Amazon Product Review Sentiment Analysis Report

This report details the Jupyter Notebook, which implements an Amazon product review sentiment analysis system. The notebook covers data preprocessing, model training using a Linear Support Vector Classifier (SVC), and deployment of an interactive web interface using Gradio.

1. Project Objective

The main objective of this project is to develop a machine learning model that can classify the sentiment of Amazon product reviews as Positive, Neutral, or Negative. The project also aims to create an easily accessible web interface for users to input reviews and receive real-time sentiment predictions.

2. Setup and Library Imports

The notebook begins by importing necessary libraries and downloading NLTK data packages.

- **NLTK Downloads**: punkt_tab, stopwords, and wordnet are downloaded from NLTK, which are essential for text tokenization, stop word removal, and lemmatization, respectively.

```
import nltk
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
```

- **Core Libraries**: Key libraries for data manipulation, text processing, and machine learning are imported:

```
import pandas as pd
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, accuracy_score
```

```
import re
```

3. Data Loading and Preprocessing

The sentiment analysis model relies on a clean and well-processed text dataset.

- **Data Loading**: The Reviews.csv dataset is loaded into a pandas DataFrame.

  ```
  df = pd.read_csv('/content/Reviews.csv')
  ```

- **Missing Value Handling**: Rows with any missing values are removed from the DataFrame to ensure data quality.

  ```
  df.dropna(inplace=True)
  ```

- **Text Preprocessing Function**: A preprocess_text function is defined to clean and normalize the review text. This function performs the following steps:
  1. **Remove Non-Alphabetic Characters**: Uses regular expressions to keep only alphabetic characters and spaces.
  2. **Lowercase Conversion**: Converts all text to lowercase.
  3. **Tokenization**: Breaks down the text into individual words (tokens) using nltk.word_tokenize.
  4. **Stop Word Removal**: Eliminates common English stop words (e.g., "the", "a", "is") that do not contribute much to sentiment.
  5. **Lemmatization**: Reduces words to their base or root form (e.g., "running" to "run", "better" to "good") using WordNetLemmatizer.
  6. **Join Tokens**: Joins the processed tokens back into a single string.

     ```
     def preprocess_text(text):
         text = re.sub(r'[^a-zA-Z\s]', '', text)
         text = text.lower()
         tokens = nltk.word_tokenize(text)
         stopwords_list = set(stopwords.words('english'))
         tokens = [token for token in tokens if token not in stopwords_list]
         lemmatizer = WordNetLemmatizer()
         tokens = [lemmatizer.lemmatize(token) for token in tokens]
         preprocessed_text = ' '.join(tokens)
         return preprocessed_text
     df['clean_text'] = df['Text'].apply(preprocess_text)
     ```

4. Data Splitting and Feature Extraction

The preprocessed text data is prepared for model training.

- **Train-Test Split**: The dataset is split into training and testing sets. clean_text is used as features (X) and Score as the target (y). A test size of 20% is used, with random_state=42 for reproducibility.

  ```
  X_train, X_test, y_train, y_test = train_test_split(df['clean_text'], df['Score'],
  test_size=0.2, random_state=42)
  ```

- **TF-IDF Vectorization**: TfidfVectorizer is used to convert the text data into numerical feature vectors. TF-IDF (Term Frequency-Inverse Document Frequency) assigns weights to words based on their importance in a document relative to the entire corpus. max_features is set to 5000 to consider the most frequent and important words.

  ```
  tfidf_vectorizer = TfidfVectorizer(max_features=5000)
  X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
  X_test_tfidf = tfidf_vectorizer.transform(X_test)
  ```

5. Model Training and Evaluation

A Linear Support Vector Classifier (SVC) is chosen for the sentiment classification task.

