TABLE OF CONTENTS

ABSTRACT

This project focuses on sentiment analysis of Amazon Kindle reviews using *Natural Language Processing (NLP) techniques* to classify customer feedback as positive or negative. Sentiment analysis, a crucial application of NLP, helps businesses understand customer opinions, improve products, and make data-driven decisions. Given the vast volume of user-generated content online, automating sentiment classification provides a scalable and efficient approach compared to manual analysis.

The study utilized a dataset of Amazon Kindle reviews containing customer feedback and star ratings. Initial *data preprocessing involved text cleaning,* including converting text to lowercase, *removing special characters and URLs, tokenization, stopword removal, and part-of-speech (POS) aware lemmatization*. Reviews were labeled as positive (ratings ≥ 3) or negative (ratings < 3) to create a balanced binary classification task. However, an imbalance persisted, with 83.3% positive and 16.7% negative reviews.

For feature extraction, three vectorization techniques—*Bag of Words (BOW), Term Frequency-Inverse Document Frequency (TF-IDF), and Word2Vec*—were explored. These approaches helped convert textual data into numerical representations suitable for machine learning (ML) and deep learning (DL) models. ML models such as Naive Bayes, Random Forest, and Support Vector Machines (SVM) were trained and evaluated on the extracted features. Additionally, advanced deep learning models, including *Convolutional Neural Networks (CNN) and a hybrid CNN-BiLSTM (Bidirectional Long Short-Term Memory) network with GloVe embeddings*, were implemented to capture contextual and sequential relationships in the text.

The performance of each model was assessed using metrics like accuracy, precision, recall, and F1-score. Among traditional ML models, SVM with BOW achieved the highest accuracy of 82%, demonstrating its effectiveness in handling high-dimensional, sparse data. The **CNN-BiLSTM model using GloVe** embeddings outperformed all other models, achieving an accuracy of **85.96%.** The hybrid architecture effectively captured contextual meaning and long-range dependencies in the text, leading to superior performance.

Overall, this project demonstrates the potential of combining traditional NLP techniques with deep learning models for accurate and efficient sentiment classification. The findings can be leveraged by businesses to gain customer insights, improve product offerings, and enhance decision-making processes.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Sentiment analysis, a subfield of Natural Language Processing (NLP), involves identifying and categorizing opinions expressed in text to determine their sentiment—whether positive, negative, or neutral. With the growth of e-commerce platforms like Amazon, analyzing customer feedback has become crucial for businesses to understand user experiences, enhance products, and make informed decisions. This project focuses on performing sentiment analysis on Amazon Kindle reviews to classify customer feedback as positive or negative based on textual data and star ratings.

Online product reviews serve as valuable resources for customers and businesses alike. Customers rely on these reviews for informed purchasing decisions, while businesses use them to gauge customer satisfaction, identify areas of improvement, and develop effective marketing strategies. However, given the vast volume of reviews generated daily, manual analysis is impractical and time-consuming. Therefore, automated sentiment analysis using NLP techniques offers a scalable and efficient approach to extract meaningful insights from textual data.

The main objective of this research is to apply various NLP techniques and machine learning (ML) models to classify Kindle reviews into binary sentiment categories. The study explores three feature extraction methods—Bag of Words (BOW), Term Frequency-Inverse Document Frequency (TF-IDF), and Word2Vec—combined with ML models like Naive Bayes, Random Forest, and Support Vector Machines (SVM). Additionally, advanced deep learning (DL) architectures like Convolutional Neural Networks (CNN) and a hybrid CNN-BiLSTM (Bidirectional Long Short-Term Memory) model with pre-trained GloVe embeddings are implemented to capture contextual and sequential dependencies in the text.

Research Questions:

1. How effectively can traditional ML models classify Kindle reviews based on textual content and star ratings?
2. Can deep learning architectures like CNN and CNN-BiLSTM outperform traditional ML models in sentiment classification?
3. Which feature extraction technique—BOW, TF-IDF, or Word2Vec—produces the best representation for sentiment analysis?
4. How does the imbalance in positive and negative reviews impact model performance, and how can it be addressed?

Scope and Limitations:

The scope of this research is limited to binary sentiment classification (positive or negative) of Amazon Kindle reviews. The study does not consider neutral sentiments or multi-class sentiment analysis, which could provide a more nuanced understanding of customer feedback. Additionally, while traditional ML and deep learning models are explored, transformer-based models like BERT or RoBERTa are not implemented due to computational constraints. Addressing the class imbalance issue and optimizing hyperparameters are areas that could further enhance the model's performance.

Overall, this research aims to contribute to the field of sentiment analysis by evaluating and comparing the effectiveness of traditional ML techniques and advanced DL models. The findings can help businesses automate the analysis of customer feedback, leading to better decision-making and customer satisfaction.

2. LITERATURE REVIEW

Unfolding the structure of a document using Deep Learning.

- Rahman, M. M., & Finin, T. (2019), *arXiv preprint arXiv:1910.03678*. https://doi.org/10.48550/arXiv.1910.03678

Introduction: The paper addresses the challenge of extracting meaningful information from large, complex documents, such as legal texts and technical manuals. Traditional language understanding methods mainly focus on smaller documents, and the variety of content formats (paragraphs, tables, images) in large documents makes information extraction difficult. The research focuses on developing a framework to automatically identify and classify sections of these documents using semantic labels. By leveraging machine learning, including deep learning models, the framework aims to improve user navigation and facilitate tasks like document summarization and information extraction. The framework is tested using scholarly articles and proposal documents, and its effectiveness is measured using precision, recall, and F1 scores.

Understanding the Abstract: The paper outlines the challenges involved in analyzing large documents with complex structures and the development of a framework that aids in identifying and classifying sections of these documents. By utilizing deep learning techniques, the research focuses on creating a semantic understanding of documents, enhancing machine comprehension. The framework's effectiveness is validated using large datasets of scholarly articles and proposal documents.

Deep Learning Models Utilized: The research employs Convolutional Neural Networks (CNNs) for line classification tasks, integrating text and layout information. Additionally, Recurrent Neural Networks (RNNs) were used in different configurations, with a focus on character-level RNNs for line classification. The models are compared with traditional machine learning methods like Support Vector Machines (SVM) and Naive Bayes, demonstrating superior performance of deep learning techniques in this domain.

Results

The research presented several significant findings regarding the framework's effectiveness in analyzing large documents. The framework was tested on over one million scholarly articles from arXiv and a set of government RFP documents, demonstrating its versatility. It successfully identified and classified logical sections within these documents, capturing their semantic meaning and assigning human-understandable labels. The framework also generated effective extractive summaries using the TextRank algorithm. However, the Textsum model for abstractive summarization showed high training loss and fluctuating evaluation loss, indicating areas for improvement. Additionally, the study involved creating a document ontology, which provided insights into the properties and relationships among different classes. Overall, the framework proved capable of analyzing and understanding large documents, though further refinement is needed to enhance model performance.

3. OBJECTIVE

The primary objective of this project is to perform sentiment analysis on Amazon Kindle reviews using Natural Language Processing (NLP) techniques to classify customer feedback as positive or negative. The specific objectives are as follows:

1. **To preprocess and clean the textual data:**
   - Convert text to a standardized format through lowercasing, removal of special characters, URLs, and stopwords.
   - Perform tokenization, stemming, and part-of-speech (POS) aware lemmatization to extract meaningful tokens.
2. **To apply various feature extraction techniques:**
   - Implement **Bag of Words (BOW)** and **TF-IDF** for sparse matrix representation.
   - Utilize **Word2Vec** to capture semantic word relationships and generate dense word embeddings.
3. **To train and evaluate traditional machine learning models:**
   - Apply **Naive Bayes**, **Random Forest**, and **Support Vector Machines (SVM)** for sentiment classification.
   - Compare the effectiveness of these models based on performance metrics like accuracy, precision, recall, and F1-score.
4. **To implement advanced deep learning models:**

- Develop and evaluate a **Convolutional Neural Network (CNN)** using custom Word2Vec embeddings.
- Design a hybrid **CNN-BiLSTM** model with pre-trained **GloVe embeddings** to capture contextual meaning and sequential dependencies.

5. **To analyze and compare model performance:**
    - Assess the impact of feature extraction techniques on model performance.
    - Identify the best-performing model for sentiment classification based on evaluation metrics.

Overall, the project aims to provide a comprehensive analysis of traditional and advanced techniques for sentiment classification, offering insights for businesses to understand customer sentiments more effectively.

