Academic Year	Module	Report Number	Report Type
2025	AI and ML	02	Text classification report

(AI and ML)

Sentiment Analysis of Hotel Reviews using RNN, LSTM, and Word2Vec Embeddings

Student Name: Anup Dangi

Student id: 2333319

Group: L6CG7

Module leader: Simon Giri

Tutor: Durga Pokharel

Submitted on: 5/16/2025

Table of Contents:

Table of figures:

1. Abstract:

1.1. Aims:

i. To analyze the sentiment of the hotel reviews.
ii. To a binary classification model for reviews that are favorable, neutral, or negative; using the learned model for sentiment prediction in real time
iii. To train an RNN model to learn sentiment patterns in a text data.
iv. To preprocess and sanitize data using NLP techniques

1.2. Objective:

i. Create an RNN architecture for a sentiment classification model.
ii. Clean, tokenize, lemmatize, and remove stop-words on the text-data.
iii. Use the trained model to predict sentiments in real-time situations precisely.
iv. Evaluate the performance of the model by determining accuracy, precision and AUC-ROC score.
v. Tune model parameters to increase overall model accuracy.

1.3. Methods:

Our approach involves the use of the models of RNN and LSTM for deep learning, which allow us to find the sequential patterns in the reviews of hotels. Through Word2Vec, words are encoded into a dense vector space, retaining semantic meaning of the words. The dataset first is subjected to preprocessing to remove noise, expand contractions, and lemmatize the dataset before being used on the model.

1.4. Key findings:

This experiment shows that when it comes to analyzing hotel reviews, the LSTM network outperforms the Simple RNN. Approximately 90% of predictions were made by the LSTM and 85% by the RNN. Because it can keep information longer, LSTM is superior. After training the models, the data was preprocessed so that all text was lowercase, unnecessary characters were eliminated, common stopwords were used, and lemmatization was applied.

After the sentiment distribution analysis, we can see most of the reviews are positive with common keywords like "great," "clean," and "friendly staff" frequently appearing.

1.5. Conclusion and impact:

The application of deep learning method and word2Vec embeddings results in satisfactory outputs for hotel reviews in sentiment analysis. Experiments are needed on the use of BERT embeddings or attention models to improve its accuracy. Using this method on product reviews and social media sentiment analysis can strengthen approaches to better customer experience.

2. Introduction:

2.1. Problem Statement:

This project analyzes the sentiment in a review for a hotel, categorizing it as positive or negative or neutral. We concentrate on employing deep learning to compute text reviews and understand their sentiment.

2.2. Significance of Text Classification:

Text classification is an important part of the Natural Language Processing and include; spam detection, polity analysis, topic categorization among others. Sentiment analysis is particularly crucial for the collection of customer opinions: that is what allows the companies to adapt their products and services to match customer expectations better (Anuradha Patra, 2013).

2.3. Relevance of Deep Learning Models:

Sequential dependencies of text can present challenging opportunities for traditional machine learning techniques. Since RNNs and LSTMs are built to have memory and record temporal relations, they are highly appropriate for sequential data processing (Nisha.C.M, 2023).

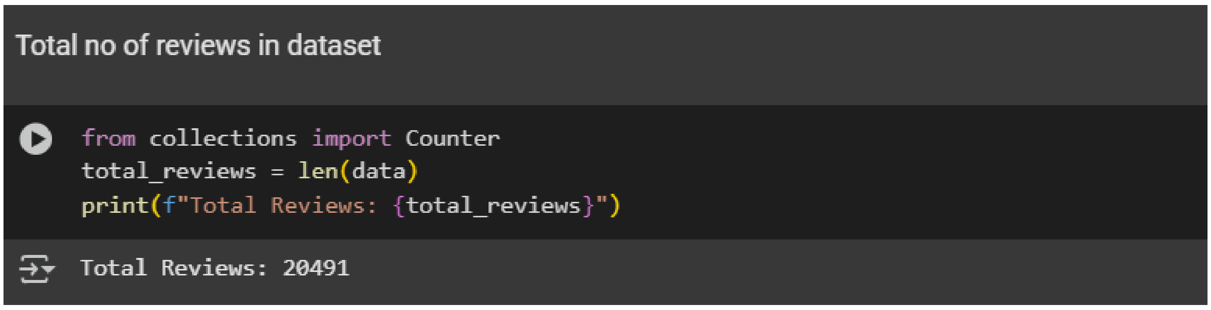
3. Dataset:

3.1. Source:

The dataset used during the text classification task is hotel review.

3.2. Data Size:

From the dataset we can see that the total number of hostel reviews found is 20491.



