UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning.

(6CS012)

Text Classification

Student Name : Abhishek Rajbhandari

Student Id : 2331518

Group : L6CG15

Mail : A.Rajbhandari@wlv.ac.uk

Tutor : Aatiz Ghimire

Module Leader : Siman Giri

Abstract

In this study, we analyze the use of deep learning techniques for hotel review sentiment analysis. Based on the dataset consisting of 20,491 hotel reviews with ratings from 1 to 5, we implemented and compared two RNN architectures: a Simple RNN and LSTM (Long Short-Term Memory), assisted by Word2Vec to generate better word representations. For preprocessing, we carefully cleaned the texts making use of tokenization, lemmatization, stop words removal, and balancing to counteract class imbalance. Experimental results revealed that the Simple RNN model gave a test accuracy of 61.89% and the LSTM model with Word2Vec embeddings gave a test accuracy of 56.30%. Looking at the models' performances through the confusion matrices and classification reports showed that the models had better performance in predicting extreme sentiments (ratings 1 and 5) rather than neutral ratings. The introduction of Word2Vec embeddings did not help to significantly enhance the task with respect to random embeddings; hence, domain-specific embeddings would perhaps render better results for this task. This research sheds light on the effectiveness of RNNs for multi-class sentiment classification of hotel reviews and points toward avenues for further improvements.

Contents Page

1. Introduction

Sentiment analysis for text classification has grown in importance over time as companies use it to automatically process and extract analyzing information from customer feedback. The hospitality industry should benefit from any system that can truly be used in classifying customer reviews to assess guest satisfaction and recommend areas for improvement.

Conventional text classification techniques merely depend on handcrafted features and rather shallow learning models. Yet, these methods usually cannot consider the sequential nature of text and the complicated semantic relations between words. Deep learning models with the capability to handle sequential data like text have shown promise for sentiment analysis; these include RNNs and LSTM networks.

This problem is of practical importance rather than being of mere academic interest. Hotel companies receive thousands of reviews every day; manual analysis is time-consuming and prone to inconsistency. An automated review classification system may be very valuable for supporting the hotel management; it would allow them to act immediately on negative feedback and on positive experiences.

In earlier text classification and sentiment analysis work, different methods have been attempted, including traditional machine learning methods such as SVMs, NB, and more recently deep learning methods. RNNs, LSTMs, for example, have been shown in studies to capture well the sequential properties of texts, whereas word embeddings, say Word2Vec, improve the performance of models by giving vector representations of words that are semantically meaningful.

This project aims to build upon this previous work by implementing and comparing RNN and LSTM models for hotel review classification, while also investigating the impact of using pre-trained Word2Vec embeddings versus random embeddings.

2. Dataset

2.1. Source and Size

The dataset used in this study is a collection of hotel reviews obtained through the university's AI/ML module resources (Hotel_review.csv). It contains 20,491 reviews along with their corresponding ratings on a scale of 1 to 5, representing the level of guest satisfaction.

2.2. Data Distribution

The original dataset showed a significant class imbalance, with higher ratings being more frequent:

- Rating 1: 1,421 reviews (6.9%)
- Rating 2: 1,793 reviews (8.7%)
- Rating 3: 2,184 reviews (10.7%)
- Rating 4: 6,039 reviews (29.5%)
- Rating 5: 9,054 reviews (44.2%)

This imbalance could lead to biased model predictions toward the majority classes (ratings 4 and 5). Therefore, data balancing techniques were applied during the preprocessing stage to ensure equal representation of all rating classes in the training and testing sets.

2.3. Data Preprocessing

The raw hotel reviews underwent several preprocessing steps to transform them into a clean and structured format suitable for deep learning models:

1. **Text Cleaning**:
    - Conversion to lowercase
    - Removal of URLs, mentions, emojis, and hashtags
    - Removal of punctuation and special characters
    - Elimination of double spaces
2. **Text Normalization**:
    - Removal of English stopwords
    - Removal of short words (fewer than 3 characters)
    - Lemmatization to reduce words to their base forms
3. **Tokenization and Sequence Preparation**:
    - Tokenization of cleaned text
    - Conversion of tokens to sequences
    - Padding of sequences to a uniform length (determined by the 95th percentile of sequence lengths)
4. **Data Balancing**:
    - Creation of balanced datasets for training (800 samples per class) and testing (200 samples per class)
    - Implementation of both undersampling for majority classes and oversampling for minority classes
5. **Data Splitting**:

- Division into training (80%) and testing (20%) sets
- Further splitting of training data to create a validation set (15%)

These preprocessing steps ensured that the data was clean, structured, and balanced for training the deep learning models.