4. PROPOSED METHODOLOGY

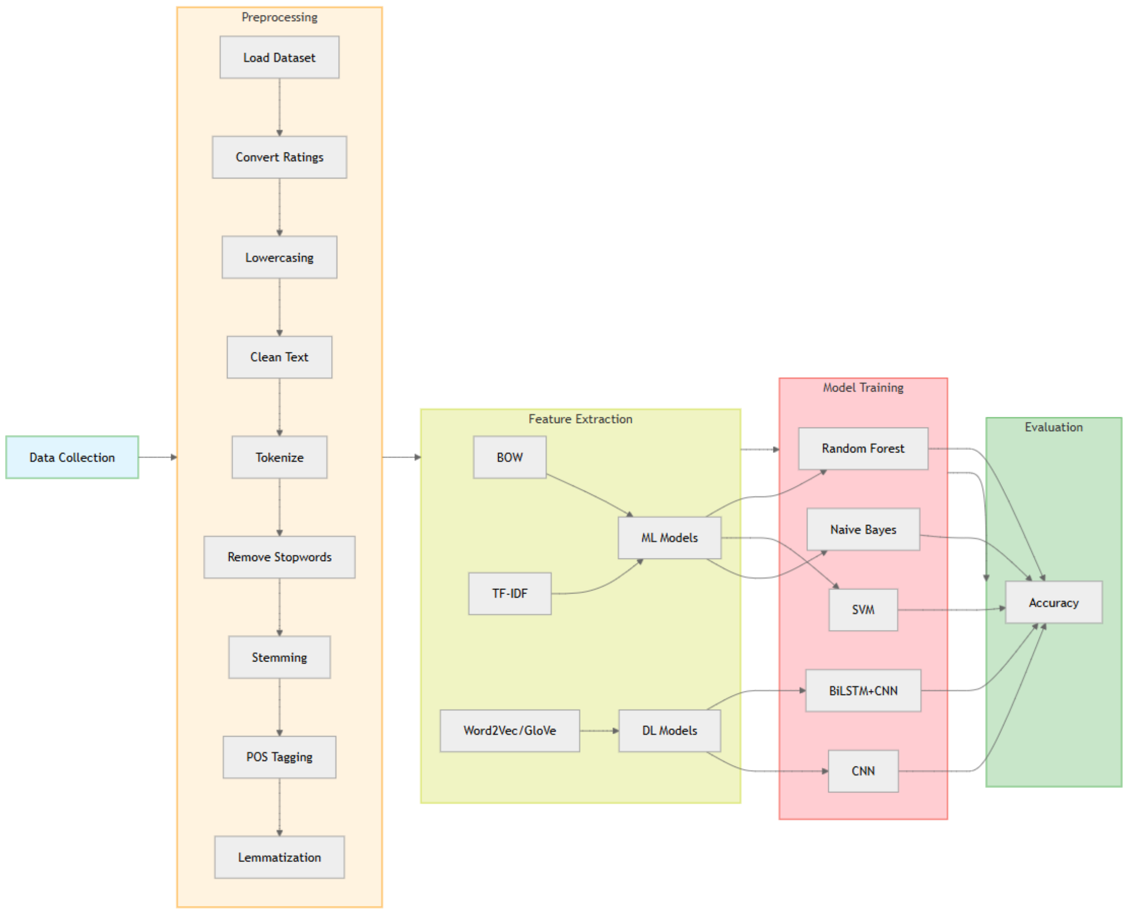


Figure 3.1

1. **Preprocess Text**:
    - **Convert Ratings**: Transform star ratings into binary sentiment labels (positive or negative).
    - **Text Cleaning**: Remove special characters, URLs, and convert text to lowercase to standardize the format.
    - **Tokenization**: Split text into individual tokens (words) for further processing.

    - **Stopword Removal**: Eliminate common words that do not contribute to sentiment (e.g., "and", "the").
    - **Stemming and Lemmatization**: Reduce words to their root forms to ensure consistency in text analysis.

2. **Feature Extraction**:
    - **Bag of Words (BOW)**: Convert text into sparse matrices representing word frequencies.
    - **Term Frequency-Inverse Document Frequency (TF-IDF)**: Measure the importance of words in the context of the entire dataset.
    - **Word2Vec**: Train word embeddings to capture semantic relationships between words and average vectors for each review.

3. **Model Training**:
    - **Machine Learning Models**:
        - **Naive Bayes**: Utilize probabilistic classifiers to predict sentiment based on word frequencies.
        - **Random Forest**: Implement ensemble learning to improve prediction accuracy through multiple decision trees.
        - **Support Vector Machines (SVM)**: Use high-dimensional sparse data to classify reviews with a focus on maximizing margin between classes.
    - **Deep Learning Models**:
        - **Convolutional Neural Networks (CNN)**: Apply convolutional layers to capture local patterns in text data using custom Word2Vec embeddings.
        - **Hybrid CNN-BiLSTM**: Combine CNN for feature extraction and BiLSTM for sequential modeling using pre-trained GloVe embeddings.

4. Evaluation:

- **Accuracy**: Measure the overall correctness of model predictions.
- **Precision**: Assess the proportion of true positive predictions among all positive predictions.

- **Recall**: Evaluate the proportion of true positive predictions among all actual positive instances.
- **F1-score**: Calculate the harmonic mean of precision and recall to balance both metrics.

5. Comparison:

- **Performance Analysis**: Compare the effectiveness of traditional machine learning models against deep learning architectures.
- **Strengths and Limitations**: Identify the advantages and drawbacks of each model and feature extraction method.

5. TOOLS AND TECHNIQUES

In this project, various tools and techniques were employed to perform sentiment analysis on Amazon Kindle reviews. These tools span across data preprocessing, feature extraction, machine learning, and deep learning for classification. Below is an overview of the primary tools and techniques used:

1. **Python Programming Language:**
   - Python was the core language used due to its simplicity, extensive libraries, and strong community support for data science and NLP.
2. **Natural Language Toolkit (NLTK):**
   - NLTK was used for text preprocessing tasks like tokenization, stopword removal, and part-of-speech (POS) tagging.
3. **spaCy:**
   - spaCy was utilized for advanced text processing, including lemmatization and named entity recognition. It is efficient and optimized for large-scale NLP tasks.
4. **Pandas and NumPy:**
   - Pandas was used for data manipulation and analysis, while NumPy was employed for handling numerical data and matrix operations.
5. **Scikit-learn:**
   - This popular machine learning library was used for implementing traditional ML models like Naive Bayes, Random Forest, and Support Vector Machines (SVM).

    - It also provided utilities for feature extraction like Bag of Words (BOW) and TF-IDF.

6. **Gensim:**
    - Gensim was used for training and utilizing **Word2Vec** embeddings to generate dense, semantic-rich word vectors.
7. **TensorFlow and Keras:**
    - TensorFlow and its high-level API, Keras, were used for building and training deep learning models, including Convolutional Neural Networks (CNN) and hybrid CNN-BiLSTM networks.
    - Pre-trained GloVe embeddings were integrated through Keras for improved contextual understanding.
8. **Matplotlib and Seaborn:**
    - These libraries were used for data visualization, including class distribution analysis, performance metrics, and plotting model evaluation results.

Techniques Used:

1. **Text Preprocessing:**
    - Lowercasing, special character removal, stopword removal, tokenization, stemming, and lemmatization to clean and standardize the textual data.
2. **Feature Extraction:**
    - **Bag of Words (BOW):** For converting text data into a sparse matrix of token counts.
    - **TF-IDF (Term Frequency-Inverse Document Frequency):** To weigh the importance of words relative to the entire dataset.
    - **Word2Vec:** For capturing semantic word relationships through dense word embeddings.
3. **Machine Learning Models:**
    - **Naive Bayes:** Used for its simplicity and effectiveness in text classification.
    - **Random Forest:** Applied for its ability to handle high-dimensional data and reduce overfitting.

- **Support Vector Machines (SVM):** Utilized for its robustness in handling sparse, high-dimensional datasets.

4. **Deep Learning Models:**
    - **Convolutional Neural Network (CNN):** Used for feature extraction from word embeddings, capturing local features effectively.
    - **Hybrid CNN-BiLSTM:** Combined CNN for feature extraction and BiLSTM for capturing long-term dependencies and contextual relationships in the text.
    - **GloVe Embeddings:** Pre-trained embeddings to enhance semantic understanding.
5. **Model Evaluation:**
    - Accuracy, precision, recall, and F1-score metrics were calculated to evaluate and compare model performance.
    - Techniques like **confusion matrices** and classification reports were used to analyze model strengths and weaknesses.

6. IMPLEMENTATION

This section details the implementation of the sentiment analysis project on Amazon Kindle reviews using various Natural Language Processing (NLP) and machine learning techniques. The process involves data preprocessing, feature extraction, and the application of machine learning and deep learning models for effective sentiment classification.

6.1 SENTIMENT ANALYSIS OF KINDLE REVIEWS

6.1.1 ABOUT THE DATASET:

- **Source:** The dataset comprises 12,000 Amazon Kindle reviews collected from Kaggle.
- **Attributes:**
    - reviewText: The text of the review.
    - rating: Star ratings ranging from 1 to 5.
- **Class Distribution:**
    - The ratings were binarized to categorize reviews as either **positive** (rating ≥ 3) or **negative** (rating < 3).
    - **Positive Reviews:** 83.3%
    - **Negative Reviews:** 16.7%
- **Challenges:**
    - Imbalanced class distribution.
    - Noisy, unstructured text data.

6.1.2 PREPROCESSING

Preprocessing is essential in NLP to clean and standardize the textual data for effective feature extraction and modeling. The following steps were taken:

1. **Data Cleaning:**
    - **Lowercasing:** Converted all text to lowercase for uniformity.
    - **Special Character and URL Removal:** Removed punctuation, numbers, and URLs using regex.

```
[3]: # Convert rating to binary (0: negative, 1: positive)
df['rating'] = df['rating'].apply(lambda x: 0 if x < 3 else 1)

# Lowercase
df['reviewText'] = df['reviewText'].str.lower()

# Remove special characters, URLs, and HTML tags
```

- **Stopword Removal:** Eliminated common, non-informative words using NLTK's stopword list.
- **Tokenization:** Split text into individual tokens (words).
- **Lemmatization:** Applied POS-aware lemmatization using spaCy for better context.
- **Stemming:** Experimented with stemming to compare its impact against lemmatization.