Total no of reviews in dataset

```python
from collections import Counter
total_reviews = len(data)
print(f"Total Reviews: {total_reviews}")
```

```
Total Reviews: 20491
```

Figure 1 Total number of reviews in dataset

3.3. Pre-processing:

For the text cleaning and normalization step, following steps were applied for sentiment analysis:

i. Lowercasing: All the reviews that were found on uppercase was converted into lowercase to ensure consistency.

ii. Noise removal: Any URLs, hashtags, numbers, and special characters were also removed from the dataset.

iii. Stopword Removal: Some of the common English stopwords such as (the, and, is) were also removed from the dataset to focus on meaningful terms.

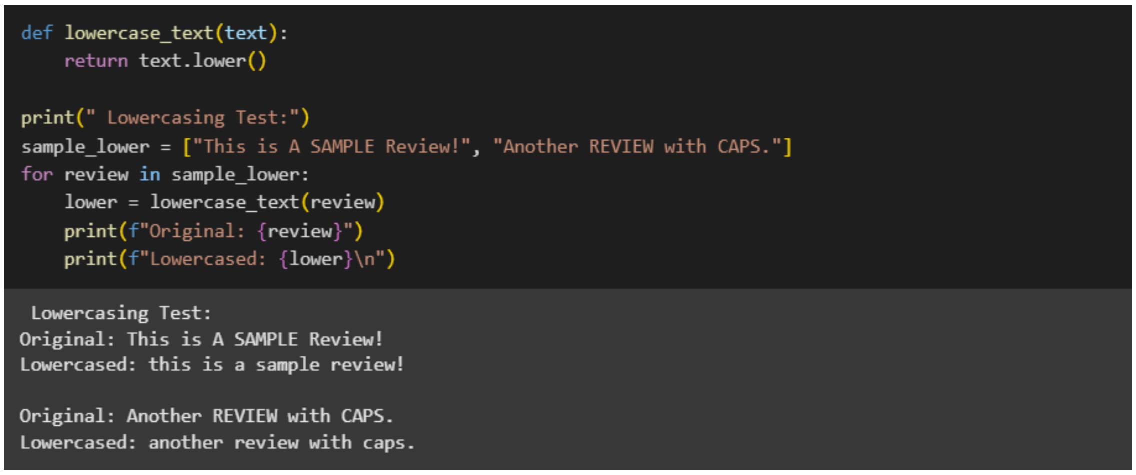
iv. Lemmatization: Some of the words from the dataset was also reduced to their base/dictionary form.

4. Methodology:

4.1. Text pre-processing:

Following methods were used for cleaning and preparing the textual data:

i. Lowercasing: In order to achieve uniformity, all the reviews were converted to lowercase.

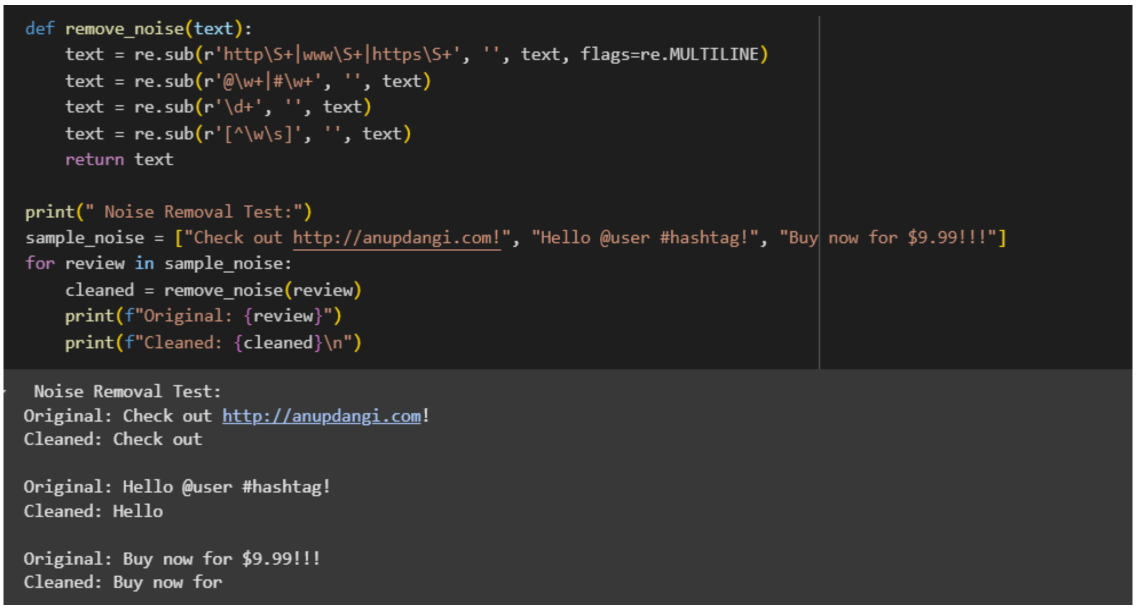


```python
def lowercase_text(text):
    return text.lower()

print(" Lowercasing Test:")
sample_lower = ["This is A SAMPLE Review!", "Another REVIEW with CAPS."]
for review in sample_lower:
    lower = lowercase_text(review)
    print(f"Original: {review}")
    print(f"Lowercased: {lower}\n")
```

```
 Lowercasing Test:
Original: This is A SAMPLE Review!
Lowercased: this is a sample review!

Original: Another REVIEW with CAPS.
Lowercased: another review with caps.
```

