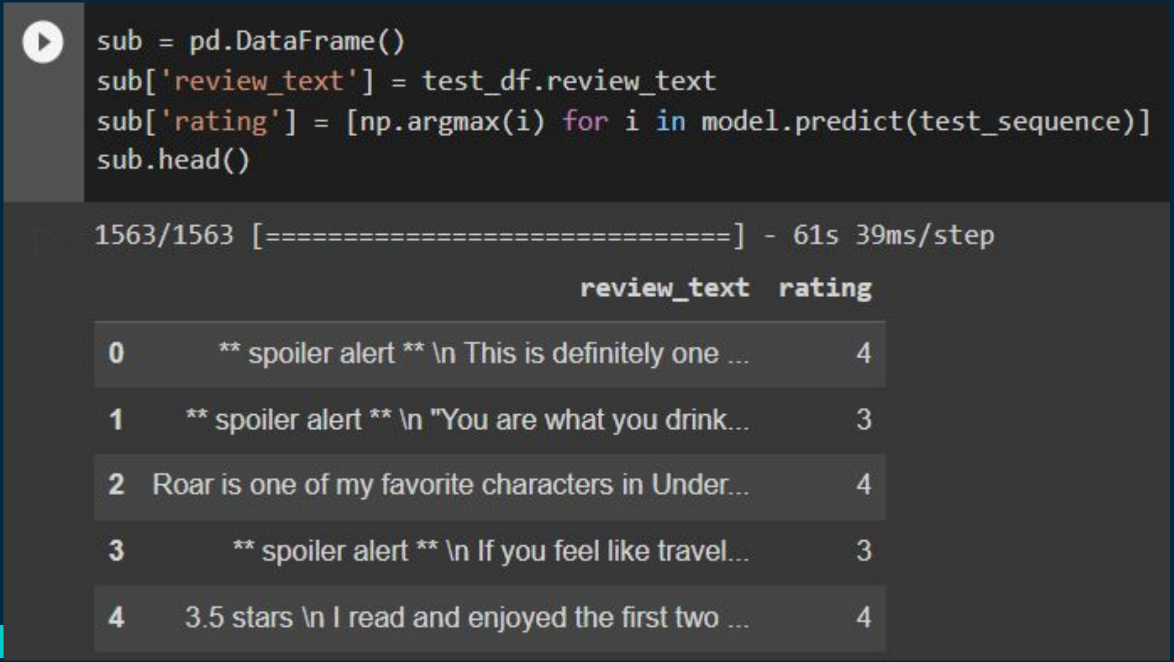
In Bidirectional RNN, the training accuracy peaked at 0.7 at the end of the 5th epoch while the validation accuracy remained constant at 0.5. Even the training loss was lesser when compared to LSTM and the validation loss increased after 2 epochs upto 1.3. It can be concluded that Bidirectional RNN has the best performance in terms of accuracy and loss when compared with the previous two models.





PREDICTIONS

The predicted ratings are acquired by calling the model object's predict method on the test sequence input data. The rating column holds the estimated ratings for each review, while the review text column has the review text from the test df dataFrame.

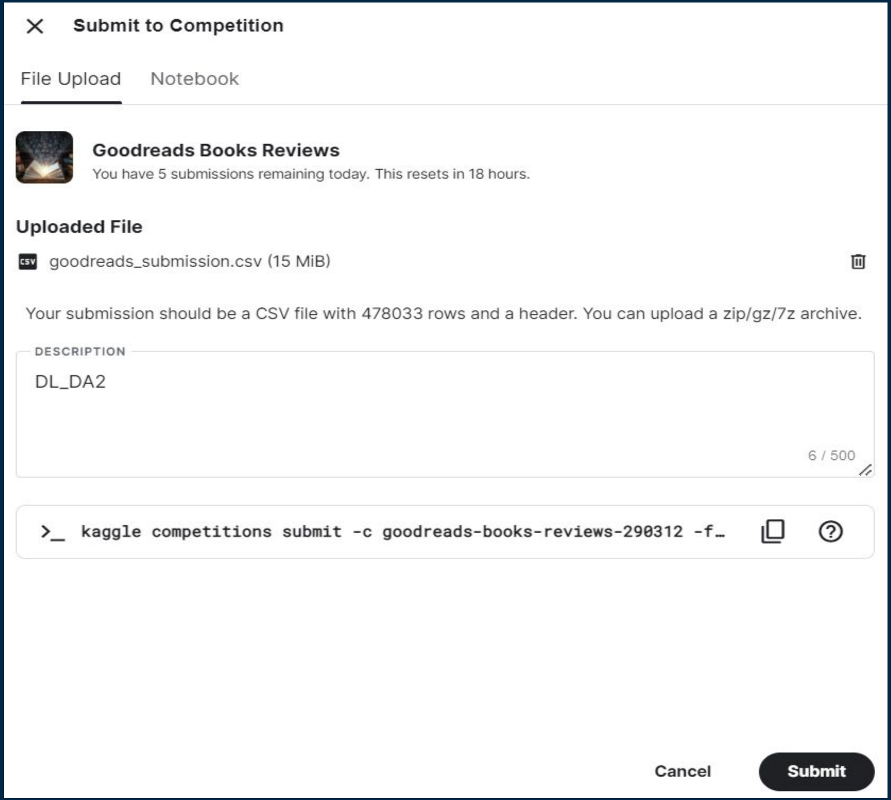


```
sub = pd.DataFrame()
sub['review_text'] = test_df.review_text
sub['rating'] = [np.argmax(i) for i in model.predict(test_sequence)]
sub.head()
```

```
1563/1563 [==============================] - 61s 39ms/step
```

