UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning.

6CS012

Text Classification

Student Id	: NP03CS4A220499
Student Name	Anish Bahadur Karki
Group	: L4CG2
Tutor	Mr. Shiv Yadav
Word Count	: 2300
Submitted on	: 15-05-2025

Table of Contents

Contents

Table of Figures

1. Introduction

Text classification is one of the main tasks of natural language processing, with the task of spam detection, sentiment analysis, and topic labelling as examples. In this project, sentiment analysis is our goal, trying to classify if a hotel review has a positive, negative or neutral sentiment. Hotel reviews are commonly nonsystematic, non-analytical and can be informal, contain slang and abbreviations, which presents challenges when analyzing.

To do so, we used sequence-dependent deep learning models for sequential text data. Recurrent Neural Networks (RNNs) are popular for dealing with such a task since they work on word-by-word basis with the input. RNNs however may have problems remembering important information in longer sentences because of the vanishing gradient problem. To make up for this, we also used Long Short-Term Memory (LSTM) networks, which are more effective in retention of such details in longer sequences.

We represented words using Word2Vec embeddings in a dense, meaningful manner. Word2Vec assists the model in learning the associations between words from their context, giving more usable data than the random vectors of the words, and consequently leads to a better understanding of meaning.

In this project, we assessed and compared the performance of RNN and LSTM models, both with randomly initialized and pretrained Word2Vec embeddings, for sentiment classification of hotel reviews.

2. Dataset

	Review	Rating
0	nice hotel expensive parking got good deal sta...	4
1	ok nothing special charge diamond member hilto...	2
2	nice rooms not 4* experience hotel monaco seat...	3
3	unique, great stay, wonderful time hotel monac...	5
4	great stay great stay, went seahawk game aweso...	5

Figure 1: Data Collections

2.1. Source:

The dataset used in this project has the name Hotel_Review.csv and has only two features, i.e., _ review and rating. Review column covers short textual feedback from the hotel guests, whereas, rating column shows the sentiment of each review – either positive, negative, or neutral. This organisation of the dataset allows its use in training deep learning for sentiment classification purposes.

2.2. Data Size:

```
[7]: label_counts = df['Rating'].value_counts()
     print(label_counts)

Rating
5    9054
4    6039
3    2184
2    1793
1    1421
Name: count, dtype: int64
```

Figure 2: Data Size

We have 21000 phrases in the dataset, these labels are spread into 2 classes.