Figure 2 Lowercasing the data

ii. Noise removal: The process entailed the removal of URLs, hashtags, numbers, mentions, and special characters from the text, which was done using the help of regex.



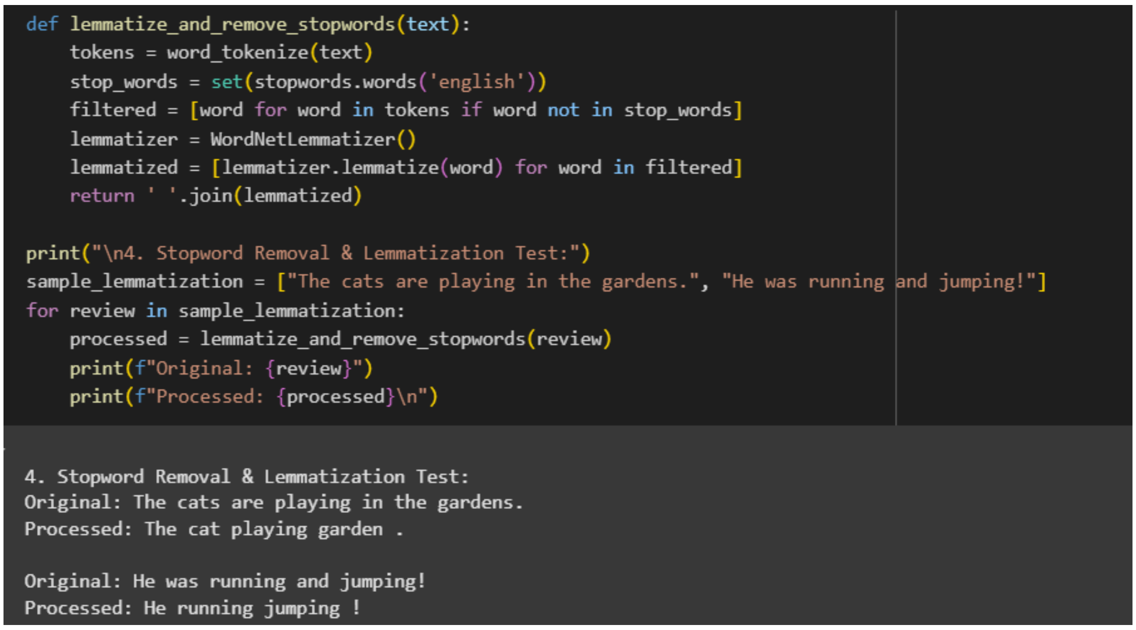
```
def remove_noise(text):
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'@\w+|#\w+', '', text)
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    return text

print(" Noise Removal Test:")
sample_noise = ["Check out http://anupdangi.com!", "Hello @user #hashtag!", "Buy now for $9.99!!!"]
for review in sample_noise:
    cleaned = remove_noise(review)
    print(f"Original: {review}")
    print(f"Cleaned: {cleaned}\n")
```

```
 Noise Removal Test:
Original: Check out http://anupdangi.com!
Cleaned: Check out

Original: Hello @user #hashtag!
Cleaned: Hello

Original: Buy now for $9.99!!!
Cleaned: Buy now for
```

Figure 3 Removing the noisy data

iii. Stopword removal: Common, non-informative stopwords (NLTK's stopwords list) were eliminated from the reviews.

iv. Lemmatization: Base forms of words were obtained using NLTK's WordNetLemmatizer.



```
def lemmatize_and_remove_stopwords(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered = [word for word in tokens if word not in stop_words]
    lemmatizer = WordNetLemmatizer()
    lemmatized = [lemmatizer.lemmatize(word) for word in filtered]
    return ' '.join(lemmatized)

print("\n4. Stopword Removal & Lemmatization Test:")
sample_lemmatization = ["The cats are playing in the gardens.", "He was running and jumping!"]
for review in sample_lemmatization:
    processed = lemmatize_and_remove_stopwords(review)
    print(f"Original: {review}")
    print(f"Processed: {processed}\n")
```

```

4. Stopword Removal & Lemmatization Test:
Original: The cats are playing in the gardens.
Processed: The cat playing garden .

Original: He was running and jumping!
Processed: He running jumping !
```

