UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning

(6CS012)

Text Classification Task

Twitter Sentimental Analysis

Student Id	: 2227486
Student Name	: Nayan Raj Khanal
Group	: L5CG4
Tutor	: Ms. Sunita Parajuli
Lecturer	: Mr. Siman Giri
Date	: 12th May, 2024

Abstract

The report dives into the development of a Text Classification Model using the 'Twitter Sentiment Analysis Dataset'. It contains tweets from various sources such as Microsoft, Amazon and others. Alongside the source it contains sentiments associated with it labelled as Positive, Negative, Neutral and Irrelevant. The aim of the task is to build a model capable of accurately labelling textual data into Positive or Negative sentiments. The report discusses the methodology involved in preprocessing the textual data in the form of data cleaning and preparation. After preprocessing the report discusses the construction of a deep sequential model architecture using recurrent neural networks (RNNs) for text classification with embedding layer, dropout layer, bidirectional LSTM layer, dense layers with ReLU activation and dense output layer with softmax activation layer. The model's hyperparameters and optimization techniques are also discussed. Furthermore, the training process, evaluation metrics and model performance analysis which includes the interpretation of train-validation loss curves, model accuracy curves, classification reports and confusion matrices are also discussed. The findings show promising results. Finally, the report highlights areas for further research and improvement and encourages others to explore and develop in the field of natural language processing.

Table of Contents

Table of Figures

1 Introduction

The dataset of choice is “Twitter Sentiment Analysis Dataset”. It has textual data in the form of tweets of users from various video game sources, Microsoft, Amazon etc and sentiment of those tweets which are either Positive, Negative, Neutral and Irrelevant. By a tweet of an user, the text classification task is to judge the sentiment of the message. (NLP, 2021)

The main aim of the task is to develop a text classification model that can accurately classify textual data into their respective categories in our case labels containing Positive or Negative. The objectives of the task are as follows:

- Preprocessing of the textual data to make it suitable for model training.
- Implementing a RNN based deep sequential model architecture using deep learning techniques for text classification.
- Train the model for set number of epochs and plot training loss/validation loss for each iterations.
- Evaluating the model's performance and analyzing the results.

The RNN based deep sequential model architecture used in this project consists of several key components:

- Embedding Layer
- Dropout Layer
- Bidirectional LSTM Layer
- Dense Layer with ReLU Activation
- Dense Output Layer with Softmax Activation

For the model compilation we utilized learning rate, Adam algorithm for optimization, Categorical_crossentropy for loss function and accuracy as the metrics. Then trained the model and plot graphs for each epochs.

The report contains introduction providing overview of the task and it’s aims and objective. Followed by methodology containing data cleaning process, model architecture, training of the model, different hyper - parameters used to train the model and evaluation. Finally concluded by summary of results and discussion of the findings.

2 Methodology

Before training the model it is crucial to make sure the data we're going to feed it is clean and is suitable for training.

2.1 Data Pre - Processing:

The 'Twitter Sentiment Analysis Dataset' had four columns but had no headers so firstly we add them for readability. After adding headers, we have four columns named, 'Header1', 'Company', 'Label', and 'Text'. Since we only need 'Label', and 'Text' we therefore drop the remaining two from the dataframe.

After adding and keeping relevant columns we move onto data cleaning. Here, we first check the missing values for all the attributes and then drop the missing values for all the attributes from the original dataframe if found any. Secondly, we check the number of duplicate values in the dataframe and likewise drop the duplicate values from the original dataframe if found any.

Then we create functions for removing:

1. URLs: In this step we will remove URLs and replace it with space.
2. Emojis: In this step we will remove emojis and replace it with space.
3. Unwanted Characters: In this step we will remove user mentions, hashtags, punctuations, double spaces.

After removing we then create functions for:

1. Tokenization: Here we will break down text or corpus into tokens. Example:
   Hi I am Nayan => [' Hi ', ' I ', ' am ', ' Nayan ']
2. Remove Punctuations: Here we will remove punctuations from current text data.
3. Remove StopWord: Here we will remove connecting parts/words of a sentence. Example: Words like 'the' or 'and'.
4. Lemmatization: Here we will try to reduce the number of tokens by replacing the word with its root word. Example: Playing, Played, Plays => Play.
5. Stemming: Here we will chop of the word at its tail to reduce token. Example: change, changing, changed => chang

6. Lower Order: In this we convert tokens to lowercase. Since we got the output in the form of list, we must map the lower_order function with each tokens and return it in list.

After that we create a function ‘Input Text Pipeline’ where we will compile every basic cleaning steps done above in one functions then tokenize and implement with our dataset and put the results in a new attribute ‘Cleaned_Texts’.