3. Methodology

3.1. Text Preprocessing

The text preprocessing pipeline, as described in the previous section, was designed to clean and normalize the hotel reviews before feeding them into the deep learning models. This process involved several steps:

```
def preprocess_text(text):
  if isinstance(text, str):
    text = lowercase(text)
    text = remove_urls(text)
    text = remove_mentions(text)
    text = remove_emoji(text)
    text = removeunwanted_characters(text)
    text = remove_stopwords(text)
    text = remove_short_words(text)
    text = lemmatize(text)
    return text
  else:
    return ""
```

After preprocessing, the text data was tokenized and converted to sequences using TensorFlow's Tokenizer:

```
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
```

To ensure uniform input dimensions for the neural networks, the sequences were padded to a maximum length determined by the 95th percentile of sequence lengths:

```
max_len = int(np.percentile(seq_lengths, 95))
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len,
padding='post', truncating='post')
```

```
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len,
padding='post', truncating='post')
```

3.2. Model Architectures

3.2.1. Simple RNN Model

The Simple RNN model was designed with the following architecture:

```
# Model parameters
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 200
rnn_units = 64

model_rnn = Sequential([
   Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_len, embeddings_regularizer=l2(0.001)),

   Bidirectional(SimpleRNN(64, return_sequences=True,
kernel_regularizer=l2(0.001), recurrent_regularizer=l2(0.001))),

   Dropout(0.4),

   GlobalMaxPooling1D(),

   Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
   Dropout(0.4),

   Dense(5, activation='softmax', kernel_regularizer=l2(0.001))
])

optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model_rnn.compile(
   optimizer=optimizer,
   loss='categorical_crossentropy',
   metrics=['accuracy']
)
```

Key components of this architecture:

- An Embedding layer with L2 regularization

- A Bidirectional SimpleRNN layer with 64 units and L2 regularization
- Dropout layers (40%) for regularization
- GlobalMaxPooling1D to reduce the dimensionality of the output
- Dense layers with ReLU and softmax activations

3.2.2. LSTM Model

```
max_words = 10000
max_len = 100
vocab_size = min(max_words, len(tokenizer.word_index) + 1)

model_lstm = Sequential([
   Embedding(input_dim=10000, output_dim=128, input_length=max_len),
   SpatialDropout1D(0.4),

   Bidirectional(LSTM(64, return_sequences=True,
                      kernel_regularizer=l2(0.01),
                      recurrent_dropout=0.2)),
   LayerNormalization(),

   Bidirectional(LSTM(64,
                      kernel_regularizer=l2(0.01),
                      recurrent_dropout=0.2)),
   Dropout(0.6),

   Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
   Dense(5, activation='softmax')
])

model_lstm.compile(
   optimizer='adam',
   loss='categorical_crossentropy',
   metrics=['accuracy', Precision(), Recall()]
)

model_lstm.summary()
```

Key components of this architecture:

- An Embedding layer
- SpatialDropout1D (40%) for regularization of embeddings

- Two stacked Bidirectional LSTM layers with L2 regularization and recurrent dropout
- Layer normalization for stabilizing training
- Dense layers with ReLU and softmax activations

3.2.3. Word2Vec Embeddings

In addition to the models with random embeddings, a model using pre-trained Word2Vec embeddings was implemented:

```
model = Sequential([
    Embedding(
        input_dim=vocab_size,
        output_dim=embedding_dim,
        weights=[embedding_matrix],
        input_length=max_len,
        trainable=False
    ),
    SpatialDropout1D(0.51),
    Bidirectional(LSTM(
        64,
        return_sequences=True,
        kernel_regularizer=l2(0.001),
        recurrent_regularizer=l2(0.001)
    )),
    Dropout(0.5),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(5, activation='softmax')
])
```