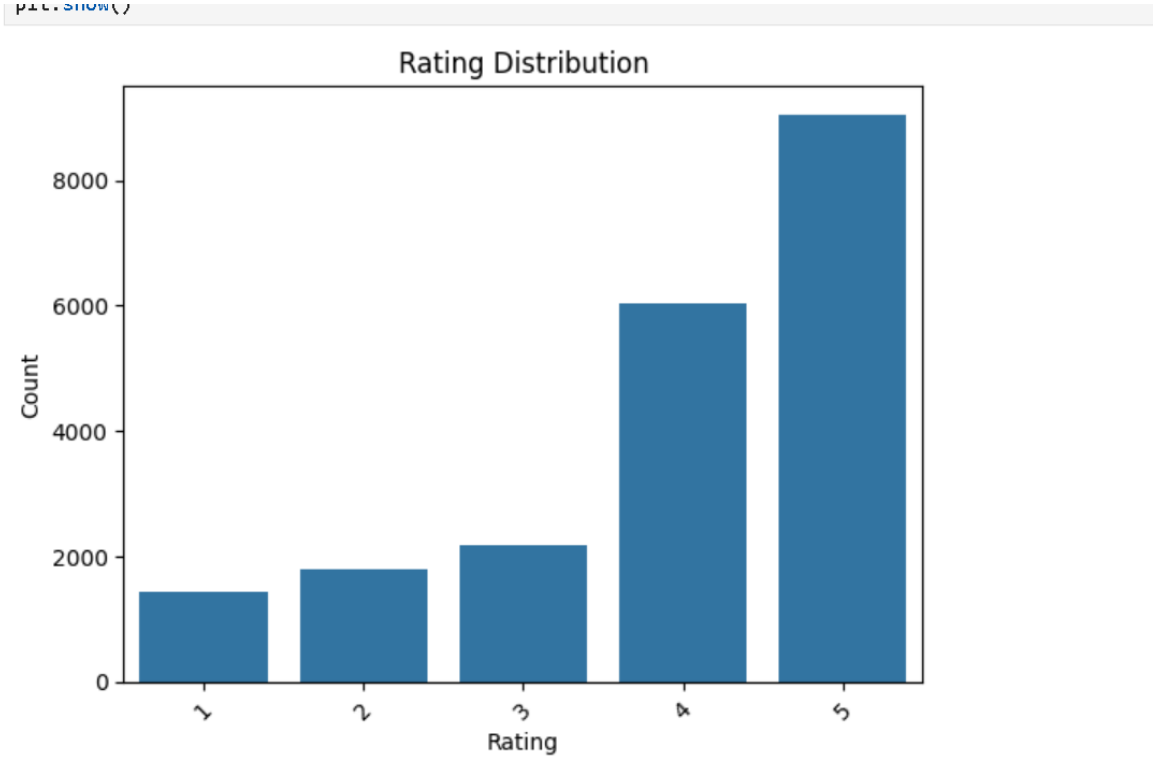

Figure 3 Ratings Distribution

This shows a clear imbalance in the dataset, with neutral sentiments appearing much more frequently than the others. Because of this imbalance, the model may tend to predict the neutral class more often unless properly handled.

Figure 4: 100 Most Frequently Used Words

2.3. Pre-processing:

```
# Preprocessing function
def preprocess_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)  # Remove URLs
    text = re.sub(r'\@\w+|\#', '', text)  # Remove mentions and hashtags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\d+', '', text)  # Remove numbers

    # Remove stopwords and lemmatize
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    processed_text = ' '.join(tokens)
    return processed_text
```

Figure 5: Preprocessing Function

- The text data was cleaned and prepared before training. The steps included:
- Converting all text to lowercase
- Removing stopwords using the NLTK stopword list
- Lemmatising words to their base form using WordNetLemmatizer
- Tokenising the cleaned text into sequences of integers using Keras Tokenizer
- Applying zero padding so that all sequences have the same length

5]:

	Review	Rating	cleaned_text
0	nice hotel expensive parking got good deal sta...	4	nice hotel expensive parking got good deal sta...
1	ok nothing special charge diamond member hilto...	2	ok nothing special charge diamond member hilto...
2	nice rooms not 4* experience hotel monaco seat...	3	nice room experience hotel monaco seattle good...
3	unique, great stay, wonderful time hotel monac...	5	unique great stay wonderful time hotel monaco ...
4	great stay great stay, went seahawk game aweso...	5	great stay great stay went seahawk game awesom...

Figure 6: Pre-Processed Text

After pre-processing, this dataset was partitioned into two sets (80/20%) training and test set. The training set had 16000 samples and the sample number for test set was 4500. Then this prepared data was used for training and validation of the models.

3. Methodology

In this section it is written, the steps followed to create and train deep learning models for sentiment classification for Hotel Review phrases are written.

3.1. **Text Preprocessing**:

- To prepare the text data for modelling, several pre-processing steps were applied:

- All text was converted to lowercase to maintain uniformity.
- Stopwords (commonly used words that add little meaning, like “the”, “is”, etc.) were removed using the NLTK stopword list.
- Lemmatization was applied using WordNetLemmatizer to reduce words to their base form (e.g., “running” becomes “run”).
- After cleaning, the dataset was split into training and test sets using an 80/20 ratio.
- The text data was then tokenized using Keras’s Tokenizer, which converts words to integer sequences.
- Finally, zero padding was used to ensure that all sequences were the same length (based on the longest sentence).

•3.2. **Model Architecture:**

3.2.1. Simple RNN:

A Keras SimpleRNN layer was utilised for a standard Recurrent Neural Network implementation. This model is used here as a baseline to see how sequential data behaves with a trivial memory-based approach.

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 259, 128)	1,280,000
simple_rnn (SimpleRNN)	(None, 64)	12,352
dense (Dense)	(None, 5)	325

Total params: 1,292,677 (4.93 MB)

Trainable params: 1,292,677 (4.93 MB)

Non-trainable params: 0 (0.00 B)

Figure 7 Sequential RNN model is designed for sentiment analysis.