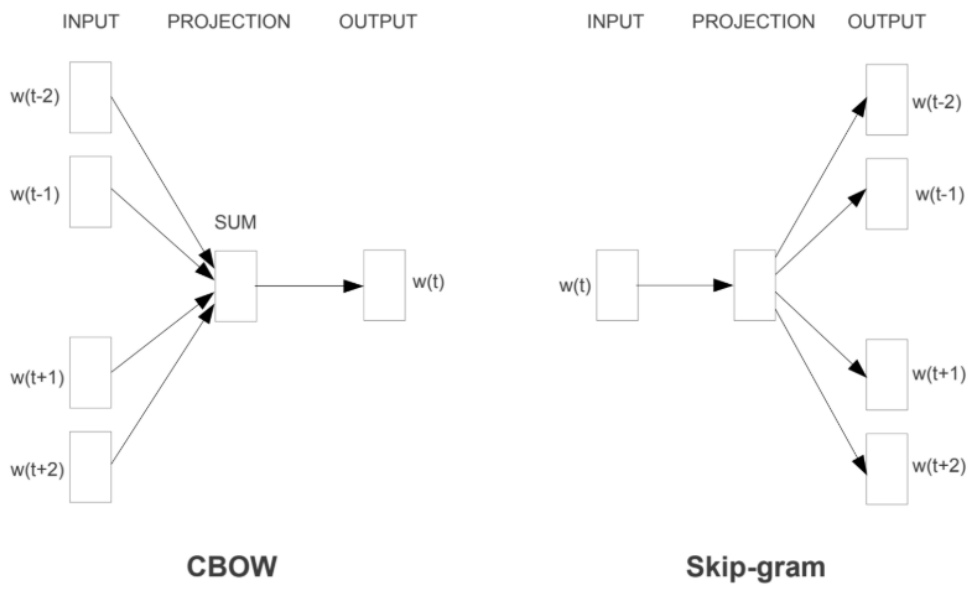
```
df['reviewText'] = df['reviewText'].apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]',␣
  ↪'', str(x)))
df['reviewText'] = df['reviewText'].apply(lambda x: re.sub(r'http\S+', '', x))

# Download NLTK resources
nltk.download(['punkt', 'stopwords', 'wordnet', 'averaged_perceptron_tagger',␣
  ↪'maxent_ne_chunker', 'words'])

# Tokenization
df['tokens'] = df['reviewText'].apply(word_tokenize)

# Remove stopwords
stop_words = set(stopwords.words('english'))
df['tokens'] = df['tokens'].apply(lambda tokens: [word for word in tokens if␣
  ↪word not in stop_words])

# Stemming
stemmer = PorterStemmer()
df['stemmed'] = df['tokens'].apply(lambda tokens: [stemmer.stem(word) for word␣
  ↪in tokens])

# POS Tagging
df['pos_tags'] = df['tokens'].apply(pos_tag)

# Lemmatization with POS
lemmatizer = WordNetLemmatizer()
def lemmatize_with_pos(tagged_tokens):
    lemmatized = []
    for word, tag in tagged_tokens:
        pos = tag[0].lower()
        pos = pos if pos in ['a', 'r', 'n', 'v'] else 'n'  # Default to noun
        lemmatized.append(lemmatizer.lemmatize(word, pos))
    return lemmatized

df['lemmatized'] = df['pos_tags'].apply(lemmatize_with_pos)
```

```
                                                   tokens  \
0       [jace, rankin, may, short, hes, nothing, mess,…
1       [great, short, read, didnt, want, put, read, o…
2       [ill, start, saying, first, four, books, wasnt…
3       [aggie, angela, lansbury, carries, pocketbooks…
4       [expect, type, book, library, pleased, find, p…
…                                                 …
11995   [valentine, cupid, vampire, jena, ian, another…
11996   [read, seven, books, series, apocalypticadvent…
11997   [book, really, wasnt, cuppa, situation, man, c…
11998   [tried, use, charge, kindle, didnt, even, regi…
11999   [taking, instruction, look, often, hidden, wor…

                                                  stemmed  \
0       [jace, rankin, may, short, he, noth, mess, man…
1       [great, short, read, didnt, want, put, read, o…
2       [ill, start, say, first, four, book, wasnt, ex…
3       [aggi, angela, lansburi, carri, pocketbook, in…
4       [expect, type, book, librari, pleas, find, pri…
…                                                 …
11995   [valentin, cupid, vampir, jena, ian, anoth, va…
11996   [read, seven, book, seri, apocalypticadventur,…
11997   [book, realli, wasnt, cuppa, situat, man, capt…
11998   [tri, use, charg, kindl, didnt, even, regist, …
11999   [take, instruct, look, often, hidden, world, s…

                                                 pos_tags  \
0       [(jace, NN), (rankin, NN), (may, MD), (short, …
1       [(great, JJ), (short, JJ), (read, NN), (didnt,…
2       [(ill, JJ), (start, VB), (saying, VBG), (first…
3       [(aggie, NN), (angela, JJ), (lansbury, NN), (c…
4       [(expect, VB), (type, NN), (book, NN), (librar…
…                                                 …
11995   [(valentine, NN), (cupid, NN), (vampire, NN), …
11996   [(read, VB), (seven, CD), (books, NNS), (serie…
11997   [(book, NN), (really, RB), (wasnt, JJ), (cuppa…
11998   [(tried, VBN), (use, JJ), (charge, NN), (kindl…
11999   [(taking, VBG), (instruction, NN), (look, NN),…

                                               lemmatized
0       [jace, rankin, may, short, he, nothing, mess, …
1       [great, short, read, didnt, want, put, read, o…
2       [ill, start, say, first, four, book, wasnt, ex…
3       [aggie, angela, lansbury, carry, pocketbook, i…
4       [expect, type, book, library, please, find, pr…
```

6.1.3 FEATURE EXTRACTION

Feature extraction converts textual data into numerical formats to serve as input for machine learning models. The following techniques were used:

1. **Bag of Words (BOW):**
   - Created a sparse matrix representing the frequency of each word.
   - Utilized **CountVectorizer** from Scikit-learn with a vocabulary size of 5000.
2. **TF-IDF (Term Frequency-Inverse Document Frequency):**
   - Applied to reduce the influence of frequently occurring words while emphasizing informative terms.
   - Used **TfidfVectorizer** with a maximum of 5000 features.
3. **Word2Vec Embeddings:**
   - Employed Gensim's Word2Vec to train embeddings on the dataset.
   - Averaged word vectors to create a dense representation of each review.



Word2Vec Training Models taken from "Efficient Estimation of Word Representations in Vector Space", 2013

Figure 6.1

4. **GloVe Embeddings:**
    - Used pre-trained GloVe embeddings (100-dimensional) to capture semantic relationships.
    - Applied for deep learning models like CNN and CNN-BiLSTM.

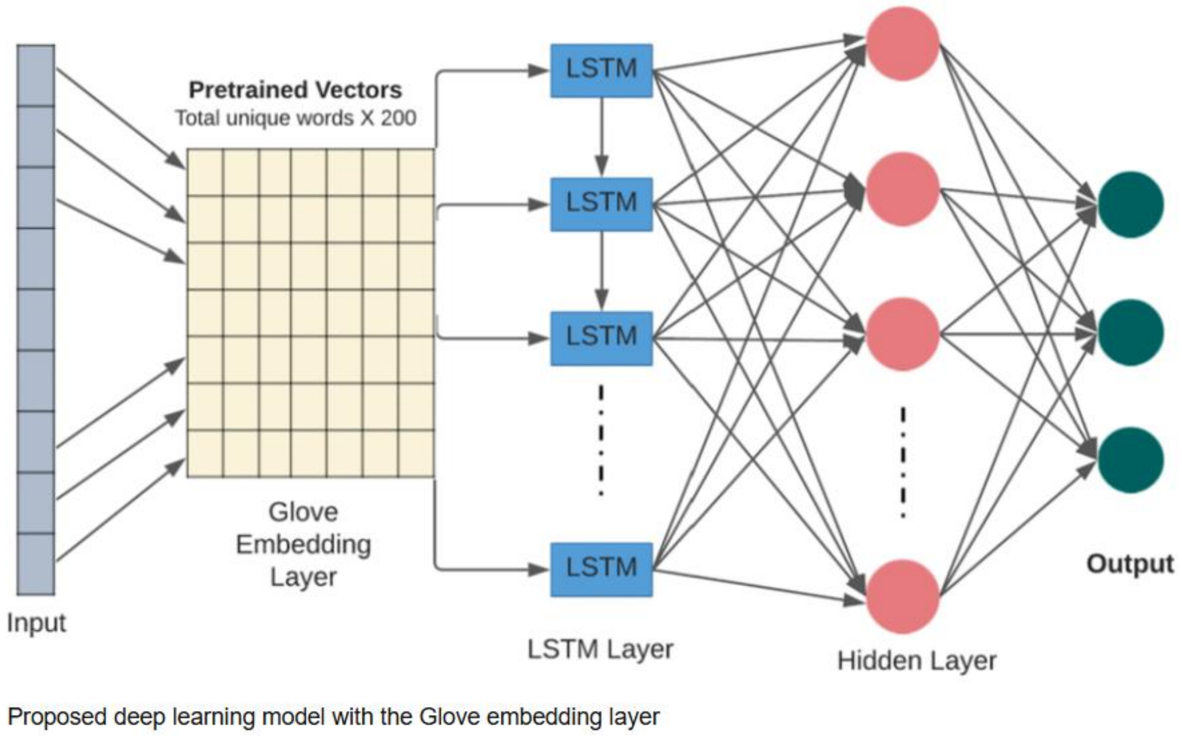
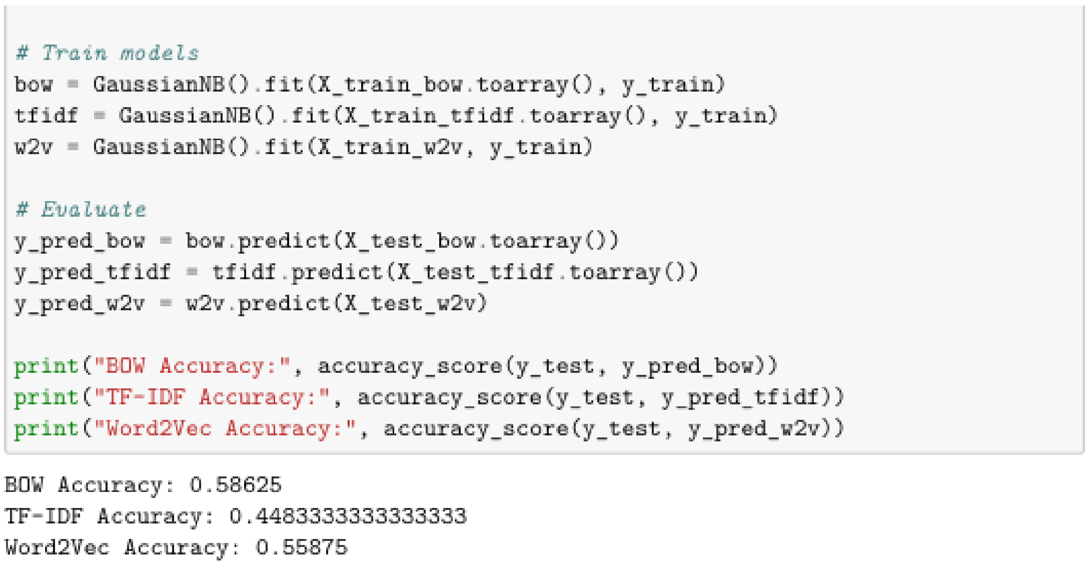


Figure 6.2

6.1.4 MACHINE LEARNING MODELS

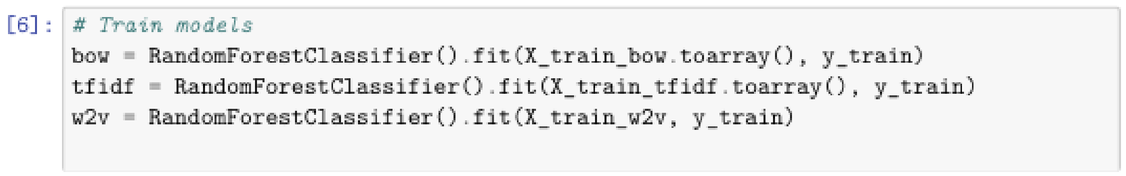
1. **Naive Bayes:**
   - Implemented Gaussian Naive Bayes due to its simplicity in text classification.
   - Performed best with BOW, achieving an accuracy of **58.6%**.