The Word2Vec embedding matrix was created from pre-trained vectors with 85.65% word coverage:

```
Vocabulary size: 10001
Found 8566 word vectors out of 10001 tokens. Word coverage: 85.65%
Embedding matrix shape: (10001, 300)
```

3.3. Loss Function and Optimizer

All models were compiled with the categorical cross-entropy loss function, which is appropriate for multi-class classification tasks. The Adam optimizer was used with varying learning rates:

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model_rnn.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

For the Word2Vec model, a fine-tuning approach was implemented, where the embeddings were initially frozen and then made trainable in a second training phase with a lower learning rate:

```
# Initial training with frozen embeddings
model.compile(
    optimizer=Adam(learning_rate=0.00025),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Fine-tuning with trainable embeddings
model.layers[0].trainable = True
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

3.4. Hyperparameters

Key hyperparameters used across the models:

- Learning rates: 0.0001 to 0.00025
- Batch sizes: 32 to 64
- Epochs: 10-20 with early stopping
- Embedding dimensions: 128-300
- RNN/LSTM units: 64-128
- Dropout rates: 0.4-0.6

- L2 regularization factors: 0.001-0.01

4. Experiments and Results

4.1. RNN vs. LSTM Performance

The Simple RNN model achieved a test accuracy of 61.89% on the original test set, showing reasonable performance given the complexity of the 5-class classification task. When evaluated on the balanced test data, the classification report showed:

```
Classification Report for RNN Model on Balanced Training Data:
              precision    recall  f1-score   support
           1       0.74      0.61      0.67       200
           2       0.46      0.41      0.43       200
           3       0.47      0.31      0.37       200
           4       0.44      0.58      0.51       200
           5       0.63      0.82      0.71       200
    accuracy                           0.55      1000
   macro avg       0.55      0.55      0.54      1000
weighted avg       0.55      0.55      0.54      1000
```

The LSTM model with Word2Vec embeddings achieved a test accuracy of 56.30% on the balanced test data:

```
Test Loss: 1.0990
Test Accuracy: 0.5630

Classification Report:
              precision    recall  f1-score   support
           1       0.71      0.71      0.71       200
           2       0.47      0.52      0.49       200
           3       0.48      0.40      0.43       200
           4       0.46      0.47      0.47       200
           5       0.68      0.71      0.70       200
    accuracy                           0.56      1000
   macro avg       0.56      0.56      0.56      1000
weighted avg       0.56      0.56      0.56      1000
```

Both models showed similar overall accuracy on the balanced test data, but with notable differences in their performance across rating classes:

1. The Word2Vec LSTM model achieved more balanced precision and recall scores across classes, particularly for extreme ratings (1 and 5).
2. The RNN model showed higher recall for class 5 (0.82) but suffered from lower recall for class 3 (0.31), indicating challenges in identifying neutral sentiment.
3. Both models achieved their best performance on classes 1 and 5 (extreme sentiments), with F1-scores around 0.70, while struggling with the middle classes (2, 3, and 4) with F1-scores in the 0.40s.

The training curves for the RNN model showed steady improvement, with the final model reaching training accuracy of 73.73% and validation accuracy of 82.20% after 10 epochs.

4.2. Computational Efficiency

Training the models required significant computational resources, with LSTM models being more demanding than Simple RNN models:

Simple RNN Model:

- Training time per epoch: Approximately 20-41 seconds
- Total training time: About 257 seconds (4.3 minutes) for 10 epochs
- Hardware: Google Colab with GPU acceleration
- Memory usage: Lower compared to LSTM model

LSTM Model:

- Training time per epoch: Approximately 146-210 seconds (2.4-3.5 minutes)
- Total training time: About 1,526 seconds (25.4 minutes) for 8 epochs
- Hardware: Google Colab with GPU acceleration
- Memory usage: Higher than RNN due to more complex architecture

Word2Vec LSTM Model:

- Initial training (20 epochs): Approximately 4-10 seconds per epoch
- Fine-tuning (10 epochs): Approximately 4-8 seconds per epoch
- Total training time: About 190 seconds (3.2 minutes)
- Hardware: Google Colab with GPU acceleration

The significant difference in training times between the basic LSTM and Word2Vec LSTM models can be attributed to different batch sizes (64 vs. 32), different

implementations, and potentially different Colab GPU allocations during different training sessions.

These metrics highlight the trade-off between model complexity and computational efficiency. While LSTM models can potentially capture more complex patterns in sequence data, they require substantially more computational resources and training time compared to simpler RNN architectures.

4.3. Training with Different Embeddings

We compared the performance of models using different embedding approaches:

Random Embeddings (Simple RNN):

- Embedding dimension: 200
- Learned during training
- Test accuracy on original test set: 61.89%
- Performance on balanced test set: 55% accuracy

Random Embeddings (LSTM):

- Embedding dimension: 128
- Learned during training
- Training accuracy: Reached 86.16% after 8 epochs
- Validation accuracy: Declined to 43.67%, indicating severe overfitting

Pre-trained Word2Vec Embeddings:

- Embedding dimension: 300
- Google News pre-trained vectors
- Word coverage: 85.65% (8,566 out of 10,001 tokens)