The above image illustrates how a sequential RNN model is designed for sentiment analysis. It has an Embedding layer with 128 components, a SimpleRNN layer with 64 units, and a Dense layer rounding it off with 5 units. The model does not have any non-trainable parameters; the total number of trainable parameters is 1,292,677.

```
97/97 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step
Classification Report:
              precision    recall  f1-score   support

           1       0.12      0.01      0.02       213
           2       0.05      0.00      0.01       269
           3       0.08      0.00      0.01       328
           4       0.41      0.13      0.20       906
           5       0.46      0.92      0.61      1358

    accuracy                           0.45      3074
   macro avg       0.22      0.21      0.17      3074
weighted avg       0.34      0.45      0.33      3074
```

Figure 8: Results for the RNN Sentiments

The report shown above demonstrates the results for the RNN sentiment model. In class 5, the model gives the highest recall of 0.92 and F1-score of 0.61. Still, the performance for classes 1–3 is low. Its accuracy is 45%, and the weighted F1-score is 0.33.

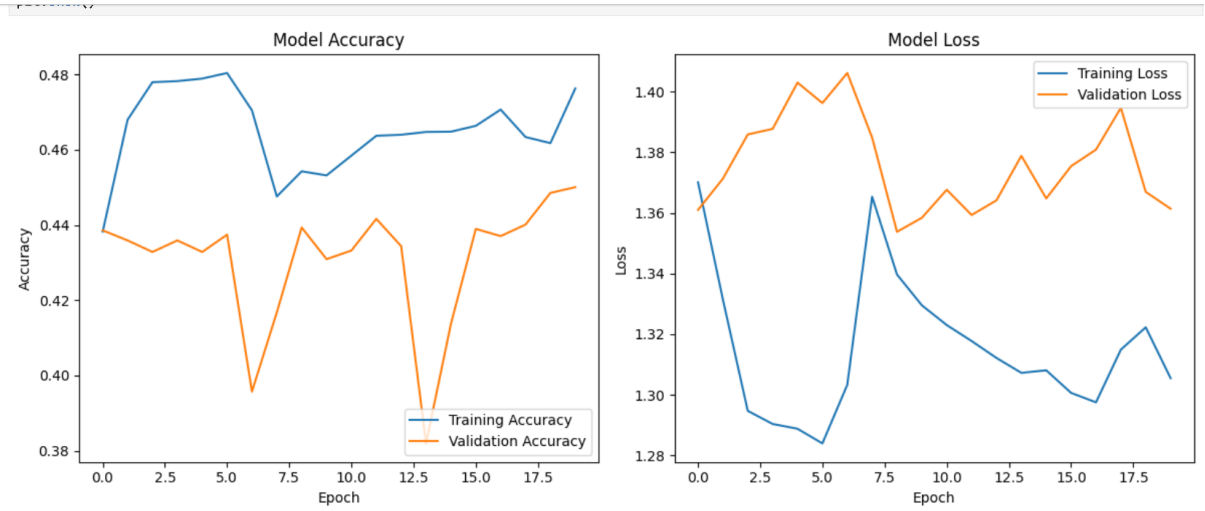

Figure 9 Comparison of Model Accuracy and Loss

The image above compares the training and validation accuracy of the RNN model during 20 epochs. With each training session, accuracy gets close to 0.48, and the validation accuracy stays close to 0.44. Validation loss is not consistent, which signals that the model might be lacking the capacity to do well on new input data.