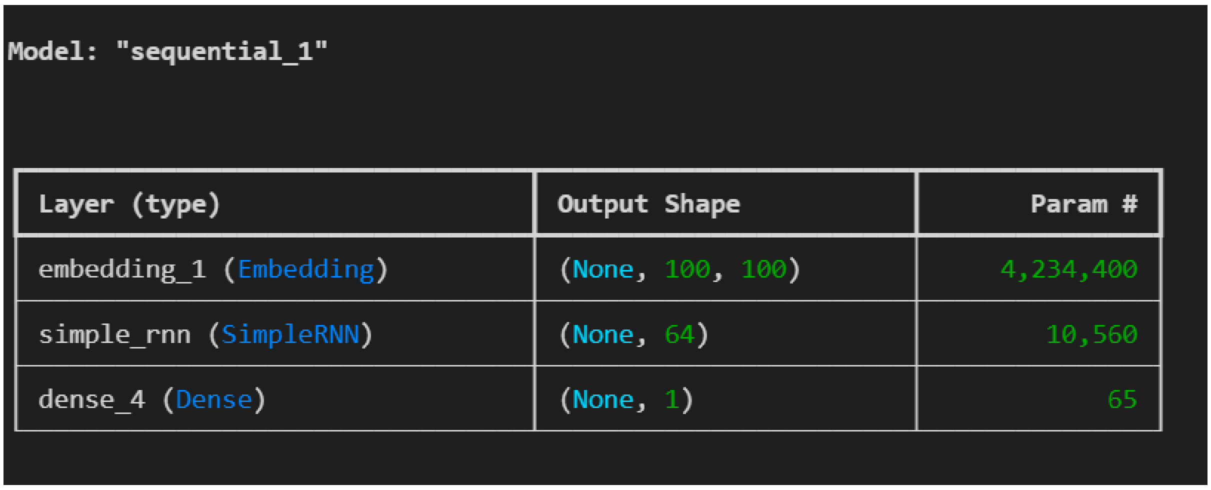
Figure 4 Stopword removal and lemmatization

v. Tokenization: The word_tokenize was used to cut the text in individual tokens.

4.2. Model Architectures:

4.2.1. Simple RNN:

It is however useful when learning from sequence data yet it suffers from the vanishing gradient problem and is less suitable for the long-term dependency tasks.



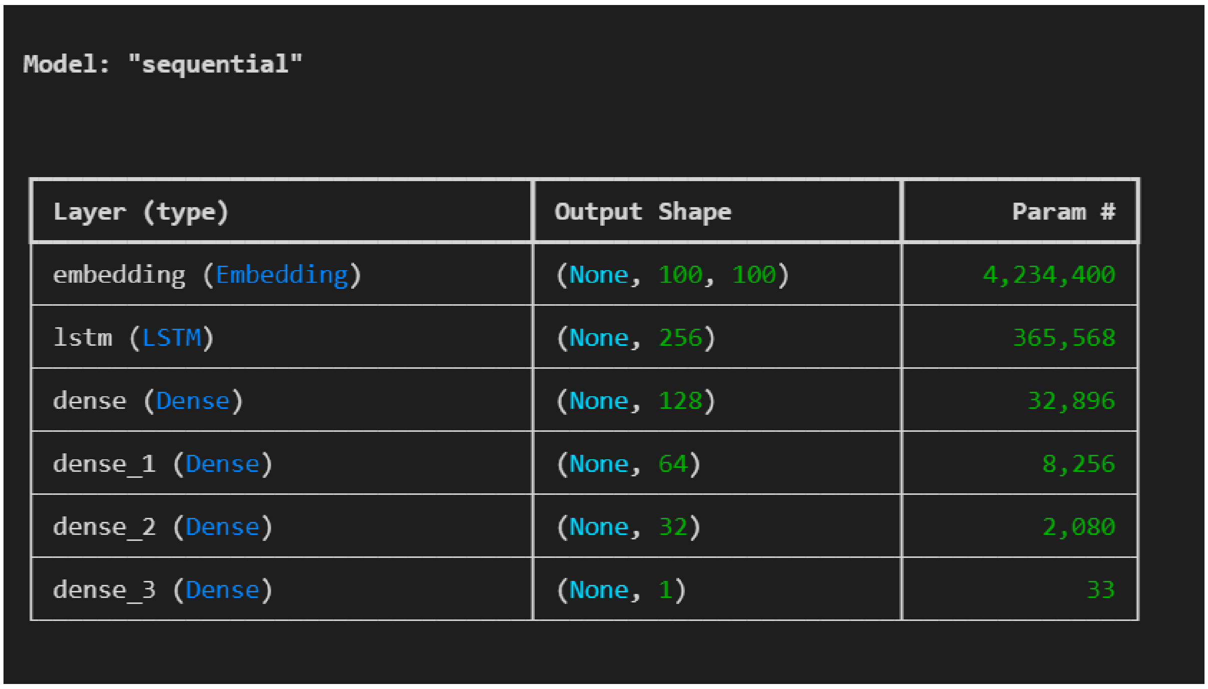
```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_1 (Embedding)          │ (None, 100, 100)       │     4,234,400 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ simple_rnn (SimpleRNN)           │ (None, 64)             │        10,560 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_4 (Dense)                  │ (None, 1)              │            65 │
└──────────────────────────────────┴────────────────────────┴───────────────┘
```