- Initial training with frozen embeddings: Achieved 59.57% validation accuracy
- Fine-tuning with trainable embeddings: Improved to 64.82% validation accuracy
- Test accuracy on balanced test set: 56.30%

The Word2Vec embeddings provided a slight improvement over random embeddings in the balanced test set (56.30% vs. 55%), but the difference was not as dramatic as might be expected. This suggests that while pre-trained embeddings contribute some benefit, they may not fully capture the domain-specific language of hotel reviews.

The two-phase training approach (freezing then fine-tuning) showed clear benefits, with validation accuracy improving by more than 5 percentage points during the fine-tuning

phase. This indicates that adapting pre-trained embeddings to the specific domain is important for optimal performance.

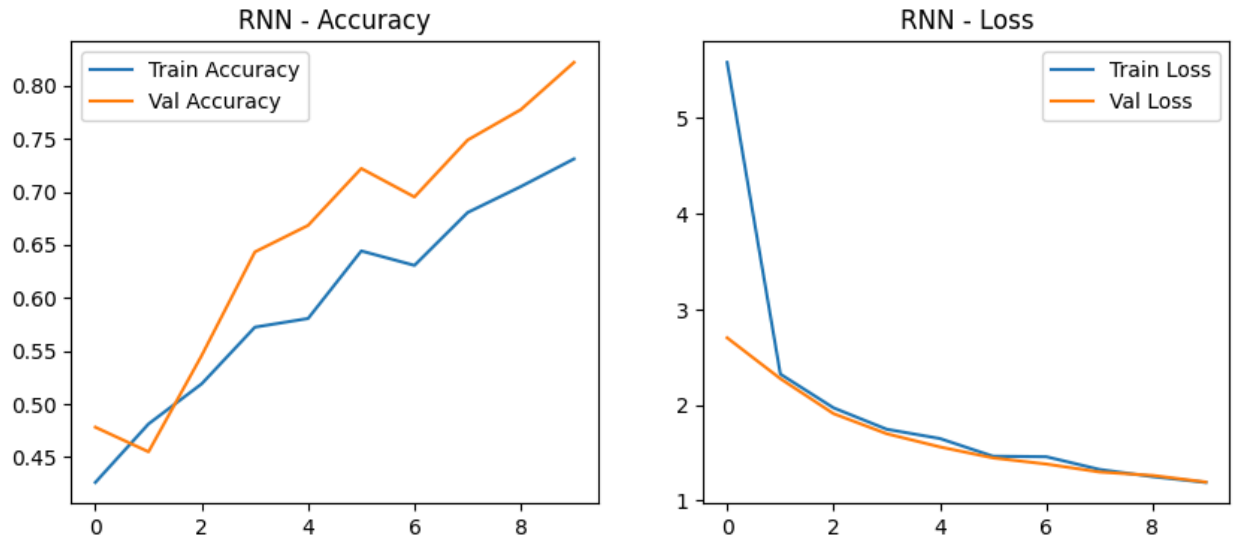
4.4. Model Evaluation

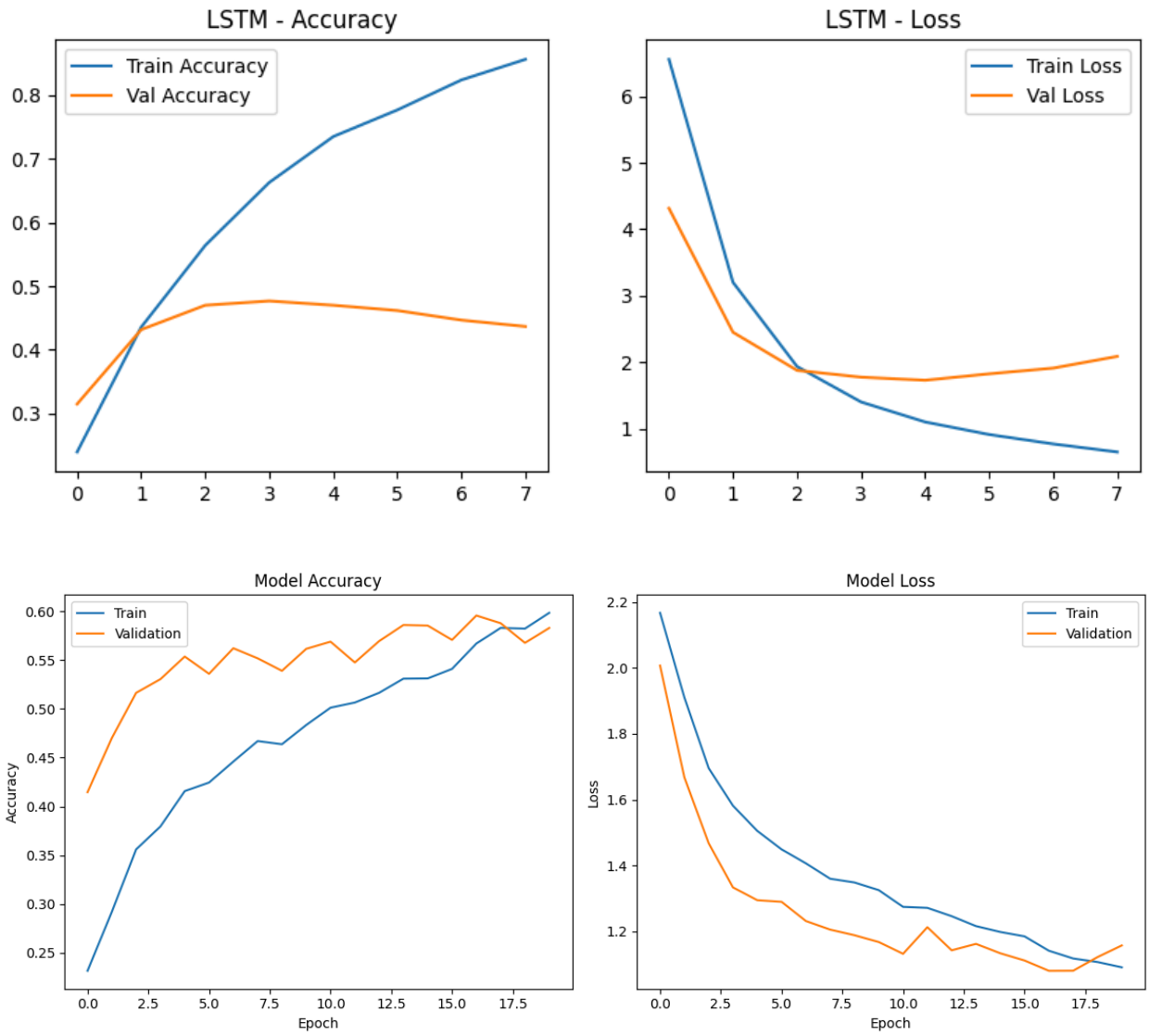
We used several evaluation metrics to assess model performance:

Accuracy:

- Simple RNN on original test set: 61.89%
- Simple RNN on balanced test set: 55.00%
- Word2Vec LSTM on balanced test set: 56.30%

Graphs :





Precision, Recall, and F1-Score: The Word2Vec LSTM model showed varying performance across different rating classes:

- Class 1 (Very Negative): Precision 0.71, Recall 0.71, F1-score 0.71
- Class 2 (Negative): Precision 0.47, Recall 0.52, F1-score 0.49
- Class 3 (Neutral): Precision 0.48, Recall 0.40, F1-score 0.43
- Class 4 (Positive): Precision 0.46, Recall 0.47, F1-score 0.47
- Class 5 (Very Positive): Precision 0.68, Recall 0.71, F1-score 0.70

This pattern, with higher metrics for extreme sentiments (ratings 1 and 5) and lower metrics for middle ratings, was consistent across models.

Confusion Matrix Analysis:

The confusion matrix provides detailed insights into the model's classification performance across different rating classes:

- **Class 1 (Very Negative)**: 143 out of 200 samples (71.5%) were correctly classified. Most misclassifications were to Class 2, indicating the model sometimes underestimated the negativity.
- **Class 2 (Negative)**: 103 out of 200 samples (51.5%) were correctly classified. Misclassifications were split between Classes 1 and 3, showing difficulty in distinguishing between different degrees of negativity.
- **Class 3 (Neutral)**: Only 79 out of 200 samples (39.5%) were correctly classified, making it the most challenging class. Misclassifications were spread across Classes 2 and 4, reflecting the inherent ambiguity of neutral sentiment.
- **Class 4 (Positive)**: 95 out of 200 samples (47.5%) were correctly classified. A significant number (57) were classified as Class 5, indicating the model sometimes overestimated positivity.
- **Class 5 (Very Positive)**: 143 out of 200 samples (71.5%) were correctly classified. Most misclassifications were to Class 4, showing confusion between degrees of positivity.

This pattern is common in sentiment analysis, where extreme sentiments (very positive or very negative) are more distinctive and easier to classify than neutral or moderately positive/negative sentiments. The diagonal dominance for Classes 1 and 5 confirms this observation.

The confusion matrix also reveals an interesting pattern of adjacent-class misclassification—most errors occur between neighboring rating classes (e.g., between 1-2, 2-3, 3-4, or 4-5). This suggests the model captures the ordinal nature of the rating scale, rarely making extreme errors like classifying a rating 1 review as rating 5 (only 1 such case occurred).

5. Conclusion and Future Work