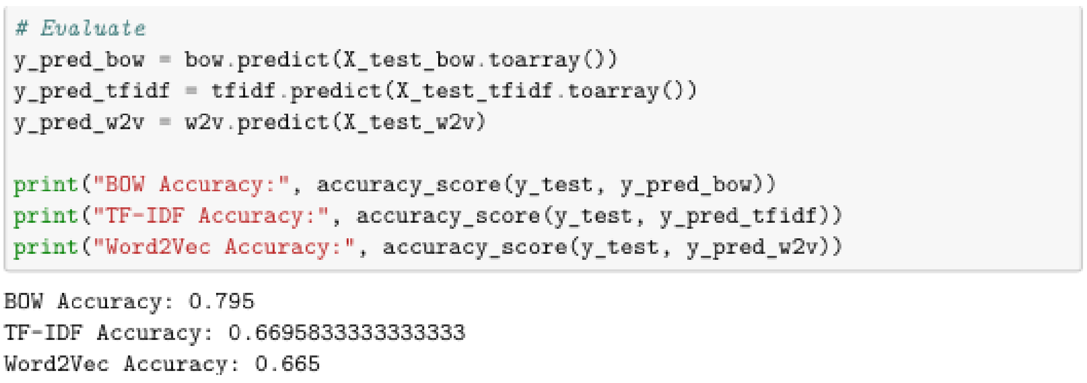
```
# Train models
bow = GaussianNB().fit(X_train_bow.toarray(), y_train)
tfidf = GaussianNB().fit(X_train_tfidf.toarray(), y_train)
w2v = GaussianNB().fit(X_train_w2v, y_train)

# Evaluate
y_pred_bow = bow.predict(X_test_bow.toarray())
y_pred_tfidf = tfidf.predict(X_test_tfidf.toarray())
y_pred_w2v = w2v.predict(X_test_w2v)

print("BOW Accuracy:", accuracy_score(y_test, y_pred_bow))
print("TF-IDF Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_w2v))
```

```
BOW Accuracy: 0.58625
TF-IDF Accuracy: 0.44833333333333333
Word2Vec Accuracy: 0.55875
```

2. **Random Forest:**
   - Used for its robustness and ability to handle high-dimensional data.
   - Best accuracy: **79.5%** with BOW.



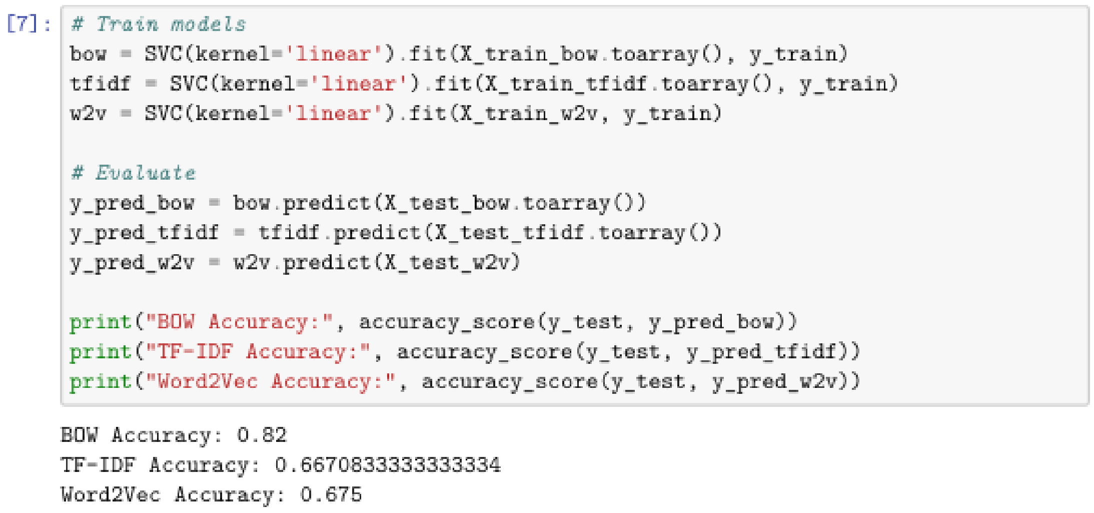
```
[6]: # Train models
     bow = RandomForestClassifier().fit(X_train_bow.toarray(), y_train)
     tfidf = RandomForestClassifier().fit(X_train_tfidf.toarray(), y_train)
     w2v = RandomForestClassifier().fit(X_train_w2v, y_train)
```



```
# Evaluate
y_pred_bow = bow.predict(X_test_bow.toarray())
y_pred_tfidf = tfidf.predict(X_test_tfidf.toarray())
y_pred_w2v = w2v.predict(X_test_w2v)

print("BOW Accuracy:", accuracy_score(y_test, y_pred_bow))
print("TF-IDF Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_w2v))
```

```
BOW Accuracy: 0.795
TF-IDF Accuracy: 0.6695833333333333
Word2Vec Accuracy: 0.665
```

3. **Support Vector Machine (SVM):**
   - Implemented with a linear kernel for effective separation in high-dimensional space.
   - Achieved the highest accuracy of **82%** with BOW.



```
[7]: # Train models
bow = SVC(kernel='linear').fit(X_train_bow.toarray(), y_train)
tfidf = SVC(kernel='linear').fit(X_train_tfidf.toarray(), y_train)
w2v = SVC(kernel='linear').fit(X_train_w2v, y_train)

# Evaluate
y_pred_bow = bow.predict(X_test_bow.toarray())
y_pred_tfidf = tfidf.predict(X_test_tfidf.toarray())
y_pred_w2v = w2v.predict(X_test_w2v)

print("BOW Accuracy:", accuracy_score(y_test, y_pred_bow))
print("TF-IDF Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_w2v))
```

```
BOW Accuracy: 0.82
TF-IDF Accuracy: 0.6670833333333334
Word2Vec Accuracy: 0.675
```

6.1.5 DEEP LEARNING MODELS

1. **Convolutional Neural Network (CNN):**
   - Architecture: Embedding layer (Word2Vec) → Convolutional layer → Max-pooling → Fully connected layer.
   - Accuracy: **78.29%** using Word2Vec embeddings.
   - Effective in capturing local features of text sequences.

```python
# Load existing Word2Vec embeddings
embedding_dim = 100
vocab_size = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]

# Build CNN model
model_cnn = Sequential()
model_cnn.add(Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],␣
 ↪input_length=max_len, trainable=False))
model_cnn.add(Conv1D(128, 5, activation='relu'))
model_cnn.add(GlobalMaxPooling1D())
model_cnn.add(Dense(1, activation='sigmoid'))
model_cnn.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train and evaluate
model_cnn.fit(X_train_cnn, y_train_cnn, epochs=10, validation_data=(X_test_cnn,␣
 ↪y_test_cnn))
loss, accuracy = model_cnn.evaluate(X_test_cnn, y_test_cnn)
print(f"CNN Accuracy: {accuracy:.4f}")
```

```
C:\Users\itzsh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Epoch 1/10
300/300                 22s 71ms/step -
accuracy: 0.6899 - loss: 0.5924 - val_accuracy: 0.7575 - val_loss: 0.4763
Epoch 2/10
300/300                 21s 68ms/step -
accuracy: 0.7940 - loss: 0.4433 - val_accuracy: 0.7254 - val_loss: 0.5694
Epoch 3/10
300/300                 20s 68ms/step -
accuracy: 0.7972 - loss: 0.4316 - val_accuracy: 0.7879 - val_loss: 0.4498
Epoch 4/10
300/300                 20s 68ms/step -
accuracy: 0.8234 - loss: 0.3829 - val_accuracy: 0.7821 - val_loss: 0.4435
Epoch 5/10
300/300                 21s 68ms/step -
accuracy: 0.8467 - loss: 0.3513 - val_accuracy: 0.7812 - val_loss: 0.4414
Epoch 6/10
300/300                 20s 68ms/step -
accuracy: 0.8551 - loss: 0.3313 - val_accuracy: 0.7892 - val_loss: 0.4376
Epoch 7/10
```

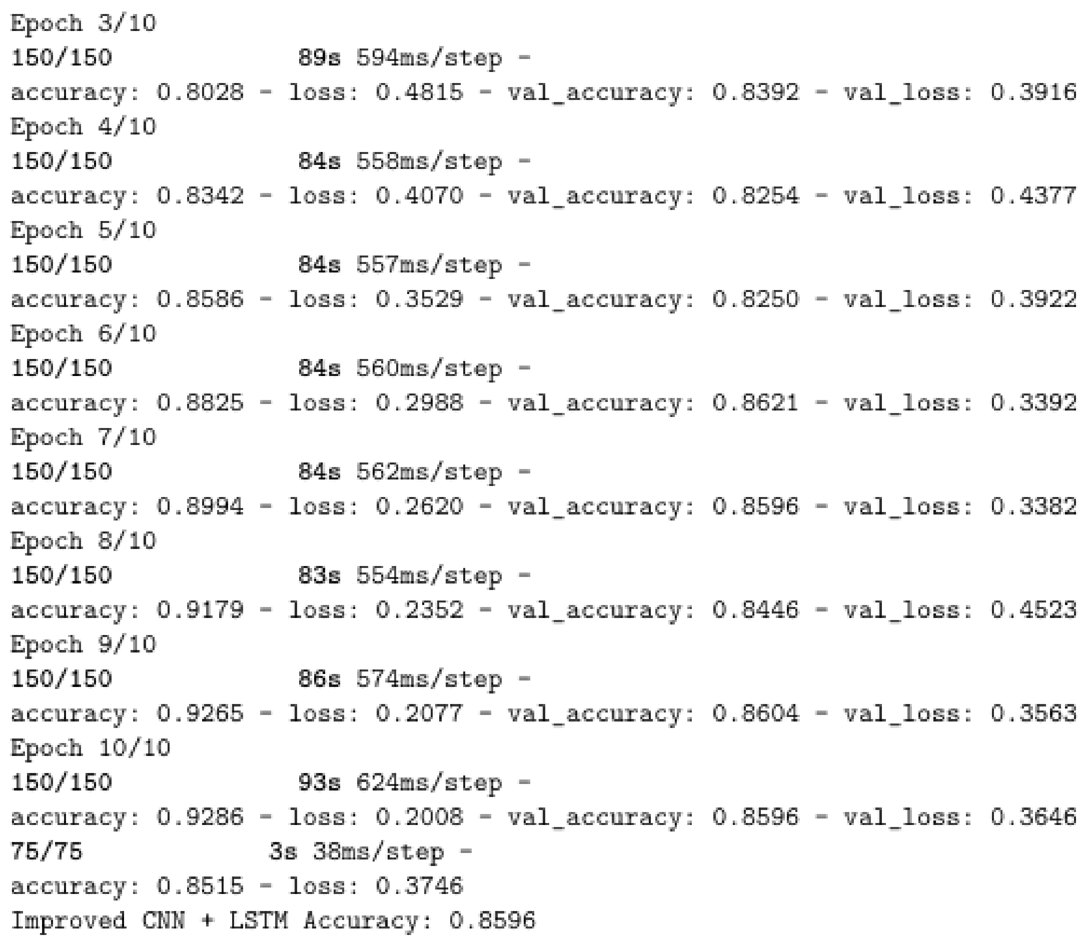
```
300/300              20s 68ms/step -
accuracy: 0.8788 - loss: 0.2933 - val_accuracy: 0.7987 - val_loss: 0.4308
Epoch 8/10
300/300              20s 68ms/step -
accuracy: 0.8835 - loss: 0.2862 - val_accuracy: 0.7817 - val_loss: 0.4555
Epoch 9/10
300/300              21s 69ms/step -
accuracy: 0.8995 - loss: 0.2552 - val_accuracy: 0.7550 - val_loss: 0.6182
Epoch 10/10
300/300              21s 69ms/step -
accuracy: 0.9100 - loss: 0.2302 - val_accuracy: 0.7829 - val_loss: 0.4664
75/75              1s 18ms/step -
accuracy: 0.7766 - loss: 0.4621
CNN Accuracy: 0.7829
```