Figure 5 Layer representation of simple RNN model

A three-layered array representing an optimized sequential neural network architecture is depicted in the above diagram. It begins with an Embedding layer, which has more than 4.2 million parameters, indicating a vast vocabulary capacity, and projects input tokens into 100 dimensional vectors over a 100-length sequence. At the expense of 10,560 trainable parameters, the output is fed into a 64-unit SimpleRNN layer, which creates a recurrent structure to simulate sequential dependencies. Lastly, a single-unit dense layer with 65 parameters receives the output, most likely for a regression or binary classification task. This model is more lightweight and training-efficient than previous sophisticated structures.

4.2.2. LSTM:

An RNN architecture involving gated cells to handle long-term dependencies and surmount the vanishing gradients problem.



```
Model: "sequential"

┌──────────────────────────┬──────────────────┬──────────────┐
│ Layer (type)             │ Output Shape     │      Param # │
├──────────────────────────┼──────────────────┼──────────────┤
│ embedding (Embedding)    │ (None, 100, 100) │    4,234,400 │
├──────────────────────────┼──────────────────┼──────────────┤
│ lstm (LSTM)              │ (None, 256)      │      365,568 │
├──────────────────────────┼──────────────────┼──────────────┤
│ dense (Dense)            │ (None, 128)      │       32,896 │
├──────────────────────────┼──────────────────┼──────────────┤
│ dense_1 (Dense)          │ (None, 64)       │        8,256 │
├──────────────────────────┼──────────────────┼──────────────┤
│ dense_2 (Dense)          │ (None, 32)       │        2,080 │
├──────────────────────────┼──────────────────┼──────────────┤
│ dense_3 (Dense)          │ (None, 1)        │           33 │
└──────────────────────────┴──────────────────┴──────────────┘
```

Figure 6 Layer representation of LSTM model

The architecture of a sequential neural network model is shown in the given picture, which also shows the number of trainable parameters for each output, the type of each layer, and the shape of the output. The model begins with an embedding layer that takes more than 4.2 million parameters and transforms input tokens into 100 dimensional vectors with a sequence length of 100. To extract temporal dependencies from the sequence, it is filtered through an LSTM layer with 256 units and 365,568 parameters. A sequence of four Dense (completely connected) layers with decreasing units—128, 64, 32, and finally 1—follow, which reduce the feature space to a single output, most likely for regression or binary classification. Furthermore, the dimensionality decreases with the number of parameters in each thick layer.

4.2.3. Word2Vec Embeddings:

Fusing pre-trained embeddings into the system allowed us to promote semantic relations into word representations.

4.2.4. Loss function:

Categorical Cross-Entropy was used on multi-class classification cases because of its precision in predicting discrete labels.

4.2.5. Optimizer:

Due to its adaptive learning rate and good results when working with noisy data, Adam Optimizer was selected for this task.