We then:

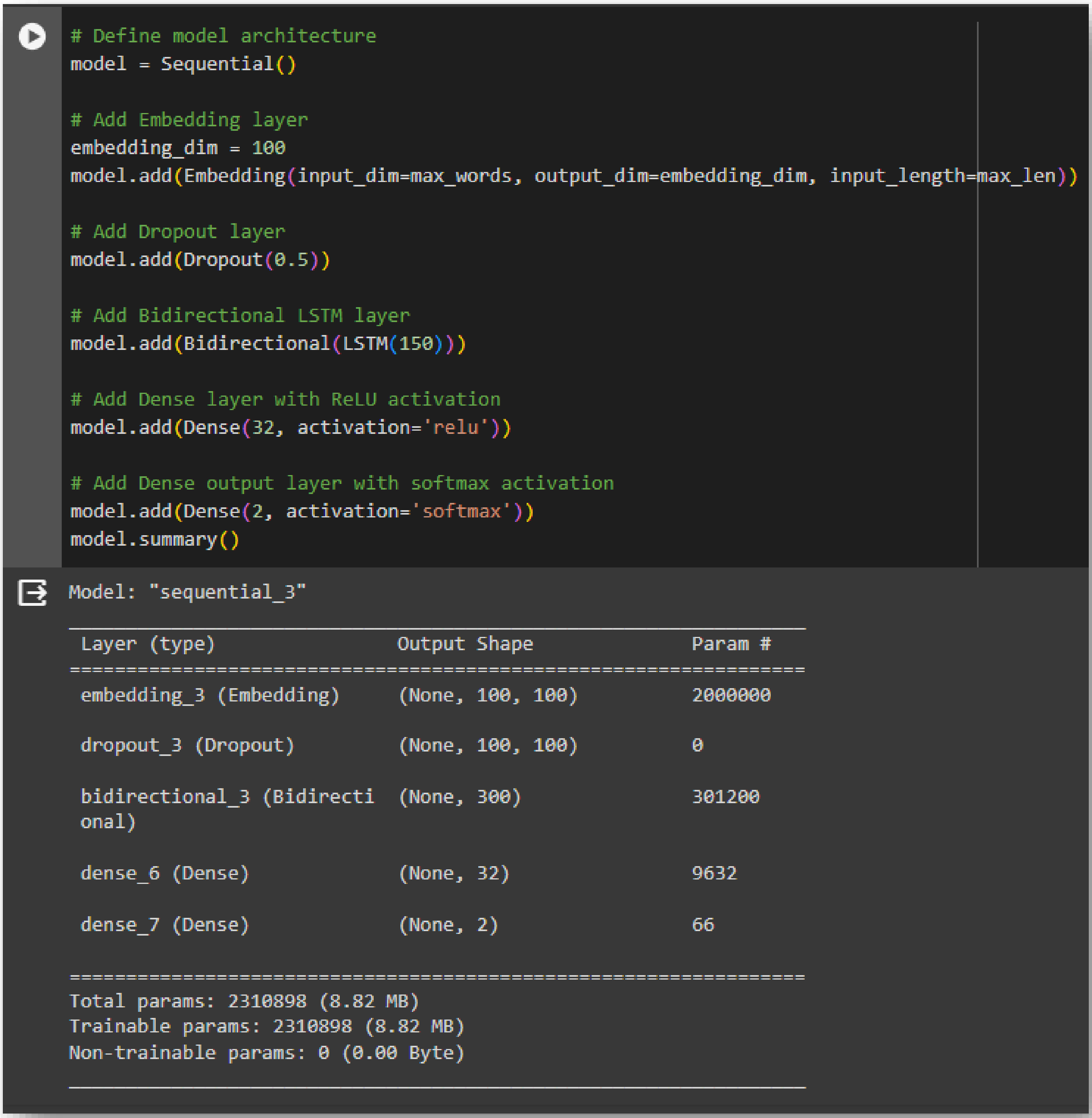
1. Use WordCount which plots of most frequent to less frequent word in the attribute 'Cleaned_Text'.
2. Create a copy of our dataframe using .copy() and then remove labels which are "Neutral" and "Irrelevant" from the dataframe as the goal is to only differentiate Positive and Negative tweets.
3. Assign the values of 'Cleaned_Text' and ‘Labels’ to the variable 'reviews' and ‘labels’ and store it as Numpy array respectively.
4. Use Label Encoder to convert categorical labels into numerical format and view the numerical labels assigned by the LabelEncoder to the classes. In our case:
   Classes: ['Negative' 'Positive']
   Numerical Labels: [0 1]
5. Convert our text data into sequence of integers to feed to our neural network by creating an instance of Tokenizer().
6. Use padding to make sure that every sequence are of equal length using pad_sequence() which ensures that all sequences have the same length by either adding padding or truncating them at the end of sequences.
7. Use One-Hot Encoding to represent categorical labels as binary vectors. In our case:

   For "negative": The one-hot encoding would be [1, 0].

   For "positive": The one-hot encoding would be [0, 1].
8. Use the train_test_split function to divide the dataset into training and testing sets in the ratio of 80:20, allowing the model to be trained on one portion of the data and evaluated on remaining for assessment.