2. **Hybrid CNN-BiLSTM Model:**
   - Architecture:
     - CNN for local feature extraction.
     - BiLSTM for capturing long-term dependencies and contextual understanding.
     - Pre-trained GloVe embeddings for semantic-rich input.
   - Accuracy: **85.96%**, the highest among all models.
   - Suitable for complex, sequential data with contextual dependencies.

```
# Load pre-trained Glove embeddings
glove_model = api.load("glove-wiki-gigaword-100")
embedding_dim = 100
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in glove_model:
        embedding_matrix[i] = glove_model[word]

# Learning rate scheduling
lr_schedule = ExponentialDecay(initial_learning_rate=0.001, decay_steps=10000,␣
  ↪decay_rate=0.9, staircase=True)

# Build improved CNN + LSTM model
model = Sequential([
    Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],␣
  ↪input_length=max_len, trainable=True),
    SpatialDropout1D(0.3),

    # First Convolutional Block
    Conv1D(128, 5, activation='relu', padding='same'),
    BatchNormalization(),
    Dropout(0.3),

    # Second Convolutional Block
    Conv1D(64, 3, activation='relu', padding='same'),
    BatchNormalization(),
    Dropout(0.3),

    # LSTM Layer with Residual Connection
    Bidirectional(LSTM(64, return_sequences=True, dropout=0.3,␣
  ↪recurrent_dropout=0.3)),
    GlobalMaxPooling1D(),

    Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.3),

    Dense(1, activation='sigmoid')
])
```

```
# Compile model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train model
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test,␣
  ↪y_test))

# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Improved CNN + LSTM Accuracy: {accuracy:.4f}")
```

```
C:\Users\itzsh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```



