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
```

```
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
```

```
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]:

```python
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]:

```python
# 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]:

```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       0.86      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]:

```python
# 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
```

```
    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]:

```
# 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]:

```
# 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'])
```

```python
# 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
```