2.2 Model Building:

For the text classification task we build an architecture which used RNN-based LSTM layers along with dense layers to build a deep sequential model. The Sequential model is a linear stack of layers and it consists of five layers:



```
# Define model architecture
model = Sequential()

# Add Embedding layer
embedding_dim = 100
model.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_len))

# Add Dropout layer
model.add(Dropout(0.5))

# Add Bidirectional LSTM layer
model.add(Bidirectional(LSTM(150)))

# Add Dense layer with ReLU activation
model.add(Dense(32, activation='relu'))

# Add Dense output layer with softmax activation
model.add(Dense(2, activation='softmax'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 100, 100)          2000000

 dropout_3 (Dropout)         (None, 100, 100)          0

 bidirectional_3 (Bidirecti  (None, 300)               301200
 onal)

 dense_6 (Dense)             (None, 32)                9632

 dense_7 (Dense)             (None, 2)                 66

=================================================================
Total params: 2310898 (8.82 MB)
Trainable params: 2310898 (8.82 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 1. Model Summary

- Embedding Layer: This layer converts input integer sequences (words/tokens) into dense vectors of fixed size. In our case the tokens from ‘Cleaned_Text’ attribute is passed through this layer and they are mapped to a corresponding vector representation which captures semantic similarities between tokens. (Carremans, 2018)
- Dropout Layer: This layer randomly drops a fraction of input units during training, helping prevent overfitting. It is a regularization technique that helps in improving model’s performance. (Yadav, 2022)
- Bidirectional LSTM Layer: This layer consists of two LSTM layers processing the input integer sequences (words/tokens) in both forward and backward directions, hence it can capture context from both past and future tokens, enhancing the model’s understanding of the text. (Nama, 2023)
- Dense Layer (ReLU): This dense uses the ReLU activation function to add non-linearities to the model. ReLU sets all negative values to zero and positives values unchanged which can speed up convergence during training and remove the vanishing gradient problem. (Verma, 2024)
- Dense Output Layer (Softmax): This layer acts as the classifier, producing the final output prediction. It contains two units, representing the two classes (positive, negative sentiment) in the classification task. It converts output scores into probabilities, ensuring that the output ranges between 0 and 1 and sum up to 1. (Sarfraz, et al., 2022)

The model utilizes hyper-parameters such as embedding dimension, LSTM units, dropout rate and learning rate to define the architecture's behavior and complexity. These are set during model initialization and optimization and directly influence model performance.

- Embedding Dimension: This parameter determines the size of the dense embedding vectors for each word. The higher the value of embedding dimension the higher the computational complexity of the model. In our case we set it to 100 (output_dim=100). (Carremans, 2018)
- LSTM Units: The number of LSTM units specifies the dimensionality of the LSTM's output space. The more LSTM units the more complex patterns it learns but may lead to overfitting. In our case we set it to 150 LSTM units (Bidirectional(LSTM(150))). (Nama, 2023)
- Dropout Rate: Dropout rate controls the fraction of neurons to drop during training. In our case we set dropout rate to 0.5, meaning 50% of the input units will be randomly set to 0 during training to reduce overfitting (model.add(Dropout(0.5))). (Yadav, 2022)
- Learning Rate: It determines the size at which the model weights are updated during backpropagation. Learning Rate needs careful tuning for optimal model performance as it impacts training process, convergence speed, model stability and final performance. In our case we set it to 0.0001 (learning_rate = 0.0001). (Khan, 2023)

From the model summary we can note that, for:

- Embedding layer the output shape has (None, 100, 100). Here, ‘None’ represents the batch size, ‘100’ represents maximum sequence length and ‘100’ represents the dimensionality of the embedding vectors. The parameter ‘2000000’ represents the total number of parameters in the Embedding layer.
- Dropout layer as it does not alter the shape so it remains the same and since it doesn’t have any trainable parameters so the number is ‘0’.
- Bidirectional LSTM layer the output shape has (None, 300). Here, ‘None’ represents the batch size, and ‘300’ represents the dimensionality of the output. The parameter ‘301200 represents the total number of parameters in the Embedding layer.
- Dense Layer (ReLU) layer the output shape has (None, 32). Here, ‘None’ represents the batch size, and ‘32 represents number of units in the dense layer. The parameter ‘9632 represents the total number of parameters in the Embedding layer.
- Dense Layer (Softmax) layer the output shape has (None, 32). Here, ‘None’ represents the batch size, and ‘2 represents number of classes. The parameter ‘66 represents the total number of parameters in the Embedding layer.

At last the Total params indicate the total number of trainable parameters in the model, the Trainable params specify the number of parameters that are updated during training through backpropagation and Non-trainable params refers to the number of parameters in the model that are not updated during training.

For the training of the model we utilized:

- Loss Function: The model utilizes 'categorical crossentropy' loss function which calculates the loss by measuring the difference between predicted and actual labels. (Team, 2023)
- Optimizer: To update the model's weights the Adam optimizer is utilized. It dynamically adjusts the learning rate for efficient training. (Vishwakarma, 2024)
- Epochs: Training takes place over several epochs, one epoch means that the model has completed one full round through the entire training dataset. Each epoch refines the model parameters, enhancing its performance. (Baeldung, 2024)
- Train Validation Loss Curve: This curve showcases the training and validation loss over epochs, providing information about the model's learning progress and identifying potential issues like overfitting or underfitting.

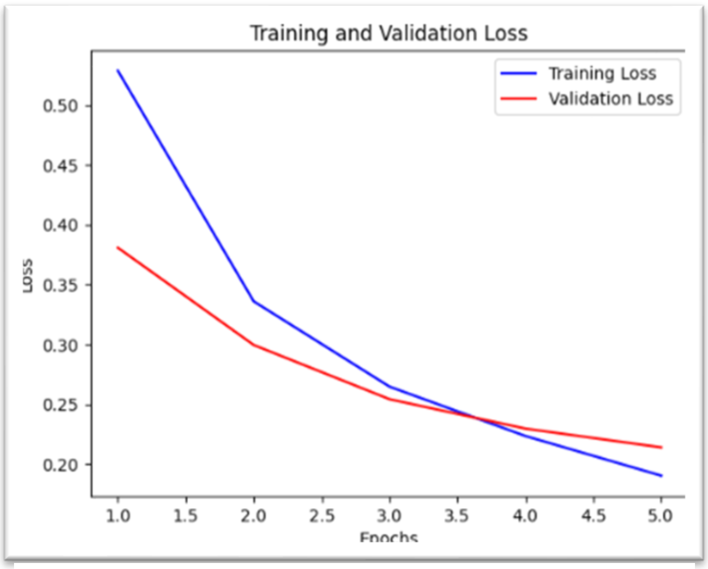
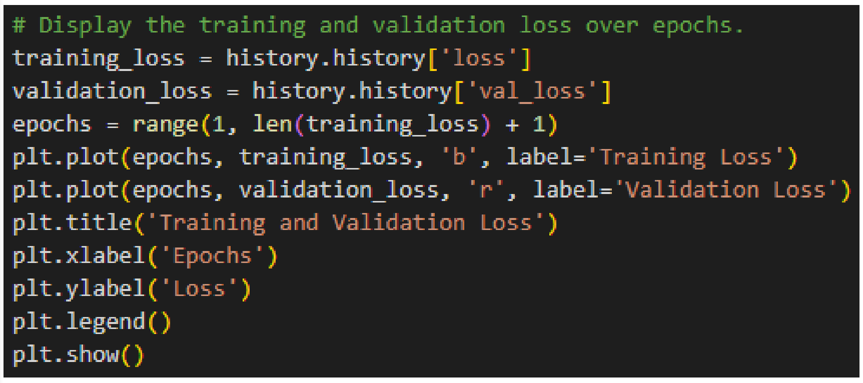


Figure 3. Training VS Validation