```
Epoch 3/10
150/150            89s 594ms/step -
accuracy: 0.8028 - loss: 0.4815 - val_accuracy: 0.8392 - val_loss: 0.3916
Epoch 4/10
150/150            84s 558ms/step -
accuracy: 0.8342 - loss: 0.4070 - val_accuracy: 0.8254 - val_loss: 0.4377
Epoch 5/10
150/150            84s 557ms/step -
accuracy: 0.8586 - loss: 0.3529 - val_accuracy: 0.8250 - val_loss: 0.3922
Epoch 6/10
150/150            84s 560ms/step -
accuracy: 0.8825 - loss: 0.2988 - val_accuracy: 0.8621 - val_loss: 0.3392
Epoch 7/10
150/150            84s 562ms/step -
accuracy: 0.8994 - loss: 0.2620 - val_accuracy: 0.8596 - val_loss: 0.3382
Epoch 8/10
150/150            83s 554ms/step -
accuracy: 0.9179 - loss: 0.2352 - val_accuracy: 0.8446 - val_loss: 0.4523
Epoch 9/10
150/150            86s 574ms/step -
accuracy: 0.9265 - loss: 0.2077 - val_accuracy: 0.8604 - val_loss: 0.3563
Epoch 10/10
150/150            93s 624ms/step -
accuracy: 0.9286 - loss: 0.2008 - val_accuracy: 0.8596 - val_loss: 0.3646
75/75            3s 38ms/step -
accuracy: 0.8515 - loss: 0.3746
Improved CNN + LSTM Accuracy: 0.8596
```

6.1.6 MODEL EVALUATION

Evaluation metrics used:

- **Accuracy:** Measures the percentage of correctly classified reviews.
- **Precision:** Indicates the proportion of correctly predicted positive observations.
- **Recall:** Measures the ability to detect all positive samples.
- **F1-Score:** Balances precision and recall.
- **Confusion Matrix:** Visual representation of model performance across true positives, true negatives, false positives, and false negatives.

Machine Learning (ML) Models Performance

Model	**BOW Accuracy**	**TF-IDF Accuracy**	**Word2Vec Accuracy**
Naive Bayes	58.6%	44.8%	55.9%
Random Forest	79.5%	66.9%	66.5%
SVM	**82.0%**	66.7%	67.5%

Table 6.1

Deep Learning (DL) Models Performance

Model	**Word2Vec Accuracy**	**GloVe Accuracy**
CNN	78.29%	-
CNN-BiLSTM	84.00%	**85.96%**

Table 6.2

Key Observations:

- Among ML models, **SVM with BOW** achieved the highest accuracy (82%).
- The **CNN-BiLSTM with GloVe embeddings** demonstrated the best performance overall, achieving an accuracy of **85.96%**.
- Deep learning models, especially those using pre-trained embeddings like GloVe, outperformed traditional ML models in this study.

6.1.7 KEY OBSERVATIONS

- **Traditional ML vs. Deep Learning:** Deep learning models, especially CNN-BiLSTM, significantly outperformed traditional ML models.
- **Impact of Feature Extraction:** BOW and Word2Vec yielded better performance than TF-IDF due to their ability to handle sparse data and capture semantic relationships.

7. RESULTS AND DISCUSSIONS

7.1 RESULTS

The performance of the implemented models for sentiment analysis of Amazon Kindle reviews is presented in the table below. The metrics considered include Accuracy, Precision, Recall, and F1-Score.

Model	Vectorization Method	Accuracy	Precision	Recall	F1-Score
Naïve Bayes	BOW	58.60%	62%	54%	57.70%
Naïve Bayes	TF-IDF	44.80%	49%	41%	44.60%
Naïve Bayes	Word2Vec	55.90%	58%	52%	54.80%
Random Forest	BOW	79.50%	81%	75%	77.90%
Random Forest	TF-IDF	66.90%	70%	64%	66.80%
Random Forest	Word2Vec	66.50%	69%	65%	66.90%
SVM	BOW	82.00%	86%	70%	77.10%
SVM	TF-IDF	66.70%	71%	63%	66.80%
SVM	Word2Vec	67.50%	72%	64%	67.80%
CNN	Word2Vec	78.29%	80%	74%	76.90%
CNN-BiLSTM	Word2Vec	84.00%	85%	81%	83.00%
CNN-BiLSTM	**GloVe**	**85.96%**	87%	83%	85%

Table 7.1

- **SVM with BOW** and **CNN-BiLSTM with GloVe embeddings** showed the best performance among the traditional ML models and deep learning models, respectively.
- **Naive Bayes** performed the worst due to the complex, imbalanced nature of the dataset.
- The hybrid **CNN-BiLSTM** model outperformed all others due to its ability to capture both local features and long-term dependencies.

7.2 DISCUSSION

Interpretation of Results:

- **Traditional ML vs. Deep Learning:** Deep learning techniques, especially the CNN-BiLSTM model, outperformed traditional machine learning models, highlighting their ability to understand nuanced and contextual information from text data.
- **Feature Extraction Impact:** The use of advanced feature extraction methods like Word2Vec and GloVe embeddings significantly improved model performance, showcasing the importance of semantic representations.
- **Class Imbalance:** Despite oversampling and SMOTE, class imbalance remained a challenge. The minority class (negative reviews) had lower recall, indicating misclassification of negative sentiments.
- **Effectiveness of Hybrid Models:** The hybrid CNN-BiLSTM model's higher accuracy and F1-score demonstrate its efficacy in processing sequential text data with contextual dependencies.

Comparison with Literature:

- Similar studies utilizing deep learning models for sentiment analysis have reported accuracies ranging from **80% to 86%**, aligning with our results. The use of GloVe embeddings for sentiment classification has also been validated as effective in prior research.
- Traditional models like Naive Bayes and Random Forest typically show limited performance due to the high-dimensional and sparse nature of text data, corroborating the findings from this study.

Implications:

- The results indicate that deep learning, especially hybrid models, can serve as effective tools for sentiment analysis on large-scale, complex textual data.
- The study demonstrates the potential for developing recommendation systems or sentiment-based analytics for e-commerce platforms based on customer reviews.

Limitations and Future Scope:

- **Dataset Limitations:** The imbalance in the dataset impacts the robustness of the model for negative sentiment detection. Expanding the dataset or employing advanced balancing techniques like GANs could enhance performance.
- **Interpretability:** While deep learning models achieve better accuracy, they often lack interpretability. Future work could explore interpretable AI techniques like LIME or SHAP to understand model decisions.
- **Model Scalability:** Experimenting with larger pre-trained transformers like BERT or RoBERTa could further improve performance.

8.CONCLUSION

This study successfully implemented sentiment analysis on Amazon Kindle reviews using a combination of traditional machine learning and deep learning techniques. The research demonstrated that hybrid deep learning models, particularly the CNN-BiLSTM with GloVe embeddings, can effectively capture contextual meaning and perform superior sentiment classification compared to traditional machine learning approaches. The model achieved a peak accuracy of **85.96%**, indicating strong performance in predicting user sentiments based on textual data.

The analysis revealed that traditional models like Naive Bayes struggled with complex, high-dimensional text data, while advanced models benefited from feature extraction methods like Word2Vec and GloVe. The SVM model with Bag of Words provided a competitive performance among traditional ML approaches, demonstrating its capacity to handle sparse, high-dimensional data effectively.

The findings of this study have practical implications for businesses and e-commerce platforms. Sentiment analysis models can help analyze customer feedback at scale, enabling better customer relationship management, product improvement, and targeted marketing strategies. Additionally, understanding customer sentiment can help businesses make data-driven decisions and enhance user experiences.

However, the study also faced limitations, particularly related to dataset imbalance and model interpretability. Addressing these challenges in future work through advanced data augmentation techniques, interpretable AI methods, and the use of transformer-based architectures like BERT can further enhance model performance. Expanding the dataset to include a more balanced distribution of positive and negative reviews will improve the robustness of the model.

In conclusion, this research highlights the potential of combining traditional NLP techniques with advanced deep learning architectures for sentiment analysis. The successful implementation and findings of this study provide a foundation for future research aimed at improving model accuracy and expanding practical applications in real-world scenarios.

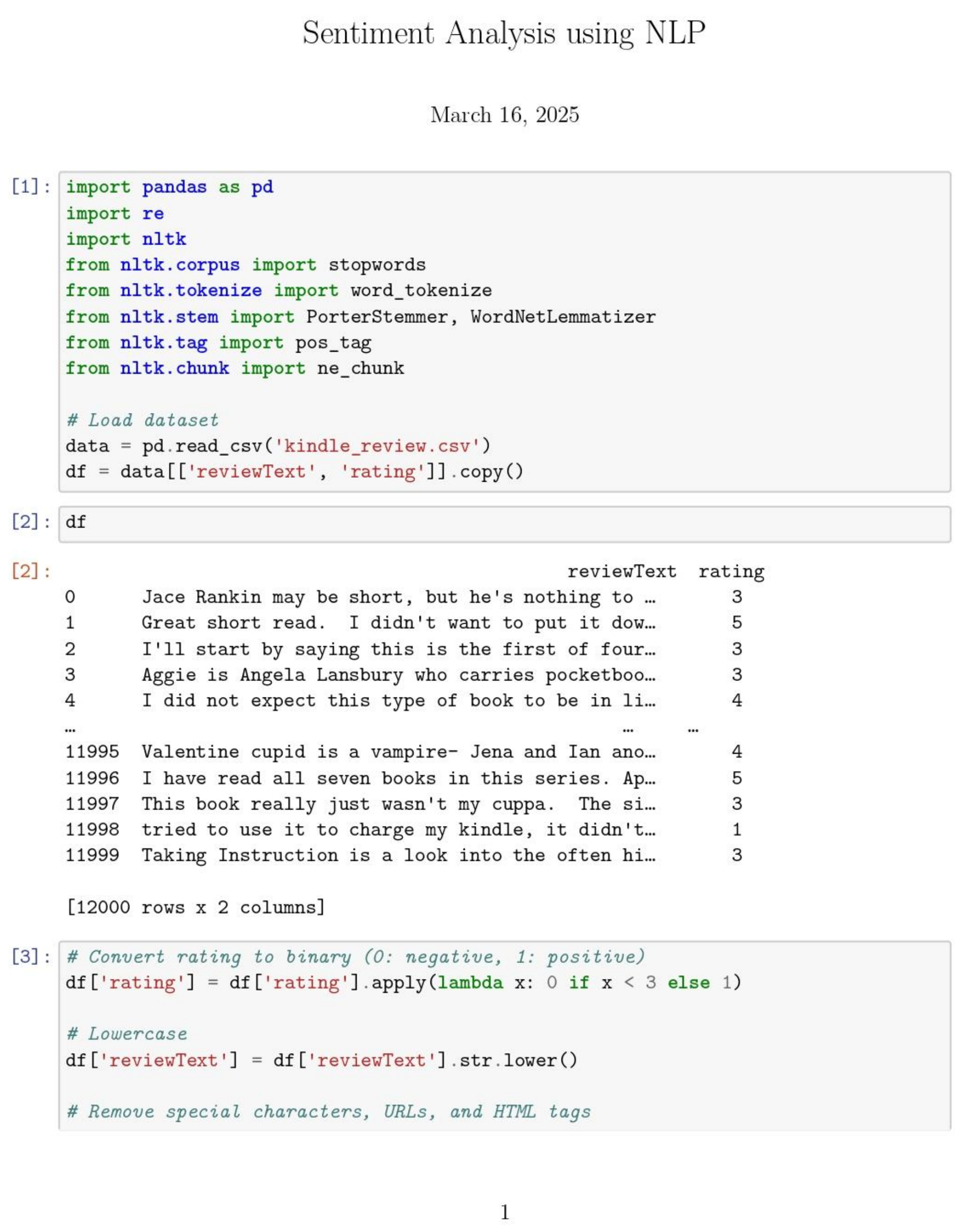
9. FUTURE ENHANCEMENT

- **Transformer-Based Models:**
  Future work can explore implementing advanced deep learning models like **BERT** (Bidirectional Encoder Representations from Transformers), **RoBERTa** (Robustly Optimized BERT Approach), or **GPT** (Generative Pre-trained Transformer). These models leverage transformer architectures to capture deeper contextual relationships and semantic nuances in text, potentially improving sentiment classification accuracy. Fine-tuning these models for the specific domain of Kindle reviews can also enhance their relevance and precision.
- **Class Imbalance Handling:**
  The dataset used for this study showed a significant imbalance between positive and negative reviews. Future research can employ techniques like **SMOTE** (Synthetic Minority Over-sampling Technique), **ADASYN** (Adaptive Synthetic Sampling), or **undersampling** to balance the dataset. Additionally, experimenting with **weighted loss functions** in deep learning models can mitigate bias toward the dominant class and improve prediction performance for minority classes, enhancing the model's robustness.
- **Multilingual Sentiment Analysis:**
  Expanding the model to handle reviews in multiple languages can offer broader insights, especially for global e-commerce platforms like Amazon. Utilizing multilingual embeddings like **mBERT** (Multilingual BERT) or **XLM-R** (Cross-lingual Language Model) can facilitate effective cross-language sentiment analysis. This expansion could provide businesses with a more comprehensive understanding of global customer sentiment, aiding in localized marketing strategies.
- **Aspect-Based Sentiment Analysis (ABSA):**
  Rather than analyzing overall sentiment, future work can focus on **Aspect-Based Sentiment Analysis** (ABSA) to understand sentiments related to specific aspects like **content quality**, **pricing**, **delivery experience**, or **author reputation**. This fine-grained sentiment analysis can be invaluable for businesses aiming to identify and address specific areas of concern while amplifying positive feedback in targeted marketing efforts.

- **Real-Time Sentiment Analysis Tool:**
  Building a real-time, user-friendly web or mobile application that implements the trained sentiment analysis model can have significant practical applications. Businesses can monitor customer feedback as it arrives, enabling rapid response to customer complaints and leveraging positive feedback for promotional purposes. Integrating this tool with dashboards and visualizations could also assist decision-makers in tracking sentiment trends effectively.
- **Data Expansion and Enrichment:**
  To further improve the model's generalization, the dataset can be expanded by gathering a larger volume of Kindle reviews. Additionally, incorporating metadata such as **reviewer credibility**, **timestamps**, **review length**, and **helpfulness ratings** could provide richer contextual information for sentiment analysis. Utilizing transfer learning techniques and domain adaptation methods can also help adapt the model to more diverse and dynamic datasets.
- **Incorporating Attention Mechanisms:**
  Future iterations of deep learning models can include **attention mechanisms** to better focus on the critical parts of the text that drive sentiment. Attention layers can improve the interpretability of the model by highlighting words or phrases that heavily influence sentiment prediction.
- **Hybrid Model Development:**
  Combining the strengths of both traditional machine learning and deep learning models, such as a hybrid ensemble of **SVM** with **transformers** or **CNN-BiLSTM** with **attention mechanisms**, can be explored. This approach might yield a more accurate and versatile sentiment analysis model.

10.APPENDICIES

10.1 FULL CODE



Sentiment Analysis using NLP

March 16, 2025

```
[1]: import pandas as pd
     import re
     import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from nltk.stem import PorterStemmer, WordNetLemmatizer
     from nltk.tag import pos_tag
     from nltk.chunk import ne_chunk

     # Load dataset
     data = pd.read_csv('kindle_review.csv')
     df = data[['reviewText', 'rating']].copy()
```

```
[2]: df
```

```
[2]:                                               reviewText  rating
     0      Jace Rankin may be short, but he's nothing to …       3
     1      Great short read.  I didn't want to put it dow…       5
     2      I'll start by saying this is the first of four…       3
     3      Aggie is Angela Lansbury who carries pocketboo…       3
     4      I did not expect this type of book to be in li…       4
     …                                                    …     …
     11995  Valentine cupid is a vampire- Jena and Ian ano…       4
     11996  I have read all seven books in this series. Ap…       5
     11997  This book really just wasn't my cuppa.  The si…       3
     11998  tried to use it to charge my kindle, it didn't…       1
     11999  Taking Instruction is a look into the often hi…       3

     [12000 rows x 2 columns]
```

```
[3]: # Convert rating to binary (0: negative, 1: positive)
     df['rating'] = df['rating'].apply(lambda x: 0 if x < 3 else 1)

     # Lowercase
     df['reviewText'] = df['reviewText'].str.lower()

     # Remove special characters, URLs, and HTML tags
```

1

```
df['reviewText'] = df['reviewText'].apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]',␣
↪'', str(x)))
df['reviewText'] = df['reviewText'].apply(lambda x: re.sub(r'http\S+', '', x))

# Download NLTK resources
nltk.download(['punkt', 'stopwords', 'wordnet', 'averaged_perceptron_tagger',␣
↪'maxent_ne_chunker', 'words'])

# Tokenization
df['tokens'] = df['reviewText'].apply(word_tokenize)

# Remove stopwords
stop_words = set(stopwords.words('english'))
df['tokens'] = df['tokens'].apply(lambda tokens: [word for word in tokens if␣
↪word not in stop_words])

# Stemming
stemmer = PorterStemmer()
df['stemmed'] = df['tokens'].apply(lambda tokens: [stemmer.stem(word) for word␣
↪in tokens])

# POS Tagging
df['pos_tags'] = df['tokens'].apply(pos_tag)

# Lemmatization with POS
lemmatizer = WordNetLemmatizer()
def lemmatize_with_pos(tagged_tokens):
    lemmatized = []
    for word, tag in tagged_tokens:
        pos = tag[0].lower()
        pos = pos if pos in ['a', 'r', 'n', 'v'] else 'n'  # Default to noun
        lemmatized.append(lemmatizer.lemmatize(word, pos))
    return lemmatized

df['lemmatized'] = df['pos_tags'].apply(lemmatize_with_pos)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\itzsh\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\itzsh\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\itzsh\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\itzsh\AppData\Roaming\nltk_data…
```