3.2.2. LSTM:

An LSTM model was applied in an implementation of a Kera's LSTM layer. LSTM (Long Short-Term Memory) has the ability to retain long connections in the text thus helping more in sentence structure and meaning comprehension particularly in lengthy phrases.

```
97/97 ━━━━━━━━━━━━━━━━━━━━ 7s 65ms/step
Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       213
           2       0.19      0.03      0.05       269
           3       0.00      0.00      0.00       328
           4       0.00      0.00      0.00       906
           5       0.44      0.99      0.61      1358

    accuracy                           0.44      3074
   macro avg       0.13      0.20      0.13      3074
weighted avg       0.21      0.44      0.28      3074
```

Figure 10: LSTM Classification Report

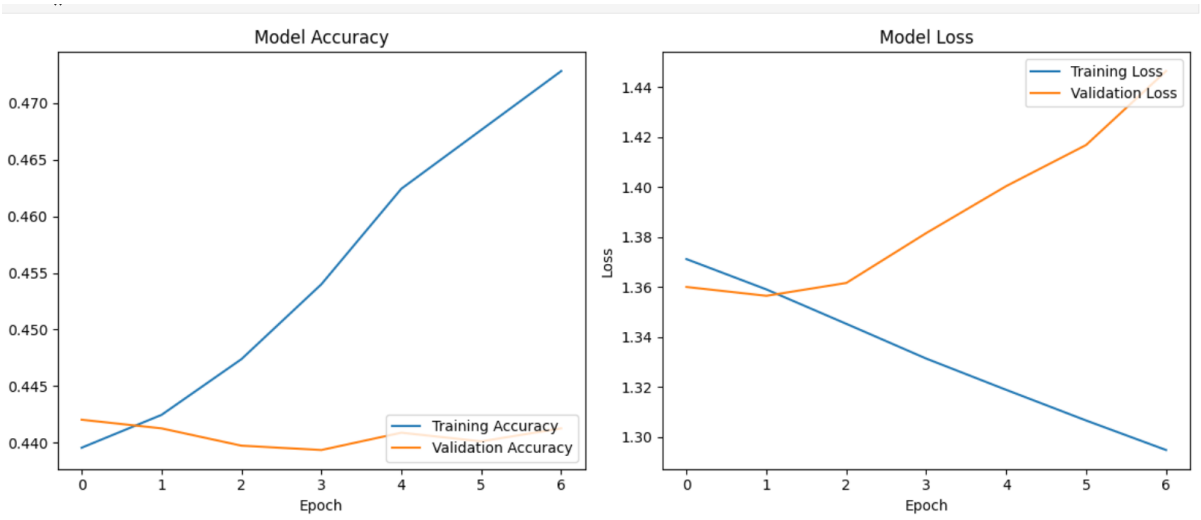

Figure 11: LSTM Model Accuracy and Loss

3.2.3. Word2Vec Embedding:

We loaded pre-trained Word2Vec embeddings (Google news vectors) using the Gensim module. We took these and extracted the embeddings, and created an embedding_matrix, which was fed into an Embedding layer of the model. This endowed and fused word's representation rather than a random embedding.