- **Model Initialization and Training**: A LinearSVC model is initialized and trained on the TF-IDF transformed training data.

  ```
  svm_classifier = LinearSVC()
  svm_classifier.fit(X_train_tfidf, y_train)
  ```

- **Prediction**: Predictions are made on the unseen test set.

  ```
  y_pred = svm_classifier.predict(X_test_tfidf)
  ```

- **Evaluation Metrics**: The model's performance is evaluated using a classification report and accuracy score.

```
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

The output shows the following performance:

```
          precision    recall  f1-score   support

       1       0.63      0.67      0.65     10515
       2       0.49      0.13      0.20      5937
       3       0.48      0.22      0.30      8460
       4       0.49      0.21      0.29     16026
       5       0.77      0.96      0.86     72743

accuracy                           0.73    113681
macro avg       0.57      0.44      0.46    113681
weighted avg    0.68      0.73      0.68    113681

Accuracy: 0.7298757048231455
```

The model achieves an overall accuracy of approximately **72.99%**. It performs particularly well for Score 5 (Positive reviews), with high precision, recall, and f1-score. Performance is lower for intermediate scores (2, 3, 4), indicating that distinguishing between these nuanced sentiments is more challenging.

6. Interactive Interface with Gradio

To make the model accessible, a web-based interface is created using Gradio.

- **Sentiment Prediction Function**: A predict_sentiment function is defined to take a raw review text, preprocess it, transform it using the trained TF-IDF vectorizer, and then use the svm_classifier to predict the sentiment score. The score is then mapped to "Positive 😊" (score >= 4), "Negative 😠" (score <= 2), or "Neutral 😐".

```
def predict_sentiment(review):
    processed_review = preprocess_text(review)
    review_tfidf = tfidf_vectorizer.transform([processed_review])
    prediction = svm_classifier.predict(review_tfidf)[0]
    sentiment = "Positive 😊" if prediction >= 4 else "Negative 😠" if prediction <= 2
else "Neutral 😐"
    return sentiment
```

- **Gradio Interface**: A gr.Interface is set up with:
  - fn: The predict_sentiment function.
  - inputs: A gr.Textbox for user input.
  - outputs: A gr.Textbox to display the sentiment prediction.
  - title: "Amazon Product Review Sentiment Analysis".
  - description: "Enter a review and get its sentiment: Positive, Neutral, or Negative."
  - share=True: This option makes the interface publicly accessible via a shareable link (temporary).

```
import gradio as gr

interface = gr.Interface(
    fn=predict_sentiment,
    inputs=gr.Textbox(label="Enter a Review"),
    outputs=gr.Textbox(label="Sentiment Prediction"),
    title="Amazon Product Review Sentiment Analysis",
    description="Enter a review and get its sentiment: Positive, Neutral, or
Negative.",
)

interface.launch(share=True)
```

This launches a web interface where users can type in a review and instantly see the predicted sentiment.

7. Conclusion and Future Enhancements

The notebook successfully demonstrates a complete pipeline for Amazon product review sentiment analysis, from data preprocessing and model training to interactive deployment. The LinearSVC model achieves a decent accuracy of approximately 73%, with strong performance on clearly positive reviews.

Possible future enhancements include:

- **Advanced Text Preprocessing**: Exploring more sophisticated techniques like handling negations, emojis, or slang.
- **Deep Learning Models**: Implementing recurrent neural networks (RNNs) or transformer-based models (e.g., BERT) for potentially higher accuracy, especially for capturing contextual nuances in text.
- **Hyperparameter Tuning**: Optimizing the TfidfVectorizer (e.g., ngram_range, min_df, max_df) and LinearSVC (e.g., C parameter) for better performance.
- **Handling Class Imbalance**: The dataset likely has an imbalance towards positive reviews (Score 5). Techniques like SMOTE or class weighting could be used to

improve performance on minority classes.

- **More Granular Sentiment**: Instead of just Positive/Neutral/Negative, the model could predict the exact score (1-5) or a more nuanced sentiment scale.
- **User Interface Improvements**: Enhancing the Gradio interface with features like example inputs, clear error messages, or visual indicators for sentiment strength.