5.1. Key Findings

This study explored the application of recurrent neural networks for sentiment analysis of hotel reviews. Our key findings include:

1. **Model Performance**: The Simple RNN model achieved 61.89% accuracy on the original test set, while the Word2Vec-enhanced LSTM model reached 56.30% accuracy on the balanced test set. Given the 5-class classification task

(compared to the typical binary positive/negative classification), these results demonstrate the viability of deep learning approaches for fine-grained sentiment analysis.

2. **Sentiment Extremity Pattern**: Both models performed significantly better at classifying extreme sentiments (ratings 1 and 5) than neutral or moderate sentiments (ratings 2, 3, and 4). For the Word2Vec model, precision and recall for classes 1 and 5 were around 0.70, while classes 2, 3, and 4 had values in the 0.40-0.50 range. This pattern is consistent with previous research in sentiment analysis and reflects the inherent difficulty in capturing subtle sentiment distinctions.
3. **Word2Vec Embeddings Impact**: Despite using pre-trained Word2Vec embeddings with good word coverage (85.65%), the performance improvement was modest. The Word2Vec LSTM model showed more balanced precision and recall across classes compared to the RNN model but did not achieve higher overall accuracy. This suggests that domain-specific embeddings trained on hotel reviews might be more effective for this task.
4. **Class Imbalance Challenges**: The original dataset exhibited significant class imbalance, with 73.7% of reviews being positive (ratings 4 and 5). Our balancing approach created a more representative distribution but required substantial undersampling of majority classes and oversampling of minority classes, which may have affected model generalization.
5. **Training Dynamics**: Both models showed signs of overfitting, with training metrics continuing to improve while validation metrics plateaued or declined. The two-phase training approach for the Word2Vec model (frozen embeddings followed by fine-tuning) helped mitigate this issue somewhat, with validation accuracy improving from 59.57% to 64.82% during fine-tuning.
6. **Adjacent Class Confusion**: Analysis of the confusion matrix revealed that most misclassifications occurred between adjacent rating classes. This indicates that while the models struggled with precise rating prediction, they rarely made extreme errors (e.g., classifying a rating 1 review as rating 5).

5.2. Limitations

Several limitations affected the performance of our models:

1. **Data Imbalance**: The original dataset had a significant imbalance with 44.2% of reviews being 5-star ratings. Although we balanced the dataset for training and testing, this balancing required undersampling majority classes and oversampling minority classes, which can introduce bias and reduce the effective size of the training data.
2. **Model Complexity vs. Dataset Size**: After balancing, the training dataset contained only 4,000 samples (800 per class), which may not be sufficient for the

complex architectures used, especially the LSTM model with fine-tuned Word2Vec embeddings. The training curves showed clear signs of overfitting, with training accuracy continuing to improve while validation accuracy plateaued or declined.

3. **Domain Mismatch in Embeddings**: While we achieved good word coverage (85.65%) with the Google News pre-trained Word2Vec embeddings, these embeddings were not specifically trained on hotel review data. The language and semantic relationships in hotel reviews may differ significantly from those in news articles, limiting the effectiveness of these general-purpose embeddings.