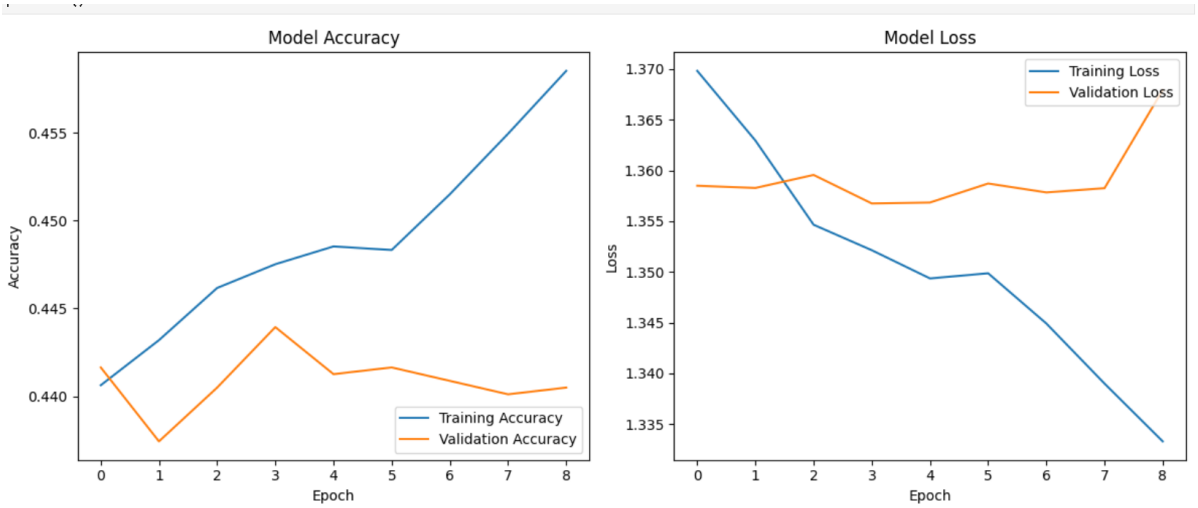

Figure 12: LSTM + Embedding Model Accuracy and Model Loss

```
print(classification_report(y_true, y_pred, target_names=target_names, ze
```

```
97/97 ━━━━━━━━━━━━━━━━━━━━ 9s 84ms/step
Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       213
           2       0.22      0.01      0.01       269
           3       0.00      0.00      0.00       328
           4       0.32      0.02      0.03       906
           5       0.44      0.99      0.61      1358

    accuracy                           0.44      3074
   macro avg       0.20      0.20      0.13      3074
weighted avg       0.31      0.44      0.28      3074
```

Figure 13: LSTM + Embedding Classification Report

3.2.4. Loss Function:

As a multiclass classification problem (two classes), the models applied here adopted categorical cross-entropy for the loss function.

3.2.5. Optimiser:

All models were compiled and trained using the Adam optimizer, which is widely used for its adaptive learning rate and fast convergence.

3.2.6. Hyperparameters:

- Key hyperparameters used in the training process:
- Learning Rate: Default used by Adam
- Batch Size: 128

- Number of Epochs: 5
- Embedding Dimension: 300 (matching the size of the Word2Vec vectors)
- Max Sequence Length: Determined from the longest input sequence

4. Experiments and Results

This section presents the experiments conducted to evaluate the performance of different models (RNN and LSTM) on financial sentiment analysis using various embeddings.

4.1. **RNN vs. LSTM Performance**

- Both RNN and LSTM models were trained on pre-processed financial phrase data.
- LSTM showed better performance than RNN in terms of:
- Accuracy: LSTM captured long-term dependencies more effectively.
- Loss: LSTM had lower validation loss during training.
- Stability: LSTM was less prone to vanishing gradient problems than RNN.
- Training Time: RNN trained faster, but LSTM provided better results.

4.2. Computational Efficiency

- RNN was faster to train but less accurate.
- LSTM took more time and memory but gave better output.
- On Google Collab:
  - GPU acceleration was used to reduce training time.
  - LSTM benefited more from the GPU than RNN due to its complexity.
- Memory usage was higher with LSTM due to its gated structure.

4.3. Training with Different Embeddings

Two types of word embeddings were tested:

Random Embeddings:

- Word vectors were randomly initialized and learned during training.
- Performance was decent but limited.

Word2Vec Embeddings (Pretrained):

- Pretrained Word2Vec vectors improved performance.

- Faster convergence and better generalization.

- LSTM + Word2Vec gave the best results overall.

Model Evaluation

Evaluation was done using the following metrics:

- **Accuracy**: The percentage of correct predictions.
- **Confusion Matrix**:
    - Shows true vs. predicted labels.
    - Helps identify class-specific errors.
- **Precision, Recall, and F1-Score**:
    - Used to measure performance per class.
    - Especially useful when classes are imbalanced.