```
# Display the training and validation loss over epochs.
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
epochs = range(1, len(training_loss) + 1)
plt.plot(epochs, training_loss, 'b', label='Training Loss')
plt.plot(epochs, validation_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Figure 2. Training VS Validation (Code Block)

In our case we can see that over time(epochs) the losses are decreasing which suggests that the model is effectively learning from the data. Overtime training loss decreases which indicates that the model is fitting the training data better and the overtime validation loss decrease indicates that the model is performing well to unseen data. The smaller the gap between training and validation loss curve, it indicates that there's no significant overfitting.

- Model Accuracy: This curve displays the model's accuracy on both the training and validation datasets over epochs, providing information about how well the model is learning from the training data and how well it performs to unseen data.

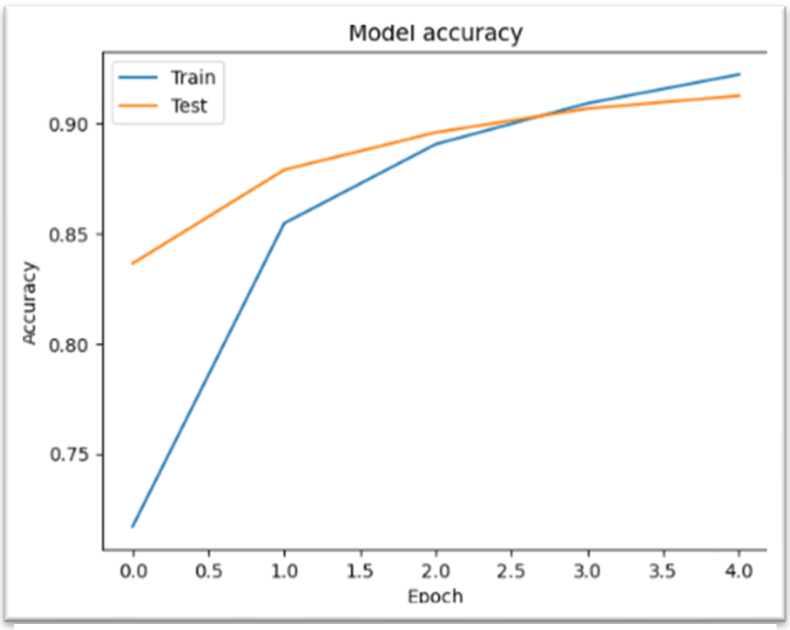
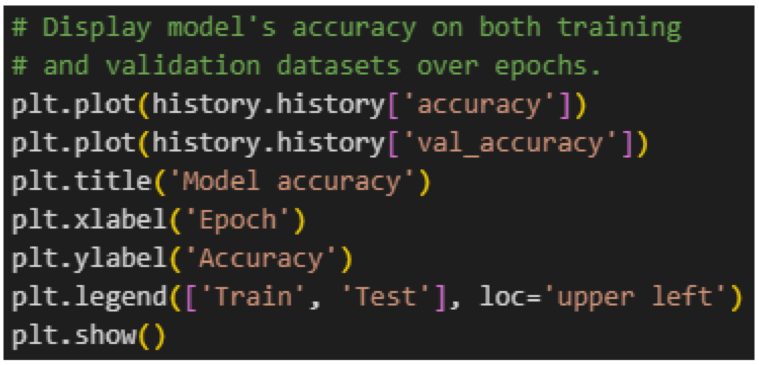