4. **Contextual Understanding**: Neither RNN nor LSTM models fully capture long-range dependencies and contextual nuances in text. This limitation is evidenced by the models' difficulty in correctly classifying neutral (rating 3) reviews, which often require more subtle understanding of context and tone.
5. **Hyperparameter Optimization**: Due to computational constraints, we conducted limited hyperparameter tuning. A more extensive grid search or Bayesian optimization might have yielded better model configurations.
6. **Feature Engineering**: Our preprocessing pipeline focused on standard text cleaning and normalization techniques. Additional feature engineering, such as n-grams, TF-IDF, or sentiment lexicons specific to hotel reviews, might have improved performance.

5.3. Future Work

Based on our findings and limitations, we suggest several directions for future work:

1. **Domain-Specific Embeddings**: Training word embeddings specifically on a large corpus of hotel reviews could better capture the unique language patterns and semantic relationships in this domain, potentially improving model performance.
2. **Data Augmentation**: Implementing more sophisticated data augmentation techniques beyond simple oversampling, such as back-translation or synonym replacement, could help address the class imbalance issue without reducing the effective size of the training dataset.
3. **Transformer Architectures**: Exploring transformer-based models like BERT, RoBERTa, or DistilBERT, which have shown state-of-the-art performance in text classification tasks. These models better capture bidirectional context and long-range dependencies that are important for sentiment analysis.
4. **Aspect-Based Sentiment Analysis**: Extending the current approach to aspect-based sentiment analysis, where the model identifies sentiment toward

specific aspects of the hotel experience (e.g., cleanliness, service, location) rather than providing an overall rating.

5. **Multi-Task Learning**: Implementing a multi-task learning approach that simultaneously predicts the rating and identifies key aspects mentioned in the review, which could improve the model's understanding of the relationship between aspects and overall sentiment.
6. **Ensemble Methods**: Combining multiple models through ensemble techniques could potentially improve overall classification performance by leveraging the strengths of different architectures.
7. **Explainable AI Techniques**: Incorporating attention mechanisms or other explainable AI techniques to provide insights into which parts of a review most strongly influence the predicted rating, making the model more interpretable and useful for hotel management.
8. **Cross-Domain Transfer Learning**: Investigating how models trained on hotel reviews perform on other domains of customer reviews (e.g., restaurants, products) to understand the transferability of sentiment analysis models across domains.

In conclusion, while our RNN and LSTM models achieved reasonable performance on the hotel review sentiment analysis task, there is significant room for improvement through the application of more advanced techniques and architectures. The insights gained from this study contribute to our understanding of deep learning approaches for multi-class sentiment analysis and provide a foundation for future research in this area.