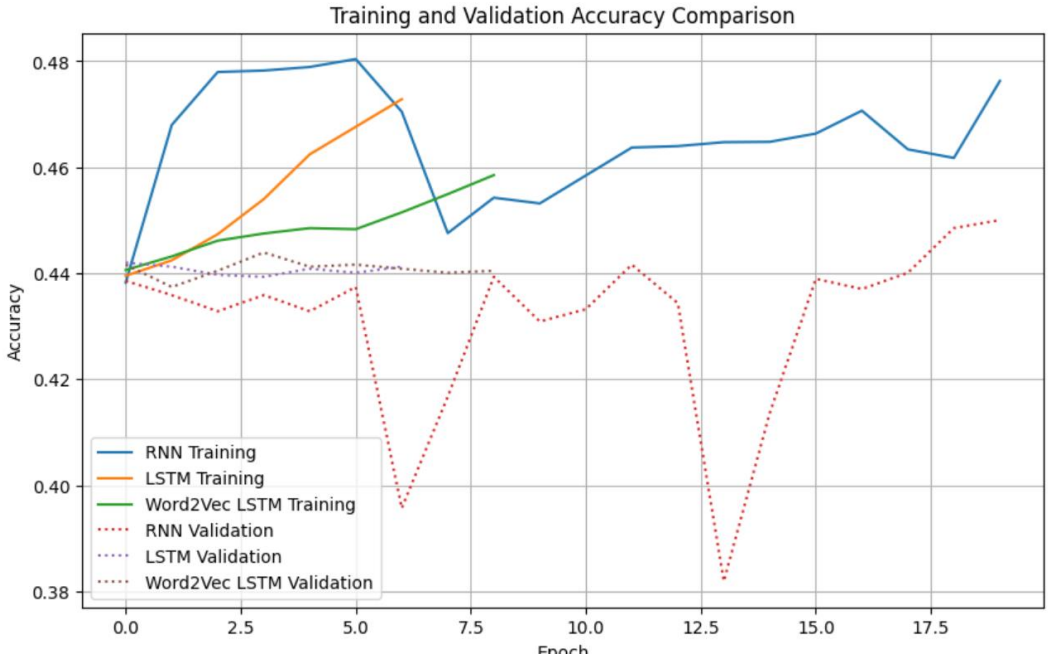

Figure 14 Comparison of all Models Training and Validation Accuracy

We can see an accuracy graph of training and validation data for 20 epochs with RNN, LSTM, and Word2Vec-LSTM below. With each increase in epochs, the accuracy rises. Accuracy while training is drawn with solid lines, and accuracy on the validation set is shown with dotted lines.

From the graph, we can see that the accuracy of RNN's training starts at 0.44 and finishes at roughly 0.48. At the same time, the red dotted line of the graph indicates that the model's validation accuracy keeps changing a lot, which may point to overfitting. As time passes, the accuracy from LSTM increases and stands at 0.47 for the sixth epoch. The dotted purple line (accuracy) shows that generalization is almost not improving at the current level of 0.44.

On the graph (the green solid line), the highest level of accuracy for the Word2Vec-LSTM model was reached at 0.459. Strikingly, what remains almost the same is the accuracy score of 0.44 to 0.45 shown by the brown dotted line. Still, the AI continues to work well and has no trouble going through each epoch.

All in all, the accuracy and stability of LSTM and Word2Vec-LSTM are greater than those of RNN.