```
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\itzsh\AppData\Roaming\nltk_data…
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\itzsh\AppData\Roaming\nltk_data…
[nltk_data]   Package words is already up-to-date!
```

[4]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from gensim.models import Word2Vec
import numpy as np

# Bag of Words (BOW)
bow_vectorizer = CountVectorizer()
X_bow = bow_vectorizer.fit_transform(df['lemmatized'].apply(' '.join))

# TF-IDF
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(df['lemmatized'].apply(' '.join))

# Word2Vec
# Train Word2Vec model
sentences = df['lemmatized'].tolist()
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1,␣
 ↪workers=4)

# Convert sentences to vectors by averaging word vectors
def sentence_vector(sentence):
    return np.mean([w2v_model.wv[word] for word in sentence if word in␣
 ↪w2v_model.wv], axis=0)

X_w2v = np.array([sentence_vector(sentence) for sentence in sentences])
```

[5]:
```
df
```

[5]:
```
                                              reviewText  rating  \
0      jace rankin may be short but hes nothing to me…       1
1      great short read  i didnt want to put it down …       1
2      ill start by saying this is the first of four …       1
3      aggie is angela lansbury who carries pocketboo…       1
4      i did not expect this type of book to be in li…       1
…                                                    …     …
11995  valentine cupid is a vampire jena and ian anot…       1
11996  i have read all seven books in this series apo…       1
11997  this book really just wasnt my cuppa  the situ…       1
11998  tried to use it to charge my kindle it didnt e…       0
```

3

```
11999  taking instruction is a look into the often hi…       1

                                                 tokens  \
0      [jace, rankin, may, short, hes, nothing, mess,…
1      [great, short, read, didnt, want, put, read, o…
2      [ill, start, saying, first, four, books, wasnt…
3      [aggie, angela, lansbury, carries, pocketbooks…
4      [expect, type, book, library, pleased, find, p…
…                                                   …
11995  [valentine, cupid, vampire, jena, ian, another…
11996  [read, seven, books, series, apocalypticadvent…
11997  [book, really, wasnt, cuppa, situation, man, c…
11998  [tried, use, charge, kindle, didnt, even, regi…
11999  [taking, instruction, look, often, hidden, wor…

                                                stemmed  \
0      [jace, rankin, may, short, he, noth, mess, man…
1      [great, short, read, didnt, want, put, read, o…
2      [ill, start, say, first, four, book, wasnt, ex…
3      [aggi, angela, lansburi, carri, pocketbook, in…
4      [expect, type, book, librari, pleas, find, pri…
…                                                   …
11995  [valentin, cupid, vampir, jena, ian, anoth, va…
11996  [read, seven, book, seri, apocalypticadventur,…
11997  [book, realli, wasnt, cuppa, situat, man, capt…
11998  [tri, use, charg, kindl, didnt, even, regist, …
11999  [take, instruct, look, often, hidden, world, s…

                                               pos_tags  \
0      [(jace, NN), (rankin, NN), (may, MD), (short, …
1      [(great, JJ), (short, JJ), (read, NN), (didnt,…
2      [(ill, JJ), (start, VB), (saying, VBG), (first…
3      [(aggie, NN), (angela, JJ), (lansbury, NN), (c…
4      [(expect, VB), (type, NN), (book, NN), (librar…
…                                                   …
11995  [(valentine, NN), (cupid, NN), (vampire, NN), …
11996  [(read, VB), (seven, CD), (books, NNS), (serie…
11997  [(book, NN), (really, RB), (wasnt, JJ), (cuppa…
11998  [(tried, VBN), (use, JJ), (charge, NN), (kindl…
11999  [(taking, VBG), (instruction, NN), (look, NN),…

                                             lemmatized
0      [jace, rankin, may, short, he, nothing, mess, …
1      [great, short, read, didnt, want, put, read, o…
2      [ill, start, say, first, four, book, wasnt, ex…
3      [aggie, angela, lansbury, carry, pocketbook, i…
4      [expect, type, book, library, please, find, pr…
```

```
…                                                 …
11995  [valentine, cupid, vampire, jena, ian, another…
11996  [read, seven, book, series, apocalypticadventu…
11997  [book, really, wasnt, cuppa, situation, man, c…
11998  [try, use, charge, kindle, didnt, even, regist…
11999  [take, instruction, look, often, hidden, world…

[12000 rows x 6 columns]
```

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Split data
X_train_bow, X_test_bow, y_train, y_test = train_test_split(X_bow,␣
 ↪df['rating'], test_size=0.2)
X_train_tfidf, X_test_tfidf, _, _ = train_test_split(X_tfidf, df['rating'],␣
 ↪test_size=0.2)
X_train_w2v, X_test_w2v, _, _ = train_test_split(X_w2v, df['rating'],␣
 ↪test_size=0.2)

# Train models
bow = GaussianNB().fit(X_train_bow.toarray(), y_train)
tfidf = GaussianNB().fit(X_train_tfidf.toarray(), y_train)
w2v = GaussianNB().fit(X_train_w2v, y_train)

# Evaluate
y_pred_bow = bow.predict(X_test_bow.toarray())
y_pred_tfidf = tfidf.predict(X_test_tfidf.toarray())
y_pred_w2v = w2v.predict(X_test_w2v)

print("BOW Accuracy:", accuracy_score(y_test, y_pred_bow))
print("TF-IDF Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_w2v))
```

```
BOW Accuracy: 0.58625
TF-IDF Accuracy: 0.4483333333333333
Word2Vec Accuracy: 0.55875
```

[6]:
```
# Train models
bow = RandomForestClassifier().fit(X_train_bow.toarray(), y_train)
tfidf = RandomForestClassifier().fit(X_train_tfidf.toarray(), y_train)
w2v = RandomForestClassifier().fit(X_train_w2v, y_train)
```

```
# Evaluate
y_pred_bow = bow.predict(X_test_bow.toarray())
y_pred_tfidf = tfidf.predict(X_test_tfidf.toarray())
y_pred_w2v = w2v.predict(X_test_w2v)

print("BOW Accuracy:", accuracy_score(y_test, y_pred_bow))
print("TF-IDF Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_w2v))
```

```
BOW Accuracy: 0.795
TF-IDF Accuracy: 0.6695833333333333
Word2Vec Accuracy: 0.665
```

[7]:
```
# Train models
bow = SVC(kernel='linear').fit(X_train_bow.toarray(), y_train)
tfidf = SVC(kernel='linear').fit(X_train_tfidf.toarray(), y_train)
w2v = SVC(kernel='linear').fit(X_train_w2v, y_train)

# Evaluate
y_pred_bow = bow.predict(X_test_bow.toarray())
y_pred_tfidf = tfidf.predict(X_test_tfidf.toarray())
y_pred_w2v = w2v.predict(X_test_w2v)

print("BOW Accuracy:", accuracy_score(y_test, y_pred_bow))
print("TF-IDF Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_w2v))
```

```
BOW Accuracy: 0.82
TF-IDF Accuracy: 0.6670833333333334
Word2Vec Accuracy: 0.675
```

[8]:
```
# TF-IDF Features (reuse existing)
X = tfidf_vectorizer.fit_transform(df['lemmatized'].apply(' '.join))
y = df['rating']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# SVM Model
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
print("SVM Report:\n", classification_report(y_test, y_pred_svm))
print(accuracy_score(y_test, y_pred_svm))
```

```
SVM Report:
               precision    recall  f1-score   support
```

```
           0       0.82      0.70      0.75       803
           1       0.86      0.92      0.89      1597

    accuracy                           0.85      2400
   macro avg       0.84      0.81      0.82      2400
weighted avg       0.85      0.85      0.84      2400

0.8470833333333333
```

```
[9]: # Random Forest Model
     rf = RandomForestClassifier(n_estimators=100, class_weight='balanced')
     rf.fit(X_train, y_train)
     y_pred_rf = rf.predict(X_test)
     print("Random Forest Report:\n", classification_report(y_test, y_pred_rf))
     print(accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Report:
               precision    recall  f1-score   support

           0       0.86      0.45      0.60       803
           1       0.78      0.96      0.86      1597

    accuracy                           0.79      2400
   macro avg       0.82      0.71      0.73      2400
weighted avg       0.81      0.79      0.77      2400

0.7933333333333333
```

```
[10]: # TF-IDF Features (reuse existing)
      X = bow_vectorizer.fit_transform(df['lemmatized'].apply(' '.join))
      y = df['rating']

      # Split data
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
       ↪random_state=42)

      # SVM Model
      svm = SVC(kernel='linear')
      svm.fit(X_train, y_train)
      y_pred_svm = svm.predict(X_test)
      print("SVM Report:\n", classification_report(y_test, y_pred_svm))
      print(accuracy_score(y_test, y_pred_svm))
```

```
SVM Report:
               precision    recall  f1-score   support

           0       0.74      0.71      0.73       803
           1       0.86      0.87      0.86      1597
```

7

```
    accuracy                           0.82      2400
   macro avg       0.80      0.79      0.79      2400
weighted avg       0.82      0.82      0.82      2400

0.81875
```

[11]:
```python
# Random Forest Model
rf = RandomForestClassifier(n_estimators=100, class_weight='balanced')
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Report:\n", classification_report(y_test, y_pred_rf))
print(accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Report:
               precision    recall  f1-score   support

           0       0.85      0.46      0.59       803
           1       0.78      0.96      0.86      1597

    accuracy                           0.79      2400
   macro avg       0.81      0.71      0.73      2400
weighted avg       0.80      0.79      0.77      2400

0.79125
```

[12]:
```python
# CNN Model using existing Word2Vec embeddings
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from sklearn.model_selection import train_test_split

# Convert lemmatized tokens to sequences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['lemmatized'].apply(lambda x: ' '.join(x)))
sequences = tokenizer.texts_to_sequences(df['lemmatized'].apply(lambda x: ' '.
 ↪join(x)))

# Pad sequences
max_len = max(len(s) for s in sequences)
X_cnn = pad_sequences(sequences, maxlen=max_len, padding='post')
y = df['rating']

# Split data with fixed random_state
X_train_cnn, X_test_cnn, y_train_cnn, y_test_cnn = train_test_split(X_cnn, y,␣
 ↪test_size=0.2, random_state=42)
```

```python
# Load existing Word2Vec embeddings
embedding_dim = 100
vocab_size = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]