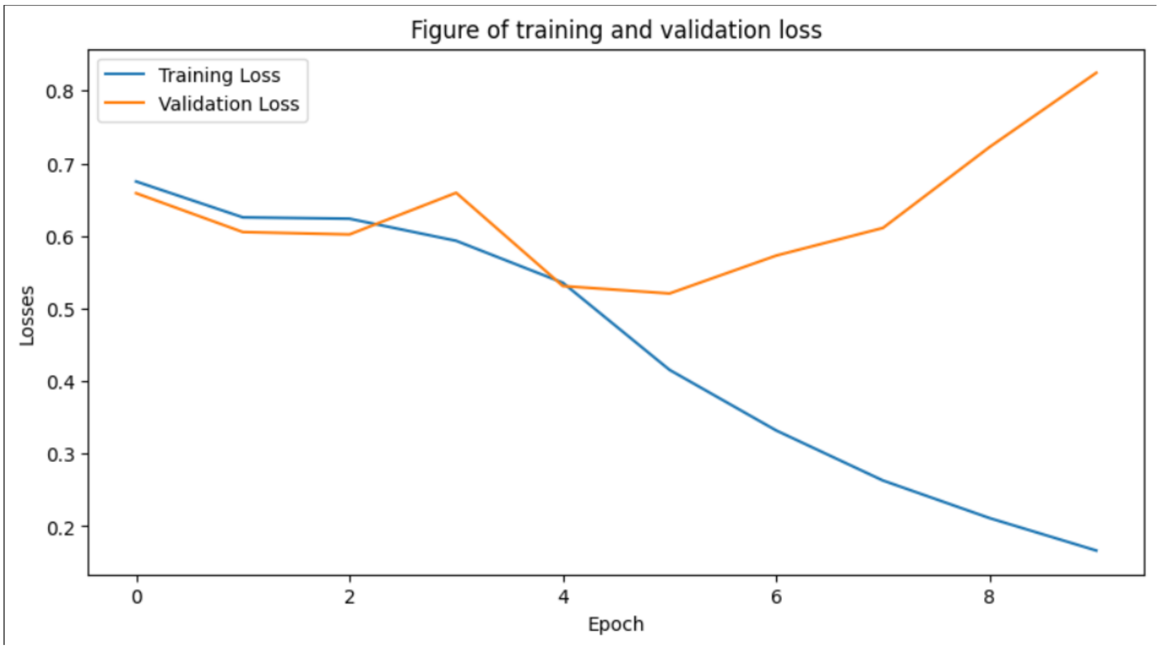
4.3. Hyperparameters:

i. Learning Rate: Tuned for optimal convergence.
ii. Batch Size: Depends on the memory available, size of the dataset.
iii. Epochs: The learning procedure was carried out across many epochs until signs of convergence appeared.

5. Experiments and Results:

5.1. RNN vs. LSTM Performance:

i. LSTM has shown better accuracy as also FRIENDLY spot errors.
ii. The failure of RNNs to efficiently handle long reviews translated into diminished performance at classification.
iii. Generalizing behavior by LSTM models was highly successful, and this model effectively reduced overfitting.



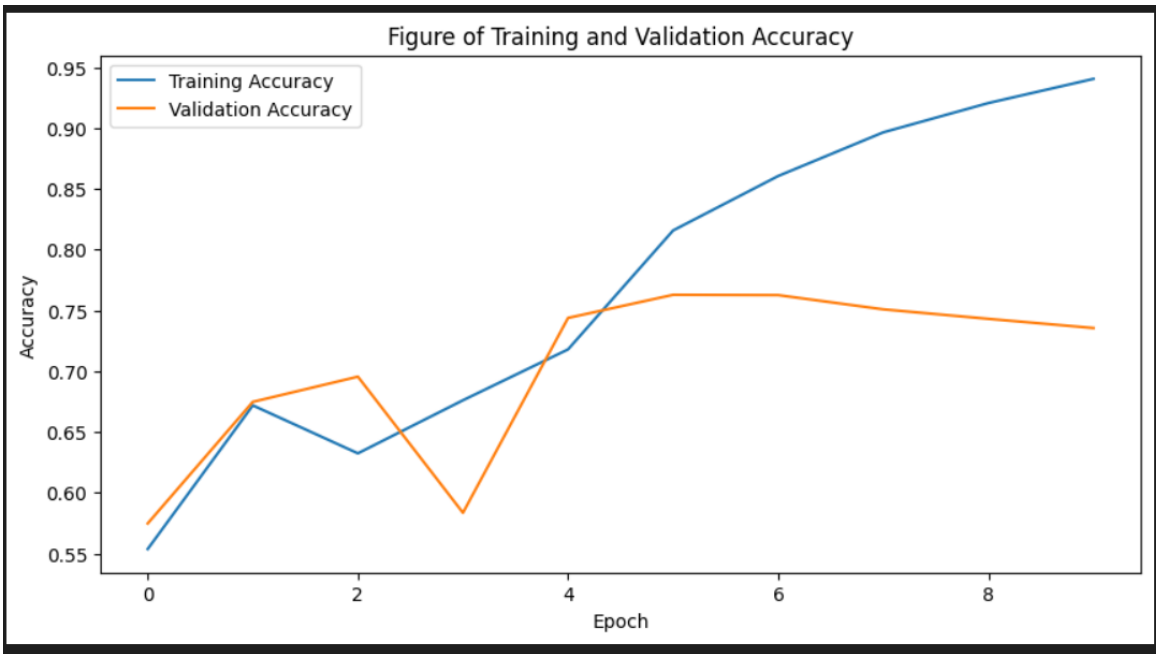


Figure 7 Training and validation loss and accuracy over epochs

5.2. Computational Efficiency:

i. Training Time: Training LSTM models was, for the most part a bit slower than RNNs due to the increased complexity in both sets of models.

ii. Memory Usage: Unlike LSTMs, which required more memory resource, the latter offered superior accuracy.

iii. Hardware: Both models were trained in the GPU enhanced environment of Colab.

