Sentiment Analysis of Amazon Reviews Using Bidirectional LSTM and Word2Vec Embeddings

Junaid Ali

June 10, 2025

Abstract

This report presents a sentiment analysis pipeline for Amazon product reviews using a Bidirectional LSTM neural network with custom-trained Word2Vec word embeddings. The workflow includes data preprocessing, embedding training, model construction, and comprehensive evaluation. The approach leverages the semantic capabilities of Word2Vec and the contextual power of BiLSTM, resulting in robust sentiment classification performance.

1 Introduction

Sentiment analysis is a core task in Natural Language Processing (NLP), aiming to automatically determine the polarity (positive or negative sentiment) of text. This project utilizes deep learning, specifically Bidirectional LSTM (BiLSTM) networks, and Word2Vec embeddings to classify Amazon product reviews. Word2Vec allows for the transformation of words into dense vector representations based on context, which enhances the model's ability to understand semantic relationships within the text [**?**, **?**].

2 Data Loading and Exploration

The dataset consists of Amazon product reviews, each labeled with a sentiment polarity. Data is loaded using `pandas`, focusing on the `polarity` (label) and `title` (review text) columns. Initial exploration ensures data integrity and provides insight into the review content.

Listing 1: Data Loading

```
import pandas as pd
df = pd.read_csv('amazon_reviews.csv', usecols=['polarity', 'title'])
print(df.head())
```

3 Data Cleaning

Text data is cleaned to improve model performance. Cleaning involves lowercasing all text and removing special characters using regular expressions. This step reduces noise and standardizes input, which is essential for effective LSTM training.

Listing 2: Text Cleaning

```
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text)
    return text

df['clean_text'] = df['title'].apply(clean_text)
```

4 Label Encoding

Sentiment labels are mapped to binary values: 0 for negative and 1 for positive. This encoding is necessary for binary classification with neural networks.

Listing 3: Label Encoding

```
df['label'] = df['polarity'].map({1: 0, 2: 1})
```

5 Tokenization and Padding

The cleaned text is tokenized using Keras' `Tokenizer`, converting each word to an integer index. Sequences are padded to a uniform length to ensure consistent input size for the neural network.

Listing 4: Tokenization and Padding

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_words = 5000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['clean_text'])
sequences = tokenizer.texts_to_sequences(df['clean_text'])
X = pad_sequences(sequences, maxlen=max_len, padding='post', truncating='post
y = df['label'].values
```

6 Train-Test Split

The dataset is split into training and testing subsets using an 80-20 ratio. This ensures unbiased evaluation of the model's generalization ability.

Listing 5: Train-Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
```

7 Word Embedding with Word2Vec

Unlike GloVe, Word2Vec is a predictive model that learns word embeddings by using a shallow neural network to predict context words, capturing semantic similarity through local context [**?**, **?**]. In this project, Word2Vec embeddings are trained on the review corpus using Gensim.

Listing 6: Training Word2Vec Embeddings

```
from gensim.models import Word2Vec

sentences = [text.split() for text in df['clean_text']]
embed_dim = 100
w2v_model = Word2Vec(sentences, vector_size=embed_dim, window=5, min_count=1,
```

An embedding matrix is constructed by mapping each word in the tokenizer's vocabulary to its corresponding Word2Vec vector.

Listing 7: Embedding Matrix Creation

```
import numpy as np
embedding_matrix = np.zeros((max_words, embed_dim))
for word, i in tokenizer.word_index.items():
    if i < max_words and word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]
```

8 Model Architecture

The model consists of:

- **Embedding Layer**: Initialized with the Word2Vec embedding matrix (non-trainable).
- **Bidirectional LSTM Layers**: Two stacked BiLSTM layers to capture context from both directions.
- **Dropout Layer**: Reduces overfitting by randomly dropping units during training.
- **Dense Output Layer**: Single neuron with sigmoid activation for binary sentiment classification.

Listing 8: Model Architecture

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense, Bidirect

model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=embed_dim,
                    weights=[embedding_matrix], input_length=max_len, trainab
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

9 Model Compilation and Training

The model is compiled with binary cross-entropy loss and the Adam optimizer, which offers fast convergence and robustness. The model is trained for 18 epochs, with accuracy monitored on both training and validation sets.

Listing 9: Model Compilation and Training

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
history = model.fit(X_train, y_train, epochs=18, batch_size=128, validation_d
```

10 Evaluation and Results

After training, the model is evaluated using accuracy, ROC-AUC, average precision, and a detailed classification report. These metrics provide insight into the model's predictive performance and class-wise effectiveness.

Listing 10: Model Evaluation

```
from sklearn.metrics import accuracy_score, roc_auc_score, average_precision_

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_pred_prob))
print("Average Precision Score:", average_precision_score(y_test, y_pred_prob
print("Classification Report:\n", classification_report(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("PR-AUC Score: ", average_precision_score(y_test, y_pred_prob))
```

Reported Results

```
Test Accuracy: 0.834
ROC AUC: 0.912
Average Precision Score: 0.911
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.83      0.83      1250
           1       0.83      0.84      0.83      1250

    accuracy                           0.83      2500
   macro avg       0.83      0.83      0.83      2500
weighted avg       0.83      0.83      0.83      2500

F1 Score: 0.8500704860950158
PR-AUC Score:  0.9166504969532622
```

11 Discussion

Strengths of the Approach:

- **Word2Vec Embedding**: By learning word vectors from the actual review corpus, the model captures domain-specific semantic relationships, improving sentiment classification [**?**, **?**].
- **Bidirectional LSTM**: This architecture captures context from both directions, which is crucial for understanding sentiment in natural language.
- **Thorough Preprocessing**: Cleaning and tokenization ensure high-quality input, which is vital for LSTM performance.
- **Comprehensive Evaluation**: Multiple metrics (accuracy, ROC-AUC, precision, recall, F1) provide a full picture of model performance.
- **Reproducibility and Clarity**: The code is modular, well-structured, and easy to follow, supporting reproducibility and further experimentation.

12 Conclusion

The presented pipeline demonstrates a robust approach to sentiment analysis using deep learning and Word2Vec embeddings. The model achieves strong results on Amazon product reviews, validating the effectiveness of combining domain-specific embeddings with BiLSTM architectures. This approach can be extended or improved with more advanced embeddings or architectures as needed.