Figure 4. Model Accuracy



```
# Display model's accuracy on both training
# and validation datasets over epochs.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 5.. Model Accuracy (Code Block)

In our case we can see that for the first epoch the model displays lower accuracy on training dataset compared to the validation dataset, suggesting some degree of underfitting. However, overtime though epochs the model's accuracy on both the training and validation datasets improves. At final epoch the model achieves high accuracy on both datasets, meaning the model has learned patterns from the training data and can perform well to unseen validation data demonstrating good overall performance.

For evaluation we carried out:

- Loss/Accuracy: The loss or the difference between predicted output and true target values came out to be 0.2137. On the other hand the accuracy or the proportion of correctly classified instances out of the total number of instances came out to be 0.9124.
- Classification Report:

  From Label Encoder: **Negative: 0 and Positive: 1**

  - For class "0" : ”1”, the precision is 0.92 : 0.91, indicating that 92% : 91% of the predictions classified as "0" : “1” were correct.

  - For class "0" : ”1”, the recall is 0.92 : 0.91, indicating that 92% : 91% of the actual “0” : "1" instances were correctly identified by the model.

  - For class "0", the F1-score is 0.92, and for class "1", it is 0.91. (F1 is the harmonic mean of precision and recall)

  - For class "0", there are 4248 instances, and for class "1", there are 3827 instances.

  The overall accuracy of the model across all classes is 0.91, indicating that it correctly classified 91% of the instances in the test data.

- Confusion Matrix:

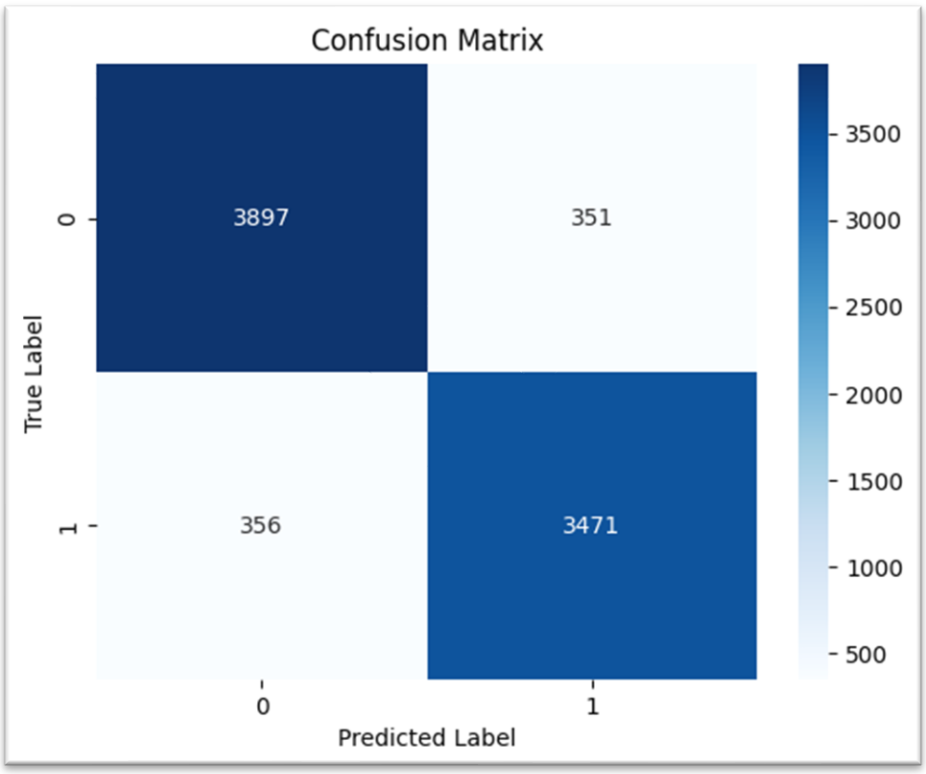
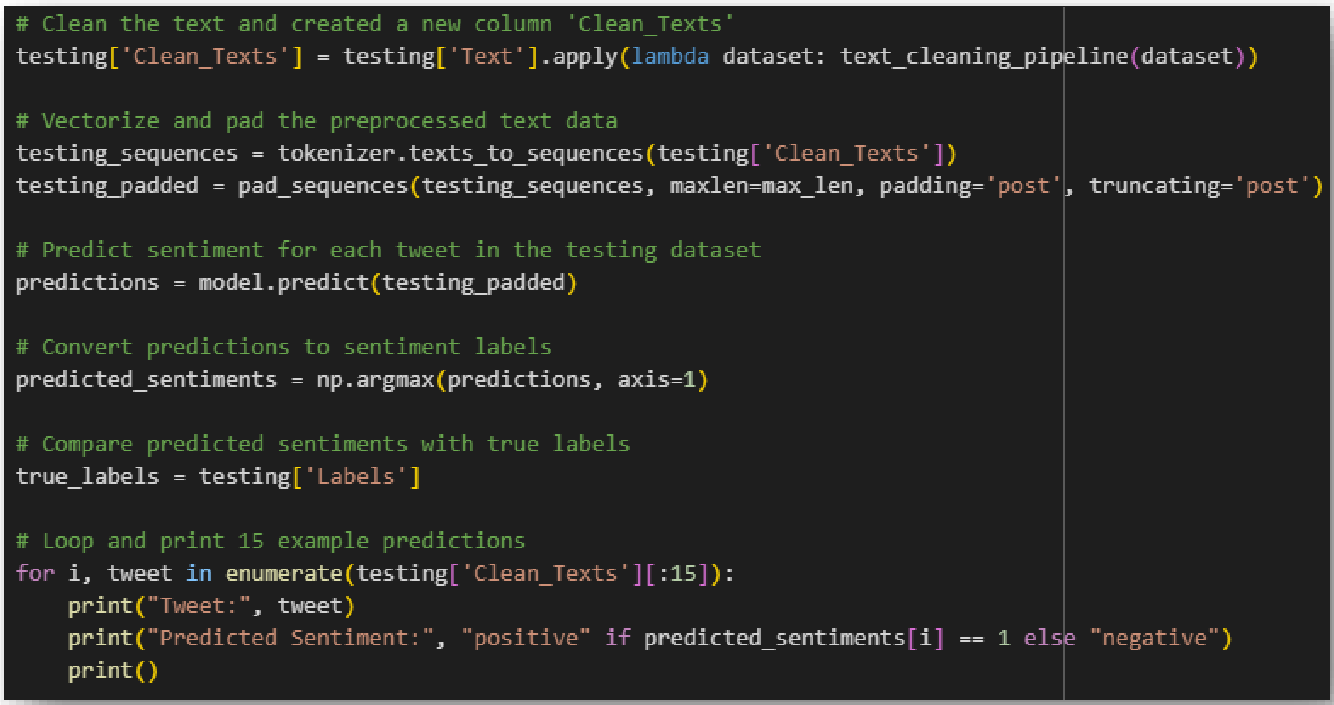


Figure 6. Confusion Matrix

Negative: 0 & Positive: 1 and y: true label & x: predicted label

- True Positives (TP) (1,1): 3471 => predicted positive, actual positive.
- True Negatives (TN) (0,0): 3897 => predicted negative, actual negative.
- False Positives (FP) (1,0): 356 => predicted positive, actual negative.
- False Negatives (FN) (0,1): 351 => predicted negative, actual positive.

For prediction we used our validation dataset.

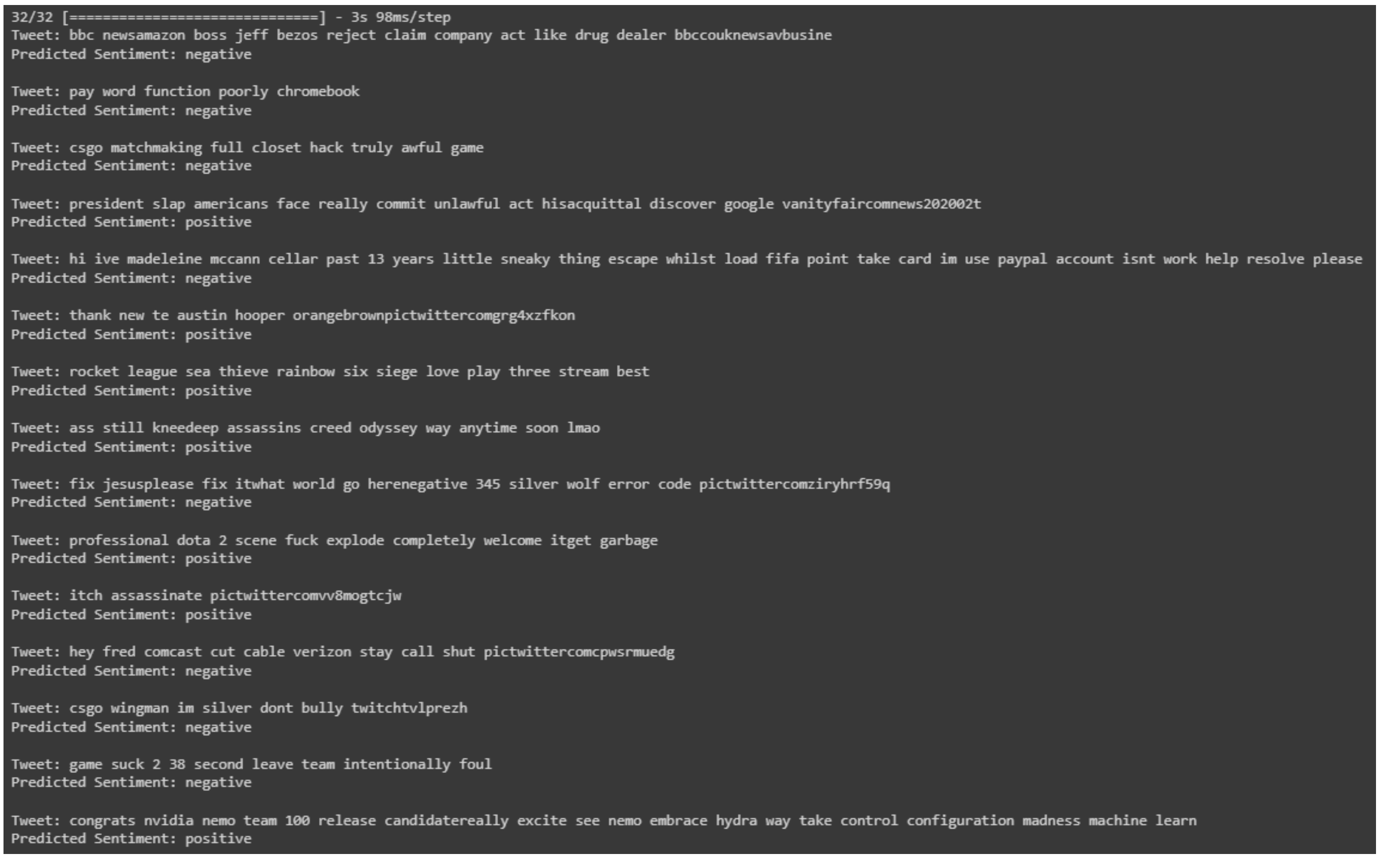