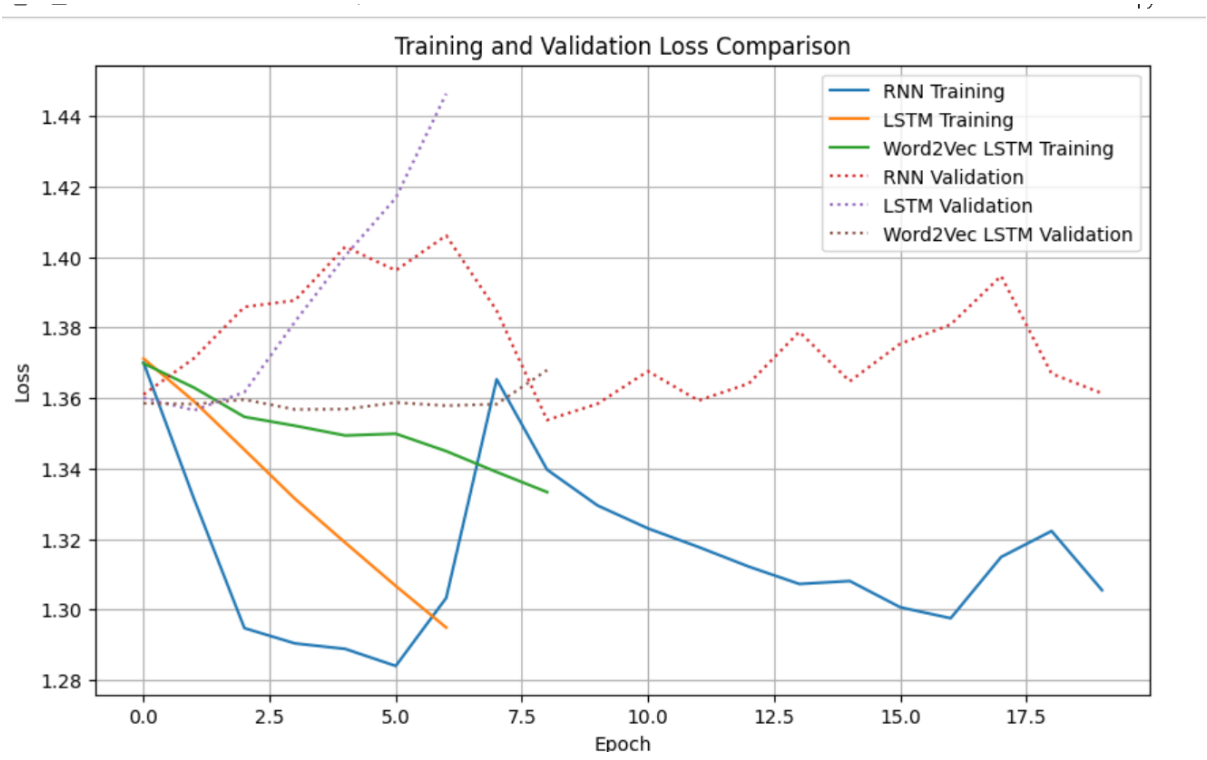

Figure 15: Training and Validation Loss Comparison

The graph shows how RNN, LSTM, Word2Vec+LSTM, and another method were trained and validated. Dropper: As the loss is significantly decreasing, it shows that the RNN model is improving. Even so, the model may be suffering from overfitting as its validation loss does not improve after training and varies instead. The same as the CNN model, the LSTM model shows a decreasing training loss, yet its validation loss increases, a stronger sign of overfitting. The reason could be that the model is very complicated, so instead of learning, it simply memorizes what it was shown. However, the Word2Vec and LSTM models do not make significant differences in error. With every training step, the loss goes down, and the validation loss hardly shifts. What this means is, the pre-trained Word2Vec model helps the algorithm learn and stay focused on the meaning, not just memorizing the training data. Of the three, Word2Vec and LSTM exhibit the least overfitting and the biggest similarity between the loss on the training and validation data. Consequently, it is better suited for carrying out text-related duties and might still improve when made richer or altered.