	review_text	rating
0	** spoiler alert ** \n This is definitely one ...	4
1	** spoiler alert ** \n "You are what you drink...	3
2	Roar is one of my favorite characters in Under...	4
3	** spoiler alert ** \n If you feel like travel...	3
4	3.5 stars \n I read and enjoyed the first two ...	4

LEADERBOARD



Submit to Competition

File Upload Notebook

Goodreads Books Reviews

You have 5 submissions remaining today. This resets in 18 hours.

Uploaded File

goodreads_submission.csv (15 MiB)

Your submission should be a CSV file with 478033 rows and a header. You can upload a zip/gz/7z archive.

DESCRIPTION

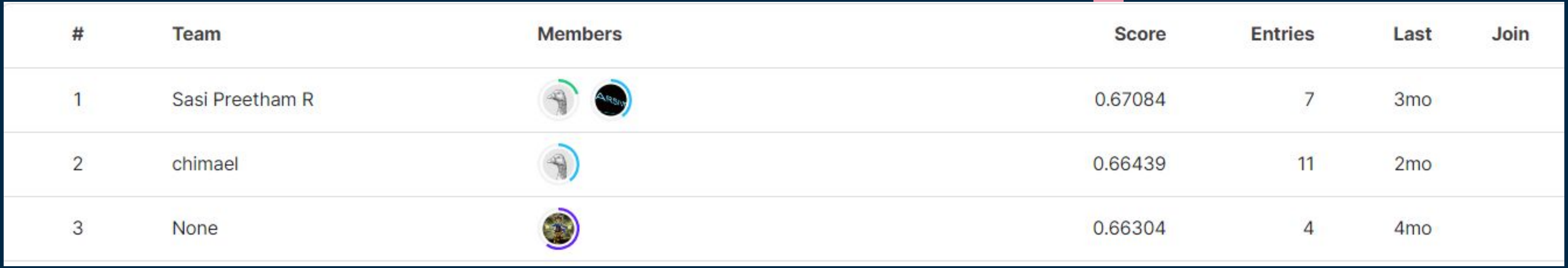
DL_DA2

6 / 500

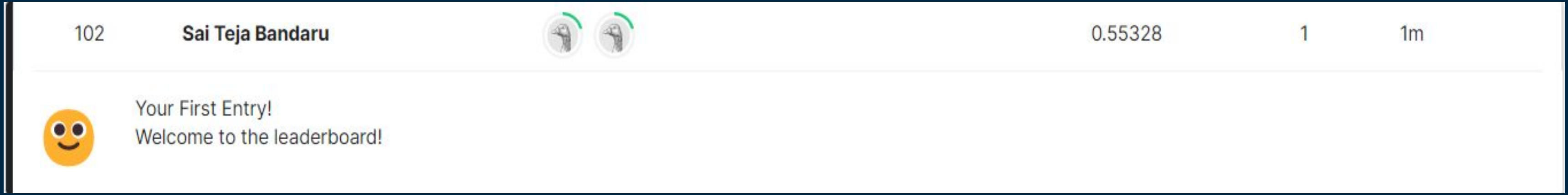
```
>_ kaggle competitions submit -c goodreads-books-reviews-290312 -f…
```

Cancel Submit

LEADERBOARD



#	Team	Members	Score	Entries	Last	Join
1	Sasi Preetham R		0.67084	7	3mo	
2	chimael		0.66439	11	2mo	
3	None		0.66304	4	4mo	



102	**Sai Teja Bandaru**		0.55328	1	1m

Your First Entry!
Welcome to the leaderboard!

THANK YOU