```python
# Clean the text and created a new column 'Clean_Texts'
testing['Clean_Texts'] = testing['Text'].apply(lambda dataset: text_cleaning_pipeline(dataset))

# Vectorize and pad the preprocessed text data
testing_sequences = tokenizer.texts_to_sequences(testing['Clean_Texts'])
testing_padded = pad_sequences(testing_sequences, maxlen=max_len, padding='post', truncating='post')

# Predict sentiment for each tweet in the testing dataset
predictions = model.predict(testing_padded)

# Convert predictions to sentiment labels
predicted_sentiments = np.argmax(predictions, axis=1)

# Compare predicted sentiments with true labels
true_labels = testing['Labels']

# Loop and print 15 example predictions
for i, tweet in enumerate(testing['Clean_Texts'][:15]):
    print("Tweet:", tweet)
    print("Predicted Sentiment:", "positive" if predicted_sentiments[i] == 1 else "negative")
    print()
```

Figure 7. Prediction (Code Block)

In top to bottom sequence we:

- Cleaned the validation data using the 'text_cleaning_pipeline' function and put the clean data in a new attribute 'Clean_Texts' in the dataframe.
- Converted the preprocessed text data in the 'Clean_Texts' column to sequence and added padding.
- Used our model to predict the sentiment for each tweet in the testing dataframe.
- Converted predictions from the model to sentiment labels.
- Compared the output predicted sentiment labels with true labels present inside the column 'Labels'.
- Used for loop to iterate over the first 15 tweets in the 'Clean_Texts' column and print the tweet text and corresponding predicted sentiment label. If the predicted sentiment label is 1, it prints "positive"; otherwise, it prints "negative".