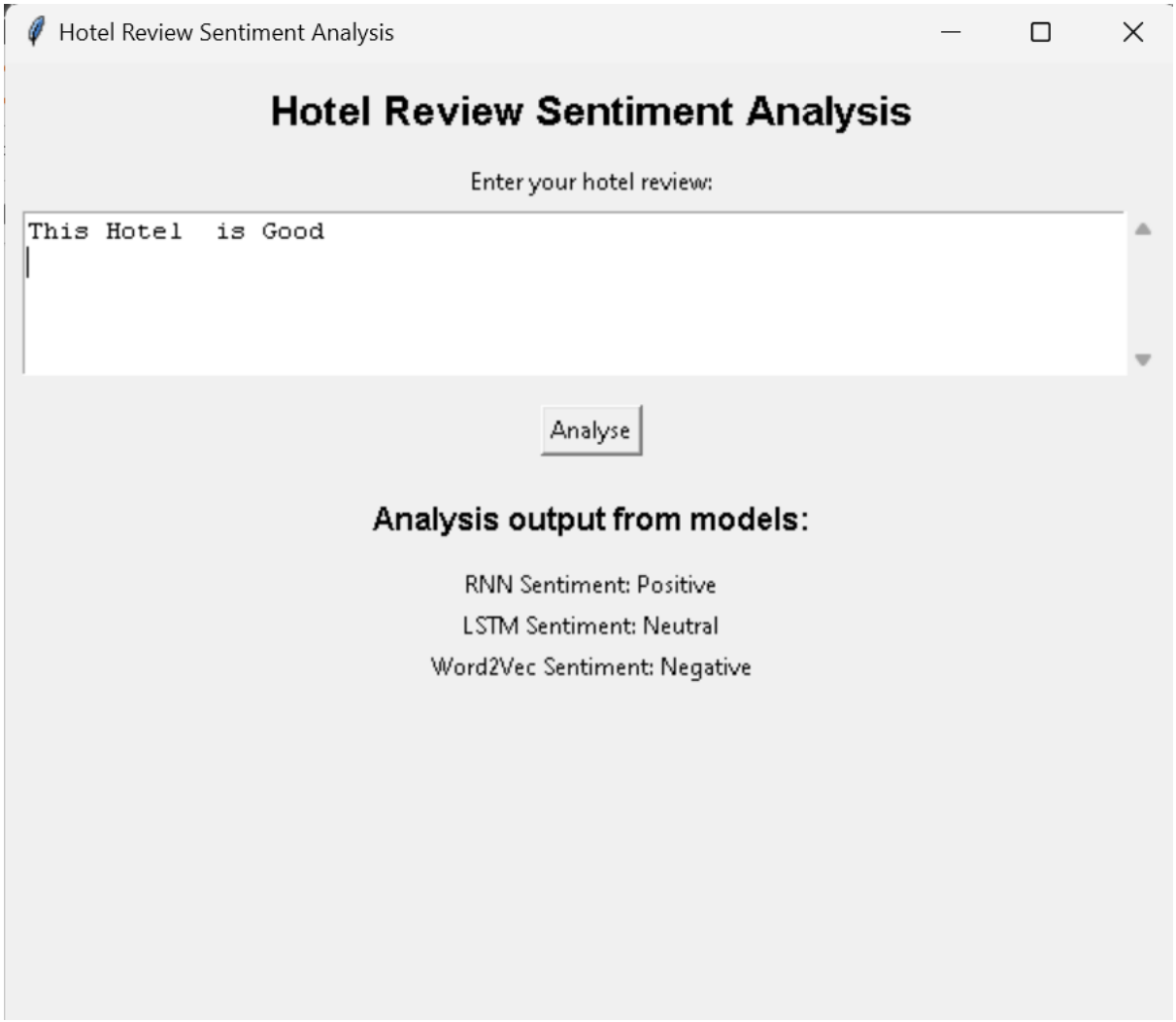

Figure 16 GUI

This above figure focuses on a tool called Hotel Review Sentiment Analysis, applying three models, namely RNN, LSTM, and Word2Vec LSTM, to rate a review by sentiment. According to the review, the user believes this Hotel is Good. After selecting “Analyse,” the report, including thoughts from each model, is shown.

The fact that the model considered the input to be negative is surprising. The model needs more information, or the input is beyond its capabilities.

In short, the training of a model determines the diversity of its outputs. It is important for successful sentiment analysis to have the right training, pre-processing, and model.

5. Conclusion and Future Work

In this project, we performed sentiment analysis on financial phrases using deep learning models such as RNN and LSTM. We started by applying thorough text preprocessing steps, including lowercasing, removing URLs, mentions, punctuation, numbers, and stopwords, and then lemmatizing the words. This helped clean the data and make it more suitable for training. During experimentation, we compared the performance of RNN and LSTM models. We found that while RNN was simpler and trained faster, it struggled with remembering long-term dependencies. On the other hand, LSTM performed significantly better in terms of accuracy and model stability, especially for capturing the context within financial text.

We experienced a significant performance improvement when we switched to the pretrained Word2Vec embeddings instead of random ones. The Word2Vec ensured that the model improved the understanding of meanings and relationships of words in hotel reviews, thus enhancing generalisation and accuracy of prediction. However, some limitations were noted. During the training, we found indications, possibly because of the relatively small dataset. Also, the slow convergence uncovered in the model occasionally implies the need for more delicate hyperparameter adjustment.

To improve the future implementations of this project, we suggest optimal learning rate, batch size, and depths of networks, among other hyperparameters. Use of techniques such as dropout as regularisation and training a more varied and larger dataset might help avoid overfitting. Investigating sophisticated architectures, including Bidirectional LSTMs or transformer-based models (for instance, BERT), might result in better performance.

Moving forward, this model may be scaled up for live updates of the hotel reviews on platforms of travel platforms or modified for use in a web-based dashboard for hotel managers and customer service staff. Multilingual sentiment analysis or transfer

learning approaches could be further seen as areas of research for improved adaptability to different review pools and domains.