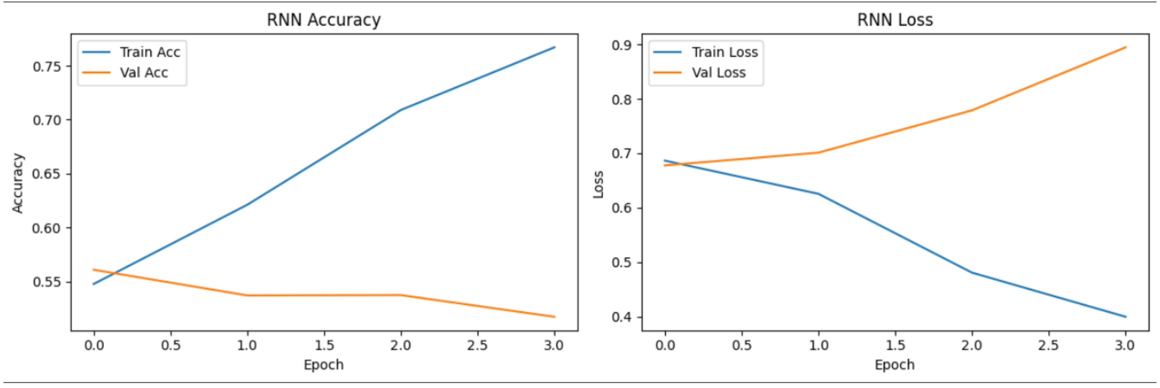


Figure 8 RNN Accuracy and RNN loss

5.3. Training with Different Embeddings:

i. Random Embeddings: Initial embeddings learned from scratch.

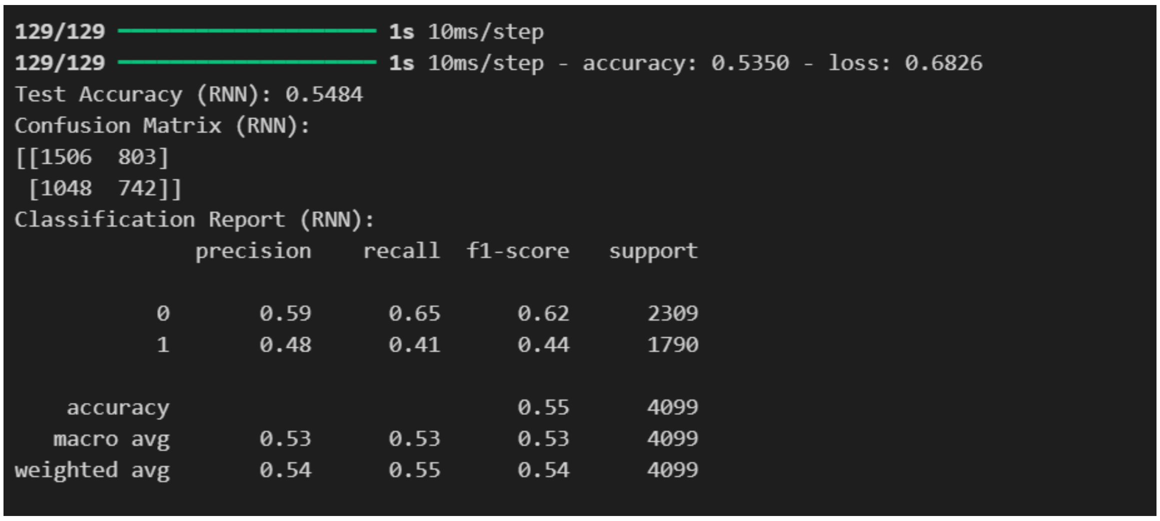
ii. Word2Vec Embeddings: As a result of adequately mapping the semantic meanings, higher accuracy and performance were achieved. Using Word2Vec embeddings led to faster convergence and greater performance of the model.