```
32/32 [==============================] - 3s 98ms/step
Tweet: bbc newsamazon boss jeff bezos reject claim company act like drug dealer bbccouknewsavbusine
Predicted Sentiment: negative

Tweet: pay word function poorly chromebook
Predicted Sentiment: negative

Tweet: csgo matchmaking full closet hack truly awful game
Predicted Sentiment: negative

Tweet: president slap americans face really commit unlawful act hisacquittal discover google vanityfaircomnews202002t
Predicted Sentiment: positive

Tweet: hi ive madeleine mccann cellar past 13 years little sneaky thing escape whilst load fifa point take card im use paypal account isnt work help resolve please
Predicted Sentiment: negative

Tweet: thank new te austin hooper orangebrownpictwittercomgrg4xzfkon
Predicted Sentiment: positive

Tweet: rocket league sea thieve rainbow six siege love play three stream best
Predicted Sentiment: positive

Tweet: ass still kneedeep assassins creed odyssey way anytime soon lmao
Predicted Sentiment: positive

Tweet: fix jesusplease fix itwhat world go herenegative 345 silver wolf error code pictwittercomziryhrf59q
Predicted Sentiment: negative

Tweet: professional dota 2 scene fuck explode completely welcome itget garbage
Predicted Sentiment: positive

Tweet: itch assassinate pictwittercomvv8mogtcjw
Predicted Sentiment: positive

Tweet: hey fred comcast cut cable verizon stay call shut pictwittercomcpwsrmuedg
Predicted Sentiment: negative

Tweet: csgo wingman im silver dont bully twitchtvlprezh
Predicted Sentiment: negative

Tweet: game suck 2 38 second leave team intentionally foul
Predicted Sentiment: negative

Tweet: congrats nvidia nemo team 100 release candidatereally excite see nemo embrace hydra way take control configuration madness machine learn
Predicted Sentiment: positive
```

Figure 8. Prediction Output

3 Final Discussions

The train-validation loss curve shows the model's learning progress over epochs having decreasing losses which means it is effectively learning without significant overfitting. The model accuracy curve on the other hand shows increasing accuracy over epochs meaning improvement in performance on both training and validation datasets, with minimal overfitting. In evaluation, the loss is 0.2137, and accuracy is 0.9124, indicating good model performance. The classification report, highlights high precision, recall, and F1-score for each class(<.90), with an overall accuracy of 91%. And the confusion matrix reveals true positives (3471), true negatives (3897), false positives (356), and false negatives (351) helping us understand the model's predictive performance.

From the 15 outputs we can see that only 3 were predicted incorrectly i.e. output 3, 10 and 11. Out of roughly 40,374 tweets this is only 15 and it predicted 12 correctly and also from the confusion matrix we can infer that the TP and TN are higher in number than FP and FN. Hence, we can infer that this model is performing optimally.

No model is perfect, even though this model performs optimally it’s crucial to acknowledge that given more training time and broader range of input we can push the model’s prediction capability. Given the limited free resources from ‘Google Collab’ the model performance is commendable. For future this model can serve as a solid foundation, potentially aiding to advancements in natural language processing and related fields.

4 References

Baeldung, 2024. *Epoch in Neural Networks.* [Online]
Available at: https://www.baeldung.com/cs/epoch-neural-networks
[Accessed 2 May 2024].

Carremans, B., 2018. *Word embeddings for sentiment analysis.* [Online]
Available at: https://towardsdatascience.com/word-embeddings-for-sentiment-analysis-65f42ea5d26e
[Accessed 30 April 2024].

Khan, M. B., 2023. *Fine-Tuning Deep Learning with Hyperparameters: A Beginner's Guide.*. [Online]
Available at: https://bootcamp.uxdesign.cc/fine-tuning-deep-learning-with-hyperparameters-a-beginners-guide-a974e19e287e
[Accessed 1 May 2024].

Nama, A., 2023. *Understanding Bidirectional LSTM for Sequential Data Processing.* [Online]
Available at: https://medium.com/@anishnama20/understanding-bidirectional-lstm-for-sequential-data-processing-b83d6283befc
[Accessed 30 April 2024].

NLP, P., 2021. *Twitter Sentiment Analysis.* [Online]
Available at: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis/data
[Accessed 30 April 2024].

Sarfraz, S. et al., 2022. A Deep Neural Network-Based Approach for Sentiment Analysis of Movie. *Complexity,* 2022 (Special), pp. 1076-2787.

Team, T. 3., 2023. *What Is Cross-Entropy Loss Function?.* [Online]
Available at: https://365datascience.com/tutorials/machine-learning-tutorials/cross-entropy-loss/
[Accessed 2 May 2024].

Verma, Y., 2024. *Dense Layers.* [Online]
Available at: https://analyticsindiamag.com/topics/what-is-dense-layer-in-neural-

network/

[Accessed 30 April 2024].

Vishwakarma, N., 2024. *What is Adam Optimizer?*. [Online]

Available at: https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/

[Accessed May 2 2024].

Yadav, H., 2022. *Dropout in Neural Networks.* [Online]

Available at: https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9

[Accessed 30 April 2024].