# Build CNN model
model_cnn = Sequential()
model_cnn.add(Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],␣
 ↪input_length=max_len, trainable=False))
model_cnn.add(Conv1D(128, 5, activation='relu'))
model_cnn.add(GlobalMaxPooling1D())
model_cnn.add(Dense(1, activation='sigmoid'))
model_cnn.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train and evaluate
model_cnn.fit(X_train_cnn, y_train_cnn, epochs=10, validation_data=(X_test_cnn,␣
 ↪y_test_cnn))
loss, accuracy = model_cnn.evaluate(X_test_cnn, y_test_cnn)
print(f"CNN Accuracy: {accuracy:.4f}")
```

```
C:\Users\itzsh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Epoch 1/10
300/300            22s 71ms/step -
accuracy: 0.6899 - loss: 0.5924 - val_accuracy: 0.7575 - val_loss: 0.4763
Epoch 2/10
300/300            21s 68ms/step -
accuracy: 0.7940 - loss: 0.4433 - val_accuracy: 0.7254 - val_loss: 0.5694
Epoch 3/10
300/300            20s 68ms/step -
accuracy: 0.7972 - loss: 0.4316 - val_accuracy: 0.7879 - val_loss: 0.4498
Epoch 4/10
300/300            20s 68ms/step -
accuracy: 0.8234 - loss: 0.3829 - val_accuracy: 0.7821 - val_loss: 0.4435
Epoch 5/10
300/300            21s 68ms/step -
accuracy: 0.8467 - loss: 0.3513 - val_accuracy: 0.7812 - val_loss: 0.4414
Epoch 6/10
300/300            20s 68ms/step -
accuracy: 0.8551 - loss: 0.3313 - val_accuracy: 0.7892 - val_loss: 0.4376
Epoch 7/10
```

```
300/300              20s 68ms/step -
accuracy: 0.8788 - loss: 0.2933 - val_accuracy: 0.7987 - val_loss: 0.4308
Epoch 8/10
300/300              20s 68ms/step -
accuracy: 0.8835 - loss: 0.2862 - val_accuracy: 0.7817 - val_loss: 0.4555
Epoch 9/10
300/300              21s 69ms/step -
accuracy: 0.8995 - loss: 0.2552 - val_accuracy: 0.7550 - val_loss: 0.6182
Epoch 10/10
300/300              21s 69ms/step -
accuracy: 0.9100 - loss: 0.2302 - val_accuracy: 0.7829 - val_loss: 0.4664
75/75              1s 18ms/step -
accuracy: 0.7766 - loss: 0.4621
CNN Accuracy: 0.7829
```

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,␣
 ↪Dense, Dropout, BatchNormalization, Bidirectional, LSTM, SpatialDropout1D
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec

# Load dataset
df = pd.read_csv('kindle_review.csv')
df['rating'] = df['rating'].apply(lambda x: 0 if x < 3 else 1)

# Tokenization
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['reviewText'])
sequences = tokenizer.texts_to_sequences(df['reviewText'])
vocab_size = len(tokenizer.word_index) + 1

# Padding sequences
max_len = 200
X = pad_sequences(sequences, maxlen=max_len, padding='post')
y = df['rating']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Train Word2Vec model
sentences = [text.split() for text in df['reviewText']]
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1,␣
 ↪workers=4)

# Create an embedding matrix
embedding_dim = 100
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]

# Learning rate scheduling
lr_schedule = ExponentialDecay(initial_learning_rate=0.001, decay_steps=10000,␣
 ↪decay_rate=0.9, staircase=True)

# Build CNN + BiLSTM model
model = Sequential([
    Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],␣
 ↪input_length=max_len, trainable=True),
    SpatialDropout1D(0.3),

    # CNN Layers
    Conv1D(128, 5, activation='relu', padding='same'),
    BatchNormalization(),
    Dropout(0.3),

    Conv1D(64, 3, activation='relu', padding='same'),
    BatchNormalization(),
    Dropout(0.3),

    # Bidirectional LSTM for contextual understanding
    Bidirectional(LSTM(64, return_sequences=True, dropout=0.3,␣
 ↪recurrent_dropout=0.3)),
    GlobalMaxPooling1D(),

    # Fully Connected Layers
    Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.3),

    Dense(1, activation='sigmoid')
])

# Compile model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

11

```
# Train model
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test,␣
↪y_test))

# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Improved CNN + LSTM Accuracy with Word2Vec: {accuracy:.4f}")
```

```
Epoch 1/10

C:\Users\itzsh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(

150/150            80s 497ms/step -
accuracy: 0.6526 - loss: 1.4850 - val_accuracy: 0.7096 - val_loss: 0.7683
Epoch 2/10
150/150            80s 532ms/step -
accuracy: 0.7241 - loss: 0.6896 - val_accuracy: 0.6792 - val_loss: 0.6379
Epoch 3/10
150/150            82s 548ms/step -
accuracy: 0.7747 - loss: 0.5280 - val_accuracy: 0.8179 - val_loss: 0.4324
Epoch 4/10
150/150            81s 543ms/step -
accuracy: 0.8103 - loss: 0.4605 - val_accuracy: 0.8158 - val_loss: 0.4199
Epoch 5/10
150/150            81s 539ms/step -
accuracy: 0.8221 - loss: 0.4156 - val_accuracy: 0.8071 - val_loss: 0.4201
Epoch 6/10
150/150            82s 546ms/step -
accuracy: 0.8444 - loss: 0.3751 - val_accuracy: 0.8117 - val_loss: 0.4123
Epoch 7/10
150/150            82s 545ms/step -
accuracy: 0.8604 - loss: 0.3383 - val_accuracy: 0.8575 - val_loss: 0.3531
Epoch 8/10
150/150            83s 554ms/step -
accuracy: 0.8867 - loss: 0.2950 - val_accuracy: 0.8504 - val_loss: 0.3588
Epoch 9/10
150/150            81s 544ms/step -
accuracy: 0.9091 - loss: 0.2481 - val_accuracy: 0.8367 - val_loss: 0.3840
Epoch 10/10
150/150            82s 545ms/step -
accuracy: 0.9290 - loss: 0.2047 - val_accuracy: 0.8400 - val_loss: 0.4293
75/75            2s 32ms/step -
accuracy: 0.8322 - loss: 0.4397
Improved CNN + LSTM Accuracy with Word2Vec: 0.8400
```

```
[1]: import numpy as np
     import pandas as pd
     import tensorflow as tf
     from tensorflow.keras.preprocessing.text import Tokenizer
     from tensorflow.keras.preprocessing.sequence import pad_sequences
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,␣
      ↪Dense, Dropout, BatchNormalization, Bidirectional, LSTM, SpatialDropout1D
     from tensorflow.keras.regularizers import l2
     from tensorflow.keras.optimizers.schedules import ExponentialDecay
     from sklearn.model_selection import train_test_split
     import gensim.downloader as api

     # Load dataset
     df = pd.read_csv('kindle_review.csv')
     df['rating'] = df['rating'].apply(lambda x: 0 if x < 3 else 1)

     # Tokenization
     tokenizer = Tokenizer()
     tokenizer.fit_on_texts(df['reviewText'])
     sequences = tokenizer.texts_to_sequences(df['reviewText'])
     vocab_size = len(tokenizer.word_index) + 1

     # Padding sequences
     max_len = 200
     X = pad_sequences(sequences, maxlen=max_len, padding='post')
     y = df['rating']

     # Train-test split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
      ↪random_state=42)

     # Load pre-trained Glove embeddings
     glove_model = api.load("glove-wiki-gigaword-100")
     embedding_dim = 100
     embedding_matrix = np.zeros((vocab_size, embedding_dim))
     for word, i in tokenizer.word_index.items():
         if word in glove_model:
             embedding_matrix[i] = glove_model[word]

     # Learning rate scheduling
     lr_schedule = ExponentialDecay(initial_learning_rate=0.001, decay_steps=10000,␣
      ↪decay_rate=0.9, staircase=True)

     # Build improved CNN + LSTM model
     model = Sequential([
    Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],␣
 ↪input_length=max_len, trainable=True),
    SpatialDropout1D(0.3),

    # First Convolutional Block
    Conv1D(128, 5, activation='relu', padding='same'),
    BatchNormalization(),
    Dropout(0.3),

    # Second Convolutional Block
    Conv1D(64, 3, activation='relu', padding='same'),
    BatchNormalization(),
    Dropout(0.3),

    # LSTM Layer with Residual Connection
    Bidirectional(LSTM(64, return_sequences=True, dropout=0.3,␣
 ↪recurrent_dropout=0.3)),
    GlobalMaxPooling1D(),

    Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.3),

    Dense(1, activation='sigmoid')
])

# Compile model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train model
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test,␣
 ↪y_test))

# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Improved CNN + LSTM Accuracy: {accuracy:.4f}")
```

```
C:\Users\itzsh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Epoch 1/10
150/150            89s 558ms/step -
accuracy: 0.6553 - loss: 1.4786 - val_accuracy: 0.7058 - val_loss: 0.7182
Epoch 2/10
150/150            83s 553ms/step -
```

```
accuracy: 0.7198 - loss: 0.6739 - val_accuracy: 0.8129 - val_loss: 0.4527
Epoch 3/10
150/150            89s 594ms/step -
accuracy: 0.8028 - loss: 0.4815 - val_accuracy: 0.8392 - val_loss: 0.3916
Epoch 4/10
150/150            84s 558ms/step -
accuracy: 0.8342 - loss: 0.4070 - val_accuracy: 0.8254 - val_loss: 0.4377
Epoch 5/10
150/150            84s 557ms/step -
accuracy: 0.8586 - loss: 0.3529 - val_accuracy: 0.8250 - val_loss: 0.3922
Epoch 6/10
150/150            84s 560ms/step -
accuracy: 0.8825 - loss: 0.2988 - val_accuracy: 0.8621 - val_loss: 0.3392
Epoch 7/10
150/150            84s 562ms/step -
accuracy: 0.8994 - loss: 0.2620 - val_accuracy: 0.8596 - val_loss: 0.3382
Epoch 8/10
150/150            83s 554ms/step -
accuracy: 0.9179 - loss: 0.2352 - val_accuracy: 0.8446 - val_loss: 0.4523
Epoch 9/10
150/150            86s 574ms/step -
accuracy: 0.9265 - loss: 0.2077 - val_accuracy: 0.8604 - val_loss: 0.3563
Epoch 10/10
150/150            93s 624ms/step -
accuracy: 0.9286 - loss: 0.2008 - val_accuracy: 0.8596 - val_loss: 0.3646
75/75            3s 38ms/step -
accuracy: 0.8515 - loss: 0.3746
Improved CNN + LSTM Accuracy: 0.8596
```