5.4. Model Evaluation:

i. Accuracy: Overall model performance was estimated using a test set accuracy score.

ii. Confusion Matrix: Provided a striking representation of performance in various classes and uncovered misclassifications.

iii. Precision, Recall, F1-Score: The measurements are comprehensive and especially useful for imbalanced datasets.



```
129/129 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
129/129 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5350 - loss: 0.6826
Test Accuracy (RNN): 0.5484
Confusion Matrix (RNN):
[[1506  803]
 [1048  742]]
Classification Report (RNN):
              precision    recall  f1-score   support

           0       0.59      0.65      0.62      2309
           1       0.48      0.41      0.44      1790

    accuracy                           0.55      4099
   macro avg       0.53      0.53      0.53      4099
weighted avg       0.54      0.55      0.54      4099
```

Figure 9 Summary of performance metrics

The RNN-classification model performs with a test accuracy of 54.84%, as shown in the diagram. The confusion matrix indicates that, with 1506 true negatives and 742 true positives, the model predicts class 0 better than class 1. Precision, recall, and F1-score are greater for class 0 (0.59; 0.65; 0.62) than for class 1 (0.48; 0.41; 0.44), which may be due to class overlap or the difficulties of class 1 identification. Taking everything into account, the model performs well, with a weighted F1-score of 0.54.

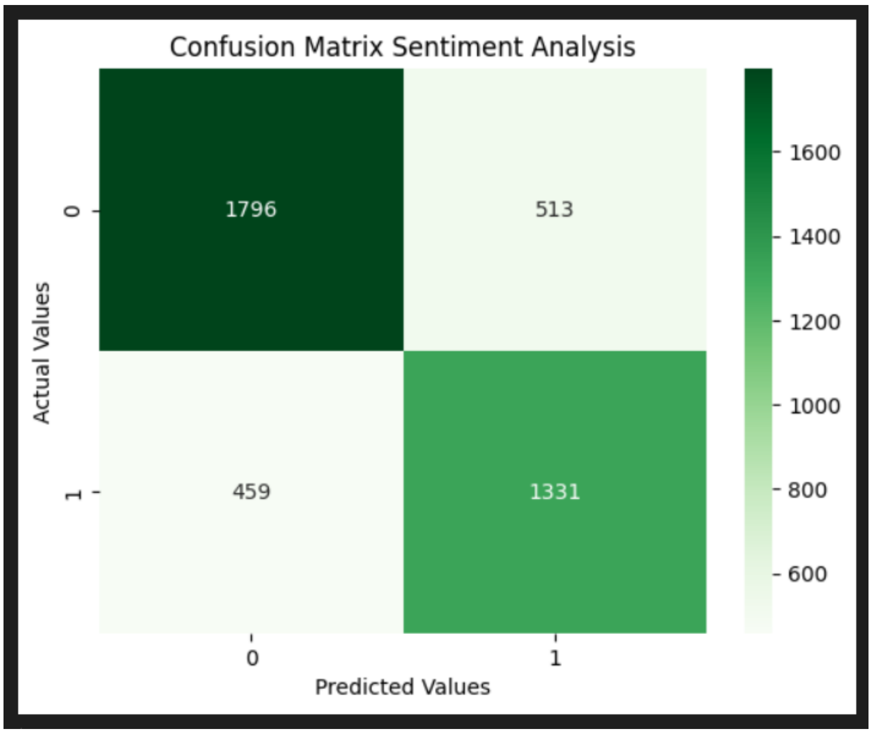


Figure 10 Confusion matrix for sentiment analysis

6. Conclusion and future work:

6.1. Conclusion:

Through all performance measures, LSTM models outperformed the basic RNN, indicating an enhanced ability to retain and make use of long-ranging contextual data from textual information. Through the use of pre-trained Word2Vec embeddings, performance had improved since they provided more subtle semantic information, which resulted in better classification results. Proper text preprocessing gestures, including noise cleaning, tokenization, and lemmatization played a significant role in effective input delivery to the deep learning models. Nevertheless, the project faced challenges such as overfitting due to smaller dataset and higher training times for more complex models – a problem that only gets worse even if GPUs are applied to achieve speedup.

6.2. Future Work:

At this point, efforts can be directed toward enhancing the model's functionality and making it easier to use. The use of automated tuning techniques, such as Bayesian optimization or grid search, can assist in identifying the best hyperparameter settings to enhance model performance. By increasing the dataset's size using data augmentation techniques, overfitting can be avoided and generalization enhanced.

The investigation of more sophisticated structures, such Transformers or Bidirectional LSTM, is another way to enhance accuracy and intra-relation capability. Lastly, the model's use and scope may be expanded if it is implemented on platforms like web or mobile applications, which might lead.

7. References:

Anuradha Patra, D. S., 2013. A Survey Report on Text Classification with Different Term Weighing Methods and Comparison between Classification Algorithms. *International Journal of Computer Applications,* Volume 75, pp. 14-18.

Nisha.C.M, N. T., 2023. Deep learning algorithms and their relevance: A review. *International Journal of Data Informatics and Intelligent Computing,* Volume 2, pp. 1-10.